МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по выполнению практических работ
по дисциплине «Объектно-ориентированное программирование»
для студентов направления подготовки
09.03.01 Информатика и вычислительная техника
Направленность (профиль)
«Программное обеспечение средств вычислительной техники и автоматизированных систем»

Ставрополь 2026

Содержание

ВВЕДЕНИЕ

1. Цель и задачи освоения дисциплины (модуля)

Цель дисциплины – формирование, согласно образовательной программы, компетенций ПК-1, ПК-2, ПК-3, ПК-4 обучающегося.

Для достижения целей освоения дисциплины «Объектно-ориентированное программирование» в процессе обучения решаются следующие задачи: ознакомление с основными понятиями и принципами объектно-ориентированного программирования, включая классы, объекты, инкапсуляцию, наследование и полиморфизм; формирование представлений об объектном моделировании предметной области и преобразовании требований к программной системе в структуру классов и объектов; изучение механизмов объектно-ориентированного программирования в языке Python, включая методы классов, свойства, наследование и переопределение методов; освоение принципов разработки расширяемого и сопровождаемого программного кода; формирование понимания принципов повторного использования программных компонентов и управления зависимостями между объектами; изучение принципов объектно-ориентированного проектирования и инженерных практик разработки программного обеспечения; ознакомление с принципами SOLID и их применением при проектировании программных систем; изучение механизмов обработки исключений и организации устойчивой логики работы программ; освоение продвинутых возможностей объектно-ориентированного программирования в Python, включая использование абстрактных базовых классов, дескрипторов и метаклассов; знакомство с основными паттернами проектирования и их применением при разработке программных систем; развитие практических навыков разработки объектно-ориентированных программ на языке Python; формирование умений анализировать архитектуру программных решений, выявлять проблемы проектирования и применять методы рефакторинга для повышения качества программного кода.

2. Место дисциплины (модуля) в структуре образовательной программы

Дисциплина относится к части, формируемой участниками образовательных отношений.

3. Перечень планируемых результатов обучения по дисциплине (модулю), соотнесённых с планируемыми результатами освоения образовательной программы

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
ПК-1. Способен разрабатывать требования и проектировать программное обеспечение	ИД-1$_{ПК-1}$ понимает принципы построения архитектуры программного обеспечения и виды архитектуры программного обеспечения; понимает методы и средства проектирования программного обеспечения, методы и средства проектирования баз данных; понимает методы и средства проектирования программных интерфейсов; понимает методы и приемы формализации задач;	Понимание принципов построения архитектуры программного обеспечения и основных видов архитектуры программных систем; понимание методов и средств проектирования программного обеспечения; понимание методов и средств проектирования программных

	понимает возможности современных и перспективных средств разработки программных продуктов, технических средств.	интерфейсов; понимание методов и приёмов формализации задач при разработке программных систем; понимание возможностей современных и перспективных средств разработки программных продуктов и используемых технических средств.
	ИД-2$_{ПК-1}$ проводит анализ исполнения требований; вырабатывает варианты реализации требований; проводит оценку и обоснование рекомендуемых решений; выбирает средства реализации требований к программному обеспечению; использует существующие типовые решения и шаблоны проектирования программного обеспечения; разрабатывает технические спецификации на программные компоненты и их взаимодействие.	Проведение анализа исполнения требований; выработка вариантов реализации требований; проведение оценки и обоснование рекомендуемых решений; выбор средств реализации требований к программному обеспечению; использование существующие типовых решений и шаблонов проектирования программного обеспечения; разработка технических спецификаций на программные компоненты и их взаимодействие.
	ИД-3$_{ПК-1}$ осуществляет оценку времени и трудоемкости реализации требований к программному обеспечению; применяет методы и средства проектирования программного обеспечения, структур данных, баз данных, программных интерфейсов	Осуществление оценки времени и трудоёмкости реализации требований к программному обеспечению; применение методов и средств проектирования программного обеспечения на основе объектно-ориентированного подхода; применение методов проектирования структур данных при разработке программных компонентов; применение методов проектирования программных интерфейсов;

		использование средств разработки и инструментов программирования при создании объектно-ориентированных программных систем.
ПК-2. Способен осуществлять концептуальное, функциональное и логическое проектирование систем среднего и крупного масштаба и сложности	ИД-10$_{ПК-2}$ Применяет методы и средства проектирования программного обеспечения, структур данных, баз данных, программных интерфейсов	Применение методов и средств проектирования программного обеспечения с использованием объектно-ориентированного подхода; применение методов проектирования структур данных при разработке программных компонентов; применение методов и средств проектирования программных интерфейсов; использование современных средств разработки программного обеспечения при реализации объектно-ориентированных программных систем.
	ИД-12$_{ПК-2}$ Разрабатывает программный код на языках программирования высокого уровня	Разработка программного кода на языке программирования высокого уровня Python; применение механизмов объектно-ориентированного программирования при разработке программных компонентов; реализация классов, объектов, методов и иерархий наследования; использование принципов инкапсуляции, полиморфизма и абстракции при создании программных решений; применение современных инструментов разработки и отладки программного обеспечения.

	ИД-13$_{ПК-2}$ Применяет принципы объектно-ориентированного программирования	Применение принципов объектно-ориентированного программирования при разработке программных систем; использование механизмов инкапсуляции, наследования, полиморфизма и абстракции при проектировании программных компонентов; разработка объектных моделей предметной области; применение принципов объектно-ориентированного проектирования при построении структуры программного обеспечения; использование паттернов проектирования и методов рефакторинга для повышения качества и сопровождаемости программного кода.
ПК-3. Способен осуществлять проектирование взаимодействия пользователя с системой и проводить эвристическую оценку графического пользовательского интерфейса	ИД-1$_{ПК-3}$ понимает стандарты, регламентирующие требования к эргономике взаимодействия человек – система; разрабатывает технические требования к интерфейсной графике; использует технологии визуализации данных и цифровую обработку изображений; выполняет верстку с использованием языков разметки и языков описания стилей; программирует с использованием сценарных языков.	Понимание требований к эргономике взаимодействия человека и программной системы; понимание принципов разработки технических требований к пользовательскому интерфейсу программных приложений; использование технологий визуализации данных при разработке программных решений; применение языков разметки и описания стилей при создании пользовательских интерфейсов программных систем; использование сценарных языков программирования при разработке

		компонентов программного обеспечения.
	ИД-2$_{ПК-3}$ разрабатывает и оформляет проектную документацию на интерфейс; создает интерактивные прототипы интерфейса; проектирует стили взаимодействия пользователя с графическим пользовательским интерфейсом программного продукта,	Разработка и оформление проектной документации, описывающей взаимодействие программных компонентов с пользовательским интерфейсом; создание прототипов программных решений, демонстрирующих взаимодействие объектов и пользовательских элементов системы; проектирование стилей взаимодействия пользователя с графическим интерфейсом программного продукта на основе объектной модели программной системы; использование объектно-ориентированного подхода при организации логики обработки пользовательских действий.
	ИД-3$_{ПК-3}$ анализирует данные о действиях пользователей при работе с интерфейсом, осуществляет формальную оценку графического пользовательского интерфейса.	Анализ данных о действиях пользователей при взаимодействии с программным интерфейсом; применение методов анализа пользовательских сценариев при разработке программных систем; проведение формальной оценки графического пользовательского интерфейса программного продукта; использование объектно-ориентированного подхода для организации обработки пользовательских действий и событий в

		программных приложениях.
ПК-4. Способен разрабатывать компоненты системных программных продуктов	ИД-3$_{ПК-4}$ создает блок-схемы алгоритмов функционирования разрабатываемых программных продуктов; выполняет оценку вычислительной сложности алгоритмов функционирования разрабатываемых программных продуктов	Создание блок-схем алгоритмов функционирования программных компонентов, реализованных с использованием объектно-ориентированного подхода; описание алгоритмов работы классов и методов разрабатываемых программных систем; выполнение оценки вычислительной сложности алгоритмов функционирования программных компонентов; анализ эффективности алгоритмов, реализованных в объектно-ориентированных программных решениях.

4. Объем учебной дисциплины (модуля) и формы контроля

Объем занятий: всего: 5 з.е. 180 акад.ч.	ОФО, В акад. часах
Контактная работа:	54.00/36.00
Лекции/из них практическая подготовка	18.00/0
Лабораторных работ/из них практическая подготовка	
Практических занятий/из них практическая подготовка	36.00/36.00
Самостоятельная работа	126.00
Формы контроля	
Экзамен	нет
Зачет	нет
Зачет с оценкой	да
Курсовая работа	нет

5. Важность практической работы студентов

Лабораторная работа является важной составляющей образовательного процесса по дисциплине «Объектно-ориентированное программирование», поскольку обеспечивает практическое освоение принципов и методов разработки программных систем на основе объектно-ориентированной парадигмы. Она способствует формированию устойчивых

навыков проектирования и реализации программных компонентов, а также закреплению теоретических знаний, полученных на лекционных занятиях.

В ходе выполнения лабораторных работ студенты изучают основные механизмы объектно-ориентированного программирования, такие как инкапсуляция, наследование, полиморфизм и абстракция, а также осваивают принципы SOLID и DRY, применяемые при проектировании программных систем. Практические задания предполагают разработку программных модулей на языке Python, использование современных инструментов разработки, создание и управление виртуальными окружениями, а также применение средств статического анализа и форматирования программного кода.

Особое внимание уделяется формированию навыков проектирования архитектуры программных систем, реализации шаблонов проектирования, разработке графических пользовательских интерфейсов с использованием библиотек tkinter и Kivy, а также организации взаимодействия пользовательского интерфейса с объектной моделью приложения. В рамках лабораторных работ студенты также получают практический опыт работы с базами данных SQLite и освоения методов тестирования программного обеспечения с использованием библиотек unittest и pytest.

Выполнение лабораторных заданий способствует развитию алгоритмического и системного мышления, навыков анализа программного кода, умения применять современные инструменты разработки и обеспечивать качество создаваемых программных решений. Студенты учатся структурировать программные проекты, разрабатывать масштабируемые и поддерживаемые программные системы, а также анализировать эффективность реализованных решений.

Таким образом, лабораторные работы обеспечивают практическую реализацию теоретических положений курса и играют ключевую роль в формировании профессиональных компетенций в области разработки программного обеспечения, позволяя студентам приобрести необходимые знания и навыки для проектирования и реализации современных программных систем.

ПРАКТИЧЕСКАЯ РАБОТА №1

Тема: Реализация классов и объектов в Python. Атрибуты, методы и создание экземпляров классов

Цель работы

Освоение базовых принципов объектно-ориентированного программирования на языке Python. Формирование практических навыков создания классов, определения атрибутов и методов, а также создания и использования экземпляров классов в программных системах.

Теоретическое обоснование

Объектно-ориентированное программирование (ООП) представляет собой парадигму разработки программного обеспечения, в основе которой лежит представление программы в виде совокупности взаимодействующих объектов. Каждый объект является экземпляром определённого класса и обладает собственным состоянием и поведением.

Класс представляет собой шаблон или модель, описывающую свойства и действия объектов. Свойства объекта реализуются в виде атрибутов класса или атрибутов экземпляра, а действия — в виде методов. Атрибуты хранят данные, характеризующие состояние объекта, а методы определяют операции, которые могут выполняться над объектом.

В языке Python класс объявляется с использованием ключевого слова `class`. Внутри класса могут определяться переменные и функции, которые формируют структуру и поведение объектов. Специальный метод `__init__` используется как конструктор класса и вызывается автоматически при создании нового экземпляра.

Экземпляр класса представляет собой конкретный объект, созданный на основе класса. Каждый экземпляр имеет собственные значения атрибутов, однако использует общие методы, определённые в классе. Обращение к атрибутам и методам осуществляется через точечную нотацию.

Использование классов и объектов позволяет структурировать программный код, повышать его читаемость и поддерживаемость, а также создавать более гибкие и масштабируемые программные системы. Освоение механизмов создания классов и работы с объектами является фундаментальным этапом изучения объектно-ориентированного программирования.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm, Jupyter Notebook или аналогичная
Средства управления виртуальными окружениями Python (venv или uv)

Методические указания по выполнению работы

Перед началом выполнения задания необходимо убедиться, что установлен интерпретатор Python и корректно настроена среда разработки. Рекомендуется создать отдельное виртуальное окружение для выполнения лабораторной работы.

При разработке программного кода следует придерживаться общепринятых правил оформления Python-кода и использовать понятные имена классов, методов и переменных. Каждый класс должен описывать логически завершённую сущность предметной области.

В ходе выполнения работы необходимо последовательно реализовать несколько классов, определить их атрибуты и методы, создать экземпляры классов и продемонстрировать их использование в программе.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка стороннего программного обеспечения без разрешения преподавателя.

Типовые задачи

1. Создание простого класса

Реализовать класс `Student`, содержащий следующие атрибуты:

- имя студента
- номер группы
- средний балл

Создать метод класса для вывода информации о студенте.
Создать несколько экземпляров класса и вывести информацию о каждом студенте.

2. Реализация класса с методами изменения состояния

Создать класс `BankAccount`, содержащий:

- номер счёта
- имя владельца
- текущий баланс

Реализовать методы:

- пополнение счёта
- снятие средств
- вывод текущего баланса

Создать экземпляр класса и выполнить несколько операций со счётом.

3. Работа с атрибутами и методами класса

Создать класс `Rectangle`, описывающий прямоугольник.

Атрибуты класса:

- ширина
- высота

Методы класса:

- вычисление площади
- вычисление периметра

Создать несколько объектов класса и вычислить параметры каждого прямоугольника.

4. Использование конструктора класса

Реализовать класс `Book`, содержащий:

- название книги
- имя автора
- год издания

Конструктор должен инициализировать атрибуты при создании экземпляра.
Создать список из нескольких объектов класса и вывести информацию о книгах.

5. Моделирование объектов предметной области

Разработать класс `Car`, содержащий следующие характеристики:

- марка автомобиля
- год выпуска
- текущая скорость

Реализовать методы:

- запуск двигателя
- изменение скорости
- остановка автомобиля

Создать объект автомобиля и продемонстрировать работу методов.

Контрольные вопросы

1. Что такое класс в объектно-ориентированном программировании?
2. Что представляет собой объект и чем он отличается от класса?
3. Какие элементы могут входить в структуру класса?
4. Что такое атрибут экземпляра класса?
5. Что такое метод класса?
6. Как создаётся экземпляр класса в Python?
7. Какую роль выполняет метод `__init__`?
8. Как осуществляется обращение к атрибутам и методам объекта?
9. Чем отличаются атрибуты класса от атрибутов экземпляра?
10. Какие преимущества даёт использование объектно-ориентированного подхода при разработке программного обеспечения?

ПРАКТИЧЕСКАЯ РАБОТА №2

Тема: Конструкторы классов и управление состоянием объектов

Цель работы

Освоение методов инициализации объектов с использованием конструкторов классов и формирование навыков управления состоянием объектов при разработке интеллектуальных агентов. Получение практического опыта моделирования поведения программных объектов, представляющих агентов и элементы среды.

Теоретическое обоснование

В системах искусственного интеллекта и многоагентных системах программные объекты часто используются для моделирования интеллектуальных агентов и компонентов окружающей среды. Каждый агент обладает определённым состоянием, которое описывает его текущее положение, внутренние параметры, накопленные знания и возможные действия.

Состояние объекта представляет собой совокупность значений его атрибутов в определённый момент времени. В процессе функционирования системы состояние объектов мо-

жет изменяться под воздействием действий агента, взаимодействия с другими агентами или изменений среды. Управление состоянием объектов является важным элементом разработки программных моделей интеллектуальных систем.

В языке Python начальная инициализация объекта осуществляется при помощи специального метода `__init__`, который называется конструктором класса. Конструктор автоматически вызывается при создании экземпляра класса и используется для задания начальных значений атрибутов объекта. Благодаря конструкторам можно задавать параметры агента, его начальное состояние и характеристики среды.

В контексте интеллектуальных агентов состояние может включать такие параметры, как текущее положение агента в среде, уровень энергии, список доступных действий, накопленный опыт или текущая цель. Методы класса позволяют изменять состояние агента, реагировать на события и реализовывать алгоритмы поведения.

Правильное проектирование конструкторов и механизмов управления состоянием объектов позволяет создавать более гибкие и расширяемые модели интеллектуальных систем, а также облегчает разработку и сопровождение программных решений.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm, Jupyter Notebook или аналогичная
Средства управления виртуальными окружениями Python (venv, uv)
Текстовый редактор или IDE для написания программного кода

Методические указания по выполнению работы

Перед началом выполнения работы необходимо создать отдельный проект Python и подготовить рабочую среду. Все программные решения рекомендуется сохранять в отдельных файлах или ячейках ноутбука.

При разработке классов следует использовать конструкторы для задания начального состояния объектов. Важно обеспечить корректную инициализацию всех необходимых атрибутов и предусмотреть методы для изменения состояния объекта в процессе работы программы.

В заданиях необходимо реализовать классы, моделирующие интеллектуальных агентов и элементы среды, определить параметры их состояния и разработать методы, позволяющие изменять эти параметры в процессе выполнения программы.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать чрезмерной нагрузки на зрение при длительной работе с экраном.
Не допускается установка программного обеспечения из непроверенных источников.

Типовые задачи

1. Создание класса интеллектуального агента

Разработать класс `Agent`, описывающий интеллектуального агента.

Конструктор класса должен инициализировать следующие параметры:

- имя агента
- уровень энергии
- текущее состояние агента

Реализовать метод, выводящий информацию о текущем состоянии агента.
Создать несколько экземпляров класса с различными параметрами.

2. Управление состоянием агента

Дополнить класс `Agent` методами:

- уменьшение уровня энергии
- восстановление энергии
- изменение состояния агента

Продемонстрировать изменение состояния объекта в процессе выполнения программы.

3. Моделирование среды для агента

Создать класс `Environment`, описывающий среду, в которой действует агент.

Конструктор класса должен принимать параметры:

- размер среды
- количество ресурсов
- уровень сложности среды

Реализовать метод отображения состояния среды.

4. Взаимодействие агента со средой

Разработать метод, позволяющий агенту взаимодействовать со средой.

Например:

- поиск ресурса
- перемещение в новую позицию
- потребление ресурса

Изменения должны отражаться в состоянии объекта агента и объекта среды.

5. Моделирование нескольких агентов

Создать несколько объектов класса `Agent`, находящихся в одной среде.

Смоделировать последовательность действий агентов:

- перемещение
- взаимодействие с ресурсами
- изменение внутренних параметров

Вывести результаты моделирования.

Контрольные вопросы

1. Что такое конструктор класса в языке Python?
2. Какую роль выполняет метод `__init__`?
3. Что понимается под состоянием объекта?
4. Почему важно корректно инициализировать атрибуты объекта?
5. Как методы класса могут изменять состояние объекта?
6. Что представляет собой программная модель агента?
7. Какие параметры могут описывать состояние интеллектуального агента?
8. Как осуществляется взаимодействие агента со средой?
9. Почему важно контролировать изменение состояния объектов в программной системе?
10. Как использование классов упрощает моделирование многоагентных систем?

ПРАКТИЧЕСКАЯ РАБОТ №3

Тема: Инкапсуляция и использование свойств (property)

Цель работы

Освоение принципа инкапсуляции при разработке программных моделей интеллектуальных агентов. Формирование навыков ограничения доступа к внутреннему состоянию объектов и управления этим состоянием с использованием механизма свойств (`property`) в языке Python.

Теоретическое обоснование

Одним из ключевых принципов объектно-ориентированного программирования является инкапсуляция. Она заключается в объединении данных и методов, работающих с этими данными, внутри одного объекта, а также в ограничении прямого доступа к внутреннему состоянию объекта. Это позволяет повысить надежность программных систем, уменьшить вероятность ошибок и обеспечить контроль над изменением состояния объекта.

В программных моделях интеллектуальных агентов инкапсуляция играет важную роль, поскольку состояние агента может включать множество параметров: уровень энергии, текущую цель, позицию в среде, набор доступных действий и другие характеристики. Неконтролируемое изменение этих параметров может привести к некорректному поведению системы.

В языке Python инкапсуляция реализуется через соглашения об именовании атрибутов и использование специальных механизмов управления доступом. Атрибуты, начинающиеся

с символа подчеркивания, считаются внутренними и не предназначены для прямого доступа извне класса.

Для более гибкого управления доступом используется механизм свойств (`property`). Свойства позволяют обращаться к атрибутам объекта как к обычным переменным, но при этом фактически вызываются методы класса. Это дает возможность выполнять дополнительные проверки, изменять формат данных или управлять логикой изменения состояния объекта.

Свойства реализуются с использованием декораторов `@property`, `@<имя>.setter` и `@<имя>.deleter`. Такой подход позволяет обеспечить безопасный доступ к данным объекта, сохраняя удобный синтаксис взаимодействия с атрибутами.

Применение инкапсуляции и свойств особенно важно при разработке интеллектуальных систем, где корректность состояния объектов напрямую влияет на поведение агентов и результаты моделирования.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm, Jupyter Notebook или аналогичная
Средства управления виртуальными окружениями Python (venv, uv)

Методические указания по выполнению работы

Перед выполнением работы необходимо повторить основные принципы объектно-ориентированного программирования и особенности использования свойств в Python.

В ходе выполнения заданий следует реализовывать классы, содержащие закрытые атрибуты, и использовать свойства для управления доступом к этим атрибутам. Необходимо предусмотреть проверки корректности данных при изменении состояния объектов.

При разработке программных решений рекомендуется моделировать параметры интеллектуальных агентов и среды, используя свойства для контроля их изменения.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка стороннего программного обеспечения без разрешения преподавателя.

Типовые задачи

1. Инкапсуляция параметров агента

Создать класс `Agent`, описывающий интеллектуального агента.

Конструктор должен инициализировать следующие параметры:

- имя агента
- уровень энергии
- текущая цель агента

Атрибут уровня энергии должен быть скрытым (закрытым).
Реализовать свойства для получения и изменения значения энергии.

2. Проверка корректности состояния объекта

Дополнить класс `Agent` свойством для управления уровнем энергии.

При изменении значения энергии необходимо:

- запрещать отрицательные значения
- ограничивать максимальный уровень энергии

При попытке установить некорректное значение должно выводиться сообщение об ошибке.

3. Управление состоянием среды

Создать класс `Environment`, содержащий скрытый атрибут количества ресурсов.

Реализовать свойства для:

- получения текущего количества ресурсов
- изменения количества ресурсов

Добавить проверку, предотвращающую установку отрицательного количества ресурсов.

4. Ограничение доступа к внутренним данным

Создать класс `Sensor`, описывающий датчик агента.

Атрибуты класса:

- тип датчика
- текущее значение показаний

Значение показаний должно быть закрытым атрибутом.
Реализовать свойства для безопасного доступа и изменения показаний.

5. Инкапсуляция поведения агента

Разработать класс `AgentState`, описывающий состояние агента.

Состояние может принимать значения:

- "idle"
- "searching"
- "executing"

Использовать свойства для управления изменением состояния и запретить установку недопустимых значений.

Контрольные вопросы

1. Что такое инкапсуляция в объектно-ориентированном программировании?
2. Почему важно ограничивать доступ к внутренним данным объекта?
3. Как в Python обозначаются внутренние атрибуты класса?
4. Что представляет собой механизм `property`?
5. Для чего используется декоратор `@property`?
6. Для чего используется декоратор `@setter`?
7. Какие преимущества дает использование свойств по сравнению с прямым доступом к атрибутам?
8. Как свойства помогают контролировать изменение состояния объекта?
9. В чем заключается роль инкапсуляции при моделировании интеллектуальных агентов?
10. Какие проблемы могут возникнуть при отсутствии контроля доступа к данным объекта?

ПРАКТИЧЕСКАЯ РАБОТА №4

Тема: Наследование классов, переопределение методов и полиморфизм (утиная типизация)

Цель работы

Освоение механизмов наследования, переопределения методов и полиморфизма в языке Python. Формирование навыков проектирования иерархий классов при моделировании различных типов интеллектуальных агентов и их поведения в программной системе.

Теоретическое обоснование

В объектно-ориентированном программировании наследование является механизмом, позволяющим создавать новые классы на основе уже существующих. Производный класс наследует атрибуты и методы базового класса, что позволяет повторно использовать программный код и создавать более сложные структуры программных систем.

Наследование широко применяется при моделировании сложных систем, включая интеллектуальные и многоагентные системы. В таких системах различные типы агентов могут обладать общими характеристиками, но при этом иметь различное поведение. Например, агенты могут различаться по стратегии принятия решений, способам взаимодействия со средой или набору доступных действий.

Переопределение методов позволяет изменять поведение метода базового класса в производном классе. Это дает возможность адаптировать общую функциональность под осо-

бенности конкретного типа объекта. В Python переопределение метода осуществляется путем повторного объявления метода в производном классе.

Полиморфизм является еще одним важным принципом объектно-ориентированного программирования. Он позволяет использовать единый интерфейс для работы с объектами различных типов. В Python широко применяется концепция утиной типизации (duck typing), согласно которой объект считается подходящим для использования, если он обладает необходимыми методами и свойствами, независимо от его конкретного типа.

Утиная типизация позволяет писать более гибкий и универсальный программный код. В контексте интеллектуальных агентов это особенно полезно, поскольку различные типы агентов могут реализовывать одинаковые методы поведения, но при этом иметь разные внутренние структуры.

Использование наследования и полиморфизма позволяет создавать расширяемые архитектуры программных систем, облегчает сопровождение кода и делает систему более адаптируемой к изменениям.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm, Jupyter Notebook или аналогичная
Средства управления виртуальными окружениями Python (venv, uv)

Методические указания по выполнению работы

Перед выполнением заданий необходимо повторить принципы объектно-ориентированного программирования, связанные с наследованием и полиморфизмом.

В ходе работы следует разработать иерархию классов, описывающих различные типы интеллектуальных агентов. Базовый класс должен содержать общие характеристики и методы, а производные классы должны расширять или изменять поведение базового класса.

При реализации задач рекомендуется продемонстрировать использование переопределения методов и принципов утиной типизации при взаимодействии объектов.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка стороннего программного обеспечения без разрешения преподавателя.

Типовые задачи

1. Создание базового класса агента

Реализовать базовый класс `Agent`, содержащий:

- имя агента
- позицию агента в среде

Методы класса:

- `perceive()` — получение информации из среды
- `act()` — выполнение действия

Методы могут выводить сообщение о выполняемом действии.

2. Наследование классов

Создать два производных класса:

- `SearchAgent`
- `ReactiveAgent`

Каждый класс должен наследоваться от базового класса `Agent`.

Переопределить метод `act()` таким образом, чтобы:

- `SearchAgent` выполнял поиск решения
- `ReactiveAgent` реагировал на изменения среды

3. Расширение функциональности производных классов

Дополнить производные классы собственными атрибутами.

Например:

`SearchAgent`

- алгоритм поиска

`ReactiveAgent`

- уровень чувствительности к изменениям среды

Продемонстрировать использование этих атрибутов в методах класса.

4. Демонстрация полиморфизма

Создать список объектов различных типов агентов.

В цикле вызвать метод `act()` для каждого агента.

Показать, что каждый объект выполняет свою реализацию метода.

5. Использование утиной типизации

Создать класс `EnvironmentMonitor`, не наследующийся от класса `Agent`, но содержащий метод `act()`.

Написать функцию, принимающую объект и вызывающую метод `act()`.

Передать в функцию объекты различных классов и показать, что функция работает с ними независимо от их типа.

Контрольные вопросы

1. Что такое наследование в объектно-ориентированном программировании?
2. Какие преимущества дает использование наследования?
3. Что такое базовый и производный класс?
4. Что означает переопределение метода?
5. Как в Python реализуется наследование классов?
6. Что такое полиморфизм?
7. Как проявляется полиморфизм в объектно-ориентированном программировании?
8. Что означает принцип утиной типизации?
9. Чем утиная типизация отличается от строгой типизации?
10. Почему полиморфизм является важным механизмом при разработке интеллектуальных и многоагентных систем?

ПРАКТИЧЕСКАЯ РАБОТА №5

Тема: Абстрактные базовые классы и абстрактные методы (abc)

Цель работы

Освоение методов проектирования иерархий классов с использованием абстрактных базовых классов и абстрактных методов. Формирование навыков разработки архитектуры программных систем, обеспечивающей единый интерфейс взаимодействия для различных типов интеллектуальных агентов.

Теоретическое обоснование

При разработке сложных программных систем, включая интеллектуальные и многоагентные системы, важно обеспечить единообразие интерфейсов взаимодействия между различными компонентами системы. Одним из способов достижения этой цели является использование абстрактных базовых классов.

Абстрактный базовый класс представляет собой класс, предназначенный для описания общей структуры и поведения группы объектов. Такой класс не предполагает создание собственных экземпляров, а используется в качестве основы для построения конкретных классов-наследников. Абстрактный класс определяет общий интерфейс, который должны реализовать все производные классы.

В языке Python для создания абстрактных базовых классов используется модуль `abc` (Abstract Base Classes). Он предоставляет специальный класс `ABC`, а также декоратор `@abstractmethod`, позволяющий объявлять абстрактные методы. Абстрактный метод

представляет собой метод, который объявляется в базовом классе, но не содержит реализации. Его реализация должна быть обязательно предоставлена в производных классах.

Использование абстрактных классов позволяет формализовать архитектуру программной системы и задать обязательные требования к структуре производных классов. Это особенно важно в системах, где существует множество типов агентов, выполняющих различные действия, но взаимодействующих с системой через единый интерфейс.

В контексте интеллектуальных агентов абстрактный класс может описывать общий интерфейс агента, включающий методы восприятия среды, принятия решений и выполнения действий. Конкретные типы агентов могут реализовывать эти методы по-разному, сохраняя при этом единый программный интерфейс.

Применение абстрактных базовых классов способствует повышению модульности, расширяемости и надежности программных систем, облегчает сопровождение кода и позволяет более четко формализовать архитектуру системы.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm, Jupyter Notebook или аналогичная
Стандартная библиотека Python, включая модуль `abc`

Методические указания по выполнению работы

Перед выполнением заданий необходимо изучить принципы использования абстрактных базовых классов и абстрактных методов.

В процессе выполнения работы следует разработать абстрактный класс, описывающий общий интерфейс интеллектуального агента. В производных классах необходимо реализовать конкретное поведение агентов, определив реализацию абстрактных методов.

Особое внимание следует уделить правильной организации иерархии классов и обеспечению соответствия интерфейсов всех производных классов требованиям абстрактного базового класса.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка стороннего программного обеспечения без разрешения преподавателя.

Типовые задачи

1. Создание абстрактного базового класса агента

Создать абстрактный класс `Agent` с использованием модуля `abc`.

Абстрактный класс должен содержать следующие абстрактные методы:

- `perceive()` — получение информации из среды
- `decide()` — принятие решения
- `act()` — выполнение действия

Попытаться создать экземпляр абстрактного класса и объяснить результат.

2. Реализация конкретного агента

Создать класс `ReactiveAgent`, наследующийся от абстрактного класса `Agent`.

Реализовать все абстрактные методы базового класса.
Методы должны моделировать простое реактивное поведение агента.

3. Реализация агента с планированием

Создать класс `PlanningAgent`, также наследующийся от абстрактного класса `Agent`.

Методы класса должны реализовывать более сложное поведение:

- анализ состояния среды
- выбор стратегии действий
- выполнение действия

4. Использование полиморфизма

Создать несколько объектов различных типов агентов (`ReactiveAgent`, `PlanningAgent`).

Поместить их в список и в цикле вызвать методы `perceive()`, `decide()` и `act()` для каждого объекта.

Показать, что объекты разных типов используют собственные реализации методов.

5. Расширение архитектуры агентов

Добавить ещё один тип агента, например `LearningAgent`, реализующий методы базового класса.

Метод `decide()` должен учитывать накопленный опыт агента.

Продемонстрировать работу нового агента вместе с ранее созданными.

Контрольные вопросы

1. Что такое абстрактный базовый класс?
2. Для чего используются абстрактные классы в объектно-ориентированном программировании?
3. Какие средства Python используются для создания абстрактных классов?
4. Что представляет собой абстрактный метод?

5. Можно ли создать экземпляр абстрактного класса?
6. Почему производные классы обязаны реализовывать абстрактные методы?
7. Как абстрактные классы помогают формировать архитектуру программной системы?
8. В чем преимущества использования абстрактных классов при разработке интеллектуальных агентов?
9. Как абстрактные классы связаны с принципом полиморфизма?
10. Какие проблемы могут возникнуть при отсутствии единых интерфейсов для различных типов агентов?

ПРАКТИЧЕСКАЯ РАБОТА №6

Тема: Специальные методы классов и перегрузка операторов

Цель работы

Освоение механизмов специальных (магических) методов классов в языке Python и формирование навыков их применения для управления поведением объектов. Получение практического опыта перегрузки операторов и представления объектов интеллектуальных агентов в программной системе.

Теоретическое обоснование

В языке Python существуют специальные методы классов, которые позволяют управлять поведением объектов при выполнении различных операций. Эти методы имеют имена, начинающиеся и заканчивающиеся двойным подчеркиванием, например `__init__`, `__str__`, `__repr__`, `__eq__`, `__add__` и другие. Их часто называют магическими методами.

Специальные методы автоматически вызываются интерпретатором Python при выполнении определённых операций над объектами. Например, метод `__str__` используется для получения строкового представления объекта, а метод `__eq__` применяется при сравнении объектов. Благодаря этим методам можно определить, как объект должен отображаться, сравниваться или участвовать в арифметических операциях.

Перегрузка операторов позволяет определить поведение стандартных операторов языка программирования для объектов пользовательских классов. Это делает программный код более естественным и удобным для восприятия. Например, можно определить, как должны складываться два объекта, как они сравниваются или как определяется их равенство.

В контексте интеллектуальных агентов специальные методы могут использоваться для моделирования взаимодействия агентов между собой, сравнения состояний агентов, объединения ресурсов или определения приоритетов действий. Например, оператор сложения может использоваться для объединения ресурсов агентов, а оператор сравнения — для определения агента с наибольшим уровнем энергии.

Использование специальных методов повышает выразительность программного кода и позволяет интегрировать пользовательские классы в стандартную модель работы Python. Это делает программные системы более гибкими и облегчает разработку сложных моделей поведения.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm, Jupyter Notebook или аналогичная
Средства управления виртуальными окружениями Python (venv, uv)

Методические указания по выполнению работы

Перед началом выполнения работы необходимо изучить основные специальные методы классов Python и принципы их использования.

В ходе выполнения заданий следует разработать классы, моделирующие интеллектуальных агентов и элементы среды. Для этих классов необходимо реализовать специальные методы, определяющие их строковое представление, сравнение объектов и поведение при выполнении арифметических операций.

При реализации задач рекомендуется продемонстрировать использование перегрузки операторов для моделирования взаимодействия агентов.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка стороннего программного обеспечения без разрешения преподавателя.

Типовые задачи

1. Строковое представление объекта агента

Создать класс `Agent`, содержащий следующие параметры:

- имя агента
- уровень энергии
- текущая позиция

Реализовать метод `__str__`, который будет возвращать удобное текстовое представление агента.

Создать несколько объектов и вывести их на экран.

2. Представление объекта для разработчика

Дополнить класс `Agent` методом `__repr__`, который будет возвращать формальное представление объекта.

Сравнить результаты работы методов `__str__` и `__repr__`.

3. Сравнение агентов

Реализовать метод `__eq__`, позволяющий сравнивать двух агентов по уровню энергии.

Создать несколько объектов и проверить корректность работы оператора `==`.

4. Перегрузка оператора сложения

Добавить в класс `Agent` метод `__add__`.

Оператор сложения должен объединять ресурсы двух агентов и возвращать нового агента с суммарным уровнем энергии.

Продемонстрировать работу оператора `+`.

5. Сравнение приоритетов агентов

Реализовать методы сравнения:

- `__lt__` (меньше)
- `__gt__` (больше)

Сравнение должно выполняться по уровню энергии агента.

Создать список агентов и выполнить сортировку по уровню энергии.

Контрольные вопросы

1. Что такое специальные (магические) методы классов в Python?
2. Какие функции выполняют методы `__str__` и `__repr__`?
3. Что такое перегрузка операторов?
4. Как реализуется перегрузка операторов в Python?
5. Для чего используется метод `__eq__`?
6. Какие методы используются для реализации операций сравнения?
7. Как можно определить поведение оператора сложения для объектов пользовательского класса?
8. Какие преимущества дает использование специальных методов?
9. Как специальные методы помогают интегрировать пользовательские классы в стандартную модель работы Python?
10. В каких случаях перегрузка операторов может быть полезна при разработке интеллектуальных систем?

ПРАКТИЧЕСКАЯ РАБОТА №7

Тема: Итераторы и генераторы в Python

Цель работы

Освоение механизмов создания и использования итераторов и генераторов в языке Python. Формирование практических навыков реализации последовательностей действий и событий в моделях интеллектуальных и многоагентных систем.

Теоретическое обоснование

Во многих задачах искусственного интеллекта и многоагентных систем необходимо работать с последовательностями действий, состояний или событий. Например, агент может последовательно выполнять шаги алгоритма поиска, перемещаться по среде, анализировать поток поступающих данных или принимать решения на основе текущего состояния системы.

В языке Python важную роль в обработке последовательностей играют итераторы и генераторы. Итератор представляет собой объект, позволяющий последовательно получать элементы некоторой структуры данных. Для реализации итератора объект должен поддерживать методы `__iter__()` и `__next__()`. Метод `__iter__()` возвращает сам объект итератора, а метод `__next__()` возвращает следующий элемент последовательности.

Итераторы позволяют эффективно обрабатывать большие наборы данных без необходимости хранения всей последовательности в памяти. Это особенно важно при моделировании процессов, в которых количество состояний или действий может быть значительным.

Генераторы являются удобным инструментом для создания итераторов. Генератор представляет собой специальную функцию, которая возвращает значения по одному с использованием оператора `yield`. При каждом вызове генератора выполнение функции продолжается с того места, где оно было остановлено ранее.

Генераторы позволяют описывать последовательности действий более компактно и наглядно. Они широко используются при обработке потоков данных, моделировании поведения агентов и организации пошаговых алгоритмов.

В контексте интеллектуальных систем итераторы и генераторы могут применяться для моделирования поведения агента во времени, последовательности принимаемых решений, обработки сенсорных данных или реализации алгоритмов поиска и планирования.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm, Jupyter Notebook или аналогичная
Средства управления виртуальными окружениями Python (venv, uv)

Методические указания по выполнению работы

Перед началом выполнения работы необходимо повторить принципы работы итераторов и генераторов в Python.

В ходе выполнения заданий следует реализовать собственные итераторы и генераторы, моделирующие последовательности действий интеллектуальных агентов. Важно проде-

монстрировать работу методов `__iter__()` и `__next__()` и использовать оператор `yield` при создании генераторов.

Рекомендуется рассматривать последовательности действий агента или изменение состояний среды как последовательные шаги итерации.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка стороннего программного обеспечения без разрешения преподавателя.

Типовые задачи

1. Реализация простого итератора

Создать класс `AgentPath`, представляющий последовательность позиций агента в среде.

Класс должен:

- хранить список координат перемещения агента
- реализовывать методы `__iter__()` и `__next__()`

При каждой итерации должен возвращаться следующий шаг перемещения агента.

2. Итерация по действиям агента

Создать класс `AgentActions`, содержащий список возможных действий агента:

- перемещение
- поиск ресурса
- взаимодействие с объектом

Реализовать возможность перебора действий агента в цикле `for`.

3. Генератор последовательности состояний

Создать генератор `agent_states()`, который последовательно возвращает состояния агента:

- ожидание
- анализ среды
- принятие решения
- выполнение действия

Использовать оператор `yield` для генерации состояний.

4. Генератор шагов моделирования

Реализовать генератор, моделирующий последовательность шагов работы агента в среде.

Каждый вызов генератора должен возвращать:

- номер шага
- текущее действие агента

Вывести несколько шагов моделирования.

5. Генератор взаимодействия агентов

Создать генератор, моделирующий взаимодействие нескольких агентов.

Каждая итерация должна возвращать:

- идентификатор агента
- выполняемое действие

Смоделировать несколько циклов взаимодействия агентов.

Контрольные вопросы

1. Что такое итератор в языке Python?
2. Какие методы должен реализовывать объект-итератор?
3. Как работает метод `__next__()`?
4. Что происходит при завершении последовательности итерации?
5. Что такое генератор?
6. Какую роль играет оператор `yield`?
7. Чем генераторы отличаются от обычных функций?
8. Какие преимущества дают генераторы при работе с большими последовательностями данных?
9. В каких задачах интеллектуальных систем могут применяться генераторы?
10. Почему итераторы и генераторы полезны при моделировании поведения агентов?

ПРАКТИЧЕСКАЯ РАБОТА №8

Тема: Менеджеры контекста и использование конструкции with

Цель работы

Освоение механизма менеджеров контекста в языке Python и формирование навыков безопасного управления ресурсами программной системы. Получение практического опыта использования конструкции `with` при разработке моделей интеллектуальных и многоагентных систем.

Теоретическое обоснование

При разработке программных систем часто возникает необходимость управлять ресурсами, такими как файлы, сетевые соединения, базы данных или вычислительные среды. Эти

ресурсы должны быть корректно открыты перед использованием и обязательно закрыты после завершения работы.

В языке Python для решения этой задачи используется механизм менеджеров контекста. Менеджер контекста представляет собой объект, который определяет поведение программы при входе в определённый блок кода и при выходе из него. Управление таким блоком осуществляется с помощью конструкции `with`.

Менеджеры контекста реализуются через специальные методы класса `__enter__()` и `__exit__()`. Метод `__enter__()` выполняется при входе в блок `with` и обычно используется для подготовки ресурса к работе. Метод `__exit__()` вызывается при завершении выполнения блока и отвечает за корректное освобождение ресурса.

Использование менеджеров контекста позволяет автоматизировать управление ресурсами и делает программный код более безопасным и понятным. Даже в случае возникновения исключения менеджер контекста гарантирует корректное завершение работы с ресурсом.

В системах искусственного интеллекта и многоагентных системах менеджеры контекста могут применяться для управления временными состояниями среды, работы с файлами журналов действий агентов, обработки данных датчиков или взаимодействия с базами данных.

Например, менеджер контекста может использоваться для организации временной среды моделирования, в которой агент выполняет определённые действия. После завершения моделирования среда автоматически возвращается в исходное состояние.

Таким образом, менеджеры контекста являются важным инструментом для управления жизненным циклом ресурсов и повышения надежности программных систем.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm, Jupyter Notebook или аналогичная
Средства управления виртуальными окружениями Python (venv, uv)

Методические указания по выполнению работы

Перед началом выполнения работы необходимо изучить принцип работы менеджеров контекста и конструкции `with`.

В ходе выполнения заданий следует реализовать собственные менеджеры контекста и использовать их для управления различными ресурсами. В частности, можно моделировать работу среды интеллектуального агента, временные изменения параметров системы или запись информации о действиях агентов.

При реализации задач рекомендуется продемонстрировать корректную работу методов `__enter__()` и `__exit__()` и убедиться, что освобождение ресурсов происходит даже при возникновении ошибок.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка стороннего программного обеспечения без разрешения преподавателя.

Типовые задачи

1. Использование менеджера контекста для работы с файлами

Создать программу, которая записывает действия интеллектуального агента в файл журнала.

Использовать конструкцию `with` для открытия файла.
После завершения записи файл должен автоматически закрываться.

2. Реализация собственного менеджера контекста

Создать класс `SimulationEnvironment`, реализующий менеджер контекста.

Методы класса должны:

- при входе в контекст выводить сообщение о запуске среды моделирования
- при выходе из контекста выводить сообщение о завершении работы среды

Использовать этот класс в конструкции `with`.

3. Управление состоянием агента в контексте

Создать менеджер контекста `EnergyControl`, который временно изменяет уровень энергии агента.

При входе в контекст уровень энергии увеличивается.
При выходе из контекста значение возвращается к исходному.

4. Логирование действий агента

Разработать менеджер контекста, записывающий начало и завершение выполнения действия агента в журнал.

Внутри блока `with` выполнить несколько действий агента.

5. Обработка исключений в менеджере контекста

Модифицировать один из созданных менеджеров контекста так, чтобы он корректно обрабатывал возможные ошибки внутри блока `with`.

Проверить работу программы при возникновении исключения.

Контрольные вопросы

1. Что такое менеджер контекста в Python?
2. Для чего используется конструкция `with`?
3. Какие методы должен реализовывать менеджер контекста?
4. Какую роль выполняет метод `__enter__()`?
5. Какую роль выполняет метод `__exit__()`?
6. Почему использование менеджеров контекста повышает надежность программ?
7. Какие ресурсы чаще всего управляются с помощью менеджеров контекста?
8. Как менеджеры контекста помогают обрабатывать исключения?
9. В каких задачах интеллектуальных систем могут применяться менеджеры контекста?
10. Какие преимущества дает автоматическое управление ресурсами в программных системах?

ПРАКТИЧЕСКАЯ РАБОТА №9

Тема: Создание виртуальных окружений Python и настройка инструментов разработки (uv, flake8, isort, black, ruff, pre-commit)

Цель работы

Освоение методов создания изолированных виртуальных окружений Python и настройка инструментов контроля качества программного кода. Формирование навыков организации профессиональной среды разработки при создании программных систем, включая проекты интеллектуальных и многоагентных систем.

Теоретическое обоснование

При разработке программных систем на языке Python важную роль играет организация среды разработки и управление зависимостями проекта. В современных программных проектах используется большое количество сторонних библиотек, которые могут иметь различные версии и несовместимые зависимости. Для решения этой проблемы применяются виртуальные окружения.

Виртуальное окружение представляет собой изолированную среду выполнения Python, в которой можно устанавливать необходимые библиотеки и инструменты независимо от других проектов и системной установки Python. Это позволяет избежать конфликтов между версиями библиотек и обеспечивает воспроизводимость программной среды.

Одним из современных инструментов для управления виртуальными окружениями и зависимостями является утилита `uv`. Она предоставляет быстрый и удобный механизм создания окружений, установки пакетов и управления зависимостями проекта.

Помимо управления зависимостями важным аспектом разработки является контроль качества программного кода. Для этого используются специальные инструменты статического анализа и форматирования кода.

Инструмент **flake8** выполняет проверку кода на соответствие стандартам оформления и выявляет потенциальные ошибки.
Инструмент **isort** автоматически сортирует и упорядочивает импорты в исходных файлах.
Инструмент **black** выполняет автоматическое форматирование кода в соответствии с единым стилем.

Инструмент **ruff** представляет собой быстрый анализатор кода, который объединяет возможности нескольких инструментов статического анализа.

Для автоматизации запуска этих проверок используется инструмент **pre-commit**, который позволяет выполнять проверки кода перед фиксацией изменений в системе контроля версий. Это помогает поддерживать единый стиль кода и предотвращать появление ошибок на ранних этапах разработки.

Использование этих инструментов является стандартной практикой современной разработки программного обеспечения и позволяет значительно повысить качество и поддерживаемость программных проектов.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Доступ к командной строке или терминалу
Установленный инструмент `uv`
Среда разработки: Visual Studio Code, PyCharm или аналогичная
Система контроля версий Git

Методические указания по выполнению работы

Перед началом выполнения работы необходимо убедиться, что установлен Python и доступна командная строка.

В процессе выполнения работы необходимо создать новый проект Python, настроить виртуальное окружение и установить необходимые инструменты разработки. Далее следует настроить автоматическую проверку кода с использованием инструментов статического анализа и форматирования.

Рекомендуется выполнять все действия через командную строку, фиксируя основные этапы настройки проекта.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка программного обеспечения из непроверенных источников.

Типовые задачи

1. Создание нового проекта Python

Создать каталог проекта.

Внутри каталога создать новый Python-проект и файл программы, например `main.py`.

2. Создание виртуального окружения

Используя утилиту `uv`, создать виртуальное окружение проекта.

Активировать виртуальное окружение и проверить установленную версию Python.

3. Установка инструментов разработки

Установить следующие инструменты разработки:

- flake8
- isort
- black
- ruff
- pre-commit

Проверить корректность их установки.

4. Проверка и форматирование программного кода

Создать Python-файл с несколькими намеренными нарушениями правил оформления кода.

Использовать:

- `flake8` для проверки кода
- `isort` для сортировки импортов
- `black` для форматирования кода
- `ruff` для статического анализа

Проанализировать полученные результаты.

5. Настройка автоматической проверки кода

Инициализировать репозиторий Git.

Создать конфигурационный файл `pre-commit`.

Настроить автоматический запуск инструментов проверки кода перед фиксацией изменений.

Выполнить тестовый коммит и проверить работу системы автоматической проверки.

Контрольные вопросы

1. Что такое виртуальное окружение Python?
2. Почему использование виртуальных окружений важно при разработке программных проектов?
3. Какие задачи решает инструмент `uv`?
4. Для чего используется инструмент `flake8`?
5. Какую функцию выполняет инструмент `isort`?
6. Для чего используется инструмент `black`?
7. Какие возможности предоставляет инструмент `ruff`?

8. Что такое статический анализ программного кода?
9. Какую роль выполняет инструмент `pre-commit`?
10. Почему автоматическая проверка кода является важной частью современного процесса разработки программного обеспечения?

ПРАКТИЧЕСКАЯ РАБОТА №10

Тема: Реализация принципов SOLID при проектировании классов

Цель работы

Освоение принципов SOLID и формирование навыков их практического применения при проектировании объектно-ориентированных программных систем. Получение опыта разработки гибкой и расширяемой архитектуры программных компонентов на примере моделей интеллектуальных агентов.

Теоретическое обоснование

При разработке сложных программных систем важным аспектом является качество архитектуры программного кода. Хорошо спроектированная система должна быть понятной, расширяемой и удобной для сопровождения. Одним из наиболее известных наборов принципов объектно-ориентированного проектирования являются принципы SOLID.

Название SOLID представляет собой акроним, образованный из первых букв пяти принципов проектирования:

Single Responsibility Principle (SRP) — принцип единственной ответственности.
Каждый класс должен отвечать только за одну часть функциональности системы. Это позволяет уменьшить сложность классов и облегчает сопровождение кода.

Open/Closed Principle (OCP) — принцип открытости/закрытости.
Классы должны быть открыты для расширения, но закрыты для изменения. Новое поведение должно добавляться за счёт создания новых классов или расширения существующих.

Liskov Substitution Principle (LSP) — принцип подстановки Барбары Лисков.
Объекты производных классов должны быть взаимозаменяемыми с объектами базовых классов без нарушения корректности работы программы.

Interface Segregation Principle (ISP) — принцип разделения интерфейсов.
Клиенты не должны зависеть от методов, которые они не используют. Лучше создавать несколько специализированных интерфейсов вместо одного универсального.

Dependency Inversion Principle (DIP) — принцип инверсии зависимостей.
Модули верхнего уровня не должны зависеть от модулей нижнего уровня. Оба типа модулей должны зависеть от абстракций.

Применение принципов SOLID способствует созданию модульных и масштабируемых программных систем. Эти принципы особенно важны при разработке интеллектуальных и многоагентных систем, где система может включать множество взаимодействующих компонентов.

При моделировании интеллектуальных агентов различные аспекты их поведения — восприятие среды, принятие решений, планирование действий — могут быть разделены на отдельные компоненты. Такой подход позволяет легко расширять функциональность системы, добавляя новые типы агентов или алгоритмы принятия решений.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm, Jupyter Notebook или аналогичная
Средства управления виртуальными окружениями Python (venv, uv)

Методические указания по выполнению работы

Перед началом выполнения работы необходимо повторить принципы SOLID и рассмотреть примеры их применения в объектно-ориентированном программировании.

В ходе выполнения заданий необходимо разработать классы, моделирующие интеллектуальных агентов и элементы среды. Особое внимание следует уделить разделению ответственности между классами и построению гибкой архитектуры системы.

Рекомендуется сначала реализовать простую модель системы, а затем выполнить её рефакторинг с применением принципов SOLID.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка стороннего программного обеспечения без разрешения преподавателя.

Типовые задачи

1. Реализация принципа единственной ответственности

Разработать класс `Agent`, выполняющий сразу несколько функций:

- хранение состояния агента
- принятие решений
- выполнение действий

После этого выполнить рефакторинг и разделить функциональность на несколько классов:

- `AgentState`
- `DecisionModule`
- `ActionExecutor`

Показать, как изменяется структура программы после применения принципа SRP.

2. Применение принципа открытости/закрытости

Создать систему агентов, в которой различные типы агентов принимают решения разными способами.

Реализовать базовый класс `DecisionStrategy`.

Создать несколько стратегий принятия решений:

- случайная стратегия
- стратегия поиска
- стратегия на основе правил

Продемонстрировать добавление новой стратегии без изменения существующего кода.

3. Проверка принципа подстановки Лисков

Создать базовый класс `Agent` и несколько производных классов.

Проверить, что объекты производных классов могут использоваться вместо объектов базового класса без изменения логики программы.

4. Разделение интерфейсов

Создать интерфейс агента, содержащий множество методов.

После этого разделить его на несколько специализированных интерфейсов, например:

- интерфейс восприятия среды
- интерфейс принятия решений
- интерфейс выполнения действий

Продемонстрировать использование этих интерфейсов различными типами агентов.

5. Реализация принципа инверсии зависимостей

Создать систему, в которой агент зависит от конкретного алгоритма принятия решений.

Затем выполнить рефакторинг, используя абстрактный интерфейс алгоритма принятия решений.

Показать, как можно легко заменить алгоритм, не изменяя код агента.

Контрольные вопросы

1. Что представляет собой набор принципов SOLID?
2. В чем заключается принцип единственной ответственности?
3. Как работает принцип открытости/закрытости?
4. Что означает принцип подстановки Лисков?
5. Почему важно разделять интерфейсы программных компонентов?
6. В чем заключается принцип инверсии зависимостей?
7. Как применение принципов SOLID влияет на архитектуру программной системы?

8. Почему принципы SOLID важны при разработке интеллектуальных систем?
9. Какие проблемы могут возникнуть при нарушении принципов SOLID?
10. Как применение принципов SOLID облегчает расширение программной системы?

ПРАКТИЧЕСКАЯ РАБОТА №11

Тема: Реализация порождающих паттернов проектирования на Python (Factory, Singleton)

Цель работы

Освоение порождающих паттернов проектирования и формирование навыков их применения при разработке программных систем. Получение практического опыта реализации паттернов Factory и Singleton на языке Python при моделировании интеллектуальных агентов и компонентов среды.

Теоретическое обоснование

При разработке сложных программных систем важно обеспечить гибкость и расширяемость архитектуры. Одним из инструментов, позволяющих решать такие задачи, являются паттерны проектирования. Паттерны проектирования представляют собой типовые решения часто возникающих задач архитектурного проектирования программных систем.

Порождающие паттерны (creational patterns) направлены на управление процессом создания объектов. Они позволяют отделить процесс создания объектов от логики их использования, что повышает гибкость архитектуры системы.

Одним из наиболее распространённых порождающих паттернов является **Factory (Фабричный метод)**. Данный паттерн используется для создания объектов без прямого указания их конкретного класса. Вместо этого используется специальный метод или класс-фабрика, который отвечает за создание объектов нужного типа. Такой подход позволяет легко добавлять новые типы объектов без изменения существующего кода системы.

В интеллектуальных системах фабричный метод может использоваться для создания различных типов агентов. Например, в системе могут существовать реактивные агенты, обучающиеся агенты или агенты планирования. Использование фабрики позволяет создавать необходимые типы агентов динамически в зависимости от параметров системы.

Другим важным порождающим паттерном является **Singleton (Одиночка)**. Этот паттерн гарантирует, что в системе существует только один экземпляр определённого класса. Singleton часто применяется для реализации объектов, представляющих глобальные ресурсы системы, например, менеджеры конфигурации, журналы событий, среды моделирования или контроллеры взаимодействия агентов.

Использование паттерна Singleton позволяет централизовать управление общими ресурсами системы и предотвратить создание нескольких экземпляров объектов, которые должны существовать в единственном числе.

Применение порождающих паттернов позволяет улучшить архитектуру программных систем, повысить модульность кода и облегчить сопровождение и расширение программных решений.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm, Jupyter Notebook или аналогичная
Средства управления виртуальными окружениями Python (venv, uv)

Методические указания по выполнению работы

Перед началом выполнения работы необходимо изучить основные принципы паттернов проектирования и их роль в архитектуре программных систем.

В ходе выполнения заданий следует разработать программные компоненты, реализующие порождающие паттерны Factory и Singleton. Рекомендуется рассматривать классы интеллектуальных агентов и элементы среды как объекты, создаваемые с использованием фабричного метода.

При реализации паттерна Singleton необходимо обеспечить создание только одного экземпляра класса и продемонстрировать его использование в различных частях программы.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка стороннего программного обеспечения без разрешения преподавателя.

Типовые задачи

1. Реализация фабрики агентов

Создать базовый класс `Agent`.

Реализовать несколько типов агентов:

- `ReactiveAgent`
- `PlanningAgent`
- `LearningAgent`

Создать класс `AgentFactory`, который будет создавать объекты агентов в зависимости от переданного типа.

2. Использование фабричного метода

Разработать функцию или метод фабрики, принимающий тип агента и возвращающий объект соответствующего класса.

Создать несколько агентов различных типов с использованием фабрики.

3. Расширение фабрики

Добавить новый тип агента, например `ExplorationAgent`.

Изменить фабрику таким образом, чтобы она могла создавать объекты нового типа без изменения существующей архитектуры системы.

4. Реализация паттерна Singleton

Создать класс `EnvironmentManager`, который представляет среду моделирования.

Реализовать паттерн Singleton таким образом, чтобы в программе существовал только один экземпляр данного класса.

5. Использование Singleton в системе агентов

Создать несколько агентов, которые получают доступ к объекту среды через Singleton.

Проверить, что все агенты работают с одним и тем же экземпляром среды.

Контрольные вопросы

1. Что такое паттерны проектирования?
2. Какие задачи решают порождающие паттерны?
3. В чем заключается идея паттерна Factory?
4. Какие преимущества дает использование фабричного метода?
5. Как фабричный метод помогает расширять программную систему?
6. В чем заключается идея паттерна Singleton?
7. В каких случаях используется паттерн Singleton?
8. Какие проблемы могут возникнуть при использовании Singleton?
9. Как порождающие паттерны помогают улучшить архитектуру программной системы?
10. Как паттерны Factory и Singleton могут применяться при разработке интеллектуальных и многоагентных систем?

ПРАКТИЧЕСКАЯ РАБОТА №12

Тема: Реализация структурных паттернов проектирования (Decorator, Adapter)

Цель работы

Освоение принципов использования структурных паттернов проектирования и формирование навыков их применения при разработке программных систем. Получение практического опыта реализации паттернов Decorator и Adapter на языке Python при моделировании поведения интеллектуальных агентов.

Теоретическое обоснование

Паттерны проектирования являются проверенными архитектурными решениями, применяемыми при разработке программного обеспечения. Они позволяют структурировать программный код, уменьшить связанность компонентов системы и повысить её гибкость.

Структурные паттерны проектирования направлены на организацию взаимодействия между объектами и классами. Их основная задача заключается в формировании удобной структуры программной системы, позволяющей эффективно комбинировать и расширять функциональность компонентов.

Одним из широко используемых структурных паттернов является **Decorator (Декоратор)**. Этот паттерн позволяет динамически добавлять объекту новую функциональность без изменения исходного класса. Декоратор оборачивает объект и расширяет его поведение, сохраняя исходный интерфейс. Такой подход особенно полезен, когда необходимо добавить дополнительную функциональность к объектам во время выполнения программы.

В контексте интеллектуальных систем декораторы могут использоваться для расширения возможностей агентов. Например, базовый агент может выполнять стандартные действия, а дополнительные декораторы могут добавлять функции логирования, мониторинга, анализа среды или обучения.

Другим важным структурным паттерном является **Adapter (Адаптер)**. Этот паттерн используется для обеспечения совместимости между объектами с несовместимыми интерфейсами. Адаптер преобразует интерфейс одного класса в интерфейс, ожидаемый другим компонентом системы.

В интеллектуальных и многоагентных системах адаптеры могут применяться для интеграции различных компонентов системы, включая внешние источники данных, сенсоры, системы моделирования среды или сторонние библиотеки.

Использование структурных паттернов проектирования позволяет строить гибкие и расширяемые архитектуры программных систем, упрощает интеграцию компонентов и повышает поддерживаемость программного кода.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm, Jupyter Notebook или аналогичная
Средства управления виртуальными окружениями Python (venv, uv)

Методические указания по выполнению работы

Перед началом выполнения работы необходимо изучить принципы структурных паттернов проектирования и особенности реализации паттернов Decorator и Adapter.

В ходе выполнения заданий следует разработать программные компоненты, моделирующие интеллектуальных агентов и элементы среды. На этих примерах необходимо продемонстрировать применение декораторов для расширения поведения объектов и адаптеров для обеспечения совместимости между различными интерфейсами.

Особое внимание следует уделить сохранению единого интерфейса взаимодействия между компонентами системы.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка стороннего программного обеспечения без разрешения преподавателя.

Типовые задачи

1. Реализация базового агента

Создать базовый класс `Agent`, содержащий метод `act()`, описывающий действие агента в среде.

Метод должен выводить информацию о выполняемом действии.

2. Реализация паттерна Decorator

Создать класс-декоратор `LoggingDecorator`, который добавляет к действию агента функцию логирования.

При вызове метода `act()` должна выводиться дополнительная информация о действии агента.

3. Расширение функциональности агента

Создать ещё один декоратор, например `EnergyMonitorDecorator`, который отслеживает уровень энергии агента при выполнении действия.

Показать, что можно комбинировать несколько декораторов.

4. Реализация паттерна Adapter

Предположим, что в системе имеется внешний класс `ExternalSensor`, имеющий метод `read_value()`.

Создать адаптер `SensorAdapter`, который преобразует интерфейс внешнего класса к интерфейсу системы агентов (например, метод `get_data()`).

5. Интеграция адаптера в систему

Создать класс агента, который использует адаптер для получения данных от внешнего сенсора.

Показать, что агент может работать с сенсором через единый интерфейс, независимо от внутренней реализации сенсора.

Контрольные вопросы

1. Что такое структурные паттерны проектирования?
2. Какие задачи решает паттерн Decorator?
3. В чем заключается идея динамического расширения функциональности объектов?
4. Какие преимущества дает использование паттерна Decorator?
5. Что такое паттерн Adapter?
6. В каких случаях используется адаптер?
7. Как адаптер помогает интегрировать несовместимые интерфейсы?
8. Какие преимущества дают структурные паттерны при разработке программных систем?
9. Как паттерны Decorator и Adapter могут применяться в интеллектуальных системах?
10. Почему использование паттернов проектирования повышает гибкость архитектуры программного обеспечения?

ПРАКТИЧЕСКАЯ РАБОТА №13

Тема: Реализация поведенческих паттернов проектирования (Strategy, Observer)

Цель работы

Освоение принципов использования поведенческих паттернов проектирования и формирование навыков их применения при разработке программных систем. Получение практического опыта реализации паттернов Strategy и Observer на языке Python для моделирования поведения интеллектуальных агентов и их взаимодействия.

Теоретическое обоснование

Паттерны проектирования представляют собой типовые решения архитектурных задач, возникающих при разработке программных систем. Среди них важное место занимают поведенческие паттерны, которые описывают способы организации взаимодействия между объектами и распределения ответственности между компонентами системы.

Поведенческие паттерны позволяют создавать гибкие механизмы управления поведением объектов и их взаимодействием. Это особенно важно при разработке интеллектуальных и многоагентных систем, где различные агенты могут использовать различные стратегии поведения и взаимодействовать между собой.

Одним из наиболее распространённых поведенческих паттернов является **Strategy (Стратегия)**. Данный паттерн позволяет определять семейство алгоритмов, инкапсулировать каждый из них в отдельный класс и делать их взаимозаменяемыми. В результате поведение объекта может изменяться динамически без изменения его структуры.

В интеллектуальных системах паттерн Strategy может использоваться для реализации различных алгоритмов принятия решений агентом. Например, агент может использовать стратегию случайного выбора действий, стратегию поиска оптимального решения или стратегию, основанную на правилах.

Другим важным поведенческим паттерном является **Observer (Наблюдатель)**. Этот паттерн используется для организации механизма уведомления объектов о происходящих со-

бытиях. Один объект (наблюдаемый) хранит список зависимых объектов (наблюдателей) и уведомляет их при изменении своего состояния.

В многоагентных системах паттерн Observer может использоваться для организации взаимодействия между агентами и средой. Например, агенты могут получать уведомления об изменениях состояния среды, появлении ресурсов или действиях других агентов.

Использование поведенческих паттернов позволяет создавать более гибкие и масштабируемые программные системы, облегчает расширение функциональности и делает архитектуру программного обеспечения более устойчивой к изменениям.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm, Jupyter Notebook или аналогичная
Средства управления виртуальными окружениями Python (venv, uv)

Методические указания по выполнению работы

Перед выполнением заданий необходимо изучить принципы поведенческих паттернов проектирования и особенности реализации паттернов Strategy и Observer.

В процессе выполнения работы следует реализовать несколько стратегий поведения интеллектуальных агентов и продемонстрировать возможность их динамической замены. Также необходимо реализовать механизм уведомления агентов о событиях среды с использованием паттерна Observer.

Особое внимание следует уделить организации взаимодействия между объектами и разделению ответственности между компонентами системы.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка стороннего программного обеспечения без разрешения преподавателя.

Типовые задачи

1. Реализация паттерна Strategy

Создать базовый интерфейс `DecisionStrategy`, определяющий метод `decide_action()`.

Реализовать несколько стратегий принятия решений:

- `RandomStrategy` — случайный выбор действия

- `RuleBasedStrategy` — выбор действия на основе правил
- `SearchStrategy` — использование алгоритма поиска

2. Использование стратегии агентом

Создать класс `Agent`, который принимает стратегию принятия решений.

Метод `act()` должен использовать текущую стратегию для выбора действия.

Продемонстрировать возможность изменения стратегии во время работы программы.

3. Реализация паттерна Observer

Создать класс `Environment`, представляющий среду моделирования.

Среда должна хранить список агентов-наблюдателей и уведомлять их при изменении состояния.

4. Реализация наблюдателей

Создать класс `AgentObserver`, который подписывается на события среды.

При получении уведомления агент должен выполнять определённое действие.

5. Моделирование взаимодействия агентов

Создать несколько агентов-наблюдателей и зарегистрировать их в среде.

Смоделировать изменение состояния среды и продемонстрировать, как агенты получают уведомления и реагируют на события.

Контрольные вопросы

1. Что такое поведенческие паттерны проектирования?
2. В чем заключается идея паттерна Strategy?
3. Какие преимущества дает использование стратегии при разработке программных систем?
4. Как паттерн Strategy позволяет изменять поведение объекта?
5. Что представляет собой паттерн Observer?
6. Какие роли существуют в паттерне Observer?
7. Как наблюдатели получают уведомления о событиях?
8. В каких задачах интеллектуальных систем может применяться паттерн Observer?
9. Какие преимущества дают поведенческие паттерны при проектировании программных систем?
10. Почему использование паттернов Strategy и Observer важно для разработки многоагентных систем?

ПРАКТИЧЕСКАЯ РАБОТА №14

Тема: Основы работы с SQLite: структура базы данных и SQL-запросы

Цель работы

Освоение базовых принципов работы с реляционными базами данных и формирование навыков создания и использования базы данных SQLite. Получение практического опыта создания таблиц, выполнения SQL-запросов и хранения данных, связанных с функционированием интеллектуальных агентов.

Теоретическое обоснование

Во многих программных системах возникает необходимость долговременного хранения данных. В интеллектуальных и многоагентных системах это могут быть данные о состоянии среды, действиях агентов, результатах взаимодействия между агентами, накопленном опыте или статистике работы системы.

Одним из наиболее распространённых способов хранения данных являются реляционные базы данных. Реляционная база данных представляет собой структурированную систему хранения информации, где данные организованы в виде таблиц. Каждая таблица состоит из строк и столбцов, где строки представляют записи, а столбцы определяют типы данных.

SQLite является лёгкой и встроенной системой управления базами данных. Она не требует отдельного сервера и может использоваться непосредственно внутри приложения. База данных SQLite хранится в одном файле, что делает её удобной для использования в небольших программных проектах, учебных системах и прототипах интеллектуальных приложений.

Для взаимодействия с реляционными базами данных используется язык **SQL (Structured Query Language)**. С помощью SQL выполняются операции создания таблиц, добавления данных, обновления записей, удаления информации и выполнения различных запросов к базе данных.

Основными типами SQL-команд являются:

- **DDL (Data Definition Language)** — команды для создания и изменения структуры базы данных (CREATE, ALTER, DROP)
- **DML (Data Manipulation Language)** — команды для работы с данными (INSERT, UPDATE, DELETE)
- **DQL (Data Query Language)** — команды для получения данных (SELECT)

В интеллектуальных системах базы данных могут использоваться для хранения информации о действиях агентов, истории взаимодействий, параметров среды или результатов моделирования. Понимание структуры базы данных и навыки работы с SQL-запросами являются важной частью разработки современных программных систем.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm, Jupyter Notebook или аналогичная
Система управления базами данных SQLite
Инструмент для работы с SQLite (например, DB Browser for SQLite)

Методические указания по выполнению работы

Перед выполнением работы необходимо ознакомиться с основными понятиями реляционных баз данных и базовыми SQL-командами.

В процессе выполнения работы необходимо создать базу данных SQLite, разработать структуру таблиц и выполнить различные SQL-запросы для работы с данными. В качестве предметной области рекомендуется использовать модели интеллектуальных агентов и среды их функционирования.

Все SQL-запросы рекомендуется выполнять через интерфейс работы с SQLite или через консольный инструмент SQLite.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка программного обеспечения из непроверенных источников.

Типовые задачи

1. Создание базы данных

Создать новую базу данных SQLite.

Определить структуру таблицы `agents`, содержащей следующие поля:

- идентификатор агента
- имя агента
- тип агента
- уровень энергии
- текущее состояние

2. Создание таблиц

Создать таблицу `environment`, содержащую параметры среды:

- идентификатор среды
- размер среды
- количество ресурсов
- уровень сложности

3. Добавление данных

Добавить в таблицу `agents` несколько записей, описывающих различных агентов.

Добавить данные в таблицу `environment`.

4. Выполнение запросов

Выполнить SQL-запросы:

- вывод всех агентов
- вывод агентов с уровнем энергии выше заданного значения
- сортировка агентов по уровню энергии

5. Изменение и удаление данных

Изменить состояние одного из агентов.

Удалить запись об агенте из таблицы.

Проверить корректность выполнения операций.

Контрольные вопросы

1. Что представляет собой реляционная база данных?
2. Какие основные элементы включает структура базы данных?
3. Что такое таблица в реляционной базе данных?
4. Что представляет собой первичный ключ?
5. Что такое язык SQL?
6. Какие основные группы SQL-команд существуют?
7. Для чего используется команда SELECT?
8. Для чего используются команды INSERT, UPDATE и DELETE?
9. Какие преимущества имеет система управления базами данных SQLite?
10. Как базы данных могут использоваться в интеллектуальных и многоагентных системах?

ПРАКТИЧЕСКАЯ РАБОТА №15

Тема: Работа с SQLite в Python: подключение к базе данных и выполнение запросов

Цель работы

Освоение практических навыков работы с базой данных SQLite из программ на Python. Формирование умений подключения к базе данных, создания таблиц, выполнения SQL-запросов и обработки результатов запросов в программном коде.

Теоретическое обоснование

Современные программные системы, включая интеллектуальные и многоагентные системы, требуют хранения и обработки большого объёма данных. Эти данные могут включать информацию о состоянии среды, действиях агентов, результатах взаимодействия, накопленном опыте и различных параметрах функционирования системы.

Для хранения структурированных данных широко используются реляционные базы данных. В таких системах данные организованы в виде таблиц, состоящих из строк и столбцов. Каждая строка представляет запись, а каждый столбец определяет тип хранимых данных.

SQLite является встроенной системой управления базами данных, которая не требует установки отдельного сервера. Она широко используется в мобильных приложениях, встроенных системах, научных проектах и учебных задачах. База данных SQLite хранится в одном файле, что упрощает её использование и переносимость.

Язык программирования Python содержит встроенный модуль **sqlite3**, позволяющий работать с базами данных SQLite непосредственно из программного кода. С помощью данного модуля можно подключаться к базе данных, выполнять SQL-запросы, добавлять и изменять данные, а также извлекать результаты запросов для последующей обработки.

Процесс взаимодействия Python-программы с базой данных обычно включает несколько этапов: установление соединения с базой данных, создание объекта курсора, выполнение SQL-запросов, получение результатов и закрытие соединения.

В интеллектуальных системах использование базы данных позволяет сохранять состояние агентов, вести историю взаимодействий, хранить параметры среды и результаты экспериментов. Интеграция базы данных с программным кодом делает систему более гибкой и позволяет анализировать накопленные данные.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm, Jupyter Notebook или аналогичная
Встроенный модуль Python `sqlite3`
Инструмент для просмотра базы данных (например, DB Browser for SQLite)

Методические указания по выполнению работы

Перед выполнением работы необходимо изучить структуру взаимодействия программы Python с базой данных SQLite и основные функции модуля `sqlite3`.

В процессе выполнения работы необходимо создать Python-программу, которая подключается к базе данных SQLite, создаёт таблицы, добавляет записи, выполняет выборку данных и отображает результаты запросов.

В качестве предметной области рекомендуется использовать данные интеллектуальных агентов и среды их функционирования.

При выполнении заданий необходимо уделить внимание корректной работе с соединением с базой данных и обработке возможных ошибок.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка программного обеспечения из непроверенных источников.

Типовые задачи

1. Подключение к базе данных

Создать Python-программу, которая подключается к базе данных SQLite с использованием модуля `sqlite3`.

Если файл базы данных отсутствует, он должен быть создан автоматически.

2. Создание таблицы

Создать таблицу `agents`, содержащую информацию об интеллектуальных агентах:

- id агента
- имя агента
- тип агента
- уровень энергии
- текущее состояние

Таблица должна быть создана с использованием SQL-запроса `CREATE TABLE`.

3. Добавление данных

Добавить несколько записей в таблицу `agents` с использованием SQL-запроса `INSERT`.

Данные могут описывать различных агентов с различными параметрами.

4. Получение данных

Выполнить SQL-запрос `SELECT` для получения всех записей из таблицы.

Вывести результаты в консоль Python-программы.

5. Фильтрация данных

Реализовать запрос, который выводит агентов с уровнем энергии выше заданного значения.

Результаты должны быть обработаны и выведены в удобном для анализа виде.

6. Обновление данных

Изменить состояние одного из агентов с использованием SQL-запроса `UPDATE`.

Проверить, что данные были успешно изменены.

7. Удаление данных

Удалить одну из записей из таблицы с использованием SQL-запроса `DELETE`.

Проверить корректность выполнения операции.

8. Закрытие соединения

Корректно закрыть соединение с базой данных после завершения работы программы.

Контрольные вопросы

1. Что представляет собой система управления базами данных SQLite?
2. Какие преимущества имеет использование SQLite в программных проектах?
3. Какой модуль Python используется для работы с SQLite?
4. Какие этапы включает процесс взаимодействия Python-программы с базой данных?
5. Что такое курсор базы данных?
6. Какие SQL-команды используются для создания таблиц?
7. Как добавить запись в таблицу базы данных?
8. Как выполнить выборку данных из базы данных?
9. Как обновить существующую запись в таблице?
10. Почему важно корректно закрывать соединение с базой данных.

ПРАКТИЧЕСКАЯ РАБОТА №16

Тема: Разработка графического интерфейса пользователя с использованием библиотеки tkinter

Цель работы

Освоение базовых принципов разработки графических пользовательских интерфейсов на языке Python. Формирование навыков создания оконных приложений, размещения элементов интерфейса и обработки пользовательских событий с использованием библиотеки tkinter.

Теоретическое обоснование

Графический пользовательский интерфейс (GUI) является важной частью современных программных систем. Он обеспечивает взаимодействие пользователя с программным обеспечением через визуальные элементы управления, такие как окна, кнопки, текстовые поля, меню и списки.

В отличие от консольных программ, графические приложения позволяют пользователю взаимодействовать с системой более интуитивно, используя элементы управления и визуальные подсказки. Это особенно важно для прикладных программных систем, где удобство взаимодействия напрямую влияет на эффективность работы пользователя.

В языке Python для разработки графических интерфейсов используется несколько библиотек, одной из наиболее распространённых является **tkinter**. Эта библиотека входит в стандартную поставку Python и предоставляет набор инструментов для создания оконных приложений.

Библиотека tkinter построена на основе объектно-ориентированной модели. Основными компонентами интерфейса являются **виджеты** — объекты, представляющие элементы управления. К ним относятся окна, кнопки, текстовые поля, метки, поля ввода и другие элементы интерфейса.

Создание графического интерфейса обычно включает следующие этапы: создание главного окна приложения, добавление элементов интерфейса, размещение элементов в окне с использованием менеджеров геометрии и обработку событий, возникающих при взаимодействии пользователя с интерфейсом.

Одним из ключевых элементов GUI-приложений является **обработка событий**. Когда пользователь нажимает кнопку, вводит текст или выбирает элемент меню, генерируется событие, которое обрабатывается программой. Реализация обработки событий позволяет связать действия пользователя с логикой программы.

В объектно-ориентированном программировании графический интерфейс часто реализуется в виде отдельных классов, которые взаимодействуют с объектной моделью приложения. Такой подход обеспечивает разделение логики программы и интерфейса пользователя, что упрощает сопровождение и расширение программной системы.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm или аналогичная
Стандартная библиотека Python с модулем `tkinter`

Методические указания по выполнению работы

Перед выполнением работы необходимо изучить основные компоненты библиотеки tkinter и принципы организации графического интерфейса.

В процессе выполнения работы необходимо создать простое графическое приложение, содержащее несколько элементов интерфейса. Следует реализовать обработку пользовательских действий, таких как нажатие кнопок или ввод данных.

Рекомендуется организовать структуру программы с использованием классов, чтобы продемонстрировать применение принципов объектно-ориентированного программирования при разработке интерфейса.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка стороннего программного обеспечения без разрешения преподавателя.

Типовые задачи

1. Создание окна приложения

Создать простое графическое приложение на Python с использованием библиотеки tkinter.

Реализовать главное окно приложения с заголовком и заданными размерами.

2. Добавление элементов интерфейса

Добавить в окно следующие элементы:

- текстовую метку (Label)
- кнопку (Button)
- поле ввода текста (Entry)

Разместить элементы в окне с использованием одного из менеджеров геометрии (`pack`, `grid` или `place`).

3. Обработка событий

Реализовать обработчик события нажатия кнопки.

При нажатии кнопки программа должна считывать текст из поля ввода и отображать его в метке.

4. Создание интерфейса на основе классов

Реализовать графическое приложение в виде класса `Application`.

Все элементы интерфейса должны быть созданы внутри конструктора класса.

5. Реализация простого интерфейса управления агентом

Создать интерфейс, позволяющий пользователю:

- задать имя агента
- выбрать тип агента
- выполнить действие (например, запуск агента)

Отобразить результат действия в интерфейсе.

Контрольные вопросы

1. Что представляет собой графический пользовательский интерфейс?
2. Какие преимущества имеет использование GUI по сравнению с консольными программами?
3. Что представляет собой библиотека tkinter?
4. Что такое виджет в графическом интерфейсе?
5. Какие основные элементы интерфейса существуют в tkinter?
6. Какие менеджеры геометрии используются для размещения элементов в окне?
7. Что такое обработка событий в графическом интерфейсе?
8. Как связать действие пользователя с выполнением функции программы?
9. Почему полезно реализовывать интерфейс в виде классов?
10. Как организовать взаимодействие графического интерфейса и объектной модели программы?

ПРАКТИЧЕСКАЯ РАБОТА №17

Тема: Разработка графического интерфейса пользователя с использованием фреймворка Kivy

Цель работы

Освоение принципов разработки графических пользовательских интерфейсов с использованием фреймворка Kivy. Формирование навыков создания кроссплатформенных интерфейсов, организации структуры приложения и обработки пользовательских событий с применением объектно-ориентированного подхода.

Теоретическое обоснование

Современные программные системы всё чаще требуют разработки графических пользовательских интерфейсов, способных работать на различных устройствах и операционных системах. Такие интерфейсы должны обеспечивать удобное взаимодействие пользователя с программным приложением и поддерживать гибкость архитектуры программной системы.

Одним из популярных инструментов разработки графических интерфейсов на языке Python является фреймворк **Kivy**. Он предназначен для создания кроссплатформенных приложений, которые могут работать на различных операционных системах, включая Windows, Linux, macOS, Android и iOS.

Фреймворк Kivy основан на объектно-ориентированном подходе и использует систему виджетов для построения пользовательского интерфейса. Виджеты представляют собой элементы интерфейса, такие как кнопки, текстовые поля, панели, изображения и другие компоненты взаимодействия с пользователем.

Одной из особенностей Kivy является использование декларативного языка **KV Language**, который позволяет описывать структуру интерфейса отдельно от программной логики приложения. Это обеспечивает разделение визуальной части приложения и логики обработки данных.

Интерфейс приложения в Kivy строится на основе иерархии виджетов. Каждый виджет может содержать дочерние элементы, образуя древовидную структуру пользовательского интерфейса. Для размещения элементов используются специальные контейнеры компоновки (layouts), такие как BoxLayout, GridLayout и FloatLayout.

Важной частью разработки графических приложений является обработка событий. Когда пользователь нажимает кнопку, вводит текст или взаимодействует с элементами интерфейса, приложение должно корректно реагировать на эти действия. В Kivy обработка событий осуществляется через методы классов и привязку событий к функциям.

Использование фреймворка Kivy позволяет создавать современные интерфейсы и применять принципы объектно-ориентированного программирования при разработке программных приложений. Такой подход обеспечивает гибкость архитектуры, упрощает расширение функциональности и повышает переносимость программных систем.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm или аналогичная
Установленный фреймворк Kivy
Средства управления виртуальными окружениями Python (venv, uv)

Методические указания по выполнению работы

Перед выполнением работы необходимо ознакомиться с архитектурой фреймворка Kivy, основными типами виджетов и принципами построения пользовательского интерфейса.

В процессе выполнения работы необходимо создать простое графическое приложение, включающее несколько элементов интерфейса и реализующее обработку пользовательских действий.

Рекомендуется разделить программную логику и описание интерфейса с использованием языка KV. Также следует реализовать интерфейс приложения с применением объектно-ориентированного подхода.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка программного обеспечения из непроверенных источников.

Типовые задачи

1. Установка и настройка среды разработки

Создать виртуальное окружение Python и установить фреймворк Kivy.

Проверить корректность установки, запустив простое приложение с пустым окном.

2. Создание базового приложения

Создать простое приложение Kivy, содержащее главное окно.

Определить основной класс приложения, наследующийся от класса `App`.

3. Добавление элементов интерфейса

Добавить в окно следующие элементы интерфейса:

- текстовую метку
- кнопку
- поле ввода текста

Использовать контейнер компоновки `BoxLayout` для размещения элементов.

4. Использование KV Language

Создать файл интерфейса на языке KV.

Перенести описание структуры интерфейса из Python-кода в файл KV.

5. Обработка событий

Реализовать обработчик нажатия кнопки.

При нажатии кнопки приложение должно считывать текст из поля ввода и отображать результат на экране.

6. Организация интерфейса с использованием классов

Создать класс пользовательского интерфейса, наследующийся от одного из базовых виджетов.

Интегрировать данный класс в основное приложение.

7. Разработка интерфейса управления объектом

Создать интерфейс для управления объектом (например, агентом или программным модулем), позволяющий:

- вводить параметры объекта
- запускать действие
- отображать результат выполнения операции.

Контрольные вопросы

1. Что представляет собой фреймворк Kivy?
2. Какие преимущества имеет Kivy при разработке интерфейсов?
3. Что такое виджет в архитектуре Kivy?
4. Какие контейнеры компоновки используются для размещения элементов интерфейса?
5. Для чего используется язык KV?
6. Как осуществляется обработка событий в приложениях Kivy?
7. Как создаётся основное приложение в Kivy?
8. Почему важно разделять интерфейс и программную логику приложения?
9. Какие типы приложений можно создавать с использованием Kivy?
10. Какие преимущества даёт использование объектно-ориентированного подхода при разработке интерфейсов?

ПРАКТИЧЕСКАЯ РАБОТА №18

Тема: Модульное тестирование объектно-ориентированных программ с использованием unittest и PyTest

Цель работы

Освоение принципов модульного тестирования программного обеспечения и формирование навыков написания тестов для объектно-ориентированных программ. Получение практического опыта использования библиотек `unittest` и `pytest` для проверки корректности работы программных компонентов.

Теоретическое обоснование

При разработке программных систем важную роль играет обеспечение качества программного кода. Одним из основных инструментов контроля качества является **модульное тестирование**. Оно представляет собой процесс проверки отдельных компонентов программы (модулей, функций, классов и методов) с целью выявления ошибок на ранних этапах разработки.

Модульное тестирование особенно важно при использовании объектно-ориентированного подхода, где система состоит из множества взаимодействующих классов и объектов. Проверка корректности работы отдельных методов и классов позволяет обнаружить ошибки до интеграции компонентов в более сложную систему.

В языке Python существует несколько инструментов для реализации модульного тестирования. Одним из базовых является встроенный модуль **unittest**, который предоставляет инфраструктуру для создания тестовых классов, подготовки тестовых данных, выполнения тестов и анализа результатов.

В библиотеке `unittest` тесты организуются в виде классов, наследующихся от `unittest.TestCase`. Внутри таких классов создаются методы тестирования, которые проверяют корректность работы функций или методов программы с помощью специальных методов проверки (assertions).

Другим популярным инструментом тестирования является **pytest**. Эта библиотека предоставляет более гибкий и удобный механизм создания тестов. В отличие от `unittest`, pytest позволяет писать тесты в виде обычных функций, автоматически обнаруживает тестовые файлы и предоставляет дополнительные возможности, такие как параметризация тестов, фикстуры и подробный вывод результатов.

Модульное тестирование позволяет автоматизировать проверку корректности работы программного кода. Это особенно важно при модификации и расширении программных систем, поскольку тесты позволяют быстро выявлять ошибки, возникающие при изменении кода.

В объектно-ориентированных системах модульные тесты могут использоваться для проверки методов классов, корректности изменения состояния объектов и правильности взаимодействия между объектами.

Аппаратура и материалы

Персональный компьютер с установленной операционной системой Windows, macOS или Linux
Интерпретатор Python версии 3.9 и выше
Среда разработки: Visual Studio Code, PyCharm или аналогичная
Библиотеки `unittest` и `pytest`
Средства управления виртуальными окружениями Python (venv, uv)

Методические указания по выполнению работы

Перед выполнением работы необходимо ознакомиться с принципами модульного тестирования и структурой тестовых проектов.

В процессе выполнения работы необходимо создать несколько тестов для классов и методов объектно-ориентированной программы. Следует реализовать тесты с использованием библиотеки `unittest`, а затем реализовать аналогичные тесты с использованием `pytest`.

Особое внимание следует уделить проверке различных сценариев работы программы, включая корректные данные, граничные значения и возможные ошибки.

Требования по технике безопасности

Работа за компьютером должна выполняться в хорошо освещённом помещении.
Рекомендуется соблюдать режим работы с компьютером: не более 2 академических часов без перерыва.
Рабочее место должно быть организовано в соответствии с санитарными нормами.
Следует избегать длительного напряжения зрения при работе с экраном монитора.
Не допускается установка программного обеспечения из непроверенных источников.

Типовые задачи

1. Подготовка тестируемого модуля

Создать Python-модуль, содержащий один или несколько классов.
Например, класс `Agent`, реализующий поведение интеллектуального агента с методами:

- изменение состояния агента
- выполнение действия
- изменение уровня энергии

2. Создание тестов с использованием unittest

Создать файл тестирования.

Реализовать тестовый класс, наследующийся от `unittest.TestCase`.

Написать тесты для проверки:

- корректности создания объекта
- изменения состояния объекта
- корректности работы методов класса.

3. Проверка граничных условий

Добавить тесты, проверяющие граничные значения параметров.

Например:

- нулевой уровень энергии
- отрицательные значения параметров

- недопустимые состояния агента.

4. Создание тестов с использованием pytest

Создать набор тестов для того же модуля с использованием библиотеки `pytest`.

Тесты должны быть реализованы в виде функций.

5. Использование параметризованных тестов

Реализовать параметризованные тесты с использованием механизма `pytest.mark.parametrize`.

Проверить работу метода класса для нескольких различных наборов входных данных.

6. Анализ результатов тестирования

Запустить тесты и проанализировать результаты их выполнения.

Определить, какие ошибки могут возникать при неправильной реализации методов.

Контрольные вопросы

1. Что представляет собой модульное тестирование программного обеспечения?
2. Почему модульное тестирование важно при разработке программных систем?
3. Какие возможности предоставляет библиотека `unittest`?
4. Как организуются тесты в библиотеке `unittest`?
5. Что представляет собой библиотека `pytest`?
6. Чем отличается pytest от unittest?
7. Что такое тестовое утверждение (assertion)?
8. Какие типы ошибок могут выявляться при модульном тестировании?
9. Что такое параметризация тестов?
10. Почему модульное тестирование особенно важно при разработке объектно-ориентированных программ?

ЗАКЛЮЧЕНИЕ

Практические работы, представленные в рамках дисциплины «Объектно-ориентированное программирование», являются важной частью учебного процесса и направлены на формирование у студентов практических навыков разработки программных систем на основе объектно-ориентированной парадигмы. Их выполнение обеспечивает закрепление теоретических знаний и развитие умений проектирования, реализации и тестирования программных компонентов.

В ходе выполнения лабораторных заданий студенты последовательно осваивают ключевые элементы объектно-ориентированного программирования: создание классов и объектов, управление состоянием объектов, применение механизмов инкапсуляции, наследования и полиморфизма, использование абстрактных классов и специальных методов. Особое внимание уделяется применению принципов проектирования программного обеспечения, включая принципы SOLID и основные паттерны проектирования.

Практические задания направлены на формирование навыков разработки программных систем различной сложности, включая создание модульной архитектуры программ, реализацию взаимодействия между программными компонентами, работу с базами данных и разработку графических пользовательских интерфейсов. В процессе работы студенты также осваивают современные инструменты разработки, тестирования и сопровождения программного обеспечения.

Комплекс лабораторных работ способствует развитию алгоритмического мышления, инженерного подхода к проектированию программных систем, навыков анализа архитектуры программного обеспечения и способности применять объектно-ориентированные методы при решении прикладных задач. Практическая направленность заданий позволяет студентам получить опыт разработки программных решений, приближенных к реальным условиям профессиональной деятельности.

Выполнение лабораторных работ по дисциплине обеспечивает практико-ориентированное освоение методов объектно-ориентированного программирования и способствует формированию профессиональных компетенций ПК-1, ПК-2, ПК-3 и ПК-4, связанных с разработкой требований к программному обеспечению, проектированием программных систем, разработкой пользовательских интерфейсов и созданием компонентов программных продуктов.

УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Перечень основной литературы:

1. Болотский, А. В.
    Исследование операций и методы оптимизации Электронный ресурс / Болотский А. В., Кочеткова О. А. - Санкт-Петербург : Лань, 2020. - 116 с. - ISBN 978-5-8114-4568-4

2. Гвоздева, Т. В.
    Проектирование информационных систем: технология автоматизированного проектирования. Лабораторный практикум Электронный ресурс / Гвоздева Т. В., Баллод Б. А. - 2-е изд., стер. - Санкт-Петербург : Лань, 2020. - 156 с. - ISBN 978-5-8114-5147-0

3. Левшина, В. В.
    Применение стандартов ИСО серии 9000 Электронный ресурс / Левшина В. В. : учебное пособие. - Красноярск : СибГУ им. академика М. Ф. Решетнёва, 2020. - 150 с. - Утверждено редакционно-издательским советом университета в качестве учебного пособия для студентов магистратуры по направлению подготовки 27.04.02 «Управление качеством», магистерская программа «Проектирование и внедрение современных систем менеджмента качества (ИСО 9001:2015)», очной, очно-заочной

4. Холопов, В. А.
    Проектирование систем автоматизации и управления: Практикум Электронный ресурс / Холопов В. А. - Москва : РТУ МИРЭА, 2020. - 73 с.

Перечень дополнительной литературы:

1. Белугина, С. В.
    Разработка программных модулей программного обеспечения для компьютерных систем. Прикладное программирование Электронный ресурс / Белугина С. В. - Санкт-Петербург : Лань, 2020. - 312 с. - ISBN 978-5-8114-4496-0

2. Кузенкова, Г. В.
    WEB-технологии. Разработка сайтов Электронный ресурс / Кузенкова Г. В. : практикум. - Нижний Новгород : ННГУ им. Н. И.

Лобачевского, 2020. - 50 с. - Рекомендовано методической комиссией Института информационных технологий, математики и механики для студентов, обучающихся по направлению 09.03.03 «Прикладная информатика»

3. Стасышин, В. М.
    Разработка информационных систем и баз данных Электронный ресурс : Учебное пособие для СПО / В. М. Стасышин. - Саратов : Профобразование, 2020. - 100 c. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-5-4488-0527-1

Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине (модулю)

1. Савельев, А. О.
    Проектирование и разработка веб-приложений на основе технологий Microsoft Электронный ресурс / А. О. Савельев, А. А. Алексеев. - Проектирование и разработка веб-приложений на основе технологий Microsoft,2020-03-31. - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. - 419 c. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 2227-8397

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по организации и проведению самостоятельной работы
по дисциплине «Объектно-ориентированное программирование»
для студентов направления подготовки
09.03.01 Информатика и вычислительная техника
Направленность (профиль)
«Программное обеспечение средств вычислительной техники и автоматизированных систем»

Ставрополь 2026

Содержание

ВВЕДЕНИЕ

1. Цель и задачи освоения дисциплины (модуля)

Цель дисциплины – формирование, согласно образовательной программы, компетенций ПК-1, ПК-2, ПК-3, ПК-4 обучающегося.

Для достижения целей освоения дисциплины «Объектно-ориентированное программирование» в процессе обучения решаются следующие задачи: ознакомление с основными понятиями и принципами объектно-ориентированного программирования, включая классы, объекты, инкапсуляцию, наследование и полиморфизм; формирование представлений об объектном моделировании предметной области и преобразовании требований к программной системе в структуру классов и объектов; изучение механизмов объектно-ориентированного программирования в языке Python, включая методы классов, свойства, наследование и переопределение методов; освоение принципов разработки расширяемого и сопровождаемого программного кода; формирование понимания принципов повторного использования программных компонентов и управления зависимостями между объектами; изучение принципов объектно-ориентированного проектирования и инженерных практик разработки программного обеспечения; ознакомление с принципами SOLID и их применением при проектировании программных систем; изучение механизмов обработки исключений и организации устойчивой логики работы программ; освоение продвинутых возможностей объектно-ориентированного программирования в Python, включая использование абстрактных базовых классов, дескрипторов и метаклассов; знакомство с основными паттернами проектирования и их применением при разработке программных систем; развитие практических навыков разработки объектно-ориентированных программ на языке Python; формирование умений анализировать архитектуру программных решений, выявлять проблемы проектирования и применять методы рефакторинга для повышения качества программного кода.

2. Место дисциплины (модуля) в структуре образовательной программы

Дисциплина относится к части, формируемой участниками образовательных отношений.

3. Перечень планируемых результатов обучения по дисциплине (модулю), соотнесённых с планируемыми результатами освоения образовательной программы

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
ПК-1. Способен разрабатывать требования и проектировать программное обеспечение	ИД-1$_{ПК-1}$ понимает принципы построения архитектуры программного обеспечения и виды архитектуры программного обеспечения; понимает методы и средства проектирования программного обеспечения, методы и средства проектирования баз данных; понимает методы и средства проектирования программных интерфейсов; понимает методы и приемы формализации задач;	Понимание принципов построения архитектуры программного обеспечения и основных видов архитектуры программных систем; понимание методов и средств проектирования программного обеспечения; понимание методов и средств проектирования программных

	понимает возможности современных и перспективных средств разработки программных продуктов, технических средств.	интерфейсов; понимание методов и приёмов формализации задач при разработке программных систем; понимание возможностей современных и перспективных средств разработки программных продуктов и используемых технических средств.
	ИД-2$_{ПК-1}$ проводит анализ исполнения требований; вырабатывает варианты реализации требований; проводит оценку и обоснование рекомендуемых решений; выбирает средства реализации требований к программному обеспечению; использует существующие типовые решения и шаблоны проектирования программного обеспечения; разрабатывает технические спецификации на программные компоненты и их взаимодействие.	Проведение анализа исполнения требований; выработка вариантов реализации требований; проведение оценки и обоснование рекомендуемых решений; выбор средств реализации требований к программному обеспечению; использование существующие типовых решений и шаблонов проектирования программного обеспечения; разработка технических спецификаций на программные компоненты и их взаимодействие.
	ИД-3$_{ПК-1}$ осуществляет оценку времени и трудоемкости реализации требований к программному обеспечению; применяет методы и средства проектирования программного обеспечения, структур данных, баз данных, программных интерфейсов	Осуществление оценки времени и трудоёмкости реализации требований к программному обеспечению; применение методов и средств проектирования программного обеспечения на основе объектно-ориентированного подхода; применение методов проектирования структур данных при разработке программных компонентов; применение методов проектирования программных интерфейсов;

		использование средств разработки и инструментов программирования при создании объектно-ориентированных программных систем.
ПК-2. Способен осуществлять концептуальное, функциональное и логическое проектирование систем среднего и крупного масштаба и сложности	ИД-10$_{ПК-2}$ Применяет методы и средства проектирования программного обеспечения, структур данных, баз данных, программных интерфейсов	Применение методов и средств проектирования программного обеспечения с использованием объектно-ориентированного подхода; применение методов проектирования структур данных при разработке программных компонентов; применение методов и средств проектирования программных интерфейсов; использование современных средств разработки программного обеспечения при реализации объектно-ориентированных программных систем.
	ИД-12$_{ПК-2}$ Разрабатывает программный код на языках программирования высокого уровня	Разработка программного кода на языке программирования высокого уровня Python; применение механизмов объектно-ориентированного программирования при разработке программных компонентов; реализация классов, объектов, методов и иерархий наследования; использование принципов инкапсуляции, полиморфизма и абстракции при создании программных решений; применение современных инструментов разработки и отладки программного обеспечения.

	ИД-13$_{ПК-2}$ Применяет принципы объектно-ориентированного программирования	Применение принципов объектно-ориентированного программирования при разработке программных систем; использование механизмов инкапсуляции, наследования, полиморфизма и абстракции при проектировании программных компонентов; разработка объектных моделей предметной области; применение принципов объектно-ориентированного проектирования при построении структуры программного обеспечения; использование паттернов проектирования и методов рефакторинга для повышения качества и сопровождаемости программного кода.
ПК-3. Способен осуществлять проектирование взаимодействия пользователя с системой и проводить эвристическую оценку графического пользовательского интерфейса	ИД-1$_{ПК-3}$ понимает стандарты, регламентирующие требования к эргономике взаимодействия человек – система; разрабатывает технические требования к интерфейсной графике; использует технологии визуализации данных и цифровую обработку изображений; выполняет верстку с использованием языков разметки и языков описания стилей; программирует с использованием сценарных языков.	Понимание требований к эргономике взаимодействия человека и программной системы; понимание принципов разработки технических требований к пользовательскому интерфейсу программных приложений; использование технологий визуализации данных при разработке программных решений; применение языков разметки и описания стилей при создании пользовательских интерфейсов программных систем; использование сценарных языков программирования при разработке

		компонентов программного обеспечения.
	ИД-2$_{ПК-3}$ разрабатывает и оформляет проектную документацию на интерфейс; создает интерактивные прототипы интерфейса; проектирует стили взаимодействия пользователя с графическим пользовательским интерфейсом программного продукта,	Разработка и оформление проектной документации, описывающей взаимодействие программных компонентов с пользовательским интерфейсом; создание прототипов программных решений, демонстрирующих взаимодействие объектов и пользовательских элементов системы; проектирование стилей взаимодействия пользователя с графическим интерфейсом программного продукта на основе объектной модели программной системы; использование объектно-ориентированного подхода при организации логики обработки пользовательских действий.
	ИД-3$_{ПК-3}$ анализирует данные о действиях пользователей при работе с интерфейсом, осуществляет формальную оценку графического пользовательского интерфейса.	Анализ данных о действиях пользователей при взаимодействии с программным интерфейсом; применение методов анализа пользовательских сценариев при разработке программных систем; проведение формальной оценки графического пользовательского интерфейса программного продукта; использование объектно-ориентированного подхода для организации обработки пользовательских действий и событий в

		программных приложениях.
ПК-4. Способен разрабатывать компоненты системных программных продуктов	ИД-3$_{ПК-4}$ создает блок-схемы алгоритмов функционирования разрабатываемых программных продуктов; выполняет оценку вычислительной сложности алгоритмов функционирования разрабатываемых программных продуктов	Создание блок-схем алгоритмов функционирования программных компонентов, реализованных с использованием объектно-ориентированного подхода; описание алгоритмов работы классов и методов разрабатываемых программных систем; выполнение оценки вычислительной сложности алгоритмов функционирования программных компонентов; анализ эффективности алгоритмов, реализованных в объектно-ориентированных программных решениях.

4. Объем учебной дисциплины (модуля) и формы контроля

Объем занятий: всего: 5 з.е. 180 акад.ч.	ОФО, В акад. часах
Контактная работа:	54.00/36.00
Лекции/из них практическая подготовка	18.00/0
Лабораторных работ/из них практическая подготовка	
Практических занятий/из них практическая подготовка	36.00/36.00
Самостоятельная работа	126.00
Формы контроля	
Экзамен	нет
Зачет	нет
Зачет с оценкой	да
Курсовая работа	нет

САМОСТОЯТЕЛЬНАЯ РАБОТА КАК ВАЖНЕЙШАЯ ФОРМА УЧЕБНОГО ПРОЦЕССА

Самостоятельная работа студентов является неотъемлемой частью освоения дисциплины *«Объектно-ориентированное программирование»* и играет ключевую роль в формировании устойчивых и глубоких знаний. Современные методы машинного обучения

требуют не только теоретического понимания алгоритмов, но и уверенного владения инструментами, навыков анализа данных, интерпретации результатов и критического мышления.

В процессе самостоятельной работы студент получает возможность:

- закрепить и углубить материал, изученный на лекциях и практических занятиях;
- научиться самостоятельно находить и изучать профессиональные источники информации;
- применять изученные алгоритмы к реальным или приближённым к реальности задачам;
- развивать практические навыки программирования, работы с библиотеками Python и визуализации данных;
- осваивать различные подходы к решению одной и той же задачи, выбирать оптимальные методы, обосновывать сделанный выбор;
- готовиться к итоговому контролю, демонстрируя уверенность в базовых понятиях и алгоритмах.

Кроме того, самостоятельная работа развивает дисциплину, инициативность, умение планировать собственное обучение и адаптироваться к постоянно развивающейся предметной области. Эти навыки особенно важны в сфере искусственного интеллекта, где темпы технологических изменений требуют постоянного самообновления знаний.

Самостоятельная работа - планируемая учебная, учебно-исследовательская, научно-исследовательская работа студентов, выполняемая во внеаудиторное (аудиторное) время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия (при частичном непосредственном участии преподавателя, оставляющем ведущую роль за работой студентов).

Самостоятельная работа студентов в ВУЗе является важным видом учебной и научной деятельности студента. Самостоятельная работа студентов играет значительную роль в рейтинговой технологии обучения. Обучение в ВУЗе включает в себя две, практически одинаковые по объему и взаимовлиянию части – процесса обучения и процесса самообучения. Поэтому СРС должна стать эффективной и целенаправленной работой студента.

Концепцией модернизации российского образования определены основные задачи профессионального образования - "подготовка квалифицированного работника соответствующего уровня и профиля, конкурентоспособного на рынке труда, компетентного, ответственного, свободно владеющего своей профессией и ориентированного в смежных областях деятельности, способного к эффективной работе по специальности на уровне мировых стандартов, готового к постоянному профессиональному росту, социальной и профессиональной мобильности".

Решение этих задач невозможно без повышения роли самостоятельной работы студентов над учебным материалом, усиления ответственности преподавателей за развитие навыков самостоятельной работы, за стимулирование профессионального роста студентов, воспитание творческой активности и инициативы.

К современному специалисту общество предъявляет достаточно широкий перечень требований, среди которых немаловажное значение имеет наличие у выпускников определенных способностей и умения самостоятельно добывать знания из различных источников, систематизировать полученную информацию, давать оценку конкретной финансовой ситуации. Формирование такого умения происходит в течение всего периода обучения через участие студентов в лабораторных занятиях, выполнение контрольных заданий и тестов, написание курсовых и выпускных квалификационных работ. При этом самостоятельная работа студентов играет решающую роль в ходе всего учебного процесса.

Формы самостоятельной работы студентов разнообразны. По данной дисциплине «Объектно-ориентированное программирование» предусмотрены следующие:

Использование материала учебно-методического комплекса дисциплины

На первом этапе необходимо ознакомиться с обязательным минимумом содержания основной образовательной программы по изучаемой дисциплине. Далее необходимо изучить рабочую программу дисциплины, в которой рассмотрено содержание тем дисциплины лекционного курса, взаимосвязь тем лекций с лабораторных занятий, темы и виды самостоятельной работы. По каждому виду самостоятельной работы предусмотрены определённые формы отчетности, представленные в учебной программе по дисциплине «Объектно-ориентированное программирование».

Работа с литературой

Для успешного освоения дисциплины, необходимо самостоятельно детально изучить представленные темы по рекомендуемым источникам информации, которые представлены в учебной программе.

Самостоятельная работа приобщает студентов к научному творчеству, поиску и решению актуальных современных проблем.

ЦЕЛИ И ОСНОВНЫЕ ЗАДАЧИ СРС

Цели самостоятельной работы:

- Углубление теоретических знаний, полученных на лекциях;
- Закрепление практических навыков, полученных на семинарах;
- Развитие умений решать прикладные задачи машинного обучения на практике;
- Формирование критического подхода к выбору моделей, параметров и метрик.

Формы самостоятельной работы:

- Изучение рекомендованных источников и материалов лекций;
- Анализ и решение предложенных задач;
- Выполнение индивидуальных практических заданий в Jupyter Notebook;
- Подготовка мини-отчётов и интерпретация полученных результатов;
- Самостоятельное повторение материала и подготовка к тестированию.

Ведущая цель организации и осуществления СРС должна совпадать с целью обучения студента – подготовкой бакалавра с высшим образованием. При организации СРС важным и необходимым условием становятся формирование умения самостоятельной работы для приобретения знаний, навыков и возможности организации учебной и научной деятельности.

Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

Задачами СРС являются:

- систематизация и закрепление полученных теоретических знаний и практических умений студентов;
- углубление и расширение теоретических знаний;
- формирование умений использовать нормативную, правовую, справочную документацию и специальную литературу;

- развитие познавательных способностей и активности студентов: творческой инициативы, самостоятельности, ответственности и организованности;
- формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развитие исследовательских умений;
- использование материала, собранного и полученного в ходе самостоятельных занятий на семинарах, на лабораторных занятиях, при написании курсовых и выпускной квалификационной работ, для эффективной подготовки к итоговым зачетам и экзаменам.

ВИДЫ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

В образовательном процессе высшего профессионального образовательного учреждения выделяется два вида самостоятельной работы – аудиторная, под руководством преподавателя, и внеаудиторная. Тесная взаимосвязь этих видов работ предусматривает дифференциацию и эффективность результатов ее выполнения и зависит от организации, содержания, логики учебного процесса (межпредметных связей, перспективных знаний и др.):

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.

Основными видами самостоятельной работы студентов без участия преподавателя по данной дисциплине являются:

- формирование и усвоение содержания конспекта лекций на базе рекомендованной лектором учебной литературы, включая информационные образовательные ресурсы (электронные учебники, электронные библиотеки и др.);
- подготовка к лабораторным работам, их оформление.

Основными видами самостоятельной работы студентов с участием преподавателей являются:

- текущие консультации;
- прием и защита лабораторных работ (во время проведения л/р).

ОРГАНИЗАЦИЯ СРС

Методика организации самостоятельной работы студентов зависит от структуры, характера и особенностей дисциплины «Объектно-ориентированное программирование», объема часов на ее изучение, вида заданий для самостоятельной работы студентов, индивидуальных качеств студентов и условий учебной деятельности.

Процесс организации самостоятельной работы студентов включает в себя следующие этапы:

- подготовительный (определение целей, составление программы, подготовка методического обеспечения, подготовка оборудования);
- основной (реализация программы, использование приемов поиска информации, усвоения, переработки, применения, передачи знаний, фиксирование результатов, самоорганизация процесса работы);
- заключительный (оценка значимости и анализ результатов, их систематизация, оценка эффективности программы и приемов работы, выводы о направлениях оптимизации труда).

Рекомендуемые темы и задания для самостоятельной проработки:

1. Основы объектно-ориентированного программирования

- Изучить понятия класса, объекта, атрибута и метода.

- Задание: реализовать класс «Студент» с атрибутами имени, группы и среднего балла и методами для вывода информации и изменения среднего балла.

2. Конструкторы и инициализация объектов
- Изучить назначение конструктора и метода __init__.
- Задание: разработать класс «Книга», который инициализирует название, автора и год издания.

3. Атрибуты класса и атрибуты экземпляра
- Изучить различие между атрибутами класса и экземпляра.
- Задание: создать класс «Автомобиль», содержащий общий атрибут количества колес и индивидуальные характеристики автомобиля.

4. Инкапсуляция
- Изучить способы ограничения доступа к атрибутам класса.
- Задание: разработать класс «Банковский счет» с приватным балансом и методами пополнения и снятия средств.

5. Свойства и декоратор @property
- Изучить использование свойств для управления доступом к данным.
- Задание: реализовать класс «Температура», в котором значение хранится в градусах Цельсия и автоматически переводится в Фаренгейты.

6. Наследование
- Изучить механизм наследования классов.
- Задание: создать базовый класс «Транспорт» и производные классы «Автомобиль» и «Велосипед».

7. Переопределение методов
- Изучить механизм переопределения методов.
- Задание: реализовать базовый класс «Фигура» и классы «Круг» и «Прямоугольник» с собственными методами вычисления площади.

8. Полиморфизм
- Изучить принцип полиморфизма и динамического связывания методов.
- Задание: реализовать несколько классов с одинаковым методом draw() и вызвать их в цикле.

9. Абстрактные классы
- Изучить модуль abc и абстрактные базовые классы.
- Задание: создать абстрактный класс «Геометрическая фигура» с абстрактным методом вычисления площади.

10. Множественное наследование
- Изучить особенности множественного наследования и MRO.
- Задание: разработать иерархию классов с двумя родительскими классами и исследовать порядок разрешения методов.

11. Композиция объектов
- Изучить принцип композиции.
- Задание: реализовать класс «Компьютер», содержащий объекты «Процессор», «Память» и «Диск».

12. Агрегация объектов
- Изучить отличие композиции от агрегации.
- Задание: реализовать класс «Курс», содержащий список студентов.

13. Специальные методы Python
- Изучить магические методы (__str__, __repr__).
- Задание: реализовать класс с переопределением этих методов для удобного отображения объектов.

14. Сравнение объектов
- Изучить методы __eq__, __lt__.
- Задание: реализовать класс «Товар» с возможностью сравнения по цене.

15. Итераторы
- Изучить методы __iter__ и __next__.
- Задание: реализовать собственный класс-итератор.

16. Генераторы
- Изучить оператор yield.
- Задание: написать генератор для последовательности чисел Фибоначчи.

17. Менеджеры контекста
- Изучить методы __enter__ и __exit__.
- Задание: создать менеджер контекста для логирования операций.

18. Принцип DRY
- Изучить способы устранения дублирования кода.
- Задание: провести рефакторинг повторяющихся функций.

19. Принцип SOLID
- Изучить пять принципов SOLID.
- Задание: определить нарушения SOLID в предложенном коде и исправить их.

20. Паттерн Singleton
- Изучить назначение паттерна Singleton.
- Задание: реализовать класс с единственным экземпляром.

21. Паттерн Factory Method
- Изучить фабричный метод.
- Задание: реализовать фабрику создания различных типов объектов.

22. Паттерн Strategy
- Изучить стратегию выбора алгоритма.
- Задание: реализовать систему сортировки с несколькими алгоритмами.

23. Паттерн Observer
- Изучить механизм подписки на события.
- Задание: реализовать систему уведомлений.

24. Паттерн Decorator
- Изучить динамическое расширение поведения объекта.
- Задание: реализовать декоратор для добавления функциональности.

25. UML-диаграммы классов
- Изучить структуру UML.
- Задание: построить UML-диаграмму для небольшой объектной системы.

26. Архитектура программных систем
- Изучить уровни архитектуры приложений.
- Задание: спроектировать архитектуру консольного приложения.

27. Организация модулей Python
- Изучить структуру пакетов.
- Задание: разбить проект на несколько модулей.

28. Виртуальные окружения Python
- Изучить создание и использование виртуальных окружений.
- Задание: создать виртуальное окружение и установить зависимости.

29. Управление зависимостями
- Изучить инструменты uv и pip.
- Задание: создать файл зависимостей проекта.

30. Линтеры и форматирование
- Изучить flake8, ruff и black.
- Задание: проверить и автоматически отформатировать проект.

31. Инструменты pre-commit
- Изучить автоматические проверки перед коммитом.
- Задание: настроить pre-commit для Python-проекта.

32. Работа с SQLite

- Изучить основы реляционных баз данных.
- Задание: создать базу данных и таблицу пользователей.

33. SQL-запросы
- Изучить команды SELECT, INSERT, UPDATE.
- Задание: реализовать операции добавления и выборки данных.

34. Работа SQLite в Python
- Изучить модуль sqlite3.
- Задание: написать программу для хранения записей в базе.

35. Объектное представление данных
- Изучить связь объектов и таблиц.
- Задание: реализовать класс для работы с записями базы данных.

36. Основы GUI
- Изучить принципы графических интерфейсов.
- Задание: создать окно приложения.

37. Интерфейс на tkinter
- Изучить основные виджеты.
- Задание: реализовать форму ввода данных.

38. События в GUI
- Изучить обработчики событий.
- Задание: реализовать обработку нажатия кнопки.

39. Интерфейс на Kivy
- Изучить основы фреймворка Kivy.
- Задание: создать простое мобильное приложение.

40. Архитектура GUI-приложений
- Изучить разделение модели и интерфейса.
- Задание: реализовать приложение с разделением логики и интерфейса.

41. Модульное тестирование
- Изучить принципы unit-тестирования.
- Задание: написать тесты для класса.

42. Тестирование с unittest
- Изучить структуру тестов.
- Задание: создать тестовый набор.

43. Тестирование с pytest
- Изучить фикстуры и параметризацию.
- Задание: реализовать несколько тестов для функций.

44. Рефакторинг кода
- Изучить методы улучшения структуры программы.
- Задание: оптимизировать существующий код.

45. Анализ алгоритмов
- Изучить оценку сложности алгоритмов.
- Задание: определить сложность нескольких алгоритмов.

46. Оптимизация алгоритмов
- Изучить способы повышения эффективности.
- Задание: переписать алгоритм с меньшей сложностью.

47. Блок-схемы алгоритмов
- Изучить графическое представление алгоритмов.
- Задание: построить блок-схему программы.

48. Проектирование программных компонентов
- Изучить принципы модульного проектирования.
- Задание: разработать структуру проекта.

49. Проектирование объектной модели
- Изучить методы моделирования предметной области.

- Задание: создать классы для системы управления библиотекой.

50. Мини-проект
- Применить изученные принципы ООП.
- Задание: разработать небольшое приложение на Python с использованием классов, базы данных и графического интерфейса.

Типовые задания для СРС:

Общий подход к оцениванию:

- Каждое задание оценивается по **10-балльной шкале**;
- Общая оценка по блоку самостоятельной работы может рассчитываться как среднее значение или сумма (по усмотрению преподавателя);
- Возможно использование **частичных баллов (0.5, 1 и т.д.)** при частично выполненных пунктах задания;
- При копировании без осмысления, неработающем коде или отсутствующей интерпретации баллы **не начисляются**.

№	Задание	Критерии оценки (макс. 10 баллов)
1	Реализовать класс «Студент» с атрибутами и методами для вывода информации	Корректная структура класса – 4; корректная работа методов – 3; тестирование и демонстрация работы – 3
2	Реализовать класс «Книга» с конструктором и методами доступа	Использование конструктора – 4; корректная работа методов – 3; демонстрация работы – 3
3	Создать класс «Автомобиль» с атрибутами класса и экземпляра	Корректное использование атрибутов – 4; реализация методов – 3; демонстрация работы – 3
4	Реализовать класс «Банковский счет» с инкапсуляцией данных	Приватные атрибуты – 4; методы работы с балансом – 3; тестирование – 3
5	Реализовать свойства с использованием @property	Корректная реализация свойства – 4; корректность вычислений – 3; демонстрация – 3
6	Создать базовый класс «Транспорт» и два производных класса	Корректная иерархия классов – 4; использование наследования – 3; демонстрация работы – 3
7	Реализовать вычисление площади фигур через наследование	Использование базового класса – 4; корректные формулы – 3; тестирование – 3
8	Реализовать полиморфизм через единый метод	Реализация методов – 4; корректное использование полиморфизма – 3; демонстрация – 3
9	Реализовать абстрактный класс геометрической фигуры	Использование abc – 4; реализация методов – 3; тестирование – 3
10	Реализовать пример множественного наследования	Корректная структура – 4; демонстрация MRO – 3; тестирование – 3

11	Реализовать композицию объектов (класс «Компьютер»)	Реализация вложенных объектов – 4; корректное взаимодействие – 3; демонстрация – 3
12	Реализовать агрегацию объектов (курс и студенты)	Правильная структура классов – 4; корректная работа коллекций – 3; демонстрация – 3
13	Реализовать магические методы __str__ и __repr__	Корректная реализация методов – 4; информативность вывода – 3; тестирование – 3
14	Реализовать сравнение объектов через __eq__	Реализация метода – 4; корректная логика сравнения – 3; тестирование – 3
15	Реализовать собственный итератор	Реализация __iter__ – 4; реализация __next__ – 3; демонстрация работы – 3
16	Реализовать генератор последовательности Фибоначчи	Использование yield – 4; корректность алгоритма – 3; демонстрация – 3
17	Реализовать контекстный менеджер для логирования	Реализация __enter__ – 4; реализация __exit__ – 3; демонстрация – 3
18	Найти и устранить дублирование кода (DRY)	Анализ кода – 4; устранение повторений – 3; объяснение решений – 3
19	Проанализировать пример нарушения SOLID	Выявление нарушений – 4; предложение решения – 3; аргументация – 3
20	Реализовать паттерн Singleton	Корректная реализация – 4; контроль экземпляров – 3; тестирование – 3
21	Реализовать паттерн Factory Method	Реализация фабрики – 4; создание объектов – 3; демонстрация – 3
22	Реализовать паттерн Strategy	Реализация стратегий – 4; переключение алгоритмов – 3; демонстрация – 3
23	Реализовать паттерн Observer	Реализация подписки – 4; уведомления – 3; демонстрация – 3
24	Реализовать паттерн Decorator	Реализация декоратора – 4; расширение функциональности – 3; демонстрация – 3
25	Построить UML-диаграмму системы классов	Корректность структуры – 4; полнота элементов – 3; оформление – 3
26	Спроектировать архитектуру небольшого приложения	Архитектура – 4; обоснование решений – 3; описание модулей – 3
27	Разделить проект Python на модули	Структура проекта – 4; корректный импорт – 3; демонстрация – 3
28	Создать виртуальное окружение Python	Создание окружения – 4; установка

		зависимостей – 3; демонстрация – 3
29	Подготовить файл зависимостей проекта	Корректность зависимостей – 4; установка – 3; тестирование – 3
30	Проверить код с помощью линтера	Использование линтера – 4; исправление ошибок – 3; демонстрация – 3
31	Настроить pre-commit проверки	Настройка – 4; интеграция с проектом – 3; демонстрация – 3
32	Создать базу данных SQLite	Создание БД – 4; создание таблиц – 3; демонстрация – 3
33	Выполнить SQL-запросы к базе данных	Корректность запросов – 4; результаты – 3; демонстрация – 3
34	Реализовать взаимодействие Python и SQLite	Использование sqlite3 – 4; корректные операции – 3; демонстрация – 3
35	Реализовать объектную модель данных	Проектирование классов – 4; связь с БД – 3; демонстрация – 3
36	Создать простое окно приложения	Использование tkinter – 4; корректность интерфейса – 3; демонстрация – 3
37	Реализовать форму ввода данных	Реализация виджетов – 4; обработка ввода – 3; демонстрация – 3
38	Реализовать обработчик события	Реализация обработчика – 4; корректность работы – 3; демонстрация – 3
39	Реализовать интерфейс на Kivy	Использование фреймворка – 4; корректность интерфейса – 3; демонстрация – 3
40	Реализовать приложение с GUI и логикой	Архитектура приложения – 4; взаимодействие компонентов – 3; демонстрация – 3
41	Написать unit-тесты для класса	Реализация тестов – 4; корректность проверок – 3; демонстрация – 3
42	Создать набор тестов с unittest	Структура тестов – 4; корректность тестирования – 3; демонстрация – 3
43	Реализовать тестирование с pytest	Использование pytest – 4; корректность тестов – 3; демонстрация – 3
44	Выполнить рефакторинг программного кода	Анализ кода – 4; улучшение структуры – 3; демонстрация – 3
45	Оценить сложность алгоритмов	Анализ алгоритмов – 4; корректность оценки – 3; объяснение – 3
46	Оптимизировать алгоритм	Улучшение алгоритма – 4; корректность решения – 3; объяснение – 3
47	Построить блок-схему алгоритма	Корректность схемы – 4; полнота – 3; оформление – 3

48	Разработать структуру программного проекта	Архитектура – 4; модульность – 3; обоснование – 3
49	Смоделировать объектную систему библиотеки	Проектирование классов – 4; корректность связей – 3; описание – 3
50	Разработать мини-проект на Python	Архитектура решения – 4; реализация функциональности – 3; демонстрация – 3

Темы для творческих заданий

1. **Менеджер задач**
   Разработать приложение с графическим интерфейсом для управления списком задач. Интерфейс должен взаимодействовать с объектной моделью задач (класс Task, список задач, методы добавления и удаления).
2. **Каталог книг**
   Создать GUI-приложение для хранения информации о книгах. Интерфейс должен позволять добавлять, удалять и просматривать объекты класса Book.
3. **Учет студентов**
   Реализовать систему учета студентов с использованием классов Student и Course. Интерфейс должен отображать список студентов и позволять добавлять новых.
4. **Калькулятор расходов**
   Разработать приложение для учета личных расходов. Интерфейс должен взаимодействовать с объектами класса Expense.
5. **Музыкальная библиотека**
   Создать приложение для управления музыкальной коллекцией. Интерфейс должен работать с объектами класса Track.
6. **Система заметок**
   Разработать интерфейс для создания и редактирования заметок, взаимодействующий с объектной моделью Note.
7. **Контакты**
   Создать адресную книгу с графическим интерфейсом. Интерфейс должен управлять объектами класса Contact.
8. **Список покупок**
   Реализовать приложение, где пользователь может добавлять товары в список покупок. Каждый элемент представлен объектом класса Item.
9. **Простая библиотечная система**
   Разработать GUI для выдачи и возврата книг, используя объектную модель Library, Book и Reader.
10. **Планировщик занятий**
    Создать приложение для управления расписанием занятий с использованием классов Schedule и Lesson.
11. **Каталог фильмов**
    Разработать GUI для просмотра и редактирования списка фильмов, реализованных как объекты класса Movie.
12. **Дневник тренировок**
    Создать приложение для учета тренировок, где каждая запись представлена объектом класса Workout.
13. **Учебный журнал**
    Разработать интерфейс для просмотра и изменения оценок студентов, используя объекты Student и Grade.

14. **Менеджер проектов**
    Создать приложение для управления проектами и задачами с использованием объектной модели Project и Task.
15. **Трекер привычек**
    Разработать систему отслеживания привычек с объектами Habit.
16. **Система бронирования**
    Создать приложение для бронирования ресурсов (например, комнат) с использованием классов Reservation и Room.
17. **Каталог рецептов**
    Разработать приложение для хранения рецептов, где каждый рецепт представлен объектом Recipe.
18. **Простой чат-интерфейс**
    Создать интерфейс для отправки сообщений между пользователями, используя объектную модель Message.
19. **Инвентаризация товаров**
    Разработать приложение для учета товаров на складе с использованием класса Product.
20. **Файловый менеджер**
    Создать GUI-приложение для просмотра файлов и папок, используя объектную модель FileItem.
21. **Система регистрации пользователей**
    Разработать форму регистрации и хранения данных пользователей в объектах класса User.
22. **Приложение для заметок с категориями**
    Реализовать интерфейс, взаимодействующий с объектами Note и Category.
23. **Менеджер задач с приоритетами**
    Добавить поддержку приоритетов задач в объектной модели Task.
24. **Календарь событий**
    Создать интерфейс календаря для управления событиями (класс Event).
25. **Каталог игр**
    Разработать приложение для хранения информации о видеоиграх.
26. **Список фильмов для просмотра**
    Реализовать систему управления списком фильмов.
27. **Система учета книг в личной библиотеке**
    Разработать интерфейс управления книгами.
28. **Менеджер заметок с поиском**
    Добавить функцию поиска по объектам Note.
29. **Система комментариев**
    Создать интерфейс добавления комментариев к объектам Post.
30. **Приложение учета времени**
    Разработать систему учета рабочего времени.
31. **Система рейтинга фильмов**
    Добавить возможность выставления оценок.
32. **Приложение управления списком дел**
    Создать приложение с отметкой выполненных задач.
33. **Учебный план**
    Создать приложение управления учебными курсами.
34. **Каталог программного обеспечения**
    Реализовать систему хранения данных о программах.
35. **Система регистрации на мероприятия**
    Создать интерфейс для регистрации пользователей.

36. **Приложение учета автомобилей**
    Создать систему хранения данных об автомобилях.
37. **Система учета сотрудников**
    Реализовать интерфейс управления сотрудниками.
38. **Каталог фотографий**
    Создать приложение управления изображениями.
39. **Система управления проектами**
    Реализовать интерфейс взаимодействия с проектной моделью.
40. **Менеджер паролей**
    Создать интерфейс хранения учетных данных.
41. **Система учета книг, выданных друзьям**
    Разработать приложение для отслеживания выдачи книг.
42. **Приложение учета фильмов, просмотренных пользователем**
    Добавить статус просмотра.
43. **Каталог учебных материалов**
    Реализовать систему управления учебными ресурсами.
44. **Система управления библиотекой музыки**
    Создать приложение для работы с объектами Track.
45. **Приложение учета домашних животных**
    Разработать систему хранения данных о питомцах.
46. **Система управления клиентами**
    Создать приложение CRM-типа.
47. **Система учета посещаемости занятий**
    Реализовать учет посещений студентов.
48. **Каталог видеокурсов**
    Разработать интерфейс управления курсами.
49. **Система учета личных целей**
    Создать приложение для отслеживания целей.
50. **Мини-проект GUI-приложения**
    Разработать собственное приложение, где интерфейс полностью взаимодействует с объектной моделью и реализует базовые принципы ООП.

Деятельность студентов по формированию и развитию навыков учебной самостоятельной работы

В процессе самостоятельной работы студент приобретает навыки самоорганизации, самоконтроля, самоуправления, саморефлексии и становится активным самостоятельным субъектом учебной деятельности.

Выполняя самостоятельную работу под контролем преподавателя студент должен:

– освоить минимум содержания, выносимый на самостоятельную работу студентов и предложенный преподавателем в соответствии с образовательными стандартами высшего образования;

– планировать самостоятельную работу в соответствии с графиком самостоятельной работы, предложенным преподавателем;

– самостоятельную работу студент должен осуществлять в организационных формах, предусмотренных учебным планом и рабочей программой преподавателя;

– выполнять самостоятельную работу и отчитываться по ее результатам в соответствии с графиком представления результатов, видами и сроками отчетности по самостоятельной работе студентов.

студент может:

сверх предложенного преподавателем (при обосновании и согласовании с ним) и минимума обязательного содержания, определяемого программой по данной дисциплине:

– самостоятельно определять уровень (глубину) проработки содержания материала;

– предлагать темы и вопросы для самостоятельной проработки;

– в рамках общего графика выполнения самостоятельной работы предлагать обоснованный индивидуальный график выполнения и отчетности по результатам самостоятельной работы;

– предлагать свои варианты организационных форм самостоятельной работы;

– использовать для самостоятельной работы методические пособия, учебные пособия, разработки сверх предложенного преподавателем перечня;

– использовать не только контроль, но и самоконтроль результатов самостоятельной работы в соответствии с методами самоконтроля, предложенными преподавателем или выбранными самостоятельно.

Самостоятельная работа студентов должна оказывать важное влияние на формирование личности будущего бакалавра, она планируется студентом самостоятельно. Каждый студент самостоятельно определяет режим своей работы и меру труда, затрачиваемого на овладение учебным содержанием по каждой дисциплине. Он выполняет внеаудиторную работу по личному индивидуальному плану, в зависимости от его подготовки, времени и других условий.

Вопросы для проверки СРС

1. Что понимается под объектно-ориентированным программированием?
2. Какие основные принципы лежат в основе объектно-ориентированного программирования?
3. Что такое класс в объектно-ориентированном программировании?
4. Что такое объект и чем он отличается от класса?
5. Что такое атрибут класса?
6. Что такое атрибут экземпляра класса?
7. Что такое метод класса?
8. Что такое конструктор класса?
9. Для чего используется метод __init__ в Python?
10. Что такое инкапсуляция и зачем она используется?
11. Как реализуется инкапсуляция в языке Python?
12. Что такое наследование?
13. Какие преимущества дает использование наследования?
14. Что такое полиморфизм?
15. Что такое абстракция?
16. Что такое абстрактный класс?
17. Что такое абстрактный метод?
18. Для чего используется модуль abc в Python?
19. Что такое множественное наследование?
20. Что такое MRO (Method Resolution Order)?
21. Как работает функция super()?
22. Что такое композиция в объектно-ориентированном программировании?
23. Чем композиция отличается от наследования?
24. Что такое агрегация объектов?
25. Какие виды отношений между объектами существуют?
26. Что такое магические методы Python?
27. Для чего используются методы __str__ и __repr__?
28. Для чего используется метод __eq__?
29. Что такое перегрузка операторов?

30. Какие методы используются для перегрузки операторов в Python?
31. Что такое итератор?
32. Какие методы должен реализовывать итератор?
33. Что такое генератор?
34. Чем генератор отличается от обычной функции?
35. Для чего используется оператор yield?
36. Что такое контекстный менеджер?
37. Какие методы реализует контекстный менеджер?
38. Для чего используется конструкция with?
39. Что такое декоратор в Python?
40. Какие виды декораторов существуют?
41. Для чего используется декоратор @property?
42. Что такое статический метод?
43. Что такое метод класса?
44. Чем статический метод отличается от метода класса?
45. Какие преимущества дает использование классов в программировании?
46. Что такое модуль Python?
47. Что такое пакет Python?
48. Как организуется структура проекта Python?
49. Что такое виртуальное окружение Python?
50. Для чего используются виртуальные окружения?
51. Какие инструменты используются для управления зависимостями Python-проекта?
52. Что такое линтер?
53. Для чего используется flake8?
54. Для чего используется ruff?
55. Для чего используется black?
56. Для чего используется isort?
57. Что такое pre-commit?
58. Для чего используется инструмент uv?
59. Что такое принцип DRY?
60. Почему важно избегать дублирования кода?
61. Что означает аббревиатура SOLID?
62. В чем заключается принцип единственной ответственности?
63. В чем заключается принцип открытости/закрытости?
64. В чем заключается принцип подстановки Лисков?
65. В чем заключается принцип разделения интерфейсов?
66. В чем заключается принцип инверсии зависимостей?
67. Что такое паттерн проектирования?
68. Какие группы паттернов проектирования существуют?
69. Для чего используется паттерн Singleton?
70. Для чего используется паттерн Factory Method?
71. Для чего используется паттерн Strategy?
72. Для чего используется паттерн Observer?
73. Для чего используется паттерн Decorator?
74. Для чего используется паттерн Adapter?
75. Для чего используется паттерн Facade?
76. Что такое архитектура программного обеспечения?
77. Какие уровни архитектуры программных систем существуют?
78. Что такое модульность программного обеспечения?
79. Что такое рефакторинг программного кода?
80. Какие цели преследует рефакторинг?
81. Что такое UML?

82. Для чего используется UML при проектировании программных систем?
83. Что отображает UML-диаграмма классов?
84. Что отображает UML-диаграмма последовательностей?
85. Что отображает UML-диаграмма деятельности?
86. Какие основные элементы содержит UML-диаграмма классов?
87. Что такое графический пользовательский интерфейс (GUI)?
88. Какие элементы входят в GUI?
89. Что такое виджет?
90. Что такое обработчик событий?
91. Что такое событийно-ориентированное программирование?
92. Какие преимущества дает использование GUI?
93. Что представляет собой библиотека tkinter?
94. Какие основные виджеты существуют в tkinter?
95. Что представляет собой фреймворк Kivy?
96. В чем особенности разработки GUI-приложений?
97. Как организуется взаимодействие интерфейса и объектной модели программы?
98. Что такое модель данных приложения?
99. Почему важно разделять интерфейс и бизнес-логику приложения?
100. Какие архитектурные шаблоны используются при разработке GUI?
101. Что такое паттерн MVC?
102. Какие компоненты входят в MVC?
103. Как взаимодействуют модель, представление и контроллер?
104. Что такое база данных SQLite?
105. Какие преимущества имеет SQLite?
106. Какой модуль Python используется для работы с SQLite?
107. Что такое SQL?
108. Какие основные SQL-команды используются для работы с базой данных?
109. Что такое первичный ключ в базе данных?
110. Что такое курсор базы данных?
111. Что такое тестирование программного обеспечения?
112. Что такое модульное тестирование?
113. Для чего используется модуль unittest?
114. Для чего используется pytest?
115. Что такое тестовый сценарий?
116. Что такое тестовый набор (test suite)?
117. Что такое алгоритм?
118. Что такое блок-схема алгоритма?
119. Что такое вычислительная сложность алгоритма?
120. Почему важно анализировать сложность алгоритмов при разработке программных систем?

ОБЩИЕ РЕКОМЕНДАЦИИ ПО ОРГАНИЗАЦИИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Основной формой самостоятельной работы студента является изучение конспекта лекций, их дополнение, рекомендованной литературы, активное участие на лабораторных занятиях. Но для успешной учебной деятельности, ее интенсификации, необходимо учитывать следующие субъективные факторы:

1. Знание школьного программного материала, наличие прочной системы зияний, необходимой для усвоения основных вузовских курсов. Это особенно важно для математических дисциплин. Необходимо отличать пробелы в знаниях, затрудняющие усвоение нового материала, от малых способностей. Затратив силы на преодоление этих

пробелов, студент обеспечит себе нормальную успеваемость и поверит в свои способности.

2. Наличие умений, навыков умственного труда:

а) умение конспектировать на лекции и при работе с книгой;

б) владение логическими операциями: сравнение, анализ, синтез, обобщение, определение понятий, правила систематизации и классификации.

3. Специфика познавательных психических процессов: внимание, память, речь, наблюдательность, интеллект и мышление. Слабое развитие каждого из них становится серьезным препятствием в учебе.

4. Хорошая работоспособность, которая обеспечивается нормальным физическим состоянием. Ведь серьезное учение — это большой многосторонний и разнообразный труд. Результат обучения оценивается не количеством сообщаемой информации, а качеством ее усвоения, умением ее использовать и развитием у себя способности к дальнейшему самостоятельному образованию.

5. Соответствие избранной деятельности, профессии индивидуальным способностям. Необходимо выработать у себя умение саморегулировать свое эмоциональное состояние и устранять обстоятельства, нарушающие деловой настрой, мешающие намеченной работе.

6. Овладение оптимальным стилем работы, обеспечивающим успех в деятельности. Чередование труда и пауз в работе, периоды отдыха, индивидуально обоснованная норма продолжительности сна, предпочтение вечерних или утренних занятий, стрессоустойчивость на экзаменах и особенности подготовки к ним,

7. Уровень требований к себе, определяемый сложившейся самооценкой.

Адекватная оценка знаний, достоинств, недостатков - важная составляющая самоорганизации человека, без нее невозможна успешная работа по управлению своим поведением, деятельностью.

Одна из основных особенностей обучения в высшей школе заключается в том, что постоянный внешний контроль заменяется самоконтролем, активная роль в обучении принадлежит уже не столько преподавателю, сколько студенту.

Зная основные методы научной организации умственного труда, можно при наименьших затратах времени, средств и трудовых усилий достичь наилучших результатов.

Эффективность усвоения поступающей информации зависит от работоспособности человека в тот или иной момент его деятельности.

Работоспособность - способность человека к труду с высокой степенью напряженности в течение определенного времени. Различают внутренние и внешние факторы работоспособности.

К внутренним факторам работоспособности относятся интеллектуальные особенности, воля, состояние здоровья.

К внешним:

- организация рабочего места, режим труда и отдыха;
- уровень организации труда - умение получить справку и пользоваться информацией;
- величина умственной нагрузки.

Выдающийся русский физиолог Н. Е. Введенский выделил следующие условия продуктивности умственной деятельности:

- во всякий труд нужно входить постепенно;
- мерность и ритм работы. Разным людям присущ более или менее разный темп работы;
- привычная последовательность и систематичность деятельности;
- правильное чередование труда и отдыха.

Отдых не предполагает обязательного полного бездействия со стороны человека, он может быть достигнут простой переменой дела. В течение дня работоспособность изменяется. Наиболее плодотворным является *утреннее время (с 8 до 14 часов),* причем максимальная работоспособность приходится на период с 10 до 13 часов, затем *послеобеденное* - (с 16 до 19 часов) и *вечернее* (с 20 до 24 часов). Очень трудный для понимания материал лучше изучать в начале каждого отрезка времени (лучше всего утреннего) после хорошего отдыха. Через 1-1,5 часа нужны перерывы по 10 - 15 мин, через 3 - 4 часа работы отдых должен быть продолжительным - около часа.

Составной частью научной организации умственного труда является овладение техникой умственного труда.

Время, которым располагает студент для выполнения учебного плана, складывается из двух составляющих: одна из них - это аудиторная работа в вузе по расписанию занятий, другая - внеаудиторная самостоятельная работа. Задания и материалы для самостоятельной работы выдаются во время учебных занятий по расписанию, на этих же занятиях преподаватель осуществляет контроль за самостоятельной работой, а также оказывает помощь студентам по правильной организации работы.

Самостоятельные занятия потребуют интенсивного умственного труда, который необходимо не только правильно организовать, но и стимулировать. При этом очень важно уметь поддерживать устойчивое внимание к изучаемому материалу. Выработка внимания требует значительных волевых усилий. Именно поэтому, если студент замечает, что он часто отвлекается во время самостоятельных занятий, ему надо заставить себя сосредоточиться. Подобную процедуру необходимо проделывать постоянно, так как это является тренировкой внимания. Устойчивое внимание появляется тогда, когда человек относится к делу с интересом.

Следует правильно организовать свои занятия по времени: 50 минут - работа, 5-10 минут - перерыв; после 3 часов работы перерыв - 20-25 минут. Иначе нарастающее утомление повлечет неустойчивость внимания. Очень существенным фактором, влияющим на повышение умственной работоспособности, являются систематические занятия физической культурой. Организация активного отдыха предусматривает чередование умственной и физической деятельности, что полностью восстанавливает работоспособность человека.

УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Перечень основной литературы:

1. Болотский, А. В.
    Исследование операций и методы оптимизации Электронный ресурс / Болотский А. В., Кочеткова О. А. - Санкт-Петербург : Лань, 2020. - 116 с. - ISBN 978-5-8114-4568-4

2. Гвоздева, Т. В.
    Проектирование информационных систем: технология автоматизированного проектирования. Лабораторный практикум Электронный ресурс / Гвоздева Т. В., Баллод Б. А. - 2-е изд., стер. - Санкт-Петербург : Лань, 2020. - 156 с. - ISBN 978-5-8114-5147-0

3. Левшина, В. В.
    Применение стандартов ИСО серии 9000 Электронный ресурс / Левшина В. В. : учебное пособие. - Красноярск : СибГУ им. академика М. Ф. Решетнёва, 2020. - 150 с. - Утверждено редакционно-издательским советом университета в качестве учебного пособия для студентов магистратуры по направлению подготовки 27.04.02 «Управление качеством», магистерская программа «Проектирование и внедрение современных систем менеджмента качества (ИСО 9001:2015)», очной, очно-заочной

4. Холопов, В. А.
    Проектирование систем автоматизации и управления: Практикум Электронный ресурс / Холопов В. А. - Москва : РТУ МИРЭА, 2020. - 73 с.

Перечень дополнительной литературы:

1. Белугина, С. В.
    Разработка программных модулей программного обеспечения для компьютерных систем. Прикладное программирование Электронный ресурс / Белугина С. В. - Санкт-Петербург : Лань, 2020. - 312 с. - ISBN 978-5-8114-4496-0

2. Кузенкова, Г. В.
    WEB-технологии. Разработка сайтов Электронный ресурс / Кузенкова Г. В. : практикум. - Нижний Новгород : ННГУ им. Н. И. Лобачевского, 2020. - 50 с. - Рекомендовано методической комиссией Института информационных технологий, математики и механики для студентов, обучающихся по направлению 09.03.03 «Прикладная информатика»

3. Стасышин, В. М.
    Разработка информационных систем и баз данных Электронный ресурс : Учебное пособие для СПО / В. М. Стасышин. - Саратов : Профобразование, 2020. - 100 с. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-5-4488-0527-1

Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине (модулю)

1. Савельев, А. О.
    Проектирование и разработка веб-приложений на основе технологий Microsoft Электронный ресурс / А. О. Савельев, А. А. Алексеев. - Проектирование и разработка веб-приложений на основе технологий Microsoft,2020-03-31. - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. - 419 с. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 2227-8397