

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания

по организации практической работы обучающихся
по дисциплине «Алгоритмы задач электроэнергетики»
для студентов направления подготовки /специальности
13.03.02 Электроэнергетика и электротехника
Направленность (профиль) Электроэнергетические системы и сети

Ставрополь 2026

СОДЕРЖАНИЕ

Практическая работа №1	3
Практическая работа №2	23
Практическая работа №3	44
Практическая работа №4	57
Практическая работа №5	66
Практическая работа №6	79
Практическая работа №7	93
Практическая работа №8	101
Практическая работа №9	116
Практическая работа №10	125
Практическая работа №11	128
Практическая работа №12	143
Список рекомендуемой литературы	150

Практическая работа №1

Тема: «Знакомство со средой программирования Python IDLE и облачными онлайн интерпретаторами языка Python».

Цель: Изучить графический интерфейс программной среды Python IDLE, научиться создавать простейшие программы на языке Python

Краткая история языка Python

Язык программирования Python является относительно «молодым» языком по сравнению со многими другими современными языками программирования. Python был задуман в еще в далеких 80-х программистом из Голландии Гвидо ван Россумом. Создавать его Гвидо начал в 1989 году как замену языку ABC, предназначенного для обучения студентов программированию. Он создавался на энтузиазме, без бюджета и поддержки. Работал Гвидо ван Россум по вечерам и в выходные дни. В результате за несколько недель был написан интерпретатор Python. Многие идеи были взяты из других языков программирования таких как C и ABC. В итоге из этого проекта вырос Python. Сегодня Python входит в тройку самых популярных языков мира. Гвидо ван Россум был поклонником комедийного телешоу «Летающий цирк Монти Пайтона» и в честь этого английского сериала, популярного в 1970 – х годах дал название своему языку программирования Python. Этот сериал принес еще одно очень популярное сегодня слово «спам». Оно обозначает навязчивую рекламу, с которой знаком каждый.

Публикация исходного текста Python произошла в 1991 году. Язык был относительно несложным. В нем было мало функций и интерпретатор «interpreter» тоже был небольшого размера. Всего было 3 версии питона:

Первая версия Python 1.0 увидела свет в январе 1994 года и развитие ее релизов продолжалось до 2000 года. Последним релизом стал Python 1.6. С

2000 по 2008 год развивалась его вторая версия. В это время проект был помещен на SourceForge. Это платформа, на которой расположены проекты с открытым кодом и был дан импульс к развитию языка сообществом программистов. В 2008 году вышла третья версия, которая развивается и в настоящее время. Поддержка второй версии постепенно прекращается и все большее число программистов пользуется третьей версией Python. На сегодняшний последний версией питона является Python 3.10. Третья версия языка осуществляет поддержку второй не полностью.

Общие сведения о языке

Для того чтобы понять, что такое Python нужно разобраться зачем вообще нужны языки программирования. Они нужны для взаимодействия человека и компьютера. Между собой люди общаются на естественном языке. Компьютеры общаются между собой на естественном для машин языке – машинном языке, языке нулей и единиц. Для того, чтобы компьютер и человек понимал друг друга нужен промежуточный язык, который, с одной стороны, был бы удобен для понимания человеку, а с другой стороны, мог бы быть переведен с помощью специальной «программы-переводчика» на язык, понятный компьютеру. Таких языков со времен изобретения компьютера было придумано огромное множество. Все они имеют что-то общее, и каждый из них обладает своими особенностями. Одну и ту же задачу можно решить на разных языках программирования. Точно так же, как один и тот же предмет или действие можно описать на разных человеческих языках. Но одна и та же задача на разных языках программирования будет занимать разное количество строк, и выполняться за разное время. Это не говорит о том, что какой-то из языков лучше, а какой-то хуже. Просто каждый язык изначально создается для решения конкретных задач и имеет в себе более широкий набор встроенных механизмов для их решения, по сравнению с другими языками, изначально не создаваемых для решения этих задач.

Python может быть полезен вам в вашей дальнейшей профессиональной деятельности для решения следующих задач:

- обработка статистических и экспериментальных данных;
- выполнение сложных математических операций;
- визуализация данных и результатов их обработки с помощью графиков, диаграмм, гистограмм и так далее;
- применение технологии искусственного интеллекта для решения поставленных профессиональных задач;
- работа с базами данных и технологией больших данных (BigData).

Все перечисленные задачи и многие другие можно решить с помощью встроенных и сторонних библиотек языка Python, поскольку Python является относительно простым высокоуровневым скриптовым языком, который служит для создания различных сценариев.

«Высокоуровневый» означает, что для программирования на Python, не требуется знания машинных кодов, команд для компьютера. Программирование при помощи машинных кодов увеличивает скорость работы программы, но очень трудоемко и требует хорошего знания архитектуры компьютера. Одним из таких низкоуровневых языков является ассемблер. Чаще всего низкоуровневое программирование используется в промышленности для специализированных контроллеров, где нет большого объема памяти. В отличие от ассемблера, Python не работает с машинными кодами. При этом программы выполняются более медленно, но они более понятны для человека. Учитывая быстродействие современных компьютеров, во многих случаях это преимущество простоты кода языка Python перевешивает его недостаток в скорости выполнения.

Под скриптовым языком программирования можно понимать то, что выполнение программы происходит сверху вниз построчно. Существует два принципиально разных типа «программ-переводчиков»: компиляторы и интерпретаторы.

Компиляторы переводят текст программы на язык, понятный компьютеру, сразу целиком. В итоге получается бинарный исполняемый файл, который можно скопировать и затем запустить на любом компьютере. Интерпретаторы переводят текст программы на язык, понятный компьютеру «на лету», построчно, и компьютер сразу же исполняет полученную команду. Поэтому, для того, чтобы программа, написанная на языке Python могла выполняться на компьютере, необходимо чтобы на нем был установлен интерпретатор языка Python.

Установка интерпретатора языка

Скачать самую последнюю версию интерпретатора языка Python можно совершенно бесплатно на официальном сайте проекта: <https://www.python.org>. При установке рекомендуется установить галочки «Install launcher for all users» и «Add Python to PATH».



Рисунок 1.1 – Установка интерпретатора языка Python

После установки, запустить интерпретатор языка Python можно через меню «Пуск→ Python→IDLE». Откроется интегрированная среда для разработки и обучения IDLE (Integrated Development and Learning Environment), изображенная на рисунке 1.2.

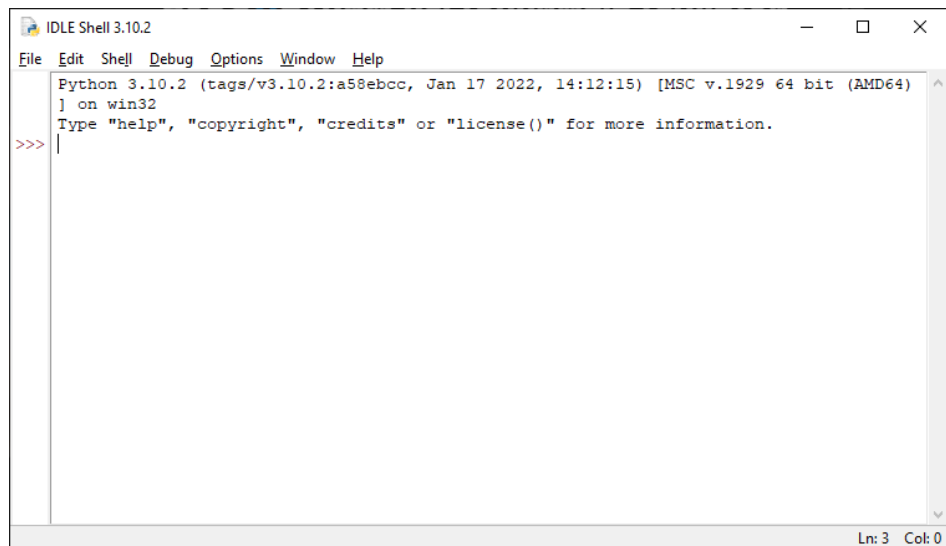


Рисунок 1.2 – Интегрированная среда разработки IDLE

Режимы работы Python

IDLE поддерживает 2 режима работы:

1. Интерактивный режим. В этом режиме команды выполняются сразу после их вызова. Результат выполнения команд сразу выводится на экран. Этот режим работы хорошо подходит для написания простых программ и тестирования работы фрагментов кода. Интерактивный режим имеет следующие взаимосвязанные недостатки:

- программный код, который вводится, нигде не сохраняется. Чтобы повторно запустить один и тот же программный код, его нужно снова ввести;
- программы, которые введены в интерактивном режиме, после выполнения интерпретатором Python исчезают. Чтобы выполнить их повторно, нужно их снова набрать или использовать операции копирования, что неэффективно.

2. Программный режим. В этом режиме сначала записывается вся программа, а потом эта программа выполняется полностью. Предварительно текст программы нужно сохранить. Программа на языке Python носит имя «сценарий» или «скрипт». Если система находится в интерактивном режиме, то программный режим можно вызвать с помощью команды «File→New File». Откроется текстовый редактор, изображенный на рисунке 1.3.

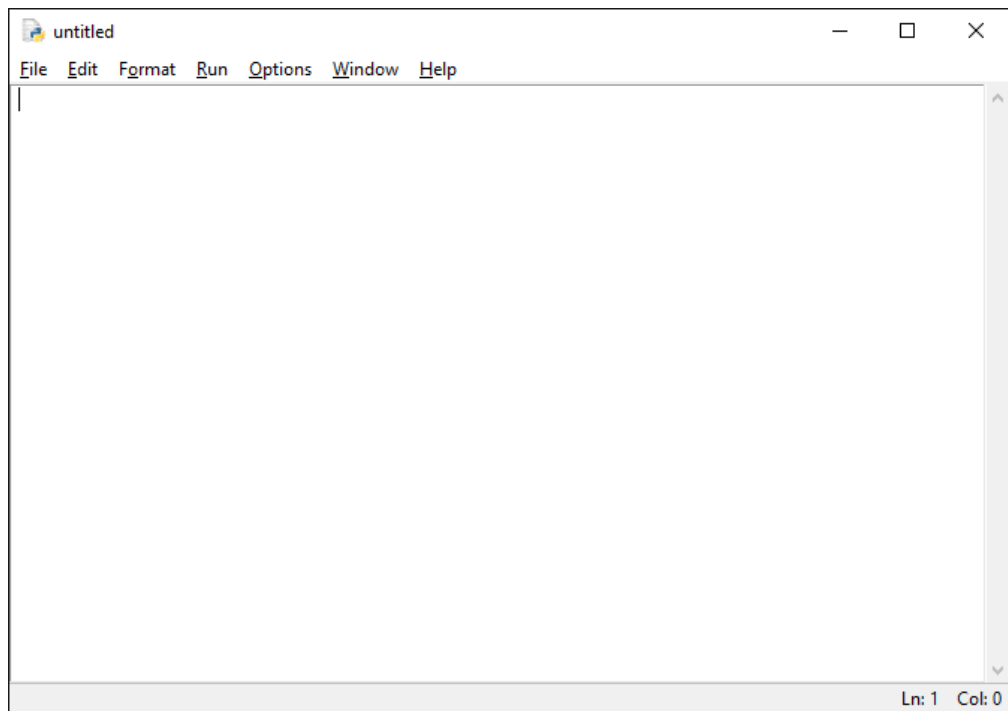


Рисунок 1.3 – Редактор исходного кода программы IDLE

После того, как текст программы будет введен, необходимо выполнить команду меню «Run→Run Module». Программа попросит выбрать путь для сохранения текста скрипта, после чего результаты выполнения программы будут отображены в окне среды IDLE.

Облачные интерпретаторы языка Python

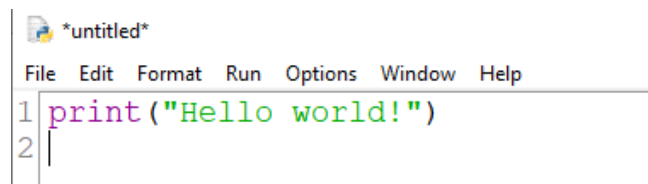
В сети интернет есть множество ресурсов, позволяющих выполнять разработку программ на языке Python без установки на своем компьютере среды разработки, например:

- <https://www.online-python.com/>
- https://www.onlinegdb.com/online_python_compiler
- <https://replit.com/languages/python3>
- <https://onlinepythoncompiler.com/>
- <https://www.codabrainy.com/python-compiler/>

Перечисленные ресурсы отличаются удобством интерфейса, скоростью работы, набором дополнительных функций и установленных компонентов языка Python.

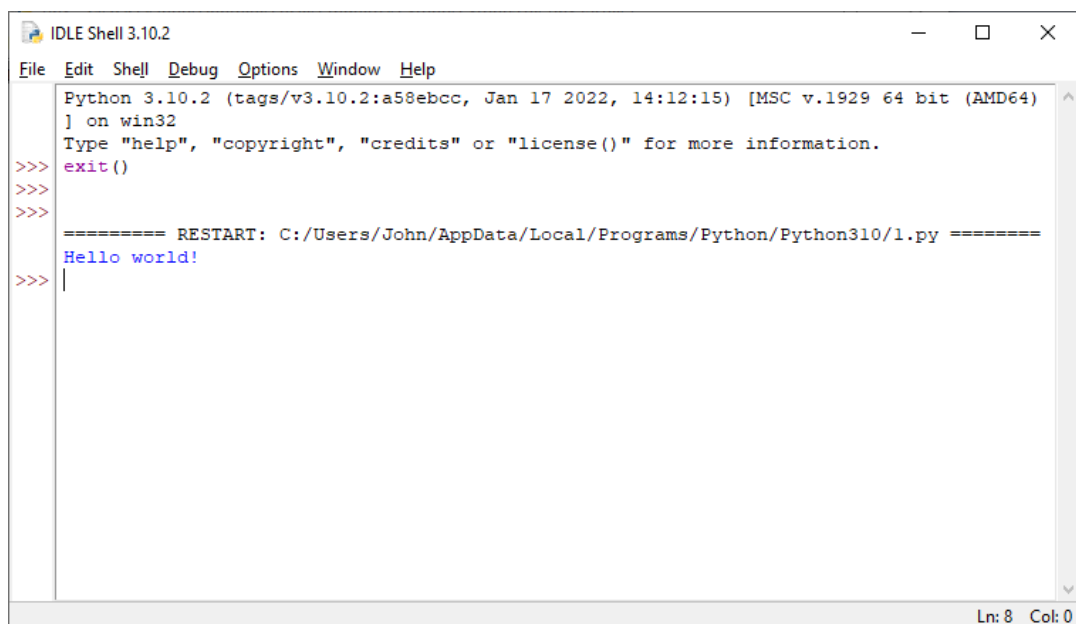
Первая программа на языке Python

Исторически так сложилось, что самая первая программа на любом языке программирования называется «Hello world!». Суть этой программы заключается в том, чтобы вывести на экран какую-нибудь простейшую фразу. Давайте напишем свою первую программу на языке Python. Для этого в среде IDLE (неважно в каком режиме, интерактивном или программном) необходимо написать следующую команду:



```
*untitled*
File Edit Format Run Options Window Help
1 print("Hello world!")
2 |
```

И запустить ее на исполнение (в интерактивном режиме просто нажать Enter, в программном режиме выполнить команду меню «Run→Run Module»). На экране появится соответствующая фраза (рисунок 1.4).



```
IDLE Shell 3.10.2
File Edit Shell Debug Options Window Help
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> exit()
>>>
>>>
===== RESTART: C:/Users/John/AppData/Local/Programs/Python/Python310/1.py =====
Hello world!
>>> |
```

Рисунок 1.4 – Первая программа на языке Python

Давайте разберемся, что означает эта команда. Ключевое слово «print» является названием функции. Функция – это заверченный кусочек программы, который выполняет определенные действия и, что самое важное, может использоваться многократно. В данном случае функция «print»

выполняет вывод информации на экран. Вся программа строится на использовании уже готовых функций, входящих в состав Python, функций, написанных другими программистами и подключенных к вашей программе в виде модулей, либо же пользовательских функций, которые вы пишете сами. Функция может не иметь аргументов, или иметь один и более аргумент. Аргумент – это как бы начальные условия, параметры, описывающие особенности выполнения функции. Аргументы функции заключаются в круглые скобки после названия самой функции и перечисляются через запятую, если их несколько. Если функция не имеет аргументов, в круглых скобках ничего не пишется. В данном случае аргументом функции является фраза «Hello world!». Она заключается в кавычки (двойные или одинарные), так как является строкой.

Ценность языков программирования заключается не в том, что они позволяют вывести на экран заранее определенные фразы, а в том, что позволяют производить вычисления и расчеты и на их основе уже принимать решение, что выводить на экран.

Для этого в любом языке программирования существуют переменные. Переменные – это своего рода ячейки памяти, в которых хранятся определенные значения и над которыми можно производить вычисления (сложение, умножение и так далее). Изначально значения в эти ячейки попадают различными способами:

- вводятся с клавиатуры;
- считываются из файла;
- считываются из баз данных;
- считываются из интернета;
- поступают от измерительных устройств;
- заранее однозначно задаются программистом.

Переменная характеризуется именем, типом данных и значением.

Имя – это уникальный идентификатор, по которому однозначно можно обратиться к ячейке памяти, не перепутав ее с какой-либо другой ячейкой.

Тип указывает, что же, собственно говоря, хранится в ячейке памяти. Типы данных можно условно разделить на основные и составные. Основные, в свою очередь, делятся на простейшие и расширенные. К простейшим типам данных языка Python относятся:

- логические (bool, могут принимать значение True либо False);
- целочисленные (int, для хранения целых чисел);
- вещественные (float, для хранения дробных чисел);
- комплексные (complex, для хранения комплексных чисел, состоящих из вещественной и мнимой части);
- строковые (str, для хранения строковых данных).

На основе этих простейших типов строятся более сложные, расширенные типы данных (списки, массивы, множества, словари), а также любые другие, составные типы данных.

Особенностью языка Python является то, что в любой переменной в любой момент времени может храниться любой тип данных. Например:

```
a=True  
a=1  
a=0.1  
a='Hello'
```

В данном случае, а – имя переменной. Имя переменной должно начинаться с буквы или символа подчеркивания. Имя переменной не может начинаться с цифры. Имя переменной может содержать только буквенно-цифровые символы и символы подчеркивания (A – z, 0–9 и _). Имена переменных чувствительны к регистру (age, Age и AGE – это три разные переменные). После знака равенства указывается значение, которые мы заносим в переменную. В первой строке мы заносим значение типа «bool», затем целочисленное, затем вещественное и в конце – строковое. Узнать, какого типа данные сейчас хранятся в переменной можно с помощью функции type(). Например:

```

File Edit Format Run Options Window Help
1 print("Hello world!")
2 a=True
3 print(type(a))
4 a=1
5 print(type(a))
6 a=0.2
7 print(type(a))
8 a='Hello'
9 print(type(a))

```

В результате получим:

```

===== RESTART: C
Hello world!
<class 'bool'>
<class 'int'>
<class 'float'>
<class 'str'>

```

Над переменными можно выполнять различные операции. Операции над переменным осуществляются с помощью операторов.

Оператор в Python – это символ, который выполняет операцию над одним или несколькими операндами. Операндом выступает переменная или значение, над которыми проводится операция.

Операторы Python бывают 7 типов:

- арифметические операторы;
- операторы сравнения;
- операторы присваивания;
- логические операторы;
- операторы принадлежности;
- операторы тождественности;
- битовые операторы.

Познакомимся с некоторыми из них:

Операторы языка Python

Арифметические операторы Python

Сложение (+): складывает значение по обе стороны оператора.

```
>>> 3+4
7
```

Вычитание (-): вычитает значение правой стороны из значения в левой.

```
>>> 3-4
-1
```

Умножение (*): перемножает значения с обеих сторон оператора.

```
>>> 3*4
12
```

Деление (/): делит значение левой стороны на значение правой. Тип данных результата деления — число с плавающей точкой.

```
>>> 3/4
0.75
```

Возведение в степень ():** возводит первое число в степень второго.

```
>>> 3**4
81
```

Деление без остатка (//): выполняет деление и возвращает целочисленное значение частного, убирая цифры после десятичной точки.

```
>>> 4//3
1
>>> 10//3
3
```

Деление по модулю (остаток от деления) (%): выполняет деление и возвращает значение остатка.

```
>>> 3%4
3
>>> 4%3
1
>>> 10%3
1
>>> 10.5%3
1.5
```

Операторы сравнения

Операторы сравнения в Python проводят сравнение операндов. Они сообщают, является ли один из них больше второго, меньше, равным или и то и то.

Меньше (<): этот оператор проверяет, является ли значение слева меньше, чем правое.

```
>>> 4<3
False
```

Больше (>): проверяет, является ли значение слева больше правого.

```
>>> 4>3
True
```

Меньше или равно (<=): проверяет, является ли левая часть меньше или равной правой.

```
>>> 7<=7
True
```

Больше или равно (>=): проверяет, является ли левая часть больше или равной правой.

```
>>> 0>=0
True
```

Равно (==): этот оператор проверяет, равно ли значение слева правому.

```
>>> 3==3.0
True
>>> 1==True
True
>>> 7==True
False
>>> 0==False
True
>>> 0.5==True
False
```

Не равно (!=)

Проверяет, не равно ли значение слева правому.

Когда условие выполнено, возвращается **True**. В противном случае – **False**. Это возвращаемое значение можно использовать в последующих инструкциях и выражениях.

Пример:

```
>>> 1!=1.0
False
```

Операторы присваивания

Оператор присваивания присваивает значение переменной. Он может манипулировать значением до присваивания. Есть 8 операторов присваивания: 1 простой и 7 с использованием арифметических операторов.

Присваивание (=): присваивает значение справа левой части. Стоит обратить внимание, что `==` используется для сравнения, а `=` — для присваивания.

```
>>> a = 7
>>> print(a)
7
```

Сложение и присваивание (+=): суммирует значение обеих сторон и присваивает его выражению слева. `a += 10` — это то же самое, что и `a = a + 10`. То же касается и все остальных операторов присваивания.

```
>>> a += 2
>>> print(a)
9
```

Вычитание и присваивание (-=): вычитает значение справа из левого и присваивает его выражению слева.

```
>>> a -= 2
>>> print(a)
7
```

Деление и присваивание (/=): делит значение слева на правое. Затем присваивает его выражению слева.

```
>>> a /= 7
>>> print(a)
1.0
```

Умножение и присваивание (*=): перемножает значения обеих сторон. Затем присваивает правое левому.

```
>>> a *= 8
>>> print(a)
8.0
```

Деление по модулю и присваивание (%=): выполняет деление по модулю для обеих частей. Результат присваивает левой части.

```
>>> a %= 3
>>> print(a)
2.0
```

Возведение в степень и присваивание (=):** выполняет возведение левой части в степень значения правой части. Затем присваивает значение левой части.

```
>>> a **= 5
>>> print(a)
32.0
```

Деление с остатком и присваивание (/=): выполняет деление с остатком и присваивает результат левой части.

```
>>> a /= 3
>>> print(a)
10.0
```

Логические операторы Python

Это союзы, которые позволяют объединять по несколько условий. В Python есть всего три оператора: **and** (и), **or** (или) и **not** (не).

И (and): если условия с двух сторон оператора **and** истинны, тогда все выражение целиком считается истинным.

```
>>> a = 7 > 7 and 2 > -1
>>> print(a)
False
```

Или (or): выражение ложно, если оба операнда с двух сторон ложные. Если хотя бы одно из них истинное, то и все выражение истинно.

```
>>> a = 7 > 7 or 2 > -1
>>> print(a)
True
```

Не (not)

Этот оператор инвертирует булевы значения выражения. **True** превращается в **False** и наоборот. В примере внизу булево значение **0** – **False**. Поэтому оно превращается в **True**.

```
>>> a = not(0)
```

```
>>> print(a)
True
```

Оператор ветвления

Еще одним важным оператором является оператор ветвления. Оператор ветвления(условия) – оператор управления, осуществляющий выполнение одного и только одного составного оператора из двух возможных в зависимости от условия. Предположим, нам нужно сравнить два числа А и В и вывести на экран наибольшее из них. В этом случае нам поможет оператор ветвления, который выглядит следующим образом:

```
if выражение:
    инструкция_1
    инструкция_2
    ...
    инструкция_n
```

После оператора *if* записывается выражение. Если это выражение истинно, то выполняются инструкции, определяемые данным оператором. Выражение является истинным, если его результатом является число не равное нулю, непустой объект, либо логическое *True*. После выражения нужно поставить двоеточие “:”.

ВАЖНО: блок кода, который необходимо выполнить, в случае истинности выражения, должен быть смещен относительно оператора *if* на один или несколько пробелов. Количество пробелов, на которое смещены команды, не имеет значения: 1,2,4,10,20 и так далее. Важно чтобы все команды, которые должны выполняться, были смещены на одинаковое количество пробелов, то есть располагались как бы в столбик:

```
if a>5:
    print("hello")
    print("hello")
    print("hello")
    print("hello")
```

Для удобства восприятия, принято смещать блок команд на 4 пробела. Именно столько пробелов интегрированная среда разработки добавит сама, если нажать на клавишу «Tab».

Примеры:

```
if 1:  
    print("hello 1")
```

Напечатает: *hello 1*

```
a = 3  
if a == 3:  
    print("hello 2")
```

Напечатает: *hello 2*

```
a = 3  
if a > 1:  
    print("hello 3")
```

Напечатает: *hello 3*

Бывают случаи, когда необходимо предусмотреть альтернативный вариант выполнения программы. Т.е. при истинном условии нужно выполнить один набор инструкций, при ложном – другой. Для этого используется конструкция *if – else*.

```
if выражение:  
    инструкция_1  
    инструкция_2  
    ...  
    инструкция_n  
else:  
    инструкция_a  
    инструкция_b  
    ...  
    инструкция_x
```

Примеры.

```
a = 3  
if a > 2:  
    print("H")  
else:  
    print("L")
```

Напечатает: *H*

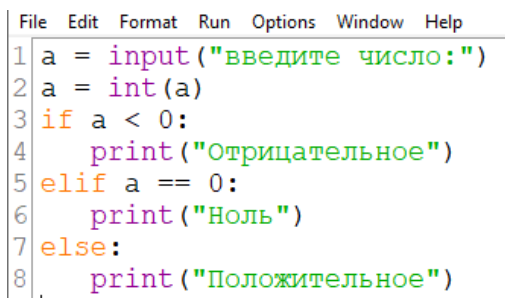
```
a = 1  
if a > 2:  
    print("H")  
else:  
    print("L")
```

Напечатает: *L*

Для реализации выбора из нескольких альтернатив можно использовать конструкцию *if–elif–else*.

```
if выражение_1:
    инструкции_(блок_1)
elif выражение_2:
    инструкции_(блок_2)
elif выражение_3:
    инструкции_(блок_3)
else:
    инструкции_(блок_4)
```

Пример.



```
File Edit Format Run Options Window Help
1 a = input("введите число:")
2 a = int(a)
3 if a < 0:
4     print("Отрицательное")
5 elif a == 0:
6     print("Ноль")
7 else:
8     print("Положительное")
```

В данном примере мы встречаем две новые функции: `input` и `int`. Функция `input` предназначена для ввода данных с клавиатуры. У нее есть один аргумент – подсказка пользователю, которая будет выведена на экран. Результатом выполнения функции является строка, которую пользователь введет с клавиатуры. Чтобы понять, для чего нужна функция `int` давайте вернемся типам данных языка Python. В языке Python существуют следующие базовые типы данных: логический, целочисленный, вещественный, комплексный и строковый. Важно то, что в зависимости от того, какой тип данных хранится в переменной будет зависеть какие операции можно будет выполнить с этой переменной. Например, над числами можно выполнять арифметические операции: складывать, вычитать, возводить в квадрат. Сложно себе представить, как можно возвести в квадрат строку, например. В то же самое время, например, сложение строк является довольно очевидной операцией: мы просто объединяем эти две строки. Таким образом, в зависимости от того, какого типа данные хранятся в

переменной будет зависеть результат операций над этими переменными. Например:

```
File Edit Format Run Options Window Help
1 a=5
2 b=6
3 c="5"
4 d="6"
5 print(a+b)
6 print(c+d)
```

На экран будет выведено две строки: 11 и 56. 11 потому что *a* и *b* являются числами и при сложении будут использованы правила сложения чисел, а 56, потому что переменные *c* и *d* являются строками и при сложении будут использоваться правила сложения строк, то есть к концу первой строки будет добавлена вторая строка. Результатом выполнения функции `input` всегда является строка. Для того, чтобы сказать программе, что нам нужна не строка, а число, мы вызываем функцию `int()`. Если нам нужно будет вещественное число, мы используем функцию `float()`, если комплексное, то функцию `complex()` и так далее. Аналогичным образом, если нам нужно преобразовать число в строку, мы используем функцию `str()`.

Еще один интересный момент: чтобы сократить количество строк в программе, мы можем использовать в качестве аргумента функции другую функцию. Например, вместо

```
a = input("введите число:")
a = int(a)
```

можно сразу написать:

```
a = int(input("введите число:"))
```

Результат будет один и тот же

Задание

1. Ознакомьте с теоретическим материалом.
2. Ответьте на контрольные вопросы, приведенные в конце работы.
3. Выполните индивидуальное задание, согласно вашему номеру варианта (Таблица 1.1).

Таблица 1.1 – Задания для самостоятельного выполнения:

№ варианта	Задачи для выполнения	№ варианта	Задачи для выполнения
1	9,6,10,11,12	9	8,9,2,11,12
2	10,7,4,11,12	10	5,6,7,11,12
3	7,5,2,11,12	11	9,5,6,11,12
4	6,8,9,11,12	12	5,7,2,11,12
5	6,7,2,11,12	13	7,5,1,11,12
6	7,10,1,11,12	14	3,9,4,11,12
7	9,10,1,11,12	15	6,7,8,11,12
8	10,8,2,11,12	16	1,2,3,11,12

Задача №1. Создайте программу решения квадратного уравнения. Коэффициенты А,В,С квадратного уравнения пользователь должен вводить с клавиатуры, корни уравнения необходимо вывести на экран.

Задача №2. Напишите следующую программу: пользователь вводит с клавиатуры 5 чисел. Необходимо найти их сумму и вывести на экран.

Задача №3. Напишите следующую программу: пользователь вводит с клавиатуры 5 чисел. Необходимо найти их произведений и вывести на экран.

Задача №4. Напишите следующую программу: пользователь вводит с клавиатуры 5 чисел. Необходимо посчитать, сколько чисел оказалось больше 10 и вывести на экран это количество.

Задача №5. Напишите следующую программу: пользователь вводит с клавиатуры 5 чисел. Необходимо посчитать количество четных и нечетных чисел и вывести на экран это количество.

Задача №6. Напишите следующую программу: пользователь вводит с клавиатуры 5 чисел. Необходимо вывести на экран сообщение, показывающее, каких чисел оказалось больше: четных или нечетных.

Задача №7. Напишите следующую программу: пользователь вводит с клавиатуры 5 чисел. Необходимо посчитать произведение четных и сумму нечетных чисел и вывести на экран полученные значения.

Задача №8. Напишите следующую программу: пользователь вводит с клавиатуры 5 чисел. Необходимо найти сумму квадратов этих чисел и вывести на экран.

Задача №9. Напишите следующую программу: пользователь вводит с клавиатуры длину и ширину прямоугольника. Необходимо посчитать и вывести на экран: периметр прямоугольника, его площадь и длину диагонали.

Задача №10. Напишите следующую программу: пользователь вводит с клавиатуры 5 чисел. Необходимо четные числа сложить по правилам сложения чисел, а нечетные числа сложить по правилам сложения строк.

Задача №11. Создайте программу вычисления мощности и напряжения в конце линии с параметрами R, X по известным данным мощности и напряжении в начале линии.

Задача №12. Создайте программу вычисления параметров схемы замещения трансформатора (R,X,B,G,KTR для каждого положения отпайки) по его паспортным данным.

Контрольные вопросы:

1. Почему язык Python получил такое название?
2. Назовите основные особенности языка Python.
3. Для каких задач может быть полезен язык Python?
4. Назовите режимы работы среда разработки IDLE. В чем их особенности?
5. Что такое функция в языке? Что такое аргумент функции?
6. Что такое переменная? Чем характеризуется переменная?
7. Что такое оператор? Какие операторы вы знаете?
8. Как выводить на экран значения переменных?
9. Как ввести с клавиатуры значения переменных?
10. Опишите синтаксис и назначение оператора ветвления. Приведите пример его использования.
11. Для чего нужны функции int(),float(),complex(),str()?

Практическая работа №2

Тема: «Работа со списками»

Цель: Научиться объявлять, инициализировать, вводить и обрабатывать списки на языке Python. Научиться использовать циклы.

На прошлом занятии были рассмотрены основные типы данных языка Python: целочисленные, вещественные, строковые и логические. Однако, помимо перечисленных, в языке Python есть поддержка более сложных типов данных. Если у вас есть список предметов (например, список марок авто), то хранение автомобилей в отдельных переменных может выглядеть так:

```
car1 = "Ford"  
car2 = "Volvo"  
car3 = "BMW"
```

Для каждой новой марки нужно выделять переменную с новым именем. Это очень неудобно, ведь если их будет не 3, а, например, 3000, нужно будет создавать 3000 новых переменных. Кроме того, работать с такими переменными тоже будет неудобно. Например, если нужно будет проверить, есть ли среди перечисленных марок какая-то определенная, нужно будет написать 3000 команд сравнения. К счастью, существуют более простые подходы к написанию программы. Для таких случаев существует специальный тип переменных, который называется «список». Список – это специальная переменная, которая может содержать более чем одно значение. Для создания списка необходимо использовать квадратные скобки. Например:

```
a= [ ]
```

Эта команда создаст пустой список. При создании списка можно сразу же поместить в него значения, через запятую. Например:

```
a=[1, 2, 3, 10]
```

Вывести на экран все элементы списка можно с помощью уже известной нам команды print:

```
a=[1,2,3,10]
print(a)
```

Обращение к элементам списка

Каждый элемент в списке имеет свой индекс. Индекс – это порядковый номер элемента в списке. Индекс начинается с нуля.



Рисунок 2.1 – Индексация элементов в списке

Для доступа к элементу списка по индексу используются квадратные скобки:

```
a=[1,2,3,10]
print(a[2])
```

выведет на экран число 3, так по индексу 2 располагается именно это число.

Существует также отрицательный индекс. Отрицательными индексами называют расположение элементов в списке справа налево. При этом нумерация начинается с -1.



Рисунок 2.2 – Индексация элементов в списке с конца

Отрицательным индексом удобно пользоваться, когда необходимо обратиться к последнему в списке элементу, не высчитывая его номер. Любой конечный элемент будет с индексом, равным -1.

Если же нужно получить из списка не один, а сразу несколько элементов, используются так называемые срезы. Срезы (slices) — это подмножества элементов списка. У них есть свой собственный синтаксис.

Записывается срез так же, как обращение к элементу, используя индекс.

Пример:

```
elements[START:STOP:STEP]
```

В этом случае берётся срез от номера start (включительно) до stop (не включая его), а step — это шаг. По умолчанию start и stop равны 0, step равен 1.

```
elements = [0.1, 0.2, 1, 2, 3, 4, 0.3, 0.4]
int_elements=elements[2:6]#выборка со 2 по 6-й элементы
print(int_elements)
```

На экран будут выведены элементы с индексами 2,3,4 и 5, то есть, в данном случае, только целые числа.

Комментарии в программе

В последнем примере встречается новый символ – решетка #. Этот символ используется для написания комментариев к программе. Все, что идет после этого символа и до конца строки программа не воспринимает. Это всего лишь подсказка, которую программист пишет сам себе или тем, кто будет читать эту программу в дальнейшем. Символ # удобно использовать, если нужно временно отключить выполнение какой либо команды. В этом случае можно полностью удалить эту команду. Но тогда нужно будет ее снова писать, когда она нам понадобится. Другой выход – написать символ # в начале строки с этой командой и она не будет выполняться. Например:

```
car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"
print(car1)
#print(car2)
#print(car3)
```

Выведет на экран только слово «Ford», так как вывод на экран двух остальных переменных «закомментирован».

Символ # называется однострочным комментарием, поскольку действует только до конца строки. Существует также и многострочный комментарий. Она начинается с трех кавычек (одинарных, или двойных, не имеет значение) и заканчивается тремя такими же кавычками.

Многострочные комментарии используются в случаях, когда нужно написать много комментариев и не хочется в начале каждой строки писать символ #, либо если нужно временно отключить сразу несколько строк программы.

```
car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"
'''print(car1)
print(car2)
print(car3)'''
```

Не выведет на экран ничего, поскольку команды вывода на экран закомментированы тройными кавычками.

Добавление элементов в список

В списках доступно добавление, изменение, удаление элементов. Рассмотрим каждый способ изменения элементов на примерах.

Для того чтобы добавить новый элемент в список, используется `list.append(x)`, где `list` — список, `x` — нужное значение.

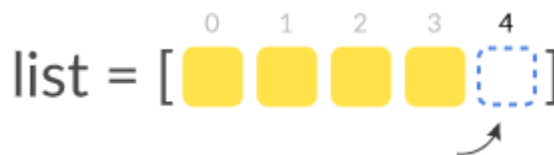


Рисунок 2.3 – Добавление элемента в конец списка

```
>>> elements = [1, 2, 3, 'word']
>>> elements.append('meow')
>>> print(elements)
[1, 2, 3, 'word', 'meow']
```

Для простого примера, рассмотрим создание списка с нуля с помощью метода `append()`:

```
>>> elements = []
>>> elements.append(1)
>>> elements.append('word')
>>> elements.append('meow')
>>> print(elements)
[1, 'word', 'meow']
```

Функция `append()` добавляет элемент в конец списка. Но если нужно добавить элемент в середину списка, необходимо использовать метод `insert`. Метод `insert` имеет следующий синтаксис:

`list.insert(i, x)`, где `list` – список, `i` – позиция, `x` – нужное значение.

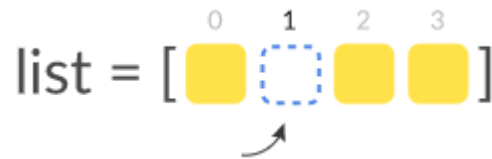


Рисунок 2.4 – Добавление элемента в середину списка

Пример:

```
>>> elements = [1, 2, 4]
>>> print(elements)
[1, 2, 4]

>>> elements.insert(2, 3)
>>> print(elements)
[1, 2, 3, 4]
```

Изменение элементов списка происходит следующим образом: нужно выбрать элемент по индексу (порядковому номеру элемента) и присвоить новое значение.



Рисунок 2.5 – Замена элемента в списке

```
>>> elements = [2, 4, 6]
>>> elements[2] = 8

>>> print(elements)
[2, 4, 8]
```

В примере выше мы заменили 6 на 8. Не забывайте, что счёт начинается с нуля, и в данном списке цифра 6 это 2-й элемент.

Для удаление из списка используют функция `list.pop(i)`, где `list` – список, `i` – индекс (позиция) элемента в списке. Если не указывать индекс, то будет удаляться последний элемент из списка.

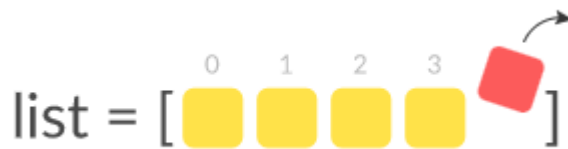


Рисунок 2.6 – Удаление последнего элемента списка

Объединение списков

Списки в Python можно объединять с помощью оператора `+` или метода `extend`.



Рисунок 2.7 – Объединение двух списков

Выглядит это так:

```
>>> a = [1, 3, 5]
>>> b = [1, 2, 4, 6]
>>> print(a + b)
[1, 3, 5, 1, 2, 4, 6]

>>> hello = ["h", "e", "l", "l", "o"]
>>> world = ["w", "o", "r", "l", "d"]
>>> hello.extend(world)    # extends не возвращает новый
список, а дополняет текущий
>>> print(hello)
['h', 'e', 'l', 'l', 'o', 'w', 'o', 'r', 'l', 'd']
```

У списков есть еще несколько полезных функций:

`list.clear()` – удаляет все элементы списка, не имеет аргументов;

`list.count(x)` – посчитает количество элементов `x` в списке;

`len(list)` – посчитает количество всех элементов в списке;

`list.remove(x)` – удалить из элемент `x` (удаляет только один элемент, который встретится первый раз);

`list.index(x)` – вернет позицию первого найденного элемента `x` в списке;

`list.copy()` – служит для копирования списков;

`list.reverse()` – меняет порядок элементов в списке на противоположный;

`list.sort()` – сортирует список.

Полный список функций можно посмотреть с помощью команды `help`:

```
>>> help(list)
```

Копирование списков

Попробуем выполнить следующую программу:

```
a=1
b=a
b=0
print(a)
print(b)
```

На экране будет выведено два числа: 1 и 0. Теперь выполним следующую программу:

```
a=[1, 2, 3]
b=a
b[0]=0
print(a)
print(b)
```

В результате на экране появится две строчки: `[0,2,3]` и `[0,2,3]`. Такой результат возникает из-за особенностей языка Python. Оператор присвоения «`=`» для простейших и расширенных типов данных ведет себя по-разному: для простейших типов он перезаписывает само значение переменной, а для расширенных типов он создает как-бы «псевдоним» присваиваемого объекта. Для того, чтобы именно перезаписать переменную новыми значениями, необходимо использовать функцию `copy()`.

```
a=[1, 2, 3]
b=a.copy()
b[0]=0
print(a)
print(b)
```

В результате на экране будет выведено `[1,2,3]` `[0,2,3]`.

Вложенные списки (матрицы)

В расчетных задачах электроэнергетики часто требуются вычисления с матрицами. В Python нет встроенных типов данных для хранения и обработки матриц. Для этого используют вложенные списки. Любую

матрицу можно рассматривать как «список списков». Рассмотрим пример матрицы размера 4 x 3:

```
matrix = [  
  [-1, 0, 1],  
  [-1, 0, 1],  
  [0, 1, -1],  
  [1, 1, -1]  
]
```

Либо в одну строку:

```
matrix = [[-1, 0, 1], [-1, 0, 1], [0, 1, -1], [1, 1, -1]]
```

Для вывода на экран можно воспользоваться уже известной функцией `print`.

Циклы

Выполнение программ, написанных на любом языке программирования, по умолчанию является последовательным. Иногда нам может понадобиться изменить выполнение программы. Выполнение определенного кода может потребоваться повторить несколько раз.

Для этого в языках программирования предусмотрены различные типы циклов, которые способны повторять определенный код несколько раз. Чтобы понять принцип работы оператора цикла, рассмотрим следующую схему.

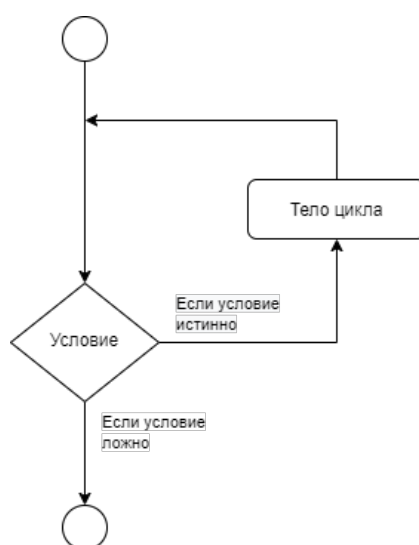


Рисунок 2.8 – Циклы с постусловием

Циклы упрощают сложные задачи до простых. Он позволяет нам изменить поток программы таким образом, что вместо того, чтобы писать один и тот же код снова и снова, мы можем повторять его конечное число раз. Например, если нам нужно вывести первые 10 натуральных чисел, то вместо того, чтобы использовать оператор print 10 раз, мы можем вывести их внутри цикла, который выполняется до 10 итераций.

В Python преимущества циклов, как и в других языках программирования, заключаются в следующем:

- Это обеспечивает возможность повторного использования кода.
- Используя циклы, нам не нужно писать один и тот же код снова и снова.
- С помощью циклов мы можем перебирать элементы сложных структур данных (списков и др.).

В Python существуют следующие два типа циклических выражений:

- Цикл while;
- Цикл for.

Циклы while

Цикл while позволяет выполнять часть кода до тех пор, пока заданное условие не станет ложным. Он также известен как цикл с предварительной проверкой условия.

Его можно рассматривать как повторяющийся оператор if. Когда мы не знаем количество итераций, цикл while является наиболее эффективным.

Синтаксис приведен ниже.

```
while expression:  
    statements
```

Здесь утверждения могут быть одним утверждением или группой утверждений. Выражение должно быть любым допустимым выражением

Python, приводящим к true или false. Истиной является любое ненулевое значение, а ложью - 0.

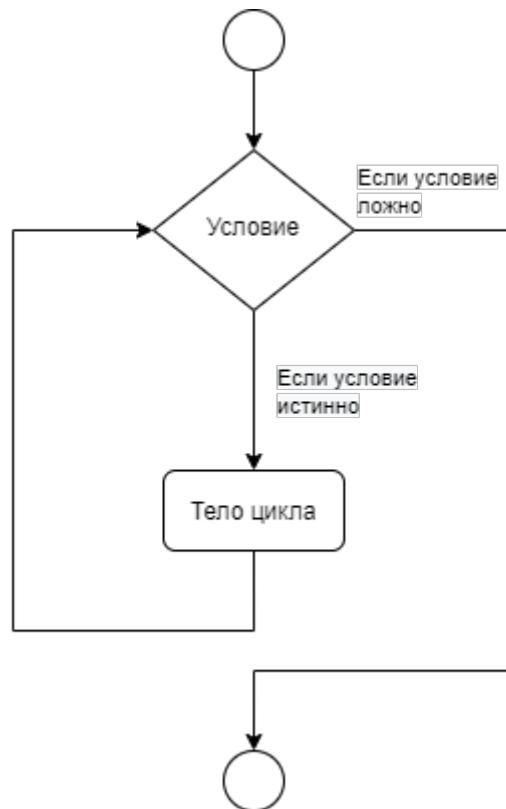


Рисунок 2.9 – Циклы с предусловием

Рассмотрим следующий пример цикла while в Python:

```
money = 10
while money > 0:
    msg="У нас есть "+str(money)+" рублей. Можно отдохнуть"
    print (msg)
    money -= 1
print ("Денег больше нет. Пора работать")
```

Запустив код, вы увидите следующий результат:

```
--
У нас есть 10 рублей. Можно отдохнуть
У нас есть 9 рублей. Можно отдохнуть
У нас есть 8 рублей. Можно отдохнуть
У нас есть 7 рублей. Можно отдохнуть
У нас есть 6 рублей. Можно отдохнуть
У нас есть 5 рублей. Можно отдохнуть
У нас есть 4 рублей. Можно отдохнуть
У нас есть 3 рублей. Можно отдохнуть
У нас есть 2 рублей. Можно отдохнуть
У нас есть 1 рублей. Можно отдохнуть
Денег больше нет. Пора работать
```

Как вы видите, цикл выполнился 10 раз. На каждой итерации цикла мы формировали сообщение, отображая текущее значения переменной money, а затем уменьшали его на 1. Как только переменная money стала равна нулю, условие продолжения цикла перестало выполняться и он завершился.

Цикл for в Python

Цикл **for** в Python используется для многократного повторения операторов или части программы. Он часто используется для обхода сложных структур данных, например, списков. Синтаксис цикла for в python приведен ниже.

```
for [элемент] in [последовательность]:  
    [список операций]
```

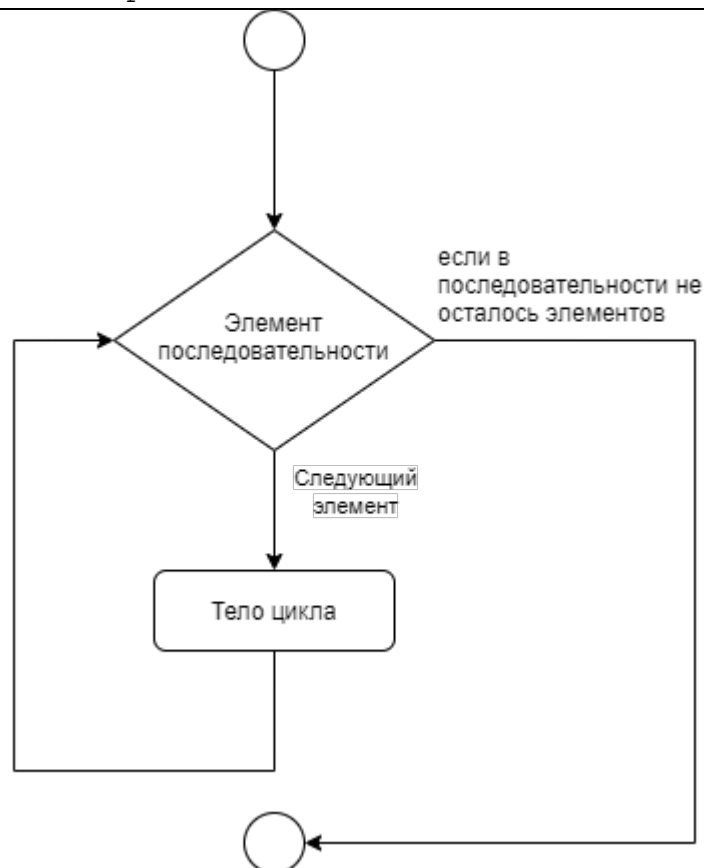


Рисунок 2.10 – Циклы с повторением для элементов последовательности

Цикл **for** в Python используется для работы с последовательностями или если число итераций известно заранее. Синтаксис выглядит следующим образом:

```
for [элемент] in [последовательность]:  
    [список операций]
```

Пример: Итерация строки с помощью цикла for

```
str = "Python"  
for i in str:  
    print(i)
```

Вывод:

```
P  
y  
t  
h  
o  
n
```

Пример: вывод на экран элементов списка

```
list = [1,2,3,4,5]  
for i in list:  
    print(i)
```

Вывод:

```
1  
2  
3  
4  
5
```

Пример: Программа для печати таблицы умножения заданного числа.

```
list = [1,2,3,4,5,6,7,8,9,10]  
n = 5  
for i in list:  
    c = n*i  
    print(c)
```

Вывод:

```
5  
10  
15  
20  
25  
30  
35  
40  
45  
50
```

Функция range

Цикл for удобно использовать совместно с функцией range. Функция range() используется для генерации последовательности чисел. Если мы

передадим `range(10)`, она сгенерирует числа от 0 до 9. Синтаксис функции `range()` приведен ниже.

```
range(stop)
или
range(start, stop, step)
```

Start – означает начало итерации.

Stop – означает, что цикл будет повторяться до `stop-1`. `range(1,5)` будет генерировать числа от 1 до 4 итераций. Это необязательный параметр.

Размер шага используется для пропуска определенных чисел в итерации. Его использование необязательно. По умолчанию размер шага равен 1. Это необязательно.

Пример: Программа для печати чисел по порядку.

```
for i in range(10):
    print(i)
```

Вывод:

```
0
1
2
3
4
5
6
7
8
9
```

Пример: Программа для печати таблицы умножения заданного числа.

```
n = int(input("Enter the number "))
for i in range(1,11):
    c = n*i
    print(n, "*", i, "=", c)
```

Вывод:

```
Enter the number 10
10 * 1 = 10
10 * 2 = 20
10 * 3 = 30
10 * 4 = 40
10 * 5 = 50
10 * 6 = 60
10 * 7 = 70
10 * 8 = 80
10 * 9 = 90
10 * 10 = 100
```

Пример: Программа для печати четного числа с использованием размера шага в range().

```
n = int(input("Введите число "))
for i in range(2,n,2):
    print(i)
```

Вывод:

```
Введите число 20
2
4
6
8
10
12
14
16
18
```

Необходимо помнить, что функция range работает только с целыми числами. Для работы с десятичными числами необходимо использовать циклы типа while, либо делить значения, полученных после генерации функцией range на заданное количество знаков после запятой (10,100,1000 и т.д.).

Операторы управления циклом

Мы можем изменить обычную последовательность выполнения цикла while или for с помощью оператора управления циклом. Python предлагает следующие управляющие операторы для использования в циклах.

Оператор **continue** - Когда встречается оператор continue, управление переходит в начало цикла. Иными словами, с помощью оператора **continue** можно пропустить итерацию цикла. Например:

```
# Вывод всех букв кроме 'a' и 't'
i = 0
str1 = 'javatpoint'

while i < len(str1):
    if str1[i] == 'a' or str1[i] == 't':
        i += 1
        continue
    print('Текущая буква :', a[i])
    i += 1
```

Вывод:

```
Текущая буква: j
Текущая буква: v
Текущая буква: p
Текущая буква: o
Текущая буква: i
Текущая буква: n
```

Оператор **break**. Оператор **break** выводит управление из цикла, иными словами он досрочно завершает цикл.

Пример:

```
# Вывод всех букв до тех пор, пока не встретится буква 't'.
i = 0
str1 = 'javatpoint'
while i < len(str1):
    if str1[i] == 't':
        i += 1
        break
    print('Текущая буква :', str1[i])
    i += 1
```

Вывод:

```
Текущая буква : j
Текущая буква : a
Текущая буква : v
Текущая буква : a
```

Обход элементов списка

Для обхода элементов списка обычно применять цикл **for**. Например:

```
l = ['Peter', 'Joseph', 'Ricky', 'Devansh']
for i in l:
    print(i)
```

Вывод:

```
Peter
Joseph
Ricky
Devansh
```

Если нам нужно знать не только значение элемента списка, но и его индекс, тогда можно использовать один из нескольких подходов:

1. Использовать цикл while совместно с функцией len:

```
l = ['Peter', 'Joseph', 'Ricky', 'Devansh']
i=0
while i<len(l):
    print("№",i,l[i])
    i+=1
```

Вывод:

```
№ 0 Peter
№ 1 Joseph
№ 2 Ricky
№ 3 Devansh
```

2. Использовать функцию range совместно с функцией len и циклом for:

Пример:

```
l = ['Peter', 'Joseph', 'Ricky', 'Devansh']
for i in range(len(l)):
    print("№",i,l[i])
```

Вывод:

```
№ 0 Peter
№ 1 Joseph
№ 2 Ricky
№ 3 Devansh
```

3. Использовать функцию enumerate совместно с циклом for:

```
l = ['Peter', 'Joseph', 'Ricky', 'Devansh']
for idx,val in enumerate (l):
    print("№",idx,val)
```

Вывод:

```
№ 0 Peter
№ 1 Joseph
№ 2 Ricky
№ 3 Devansh
```

Все три способа позволяют получить один и тот же результат и могут быть использованы на выбор.

Вложенные циклы

Python позволяет нам вложить любое количество циклов for или while внутрь друг друга. Внутренний цикл выполняется n раз за каждую итерацию внешнего цикла. Синтаксис приведен ниже.

```
for iterating_var1 in sequence:  #Внешний цикл
    for iterating_var2 in sequence:  #Внутренний цикл
        #block of statements
#Other statements
```

Пример: Вложенный цикл for

```
rows = int(input("Введите количество строк:"))
# Внешний цикл напечатает нужно количество строк
for i in range(0,rows+1):
    # Внутренний цикл напечатает нужное количество звездочек в
    # каждой строке
    for j in range(i):
        print("*",end = ' ')
    print()
```

Вывод:

```
Enter the rows:5
*
**
***
****
*****
```

Пример: Программа для печати пирамиды чисел.

```
rows = int(input("Введите количество строк"))
for i in range(0,rows+1):
    for j in range(i):
        print(i,end = ' ')
    print()
```

Вывод:

```
1
22
333
4444
55555
```

В приведенных примерах встречается ранее не рассмотренный дополнительный аргумент функции print – «end». Он позволяет поменять символ окончания строки. По умолчанию символом окончания является символ '\n', что означает закончить строку и начать новую. В данном случае мы меняем его на пустой символ: end = ". У функции print есть еще один дополнительный аргумент: sep. В случае, если для вывода на экран в

функцию `print` передается несколько переменных, он позволяет задавать разделительный символ между этими элементами. По умолчанию `sep` равен пробелу.

Генераторы списков

Для генерации предварительно заполненных списков можно использовать несколько вариантов:

1. Использовать функцию `range()`:

```
l=list(range(1,10,2))
print(l)
```

Результат:

```
[1, 3, 5, 7, 9]
```

Такой подход возможен, если нам нужно сформировать последовательность чисел с определенной арифметической прогрессией.

2. Использовать функцию `append` совместно с циклами `for` или `while`:

```
l=[]
for i in range(10):
    l.append(input("l["+str(i)+"]="))
```

3. Использовать генераторы списков (list comprehension):

Генераторы списков имеют следующий синтаксис:

```
[ <expression> for <element> in <iterable> ]
```

Создает новый `list`, применяя выражение `<expression>` к каждому элементу `<element>` итерируемого объекта `<iterable>`.

Приведенный выше пример будет выглядеть следующим образом:

```
l=[input("l["+str(i)+"]=") for i in range(10)]
```

Генераторы списков могут быть вложенными друг в друга:

```
l=[[input("l["+str(j)+"]"+"["+str(i)+"]=") for i in range(3)]
for j in range(3)]
```

Этот пример создаст матрицу 3x3.

Генератор случайных чисел

Для генерации случайных чисел в языке Python существует модуль `random`. Для его использования необходимо в самом начале программы добавить строчку

```
import random
```

Для генерации случайного целого числа существует функция `randint()`, а для генерации случайного вещественного числа необходимо использовать функцию `uniform()`. Пример:

```
import random
a=random.randint(0,10)
b= random.uniform(0,10)
print(a)
print(b)
```

Результат:

```
7
3.8287008131361047
```

Задание

Таблица 2.1 – Задания для самостоятельного выполнения:

№ варианта	Задачи для выполнения	№ варианта	Задачи для выполнения
1	9,6,10,11,12	9	8,9,2,11,12
2	10,7,4,11,12	10	5,6,7,11,12
3	7,5,2,11,12	11	9,5,6,11,12
4	6,8,9,11,12	12	5,7,2,11,12
5	6,7,2,11,12	13	7,5,1,11,12
6	7,10,1,11,12	14	3,9,4,11,12
7	9,10,1,11,12	15	6,7,8,11,12
8	10,8,2,11,12	16	1,2,3,11,12

1. Создайте программу, которая выполняет следующие операции: создает матрицу размером 5x5 элементов, заполняет ее случайными числами, выводит на экран созданную матрицу, считает сумму элементов матрицы, вычисляет среднее значение всех элементов матрицы, вычисляет сумму элементов по главной и побочной диагонали матрицы, вычисляет среднее значение по каждой строке и каждому столбцу матрицы.

2. Создайте программу, которая выполняет следующие действия: ждет от пользователя ввода целого числа N в диапазоне от 1 до 10, создает две матрицы размером $N \times N$, заполняет их случайными числами и выводит на экран, выполняет сложение и вычитание этих матрицы, результаты также выводит на экран.

Задача №3. Пользователь вводит с клавиатуры несколько чисел. Количество заранее неизвестно. Ввод заканчивается, как только пользователь введет число 0. Посчитайте арифметическую сумму введенных чисел.

Задача №4. Пользователь вводит с клавиатуры несколько чисел. Количество заранее неизвестно. Ввод заканчивается, как только пользователь введет символ 's'. Посчитайте произведение введенных чисел.

Задача №5. Пользователь вводит с клавиатуры несколько чисел. Количество заранее неизвестно. Ввод заканчивается, как только пользователь введет слово 'break'. Необходимо определить, сколько чисел оказалось больше 10.

Задача №6. Пользователь вводит с клавиатуры несколько чисел. Количество заранее неизвестно. Ввод заканчивается, как только пользователь введет слово 'stop'. Необходимо посчитать количество четных и нечетных чисел.

Задача №7. Пользователь вводит с клавиатуры несколько чисел. Количество заранее неизвестно. Ввод заканчивается, как только пользователь введет число 0. Необходимо посчитать произведение четных и сумму нечетных чисел.

Задача №8. Пользователь вводит с клавиатуры несколько чисел. Количество заранее неизвестно. Ввод заканчивается, как только пользователь введет число 0. Необходимо посчитать сумму квадратов этих чисел.

Задача №10. Пользователь вводит с клавиатуры несколько чисел. Количество заранее неизвестно. Ввод заканчивается, как только пользователь введет число 0. Необходимо четные числа сложить по правилам сложения чисел, а нечетные числа сложить по правилам сложения строк.

Задача №11. У вас есть 3 списка: список удельных активных сопротивлений R , список удельных индуктивных сопротивлений X , и список длин линий L . Создайте список полных комплексных сопротивлений линий Z . Для создания комплексных чисел воспользуйтесь поиском в интернете.

Задача №12. Известно напряжение в начале линии, полная мощность нагрузки в конце линии и ее полное сопротивление Z . Найдите напряжение в конце линии.

Контрольные вопросы

1. Что такое список в языке Python?
2. Как создать список?
3. Какие операции со списками вы знаете?
4. Как получить доступ к отдельным элементам списка?
5. Что такое комментарии в программе? Какие способы создания комментариев вы знаете?
6. Что такое циклы? Какие способы задания циклов вы знаете?
7. Какие операторы управления циклом вы знаете?
8. Как создавать комплексные числа? Какие операции над комплексными числами в языке Python вы знаете?

Практическая работа №3

Тема: «Функции в Python»

Цель: Изучить основы создания собственных функций, научиться разбивать задачу на подзадачи

До сих пор мы писали программы единым, функционально неделимым, кодом. Мы писали маленькие программы, поэтому не было потребности в объявлении своих функций. Для написания больших программ лучше пользоваться функциями.

Функция — это фрагмент программного кода, который решает какую-либо задачу.

Его можно вызывать в любом месте основной программы. Функции помогают избегать дублирования кода при многократном его использовании.

До сих пор мы использовали уже готовые функции языка Python, которые за нас написали другие программисты. Однако невозможно написать заранее функции на все случаи жизни, поэтому рано или поздно придется создавать свои собственные функции, решающие специфические задачи.

Для создания функции существует ключевое слово «def». Далее необходимо указать имя функции и в круглых скобках перечислить ее параметры. Затем идет знак двоеточия и блок команд, которые выполняет функция. Если у функции есть какой-то результат, который необходимо использовать далее, пишется ключевое слово «return» и переменная, в которой хранится этот результат. В общем случае, синтаксис создания функции следующий:

```
def my_function(parameters):  
    function_block  
    return expression
```

Оператор `return` является не обязательным. Давайте рассмотрим простейший случай: функция, которая не имеет параметров и ничего не делает:

```
def simple_function():  
    pass
```

Ключевое слово «`pass`» нужно из-за особенностей языка Python. Интерпретатору языка нужно понимать, где начинается и заканчивается функция. После названия функции должны следовать команды, которые она выполняет. Но так как в нашем случае функция не выполняет никакие команды, нам нужно сообщить об этом с помощью слова `pass`, чтобы интерпретатор понимал, что функция закончилась и дальше уже будет код другой части программы.

Для «запуска», или, как говорят программисты, «вызова» этой функции из основной части нашей программы необходимо записать название этой функции и в круглых скобках передать аргументы. В данном случае функция не имеет параметров, а значит и аргументы передавать не нужно. То есть, вызов функции будет выглядеть следующим образом:

```
simple_function()
```

Пусть теперь наша функция выполняет какие-то полезные действия. Например, выводит что-то на экран:

```
def simple_function():  
    print("Hello, world")  
    print("Этот пример демонстрирует работу с  
функциями")  
    print("Числа от 1 до 10:")  
    for i in range(10):  
        print(i+1)
```

Теперь в результате вызова функции

```
simple_function()
```

на экране появится следующий результат:

```
Hello, world
Этот пример демонстрирует работу с функциями
Числа от 1 до 10:
1
2
3
4
5
6
7
8
9
10
```

Как было уже сказано, важным результатом использования функции является возможность многократного повторения. Например, если нужно 5 раз вывести на экран эту информацию, мы просто выполним 5 раз вызов функции:

```
simple_function()
simple_function()
simple_function()
simple_function()
simple_function()
```

Или еще проще:

```
for i in range(5):
    simple_function()
```

Параметры функции

Рассмотренная функция выполняет всегда одни и те же действия. Но преимущества применения функций заключается еще и в том, что мы можем передавать определенные начальные условия, от которых будет зависеть результат выполнения функции. Эти условия называются параметрами функции. Параметры функции указываются в скобках при объявлении функции. Допустим, в нашем примере мы хотим, чтобы функция выводила имя того, кто ее вызывает, а количество чисел было не определено заранее, а было настраиваемым. В этом случае, наша функция будет иметь следующий вид:

```
def simple_function(name, num):  
    print("Hello, world")  
    print("Меня зовут ", name)  
    print("Этот пример демонстрирует работу с  
функциями")  
    print("Числа от 1 до ", num, ":")  
    for i in range(num):  
        print(i+1)
```

name и num – это **параметры** функции. Теперь для того, чтобы вызвать эту функцию в нее нужно передать **аргументы**:

```
simple_function("Иван", 5)
```

Результат будет следующий:

```
Hello, world  
Меня зовут Иван  
Этот пример демонстрирует работу с функциями  
Числа от 1 до 5 :  
1  
2  
3  
4  
5
```

“Иван” и 5 – это **аргументы** функции.

Существует несколько способов передачи аргументов в функцию:

1. Рассмотренный случай называется позиционным способом передачи аргументов. Функция определяет где какой параметр согласно порядку, в котором передаются аргументы. То есть,

```
simple_function("Иван", 5)
```

и

```
simple_function(5, "Иван")
```

не равнозначны друг другу. Второй вариант вызовет ошибку, так как функция попытается сгенерировать числа от 1 до Иван, что невозможно.

2. Существует также второй способ передачи аргументов, который называется именованный. В этом случае порядок передачи аргументов значения не имеет, но нужно явно указать имя каждого параметра. Например:

```
simple_function(name="Иван", num=5)
```

или

```
simple_function(num=5, name="Иван")
```

Эти два вызова абсолютно равнозначны.

3. Смешанный способ. В этом случае часть аргументов передается по порядку, а оставшаяся часть может передаваться по имени. Например:

```
simple_function("Иван", num=5)
```

В этом примере целесообразность использования этого метода неочевидна, но в некоторых случаях, когда количество параметров в функции велико, этот способ может быть удобен.

4. Необязательные параметры. В приведенном примере оба параметра являются обязательными. То есть, если мы выполним вызов функции, указав только один параметр:

```
simple_function("Иван")
```

или

```
simple_function(5)
```

программа выдаст ошибку. Но в языке Python существует возможность задавать значения параметров по умолчанию. Например, пусть, если пользователь явно не указал, по умолчанию будут выводиться числа от 1 до 10. В этом случае в списке параметров нашей функции после переменной `num` нужно дописать «`=10`»:

```
def simple_function(name, num=10) :  
    print("Hello, world")  
    print("Меня зовут ", name)  
    print("Этот пример демонстрирует работу с  
функциями")  
    print("Числа от 1 до ", num, ":")  
    for i in range(num):  
        print(i+1)
```

Теперь вызов функции:

```
simple_function("Иван")
```

не будет вызывать ошибок.

Значения по умолчанию можно задавать для любого количества параметров. Важно соблюдать одно правило: как только мы задали для

какого-то параметра значение по умолчанию, для всех оставшихся параметров тоже должны быть заданы значений по умолчанию, иначе будет возникать ошибка.

Функции с переменным количеством аргументов

В языке Python существует также возможность создавать функции с заранее неизвестным количеством аргументов. Например, если мы хотим посчитать сумму нескольких чисел, но заранее не знаем, сколько чисел будет, можно использовать в списке параметров знак звездочка:

```
def summa(name, *nums):  
    print("Меня зовут ", name)  
    s=0  
    for i in nums:  
        s+=i  
    return s  
  
a=summa("Иван", 5, 6, 7)  
print(a)
```

Важно помнить, что такой параметр должен быть всегда один в функции и быть самым последним в списке неименованных аргументов (если их несколько).

Возвращаемое значение

Функция может выполнять какие-то действия и не иметь численного результата. Например, приведенная выше функция только выводила на экран сообщения. Такие функции еще иногда называют процедурами. Но функция может иметь какой-то результат. Например, пусть приведенная выше функция в качестве результата будет иметь сумму выведенных на экран чисел. Чтобы использовать получившееся число в программе, его нужно «вернуть» в точку вызова с помощью ключевого слова return:

```
def simple_function(name, num=10):  
    print("Hello, world")  
    print("Меня зовут ", name)
```

```
print("Этот пример демонстрирует работу с  
функциями")  
s=0  
print("Числа от 1 до ", num, ":")  
for i in range(num):  
    print(i+1)  
    s+=i+1  
return s
```

Теперь мы можем использовать полученное число, приравняв переменной вызов функции:

```
a=simple_function("Иван", 5)
```

В переменной а теперь будет храниться результат вычисления суммы чисел 1+2+3+4+5. Мы можем вывести его на экран, либо использовать в дальнейших действиях:

```
a=simple_function("Иван", 5)  
print(a)
```

Функция может возвращать не одно, а несколько значений. Например, если мы хотим посчитать не только сумму чисел от 1 до num, но и сумму квадратов этих чисел, мы можем доработать нашу функцию:

```
def simple_function(name, num=10):  
    print("Hello, world")  
    print("Меня зовут ", name)  
    print("Этот пример демонстрирует работу с  
функциями")  
    s=0  
    s2=0  
    print("Числа от 1 до ", num, ":")  
    for i in range(num):  
        print(i+1)  
        s+=i+1  
        s2+=(i+1)**2  
    return s, s2
```

Результаты выполнения нашей функции можно присвоить сразу двум переменным:

```
a,b=simple_function("Иван",5)
print(a,b)
```

Важные замечания:

1. Необходимо помнить, что в качестве параметров и возвращаемых значений могут выступать не только простые типы данных, но и более сложные, например уже рассмотренные списки.

2. Если в функцию параметры не того типа, может возникнуть ошибка. Поэтому необходимо всегда четко отслеживать, переменные каких типов мы передаем в функцию.

3. Важно иметь ввиду, что функция – это изолированный участок кода. Все переменные, которые мы добавим внутри функции, за ее пределами не будут доступны. Точно так же, как переменные, объявленные вне функции, не могут быть использованы внутри функции.

4. Важным является то, что описать функцию нам нужно до того, как мы будем ее использовать. Это связано с тем, что, как было сказано ранее, программа выполняется построчно, одна за другой. Поэтому, если в тексте программы встречается вызов функции, которая не была описана ранее, возникает ошибка. Например:

```
def my_sum(a,b):
    res=a+b
    return res

print(my_sum(1,2))
```

В результате на экране выведется число 3. Но если мы напишем следующим образом:

```
print(my_sum(1,2))

def my_sum(a,b):
    res=a+b
    return res
```

то программа выведет сообщение об ошибке.

Дополнительные параметры функции print

В предыдущих работах мы многократно использовали функцию print и передавали ей аргумент – строку или число для вывода на экран. Но на самом деле функция print может иметь не один, а множество параметров. Рассмотрим их. Полная версия print выглядит так:

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

*objects – объект/объекты которые необходимо вывести;

sep – разделитель между объектами. В качестве своего значения можно передавать строку или None (по умолчанию пробел " ");

end – символ в конце строки (по умолчанию перенос строки \n);

file – объект для вывода (по умолчанию экран);

flush – принудительный вывод, даже если строка не закончена (по умолчанию False).

Как можно видеть, функция имеет много параметров по умолчанию, которые позволяют гибко настроить работу функции.

Например, если мы хотим вывести через запятую 5 чисел, можно написать следующий код:

```
print(1, 2, 3, 4, 5, sep=', ')
```

Форматирование строк

Если нам нужно вывести какую-то сложную фразу, можно использовать функцию print с переменным числом аргументов. Например, если мы хотим вывести фразу:

Меня зовут NAME. Мне XX лет. Где NAME и XX – заранее неопределенные переменные, то мы можем написать следующий код:

```
NAME="Иван"  
XX=20  
print("Меня зовут ", NAME, ". Мне ", XX, " лет")
```

Однако такой способ не всегда удобен. Кроме того, он не позволяет настроить, например, количество знаков после запятой для вывода вещественных чисел. Для решений этих проблем в языке Python есть так называемая интерполяция строк. Выглядит следующим образом:

```
NAME="Иван"  
XX=20  
print(f"Меня зовут {NAME}. Мне {XX} лет")
```

Как видно из приведенного примера, для запуска интерполяции строк, необходимо перед началом строки добавить букву f. Далее в строке используются фигурные скобки. Вместо них подставляются конкретные значения переменных. Важно отметить, что вместо переменных мы можем использовать имена функций и вообще любые выражения. Для настройки количества знаков после запятой, можно воспользоваться следующим синтаксисом:

```
print(f"Меня зовут {NAME}. Мне {XX:6.3f} лет")
```

Как видно из примера, после переменной XX ставится символ двоеточие и далее указывается общая длина выводимого числа и количество знаков после запятой. Далее ставится символ f, что означает fixed-point (с фиксированной точкой). Первое число 6 можно не указывать:

```
print(f"Меня зовут {NAME}. Мне {XX:.3f} лет")
```

Задание

1. Ознакомьте с теоретическим материалом.
2. Ответьте на контрольные вопросы, приведенные в конце работы.
3. Выполните индивидуальное задание, согласно вашему номеру варианта (Таблица 3.1).

Задача №1. Пользователь вводит с клавиатуры несколько чисел. Количество заранее неизвестно, но известно, что ввод должен закончиться, как только пользователь введет символ конца ввода. Заранее этот символ

неизвестен, пользователь должен ввести его сам. Напишите функцию, которая будет иметь один аргумент – символ конца ввода, создавать список введенных пользователем чисел и возвращать его, как только пользователь введет символ конца ввода.

Задача №2. Создайте функцию для решения квадратного уравнения. Используйте комментарии и список. Используйте функцию ввода, созданную вами в задаче №1

Задача №3. Пользователь вводит с клавиатуры несколько чисел. Количество заранее неизвестно. Ввод заканчивается, как только пользователь введет число 0. Напишите функцию вычисления арифметической суммы введенных чисел. Используйте комментарии, циклы, списки. Используйте функцию ввода, созданную вами в задаче №1.

Задача №4. Пользователь вводит с клавиатуры несколько чисел. Количество заранее неизвестно. Ввод заканчивается, как только пользователь введет символ 's'. Напишите функцию вычисления произведения введенных чисел. Используйте комментарии, циклы, списки. Используйте функцию ввода, созданную вами в задаче №1

Задача №5. Пользователь вводит с клавиатуры несколько чисел. Количество заранее неизвестно. Ввод заканчивается, как только пользователь введет слово 'break'. Напишите функцию, которая подсчитывает, сколько чисел оказалось больше 10. Используйте комментарии, циклы, списки. Используйте функцию ввода, созданную вами в задаче №1

Задача №6. Пользователь вводит с клавиатуры несколько чисел. Количество заранее неизвестно. Ввод заканчивается, как только пользователь введет слово 'stop'. Необходимо создать функцию, которая будет считать количество четных и нечетных чисел. Используйте комментарии, циклы, списки. Используйте функцию ввода, созданную вами в задаче №1.

Задача №7. Пользователь вводит с клавиатуры несколько чисел. Количество заранее неизвестно. Ввод заканчивается, как только пользователь введет число 0. Напишите функцию, которая будет считать произведение

четных и сумму нечетных чисел. Используйте комментарии, циклы, списки. Используйте функцию ввода, созданную вами в задаче №1.

Задача №8. Пользователь вводит с клавиатуры несколько чисел. Количество заранее неизвестно. Ввод заканчивается, как только пользователь введет число 0. Необходимо создать функцию, которая будет считать сумму квадратов этих чисел.

Задача №9. Напишите следующие функции:

- вычисление периметра прямоугольника;
- вычисление площади прямоугольника;
- вычисление длины диагонали прямоугольника.

Задача №10. Пользователь вводит с клавиатуры несколько чисел. Количество заранее неизвестно. Ввод заканчивается, как только пользователь введет число 0. Необходимо четные числа сложить по правилам сложения чисел, а нечетные числа сложить по правилам сложения строк. Используйте комментарии, циклы, списки. Используйте функцию ввода, созданную вами в задаче №1.

Задача №11.

Известно напряжение в начале и конце линии, и ее сопротивление. Напишите функцию вычисления тока в линии и нагрузочных потерь.

Задача №12.

Напишите функцию вычисления параметров схемы замещения трансформатора по его паспортным данным.

Таблица 3.1 – Варианты заданий

№ варианта	Задачи для выполнения	№ варианта	Задачи для выполнения
1	1,9,6	9	1,8,9
2	1,10,7	10	1,5,6
3	1,7,5	11	1,9,5
4	1,6,8	12	1,5,7
5	1,6,7	13	1,7,5
6	1,7,10	14	1,3,9
7	1,9,10	15	1,6,7
8	1,10,8	16	1,2,3,11,12

Контрольные вопросы:

1. Что такое функция в языке Python? Для чего они нужны?
2. Как создать свою функцию?
3. Что такое параметры функции?
4. Что такое аргументы функции?
5. Какие способы передачи аргументов вы знаете?
6. Что такое возвращаемое значение?
7. Какие дополнительные аргументы функции print вы знаете?
8. Как можно настроить количество знаков после запятой при выводе на экран?

Практическая работа №4

Тема: «Файлы, словари, кортежи»

Цель: Научиться работать с текстовыми файлами и словарями

Словари и кортежи

На предыдущих занятиях был рассмотрен такой полезный тип данных как список. Помимо списков в языке Python существуют еще несколько удобных встроенных типов данных. К ним относятся словари и кортежи.

Словари (dict) – это одна из наиболее часто используемых структур данных, позволяющая хранить объекты, для доступа к которым используется ключ. Данные в словаре хранятся в формате ключ – значение. Если вспомнить такую структуру как список, то доступ к его элементам осуществляется по индексу, который представляет собой целое число. Причем индексы создаются автоматически, независимо от нас. В словаре аналогом индекса является ключ, при этом ответственность за его создание ложится на программиста.

Пустой словарь можно создать, используя функцию dict(), либо просто указав пустые фигурные скобки.

```
>>> d1 = dict()
>>> print(type(d1))
<class 'dict'>
>>> d2 = {}
>>> print(type(d2))
<class 'dict'>
```

Если необходимо создать словарь с заранее подготовленным набором данных, то можно использовать один из перечисленных выше подходов, но с перечислением групп ключ-значение.

```
>>> d1 = dict(Ivan="manager", Mark="worker")
>>> print(d1)
{'Mark': 'worker', 'Ivan': 'manager'}
>>> d2 = {"A1": "123", "A2": "456"}
>>> print(d2)
{'A2': '456', 'A1': '123'}
```

Чтобы добавить элемент в словарь нужно указать новый ключ и значение.

```
>>> d1 = {"Russia": "Moscow", "USA": "Washington"}
>>> d1["China"] = "Beijing"
>>> print(d1)
{'Russia': 'Moscow', 'China': 'Beijing', 'USA': 'Washington'}
```

Для удаления элемента из словаря можно воспользоваться командой `del`, либо функцией словаря `pop()`. Пример:

```
>>> d2 = {"A1": "123", "A2": "456"}
>>> del d2["A1"]
>>> print(d2)
{'A2': '456'}
```

```
>>> d = {"A1": "123", "A2": "456"}
>>> d.pop("A1")
'123'
>>> print(d)
{'A2': '456'}
```

Как видно из примера, функция `pop` помимо удаления еще и в качестве результата выполнения возвращает значение удаленного элемента. В некоторых ситуациях это бывает полезно.

Доступ к элементу словаря, осуществляется так же как доступ к элементу списка, только в качестве индекса указывается ключ.

```
>>> d1 = {"Russia": "Moscow", "USA": "Washington"}
>>> d1["Russia"]
'Moscow'
```

Проверить наличие ключа в словаре можно с помощью оператора `in`:

```
>>> d2 = {"A1": "123", "A2": "456"}
>>> "A1" in d2
True
>>> "A3" in d2
False
```

У словарей доступен следующий набор дополнительных функций:

– **`clear()`**. Удаляет все элементы словаря. Пример:

```
>>> d2 = {"A1": "123", "A2": "456"}
>>> print(d2)
{'A2': '456', 'A1': '123'}
>>> d2.clear()
>>> print(d2)
{}
```

– **get(key,default=None)**. Возвращает значение из словаря по ключу key.

Если ключ не найден, возвращает переменную default. Пример:

```
>>> d = {"A1": "123", "A2": "456"}
>>> d.get("A1")
'123'
```

– **keys()**. Возвращает все ключи словаря.

```
>>> d = {"A1": "123", "A2": "456"}
>>> d.keys()
dict_keys(['A2', 'A1'])
```

Функция keys() может быть полезна, когда нужно со всеми ключами словаря провести какие-то действия (самый простой вариант – вывести на экран). В это случае функция keys() может быть использована в цикле

```
d = {"A1": "123", "A2": "456"}
for k in d.keys():
    print(k)
```

На экране будут выведены строки «A1» и «A2».

– **values()**. Функция возвращает значения элементов словаря:

```
>>> d = {"A1": "123", "A2": "456"}
>>> d.values()
dict_values(['456', '123'])
```

Аналогично с функцией keys(), функция values() будет полезна, когда нужно произвести какие-то действия со всеми значениями словаря через цикл:

```
d = {"A1": "123", "A2": "456"}
for k in d.values():
    print(k)
```

В результате выполнения на экран будет выведены значения «123» и «456».

– **items()**. Возвращает элементы словаря (ключ, значение). По сути является комбинацией двух предыдущих функций. Поэтому будет полезна, когда через цикл необходимо выполнить какие-то действия со всеми парами «ключ» - «значение» словаря. Например:

```
d = {"A1": "123", "A2": "456"}
for k,v in d.items():
    print(k, "->", v)
```

На экран будет выведен следующий текст:

```
A1 -> 123
A2 -> 456
```

В качестве ключа могут выступать практически любые типы данных, за некоторым исключением. Например, целые и дробные числа, строки, логические переменные могут выступать в качестве ключа, а вот списки в качестве ключа выступать не могут. Дело в том, что считается, что список в любой момент времени может измениться (в него могут быть добавлены новые, удалены старые элементы и так далее). Чтобы исправить эту ситуацию, в языке Python существует очень похожий на списки тип данных, который называется кортежем.

Кортеж – последовательность элементов, которые разделены между собой запятой и заключены в скобки, это неизменяемый упорядоченный тип данных. Грубо говоря, кортеж - это список, который нельзя изменить. То есть, в кортеже есть только права на чтение. Поэтому элементы кортежа должны добавляться в него непосредственно в момент его создания. Создаются кортежи почти точно так же, как и списки, только скобки ставятся не квадратные, а круглые.

Например:

```
T = ('login', 'password')
```

Точно также, как строка может быть преобразована в число, а число в строку, ранее созданный список может быть преобразован в кортеж, а кортеж может быть преобразован в список. Для создания кортежа из списка существует функция `tuple()`, а для создания списка из кортежа существует функция `list()`:

```
sp=['a','b','c']  
tp=(1,2,3)  
sp2=list(tp)  
tp2=tuple(sp)
```

Для доступа к элементам кортежа точно также используются целочисленные индексы.

Работа с файлами

До сих пор для ввода и вывода данных мы использовали экран и

клавиатуру. Но довольно часто получать данные приходится не с клавиатуры, а из файлов. Для открытия файла используется функция `open()`, которая возвращает файловый объект. Наиболее часто используемый вид данной функции выглядит так

```
open(имя_файла, режим_доступа)
```

Имя файла может быть относительным (относительно исполняемой программы) или абсолютным (полный путь к файлу с указанием буквы диска).

Для указания режима доступа используются следующие символы:

‘r’ – открыть файл для чтения;

‘w’ – открыть файл для записи;

‘x’ – открыть файл с целью создания, если файл существует, то вызов функции `open` завершится с ошибкой;

‘a’ – открыть файл для записи, при этом новые данные будут добавлены в конец файла, без удаления существующих;

‘b’ – бинарный режим;

‘t’ – текстовый режим;

‘+’ – открывает файл для обновления.

По умолчанию файл открывается на чтение в текстовом режиме.

У файлового объекта есть следующие атрибуты.

file.closed – возвращает `true` если файл закрыт и `false` в противном случае;

file.mode – возвращает режим доступа к файлу, при этом файл должен быть открыт;

file.name – имя файла.

```
>>> f = open("test.txt", "r")
>>> print("file.closed: " + str(f.closed))
file.closed: False
>>> print("file.mode: " + f.mode)
file.mode: r
>>> print("file.name: " + f.name)
file.name: test.txt
```

Для закрытия файла используется метод `close()`.

Чтение данных из файла осуществляется с помощью методов `read(размер)` и `readline()`.

Метод `read(размер)` считывает из файла определенное количество символов, переданное в качестве аргумента. Если использовать этот метод без аргументов, то будет считан весь файл.

```
>>> f = open("test.txt", "r")
>>> f.read()
'1 2 3 4 5\nWork with file\n'
>>> f.close()
```

В качестве аргумента метода можно передать количество символом, которое нужно считать.

```
>>> f = open("test.txt", "r")
>>> f.read(5)
'1 2 3'
>>> f.close()
```

Метод `readline()` позволяет считать строку из открытого файла.

```
>>> f = open("test.txt", "r")
>>> f.readline()
'1 2 3 4 5\n'
>>> f.close()
```

Построчное считывание можно организовать с помощью оператора `for`.

```
>>> f = open("test.txt", "r")
>>> for line in f:
...     print(line)
...
1 2 3 4 5
Work with file

>>> f.close()
```

Для записи данных файл используется метод `write(строка)`, при успешной записи он вернет количество записанных символов.

```
>>> f = open("test.txt", "a")
>>> f.write("Test string")
11
>>> f.close()
```

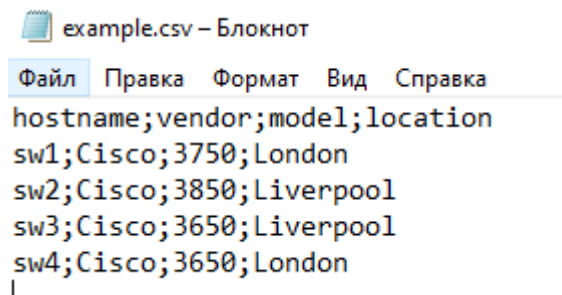
Также для записи в файл удобно использовать уже известную нам функцию `print`, если передать ей в качестве аргумента открытый файл. Например:

```
print("Hello world") # Выведет на экран

f=open("E:\\out.txt", "w")|
print("Hello world", file=f) # Выведет в файл
f.close()
```

Текстовые файлы с разделителями

Довольно часто используются так называемые текстовые файлы с разделителями. В каждой строке такого файла записан не один параметр, а несколько, отделенные друг от друга пробелом, либо другим символом, например, ';' или ','. Такие файлы называются CSV. CSV (comma-separated value) – это формат представления табличных данных (например, это могут быть данные из таблицы или данные из БД). В этом формате каждая строка файла – это строка таблицы. Несмотря на название формата, разделителем может быть не только запятая. И хотя у форматов с другим разделителем может быть и собственное название, например, TSV (tab separated values), тем не менее, под форматом CSV понимают, как правило, любые разделители. Пример файла в формате CSV:



```
example.csv – Блокнот
Файл  Правка  Формат  Вид  Справка
hostname;vendor;model;location
sw1;Cisco;3750;London
sw2;Cisco;3850;Liverpool
sw3;Cisco;3650;Liverpool
sw4;Cisco;3650;London
```

В стандартной библиотеке Python есть модуль csv, который позволяет работать с файлами в CSV формате. Для его подключения необходимо в начале программы добавить строчку «import csv».

Пример чтения файлов в формате csv:

```
import csv

file=open("E:\\example.csv")|
reader=csv.reader(file,delimiter=';')
for line in reader:
    print(line)
```

Чаще всего заголовки столбцов удобнее получить отдельным объектом. Это можно сделать таким образом:

```
import csv
|
file=open("E:\\example.csv")
reader=csv.reader(file,delimiter=';')
headers = next(reader)
print('Заголовки: ', headers)
for line in reader:
    print(line)
```

Иногда в результате обработки гораздо удобнее получить словари, в которых ключи – это названия столбцов, а значения – значения столбцов. Для этого в модуле есть функция DictReader.

```
import csv

file=open("E:\\example.csv")
reader=csv.DictReader(file,delimiter=';')
for line in reader:
    print(line)
```

Аналогичным образом с помощью модуля csv можно и записать файл в формате CSV.

```
import csv

data = [['hostname', 'vendor', 'model', 'location'],
        ['sw1', 'Cisco', '3750', 'London, Best str'],
        ['sw2', 'Cisco', '3850', 'Liverpool, Better str'],
        ['sw3', 'Cisco', '3650', 'Liverpool, Better str'],
        ['sw4', 'Cisco', '3650', 'London, Best str']]

file=open("E:\\example_write.csv", "w", newline='')
wr=csv.writer(file,delimiter=";")
for row in data:
    wr.writerow(row)
file.close()
```

С помощью DictWriter можно записать словари в формат CSV. В целом DictWriter работает так же, как writer, но так как словари не упорядочены, надо указывать явно в каком порядке будут идти столбцы в файле. Для этого используется параметр fieldnames.

```
import csv

data = [
    {'hostname': 'sw1', 'location': 'London', 'model': '3750', 'vendor': 'Cisco'},
    {'hostname': 'sw2', 'location': 'Liverpool', 'model': '3850', 'vendor': 'Cisco'},
    {'hostname': 'sw3', 'location': 'Liverpool', 'model': '3650', 'vendor': 'Cisco'},
    {'hostname': 'sw4', 'location': 'London', 'model': '3650', 'vendor': 'Cisco'}
]

file=open("E:\\example_write.csv", "w", newline="")
writer = csv.DictWriter(file, fieldnames=["hostname", "location", "model", "vendor"], delimiter=";")
writer.writeheader()
for d in data:
    writer.writerow(d)
file.close()
```

Задание

1. Создайте программу, которая выполняет сложение матриц типа. Матрицы должны храниться в текстовых файлах. Результат умножения матриц должен выводиться на экран и сохраняться в отдельный файл.

2. Разработайте форматы входного и выходного файлов, согласно вашему заданию на расчетно-графическую работу. Создайте функции чтения исходных данных из этих файлов и записи результатов вычисления в эти файлы. Используйте словари.

Контрольные вопросы

1. Что такое словари?
2. Что такое кортежи?
3. В чем отличие словарей, списков и кортежей?
4. Как добавлять элемент в словарь?
5. Как удалить элемент из словаря?
6. Как прочитать значение в словаре по ключу?
7. Какие функции для работы с файлами в Python вы знаете?

Практическая работа №5

Тема: Объектно-ориентированное программирование

Цель: Изучить основные понятия объектно-ориентированного программирования, научиться создавать свои классы

До сих пор мы разрабатывали программы, используя методологию структурного программирования. Структурное программирование – методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков.

Методология структурного программирования предложена в 1970-х годах и была прорывной для того времени, так как позволила формализовать процесс разработки программ, выделять часто используемые блоки кода и многократно их использовать.

Однако с ростом сложности программ начали проявляться и недостатки структурного программирования. Основной недостаток структурного программирования заключается в том, что данные (переменные) и подпрограммы (функции) формально не связаны, что плохо подходит для описания реальной картины мира.

Объектно-ориентированное программирование (ООП) – подход к программированию, при котором основными концепциями являются понятия объектов и классов.

Класс – это определяемый разработчиком тип данных.

Тип данных характеризуется:

- способом хранения и представления этих данных;
- назначением этих данных;
- набором действий, которые над этими данными могут производиться.

Чтобы понять, для чего же в действительности нужны классы, проведём аналогию с каким-нибудь объектом из повседневной жизни, например, с велосипедом. Велосипед — это объект, который был построен согласно чертежам.

Так вот, эти самые чертежи играют роль классов в ООП.

Таким образом классы — это некоторые описания, схемы, чертежи по которым создаются объекты. Для создания объекта в ООП необходимо сначала составить чертежи, то есть классы.

Классы имеют свои функции, которые называются методами класса и свои параметры, которые называются полями, или свойствами класса.

Создание класса в Python начинается с инструкции `class`. Вот так будет выглядеть минимальный класс:

```
1. class C:  
2.     pass
```

Класс состоит из объявления (инструкция `class`), имени класса (нашем случае это имя `C`) и тела класса, которое содержит атрибуты и методы (в нашем минимальном классе есть только одна инструкция `pass`).

Для того чтобы создать объект класса необходимо воспользоваться следующим синтаксисом:

имя_объекта = имя_класса()

Как уже было сказано, класс может содержать атрибуты и методы. Добавление методов происходит аналогично созданию функций. Но есть одна особенность языка Python, которую нужно иметь в виду. Первым аргументом метода должно быть ключевое слово «`self`». `self` — это ссылка на текущий экземпляр класса. Названия методов пользователь придумывает сам. Но есть некоторые заранее зарезервированные имена методов, которые выполняют особую функцию. Один из таких методов — метод «`__init__`». Он вызывается каждый раз при создании нового объекта. В этом методе должны быть описаны все атрибуты класса.

Для наглядного примера, опишем класс «узел сети». Узел сети обладает следующими основными свойствами:

- номер узла;
- имя узла;

- тип узла (базисный или нагрузочный);
- номинальное напряжение узла;
- активная мощность нагрузки;
- реактивная мощность нагрузки;
- расчетное напряжение.

Давайте подумаем, какие типы данных лучше всего подходят для моделирования этих свойств в программе.

Номер узла – это целое число, поэтому его проще всего моделировать с помощью переменной типа `int`.

Имя узла представляет собой строку.

Тип узла может принимать два значения – базисный и нагрузочный. Поэтому его лучше моделировать с помощью переменной типа `bool`.

Номинальное напряжение, активная и реактивная мощность нагрузки представляют собой десятичные числа, поэтому целесообразно моделировать с помощью переменных `float`.

Расчетное напряжение представляет собой комплексное число. Для его моделирования можно использовать две переменные типа `float`. Первое число будет моделировать модуль напряжения, второе – фазовый угол. Однако в данном случае целесообразно использовать комплексное число, поскольку это позволит использовать уравнения расчета режимов в комплексной форме.

С учетом сказанного, описания класса узла будет выглядеть следующим образом:

```

3 class NODE:
4     def __init__(self):
5         self.ny      = 0          # Номер узла
6         self.name     = ""        # Имя узла
7         self.type     = False     # Тип узла False - Базисный, True - нагрузочный
8         self.unom     = 0.0       # Номинальное напряжение
9         self.P        = 0.0       # Активная мощность нагрузки
10        self.Q        = 0.0       # Реактивная мощность нагрузки
11        self.V        = 0+0j      # Расчетное напряжение

```

Для того, чтобы использовать класс «Узел» в нашей программе, необходимо написать следующий код:

```
11 | nod=NODE ()|
```

nod – это название переменной типа NODE. Теперь мы можем обращаться к свойствам этой переменной через знак точка и выполнять с этими свойствами любые операции. Приведенный ниже пример создает объект (переменную) типа «Узел» со следующими свойствами: номер узла – 1, название – «Узел1», тип – «нагрузочный», номинальное напряжение – «110», активная нагрузка – «10», реактивная нагрузка – «5» и выводит на экран.

```
1 import cmath
2
3 class NODE:
4     def __init__(self):
5         self.ny      = 0          # Номер узла
6         self.name     = ""        # Имя узла
7         self.type     = False     # Тип узла False - Базисный, True - нагрузочный
8         self.unom     = 0.0       # Номинальное напряжение
9         self.P        = 0.0       # Активная мощность нагрузки
10        self.Q         = 0.0       # Реактивная мощность нагрузки
11        self.V         = 0+0j      # Расчетное напряжение
12
13 nod=NODE ()
14 nod.ny=1
15 nod.name="Узел1"
16 nod.type=True
17 nod.unom=110.0
18 nod.P=10.0
19 nod.Q=5.0
20 nod.V=110+10j
21
22 print("Номер узла: ",nod.ny)
23 print("Имя узла: ",nod.name)
24
25 if nod.type==False:
26     print("Тип узла: базисный")
27 else:
28     print("Тип узла: нагрузочный")
29
30 print("Номинальное напряжение: ",nod.unom)
31 print("P: ",nod.P)
32 print("Q: ",nod.Q)
33 print("Модуль напряжения: ",abs(nod.V))
34 print("Фазовый угол: ",cmath.phase(nod.V)*180/cmath.pi)
```

В последней строчке используется функция для вычисления фазового угла комплексного числа и ее перевода из радиан в градусы. Для ее использования необходимо подключить модуль «cmath» через команду import. Модуль «cmath» содержит в себе набор функций для работы с комплексными числами.

Как видно из примера, доступ к свойствам (полям) объекта осуществляется через операцию точка «.». Через операцию «.» возможен доступ только не только к атрибутам объекта, но и к его методам.

Аналогичным образом создадим класс «Линия»:

```
14 class LINE:
15     def __init__(self):
16         self.ip      = 0          # Номер узла начала
17         self.iq      = 0          # Номер узла конца
18         self.R        = 0.0       # Активное сопротивление линии
19         self.X        = 0.0       # Реактивное сопротивление линии
20         self.G        = 0.0       # Активная проводимость линии
21         self.B        = 0.0       # Емкостная проводимость линии
22         self.KTR       = 0.0      # Коэффициент трансформации
```

Теперь создадим класс «Схема сети», который будет содержать таблицу узлов, таблицу ветвей, выполнять загрузку схемы сети из файла, сохранение схемы сети в файл и выводить схему сети на экран.

```
25 class SCHEME:
26     def __init__(self):
27         self.nodes=[]             # Таблица узлов
28         self.lines=[]            # Таблица ветвей
29
30     def load_scheme(self,fname):# Загрузить схему из CSV-файла
31         pass
32
33     def save_scheme(self,fname):# Сохранить схему в текстовый файл
34         pass
35
36     def display(self):            # Вывести схему на экран
37         pass
```

Начнем с функции вывода на экран:

```
37     def display(self):            # Вывести схему на экран
38
39         print("Узлы:")
40         for n in self.nodes:
41             for i,v in vars(n).items():
42                 if(i=='V'):
43                     print(abs(v),end="\t")
44                     print(cmath.phase(v)*180/cmath.pi,end="\t")
45                 else:
46                     print(v,end="\t")
47             print()
48
49         print()
50         print("Ветви:")
51         for n in self.lines:
52             for i,v in vars(n).items():
53                 print(v,end="\t")
54         print()
55
```

В данном случае, функция выводит на экране построчно информацию об узлах и ветвях сети. Для расчетного напряжения мы используем отдельный вывод абсолютного значения и фазового угла.

Для сохранения результатов в файл можно использовать эту же функцию, если передать во все вызовы функции print файл, в который мы хотим сохранить. Для этого немного доработаем приведенную выше функцию:

```
73     def display(self, f=None):          # Вывести схему на экран
74
75         print("Узлы:", file=f)
76         for n in self.nodes:
77             for i, v in vars(n).items():
78                 if (i == 'V'):
79                     print(abs(v), end="\t", file=f)
80                     print(cmath.phase(v)*180/cmath.pi, end="\t", file=f)
81                 else:
82                     print(v, end="\t", file=f)
83             print(file=f)
84
85         print(file=f)
86         print("Ветви:", file=f)
87         for n in self.lines:
88             for i, v in vars(n).items():
89                 print(v, end="\t", file=f)
90             print(file=f)
```

Мы добавили в функцию display еще один аргумент f, который по умолчанию отсутствует (f=None). Теперь эту функцию можно использовать и для вывода на экран и для сохранения в файл. Для сохранения в файл необходимое его предварительно открыть, это будем делать в функции save_scheme:

```
51     def save_scheme(self, fname):      # Сохранить схему в файл
52         file=open(fname, "w")
53         self.display(file)
54         file.close();
--
```

Аналогичным образом добавим реализацию функции загрузки схемы сети из файла. Проще всего создавать входной файл в программе Excel в формате CSV. Предположим, что входной файл имеет следующий формат:

	A	B	C	D	E	F	G
1	№	Название	Тип	P	Q	Уном	
2	1	Узел №1		10	30	110	
3	2	Узел №2	Нагр	10	30	110	
4	3	Узел №3	Нагр	10	30	110	
5							
6	№ нач.	№ кон.	R	X	G	B	KTR
7	1	2	0.1	0.05	0	0	0
8	2	3	0.1	0.05	0	0	0
9	1	3	0.1	0.05	0	0	0

То есть он имеет две таблицы – таблицу узлов и таблицу ветвей, разделенных пустой строкой. В блокноте такой файл будет иметь следующий вид:

```

Схема.csv – Блокнот
Файл  Правка  Формат  Вид  Справка
№;Название;Тип;P;Q;Уном;
1;Узел №1;;10;30;110;
2;Узел №2;Нагр;10;30;110;
3;Узел №3;Нагр;10;30;110;
;;;;;
№ нач.;№ кон.;R;X;G;B;KTR
1;2;0.1;0.05;0;0;0
2;3;0.1;0.05;0;0;0
1;3;0.1;0.05;0;0;0

```

Обратите внимание на 2 важных момента:

- для базисного узла необходимо оставить пустое значение в колонке «Тип»;
- в качестве разделителя целой и дробной части используется символ точка.

Для чтения таких файлов можно использовать модуль CSV, описанный ранее.

```

33     def load_scheme(self, fname):      # Считать схему из CSV
34         file=open(fname)
35
36         for line in csv.DictReader(file, delimiter=';'):
37             if (line['№'] == ''):
38                 break
39             nd=NODE(line)
40             self.nodes.append(nd)
41
42         for line in csv.DictReader(file, delimiter=';'):
43             if (line['№ нач.'] == ''):
44                 break
45             ln=LINE(line)
46             self.lines.append(ln)
47         file.close()

```

В приведенном примере CSV файл построчно преобразовывается в словарь и передается в качестве параметра при инициализации в класс NODE и LINE. Поэтому необходимо доработать функцию `__init__` в этих классах:

```

4 class NODE:                                # Класс "Узел"
5     def __init__(self, csv={}):             # Инициализация переменных
6         self.ny = int(csv.get('№', 0))      # Номер узла
7         self.name = csv.get('Название', '') # Название узла
8         self.type = bool(csv.get('Тип', True)) # Тип узла False - Базисный, True - нагрузочный
9         self.unom = float(csv.get('Uном', 0.0)) # Номинальное напряжение
10        self.P = float(csv.get('P', 0.0))    # Активная мощность нагрузки
11        self.Q = float(csv.get('Q', 0.0))    # Реактивная мощность нагрузки
12        self.V = 0+0j                        # Расчетное напряжение

34 class LINE:                                # Класс "Ветвь"
35     def __init__(self, csv={}):             # Инициализация переменных
36         self.ip = int(csv.get('№ нач.', 0)) # Номер узла начала
37         self.iq = int(csv.get('№ кон.', 0)) # Номер узла конца
38         self.R = float(csv.get('R', 0.0))   # Активное сопротивление линии
39         self.X = float(csv.get('X', 0.0))   # Реактивное сопротивление линии
40         self.G = float(csv.get('G', 0.0))   # Активная проводимость линии
41         self.B = float(csv.get('B', 0.0))   # Емкостная проводимость линии
42         self.KTR = float(csv.get('KTR', 0.0)) # Коэффициент трансформации

```

Мы добавили в функции инициализации дополнительный параметр – переменную `csv`, которая по умолчанию равна пустому словарю. И далее мы ищем в этом словаре соответствующие переменные, которые ранее были считаны из строки CSV-файла и преобразуем их в нужный нам тип данных.

После того, как класс схемы сети описан, его можно использовать непосредственно в нашей программе:

```

100 s=SCHEME()
101 s.load_scheme("E:\\Схема.csv");
102 s.display()
103 s.save_scheme("E:\\Результат.txt")
104

```

В строке 100 создается объект типа «Схема».

В строке 101 вызывается метод `load_scheme`, в который передается имя входного файла, содержащего информацию о схеме сети.

В строке 102 вызывается метод, отвечающий за вывод схемы сети на экран.

В строке 103 выполняется сохранение схемы сети в соответствующий файл.

Прежде чем переходить к алгоритмам расчета режима, необходимо еще решить вопрос, каким образом, узлы и ветви будут друг с другом связаны. В приведенном выше примере, таблица узлов и таблица ветвей не зависят друг от друга. Все алгоритмы расчета режима используют уравнения узловых напряжений. Это означает, что каждый узел должен иметь информацию о всех ветвях, которые с ним связаны. Решить эту задачу можно двумя способами:

1. Каждый раз, когда нам нужно узнать, какие ветви связаны, например, с первым узлом, выполнять поиск по таблице ветвей, где узел начала или узел конца равен единице.

2. Выполнить этот поиск один раз и каким-то образом сохранить результаты этого поиска.

Первый способ является наиболее простой и очевидный, однако не совсем рациональный, так как будет выполняться многократное выполнение одних и тех же действий.

Второй способ является более правильным, но для его реализации необходимо добавить некоторые переменные в наши уже созданные классы узла и ветви.

Добавим в класс ветви две переменных: `_ip`, `_iq`. Первая будет хранить в себе узел начала, а вторая – узел конца. По умолчанию зададим этим переменным значение `None`, то есть изначально они ничего не будут хранить. Тогда функция инициализации класса ветви будет выглядеть так:

```

34 class LINE:                                     # Класс "Ветвь"
35     def __init__(self, csv={}):                 # Инициализация переменных
36         self.ip = int(csv.get('№ нач.', 0))     # Номер узла начала
37         self.iq = int(csv.get('№ кон.', 0))     # Номер узла конца
38         self.R = float(csv.get('R', 0.0))       # Активное сопротивление линии
39         self.X = float(csv.get('X', 0.0))       # Реактивное сопротивление линии
40         self.G = float(csv.get('G', 0.0))       # Активная проводимость линии
41         self.B = float(csv.get('B', 0.0))       # Емкостная проводимость линии
42         self.KTR = float(csv.get('KTR', 0.0))   # Коэффициент трансформации
43         self._ip = None                         # Узел начала
44         self._iq = None                         # Узел конца
45

```

Аналогичным образом, можно добавить переменную, хранящую в себе связанную линию и для узла. Но отличие будет заключаться в том, что количество ветвей, связанных с узлом, заранее не известно и может быть любым. Поэтому, для хранения связанных линий целесообразно использовать список. Назовем его `_inc`. Тогда функция инициализации класса узла будет выглядеть следующим образом:

```

4 class NODE:                                     # Класс "Узел"
5     def __init__(self, csv={}):                 # Инициализация переменных
6         self.ny = int(csv.get('№', 0))         # Номер узла
7         self.name = csv.get('Название', '')     # Название узла
8         self.type = bool(csv.get('Тип', True))  # Тип узла False - Базисный, True - нагрузочный
9         self.unom = float(csv.get('Uном', 0.0)) # Номинальное напряжение
10        self.P = float(csv.get('P', 0.0))       # Активная мощность нагрузки
11        self.Q = float(csv.get('Q', 0.0))       # Реактивная мощность нагрузки
12        self.V = 0+0j                           # Расчетное напряжение
13        self._inc = []                          # Список связанных ветвей

```

Тогда, например, для того, чтобы вывести сопротивления всех линий, связанных с этим узлом, необходимо будет выполнить следующую операцию:

```

15         for l in self._inc:
16             print(l.R, l.X)

```

Мы выделили переменные для сохранения связей между узлами и ветвями сети. Теперь необходимо непосредственно связать их друг с другом. Для этого добавим в класс схемы функцию, которая будет отвечать за «сборку» схемы сети, назовем ее `assemble()`.

```

81     def assemble(self):
82         for n in self.nodes:
83             n._inc.clear()
84
85         for l in self.lines:
86             for n in self.nodes:
87
88                 if n.ny==l.ip:
89                     l._ip=n
90                     n._inc.append(l)
91
92                 if n.ny==l.iq:
93                     l._iq=n
94                     n._inc.append(l)
95

```

В строках 82 и 83 выполняется обход всех узлов сети и удаляются все связи со всеми ветвями (это делается на тот случай, если функция вызывается не первый раз и схема уже «собрана»).

В строках 85-94 выполняется цикл по каждой ветви сети.

Для каждой ветви в таблице узлов ищется узел начала и узел конца.

Если узел начала найден (строка 88) то мы его сохраняем в переменную `_ip` ветви, а в список связанных ветвей этого узла добавляем текущую ветвь.

Аналогичные действия происходят, если найден узел конца (строки 92-94).

Поскольку мы добавили в наши классы новые переменные, функции вывода на экране, описанные ранее будут работать не совсем корректно. Они будут выводить эти новые переменные в виде адресов объектов, которые в них хранятся:

```

Узлы:
1      Узел №1 True    110.0   10.0   30.0   0.0   0.0   [<__main__.LINE
object at 0x042ACE10>, <__main__.LINE object at 0x042ACE50>]
2      Узел №2 True    110.0   10.0   30.0   0.0   0.0   [<__main__.LINE
object at 0x042ACE10>, <__main__.LINE object at 0x042ACD70>]
3      Узел №3 True    110.0   10.0   30.0   0.0   0.0   [<__main__.LINE
object at 0x042ACD70>, <__main__.LINE object at 0x042ACE50>]

Ветви:
1      2      0.1      0.05   0.0      0.0      0.0      <__main__.NODE object at
0x042ACC90> <__main__.NODE object at 0x042ACCB0>
2      3      0.1      0.05   0.0      0.0      0.0      <__main__.NODE object at
0x042ACCB0> <__main__.NODE object at 0x042ACD90>
1      3      0.1      0.05   0.0      0.0      0.0      <__main__.NODE object at
0x042ACC90> <__main__.NODE object at 0x042ACD90>

```

Чтобы этого не было, мы специально названия этих переменных начинали со знака подчеркивания, чтобы можно было их как-то отличить от

других переменных. Теперь перед выводом на экран можно проверить название переменной, и если оно начинается со знака подчеркивания, не выводить ее. С учетом сказанного, функция вывода на экран будет доработана следующим образом:

```
73     def display(self, f=None):          # Вывести схему на экран
74
75         print("Узлы:", file=f)
76         for n in self.nodes:
77             for i, v in vars(n).items():
78                 if i[0] == '_':
79                     continue
80
81                 if i == 'v':
82                     print(abs(v), end="\t", file=f)
83                     print(cmath.phase(v) * 180 / cmath.pi, end="\t", file=f)
84                 else:
85                     print(v, end="\t", file=f)
86         print(file=f)
87
88         print(file=f)
89         print("Ветви:", file=f)
90         for n in self.lines:
91             for i, v in vars(n).items():
92                 if i[0] == '_':
93                     continue
94                 print(v, end="\t", file=f)
95         print(file=f)
96
```

Задание

1. Создайте класс, описывающий узел сети, согласно приведенному примеру. Добавьте в него методы, вычисляющие:

- полную мощность нагрузки;
- ток нагрузки;
- собственную узловую проводимость.

2. Создайте класс, описывающий ветвь сети, согласно приведенному примеру. Добавьте в него методы, вычисляющие:

- нагрузочные потери мощности;
- потери мощности холостого хода;
- полные потери мощности;
- падение напряжения в линии;
- поток мощности в начале ветви;

– поток мощности в конце ветви.

3. Создайте класс, описывающий схему сети, согласно приведенному примеру. Добавьте в класс схемы сети справочник типов трансформаторов и типов проводов.

Контрольные вопросы

1. Что такое объектно-ориентированное программирование?
2. Что такое класс? Какие параметры есть у класса?
3. Как объявлять класс?
4. Как создать объект класса?
5. Как использовать атрибуты и методы класса?

Практическая работа №6

Тема: Метод Зейделя

Цель: изучить алгоритм расчета режима методом Зейделя

Расчет установившихся режимов ЭС в классическом виде заключается в определении напряжений в узлах сети, используя которые, затем находят потокораспределение и потери мощности. Математически эта задача формулируется как решение системы нелинейных алгебраических уравнений, описывающих режим. В основе такого описания состояния ЭС лежат законы Кирхгофа и Ома, устанавливающие связи между токами, напряжениями и параметрами сети. Непосредственное применение уравнений Кирхгофа неэффективно и в алгоритмах для ЭВМ не используется. Для построения соответствующих алгоритмов электрического расчета наиболее эффективными и удобными для реализации на ЭВМ являются уравнения узловых напряжений, связывающие напряжения в узлах ЭС и мощности (токи), подводимые к этим узлам, через параметры схемы. Уравнения узловых напряжений (УУН) следуют из первого закона Кирхгофа в результате представления по закону Ома токов во всех ветвях через узловые напряжения и проводимости ветвей.

Учитывая все вышесказанное, установившийся режим электрической системы математически может быть описан двумя группами уравнений.

Первая группа – линейные алгебраические уравнения, связывающие токи и напряжения элементов схемы замещения, представляющих собой линейную электрическую цепь. Состояние линейной электрической цепи может описываться уравнениями, составленными на основе первого закона Кирхгофа и закона Ома – узловыми уравнениями (6.1) или составленными на основе второго закона Кирхгофа и закона Ома – контурными уравнениями (6.2):

$$Y\dot{U} = \sqrt{3}\dot{J}; \quad (6.1)$$

$$Z_K \dot{I}_K = \dot{E}_K. \quad (6.2)$$

Узловые уравнения (6.1) электрической цепи произвольной конфигурации, содержащей n узлов (не считая узла земли), записаны в предположении, что активными элементами этой цепи являются только источники тока, замещающие источники и потребители электроэнергии. Для этих условий в системе n уравнений (6.1)

Y — квадратная матрица узловых проводимостей размерностью $n \times n$;

$\dot{U}$ — вектор-столбец линейных (междуфазных) напряжений всех n узлов;

$\dot{J}$ — вектор-столбец фазных значений токов источников (задающих токов) во всех n узлах, записываемых со знаком плюс для генераторов и со знаком минус для нагрузок (т.е. за положительное направление принято направление к узлу.)

Система контурных уравнений (6.2) электрической цепи произвольной конфигурации, содержащей n узлов (не считая узла земли) и m ветвей, имеет порядок $k=m-n+1$. Наиболее простой вид эта система приобретает при отсутствии задающих токов в узлах и задании активных элементов только источниками ЭДС. Здесь

Z_K — квадратная неособенная матрица контурных сопротивлений размерностью $k \times k$;

$\dot{I}_K$ — вектор-столбец из k контурных токов;

$\dot{E}_K$ — вектор-столбец из k контурных ЭДС, представляющих собой алгебраические суммы ЭДС ветвей, входящих в каждый независимый контур. Сложность и неоднозначность процедуры выделения независимых контуров и, как следствие этого, сложность формирования контурных уравнений для разветвленных электрических цепей с большим числом элементов (особенно при изменении конфигурации сети в процессе расчета), существенная зависимость свойств этих уравнений от выбранной системы

независимых контуров, увеличение количества ветвей схемы замещения при задании активных элементов источниками ЭДС, трудности учета трансформаторов с неноминальными коэффициентами трансформации – все это привело к тому, что контурные уравнения для расчетов установившихся режимов электрических систем в настоящее время, практически, не используются.

Вторая группа – нелинейные уравнения, связывающие мощности, напряжения и токи в каждом узле схемы замещения ЭС. Приняв за положительное направление мощности к узлу, выражение мощности источника или потребителя в узле i трехфазной цепи имеет следующий вид:

$$\dot{S}_i = P_i + jQ_i = \sqrt{3}\dot{U}_i\hat{J}_i$$

В матричном виде для n независимых узлов схемы замещения ЭС вектор-столбец узловых мощностей определяется следующим образом:

$$\dot{S} = \sqrt{3}\dot{U}_\delta\hat{J} \quad (6.3)$$

где $\dot{U}_\delta$ – диагональная матрица порядка n линейных напряжений узлов схемы замещения ЭС.

Объединив (6.1) и (6.3) можно получить систему n уравнений с комплексными коэффициентами и переменными, описывающую установившийся режим энергосистемы. Подстановка задающих токов из (6.3) в (6.1) приводит к системе узловых уравнений в форме баланса токов, матричная запись которой имеет вид

$$Y\dot{U} = \hat{U}_\delta^{-1}\hat{S}. \quad (6.4)$$

При подстановке задающих токов из (6.1) в (6.3) получаем систему узловых уравнений в форме баланса мощностей, матричная запись которой имеет вид

$$\hat{U}_\delta Y\dot{U} = \hat{S}, \quad (6.5)$$

где $\hat{S} = (\hat{S}_i)$, $i = 1, \dots, n$ – вектор-столбец комплексно-сопряженных значений мощностей источников или потребителей, подсоединенных к узлам

схемы замещения ЭС;

$\hat{U}_o = \text{diag}(\hat{U}_i), \quad i = 1, \dots, n$ – диагональная матрица комплексно-сопряженных значений напряжений в узлах схемы замещения;

$\dot{U} = (\dot{U}_i), \quad i = 1, \dots, n$ – вектор-столбец напряжений в узлах схемы;

$Y = (Y_{ij}), \quad i, j = 1, \dots, n$ – квадратная матрица узловых проводимостей.

Матричные узловые уравнения в форме баланса токов (6.4) или в форме баланса мощностей (6.5) – это системы нелинейных уравнений, которые характеризуются как простотой формирования, так и большими возможностями с точки зрения эффективной организации процесса их решения. Поэтому именно эти уравнения реализованы в современных программно-вычислительных комплексах для расчета установившихся режимов сложных энергосистем.

Нелинейные уравнения (6.4) и (6.5) содержат в общем случае n комплексных уравнений с n комплексными неизвестными напряжениями в узлах. Рассмотрим матричное узловое уравнение в форме баланса токов (6.4) в развернутой записи, задав мощности во всех узлах схемы замещения ЭС, т.е. считая генераторные узлы неопорными:

$$\begin{bmatrix} Y_{11} & \dots & Y_{1i} & \dots & Y_{1n} \\ \dots & \dots & \dots & \dots & \dots \\ Y_{i1} & \dots & Y_{ii} & \dots & Y_{in} \\ \dots & \dots & \dots & \dots & \dots \\ Y_{n1} & \dots & Y_{ni} & \dots & Y_{nn} \end{bmatrix} \times \begin{bmatrix} \dot{U}_1 \\ \vdots \\ \dot{U}_i \\ \vdots \\ \dot{U}_n \end{bmatrix} = \begin{bmatrix} \frac{1}{\hat{U}_1} & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \frac{1}{\hat{U}_i} & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \frac{1}{\hat{U}_n} \end{bmatrix} \times \begin{bmatrix} \hat{S}_1 \\ \vdots \\ \hat{S}_i \\ \vdots \\ \hat{S}_n \end{bmatrix} \quad (6.6)$$

На первый взгляд кажется, что система уравнений (6.6) из n уравнений с n неизвестными может дать решение для всех искомых комплексных напряжений. Однако это не совсем так по следующим причинам:

1. Из (6.6) видно, что если система комплексных величин напряжений $\dot{U}_i$, $i = 1, \dots, n$ является решением этих уравнений, то решением будет также система $\dot{U}_{*i} e^{j\phi}$, $i = 1, \dots, n$, все векторы которой повернуты относительно предыдущих на угол ϕ . Множитель $e^{j\phi}$ войдет после подстановки решений во все члены каждого из уравнений и может быть сокращен, так как $e^{-j\phi} e^{j\phi} = 1$:

$$\sum_{j=1}^n Y_{ij} \dot{U}_{*i} e^{j\phi} = \frac{\hat{S}_i}{\hat{U}_{*i} e^{-j\phi}}, \quad i = 1, \dots, n.$$

В итоге получается тождество, отвечающее исходной предпосылке, что $\dot{U}_{*i}$, $i = 1, \dots, n$ – решение (6.6). Сказанное означает, что координатную систему осей комплексной плоскости можно задать произвольно, т.е. система (6.6) имеет бесконечное множество решений. Физически это ясно, так как поток мощности в продольной ветви зависит не от абсолютных значений, а от разности фаз векторов напряжений узлов, ограничивающих ветвь (т.е. определяется относительным углом сдвига фаз). Математически это означает, что для того, чтобы избежать неоднозначности (множества решений), необходимо задать в одном из узлов, называемом базисным, фазовый угол $\delta_{баз}$.

2. Очевидно, что если во всех узлах энергосистемы, как было сказано выше, заданы мощности источников и нагрузок, то их алгебраическая сумма равна потерям мощности в сети. Это означает, что расчет режима будет вестись на заданные потери, которые, как известно, в свою очередь, зависят от искомых напряжений сети. Полученное в этих условиях решение может оказаться технически недопустимым (например, если величина потерь занижена, то решение будет получено при завышенных уровнях напряжений). При задании суммы мощностей нагрузок, превышающей сумму мощностей источников, может оказаться, что решения не существует.

Таким образом, если в уравнениях электрического режима сети (6.6) не задать ни одного напряжения, то можно получить решение, не имеющее

практического значения или не получить его вовсе. Сказанное приводит к тому, что для решения практических задач расчета электрического режима сложной сети в одном из ее узлов, называемом базисным, необходимо задать помимо фазового угла $\delta_{баз}$ еще и модуль напряжения $U_{баз}$. Итак, базисный узел – это узел, в котором должно быть задано комплексное значение напряжения $\dot{U}_{баз}$ в виде его модуля и фазового угла, который обычно принимают равным нулю ($\delta_{баз} = 0$).

Задание комплексного значения напряжения в одном из узлов приводит к необходимости исключения из системы (6.6) одного из комплексных уравнений. Это уравнение отвечает так называемому балансирующему узлу, узлу, мощность в котором $\dot{S}_{бал}$ определяется из условия баланса мощностей в сети, включая потери в ней. Обычно, если это не противоречит техническому содержанию задачи, базисный и балансирующий узлы совмещают. Тогда матрица Y сохраняет свои особенности, позволяющие повысить эффективность методов решения систем нелинейных уравнений вида (6.4) или (6.5).

При расчете установившегося режима ЭС не любой узел может быть взят в качестве балансирующего. Основные условия, которым должен удовлетворять балансирующий узел, следующие:

- 1) источники питания, связанные с этим узлом, должны иметь достаточные резервы активной и реактивной мощности;
- 2) малая зависимость напряжения в этом узле от режима;
- 3) физическая аналогия, т.е. в качестве балансирующего узла необходимо выбирать наиболее мощную станцию, ведущую частоту в системе или шины эквивалентированной части системы более высокой ступени иерархии, которые могут взять на себя небаланс мощности, обусловленный тем, что величина потерь мощности в сети до начала расчета установившегося режима неизвестна.

В дальнейшем для простоты изложения будем считать, что базисным

по напряжению и балансирующим по мощности принимается один и тот же узел n , который для краткости будем называть балансирующим. Тогда систему уравнений (6.6) можно записать в блочно-матричном виде:

$$\begin{bmatrix} Y & \vdots & Y_{j\bar{b}az} \\ \text{---} & \text{---} & \text{---} \\ Y_{\bar{b}azj} & \vdots & Y_{\bar{b}az\bar{b}az} \end{bmatrix} \times \begin{bmatrix} \dot{U} \\ \text{---} \\ \dot{U}_{\bar{b}az} \end{bmatrix} = \begin{bmatrix} \hat{U}_\delta^{-1} \hat{S} \\ \text{---} \\ \frac{\hat{S}_{\bar{b}az}}{\hat{U}_{\bar{b}az}} \end{bmatrix}. \quad (6.7)$$

Исключив из (6.7) уравнение для балансирующего узла, получаем систему (6.8) из $(n-1)$ -го нелинейных узловых уравнений в форме баланса токов с $(n-1)$ неизвестными комплексными напряжениями в узлах:

$$Y\dot{U} = \hat{U}_\delta^{-1} \hat{S} - Y_{j\bar{b}az} \hat{U}_{\bar{b}az}, \quad (6.8)$$

где Y – квадратная матрица узловых проводимостей, размерность которой $(n-1) \times (n-1)$;

$\dot{U} = (\dot{U}_i)$, $i = 1, \dots, (n-1)$ – вектор-столбец напряжений в узлах схемы;

$\hat{U}_\delta^{-1} = \text{diag}(1/\hat{U}_i)$, $i = 1, \dots, (n-1)$ – обратная диагональная матрица комплексно-сопряженных значений напряжений в узлах схемы замещения;

$\hat{S} = (\hat{S}_i)$, $i = 1, \dots, (n-1)$ – вектор-столбец комплексно-сопряженных значений мощностей в узлах схемы замещения;

$Y_{j\bar{b}az} = (Y_{j\bar{b}az})$, $j = 1, \dots, (n-1)$ – столбец проводимостей ветвей, связанных с базисным;

$\dot{U}_{\bar{b}az}$ – комплексное напряжение в базисном узле.

Записав систему уравнений (6.5) в блочно-матричном виде, приняв аналогично предыдущему случаю базисный узел n в качестве балансирующего и исключив из системы уравнение, соответствующее этому узлу, получаем систему (6.9) из $(n-1)$ нелинейных узловых уравнений в форме баланса мощности с $(n-1)$ неизвестными комплексными значениями напряжений в узлах:

$$\hat{U}_\delta Y \dot{U} + \hat{U}_\delta Y_{j\delta az} \dot{U}_{\delta az} = \hat{S} \quad (6.9)$$

В результате решения систем нелинейных уравнений (6.8) или (6.9) определяются комплексные значения напряжений $\dot{U}_i$, $i = 1, \dots, (n-1)$ во всех узлах за исключением балансирующего, совмещенного с базисным, напряжение $\dot{U}_{\delta az}$ в котором задано. После этого, при известных значениях напряжений во всех узлах ЭС, мощность в балансирующем узле можно найти из уравнения баланса мощности в нем:

$$\dot{S}_{\delta al} = \dot{U}_{\delta al} \sum_{j=1}^n \hat{Y}_{\delta al j} \hat{U}_j. \quad (6.10)$$

Кроме того, можно найти значения токов в ветвях схемы замещения, потоки мощности в начале и конце каждой ветви, потери мощности в ней и суммарные потери мощности в ЭС $\Delta P_{эс}$, $\Delta Q_{эс}$, режим для которой рассчитан.

Узловые уравнения в форме баланса токов (6.8) и в форме баланса мощностей (6.9) как было сказано выше – системы нелинейных уравнений, которые могут быть решены только итерационными методами.

Различные итерационные методы отличаются областью сходимости и показателями вычислительной эффективности. Вычислительная эффективность того или иного итерационного метода (при условии сходящегося итерационного процесса) оценивается объемом вычислений на одной итерации и числом итераций, необходимых для получения решения с заданной точностью, т.е. скоростью сходимости итерационного процесса к решению.

Отсутствие абсолютно сходящихся методов, а также методов, превосходящих остальные по показателям вычислительной эффективности, привело к тому, что для решения уравнений установившегося режима в настоящее время используется несколько методов, каждый из которых имеет свои преимущества и недостатки. Кроме того, опыт расчета режимов показывает, что эффективность того или иного метода зависит от формы записи уравнений. Одни методы, такие как метод Зейделя, методы,

основанные на использовании метода Гаусса, обращения или факторизации матрицы коэффициентов системы линейных алгебраических уравнений, в которую обращается система (6.8) на каждой итерации, позволяют решать $(n-1)$ уравнение с комплексными коэффициентами, получая решение в виде $(n-1)$ -го комплексного значения напряжений $\dot{U}_i$ в узлах. Другие методы, такие как метод Ньютона, для своей реализации требуют преобразования $(n-1)$ -го уравнения с комплексными коэффициентами и переменными в $(2n-2)$ уравнения с вещественными коэффициентами и переменными, которые могут быть записаны либо в полярной, либо в прямоугольной системах координат.

Методы, используемые в современных программно-вычислительных комплексах для расчета установившихся режимов, могут быть разделены на две основные группы:

1) методы, предусматривающие на каждой итерации последовательное решение уравнений, относящихся к каждому узлу, друг за другом, так называемые поузловые методы;

2) методы, предусматривающие на каждой итерации одновременное решение системы уравнений сразу для всех узлов.

К первой группе методов относится метод Зейделя, который может быть использован только для решения узловых уравнений в форме баланса токов (6.8), поскольку в каждой итерации, когда в правой части уравнений системы вычислены задающие токи, процесс сводится к решению системы линейных алгебраических уравнений. Коэффициенты уравнений и переменные при этом могут записываться либо в комплексной форме, либо в прямоугольной системе координат.

Таким образом, метод Зейделя основан на решении в данной итерации последовательно для одного узла за другим уравнений следующего вида:

$$U_i^{(k)} = \frac{1}{Y_{ii}} \left(\frac{\hat{S}_i}{\dot{U}_i^{(k-1)}} - Y_{i\text{баз}} \dot{U}_{\text{баз}} - \sum_{j=1, j \in G_i}^{i-1} Y_{ij} \dot{U}_j^{(k)} - \sum_{j=i+1, j \in G_i}^n Y_{ij} \dot{U}_j^{(k-1)} \right), \quad (6.11)$$

используя значения напряжений, найденные из предшествующих уравнений системы на этой же итерации.

Здесь k – номер итерации;

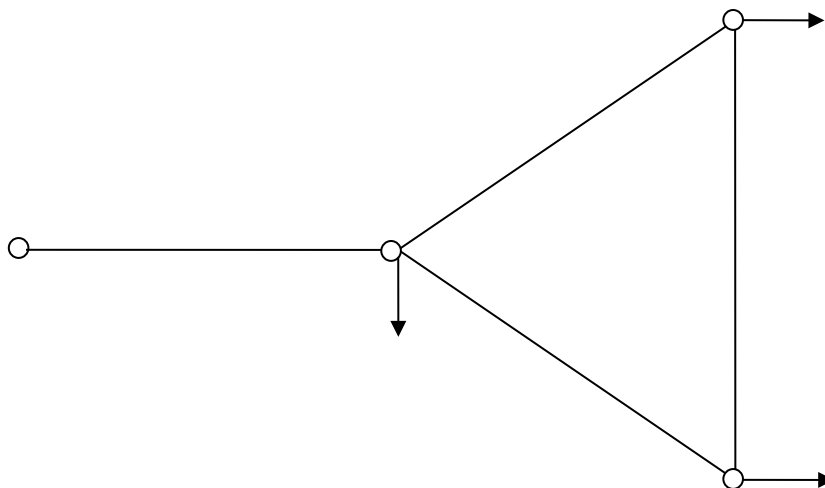
Y_{ij} – взаимная проводимость между узлами i и j ;

Y_{ii} – собственная узловая проводимость узла i ;

G_i – множество узлов индекса j , непосредственно связанных с i -м узлом.

В алгоритмическом отношении этот метод весьма прост, так как определение напряжения в каждом из узлов на итерации производится по единственной формуле (6.11). К достоинствам метода можно отнести и учитываемую естественным образом при реализации этого метода слабую заполненность матрицы узловых проводимостей Y . Широкий опыт расчетов режимов этим методом в прошлом показал, что итерационный процесс решения носит монотонный характер и сходится, несмотря на возможность ускорения сходимости, весьма медленно (количество итераций может изменяться от нескольких сотен до десятков тысяч).

Рассмотрим метод Зейделя на примере простой замкнутой схемы сети.



Для начала предположим, что все линии имеют только продольные параметры (R, X) и в сети отсутствуют трансформаторы. Согласно формуле (6.11), прежде всего, нам необходимо знать напряжения j -х узлов, то есть

узлов, связанных с рассматриваемым в данный момент. Вспомните структуру класса CNode. Каждый узел в переменной inc хранит ссылки на линии, с которыми он связан. Через эти ссылки мы можем получить доступ к противоположным узлам этой линии. Для этого, в класс ветви необходимо добавить следующую функцию:

```
39|     def get_end(self, ny):
40|         if(ny==self.ip): return self._iq;
41|         if(ny==self.iq): return self._ip;
42|         print("Ошибка в топологии сети")
43|         return None
```

Функция имеет один параметр – номер узла. Если передаваемый номер узла равен номеру узла начала, то функция возвращает узел конца.

Если передаваемый номер узла равен номеру узлу конца линии, то функция возвращает узел начала.

Если передаваемый номер узла не равен ни номеру узлу начала, ни номеру узлу конца, выводится соответствующее сообщение об ошибке.

Получив противоположный узел, можно получить доступ к любому параметру этого узла, в том числе и к его напряжению.

Далее нам необходимо вычислить собственную узловую проводимость узла i . Она вычисляется как сумма проводимостей линий, подходящих к узлу, взятая с обратным знаком. Добавим в класс узла функцию, вычисляющую собственную узловую проводимость:

```
15|     def Yii(self):
16|         Yii=complex()
17|         for l in self._inc:
18|             Z=complex(l.R,l.X)
19|             Yii-=1/Z
20|         return Yii
21|
```

Далее нам необходима функция, вычисляющая проводимость каждой ветви:

```
35|     def Y(self):
36|         Z=complex(self.R,self.X)
37|         return 1/Z
--|
```

С учетом введенных дополнительных функций, вычисление 3-х последних слагаемых в скобках формулы (6.11) для узла n будет выглядеть следующим образом:

```
118 |         sum=complex()
119 |         for l in n._inc:
120 |             sum+=l.Y()*l.get_end(n.ny).V
```

После этого, вычислить напряжение в i-м узле не составит сложности:

```
122 |         S=complex(n.P,n.Q)
123 |         n.V=1/n.Yii()*(S.conjugate()/n.V.conjugate()-sum)
```

Функция conjugate вычисляет сопряженное комплексное число.

С учетом всего вышесказанного, расчет режима методом Зейделя будет выглядеть следующим образом:

```
115 |     def zeidel(self,maxIters):           # Расчет режима методом Зейделя
116 |
117 |         for n in self.nodes:             # Обходим в цикле все узлы
118 |             n.V=complex(n.unom,0)        # Задаем начальные приближения
119 |
120 |         for it in range(maxIters):       # Итерационный расчет методом Зейделя
121 |
122 |             for n in self.nodes:
123 |                 sum=complex()
124 |                 for l in n._inc:
125 |                     sum+=l.Y()*l.get_end(n.ny).V
126 |
127 |             if n.type==True:
128 |                 S=complex(n.P,n.Q)
129 |                 n.V=1/n.Yii()*(S.conjugate()/n.V.conjugate()-sum)
130 |             else:
131 |                 S=n.V*(n.V*n.Yii()).conjugate()
132 |                 n.P=S.real
133 |                 n.Q=S.imag
```

В строках 117-118 для каждого узла в качестве начальных приближений расчетного напряжения берется номинальное напряжение узла.

Далее идет итерационный расчет методом Зейделя. Количество итераций обычно от 20 до 100, в зависимости от сложности сети.

На каждой расчетной итерации выполняется обход всех узлов сети. Если текущий узел является базисным (строки 130-133), вычисляется мощность его генерации. Если текущий узел является нагрузочным (строки 127-129), выполняется уточнение напряжения в этом узле согласно формуле (6.11).

Приведенный пример не учитывает поперечные проводимости ветвей и трансформаторные ветви. Предположим, что мы используем П-образную схему замещения линий и Г-образную схему замещения трансформаторов. В этом случае необходимо немного доработать уравнения и функции вычисления собственной узловой проводимости узлов и проводимостей ветвей.

```

15     def Yii(self):
16         Yii=complex()
17         for l in self._inc:
18             Yl=complex(l.G,-l.B)/1000000
19             if l.KTR==0:         Yii-=Yl/2
20             elif l.ip==self.ny: Yii-=Yl
21             Z=complex(l.R,l.X)
22             Y=1/Z
23             if l.KTR==0 or l.ip==self.ny:   Yii-=Y
24             else:                           Yii-=Y/(l.KTR*l.KTR)
25         return Yii|
26
40     def Y(self):
41         Y=1/complex(self.R,self.X)|
42         if self.KTR==0:     return Y
43         else:               return Y/self.KTR
44

```

Для ветвей типа «ЛЭП» проводимость делится пополам (так как используется П-образная схема замещения), для ветвей типа «трансформатор» проводимость целиком прибавляется к узлу начала.

В остальном функция расчета режима методом Зейделя не меняется. Расчет режима, выполняющий методом Зейделя 100 итераций будет выглядеть следующим образом:

```

133 # Начало программы
134 s=SCHEME()
135 s.load_scheme("E:\\Схема.csv");
136 s.assemble()
137 s.display()
138 s.zeidel(100)
139 s.display()
140 s.save_scheme("E:\\Результат.txt")

```

Общий объем программы составил около 140 строк.

Задание

1. Разработайте программу, реализующую расчет режима методом Зейделя.
2. Протестируйте программу на примере небольшой схемы сети. Сравните полученные результаты с программой RastrWin.

Контрольные вопросы

1. Какими группами уравнений может быть описан установившийся режим электрической системы?
2. Какие методы расчета режимов ЭЭС вам известны?
3. В чем состоит метод Зейделя?

Практическая работа №7

Тема: «Метод в два этапа»

Цель: Изучит алгоритм расчета режима методом «в два этапа»

Еще одним методом расчета режима, успешно применяющимся при расчете на ЭВМ является, так называемый, метод «в два этапа». Метод может применяться только для разомкнутых сетей и суть его состоит в многократном повторении двух расчетных операций:

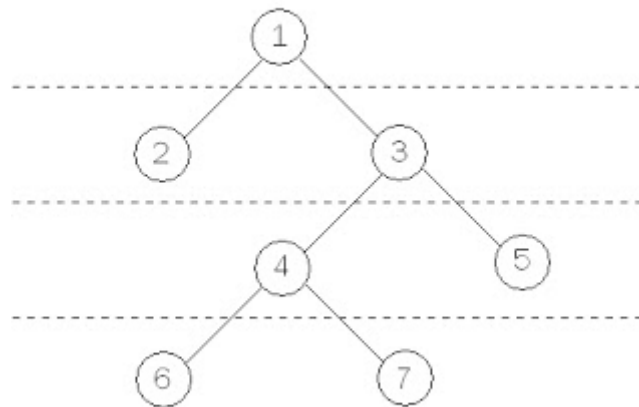
1-я операция заключается в определении потоков и потерь мощности в элементах сети, последовательно перемещаясь по этим элементам в схеме замещения от концов сети (от нагрузок) к ее началу (электростанции).

2-я операция состоит в определении рабочих напряжений в узлах и падений напряжения в элементах сети, последовательно перемещаясь по этим элементам в схеме замещения от начала сети (от заданного базисного напряжения на шинах станции) к концам сети (к нагрузкам).

В качестве начальных приближений задаются номинальные напряжения узлов сети. Критерием остановки, как правило, является достижение максимальной поправки по напряжению во всех узлах сети на втором этапе алгоритма некоторого заранее заданного значения. Кроме того, итерационный процесс может быть остановлен после выполнения заданного количества итераций.

С вычислительной точки зрения, алгоритм довольно простой и базируется на двух основных формулах: первом законе Кирхгофа в мощностях (либо в токах) и законе Ома. Основная сложность, которую приходится решать, при реализации данного метода на ЭВМ, является определение нужной последовательности обхода узлов и ветвей сети. Если нарисовать схему сети, любой человек, интуитивно и безошибочно легко определит порядок обхода узлов и ветвей сети, необходимый для этого алгоритма. Однако у компьютера нет «глаз» и уж тем более интуиции. Все

что у него есть – это таблица улов и таблица ветвей. И программисту необходимо четко задать последовательность действий, как, имея эти таблицы, определить прямой (для второго этапа) и обратный (для первого этапа) порядок обхода узлов и ветвей. Данный алгоритм относится к так называемым «алгоритма на графах» и называется «алгоритм поиска в ширину». Для того чтобы объяснить в чем смысл этого алгоритма, давайте представим схему сети в виде ветвящегося дерева, изображенного на рисунке ниже.



Любую разомкнутую схему сети можно представить в виде такого дерева. Вершиной этого дерева является центр питания (базисный узел). Чтобы понять суть алгоритма поиска в ширину, давайте проведем имитацию «горения» графа. Представьте себе, что мы поджигаем самую первую вершину, дальше огонь перекидывается на смежные с ней вершины, с них на смежные с ними и т.д. Причем огонь идет только в одном направлении, от вершины – к концам. Он не может вернуться обратно, так как предыдущие ветви и узлы уже сгорели. Имитируя такой процесс горения, можно определить в каком порядке будут «гореть» узлы и ветви. Запомнив этот порядок, мы получим последовательность прямого обхода графа для второго этапа метода «в два этапа». А если эти порядок перевернуть, то получим

последовательность обратного обхода графа для первого этапа метода «в два этапа».

Давайте подумаем, что нам необходимо, чтобы эмитировать такого «горение» графа. Прежде всего, нам необходима переменная, в которую мы будем сохранять порядок «сгорания» ветвей графа. В качестве такой переменной лучше всего использовать список. Добавим эту переменную в класс схемы сети, назовем ее order (порядок):

```
50 class SCHEME:
51     def __init__(self):
52         self.nodes=[]
53         self.lines=[]
54         self.order=[]
```

Класс Схема сети
Инициализация класса
Таблица узлов
Таблица ветвей
Порядок обхода ветвей для метода "в два этапа"

Далее, для того, чтобы отмечать на схеме, какие узлы и ветви уже «сгорели», давайте введем для узлов и ветвей дополнительную переменную, которую назовем _marker.

```
4 class NODE:
5     def __init__(self, csv={}):
6         self.ny = int(csv.get('N',0))
7         self.name = csv.get('Название','')
8         self.type = bool(csv.get('Тип',True))
9         self.unom = float(csv.get('Уном',0.0))
0         self.P = float(csv.get('P',0.0))
1         self.Q = float(csv.get('Q',0.0))
2         self.V = 0+0j
3         self._inc = []
4         self._marker= False
```

Класс "Узел"
Инициализация переменных
Номер узла
Название узла
Тип узла False - Базисный, True - нагрузочный
Номинальное напряжение
Активная мощность нагрузки
Реактивная мощность нагрузки
Расчетное напряжение
Список связанных ветвей
#

```
28 class LINE:
29     def __init__(self, csv={}):
30         self.ip = int(csv.get('N нач.',0))
31         self.iq = int(csv.get('N кон.',0))
32         self.R = float(csv.get('R',0.0))
33         self.X = float(csv.get('X',0.0))
34         self.G = float(csv.get('G',0.0))
35         self.B = float(csv.get('B',0.0))
36         self.KTR = float(csv.get('KTR',0.0))
37         self._ip = None
38         self._iq = None
39         self._marker= False
```

Класс "Ветвь"
Инициализация переменных
Номер узла начала
Номер узла конца
Активное сопротивление линии
Реактивное сопротивление линии
Активная проводимость линии
Емкостная проводимость линии
Коэффициент трансформации
Узел начала
Узел конца
#

Тип этой переменной будет bool, так как она может принимать только два значения: false – еще не «сгорела» и «true» – уже «сгорела».

Давайте начнем реализовывать метод «в два этапа». Для этого в классе схемы сети создадим метод «two_steps», который будет иметь два параметра: максимальное количество итераций и максимальная поправка по напряжению на очередной итерации, по достижению которой расчет можно будет считать успешно завершенным.

```

137 | def two_steps(self,maxIters,maxdU):      # Расчет режима методом "В два этапа"
138 |     self.order.clear()
139 |
140 |     for l in self.lines:
141 |         l._marker=False
142 |
143 |     for n in self.nodes:
144 |         n._marker=False
145 |         n.V=complex(n.unom,0)
146 |

```

Вначале мы выполняем очистку списка order. Очистка выполняется на тот случай, если расчет режима выполняется повторно и в нем содержится информация, сохранившаяся от предыдущего расчета.

Далее выполняется «инициализация» маркеров – мы указываем, что все ветви и узлы сети еще не «сгорели». Это также выполняется на случай, если расчет режима выполняется повторно.

Кроме того, в строке 145 выполняется задание начального приближения расчетного напряжения, равного номинальному напряжению узла.

При реализации метода «в два этапа», нам также понадобится список, который будет хранить ссылки на узлы, которые должны будут «сгореть» в ближайшее время. Назовем его stack. Первоначально в него помещается узел центра питания (node.type==0).

```

146 |         |
147 |         stack=[]
148 |         for n in self.nodes:
149 |             if n.type==False:
150 |                 stack.append(n)
151 |             break

```

Далее, пока этот массив не станет пустым, выполняется следующий цикл:

```

155         while len(stack)>0:
156             n=stack.pop()
157             if n._marker==True:
158                 print("Сеть содержит замкнуты контура. Расчет методом 'в два этапа' невозможен")
159                 return False
160
161             n._marker=True
162
163             for l in n._inc:
164                 if l._marker==True:continue
165                 l._marker=True
166
167                 if l.iq==n.ny:
168                     l.ip, l.iq = l.iq, l.ip
169                     l._ip,l._iq = l._iq,l._ip
170
171             self.order.append(l)
172             stack.append(l._iq)

```

В строке 156 мы получаем очередной узел, который должен «сгореть» и одновременно удаляем его из массива, чтобы затем не обрабатывать его повторно.

В строке 157 мы проверяем, «сгорел» это узел или еще нет. Если узел «сгорел», это означает, что мы каким-то образом вернулись туда, где уже были. Это возможно только в одном случае: сеть содержит «циклы», то есть замкнутые контура. Алгоритм «в два этапа» не умеет считать такие сети, поэтому выводим соответствующее сообщение и завершаем выполнение функции.

Далее мы отмечаем текущий узел как «сгоревший» (строка 161).

В строках 163-172 мы в цикле обходим все ветви, связанные с текущим узлом. Если какая-либо из этих ветвей уже «сгорела», то просто пропускаем ее (строка 164).

В противном случае, запоминаем эту ветвь в переменную order (строка 171), отмечаем ее как «сгоревшую» (165) и добавляем следующий узел (конечный узел ветви), который должен будет «сгореть» в массив (172).

В строках 167-169 выполняется «разворот» некоторых линий, если это необходимо. Под «разворотом» в данном случае понимается перемена местами узла начала и конца. Допустим, пользователь задал в файле 3 линии: 1-2, 3-2, 3-4. Очевидно, что порядок обхода будет следующий: ветвь 1-2, затем ветвь 2-3, затем ветвь 3-4. Но в списке нет ветви 2-3, есть только ветвь 3-2. Физически это одна и та же ветвь, поэтому для упрощения алгоритма проще ее просто перенумеровать.

По завершении цикла переменная `order` будет содержать нужный нам порядок обхода ветвей сети. Его можно использовать для реализации первого и второго этапа алгоритма. Сам итерационный процесс будет выглядеть следующим образом:

```

173         for it in range(maxIters):
174             self.backward()
175             dU=self.forward()
176             print("Итерация №",it,"\t|поправка по напряжению: ",dU)
177             if dU<maxdU:
178                 for n in self.nodes:
179                     if n.type==False:
180                         n.P=-n._S.real
181                         n.Q=-n._S.imag
182             print("Расчет режима завершен")
183             return True
184         return False

```

`backward()` – это функция, реализующая первый этап алгоритма (обратный обход);

`forward()` – это функция, реализующая второй этап алгоритма (прямой обход). Она возвращает максимальную поправку по напряжению, произведенную в узлах сети. Если эта поправка меньше заданного значения, расчет считается завершенным. В строках 178-181 выполняется расчет мощности балансирующего узла сети. Для этого расчета используется переменная `_S`. Ее необходимо добавить в класс узла сети:

```

4 class NODE:
5     def __init__(self, csv={}):
6         self.ny = int(csv.get('№',0))
7         self.name = csv.get('Название','')
8         self.type = bool(csv.get('Тип',True))
9         self.unom = float(csv.get('Уном',0.0))
10        self.P = float(csv.get('P',0.0))
11        self.Q = float(csv.get('Q',0.0))
12        self.V = 0+0j
13        self._inc = []
14        self._marker= False
15        self._S = complex()

```

Класс "Узел"
 # Инициализация переменных
 # Номер узла
 # Название узла
 # Тип узла False - Базисный, True - нагрузочный
 # Номинальное напряжение
 # Активная мощность нагрузки
 # Реактивная мощность нагрузки
 # Расчетное напряжение
 # Список связанных ветвей
 #
 #

Переменная `_S` будет хранить сумму мощностей, исходящих из текущего узла сети. Это те слагаемые в уравнении первого закона Кирхгофа, которые имеет отрицательный знак. Для конечных узлов в этой переменной будет храниться только мощности нагрузок, увеличенные на потери холостого хода в линиях. Для всех остальных узлов к этому значению будут добавлены потоки мощности отходящих ветвей.

Также для класса линии добавим переменную, которая будет хранить поток мощности, проходящий через нее (поток мощности в продольной части линии):

```

29 class LINE:
30     def __init__(self, csv={}):
31         self.ip = int(csv.get('№ нач.', 0))
32         self.iq = int(csv.get('№ кон.', 0))
33         self.R = float(csv.get('R', 0.0))
34         self.X = float(csv.get('X', 0.0))
35         self.G = float(csv.get('G', 0.0))
36         self.B = float(csv.get('B', 0.0))
37         self.KTR = float(csv.get('KTR', 0.0))
38         self._ip = None
39         self._iq = None
40         self._marker = False
41         self._S = complex()

```

Класс "Ветвь"
 # Инициализация переменных
 # Номер узла начала
 # Номер узла конца
 # Активное сопротивление линии
 # Реактивное сопротивление линии
 # Активная проводимость линии
 # Емкостная проводимость линии
 # Коэффициент трансформации
 # Узел начала
 # Узел конца
 #
 # Поток мощности по линии (в метода в два этапа)

С учетом всего сказанного, функция backward() будет иметь следующий вид:

```

187 def backward(self):
188     for n in self.nodes:
189         n._S = complex(n.P, n.Q)
190         for l in n._inc:
191             if l.KTR == 0:
192                 n._S += complex(l.G, l.B) / 2000000 * abs(n.V) * abs(n.V)
193             elif l.ip == n.ny:
194                 n._S += complex(l.G, l.B) / 1000000 * abs(n.V) * abs(n.V)
195
196     for l in reversed(self.order):
197         Z = complex(l.R, l.X)
198         if l.KTR != 0:
199             Z *= l.KTR
200         S = abs(l._iq._S)
201         U = abs(l._iq.V)
202         dS = Z * (S * S / (U * U))
203         l._S = l._iq._S + dS
204         l._ip._S += l._S

```

В функции используются известные формулы вычисления потерь мощности холостого хода (строки 191, 192) и нагрузочные потери в линии (строка 200). Для трансформаторов осуществляется операция приведения сопротивления с учетом коэффициента трансформации (строки 196-197).

Функция forward() также использует простейшую формулу вычисления падения напряжения:

```

204 def forward(self):
205     maxdU = 0
206     for l in self.order:
207         U2 = l._ip.V - complex(l.R, l.X) * l._S.conjugate() / l._ip.V.conjugate()
208
209         if l.KTR != 0:
210             U2 *= l.KTR
211
212         maxdU = max(abs(U2 - l._iq.V), maxdU)
213         l._iq.V = U2
214     return maxdU

```

Здесь переменная `maxdU` хранит максимальную поправку по напряжению на текущей итерации. Первоначально она задается 0. Затем для каждого узла вычисляется новое значение напряжения и разница с предыдущим значением напряжения (фактически, поправка напряжения в текущем узле). С учетом всех добавленных функций, расчет методом «в два этапа» будет выполняться следующим образом:

```
218 | s=SCHEME()  
219 | s.load_scheme("E:\\Схема.csv");  
220 | s.assemble()  
221 | s.display()  
222 | s.two_steps(20,0.00001)  
223 | s.display()  
224 | s.save_scheme("E:\\Результат.txt")
```

Задание

1. Разработайте программу, реализующую расчет режима методом «в два этапа».
2. Протестируйте программу на примере небольшой схемы сети. Сравните полученные результаты с программой RastrWin.

Контрольные вопросы

1. В чем суть метода «в два этапа»?
2. Опишите алгоритм поиска «в ширину».

Практическая работа №8

Тема: библиотека NumPy

Цель: Получить общее представление о модулях и пакетах, научиться подключать и использовать дополнительные модули и пакеты библиотеки NumPy для выполнения сложных математических расчетов

Модули и пакеты значительно упрощают работу программиста. Функции, объекты и константы, которыми приходится часто пользоваться можно упаковать в модуль, и, в дальнейшем, загружать его в свои программы при необходимости.

Под модулем в Python понимается файл с расширением .py. Модули предназначены для того, чтобы в них хранить часто используемые функции, классы, константы и т.п. Можно условно разделить модули и программы: программы предназначены для непосредственного запуска, а модули для импортирования их в другие программы. Стоит заметить, что модули могут быть написаны не только на языке Python, но и на других языках (например C).

Самый простой способ импортировать модуль в Python это воспользоваться конструкцией:

```
import имя_модуля
```

Импорт и использование модуля math, который содержит математические функции, будет выглядеть вот так.

```
>>> import math
>>> math.factorial(5)
120
```

За один раз можно импортировать сразу несколько модулей, для этого их нужно перечислить через запятую после слова import:

```
import имя_модуля1, имя_модуля2
```

```
>>> import math, datetime
>>> math.cos(math.pi/4)
```

```
0.707106781186547
>>> datetime.date(2017, 3, 21)
datetime.date(2017, 3, 21)
```

Если вы хотите задать псевдоним для модуля в вашей программе, можно воспользоваться вот таким синтаксисом:

```
import имя_модуля as новое_имя
```

```
>>> import math as m
>>> m.sin(m.pi/3)
0.866025403784438
```

Используя любой из вышеперечисленных подходов, при вызове функции из импортированного модуля, вам всегда придется указывать имя модуля (или псевдоним). Для того, чтобы этого избежать делайте импорт через конструкцию `from ... import...`

```
from имя_модуля import имя_объекта
```

```
>>> from math import cos
>>> cos(3.14)
0.999998731727539
```

При этом импортируется только конкретный объект (в нашем примере: функция `cos`), остальные функции недоступны, даже если при их вызове указать имя модуля.

Для импортирования нескольких функций из модуля, можно перечислить их имена через запятую.

```
from имя_модуля import имя_объекта1, имя_объекта2
```

```
>>> from math import cos, sin, pi
>>> cos(pi/3)
0.5000000000000000
>>> sin(pi/3)
0.866025403784438
```

Импортируемому объекту можно задать псевдоним.

```
from имя_модуля import имя_объекта as псевдоним_объекта
```

```
>>> from math import factorial as f
```

```
>>> f(4)
24
```

Если необходимо импортировать все функции, классы и т.п. из модуля, то воспользуйтесь следующей формой оператора `from ... import ...`

```
from имя_модуля import *
```

```
>>> from math import *
>>> cos(pi/2)
6.123233995736766e-17
>>> sin(pi/4)
0.707106781186547
>>> factorial(6)
720
```

Пакеты в Python

Пакет в Python – это каталог, включающий в себя другие каталоги и модули, но при этом дополнительно содержащий файл `__init__.py`. Пакеты используются для формирования пространства имен, что позволяет работать с модулями через указание уровня вложенности (через точку).

Для импортирования пакетов используется тот же синтаксис, что и для работы с модулями.

Рассмотрим следующую структуру пакета:

```
fincalc
|-- __init__.py
|-- simper.py
|-- compper.py
|-- annuity.py
```

Рисунок 8.1 – Структура типового пакета

Пакет `fincalc` содержит в себе модули для работы с простыми процентами (`simper.py`), сложными процентами (`compper.py`) и аннуитетами (`annuity.py`).

Для использования функции из модуля работы с простыми процентами, можно использовать один из следующих вариантов:

```
import fincalc.simper
```

```
fv = fincalc.simper.fv(pv, i, n)
import fincalc.simper as sp
fv = sp.fv(pv, i, n)
from fincalc import simper
fv = simper.fv(pv, i, n)
```

Модуль NumPy для выполнения сложных математических расчетов

NumPy – это открытая бесплатная Python-библиотека для работы с многомерными массивами. NumPy чаще всего используют в анализе данных и обучении нейронных сетей – в каждой из этих областей нужно проводить много вычислений с такими матрицами.

NumPy не просто работает с многомерными массивами, но и делает это быстро. Вообще, интерпретируемые языки производят вычисления медленнее компилируемых, а Python как раз язык интерпретируемый. NumPy же сделана так, чтобы эффективно работать с наборами чисел любого размера в Python. Это достигается благодаря тому, что библиотека частично написана на Python, а частично на C и C++ – в тех местах, которые требуют скорости. Кроме того, код NumPy оптимизирован под большинство современных процессоров.

NumPy предоставляет общие математические и числовые операции в виде пре-скомпилированных, быстрых функций. Они объединяются в высокоуровневые пакеты и обеспечивают функционал, который можно сравнить с функционалом MatLab. NumPy (Numeric Python) предоставляет базовые методы для манипуляции с большими массивами и матрицами. SciPy (Scientific Python) расширяет функционал numpy огромной коллекцией полезных алгоритмов, таких как минимизация, преобразование Фурье, регрессия, и другие прикладные математические техники. Обычно NumPy входит в стандартный дистрибутив установки Python. Для его использования необходимо в коде программы подключить пакет через стандартную команду `import`:

```
import numpy
```

Если команда вызвала ошибку, необходимо установить NumPy самостоятельно. Для этого в командной строке Windows от имени администратора необходимо выполнить следующую команду:

```
pip install NumPy
```

Есть несколько путей импорта. Стандартный метод это — использовать простое выражение `import`, как было показано выше. Тем не менее, для большого количества вызовов функций `numpy`, становится утомительно писать `numpy.X` снова и снова. Вместо этого обычно используют псевдоним:

```
import numpy as np
```

Главной особенностью `numpy` является объект массив (`array`). Массивы схожи с уже изученными ранее списками в `python`, исключая тот факт, что элементы массива должны иметь одинаковый тип данных (`float` или `int`). Это позволяет проводить числовые операции с большим объемом информации в разы быстрее и, главное, намного эффективнее чем со списками.

Создание массивов

Есть несколько способов создания массивов:

1. С помощью команды `empty`. Синтаксис команды следующий:

```
numpy.empty(shape, dtype=float, order='C', *, like=None)
```

`shape` — это размерность нашего массива. Если массив одномерный, указывается одно число. Если массив многомерный (матрица), в круглых скобках нужно указать количество строк и количество столбцов через запятую.

Остальные параметры являются не обязательными. `dtype` указывает, какого типа данные будут храниться в массиве (по умолчанию вещественные), `order` и `like` задают способ хранения в памяти, в наших задачах он не имеет значения.

Например:

```
import numpy as np
a=np.empty(10)
print(a)
```

Создает массив из 10 чисел. В результате выполнения программы получим:

```
[6.23042070e-307 3.56043053e-307 1.60219306e-306 2.44763557e-307
 1.69119330e-306 1.33514617e-307 3.56043053e-307 1.37961641e-306
 9.45697982e-308 1.06101441e-312]
```

Создать матрицу размером 5x3:

```
import numpy as np
a=np.empty((5,3))
print(a)
```

В результате выполнения программы получим:

```
[[2.23199246e-305 0.00000000e+000 0.00000000e+000]
 [0.00000000e+000 0.00000000e+000 0.00000000e+000]
 [0.00000000e+000 0.00000000e+000 0.00000000e+000]
 [0.00000000e+000 0.00000000e+000 0.00000000e+000]
 [0.00000000e+000 0.00000000e+000 1.76125651e-312]]
```

Обратите внимание, не все элементы созданных массивов равны нулю. Строго говоря, метод `empty` создает неинициализированный массив, то есть в нем могут храниться абсолютно любые, заранее неизвестные числа.

2. Если мы хотим создать массив с нулевыми значениями, существует функция `zeros`. Она имеет точно такие же параметры, как и функция `empty`, но гарантирует, что все элементы будут равны нулю. Пример:

```
import numpy as np
a=np.zeros((5,3))
print(a)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

3. Если мы хотим инициализировать массив не нулями, а каким-то другими числами, существует функция `full`:

```
numpy.full(shape, fill_value, dtype=None, order='C', *, like=None)
```

Пример:

```
import numpy as np
a=np.full((5,3),5)
print(a)
```

```
[[5 5 5]
 [5 5 5]
 [5 5 5]
 [5 5 5]
 [5 5 5]]
```

4. Массивы можно создавать также из уже готовых списков. Для этого существует функция array. Пример:

```
import numpy as np
l=[[1,2,3],[4,5,6]]
a=np.array(l)
print(a)
```

```
[[1 2 3]
 [4 5 6]]
```

Справедлива и обратная операция. Массивы можно преобразовать обратно в списки:

```
import numpy as np
l=[[1,2,3],[4,5,6]]
a=np.array(l)
b=a.tolist()
print(a)
print(b)
```

```
[[1 2 3]
 [4 5 6]]
[[1, 2, 3], [4, 5, 6]]
```

Массивы можно создавать из списка:

```
a = np.array([1, 4, 5, 8], float)
```

Обратите внимание, что в отличие от списка, при создании массива нужно указать, какой тип данных он будет хранить.

Обращение к элементам массива

Обращаться к элементам массива можно также, как к элементам списка:

```
import numpy as np
l=[[1,2,3],[4,5,6]]
a=np.array(l)
print(a[0][0])
print(a[0][:2])
print(a[0,0])

1
[1 2]
1
```

Обратите внимание на последнюю строчку кода. В отличие от списков, массивы позволяют обращаться к элементам массива, указывая индекс через запятую в скобках.

Перебор элементов массива

Обход элементов списка осуществляется аналогично спискам:

```
>>> a = np.array([1, 4, 5], int)
>>> for x in a:
...     print x
1
4
5
```

Для многомерных массивов итерация будет проводиться по первой оси, так, что каждый проход цикла будет возвращать «строку» массива:

```
>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> for x in a:
...     print x
[ 1.  2.]
[ 3.  4.]
[ 5.  6.]
```

Множественное присваивание также доступно при итерации:

```
>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> for (x, y) in a:
...     print x * y
2.0
12.0
30.0
```

Базовые операции над массивами

1. Математические операции над массивами. Над массивами можно выполнять стандартные математические операции: сложение, умножение и так далее. Но должен соблюдаться один важный принцип: элемент–элемент. Это означает, что массивы должны быть одинакового размера во время сложения, вычитания и тому подобных операций:

```
>>> a = np.array([1,2,3], float)
>>> b = np.array([5,2,6], float)
>>> a + b
array([6., 4., 9.])
>>> a - b
array([-4., 0., -3.])
>>> a * b
array([5., 4., 18.])
>>> b / a
array([5., 1., 2.])
>>> a % b
array([1., 0., 3.])
>>> b**a
array([5., 4., 216.])
```

Для двумерных массивов, умножение остается поэлементным и не соответствует умножению матриц. Для этого существуют специальные функции, которые будут рассмотрены позже.

```
>>> a = np.array([[1,2], [3,4]], float)
>>> b = np.array([[2,0], [1,3]], float)
>>> a * b
array([[2., 0.], [3., 12.]])
```

При несоответствии в размере возникают ошибки:

```
>>> a = np.array([1,2,3], float)
>>> b = np.array([4,5], float)
>>> a + b
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: operands could not be broadcast together with shapes (3,) (2,)
```

Однако, если размерность массивов не совпадает, они будут преобразованы для выполнения математических операций. Это зачастую означает, что меньший массив будет использован несколько раз для завершения операций. Рассмотрим такой пример:

```
>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> b = np.array([-1, 3], float)
>>> a
array([[ 1.,  2.],
       [ 3.,  4.],
       [ 5.,  6.]])
>>> b
array([-1.,  3.])
>>> a + b
array([[ 0.,  5.],
       [ 2.,  7.],
       [ 4.,  9.]])
```

Тут, одномерный массив `b` был преобразован в двухмерный, который соответствует размеру массива `a`. По существу, `b` был повторен несколько раз, для каждой «строки» `a`. Иначе его можно представить так:

```
array([[ -1.,  3.],
       [ -1.,  3.],
       [ -1.,  3.]])
```

2. Статистические функции. Для получения каких-либо свойств массивов существует много функций. Элементы могут быть суммированы или перемножены:

```
>>> a = np.array([2, 4, 3], float)
>>> a.sum()
9.0
>>> a.prod()
24.0
```

Также существуют функции вычисления среднего арифметического (mean), дисперсии (var) и среднего квадратического отклонения (std):

```
>>> a = np.array([2, 1, 9], float)
>>> a.mean()
4.0
>>>
a.var()
12.666666666666666
>>> a.std()
3.5590260840104371
```

Функции вычисления максимального и минимального значения:

```
>>> a = np.array([2, 1, 9], float)
>>> a.min()
1.0
>>> a.max()
9.0
```

И множество других базовых операций, который можно найти в документации к NumPy.

Векторная и матричная математика

NumPy обеспечивает много функций для работы с векторами и матрицами. Транспонировать матрицу (поменять местами столбцы и строки) можно с помощью функции transpose:

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6]])
print(a)
print(a.transpose())
```

Функция dot возвращает скалярное произведение векторов:

```
>>> a = np.array([1, 2, 3], float)
>>> b = np.array([0, 1, 1], float)
>>> np.dot(a, b)
5.0
```

Функция dot также может умножать матрицы:

```
>>> a = np.array([[0, 1], [2, 3]], float)
>>> b = np.array([2, 3], float)
>>> c = np.array([[1, 1], [4, 0]], float)
>>> a
array([[ 0.,  1.],
       [ 2.,  3.]])
>>> np.dot(b, a)
array([ 6., 11.])
>>> np.dot(a, b)
array([ 3., 13.])
>>> np.dot(a, c)
array([[ 4.,  0.],
       [14.,  2.]])
>>> np.dot(c, a)
array([[ 2.,  4.],
       [ 0.,  4.]])
```

NumPy также предоставляет набор встроенных функций и методов для работы с линейной алгеброй. Это всё можно найти в под-модуле linalg. Определитель матрицы ищется следующим образом:

```
>>> a = np.array([[4, 2, 0], [9, 3, 7], [1, 2, 1]], float)
>>> a
array([[ 4.,  2.,  0.],
       [ 9.,  3.,  7.],
       [ 1.,  2.,  1.]])
>>> np.linalg.det(a)
-48.
```

Для решения линейных систем уравнений существует функция linalg.solve:

```
import numpy as np
# Решить систему уравнений:
# 2*x1+x2=3
# x1+2*x2=4
A=np.array([[2,1],[1,2]])
B=([3,4])
X=np.linalg.solve(A,B)
print(X)
```

```
[0.66666667 1.66666667]
```

Для использования этой функции необходимо познакомиться с таким понятием как исключение. Дело в том, что система уравнений не всегда может иметь решение. В этом случае программа вызовет ошибку и надо правильно ее обработать, чтобы была возможность продолжать выполнение алгоритма дальше. Исключениями (exceptions) в языках программирования называют проблемы, возникающие в ходе выполнения программы, которые допускают возможность дальнейшей ее работы в рамках основного алгоритма. Типичным примером исключения является деление на ноль, невозможность считать данные из файла (устройства), отсутствие доступной памяти, доступ к закрытой области памяти и т.п. Для обработки таких ситуаций в языках программирования, как правило, предусматривается специальный механизм, который называется обработка исключений.

В языке Python обработка исключений осуществляется с помощью конструкции try/except. Рассмотрим ее на пример деления на 0. Если мы напишем:

```
b=0
a=1/b
```

то программа выдаст ошибку:

```
Traceback (most recent call last):
  File "F:\
line 379, in <module>
    a=1/b
ZeroDivisionError: division by zero
```

Чтобы избежать эту ошибку, лучше обернуть операцию деления в блок try/except:

```
b=0
try:
    a=1/0
except:
    print("Ошибка в программе")
```

Тогда программы выведет сообщение об ошибке и продолжит свое выполнение дальше. Аналогичным образом, вызов функции `linalg.solve` лучше помещать в блок `try/except`, чтобы была возможность определить, когда система не имеет решения.

Найти обратную матрицу позволяет функция `numpy.linalg.inv`:

```
import numpy as np
A=np.array([[2,1],[1,2]])
print(np.linalg.inv(A))

[[ 0.66666667 -0.33333333]
 [-0.33333333  0.66666667]]
```

В библиотеке существует также множество других функций, позволяющих производить разложение матриц, приведение к треугольному виду и др. Подробное описание всех возможностей библиотеки можно найти в официальной документации в глобальной сети «Интернет».

Задание

1. Разработайте программу, осуществляющую перемножение матриц. Матрицы должны храниться в файлах.
2. Разработайте программу для вычисления обратной матрицы. Матрица должна храниться в файле.
3. Разработайте программу для вычисления определителя матрицы. Матрица должна храниться в массиве.
4. Разработайте программу для решения системы линейных уравнений, которая хранится в файле.

Контрольные вопросы

1. Что такое массив? Чем он отличается от списков?
2. Какие способы создания массивов вы знаете?
3. Для чего нужна библиотека NumPy? Как ее использовать в ваших программах?

4. За счет чего достигается высокое быстродействие библиотеки NumPy?

5. Какие операции с массивами в Python вы знаете?

6. Как выполнить перемножение матриц?

7. Как решить систему линейных уравнений?

8. Как найти определитель матрицы?

9. Как найти обратную матрицу?

10. Как сложить матрицы?

11. Что такое исключения (exceptions)? Как их обрабатывать и для чего это нужно?

Практическая работа №9

Тема: «Метод Ньютона»

Цель: изучить алгоритм расчета режима методом Ньютона

Как было указано ранее, наиболее простым в алгоритмическом отношении методом расчета режима является метод Зейделя, так как определение напряжения в каждом из узлов на итерации производится по единственной формуле. Однако итерационный процесс при этом методе сходится весьма медленно (количество итераций может изменяться от нескольких сотен до десятков тысяч). Помимо того, этим методом не удастся получить решение в случае неоднородности элементов схемы замещения и в схемах с отрицательными реактивными сопротивлениями отдельных ветвей, которые образуются при наличии УПК. Поэтому в современных программно-вычислительных комплексах метод Зейделя как метод, мало чувствительный к начальным приближениям, используется только в качестве стартового алгоритма к последующему решению уравнений другими методами.

Для повышения вычислительной эффективности решения нелинейных узловых уравнений установившегося режима в общем случае задания исходных данных могут быть использованы методы, основанные на одновременном решении всех уравнений системы в каждой итерации. Это градиентные методы и метод Ньютона. Применение этих методов требует перехода от $(n-1)$ -го уравнений с комплексными коэффициентами и переменными к $(2n-2)$ -м уравнениям с вещественными коэффициентами и переменными. Это связано с тем, что применение этих методов требует дифференцирования уравнений по искомым переменным (как известно, производные по комплексным переменным не определены).

Переход от комплексных к вещественным параметрам осуществляется на основе записи комплексных чисел в прямоугольной или полярной системах координат. При записи уравнений в прямоугольной системе

координат напряжение любого узла представляется в форме $\dot{U}_i = U'_i + jU''_i$. При использовании полярной системы координат $\dot{U}_i = U_i(\cos \delta_i + j \sin \delta_i)$. От выбора системы координат, в которой записываются уравнения установившегося режима, зависят потребность в памяти ПК и, следовательно, объем задачи, которую можно решить при выделенном объеме памяти, а также время счета и характер сходимости итерационного процесса.

Форма уравнений и сложность вычислений по (6.9) в значительной степени зависит от используемой системы координат, то есть от способа представления комплексных чисел. Как известно, существует 3 формы представления комплексных чисел:

- прямоугольная;
- тригонометрическая (полярная);
- показательная (экспоненциальная).

Наиболее короткие уравнения получаются при использовании следующей системы координат:

- прямоугольные координаты для мощности: $S = P + jQ$;
- полярные координаты для напряжений: $\dot{U}_i = U_i(\cos \delta_i + j \sin \delta_i)$;
- прямоугольные координаты для собственной узловой проводимости:

$$Y_{ii} = g_{ii} - jb_{ii};$$

- полярные координаты для проводимостей ветвей:

$$Y_{ij} = y_{ij}(\cos \alpha_{ij} - j \sin \alpha_{ij}).$$

В этом случае, уравнения (6.9) примут вид:

$$\begin{cases} U_i^2 g_{ii} + U_i \sum_{j \neq i} U_j y_{ij} \cos(\delta_i - \delta_j + \alpha_{ij}) = P_i \\ U_i^2 b_{ii} + U_i \sum_{j \neq i} U_j y_{ij} \sin(\delta_i - \delta_j + \alpha_{ij}) = Q_i \end{cases} \quad (9.1)$$

Система уравнений (9.1) содержит дифференцируемые функции искомых переменных и, следовательно, для их решения можно использовать метод Ньютона и его модификации.

В матричном виде система уравнений (9.1) может быть записана так:

$$W(x) = 0,$$

где W – вектор-функция небалансов мощности в узлах системы;

x – вектор переменных (модули напряжений и их фазовые углы).

Итерационная формула метода Ньютона, как известно, имеет следующий вид:

$$\frac{\partial W(x^{(k-1)})}{\partial x^{(k-1)}} \Delta x^{(k)} = -W(x^{(k-1)}), \quad (9.2)$$

т.е. решение системы нелинейных алгебраических уравнений методом Ньютона сводится к многократному решению в итерационном цикле системы линейных алгебраических уравнений (9.2), где k – номер итерации, а остальные элементы применительно к системе (9.1) имеют следующий вид:

$$W(x) = \begin{bmatrix} P_{нб}(\delta, U) \\ Q_{нб}(\delta, U) \end{bmatrix} - \text{вектор-функция небалансов;}$$

$$\Delta x^{(k)} = \begin{bmatrix} \Delta \delta^{(k)} \\ \Delta U^{(k)} \end{bmatrix} - \text{вектор-столбец приращении переменных;}$$

$$\frac{\partial W(x^{(k-1)})}{\partial x^{(k-1)}} - \text{квадратная матрица частных производных функций}$$

небалансов мощности в узлах по искомым переменным (матрица Якоби).

Система линейных алгебраических уравнений (9.2) решается на каждой итерации относительно приращении $\Delta x^{(k)}$, которые позволяют определить значения переменных на следующей итерации:

$$x^{(k)} = x^{(k-1)} + \Delta x^{(k)}. \quad (9.3)$$

Матрица Якоби имеет следующий вид:

$$\frac{\partial W(x)}{\partial x} = \begin{bmatrix} \frac{\partial P_{нб}}{\partial \delta} & \frac{\partial P_{нб}}{\partial U} \\ \frac{\partial Q_{нб}}{\partial \delta} & \frac{\partial Q_{нб}}{\partial U} \end{bmatrix}, \quad (9.4)$$

где $\frac{\partial P_{нб}}{\partial \delta}$ – квадратная подматрица порядка $(n-1) \times (n-1)$;

$\frac{\partial P_{nb}}{\partial U}$ – прямоугольная подматрица порядка $(n-1) \times (n-1)$

$\frac{\partial Q_{nb}}{\partial \delta}$ – прямоугольная подматрица порядка $(n-1) \times (n-1)$;

$\frac{\partial Q_{nb}}{\partial U}$ – квадратная подматрица порядка $(n-1) \times (n-1)$.

Элементы указанных подматриц нетрудно определить аналитически, дифференцируя уравнения небалансов активных и реактивных мощностей в узлах (7.1) по соответствующим переменным δ и U .

Расчеты установившихся режимов методом Ньютона, выполненные для схем различного объема и конфигурации, выявили ряд особенностей этого метода:

1) метод весьма чувствителен к выбору начального приближения неизвестных, т.е. для получения локального решения методом Ньютона необходимо хорошее начальное приближение. В современных программно-вычислительных комплексах до начала расчета режима методом Ньютона используется стартовый алгоритм, реализующий метод Зейделя, приближающий значения переменных к решению;

2) метод обладает «квадратичной сходимостью» (т.е. погрешность последующего приближения связана с погрешностью предыдущего приближения квадратичной зависимостью), которая обеспечивается практически за 3 – 5 итераций вне зависимости от объема решаемой задачи и точности расчета;

3) в случае плохой обусловленности матрицы Якоби, когда $\det\left(\frac{\partial W(x)}{\partial x}\right) \rightarrow 0$, сходимость метода Ньютона может существенно замедляться. Это имеет место, когда:

а) рассчитываются режимы, близкие к пределу по существованию;

б) схемы характеризуются резкой неоднородностью параметров;

в) исходные данные задаются с большой погрешностью.

4) хорошая стыковка метода Ньютона с методами решения

оптимизационных задач, что позволяет совместить их решение с расчетом режима.

На практике часто используются различные модификации метода Ньютона, которые отличаются между собой сходимостью итерационного процесса, числом итераций, требуемой оперативной памятью ПК.

В методе Ньютона с обращением матрицы Якоби система линейных алгебраических уравнений (9.2) решается на каждой итерации относительно вектора приращений независимых переменных с использованием обратной матрицы Якоби:

$$\Delta x^{(k)} = - \left[\frac{\partial W(x^{(k-1)})}{\partial x^{(k-1)}} \right]^{-1} \times W(x^{(k-1)}), \quad (9.5)$$

т.е. значения переменных в каждой следующей итерации можно определить по выражению (9.6):

$$x^{(k)} = x^{(k-1)} - \left[\frac{\partial W(x^{(k-1)})}{\partial x^{(k-1)}} \right]^{-1} \times W(x^{(k-1)}). \quad (9.6)$$

Введение параметра t ($0 < t < 1$), т.е. переход от итерационной формулы (9.6) к выражению (9.7), соответствует переходу от метода Ньютона к его модификации, называемой «метод Ньютона по параметру» (при $t = 1$ имеем метод Ньютона):

$$x^{(k)} = x^{(k-1)} - t \times \left[\frac{\partial W(x^{(k-1)})}{\partial x^{(k-1)}} \right]^{-1} \times W(x^{(k-1)}). \quad (9.7)$$

Один из алгоритмов выбора параметра t требует вычисления на каждой итерации нормы вектора небалансов и нормы вектора, построенного на матрице Гессе – матрице вторых производных. Применение метода по параметру позволяет всегда обеспечить сходимость итерационного процесса, если только на пути этого процесса нет поверхности $\left| \frac{\partial W}{\partial x} \right| = 0$, так как в этом случае согласно (9.6) решения не существует.

Основным недостатком методов Ньютона и по параметру является

большой объем вычислений на итерации: вычисление элементов матрицы Якоби и вектора небалансов, решение системы линейных алгебраических уравнений. Поэтому на практике применяются модификации метода Ньютона, которые характеризуются меньшим объемом вычислений на итерации. К ним относятся модифицированный метод Ньютона и метод Ньютона с блочной диагонализацией. В основу этих модификаций положено то, что изменения элементов матрицы Якоби влияют только на сходимость итерационного процесса и не влияют на решение, т.е. $W(x^{(k)}) \rightarrow 0$, если $\Delta x^{(k)} \rightarrow 0$.

Модифицированный метод Ньютона заключается в том, что элементы матрицы Якоби вычисляются только один раз (в точке исходного приближения), т.е. итерационная формула (9.2) преобразуется в формулу:

$$\frac{\partial W(x^{(0)})}{\partial x^{(0)}} \Delta x^{(k)} = -W(x^{(k-1)}). \quad (9.8)$$

Эту матрицу можно однократно факторизовать и использовать в процессе итераций в факторизованном виде, что значительно уменьшает объем вычислений на каждой итерации.

Метод Ньютона с блочной диагонализацией позволяет существенно уменьшить объем расчетов на итерации за счет отбрасывания недиагональных блоков матрицы Якоби, т.е. полагая, что $\frac{\partial P_{нб}}{\partial U} = 0$ и

$\frac{\partial Q_{нб}}{\partial \delta} = 0$. Тогда исходная система уравнений (9.2) приобретает вид:

$$\begin{bmatrix} \frac{\partial P_{нб}}{\partial \delta} & 0 \\ 0 & \frac{\partial Q_{нб}}{\partial U} \end{bmatrix} \times \begin{bmatrix} \Delta \delta^{(k)} \\ \Delta U^{(k)} \end{bmatrix} = - \begin{bmatrix} P_{нб}(\delta^{(k-1)}, U^{(k-1)}) \\ Q_{нб}(\delta^{(k-1)}, U^{(k-1)}) \end{bmatrix}, \quad (9.9)$$

а итерационный процесс в этом случае будет заключаться в решении двух систем уравнений (9.10):

$$\begin{aligned}\frac{\partial P_{нб}}{\partial \delta} \Delta \delta^{(k)} &= -P_{нб}(\delta^{(k-1)}, U^{(k-1)}), \\ \frac{\partial Q_{нб}}{\partial U} \Delta U^{(k)} &= -Q_{нб}(\delta^{(k-1)}, U^{(k-1)}).\end{aligned}\tag{9.10}$$

Такое упрощение сказывается только на сходимости метода Ньютона и несмотря на некоторое уменьшение скорости сходимости вычислительная эффективность решения может повыситься за счет снижения объема расчетов на итерации.

Модифицированный метод Ньютона и метод Ньютона с блочной диагонализацией отличаются между собой областью сходимости, числом итераций и временем, требуемым для получения решения с заданной точностью, однако значительное уменьшение объема расчетов на каждой итерации повышают вычислительную эффективность, несмотря на некоторое снижение скорости сходимости.

Реализуем классический алгоритм метода Ньютона. Алгоритм метода Ньютона состоит из двух основных этапов:

- 1) составление матрицы частных производных (матрицы Якоби);
- 2) решение этой матрицы, то есть решение системы вида (9.2).

Как было показано в предыдущей работе, библиотека NumPy позволяет довольно просто решать системы линейных уравнений. Осталось реализовать первый этап метода Ньютона, а именно вычисление матрицы Якоби. Для этого можно воспользоваться численным дифференцированием. Численное дифференцирование – совокупность методов приближённого вычисления значения производной некоторой функции, заданной таблично или имеющей сложное аналитическое выражение.

Производная функции f в точке x определяется с помощью предела:

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

Для приближенного вычисления производной можно воспользоваться формулой

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

с некоторым достаточно малым шагом h .

С учетом всего вышесказанного, функция расчета режима методом Ньютона будет выглядеть следующим образом:

```

78 | def newton(self):
79 |     cnt=0
80 |     for n in self.nodes:
81 |         n.V=complex(n.Unom)
82 |         if n.type!=0:
83 |             cnt+=2
84 |
85 |     for it in range(20):
86 |         A = np.zeros((cnt,cnt))
87 |         B = np.zeros(cnt)
88 |         row = 0
89 |         for i in self.nodes:
90 |             if i.type==0:
91 |                 continue
92 |             B[row] = i.Neb().real-i.P
93 |             B[row+1] = i.Neb().imag-i.Q
94 |             col=0
95 |             for j in self.nodes:
96 |                 if j.type==0:
97 |                     continue
98 |                 A[row][col] = i.dFdU(j).real
99 |                 A[row][col+1] = i.dFdD(j).real
100 |                 A[row+1][col] = i.dFdU(j).imag
101 |                 A[row+1][col+1] = i.dFdD(j).imag
102 |                 col+=2
103 |             row+=2
104 |         X = np.linalg.solve(A,B)
105 |         col = 0
106 |         for n in self.nodes:
107 |             if n.type!=0:
108 |                 n.V = cmath.rect(abs(n.V)+X[col],cmath.phase(n.V)+X[col+1])
109 |                 col+=2

```

В строках 80-83 задается начальное приближение расчетных напряжений, равное номинальному напряжению узла. Одновременно подсчитывается размерность будущей матрицы (переменная cnt).

На строке 85 начинается вычислительный итерационный процесс методом Ньютона. Создается матрица A и вектор свободных членов B будущей систему уравнений с помощью библиотеки NumPy.

В строках 98-101 заполняется матрица Якоби, используя формулы, описанные выше и функции численного вычисления частных производных dFdU и dFdD.

В строках 106-109 задаются новые приближения модуля напряжения и фазового угла, полученные по результатам решения линейной системы уравнений с помощью библиотеки NumPy.

Функции dFdU и dFdD необходимо добавить в класс узла:

```
196 | def dFdU(self,N):
197 |     U      = abs(N.V)
198 |     D      = cmath.phase(N.V)
199 |     N.V     = cmath.rect(U-0.001,D)
200 |     F1     = self.Neb()
201 |     N.V     = cmath.rect(U+0.001,D)
202 |     F2     = self.Neb()
203 |     N.V     = cmath.rect(U,D)
204 |     dF      = (F1-F2)/0.002
205 |     return dF
206 |
207 | def dFdD(self,N):
208 |     U      = abs(N.V)
209 |     D      = cmath.phase(N.V)
210 |     N.V     = cmath.rect(U,D-0.001)
211 |     F1     = self.Neb()
212 |     N.V     = cmath.rect(U,D+0.001)
213 |     F2     = self.Neb()
214 |     N.V     = cmath.rect(U,D)
215 |     dF      = (F1-F2)/0.002
216 |     return dF
```

Функция dFdU вычисляет частную производную функции небаланса мощности по напряжению узла N, а функция dFdD вычисляет частную производную функции небаланса мощности по фазе узла N. При численном дифференцировании используется шаг, равный 0.001.

Пример расчета режима методом Ньютона:

```
377 | s=SCHEME()
378 | s.load_scheme("E:\Cхема.csv")
379 | s.assemble()
380 | s.display()
381 | s.Newton()
382 | s.display()
383 | s.save_scheme("E:\out.txt")
```

Задание

1. Разработайте программу, реализующую расчет режима методом Ньютона.
2. Протестируйте программу на примере небольшой схемы сети. Сравните полученные результаты с программой RastrWin.

Контрольные вопросы

1. В чем состоит суть метода Ньютона.
2. Приведите основные формулы метода Ньютона.

Практическая работа №10

Тема: «Расчет потерь электроэнергии»

Как известно, потери электроэнергии в электрической сети могут быть разделены на:

- нагрузочные потери;
- потери холостого хода.

Для расчета потерь холостого хода в классе «LINE» создадим функцию `dSxx()`, которая будет возвращать комплексное число. Действительная часть возвращаемого числа – потери активной мощности, мнимая часть возвращаемого числа – потери реактивной мощности. При использовании «П» - образной схемы замещения для линий и «Г» - образной схемы замещения для трансформаторов, функция `dSxx` будет иметь следующий вид:

```
54 |     def dSxx(self):
55 |         Y=complex(self.G,self.B)/1000000
56 |         U1=abs(self._ip.V)
57 |         U2=abs(self._iq.V)
58 |
59 |         if self.KTR==0: return Y/2*(U1*U1+U2*U2)
60 |         else:           return Y*U1*U1
```

Для расчета нагрузочных потерь мощности аналогичным образом создадим в классе «LINE» функцию «`dSnagr()`», которая также будет возвращать комплексное число. Функция будет иметь следующий вид:

```
62 |     def dSnagr(self):
63 |         Y=1/complex(self.R,self.X)
64 |         dU=0
65 |         if self.KTR==0:
66 |             dU=abs(self._ip.V-self._iq.V)
67 |         else:
68 |             dU=abs(self._ip.V-self._iq.V/self.KTR)
69 |         return Y.conjugate()*dU*dU
```

Для вывода на экран информации о потерях мощности в каждой линии и трансформаторе, в классе «SCHEME» создадим функцию dS(), текст которой приведен ниже.

```
233     def dS(self):
234         dSxx_line = complex()
235         dSxx_tr   = complex()
236         dSnagr_line = complex()
237         dSnagr_tr   = complex()
238         print("№_нач\t№_кон\tSxx\tSnagr")
239         for l in self.lines:
240             print(l.ip, "\t", l.iq, "\t", l.dSxx(), "\t", l.dSnagr())
241             if l.KTR==0:
242                 dSxx_line += l.dSxx()
243                 dSnagr_line += l.dSnagr()
244             else:
245                 dSxx_tr += l.dSxx()
246                 dSnagr_tr += l.dSnagr()
247
248         print("Нагрузочные потери в линиях: \t", dSnagr_line)
249         print("Потери XX в линиях: \t", dSxx_line)
250         print("Нагрузочные потери в трансформаторах: \t", dSnagr_tr)
251         print("Потери XX в трансформаторах: \t", dSxx_tr)
```

Помимо поэлементного вывода о результатах расчета потерь мощности, функция также выводит суммарные нагрузочные потери и потери холостого хода в линиях и трансформаторах.

Задание для выполнения

1. Разработайте программу, реализующую расчет потерь мощности в элементах сети.
2. Реализуйте сохранение результатов расчета потерь мощности в текстовый файл. Предусмотрите возможность выбора количества знаков после запятой для сохранения результатов.
3. Доработайте функции расчета нагрузочных потерь и потерь холостого хода для использования «Т» - образной и обратной «Г» - образной схем замещения линий и трансформаторов соответственно.

Контрольные вопросы

1. Какая формула используется для определения потерь холостого хода в линиях электропередач?

2. Какая формула применяется для определения потерь холостого хода в трансформаторах?

3. Какая формула применяется для определения нагрузочных потерь мощности в трансформаторах?

4. Какая формула применяется для определения нагрузочных потерь в линиях электропередач?

5. Какая формула используется для определения потерь мощности в компенсирующих устройствах?

Практическая работа №11

Тема: «Визуализация данных»

Цель: Научиться создавать простейший пользовательский интерфейс приложения

Язык Python обрел свою популярность в том числе благодаря множеству удобных библиотек для визуализации данных. Одной из наиболее распространенных таких библиотек является библиотека Matplotlib. Для ее установки необходимо через командную строку операционной системы выполнить следующую команду:

```
pip install matplotlib
```

При выполнении этой команды будет автоматически скачана последняя версия программы и записана в нужный каталог системы. Далее необходимо импортировать функции для работы с графикой. Библиотека Matplotlib имеет несколько модулей. Практически все задачи, связанные с построением графиков, можно решить, используя возможности, которые предоставляет модуль pyplot. Для его подключения в программу его необходимо импортировать, используя соответствующую команду:

```
import matplotlib.pyplot as plt
```

Здесь мы задаем псевдоним plt, который позволяет сократить объем кода и сделать его более читаемым. Вместо слова plt вы можете придумать любой другой псевдоним.

Библиотека Matplotlib имеет огромное количество функций и их параметров. Рассмотреть все возможности библиотеки в рамках одного занятия не предоставляется возможным. Целью настоящей лабораторной работы является получение общих представлений об устройстве библиотеки

и ее основных функциях, достаточных для дальнейшего ее самостоятельного изучения с помощью официальной документации.

Рассмотрим основные термины и понятия, касающиеся изображения графика, с которыми вам необходимо будет познакомиться, для того, чтобы в дальнейшем у вас не было трудностей при работе с документацией по библиотеке `matplotlib`.

Корневым элементом при построения графиков в системе `Matplotlib` является Фигура (Figure). Все, что нарисовано на рисунке является элементами фигуры. Рассмотрим ее составляющие более подробно.

График

На рисунке представлены два графика – линейный и точечный. `Matplotlib` предоставляет огромное количество различных настроек, которые можно использовать для того, чтобы придать графику вид, который вам нужен: цвет, толщина и тип линии, стиль линии и многое другое, все это вы можете найти в документации к библиотеке.

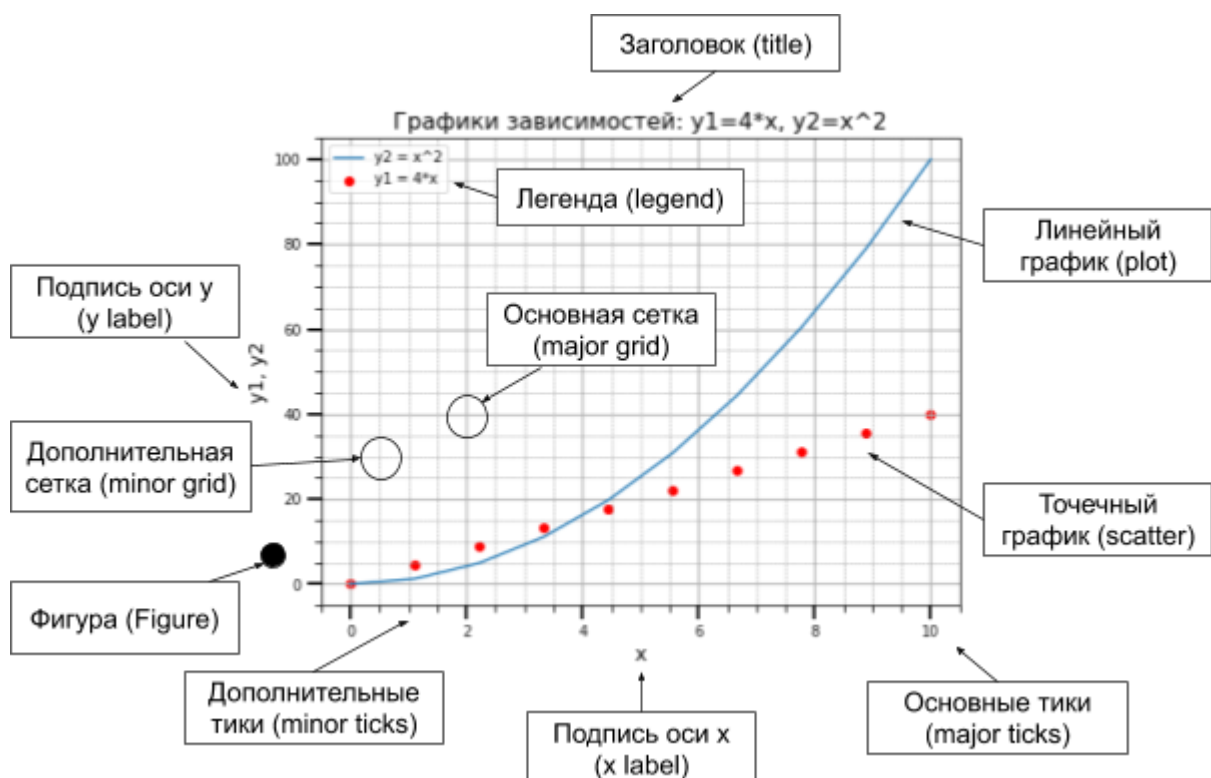


Рисунок 11.2 – Основные графические элементы библиотеки `Matplotlib`

Оси

Вторым, после непосредственно самого графика, по важности элементом фигуры являются оси. Для каждой оси можно задать метку (подпись), основные (major) и дополнительные (minor) тики, их подписи, размер и толщину, также можно задать диапазоны по каждой из осей.

Сетка и легенда

Следующими элементами фигуры, которые значительно повышают информативность графика являются сетка и легенда. Сетка также может быть основной (major) и дополнительной (minor). Каждому типу сетки можно задавать цвет, толщину линии и тип. Для отображения сетки и легенды используются соответствующие команды.

Основным элементом изображения, которое строит pyplot является Фигура (Figure), на нее накладываются один или более графиков, осей, надписей и т.п. Для построения графика используется команда plot(). В самом минимальном варианте можно ее использовать без параметров:

```
import matplotlib.pyplot as plt
plt.plot()
plt.show()
```

Первая функция (plot) строит график, вторая функция (show) отображает его на экране.

В результате выполнения кода будет выведено пустое поле:

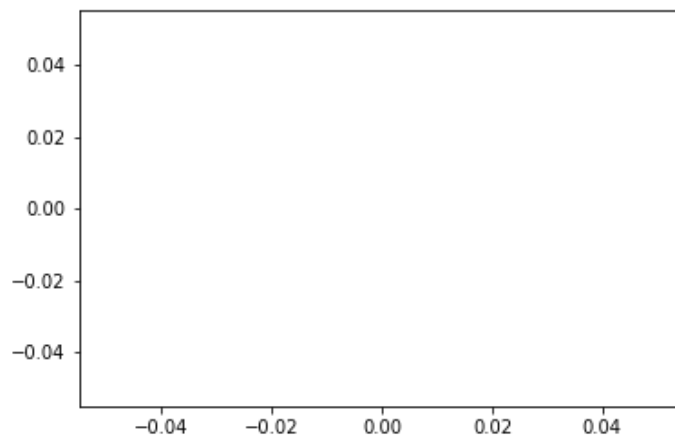


Рисунок 11.3 – Построение пустого графика

Если в качестве параметра функции `plot()` передать список, то значения из этого списка будут отложены по оси ординат (ось *y*), а по оси абсцисс (ось *x*) будут отложены индексы элементов массива:

```
plt.plot([1, 7, 3, 5, 11, 1])
```

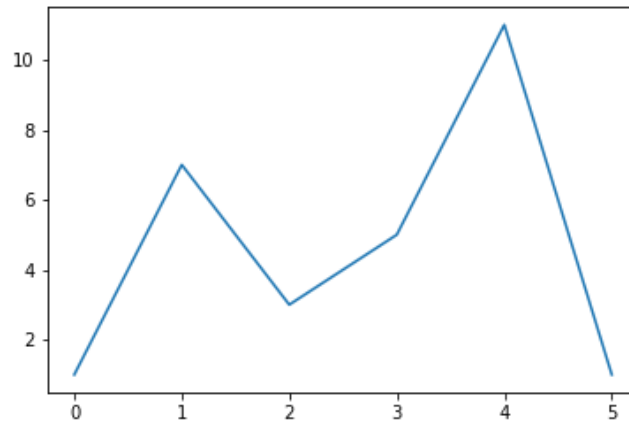


Рисунок 11.4 – Построение графика на основе списка

Для того, чтобы задать значения по осям *x* и *y* необходимо в `plot()` передать два списка:

```
plt.plot([1, 5, 10, 15, 20], [1, 7, 3, 5, 11])
```

В результате получим следующий график:

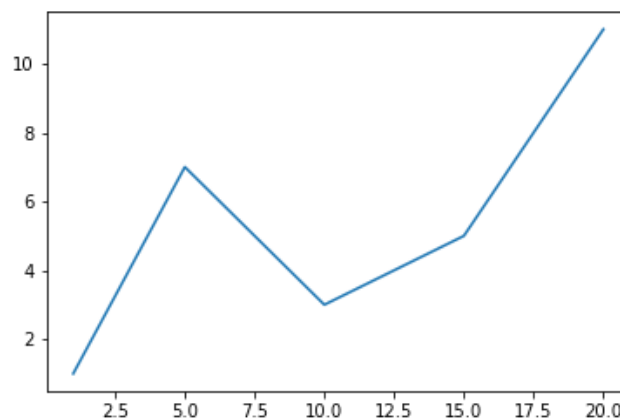


Рисунок 11.5 – Построение графика на основе двух списков

Таким образом, если нам нужно вывести какие-то данные, очевидно, нам нужно предварительно их подготовить. Например, если мы хотим

построить график синусоиды, можно использовать рассмотренные ранее циклы и словари:

```
import math
import matplotlib.pyplot as plt
a=dict()
for i in range(0, 360):
    a[i]=math.sin(i/180*math.pi)

plt.plot(a.keys(),a.values())
plt.show()
```

В результате будет создан следующий график:

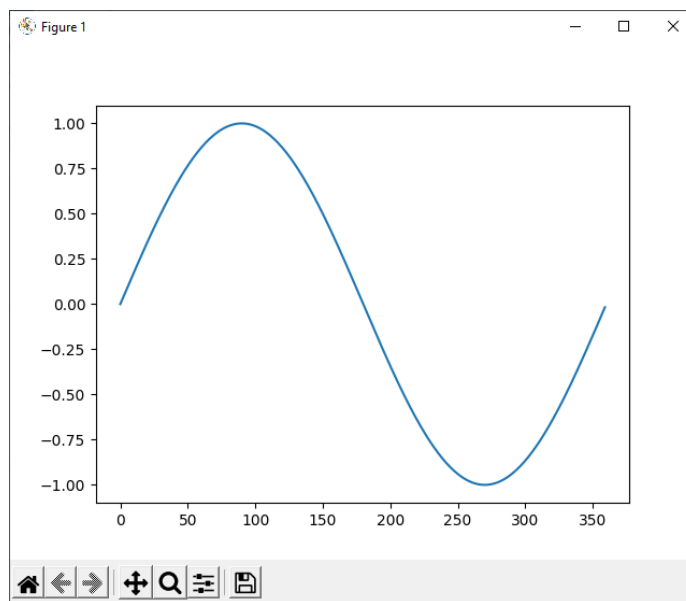


Рисунок 11.6 – Пример построение графика синусоиды

Здесь встречается не рассмотренная ранее библиотека `math`. Она содержит множество наиболее часто используемых математических функций, в том числе необходимую нам тригонометрическую функцию синуса. Список всех доступных функций библиотеки `math` можно найти в документации на официальном сайте проекта Python.

Текстовые надписи на графике

Наиболее часто используемые текстовые надписи на графике это:

- наименование осей;

- наименование самого графика;
- текстовое примечание на поле с графиком;
- легенда.

Рассмотрим кратко данные элементы.

Для задания подписи оси x используется функция `xlabel()`, оси y – `ylabel()`. Разберемся с аргументами данных функций.

Аргументов у этих функций довольно много и они позволяют достаточно тонко настроить внешний вид надписей. В рамках настоящей работы мы рассмотрим только основные из них. Полный список всегда можно посмотреть в официальной документации. Для функций `xlabel()/ylabel()` основными являются следующие аргументы:

- Текст подписи: **str** (первый аргумент);
- Расстояние между областью графика, включающую оси, и меткой: **labelpad** (численное значение, по умолчанию отсутствует).

Функции `xlabel()/ylabel()` принимают в качестве аргументов параметры класса `matplotlib.text.Text`, некоторые из них нам могут пригодиться:

- Размер шрифта: `fontsize` или `size` – число либо значение из списка: `{‘xx-small’, ‘x-small’, ‘small’, ‘medium’, ‘large’, ‘x-large’, ‘xx-large’}`.
- Стилль шрифта: `fontstyle` – значение из списка: `{‘normal’, ‘italic’, ‘oblique’}`;

Толщина шрифта: `fontweight` – число в диапазоне от 0 до 1000 либо значение из списка: `{‘ultralight’, ‘light’, ‘normal’, ‘regular’, ‘book’, ‘medium’, ‘roman’, ‘semibold’, ‘demibold’, ‘demi’, ‘bold’, ‘heavy’, ‘extra bold’, ‘black’}`.

Цвет шрифта – `color`: один из доступных способов определения цвета (см. Цвет линии в официальной документации).

Пример использования:

```
plt.xlabel('Day', fontsize=15, color='blue')
```

Заголовок графика

Для задания заголовка графика используется функция `title()`:

```
plt.title('Chart price', fontsize=17)
```

Из параметров отметим следующие:

- Текст заголовка (первый параметр, label, тип - str).
- Выравнивание заголовка: loc, значение из набора {'center', 'left', 'right'}

Для функции title() также доступны параметры конструктора класса matplotlib.text.Text, часть из которых рассмотрены в описании аргументов функций xlabel() / ylabel().

Текстовое примечание

За размещение текста на поле графика отвечает функция text(), которой вначале передаются координаты позиции надписи, после этого – текст самой надписи.

```
plt.text(1, 1, 'type: Steel')
```

Легенда

Легенда будет размещена на графике, если вызвать функцию legend(). Функция позволяет настроить расположение легенды, размер шрифта и многие другие параметры. В рамках настоящей работы мы не будем рассматривать аргументы функции, при желании вы можете изучить их самостоятельно.

Давайте немного наведем красоту на наших графиках, используя рассмотренные функции:

```
import math
import matplotlib.pyplot as plt
a=dict()

for i in range(0, 360):
    a[i]=math.sin(i/180*math.pi)

plt.title("y = sin(x)") # заголовок
plt.xlabel("x") # ось абсцисс
plt.ylabel("y") # ось ординат
plt.grid() # включение отображение сетки
```

```
plt.plot(a.keys(),a.values())
plt.legend(['Синус'])
plt.show()
```

Результат работы кода:

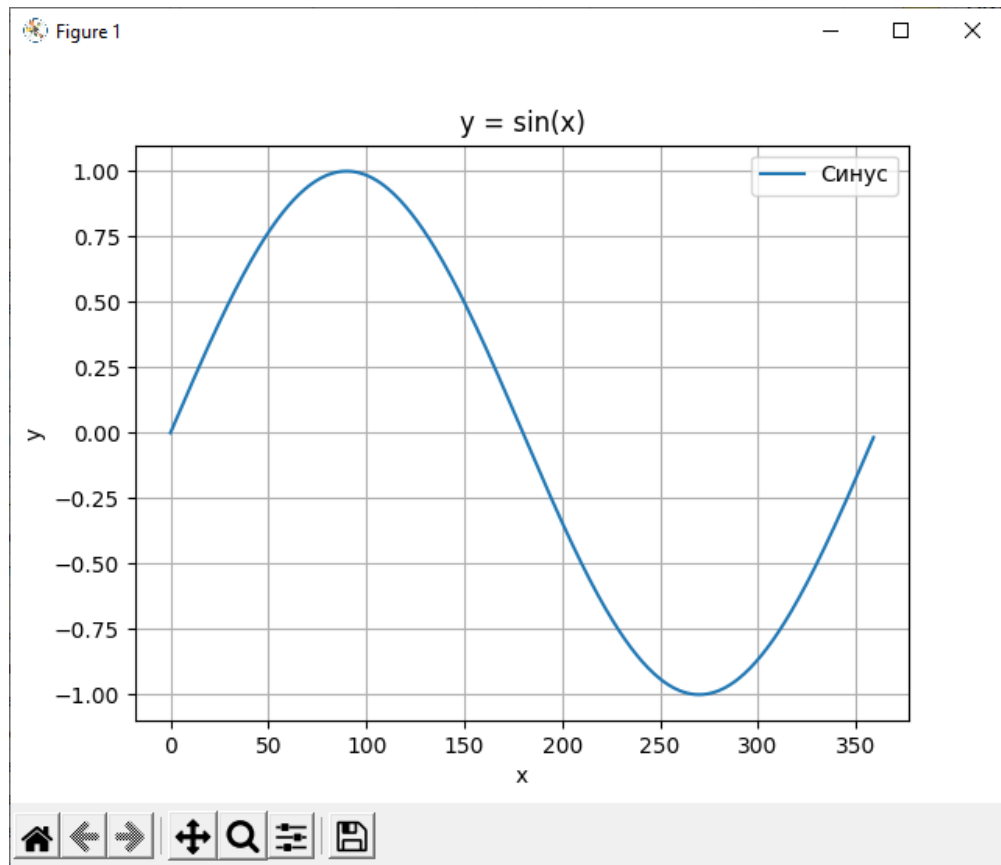


Рисунок 11.7 – Настройка параметров графика

Помимо рассмотренных функций, здесь мы использовали еще одну функцию `grid`. Она позволяет отобразить сетку на графике. Функция позволяет настроить цвет сетки, стиль линии, их толщину, шаг сетки, отображения по осям и так далее.

Например,

```
plt.grid(color='r', linestyle='--', linewidth=2)
```

нарисует следующую сетку:

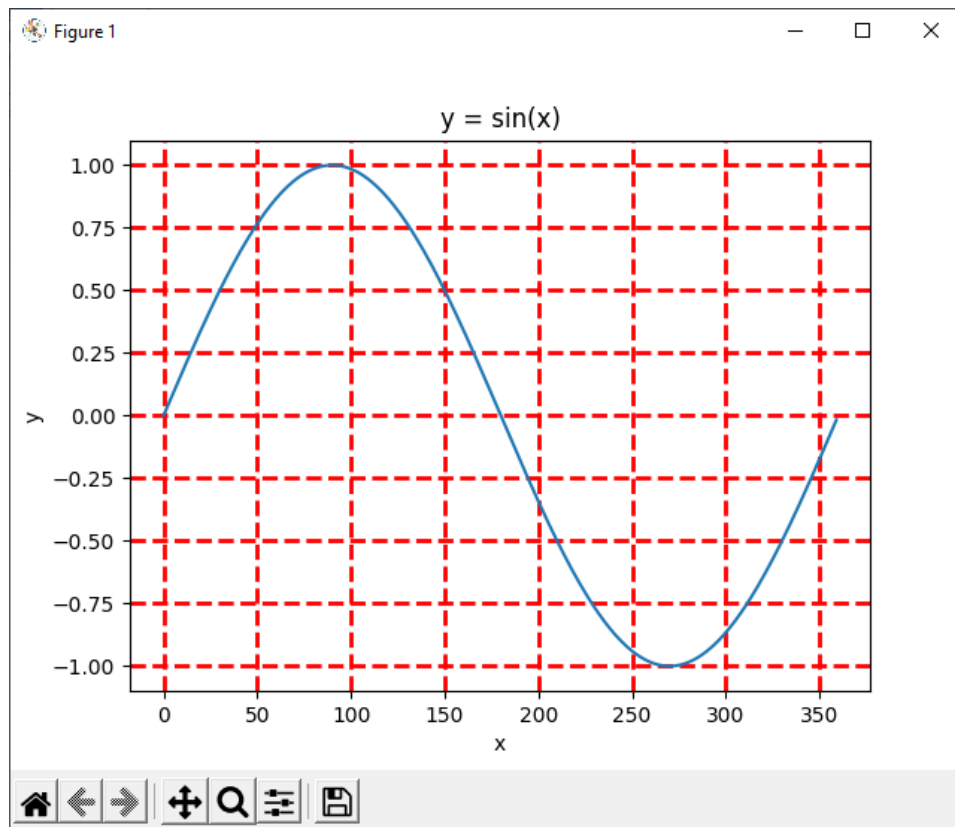


Рисунок 11.8 – Настройка параметров сетки

Список всех параметров функции `grid` и описание множества других полезных функции пакета `matplotlib.pyplot` можно найти в документации на официальном сайте библиотеки `matplotlib`.

Кстати функция `plot` тоже позволяет настраивать цвет, толщину и стиль линии с помощью этих же параметров (`color`, `linestyle`, `linewidth`).

Если нам нужно построить два графика на одном листе, мы просто вызываем второй раз функцию `plot`.

```
import math
import matplotlib.pyplot as plt
a=dict()
b=dict()
for i in range(0, 360):
    a[i]=math.sin(i/180*math.pi)
    b[i]=math.cos(i/180*math.pi)

plt.title("y = sin(x)") # заголовок
plt.xlabel("x") # ось абсцисс
plt.ylabel("y") # ось ординат
plt.grid(color='r', linestyle='--', linewidth=2) #
# включение отображение сетки
```

```
plt.plot(a.keys(),a.values(),linestyle = '--',color = (0, 0, 0, 1),linewidth = 12)
plt.plot(b.keys(),b.values())
plt.show()
```

В результате выполнению получим следующий график:

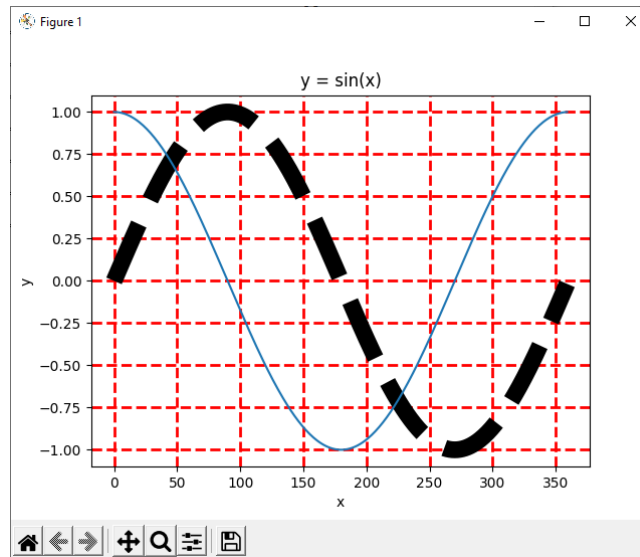


Рисунок 11.9 – Отображение двух графиков в одной рабочей области

Библиотека также позволяет отобразить два графика не накладывая их друг на друга.

```
import math
import matplotlib.pyplot as plt
a=dict()
b=dict()
for i in range(0, 360):
    a[i]=math.sin(i/180*math.pi)
    b[i]=math.cos(i/180*math.pi)

plt.figure(figsize=(9, 9))
plt.subplot(2, 1, 1)
plt.title("y = sin(x)") # заголовок
plt.xlabel("x") # ось абсцисс
plt.ylabel("y") # ось ординат
plt.grid(color='r', linestyle='--', linewidth=2) #
включение отображение сетки
plt.plot(b.keys(),b.values())
plt.subplot(2, 1, 2)
plt.title("y = cos(x)") # заголовок
plt.xlabel("x") # ось абсцисс
plt.ylabel("y") # ось ординат
plt.grid(color='r', linestyle='--', linewidth=2) #
включение отображение сетки
```

```
plt.plot(a.keys(),a.values(),linestyle = '--',color = (0, 0, 0, 1),linewidth = 12)
plt.show()
```

Результатом выполнения кода будет следующий график:

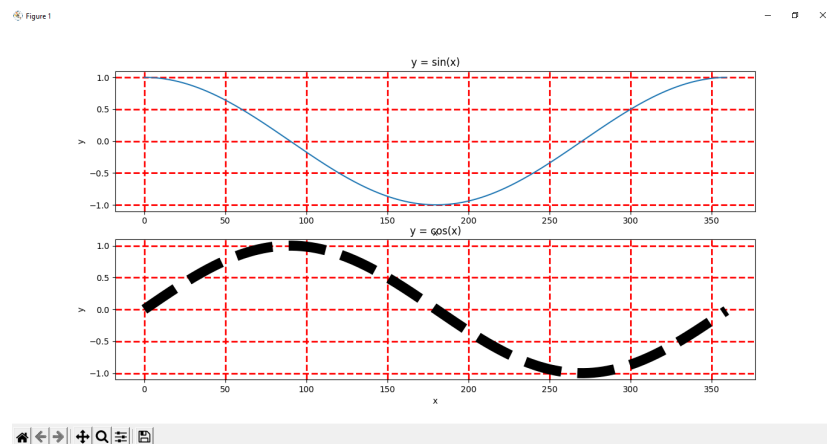


Рисунок 11.10 – Отображение графиков в разных рабочих областях

Здесь мы воспользовались новыми функциями:

`figure()` – функция для задания глобальных параметров отображения графиков. В нее, в качестве аргумента, мы передаем кортеж, определяющий размер общего поля (размер указывается в дюймах; при необходимости указания в сантиметрах, необходимо разделить нужные размеры на 2.54).

`subplot()` – функция для задания местоположения поля с графиком. Существует несколько способов задания областей для вывода через функцию `subplot()` мы воспользовались следующим: первый аргумент – количество строк, второй – столбцов в формируемом поле, третий – индекс (номер поля, считаем сверху вниз, слева направо).

Тип графика

До этого момента мы работали только с линейными графиками, функция `plot()` позволяет задать тип графика: линейный либо точечный, при этом для точечного графика можно указать соответствующий маркер. Приведем пару примеров:

```
plt.plot(x, y, 'ro')
```

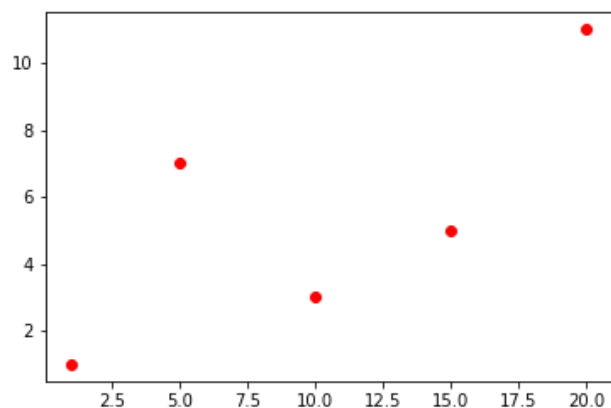


Рисунок 11.11 – Точечный график с круговым маркером

```
plt.plot(x, y, 'bx')
```

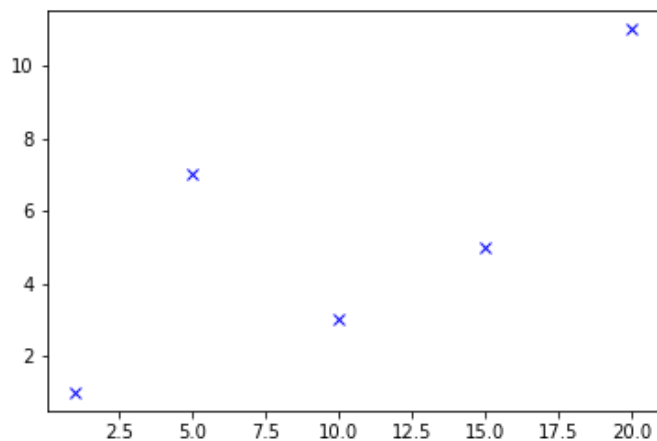


Рисунок 11.12 – Точечный график с маркером типа x

Первая буква задает цвет маркера ('b' – синий; 'g' – зеленый, 'r' – красный, 'c' – бирюзовый; 'm' – фиолетовый (пурпурный); 'y' – желтый; 'k' – черный; 'w' – белый), вторая буква задает тип маркера: '.' – точка, ',' – пиксель; 'o' – окружность; 'v' – треугольник, направленный вниз; '^' – треугольник, направленный вверх; '<' – треугольник, направленный влево; '>' – треугольник, направленный вправо; 's' – квадрат; 'p' – пятиугольник; '*' – звезда; 'h' – шестиугольник; '+' – плюс; 'x' – X-образный маркер; 'D' – ромб; '|' – вертикальная линия; '_' – горизонтальная линия. Размер маркера можно менять.

Столбчатые и круговые диаграммы

Помимо линейных графиков, библиотека Matplotlib позволяет также строить столбчатые и линейные диаграммы. Для их построения используются функции:

`bar()` – для построения вертикальной диаграммы;

`barh()` – для построения горизонтальной диаграммы;

`pie()` – для построения круговых диаграмм.

Примеры:

```
import matplotlib.pyplot as plt

fruits=["apple","peach","orange","bannana","melon"]
counts = [34, 25, 43, 31, 17]

plt.bar(fruits, counts)
plt.title("Fruits!")
plt.xlabel("Fruit")
plt.ylabel("Count")
plt.show()
```

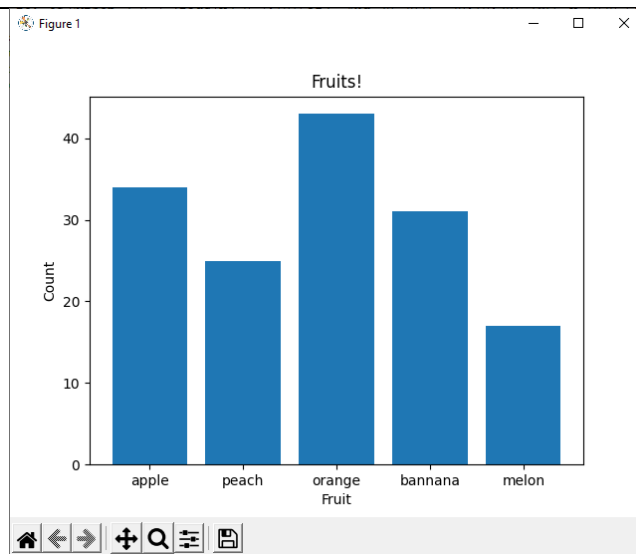


Рисунок 11.13 – Столбчатая диаграмма

```
import matplotlib.pyplot as plt
fruits=["apple","peach","orange","bannana","melon"]
counts = [34, 25, 43, 31, 17]
plt.barh(fruits, counts)
plt.title("Fruits!")
plt.xlabel("Fruit")
plt.ylabel("Count")
plt.show()
```

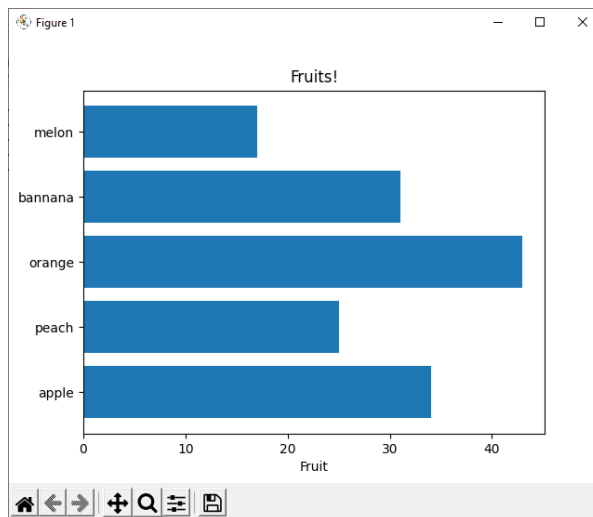


Рисунок 11.14 – Вертикальная столбчатая диаграмма

```
import matplotlib.pyplot as plt
fruits=["apple","peach","orange","bannana","melon"]
counts = [34, 25, 43, 31, 17]
plt.pie(counts,labels=fruits)
plt.title("Fruits!")
plt.show()
```

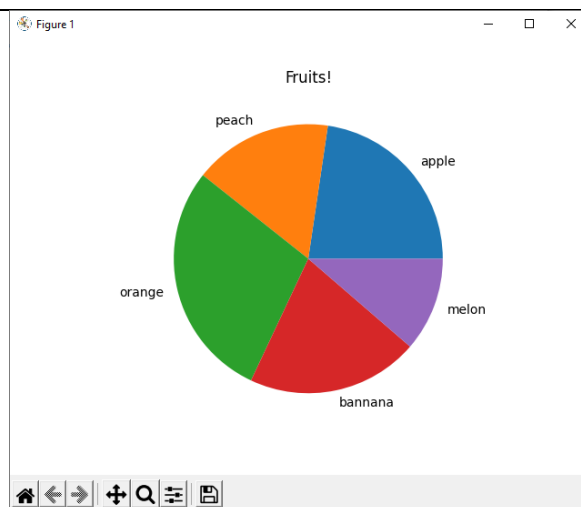


Рисунок 11.15 – Круговая столбчатая диаграмма

Также как и линейные графики, диаграммы имеют множество настроек. Рассмотреть их все в рамках одной лабораторной работы не предоставляется возможным, но вы можете это сделать самостоятельно, используя официальную документацию библиотеки Matplotlib.

Задание для выполнения

1. Разработайте программу, которая будет отображать в виде графика модули напряжений по узлам.
2. Разработайте программу, которая будет отображать в виде круговой диаграммы структура потерь (нагрузочные потери и потери холостого хода по линиям и трансформаторам).
3. Разработайте программу, которая будет отображать в виде столбчатой диаграммы структуру потерь для каждой ветви.

Контрольные вопросы

1. Для чего нужна библиотека Matplotlib?
2. Расскажите об основных терминах и понятиях библиотеки Matplotlib.
3. Какие основные типы графиков и диаграмм можно построить с помощью библиотеки Matplotlib?
4. Какая функция используется для построения линейных графиков? Какие у нее есть параметры?
5. Какая функция используется для построения точечных графиков? Какие у нее есть параметры?
6. Какая функция используется для построения столбчатых диаграмм?
7. Какая функция используется для построения круговых диаграмм?
8. Как отобразить два и более графиков на одном листе?

Практическая работа №12

Тема: «Графический интерфейс пользователя»

Цель: Познакомиться с простейшими возможностями языка Python для создания графического интерфейса пользователя

Язык Python позволяет создавать не только консольные (работающие в режиме командной строки) приложения, но и относительно просто разработать приложение с простейшим графическим пользовательским интерфейсом. Одной из самых простых библиотек, позволяющих быстро создать приложение с графическим интерфейсом является библиотека PySimpleGUI. Для ее использования сначала необходимо ее установить. Для этого в командной строке Windows необходимо набрать следующую команду:

```
pip install pysimplegui
```

Далее в коде нашей программы необходимо подключить эту библиотеку:

```
import PySimpleGUI as sg
```

Создание графического интерфейса происходит всего одной строчкой:

```
window = sg.Window('Программа расчета режима', layout )
```

Первый аргумент функции – это заголовок окна, а второй – список элементов управления в окне. Давайте создадим приложение, в котором будут следующие элементы:

- кнопка загрузки файла схемы;
- кнопка расчета режима;
- кнопка сохранения результатов в файл;
- кнопка отображения графика;
- таблица узлов;

- таблица ветвей.

В этом случае переменная layout будет иметь следующий вид:

```
272 layout = [  
273     [sg.Button('Открыть схему', key='-OPEN-', enable_events=True),  
274     sg.Button('Рассчитать режим', key='-CALC-', enable_events=True),  
275     sg.Button('Сохранить результат', key='-SAVE-', enable_events=True),  
276     sg.Button('Построить график', key='-GRAPH-', enable_events=True)  
277 ],  
278     [sg.Text('Таблица узлов:'),  
279     [sg.Table(values=[], headings=node_cols, key='-NODES-', auto_size_columns=False, col_widths=node_width)],  
280     [sg.Text('Таблица ветвей:'),  
281     [sg.Table(values=[], headings=line_cols, key='-LINES-', auto_size_columns=False, col_widths=line_width)],  
282 ]
```

Как мы видим, это список, состоящий из элементов sg.Button – кнопка, sg.Text – текст, sg.Table – таблица. Библиотека PySimpleGUI позволяет добавлять и другие известные элементы (переключатели, ползунки, флажки и так далее). Но мы будем использовать в своей программе только 3 из них. Подробную документацию по каждому типу элементов можно найти в интернете.

Как видно из примера, каждый элемент имеет несколько параметров:

- заголовок элемента;
- уникальное имя (key);
- дополнительные настройки.

Переменная enable_events=True говорит о том, что в программе необходимо реагировать на действия пользователей, связанные с этим элементом (нажатие на кнопку, выбор элементов в списке и так далее).

В таблицах есть два дополнительных параметра: headings и col_width.

Это список имен и ширины колонок. Объявим их:

```
266 node_cols=['№ узла', 'Название', 'Тип', 'Уном', 'кВ', 'Р, МВт', 'Q, Мвар', 'U, кВ', 'delta, град']  
267 node_width=[10,10,10,10,10,10,10,10,10]  
268  
269 line_cols = ['№ нач.', '№ кон.', 'R, Ом', 'X, Ом', 'G, мкСм', 'B, мкСм', 'КТР']  
270 line_width=[10,10,12,12,12,12,12]  
271
```

Для обработки действий пользователя необходимо использовать следующую конструкцию:

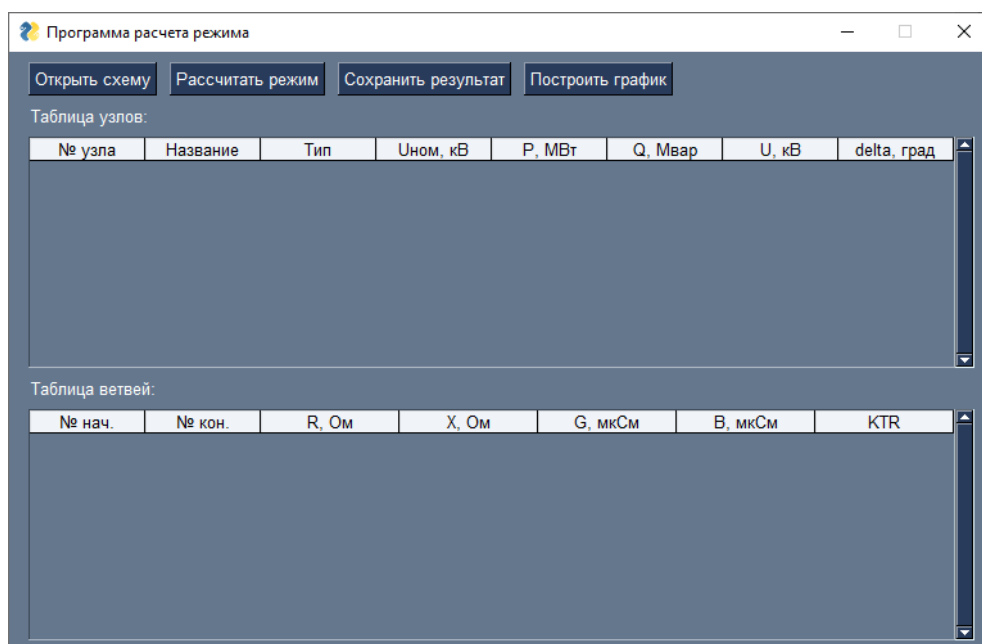
```

291 while True:
292     event, values = window.read()
293     if event == sg.WIN_CLOSED:
294         break
295     if event=='-OPEN-':
296         pass
297
298     if event=='-CALC-':
299         pass
300
301     if event=='-SAVE-':
302         pass
303
304     if event=='-GRAPH-':
305         pass
306
307 window.close()
308

```

Здесь в непрерывном цикле мы проверяем, какое действие произошло. Если пользователь нажал на кнопку «Заккрыть» (строка 293), закрываем приложение (строка 307).

После запуска приложение будет иметь следующий вид:



Теперь выполним обработку нажатия каждой из кнопок. Если пользователь нажал кнопку «Открыть схему»:

```

295     if event=='-OPEN-':
296         file=sg.popup_get_file("Выберите файл",no_window=True)
297         if file!='':
298             s.load_scheme(file)
299             window['-NODES-'].update(get_list(s.nodes))
300             window['-LINES-'].update(get_list(s.lines))
301

```

Функция `popup_get_file` открывает стандартное диалоговое окно выбора файла. Функция `update` обновляет соответствующую таблицу. В нее нужно передать список, заполненный данными. Его предварительно формирует функция `get_list`, которая выглядит следующим образом:

```

252 def get_list(lst):
253     ret=[]
254     for i in lst:
255         l=[]
256         for n,v in vars(i).items():
257             if n[0]==' ':continue
258             if n=='V':
259                 l.append(abs(v))
260                 l.append(cmath.phase(v)*180/cmath.pi)
261             else:
262                 l.append(v)
263         ret.append(l)
264     return ret

```

Здесь мы пропускаем все переменные, которые начинаются с префикса «_», а комплексную переменную V представляем в виде модуля и фазового угла. После выбора файла со схемой, программа будет выглядеть следующим образом:

Программа расчета режима

Открыть схему | Рассчитать режим | Сохранить результат | Построить график

Таблица узлов:

№ узла	Название	Тип	Уном, кВ	P, МВт	Q, Мвар	U, кВ	delta, град
1	Узел №1	False	110.0	0.0	0.0	0.0	0.0
2	Узел №2	True	110.0	0.0	0.0	0.0	0.0
3	Узел №3	True	110.0	1.0	3.0	0.0	0.0
4	Узел №4	True	110.0	2.0	4.0	0.0	0.0
5	Узел №5	True	10.0	2.0	1.0	0.0	0.0

Таблица ветвей:

№ нач.	№ кон.	R, Ом	X, Ом	G, мкСм	B, мкСм	KTR
1	2	4.95	10.5	10.0	-50.0	0.0
2	3	4.95	10.5	10.0	-50.0	0.0
2	4	4.95	10.5	10.0	-50.0	0.0
4	5	3.0	10.0	10.0	100.0	0.09

При нажатии на кнопку «Рассчитать режим», необходимо запустить одну из функций расчета режима и обновить таблицы:

```

302     if event=='-CALC-':
303         s.assemble()
304         s.two_steps(20,0.00001)
305         window['-NODES-'].update(get_list(s.nodes))
306         window['-LINES-'].update(get_list(s.lines))

```

В таблице будут выведены соответствующие результаты:

Программа расчета режима

Открыть схему | Рассчитать режим | Сохранить результат | Построить график

Таблица узлов:

№ узла	Название	Тип	Уном, кВ	P, МВт	Q, Мвар	U, кВ	delta, град
1	Узел №1	False	110.0	-5.537727385	-7.527374517	110.0	0.0
2	Узел №2	True	110.0	0.0	0.0	109.00626096	-0.089608588
3	Узел №3	True	110.0	1.0	3.0	108.69676506	-0.078643234
4	Узел №4	True	110.0	2.0	4.0	108.24449008	-0.161173255
5	Узел №5	True	10.0	2.0	1.0	9.7286723777	-0.244417619

Таблица ветвей:

№ нач.	№ кон.	R, Ом	X, Ом	G, мкСм	B, мкСм	KTR
1	2	4.95	10.5	10.0	-50.0	0.0
2	3	4.95	10.5	10.0	-50.0	0.0
2	4	4.95	10.5	10.0	-50.0	0.0
4	5	3.0	10.0	10.0	100.0	0.09

Обработка кнопки «Сохранить результат» будет выглядеть следующим образом:

```

308     if event=='-SAVE-':
309         file=sg.popup_get_file("Выберите файл",no_window=True,save_as=True)
310         if file!='':
311             s.save_scheme(file)
312

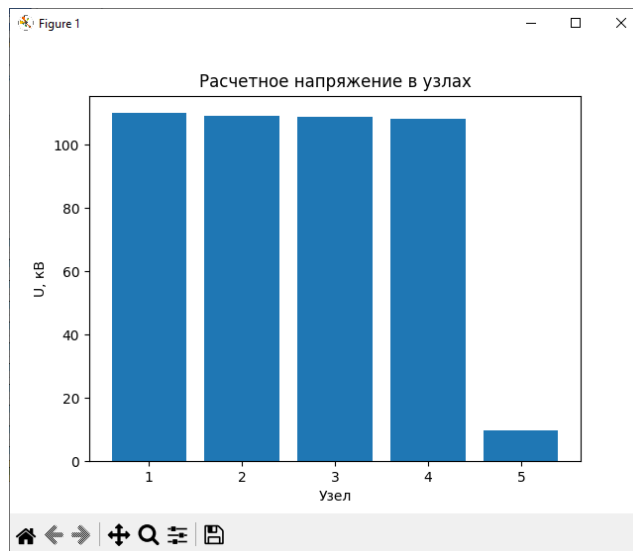
```

И, наконец, кнопка «Построить график» будет строить столбчатую диаграмму по каждому узлу сети:

```

313     if event=='-GRAPH-':
314         x=[]
315         y=[]
316         for i in s.nodes:
317             x.append(i.ny)
318             y.append(abs(i.v))
319
320         plt.title("Расчетное напряжение в узлах")
321         plt.xlabel("Узел")
322         plt.ylabel("U, кВ")
323         plt.bar(x, y)
324         plt.show()
325

```



Задание для выполнения

Разработайте программу с графическим интерфейсом, которая будет позволять выводить на экран:

- диаграмму, отражающую нагрузочные потери в каждой линии сети;
- диаграмму, отражающую нагрузочные потери в каждом трансформаторе сети;
- диаграмму, отражающую потери холостого хода в каждой линии сети;
- диаграмму, отражающую потери холостого хода в каждом трансформаторе сети;
- столбчатую диаграмму, отражающую падение напряжения по каждой линии;
- круговую диаграмму, отражающую структуру потерь электроэнергии в сети.

Контрольные вопросы

1. Что необходимо сделать, чтобы использовать в своих приложениях библиотеку PySimpleGUI?
2. Какие элементы библиотеки PySimpleGUI вы знаете? Как их создавать?

3. Какой функцией создает графическое окно приложения? Какие параметры имеет эта функция?
4. Как обрабатывать действия пользователя в библиотеке PySimpleGUI?

Список рекомендуемой литературы

Основная литература:

1. Седер, Н. Python. Экспресс-курс : практическое руководство / Н. Седер. - Санкт-Петербург : Питер, 2021. - 480 с. - (Серия «Библиотека программиста»). - ISBN 978-5-4461-0908-1. - Текст : электронный. - URL: <https://znanium.com/catalog/product/1756159> (дата обращения: 09.11.2022). – Режим доступа: по подписке.

2. Применение цифровых вычислительных машин в электроэнергетике : учеб. пособие / [О. В. Щербачев, А. Н. Зейлигер, К. П. Кадомская и др.] ; под ред. О. В. Щербачева. - Л. : Энергия, 1980. - 240 с. : ил. - Библиогр.: с. 229-232. - Предм. указ.: с. 233-234

3. Электрические системы. : Математические задачи электроэнергетики : учебник для вузов / [В. А. Веников, Э. Н. Зуев, И. В. Литкенс и др.] ; под ред. В. А. Веникова. - Изд. 2-е, перераб. и доп. - М. : Высшая школа, 1981. - 288 с. : ил. - Гриф: Доп. МО для электроэнергетических спец. - Библиогр: с. 285-286.

4. Дейтел, П. Python: Искусственный интеллект, большие данные и облачные вычисления: практическое руководство / П. Дейтел, Х. Дейтел. - Санкт-Петербург : Питер, 2020. - 864 с. - (Серия «Для профессионалов»). - ISBN 978-5-4461-1432-0. - Текст : электронный. - URL: <https://znanium.com/catalog/product/1733685> (дата обращения: 09.11.2022). – Режим доступа: по подписке.

Дополнительная литература:

5. Идельчик, В. И. Расчеты и оптимизация режимов электрических сетей и систем / В. И. Идельчик. - М. : Энергоатомиздат, 1988. - 288 с. : ил. - Библиогр.: с. 281-285. - ISBN 5-283-01103-8

6. Герасименко, А. А. Передача и распределение электрической энергии : учеб. пособие / А. А. Герасименко, В. Т. Федин. – Ростов н/Д : Феникс ; Красноярск : Издательские проекты, 2006. – 720 с. : ил. – (Высшее образование). – Библиогр.: с. 667-671. – ISBN 5-222-08485-X (Феникс). – ISBN 5-98399-023-3 (Издательские проекты).

7. Чан, Д. Python: быстрый старт : практическое руководство / Д. Чан. - Санкт-Петербург: Питер, 2021. - 224 с. - (Серия «Библиотека программиста»). - ISBN 978-5-4461-1800-7. - Текст: электронный. - URL: <https://znanium.com/catalog/product/1739605> (дата обращения: 09.11.2022). – Режим доступа: по подписке.

Учебно-методическая литература:

8. Алгоритмы задач электроэнергетики. Методические указания по выполнения практических работ. Направление подготовки 13.03.02. Направленность (профиль) «Электроэнергетические системы и сети». /сост. Зеленский Е.Г.

9. Алгоритмы задач электроэнергетики. Методические указания по самостоятельной работе студентов. Направление подготовки 13.03.02. Направленность (профиль) «Электроэнергетические системы и сети». /сост. Зеленский Е.Г.

Интернет-ресурсы:

10. <https://www.python.org/>

11. <https://www.codabrainy.com/>

12. <https://znanium.com/>

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания

по организации самостоятельной работы обучающихся
по дисциплине «Алгоритмы задач электроэнергетики»
для студентов направления подготовки /специальности
13.03.02 Электроэнергетика и электротехника
Направленность (профиль) Электроэнергетические системы и сети

Ставрополь 2025

СОДЕРЖАНИЕ

<u>Введение</u>	153
<u>1 Общая характеристика самостоятельной работы студента при изучении дисциплины</u>	154
<u>2 План-график выполнения самостоятельной работы</u>	157
<u>3 Контрольные точки и виды отчетности по ним</u>	158
<u>4 Методические рекомендации по изучению теоретического материала</u>	159
<u>5 Методические указания по подготовке к практическим занятиям</u>	167
<u>6 Методические рекомендации по выполнению расчетно-графической работы</u>	168
<u>Список рекомендуемой литературы</u>	171

Введение

Самостоятельная работа является внеаудиторной и предназначена для самостоятельного изучения определенных тем дисциплины по материалам, рекомендованным преподавателем, а также в порядке подготовки к практическим занятиям.

Цель дисциплины «Алгоритмы задач электроэнергетики» – формирование согласно ООП ВО набора профессиональных компетенций будущего бакалавра по направлению 13.03.02 «Электроэнергетика и электротехника».

Для достижения этой цели в процессе изучения дисциплины решаются следующие задачи: ознакомление с основными понятиями и терминами изучаемой дисциплины, которыми будущий специалист будет оперировать в своей практической деятельности; освоение базовых алгоритмов, применяемых для решения задач электроэнергетики.

1 Общая характеристика самостоятельной работы студента при изучении дисциплины

Основная задача высшего образования заключается в формировании творческой личности специалиста, способного к саморазвитию, самообразованию, инновационной деятельности. Решение этой задачи вряд ли возможно только путем передачи знаний в готовом виде от преподавателя к студенту. Для решения этой задачи в учебные планы всех специальностей включена самостоятельная работа, составляющая 50% учебного времени студента.

В широком смысле под самостоятельной работой студентов следует понимать совокупность всей самостоятельной деятельности студентов, как в учебной аудитории, так и вне ее, в контакте с преподавателем и в его отсутствие.

В учебном процессе выделяют два вида самостоятельной работы:

- аудиторная;
- внеаудиторная.

Аудиторная самостоятельная работа по дисциплине выполняется на учебных занятиях под непосредственным руководством преподавателя и по его заданию.

Внеаудиторная самостоятельная работа выполняется студентом по заданию преподавателя, но без его непосредственного участия.

Содержание внеаудиторной самостоятельной работы определяется в соответствии с рекомендуемыми видами заданий согласно примерной и рабочей программ учебной дисциплины. Видами заданий для внеаудиторной самостоятельной работы по дисциплине «Алгоритмы задач электроэнергетики» являются:

- для овладения знаниями: чтение текста (учебника, первоисточника, дополнительной литературы), составление плана текста, графическое

изображение структуры текста, конспектирование текста, выписки из текста, работа со словарями и справочниками, ознакомление с нормативными документами, учебно-исследовательская работа, использование аудио- и видеозаписей, компьютерной техники и Интернета и др.;

– для закрепления и систематизации знаний: обработка текста, повторная работа над учебным материалом (учебника, первоисточника, дополнительной литературы, аудио и видеозаписей, составление плана, составление таблиц для систематизации учебного материала, ответ на контрольные вопросы, заполнение рабочей тетради, аналитическая обработка текста (аннотирование, рецензирование, реферирование, конспект-анализ и др.);

– для закрепления навыков – подготовка к практическим работам, оформление отчетов по практическим работам, выполнение дополнительных заданий, выданных преподавателем.

Самостоятельная работа может осуществляться индивидуально или группами студентов в зависимости от цели, объема, конкретной тематики самостоятельной работы, уровня сложности, уровня умений студентов. Контроль результатов внеаудиторной самостоятельной работы студентов может осуществляться в пределах времени, отведенного на обязательные учебные занятия по дисциплине и внеаудиторную самостоятельную работу студентов по дисциплине, может проходить в письменной, устной или смешанной форме.

Следует с первых дней обучения в вузе приучать себя к целесообразному распределению занятий по месяцам и неделям семестра. При этом очень важна равномерная работа как основное условие успешных занятий в вузе.

Высшей ступенью обучения является приобретение навыков самостоятельной творческой работы, а творческому применению знаний нужно учиться параллельно с их приобретением.

Задачами самостоятельной работы студентов по дисциплине «Алгоритмы задач электроэнергетики» являются:

- формирования готовности студентов к поиску, обработке и применению информации для решения профессиональных задач;
- систематизация и закрепление полученных теоретических знаний и практических умений студентов;
- развитие познавательных способностей и активности студентов, творческой инициативы, самостоятельности, ответственности и организованности;
- формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- выработка навыков эффективной самостоятельной профессиональной деятельности.

2 План-график выполнения самостоятельной работы

Общий объём самостоятельной работы в рамках изучения дисциплины «Алгоритмы задач электроэнергетики» составляет 45 астрономических часа.

В структуре самостоятельной работы предусмотрено самостоятельное изучение теоретического материала, подготовка к практическим занятиям, выполнение расчетно-графической работы.

В рамках самостоятельной работы предусмотрена подготовка студента к проведению и защите практических работ. В рамках изучения дисциплины студент должен выполнить следующие практические работы:

- Знакомство со средой программирования IDLE
- Работа со списками
- Функции в Python
- Файлы, строки
- Объектно-ориентированное программирование
- Матрицы и операции над ними, решение систем линейных уравнений
- Метод Зейделя
- Метод Ньютона
- Метод "в два этапа"
- Расчет потерь электроэнергии
- Визуализация данных
- Визуализация данных, дополнительные возможности

3 Контрольные точки и виды отчетности по ним

Успеваемость студентов по дисциплине оценивается в ходе текущего и промежуточного контроля. Текущий контроль осуществляется в течение семестра на практических и лекционных занятиях. В начале семестра студенческая группа получает график контрольных мероприятий с указанием форм и сроков их проведения, с указанием баллов за каждое контрольное мероприятие. Контрольные точки по самостоятельной работе приведены в Таблице 3.1.

Таблица 3.1 – Контрольные точки по самостоятельной работе

№ п/п	Вид деятельности студентов	Сроки выполнения	Количество баллов
8 семестр			
1	Практическое занятие 4	4	10
2	Практическое занятие 8	8	10
3	Практическое занятие 12	12	10
4	Лекция 12	12	5
5	Расчетно-графическая работа	12	20
	Итого за 8 семестр:		55
	Итого:		55

В начале семестра каждому студенту выдается индивидуальное задание для выполнения Расчетно-графической работы. В конце семестра происходит защита выполненной работы. Защита работы проходит в форме доклада студента по выполненной работе и ответов на вопросы преподавателя.

Максимальное количество баллов студент получает, если оформление работы соответствует установленным требованиям, студент может правильно ответить на контрольные вопросы, а отчет по работе полностью раскрывает суть работы. Основанием для снижения оценки являются:

- неполное выполнение задания по расчетно-графической работе;
- некорректность результатов и выводов;
- затруднение в ответах на контрольные вопросы по расчетно-графической работе.

4 Методические рекомендации по изучению теоретического материала

Большая часть самостоятельной работы студента состоит в изучении литературы. Одна из задач студента – научиться самостоятельно работать с книгой, а это требует определенных затрат энергии и времени. Самостоятельная работа с книгой может быть успешной, если текст не только прочитан, но и законспектирован. Существует несколько форм записей, но любая форма записи не даст нужного результата, если не будет пробуждать мысли того, кто ее ведет, если отсутствует активная работа ума и формирование своих выводов из прочитанного.

Выбор формы записи зависит от индивидуальных особенностей человека, его образованности и опыта. При этом не меньшую роль играет назначение записей, то есть то, какие задачи ставит перед собой человек (для самообразования, для выступления на семинаре, для использования в будущем).

Введение записей мобилизует наряду со зрительной памятью, также и моторную память. Кроме того, у человека, систематически ведущего записи изучаемой литературы, создается свой фонд материалов для быстрого повторения и мобилизации накопленных знаний.

Все записи должны быть убористыми и компактными. Интервалы между строками должны быть достаточными, чтобы вписывать дополнения. Рекомендуется вести записи ручкой, а карандашом или ручкой другого цвета пользоваться для отметок и выделений при последующей работе. Полезно также датировать записи.

Записи могут носить различный характер: план, выписки, тезисы, аннотирование, конспектирование.

1. План - наиболее краткая формой записи. Это перечень вопросов, рассматриваемых в книге или статье. План обычно раскрывает структуру

произведения, логику автора, способствует лучшей ориентации в содержании.

Так, составленным планом можно воспользоваться, чтобы вспомнить прочитанное или быстро отыскать в книге нужное место. Представление об основных пунктах плана дает оглавление книги, поэтому во многих случаях наименования глав и разделов можно использовать в качестве пунктов. Составление плана приучает логически мыслить, вырабатывать умение сжато и последовательно излагать суть вопроса в письменной и устной форме.

Существует два способа составления плана: работа над ним по ходу чтения и составление плана после ознакомления с произведением. При этом план получается более последовательным и стройным.

Трудность составления плана состоит в том, что надо выяснить для себя, прежде всего, построение изучаемого текста, ход мыслей автора и лишь затем изложить содержание работы кратко и ясно. При всей своей краткости план дает представление о содержании прочитанного. Следует учесть, что форма плана не исключает цитирования отдельных мест и обобщений. Различают простой и развернутый план. В отличие от простого плана развернутый план не только содержит перечисление вопросов, но и раскрывает основные идеи произведения, может включать выдержки из него, схемы, таблицы. Планом, особенно развернутым, необходимо пользоваться при написании выступления или статьи.

В целом развернутый план дает гораздо большее представление о произведении, его основных идеях, задачах, которые в нем решаются. Он может включать положения, замечания, собственные мысли студента.

Важно знать, что составление планов помогает вырабатывать способность к отвлеченному, абстрактному мышлению, но наибольшую пользу составление плана даст подготовленным лицам, которые бывают достаточно лишь взглянуть на перечень основных вопросов, чтобы воспроизвести содержание прочитанного.

2. Тезисы – более сложная и совершенная форма записи, чем составление плана.

Это сжатое изложение основных мыслей прочитанного произведения или подготовляемого вступления. Особенностью тезисов является их утвердительный характер.

В них сосредотачивается самое главное, только выводы и обобщения, в них меньше доказательств, иллюстрации и пояснений. Тезисы не должны повторять дословно текст, но в ряде мест могут быть близки к нему, воспроизводя некоторые характерные выражения автора, важные для понимания хода его мыслей. Составление тезисов помогает глубже понять основные идеи произведения, выделить главное в нем; приучают сжато, точно и четко сформулировать свои мысли, повышает культуру речи и письма. При составлении тезисов учитывают следующее. Прежде всего, если произведение небольшое, необходимо внимательно изучать его в целом, если большое – изучать по главам и разделам. Затем, когда будут ясны основные идеи, кратко и последовательно излагать их в виде пунктов.

Различают простые и сложные, развернутые тезисы. Если записывают только утверждение чего – либо, такой тезис называют простым, а сложным тезисом будет выражение главной мысли, содержащее, кроме утверждения, еще и краткое ее доказательство.

Часто тезисы формулируются самим автором как выводы и обобщения в заключении книги или разделах книги. Нередко тезисы выделяются в тексте другим шрифтом.

Рекомендуется делать тезисные записи своими словами, причем можно записывать один абзац за другим, учитывая смысловую связь между ними. Но в большинстве случаев следует составлять сводный тезис, сложный по форме. При этом объединяется несколько утверждений, тесно связанных между собой.

Тезисы по содержанию очень близки к конспекту, но конспект носит более описательный характер, и его положения не столь категоричны, как в тезисах. Кроме того, конспект представляет собой более полную форму записи.

Следует отметить, что различие между формами записей условно, но в любой форме запись – важнейшая часть самостоятельной работы с книгой.

3. Выписки. Это записи текста из книги: теоретических положений, статистических данных, имеющих по мнению читателя важное значение.

Достоинство выписок состоит в точности воспроизведения текста книги, удобстве пользования записями при последующей работе, в накоплении обобщений и фактического материала. Выписки полезны для повторения, освежения в памяти прочитанного, для быстрой мобилизации своих знаний, когда необходимо в короткий срок вспомнить материал. Выписки выделяют из текста самое главное и тем самым помогают глубже понять его. Без них трудно обойтись при подготовке доклада, реферата, выступления. Выписки следует рассматривать как составную часть тезисов и конспектов.

Выписывать текст можно и по ходу чтения и после его завершения. В последнем случае надо замечать места, которые потом будут выписаны. Необходимо каждую выписку снабжать ссылкой на источник с указанием соответствующей страницы. Это нужно, чтобы в последствии можно было быстро найти в книге соответствующее место. Целесообразно выписывать из текста только такие места, в которых содержится самое главное, суть вопроса. Выписки должны быть ориентированы на изучение произведения в целом, а не отдельных мест, поскольку положения, вырванные из общего контакта, понимаются нередко совсем не так, как этого хотел автор. Иначе говоря, отдельно взятые, лишённые пояснений выдержки могут быть не поняты или поняты неправильно.

Выписки бывают дословные (цитаты) и «свободные», когда мысли автора излагаются своими словами. Следует учесть, что большие отрывки, которые трудно цитировать, целесообразнее в краткой форме переложить своими словами, но «яркие» и важные места лучше выписывать дословно. Каждую цитату следует заключать в кавычки. Если ее берут из середины предложения, то после вводных кавычек ставят три точки. Ставят их и в конце цитаты, если из предложения опущены последние слова.

Следует знать, что какого-либо единого метода выписок, годного для всех случаев, не существует, поскольку у каждого человека свои особенности мышления и восприятия, свой подход к теме. Все это влияет на содержание и характер выписок.

Выписки рекомендуется хранить в картотеке, конвертах или папках, на которых следует обозначить общую тему.

4. Аннотация – еще одна форма записи, являющаяся кратким обобщением содержания книги. Ею удобно пользоваться, если имеется намерение вернуться к изучаемому произведению. Аннотация может быть необходима и для того, чтобы не забыть о нем.

Для составления аннотации надо сначала полностью прочитать и глубоко продумать произведение. При всей своей краткости аннотация может содержать отдельные фрагменты авторского текста, а не только оценку книги или статьи.

5. Резюме очень близко к аннотации. Это запись, являющаяся краткой оценкой прочитанного материала. Различие между ними состоит в том, что аннотация сжато характеризует произведение в целом, а резюме концентрирует внимание на его выводах, главных итогах.

6. Конспект – наиболее совершенная и наиболее сложная форма записи. Слово «конспект» происходит от латинского «conspectus», что означает «обзор, изложение». В правильно составленном конспекте обычно выделено самое основное в изучаемом тексте, сосредоточено внимание на

наиболее существенном, в кратких и четких формулировках обобщены важные теоретические положения.

Конспект представляет собой относительно подробное, последовательное изложение содержания прочитанного. На первых порах целесообразно в записях ближе держаться тексту, прибегая зачастую к прямому цитированию автора. В дальнейшем, по мере выработки навыков конспектирования, записи будут носить более свободный и сжатый характер.

Конспект книги обычно ведется в тетради. В самом начале конспекта указывается фамилия автора, полное название произведения, издательство, год и место издания. При цитировании обязательная ссылка на страницу книги. Если цитата взята из собрания сочинений, то необходимо указать соответствующий том. Следует помнить, что четкая ссылка на источник – непереносимое правило конспектирования. Если конспектируется статья, то указывается, где и когда она была напечатана.

Конспект подразделяется на части в соответствии с заранее продуманным планом. Пункты плана записываются в тексте или на полях конспекта. Писать его рекомендуется четко и разборчиво, так как небрежная запись с течением времени становится малопонятной для ее автора. Существует правило: конспект, составленный для себя, должен быть по возможности написан так, чтобы его легко прочитал и кто-либо другой.

Формы конспекта могут быть разными и зависят от его целевого назначения (изучение материала в целом или под определенным углом зрения, подготовка к докладу, выступлению на занятии и т.д.), а также от характера произведения (монография, статья, документ и т.п.). Если речь идет просто об изложении содержания работы, текст конспекта может быть сплошным, с выделением особо важных положений подчеркиванием или различными значками.

В случае, когда не ограничиваются переложением содержания, а фиксируют в конспекте и свои собственные суждения по данному вопросу

или дополняют конспект соответствующими материалами и других источников, следует отводить место для такого рода записей. Рекомендуется разделить страницы тетради пополам по вертикали и в левой части вести конспект произведения, а в правой свои дополнительные записи, совмещая их по содержанию.

Конспектирование в большей мере, чем другие виды записей, помогает вырабатывать навыки правильного изложения в письменной форме важные теоретических и практических вопросов, умение четко их формулировать и ясно излагать своими словами.

Таким образом, составление конспекта требует вдумчивой работы, затраты времени и труда. Зато во время конспектирования приобретаются знания, создается фонд записей.

Конспект может быть текстуальным или тематическим. В текстуальном конспекте сохраняется логика и структура изучаемого произведения, а запись ведется в соответствии с расположением материала в книге. За основу тематического конспекта берется не план произведения, а содержание какой-либо темы или проблемы.

Текстуальный конспект желательно начинать после того, как вся книга прочитана и продумана, но это, к сожалению, не всегда возможно. В первую очередь необходимо составить план произведения письменно или мысленно, поскольку в соответствии с этим планом строится дальнейшая работа. Конспект включает в себя тезисы, которые составляют его основу. Но, в отличие от тезисов, конспект содержит краткую запись не только выводов, но и доказательств, вплоть до фактического материала. Иначе говоря, конспект – это расширенные тезисы, дополненные рассуждениями и доказательствами, мыслями и соображениями составителя записи.

Как правило, конспект включает в себя и выписки, но в него могут войти отдельные места, цитируемые дословно, а также факты, примеры, цифры, таблицы и схемы, взятые из книги. Следует помнить, что работа над

конспектом только тогда будет творческой, когда она не ограничена текстом изучаемого произведения. Нужно дополнять конспект данными из другими источников.

В конспекте необходимо выделять отдельные места текста в зависимости от их значимости. Можно пользоваться различными способами: подчеркиваниями, вопросительными и восклицательными знаками, репликами, краткими оценками, писать на полях своих конспектов слова: «важно», «очень важно», «верно», «характерно».

В конспект могут помещаться диаграммы, схемы, таблицы, которые придадут ему наглядность.

Составлению тематического конспекта предшествует тщательное изучение всей литературы, подобранной для раскрытия данной темы. Бывает, что какая-либо тема рассматривается в нескольких главах или в разных местах книги. А в конспекте весь материал, относящийся к теме, будет сосредоточен в одном месте. В плане конспекта рекомендуется делать пометки, к каким источникам (вплоть до страницы) придется обратиться для раскрытия вопросов. Тематический конспект составляется обычно для того, чтобы глубже изучить определенный вопрос, подготовиться к докладу, лекции или выступлению на семинарском занятии. Такой конспект по содержанию приближается к реферату, докладу по избранной теме, особенно если включает и собственный вклад в изучение проблемы.

5 Методические указания по подготовке к практическим занятиям

В рамках самостоятельной работы предусмотрена подготовка студента к практическим работам. Практические занятия позволяют интегрировать теоретические знания и формировать практические умения и навыки студентов в процессе учебной деятельности.

Цели практических занятий по дисциплине «Алгоритмы задач электроэнергетики»:

- 1) закрепление теоретического материала путем систематического контроля за самостоятельной работой студентов;
- 2) формирование умений использования теоретических знаний в процессе выполнения практических работ;
- 3) развитие аналитического мышления путем обобщения результатов практических работ;
- 4) формирование навыков оформления результатов практических работ в виде таблиц, графиков, выводов.

На практических занятиях осуществляются следующие формы работ со студентами: индивидуальная (оценка знаний, выполненных тестовых заданий, проверка рабочих тетрадей); групповая (выполнение заданий малыми группами по 2-4 человека); фронтальная (подведение итогов выполнения практических работ).

6 Методические рекомендации по выполнению расчетно-графической работы

Расчетно-графическая работа является важнейшим элементом самостоятельной работы студентов. Основной целью расчетно-графической работы является получение будущим специалистом практических навыков по решению комплексных задач, возникающих в практической деятельности, а также создание и развитие навыков исследовательской работы, умения работать с научной литературой, делать на основе ее изучения выводы и обобщения,

Основными задачами расчетно-графической работы являются:

- 1) овладение первичными навыками ведения исследовательской работы; развитие творческих способностей индивидуально для каждого студента;
- 2) подготовка студента к выполнению выпускной-квалификационной работы;
- 3) усвоение методов грамотного ведения, оформления и редактирования деловой переписки, а также выполнения практической аналитической работы.

Умение вести исследование – подбирать, анализировать, обобщать материал, системно излагать его научным стилем, обосновывать выводы, оформлять работу – отличает специалиста с высшим образованием. Расчетно-графические работы последовательно готовят выпускника, наращивая владение элементами исследовательской работы.

В соответствии с целью и задачами назначение расчетно-графической работы в учебном процессе конкретизируются в овладении студентами следующих знаний и навыков:

- а) работа с библиографией: пользование каталогами и справочной литературой, статистическими и инструктивными материалами;

- б) разработки плана работы;
- в) понимания и грамотного написания введения к любой исследовательской работе;
- г) методики и стиля изложения материалов работы;
- д) редакционного оформления работы в соответствии с общеустановленными требованиями;
- е) написание заключения, уяснение его назначения в работе;
- ж) составление списка использованной литературы;
- и) назначение приложений и их оформление.

Безусловно, овладение этими знаниями должно происходить постепенно, от курса к курсу, от предыдущей работы к каждой последующей круг требований должен расширяться, а их уровень возрастать.

Обязательным является наличие табличных, цифровых и графических данных; проблемно-поисковый характер работы, овладение методами доказательности приводимого материала, обоснование своей точки зрения и путей решения проблем.

Порядок работы над расчетно-графической работой включает следующие этапы:

- выбор варианта работы и беседа с руководителем;
- сбор материала, поиск литературы по теме, составление личного рабочего плана;
- выполнение расчетной части работы;
- подготовка графической части работы;
- оформление первого варианта работы;
- сдача первого варианта расчетно-графической работы руководителю;
- доработка текста по замечаниям, его окончательное оформление и защита работы в виде ответов на дополнительные вопросы руководителя.

Работа над расчетно-графической работой не должна откладываться на последние дни. Относиться к ней надлежит со всей ответственностью и

добросовестностью. Только систематический, правильно спланированный и организованный труд позволит добиться хорошего результата точно к установленному сроку

После завершения окончательного варианта работы руководитель готовит свое заключение и выставляет оценку.

Основные критерии оценки расчетно-графической работы вытекают из предъявляемых к ней требований. Такими критериями являются:

1. Глубина анализа, умение разобраться в затронутых проблемах.
2. Самостоятельность, творческий подход к рассматриваемой проблеме.
3. Полнота решения всех тех задач, которые были поставлены перед студентом.
4. Грамотность, логичность в изложении материала
5. Качество оформления.

Каждая расчетно-графическая работа с учетом ее содержания оценивается по пятибалльной системе. Работу, которую преподаватель признал неудовлетворительной, возвращается для переработки с учетом высказанных в отзыве замечаний.

Список рекомендуемой литературы

Основная литература:

1. Седер, Н. Python. Экспресс-курс : практическое руководство / Н. Седер. - Санкт-Петербург: Питер, 2021. - 480 с. - (Серия «Библиотека программиста»). - ISBN 978-5-4461-0908-1. - Текст :электронный. - URL: <https://znanium.com/catalog/product/1756159> (дата обращения: 09.11.2022). – Режим доступа: по подписке.

2. Электрические системы: Математические задачи электроэнергетики : учебник для вузов / [В. А. Веников, Э. Н. Зуев, И. В. Литкенс и др.] ; под ред. В. А. Веникова. - Изд. 2-е, перераб. и доп. - М. : Высшая школа, 1981. - 288 с. : ил. - Гриф: Доп. МО для электроэнергетических спец. - Библиогр: с. 285-286.

3. Дейтел, П. Python: Искусственный интеллект, большие данные и облачные вычисления: практическое руководство / П. Дейтел, Х. Дейтел. - Санкт-Петербург : Питер, 2020. - 864 с. - (Серия «Для профессионалов»). - ISBN 978-5-4461-1432-0. - Текст : электронный. - URL: <https://znanium.com/catalog/product/1733685> (дата обращения: 09.11.2022). – Режим доступа: по подписке.

Дополнительная литература:

4. Герасименко, А. А. Передача и распределение электрической энергии : учеб. пособие / А. А. Герасименко, В. Т. Федин. – Ростов н/Д : Феникс ; Красноярск : Издательские проекты, 2006. – 720 с. : ил. – (Высшее образование). – Библиогр.: с. 667-671. – ISBN 5-222-08485-X (Феникс). – ISBN 5-98399-023-3 (Издательские проекты).

5. Чан, Д. Python: быстрый старт : практическое руководство / Д. Чан. - Санкт-Петербург: Питер, 2021. - 224 с. - (Серия «Библиотека программиста»). - ISBN 978-5-4461-1800-7. - Текст: электронный. - URL: <https://znanium.com/catalog/product/1739605> (дата обращения: 09.11.2022). –

Режим доступа: по подписке.

Учебно-методическая литература:

6. Алгоритмы задач электроэнергетики. Методические указания по выполнению практических работ. Направление подготовки 13.03.02. Направленность (профиль) «Электроэнергетические системы и сети». /сост. Зеленский Е.Г.

7. Алгоритмы задач электроэнергетики. Методические указания по выполнению расчетно-графической работы. Направление подготовки 13.03.02. Направленность (профиль) «Электроэнергетические системы и сети». /сост. Зеленский Е.Г.

Интернет-ресурсы:

8. <https://www.python.org/>

9. <https://www.codabrainy.com/>

10. <https://znanium.com/>

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

**по выполнению расчетно-графической работы
по дисциплине
«Алгоритмы задач электроэнергетики»**

Направление подготовки 13.03.02 «Электроэнергетика и электротехника»
Направленность (профиль):
«Электроэнергетические системы и сети»
Форма обучения: очная

Ставрополь, 2025

СОДЕРЖАНИЕ

<u>1 Структура расчетно-графической работы</u>	175
<u>2 Требования к оформлению расчетно-графической работы</u>	178
<u>3 Контрольные сроки представления отдельных разделов расчетно-графической работы</u>	181
<u>4 Варианты заданий</u>	182
<u>Список рекомендуемой литературы</u>	150

1 Структура расчетно-графической работы

Расчетно-графическая работа должна включать в себя следующие разделы:

- Введение
- Описание метода расчета режима
- Разработка алгоритма расчета режима
- Разработка диаграммы классов
- Разработка программы расчета режима
- Описание тестовой схемы
- Результаты тестовых расчетов
- Сравнение тестовых расчетов с программным комплексом RastrWin
- Заключение

Во введение приводится краткое описание актуальности задачи расчета режимов, перечисляются существующие методы расчета режимов, приводится их краткая характеристика. Во введении также описывается структура расчетно-графической работы, с указанием название разделов и их содержания.

В разделе «Описание метода расчета режима» приводится математическая формулировка задачи расчета режима с учетом заданных схем замещения, подробно рассматривается один из методов ее решения с указанием его характерных особенностей по сравнению с другими методами, существующие ограничения применения этого метода, его вычислительная сложность и точность.

В разделе «Разработка алгоритма расчета режима» приводится блок-схема алгоритма заданного метода расчета режима с кратким или подробным описанием каждого из этапов алгоритма.

В разделе «Разработка диаграммы классов» приводится перечень программных классов, которые были разработаны при реализации алгоритма

расчета режима, их диаграмма в виде отдельного рисунка (пример – рисунок 1) с указанием полей и методов класса и подробная спецификация класса с указанием назначения каждого поля и метода класса (пример – таблица 1).

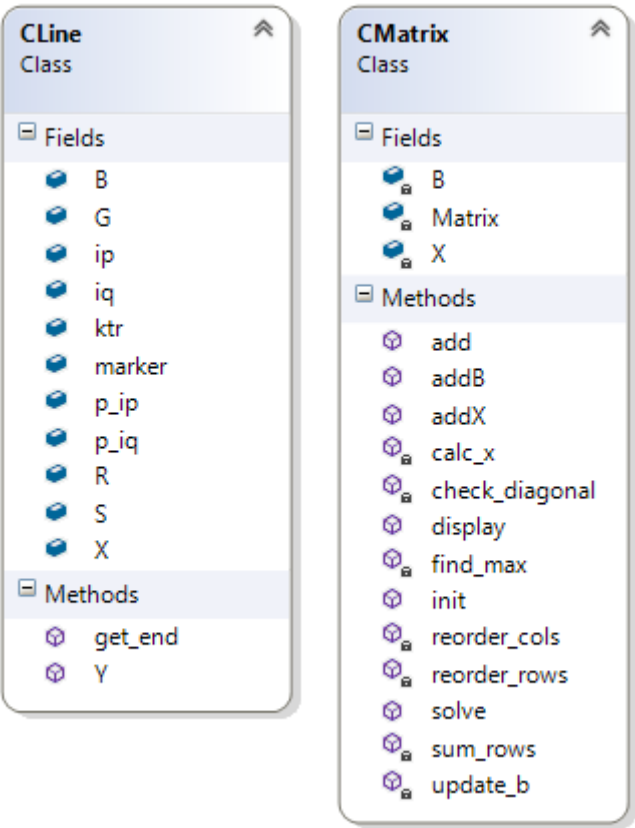


Рисунок 1 – Пример диаграммы классов

Таблица 1 – Пример спецификации класса

Имя класса	Имя свойства	Имя метода	Назначение
CLine	B		Переменная для хранения реактивной проводимости ветви
-	-	-	-
-	-	-	-
-	-	-	-

В разделе «Разработка программы расчета режима» приводится краткое описание разработанного приложения с указанием формата входных и

выходных данных, перечня параметров командной строки. Здесь же приводится листинг программного кода разработанного приложения на языке Python.

В разделе «Описание тестовой схемы» приводится описание схемы, на которой выполнялось тестирование разработанного приложения. Приводится графическое изображение схемы, отражающее ее топологию и табличный эквивалент, содержащий подробное описание параметров элементов этой схемы.

В разделе «Результаты тестовых расчетов» приводятся копии экрана, на которых изображены результаты работы программы. На них должны быть отчетливо видны расчетные узловые напряжения.

В разделе «Сравнение тестовых расчетов с программным комплексом RastrWin» приводятся результаты расчетов тестовой схемы в промышленном программном комплексе «RastrWin» и сравнение полученных результатов с результатами расчета в разработанной программе. При расхождении результатов делается вывод о возможных причинах полученных результатов.

В «Заключении» приводятся краткие результаты о проделанной работе.

2 Требования к оформлению расчетно-графической работы

Объем расчетно-графической работы должен составлять 20-30 страниц формата А4. Расчетно-графическая работа выполняется в течение всего семестра и заканчивается его защитой на последнем практическом занятии.

Текст расчетно-графической работы выполняют с использованием компьютера на одной стороне листа белой бумаги, формата А4, шрифт – Times New Roman 14-го размера, межстрочный интервал – 1,5. Поля: в начале строк – 30 мм; в конце строк – 15 мм; от верхней и нижней строки текста – 20 мм. Автоматический перенос слов – запрещен.

Номера страниц проставляют в правом нижнем углу листа, на титульном листе номер не ставится, но он участвует в нумерации.

Размер абзацного отступа 12,5 мм (1,25 см).

Перечисления записывают строчными буквами, в конце каждого ставят точку с запятой, а в конце последнего – точку.

Например:

К техническим условиям относят:

- а) ;
- б) :
 - 1) ;
 - 2) .

На предприятии преобладают одноэтажные здания, такие как:

- компрессорная, котельная и литейный цех;
- основные производственные цеха.

Рисунки оформляются по центру, подписываются снизу.

Например:



Рисунок 1 – Расчетная схема электроснабжения

На каждый рисунок в тексте должны быть ссылки (упоминание рисунка тексте).

Таблицы выравниваются по центру, подписываются над таблицей с левого края листа без абзацного отступа.

Оформление формул.

Если формула без расчетов, то она выравнивается по центру

Например:

$$I_{\max p} = \frac{S_p}{\sqrt{3} \cdot U_n}, \quad (1.37)$$

где S_p – расчетная мощность линии, кВА;

n – число линий;

U_n – номинальное напряжение сети, кВ.

Если формула с расчетами, то выравнивается по левому краю с абзацного отступа 1,25 см.

Например:

Отсюда экономически целесообразное сечение

$$F_{\text{ЭК}} = \frac{I_{\text{РН}}}{j_{\text{ЭК}}} = \frac{77,26}{1,7} = 45,45 \text{ мм}^2.$$

Полученное сечение округляем до ближайшего стандартного $F_{\text{табл.}} = 50 \text{ мм}^2$.

Формулы, следующие одна за другой и не разделенные текстом, разделяют запятой. Переносить формулы на следующую строку допускается только на знаках выполняемых операций, при этом знак в начале следующей строки повторяют. При переносе формулы на знаке умножения применяют знак «х». На знаке деления переносы не допускаются.

Например:

$$\begin{aligned} S_{\text{ПЗ}} &= \sqrt{(P_3 + P_7 + P_{11})^2 + (Q_3 + Q_7 + Q_{11})^2} = \\ &= \sqrt{(872,2 + 490,08 + 37,5)^2 + (837,66 + 363,02 + 28,53)^2} = \\ &= \sqrt{3470341,27} = 1862,89 \text{ кВА} \end{aligned}$$

3 Контрольные сроки представления отдельных разделов расчетно-графической работы

Расчетно-графическая работа делится на 9 этапов. Контрольные сроки выполнения отдельных этапов расчетно-графической работы приведены в таблице 2.

Таблица 2 – Контрольные сроки выполнения отдельных этапов расчетно-графической работы

Этап	Наименование раздела (объем в %)	Плановый срок выполнения
1	Выбор тестовой схемы, расчет эталонного режима в RastrWin (10%)	1 неделя
2	Подпрограмма ввода схемы сети из файла (10%)	2 неделя
3	Подпрограмма вывода схемы сети с результатами расчетов в файл (10%)	3 неделя
4	Разработка алгоритма расчета режима (10%)	4 неделя
5	Разработка диаграммы классов для реализации алгоритма расчета режима(10%)	5 неделя
6	Разработка подпрограммы расчета режима (30%)	9 неделя
7	Тестирование программы расчета режима, устранение ошибок (10%)	10 неделя
8	Написание отчета (5%)	11 неделя
9	Защита отчета (5%)	12 неделя

4 Варианты заданий

Тема работы: «Разработка программы расчета режима сети на языке Python».

Цель работы: Формирование знаний и навыков применения электронных вычислительных машин для решения расчетных задач электроэнергетики

Задачи: Изучение интегрированной среды разработки Python IDLE, основ языка программирования Python, алгоритмов расчета режима электрических сетей приобретение практических навыков их реализации.

Перечень подлежащих разработке вопросов: Составить текстовую схему сети, рассчитать эталонный режим в промышленном программном комплексе RastrWin, разработать формат входного и выходного файлов в соответствии с индивидуальным заданием, разработать подпрограмму ввода/вывода схемы сети и результатов расчета, разработать алгоритм расчета режима, реализовать его на языке Python, выполнить тестирования разработанной программы, сравнив полученные результаты с результатами расчетов в ПК RastrWin, исправить ошибки при возникновении существенных расхождений в результатах.

Тестовая схема должна содержать не менее 10 узлов и не менее 3-х трансформаторов. Перечень графического материала: схема тестовой сети, блок-схема алгоритма расчета режима, диаграмма классов

Исходные данные по вариантам приведены в таблице 3

Таблица 3 – Исходные данные по вариантам

№ варианта	Метод расчета режима	Схема замещения ЛЭП	Схема замещения тр-ов	Входной файл	Выходной файл
1	«в два этапа»	«П»	«Г» (прямая)	таблица узлов, таблица ветвей	результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета перетоков мощности по линиям и трансформаторам, результаты расчета токов в линиях
2			«Г» (обратная)		результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам, линию с максимальными потерями мощности
3			«Т»		результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам
4		«Т»	«Г» (прямая);	таблица узлов, таблица ветвей, таблица типов трансформаторов (справочник)	результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета перетоков мощности по линиям и трансформаторам, результаты расчета токов в линиях
5					результаты расчетов параметров трансформаторов, результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам
6			«Т»		результаты расчета параметров трансформаторов, результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам, относительную загрузку трансформаторов (в % от номинальной мощности)
7			«Г» (обратная)	таблица узлов, таблица ветвей, таблица типов проводов (справочник), таблица типов трансформаторов (справочник)	результаты расчетов параметров линий и трансформаторов, результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам
8				результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам, линию с максимальными потерями мощности	
9				таблица узлов, таблица ветвей, таблица типов	результаты расчетов параметров линий, результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам, результаты расчетов перетоков мощности по линиям и

№ варианта	Метод расчета режима	Схема замещения ЛЭП	Схема замещения тр-ов	Входной файл	Выходной файл
				проводов (справочник)	трансформаторам
10					результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам
11			«Г» (прямая)		результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета перетоков мощности по линиям и трансформаторам, результаты расчета токов в линиях
12		«П»	«Г» (обратная)	таблица узлов, таблица ветвей	результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам, линию с максимальными потерями мощности
13			«Т»		результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам
14			«Г» (прямая);		результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета перетоков мощности по линиям и трансформаторам, результаты расчета токов в линиях
15	Зейделя	«Т»		таблица узлов, таблица ветвей, таблица типов трансформаторов (справочник)	результаты расчетов параметров трансформаторов, результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам
16			«Т»		результаты расчета параметров трансформаторов, результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам, относительную загрузку трансформаторов (в % от номинальной мощности)
17				таблица узлов, таблица ветвей, таблица типов проводов	результаты расчетов параметров линий и трансформаторов, результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам
18			«Г» (обратная)	(справочник), таблица типов трансформаторов (справочник)	результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам, линию с максимальными потерями мощности
19				таблица узлов,	результаты расчетов параметров линий, результаты расчета напряжений в узлах сети,

№ варианта	Метод расчета режима	Схема замещения ЛЭП	Схема замещения тр-ов	Входной файл	Выходной файл
				таблица ветвей, таблица типов проводов (справочник)	результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам, результаты расчетов перетоков мощности по линиям и трансформаторам
20					результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам
21			«Г» (прямая)		результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета перетоков мощности по линиям и трансформаторам, результаты расчета токов в линиях
22			«Г» (обратная)	таблица узлов, таблица ветвей	результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам, линию с максимальными потерями мощности
23			«Т»		результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам
24			«Г» (прямая);		результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета перетоков мощности по линиям и трансформаторам, результаты расчета токов в линиях
25	Ньютона			таблица узлов, таблица ветвей, таблица типов трансформаторов (справочник)	результаты расчетов параметров трансформаторов, результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам
26			«Т»		результаты расчета параметров трансформаторов, результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам, относительную загрузку трансформаторов (в % от номинальной мощности)
27				таблица узлов, таблица ветвей, таблица типов проводов (справочник), таблица типов трансформаторов	результаты расчетов параметров линий и трансформаторов, результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам
28			«Г» (обратная)		результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам, линию с максимальными потерями мощности

№ варианта	Метод расчета режима	Схема замещения ЛЭП	Схема замещения тр-ов	Входной файл	Выходной файл
				(справочник)	
29				таблица узлов, таблица ветвей, таблица типов проводов	результаты расчетов параметров линий, результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам, результаты расчетов перетоков мощности по линиям и трансформаторам
30				(справочник)	результаты расчета напряжений в узлах сети, результаты расчета мощности в центре питания, результаты расчета потерь мощности по линиям и трансформаторам

Список рекомендуемой литературы

Основная литература:

1. Седер, Н. Python. Экспресс-курс : практическое руководство / Н. Седер. - Санкт-Петербург : Питер, 2021. - 480 с. - (Серия «Библиотека программиста»). - ISBN 978-5-4461-0908-1. - Текст : электронный. - URL: <https://znanium.com/catalog/product/1756159> (дата обращения: 09.11.2022). – Режим доступа: по подписке.

2. Применение цифровых вычислительных машин в электроэнергетике : учеб. пособие / [О. В. Щербачев, А. Н. Зейлигер, К. П. Кадомская и др.] ; под ред. О. В. Щербачева. - Л. : Энергия, 1980. - 240 с. : ил. - Библиогр.: с. 229-232. - Предм. указ.: с. 233-234

3. Электрические системы. : Математические задачи электроэнергетики : учебник для вузов / [В. А. Веников, Э. Н. Зуев, И. В. Литкенс и др.] ; под ред. В. А. Веникова. - Изд. 2-е, перераб. и доп. - М. : Высшая школа, 1981. - 288 с. : ил. - Гриф: Доп. МО для электроэнергетических спец. - Библиогр: с. 285-286.

4. Дейтел, П. Python: Искусственный интеллект, большие данные и облачные вычисления: практическое руководство / П. Дейтел, Х. Дейтел. - Санкт-Петербург : Питер, 2020. - 864 с. - (Серия «Для профессионалов»). - ISBN 978-5-4461-1432-0. - Текст : электронный. - URL: <https://znanium.com/catalog/product/1733685> (дата обращения: 09.11.2022). – Режим доступа: по подписке.

Дополнительная литература:

5. Идельчик, В. И. Расчеты и оптимизация режимов электрических сетей и систем / В. И. Идельчик. - М. : Энергоатомиздат, 1988. - 288 с. : ил. - Библиогр.: с. 281-285. - ISBN 5-283-01103-8

6. Герасименко, А. А. Передача и распределение электрической энергии : учеб. пособие / А. А. Герасименко, В. Т. Федин. – Ростов н/Д : Феникс ; Красноярск : Издательские проекты, 2006. – 720 с. : ил. – (Высшее

образование). – Библиогр.: с. 667-671. – ISBN 5-222-08485-X (Феникс). – ISBN 5-98399-023-3 (Издательские проекты).

7. Чан, Д. Python: быстрый старт : практическое руководство / Д. Чан. - Санкт-Петербург: Питер, 2021. - 224 с. - (Серия «Библиотека программиста»). - ISBN 978-5-4461-1800-7. - Текст: электронный. - URL: <https://znanium.com/catalog/product/1739605> (дата обращения: 09.11.2022). – Режим доступа: по подписке.

Учебно-методическая литература:

8. Алгоритмы задач электроэнергетики. Методические указания по выполнению практических работ. Направление подготовки 13.03.02. Направленность (профиль) «Электроэнергетические системы и сети». /сост. Зеленский Е.Г.

9. Алгоритмы задач электроэнергетики. Методические указания по самостоятельной работе студентов. Направление подготовки 13.03.02. Направленность (профиль) «Электроэнергетические системы и сети». /сост. Зеленский Е.Г.

Интернет-ресурсы:

10. <https://www.python.org/>

11. <https://www.codabrainy.com/>

12. <https://znanium.com/>