МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ
КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им.И.АРАБАЕВА
ОСПО ИНСТИТУТА НОВЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

«УТВЕРЖДАЮ»
Директор ИНИТ
КГУ им.И.Арабаева
к.т.н., и.о.доц. У.Керимов

_____
«____» _____2023г.

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС

по дисциплине Основы алгоритмизации и программирования
для студентов специальности 230109 – «Программное обеспечение вычислительной_ техники и автоматизированных систем»
форма обучения Очное
Курс 1,2 **Семестр** 1,2,3,4
Часов: всего 54 **, лекций** 32 **, практ. зан.** 22
СРС 24

Учебно-методический комплекс разработал(а) преподователи
Тыналиева Чынара Таласбековна, Масянова Айгуль Каныбековна

Рассмотрена и утверждена на заседании ОСПО ИНИТ КГУ им.И. Арабаева
Протокол № ___ от «_____» _____20____г.

Зав. ОСПО ИНИТ: Н.С.Сейтказиева _____
Одобрено учебно-методическим советом ИНИТ КГУ им.И. Арабаева
Протокол № ___ от «_____» _____20____г.

Председатель УМС: _____

Бишкек 2023г.

- Введение
- Цели и задачи дисциплины, ее значение в учебном процессе
- Межпредметные связи. Перечень дисциплин и их разделов, усвоение которых необходимо при изучении данной дисциплины
- Компетенции по Госстандарту
- Структура дисциплины
- Методическая разработка лекций по дисциплине (Краткий курс лекций, презентации)
- Методическая разработка аудиторных форм работы (Краткое содержание практических занятий)
- Учебно-методические материалы
- Критерии баллов — рейтинговой оценки знаний и умений студентов
- Формы текущего и итогового контроля
- Вопросы к модулям
- Темы для самостоятельной работы студентов
- Тестовые задания
- Список литературы и учебно-методическая литература по дисциплине, разработанная преподавателями отделения
- Глоссарий

Введение

Дисциплина “Основы алгоритмизации и программирования” является общепрофессиональной дисциплиной, развивает логическое мышление и формирует базовый уровень знаний.

При изучении дисциплины происходит формирование у студентов знаний об основных принципах алгоритмизации и теории алгоритмов, программе и программировании, а также формирование практических навыков создания прикладных программных продуктов на основе современных технологий программирования с использованием одного из наиболее распространенных алгоритмических языков.

Решение задачи - это преобразование исходной информации в искомый результат. Преобразование состоит из последовательности действий. Компьютер действует по определенным правилам - алгоритму. Алгоритмы в интуитивном смысле не являются математическими объектами. Никакие математические действия к ним неприменимы.

Относящаяся к алгоритму информация и сам алгоритм следует представить на некотором языке. Описание алгоритма зависит от исполнителя - человека или технического средства(компьютер).

Запись алгоритма для человека может иметь обычный текстуальный вид. Компьютер “понимает” только определенные конструкции (команды), алгоритм для компьютера следует представить на языке машины.
Непосредственная запись алгоритма на языке машины нерациональна, а естественный язык не подходит для компьютера. Поэтому появились языки программирования,которые: удовлетворяют требованиям однозначности представления алгоритма; более удобны для человека; непосредственно “не понятны” процессору компьютера.

Таким образом, специалист должен обладать следующими знаниями и навыками:
1) уметь строить алгоритм;
2) знать языки программирования;
3) уметь работать в соответствующей системе программирования.

Мы выбираем Python в качестве первого языка программирования и в качестве инструмента для изучения программирования.
Выбор Python обусловлен такими его преимуществами как ясность кода и быстрота реализации на нем программ.

Цели и задачи дисциплины, ее значение в учебном процессе

Учебная дисциплина направлена на формирование у студентов знаний об основных принципах алгоритмизации и теории алгоритмов, программе и программировании, а также формирование практических навыков создания прикладных программных продуктов на основе современных технологий программирования с использованием одного из наиболее распространенных алгоритмических языков. Расширение знаний учащихся в направлении изучения языков программирования, в частности, языка программирования Python.

Цель изучения дисциплины

Целью дисциплины «Основы алгоритмизации и программирования» является: изучение и освоение базовых понятий и приемов программирования, применяемых на всех основных этапах разработки программ; изучение методов программирования для овладения знаниями в области технологии программирования; подготовка к осознанному использованию как языков программирования, так и методов программирования. Воспитательной целью дисциплины является формирование у студентов научного, творческого подхода к освоению технологий, методов и средств производства программного обеспечения.

Задачи изучения дисциплины

Введение в основные понятия: Знакомство с различными понятиями алгоритмизации и программирования, такими как алгоритмы, структура данных, типы данных, управляющие конструкции и т.д.

Изучение языков программирования: Предоставление студентам базовых знаний о популярных языках программирования и их применение в различных областях.

Разработка алгоритмического мышления: Помощь студентам в освоении навыков абстрактного мышления, способным разработать эффективные алгоритмы для различных задач.

Изучение методов решения проблем: Подготовка студентов к разработке методов и стратегий решения эффективных проблем с использованием вычислительных методов.

Практическое программирование: Проведение практических занятий и лабораторных работ, в ходе которого студенты изучают, как применять изученные концепции на примере создания программного обеспечения.

Обучение основам отладки и тестирования: подготовка студентов к умению выявлять и устранять ошибки в коде, а также тестировать программу для обеспечения их работы.

Развитие навыков работы в сети: Содействие развитию навыков совместной работы над программными проектами, что важно для работы в среде разработчиков.

Эти задачи помогают студентам освоить основы алгоритмизации и программирования, а также подготовить их к более продвинутым темам в области информационных технологий и компьютерных наук.

Межпредметные связи. Перечень дисциплин и их разделов, усвоение которых необходимо при изучении данной дисциплины

Изучение дисциплин «Основы алгоритмизации и программирования» тесно связано с рядом других дисциплин, которые играют главную роль в области информационных технологий и компьютерных наук. Ниже приводится перечень некоторых из таких дисциплин и их разделов, определения которых могут использоваться при изучении данных дисциплин:

Математика:

Дискретная математика: Теория множеств, комбинаторика, теория графов и другие концепции, которые широко используются в алгоритмах и структурах данных.

Теория вычислительных процессов:Автоматы и формальные языки: основы теории автоматов, формальных языков и вычислительной сложности, которые являются ключевыми для понимания работы алгоритмов.

Логика и теория алгоритмов:

Математическая логика: основы символьной логики и теоретические доказательства, которые можно применять для разработки алгоритмов и доказывать их корректность.

Структуры данных и алгоритмы:

Алгоритмы и структуры данных: изучение различных методов сортировки, поиска, обхода графов и других, а также основных структур данных, таких как массивы, упорядоченные, деревья и графы.

Основы информатики:

Архитектура компьютера: Понимание основных направлений работы компьютеров, их структуры и памяти, что помогает понять, как программа работает на уровне аппаратного обеспечения.

Языки программирования: Синтаксис и семантика языков программирования: Основы синтаксиса и семантики популярных языков программирования, таких как C++, Java, Python и других.

Теория вероятностей и статистика: Основы вероятностных моделей: Понимание вероятностных распределений и статистических методов, которые можно использовать при анализе эффективности алгоритмов.

Понимание и изучение этих дисциплин и их разделов помогают студентам лучше овладеть навыками по основам алгоритмизации и программирования, а также применять эти знания в различных областях информационных технологий.

Компетенции по Госстандарту

Выпускник по специальности 230109 – «Программное обеспечение вычислительной техники и автоматизированных систем» в соответствии с целями основной профессиональной образовательной программы и задачами профессиональной деятельности, указанными в пунктах 11 и 15 настоящего Государственного образовательного стандарта, должен обладать следующими компетенциями:

а) общими (ОК):

ОК-1. Уметь организовывать собственную деятельность, выбирать методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.

ОК-2. Решать проблемы, принимать решения в стандартных и нестандартных ситуациях, проявлять инициативу и ответственность.

ОК-3. Осуществлять поиск, интерпретацию и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.

ОК-4. Использовать информационно – коммуникационные технологии в профессиональной деятельности.

ОК-5. Уметь работать в команде, эффективно общаться с коллегами, руководством, клиентами.

ОК-6. Брать ответственность за работу членов команды (подчиненных), за результат выполнения заданий.

ОК-7. Управлять собственным личностным и профессиональным развитием, адаптироваться к изменениям условий труда и технологий в профессиональной деятельности.

ОК-8. Быть готовым к организационно – управленческой работе с малыми коллективами.

ОК-9. Способен приобретать новые знания, с большой степенью самостоятельности, с использованием современных образовательных и информационных технологий.

ОК-10. Способен на научной основе оценить свой труд, оценивать с большой степенью самостоятельности, результаты своей деятельности.

б) профессиональными, соответствующими основным видам профессиональной деятельности (ПК):

<u>производственно-технологическая деятельность:</u>

ПК-1. Владеет знаниями об архитектуре и технических характеристиках персональных компьютеров;

ПК-2. Способен дать характеристику и определить возможности языков, среды программирования;

ПК-4. Владеет технологией проектирования баз данных; организацией структур баз данных;

ПК-5. Владеет знаниями о характеристиках и особенностях эксплуатации локальных вычислительных сетей различных типов;

ПК-6. Способен использовать методы программной защиты информации;

ПК-7. Способен выполнять отладку и тестирование программного продукта;

ПК-8. Способен осуществлять модификацию, адаптацию и настройку программных продуктов;

организационно-управленческая деятельность:

ПК-9. Владеет знаниями об основных положениях действующей нормативной документации;

ПК-10. Владеет основами организации деятельности промышленного предприятия (организации) и управления им;

ПК-11. Владеет знаниями о правилах и нормах охраны труда, техники безопасности, промышленной санитарии и противопожарной защиты.

ПК-12. Способен оценивать экономическую эффективность созданного программного продукта;

ПК-13. Способен реализовывать функции сопровождения программных продуктов;

ПК-14. Способен осуществлять разработку и сопровождение сетевых приложений;

ПК-15. Способен разрабатывать структуру локальной или удаленной базы данных;

ПК-16. Способен обеспечивать эффективное применение пакетов прикладных программ;

Объем дисциплины и виды учебной работы

Виды учебной работы	Всего часов	Семестр

Общая трудоемкость дисциплины	123	51-45
Аудиторные занятия	97	39-60
Лекции	54	22-32
Практические занятия	36	14-22
Прием и контроль	9	3-6
Самостоятельная работа	24	24
Вид итогового контроля (зачет, экзамен)	экзамен	экзамен

Структура дисциплины

План изучения дисциплины (лекции I и II семестр):

№	Темы дисциплины	Кол.часов
	Глава №1.Теоретические основы алгоритмизации и программирования	
1.	Алгоритм.Свойства алгоритма. Формы записи алгоритмов. Назначение функциональных блоков.	**2**
2.	Основные этапы решения задач. Основные структуры алгоритмов. Данные и их типы.	**2**
3.	Языки программирования. Классификация языков программирования. Методы и принципы программирования. Виды программного обеспечения (ПО).	**2**
	Глава №2. Язык программирования PYTHON	

4.	Введение в язык программирования Python. Среда Python. Первый запуск рабочей среды. Основные элементы языка Python.	**2**
5.	Операции, переменные, литералы, типы данные в Python. Ввод и вывод данных в программах на языке Python. Операторы и выражения.	**2**
	Глава № 3. Основные алгоритмические инструкции программирования	
6.	Линейные алгоритмы: операции над целочисленными, вещественными, логическими данными.	**2**
7.	Готовые модули Math, Random в Python.	**2**
8.	Разветвляющийся алгоритм: Простой условный оператор, Сокращенный условный оператор, Составной условный оператор.	**2**
9.	Многозначные ветвления. Алгоритмы поиска максимального и минимального элементов.	**2**
10.	Циклический алгоритм: Оператор цикла While. Инструкции break и continue.	**2**
11.	Оператор цикла For. Функция range().	**2**
12.	Вложенные циклы.	**2**
	Глава № 4. Коллекции данных в Python.	
13.	Работа со строками: основные понятия, операции с ними. Методы строк.Форматирование строк. Базовые алгоритмы обработки строк.	**2**
14.	Работа с множествами в Python.	**2**
15.	Работа со списками в Python.	**2**
16.	Работа с кортежами в Python.	**2**
17.	Работа со словарями в Python.	**2**
18.	Обработка вложенных последовательностей (ВП). Базовые алгоритмы ВП.	**2**

	Глава №5. Работа с функциями. Создание модулей.	
19.	Введение в функции Python.Основные принципы функций в Python.Синтаксис создания функций. Аргументы и возвращаемые значения.	**2**
20.	Область видимости и рекурсия.Область видимости переменных в Python. Локальные и глобальные переменные.Рекурсивные функции и их применение.	**2**
21.	Модули в Python.Роль модулей в организации кода. Создание собственных модулей.Импортирование модулей в Python.	**2**
22.	Продвинутые функциональные возможности.Анонимные функции (лямбда-выражения). Функции высшего порядка.Замыкания.	**2**
	Глава №6. Работа с графикой в Python.	
23.	Рисования с помощью модуля Turtle.	**2**
24.	Использование функций в модуле Turtle.	**2**
	Глава №7. Работа с файлами в Python	
25.	Введение в работу с файлами.Типы файлов (текстовые, бинарные).Основные операции: создание, открытие, чтение, запись и закрытие файлов.	**2**
26.	Открытие и закрытие файлов.Использование функции open для открытия файлов. Чтение и запись текстовых файлов	**2**
27.	Работа с бинарными файлами .Чтение и запись бинарных файлов.Сериализация данных в бинарных файлах. Преимущества и особенности бинарных файлов.	**2**
	Всего:	**54**

План изучения дисциплины(практические):

№	Темы практических работ	Кол.часов
	Практическая работа №1	

1.	Введение в язык программирования Python.Изучить основные типы данных, команды ввода и вывода данных.	**2**
	Практическая работа №2	
2.	Линейный алгоритм: Математические операции в Python.Библиотека (модуль) math	**2**
	Практическая работа №3	
3.	Разветвляющиеся алгоритм: Структура ветвление в Python.	**2**
	Практическая работа №4	
4.	Работа с множественным ветвлением.	**2**
	Практическая работа №5	
5.	Циклический алгоритм: Работа с циклами в Python:Цикл while в Python.	**2**
	Практическая работа №6	
6.	Цикл for в Python.Работа с вложенными циклами.	
	Практическая работа №7	
7.	Работа со строками в Python. Методы работы со строками	**2**
	Практическая работа №8	
8.	Работа с множествами	**2**
	Практическая работа №9	
9.	Работа со списками. Операции над списками в Python	**2**
	Практическая работа №10	
10.	Работа с кортежами. Классические способы обработки кортежей.	**2**

	Практическая работа №11	
11.	Работа со словарями	**2**
	Практическая работа №12	
12.	Работа над формированием вложенных последовательностей	**2**
	Практическая работа №13	
13.	Работа над создание пользовательских функций	**2**
	Практическая работа №14	
14.	Область видимости и рекурсия.Изучение локальных и глобальных переменных.Разработка рекурсивных функций для решения задач.	**2**
	Практическая работа №15	
15.	Работа над созданием модулей,импортирование модуля в основную программу.	**2**
	Практическая работа №16	
16.	Работа с графикой: Рисование с помощью модуля Turtle, применяя функции.	**2**
	Практическая работа №17	
17.	Создание и запись в текстовый файл.Чтение текстового файла и обработка данных	**2**
	Практическая работа №18	
18.	Применение бинарных файлов в реальных приложениях (например, сохранение изображений).	**2**
	Всего:	**36**

Методическая разработка лекций по дисциплине (Краткий курс лекций, презентации)

Глава №5. Работа с функциями. Создание модулей.

Тема №19. Введение в функции Python.Основные принципы функций в Python.Синтаксис создания функций. Аргументы и возвращаемые значения.

Цель лекции: Познакомить студентов с основами функций и модулей в Python, а также научить их создавать собственные функции и использовать стандартные модули.

Что такое функция?

В программировании функция - это набор инструкций, объединенных вместе и предназначенных для выполнения конкретной задачи или выполнения конкретной операции. Функции являются основными строительными блоками программы и служат для организации кода в более читаемую и управляемую структуру.

Основные характеристики функций в программировании:

1)Имя функции: Функция имеет имя, по которому к ней можно обратиться при вызове.

2)Параметры (аргументы): Функция может принимать аргументы или параметры, которые представляют значения, передаваемые в функцию при вызове.

3)Тело функции: Тело функции содержит инструкции, которые выполняются при вызове функции. Это набор команд и выражений, реализующих конкретную логику.

4)Возвращаемое значение: Функция может возвращать результат выполнения в виде значения, и функция может не возвращать ничего.

Пример определения и вызова функции на Python:

```
# Определение функции с именем "приветствие"
def приветствие(имя):
    сообщение = f"Салам, {имя}!"
    return сообщение

# Вызов функции и передача аргумента "Чынара"
результат = приветствие("Чынара")
print(результат)  # Вывод
```

результат:

В этом примере приветствие - это функция, она принимает аргумент имя, создает приветственное сообщение с использованием этого аргумента и возвращает его как результат. При вызове функции с аргументом "Чынара" выводится "Салам, Чынара!".

Зачем используются функции?

Функции используются в программировании по нескольким важным причинам:

1)Структурирование кода: Функции позволяют организовать код программы на более мелкие, легко управляемые блоки. Каждая функция выполняет конкретную задачу или операцию, что делает код более читаемым и понятным.

2)Повторное использование кода: Однажды написанные функции могут быть повторно использованы в различных частях программы или в других программах. Это сокращает дублирование кода и упрощает сопровождение.

3)Абстракция: Функции позволяют абстрагировать (сокращать) сложность. Вместо деталей реализации, пользователь функции может сосредотачиваться на её интерфейсе и том, что функция делает. Это облегчает понимание и использование функций.

4)Упрощение отладки: Когда ошибка возникает в коде, функции помогают ограничить область поиска ошибки. Вы можете тестировать и отлаживать функции независимо друг от друга.

5)Модульность: Функции могут быть организованы в модули, что упрощает структурирование и организацию кода программы.

6)Реализация алгоритмов: Функции позволяют разбить сложные задачи на более мелкие, более управляемые шаги. Это помогает при реализации алгоритмов и логики программы.

7)Улучшение читаемости: Использование функций делает код более читаемым и понятным. Имена функций могут служить документацией для того, что делает код.

8)Поддерживаемость и расширяемость: Путем разбиения программы на функции, вы делаете её более легко поддерживаемой и расширяемой. Новые функции могут быть добавлены без изменения существующего кода.

В общем, функции - это мощный инструмент в программировании, который помогает сделать код более структурированным, эффективным и легко поддерживаемым.

Ключевое слово **def** используется в Python для определения (создания) функций. Синтаксис определения функции выглядит следующим образом:

```
def имя_функции(параметры):
    # Тело функции
    # Выполняемые инструкции
    return результат
```

где:

имя_функции - это имя, которое вы придумываете для функции.

параметры - это список аргументов, которые функция принимает (они могут быть пустыми, если функция не принимает аргументов).

Тело функции - это блок кода, который выполняется при вызове функции. Он должен быть с отступом (обычно 4 пробела или 1 табуляция) от начала строки.

return - это ключевое слово, которое используется для возвращения значения из функции (это опционально).

Передача аргументов в функцию:

Параметры функций - это способ передать данные или аргументы внутрь функции, чтобы она могла их использовать. Аргументы могут быть переданы функции при её вызове.

Пример передачи аргументов в функцию:

```
def приветствие(имя):
    print(f"Привет, {имя}!")

приветствие("Чынара")  # Вызываем функцию с аргументом "Чынара"
```

результат:

```
Привет, Чынара!
```

В этом примере *имя* - это параметр функции *приветствие*, и мы передаем аргумент "Чынара" при вызове функции.

Позиционные и именованные аргументы:

1)Позиционные аргументы: При передаче аргументов в функцию важен порядок. Аргументы, переданные первыми, соответствуют первым параметрам функции, аргументы, переданные вторыми, соответствуют вторым параметрам и так далее. Это называется "позиционными аргументами".

Пример:

```
def сложение(a, b):
    сумма = a + b
    print(сумма)

сложение(3, 5)  # 3 и 5 - позиционные аргументы
```

результат:

2)Именованные аргументы: Вы можете передавать аргументы в функцию с указанием имени параметра, к которому они относятся. Это называется "именованными аргументами". Это позволяет вам передавать аргументы в произвольном порядке, даже если у функции много параметров.

Пример:

```
def приветствие(имя, возраст):
    print(f"Привет, {имя}! Тебе {возраст} лет.")

приветствие(возраст=30, имя="Чынара")  # именованные аргументы
```

результат:

```
Привет, Чынара! Тебе 30 лет.
```

Значения по умолчанию для аргументов:

В Python вы можете установить значения по умолчанию для параметров функции. Если аргумент не передается при вызове функции, используется значение по умолчанию.

Пример:

```
def приветствие(имя, возраст=30):
    print(f"Привет, {имя}! Тебе {возраст} лет.")

приветствие("Чынара")  # Возраст не передан, используется значение по умолчанию 30
приветствие("Тимур", 25)  # Передано значение 25, оно заменяет значение по умолчанию
```

результат:

```
Привет, Чынара! Тебе 30 лет.
Привет, Тимур! Тебе 25 лет.
```

В этом примере возраст имеет значение по умолчанию 30, но при необходимости его можно переопределить, передав явное значение в вызове функции.

Знание о передаче аргументов и значениях по умолчанию позволяет создавать гибкие и мощные функции, которые могут работать с различными данными.

Тест на тему - clck.ru/36Dp46

Тема №20.Область видимости и рекурсия.Область видимости переменных в Python. Локальные и глобальные переменные.Рекурсивные функции и их применение.

Область видимости и переменные - это важные концепции в программировании, которые определяют, где и как можно использовать переменные в коде. Они помогают организовать данные в программе и контролировать доступ к ним.

1)Локальные переменные:

- Локальные переменные объявляются внутри определенной области видимости, обычно внутри функции или блока кода.
- Они видимы только внутри той функции или блока, в котором были объявлены.
- Попытка доступа к локальной переменной за пределами ее области видимости вызовет ошибку.
- Локальные переменные обычно используются для временного хранения данных внутри функции.

Пример на Python:

```
def example_function():
    x = 10  # x - локальная переменная
    print(x)

example_function()  # Вернет 10
print(x)  # Ошибка: x не определена вне функции
```

результат:

```
10
    print(x)  # Ошибка: x не определена вне функции
NameError: name 'x' is not defined
```

2)Глобальные переменные:

- Глобальные переменные объявляются за пределами всех функций и блоков кода.
- Они видимы во всем коде программы после своего объявления.
- Глобальные переменные можно использовать как для чтения, так и для записи в любой части программы.
- Глобальные переменные могут быть удобными, но также могут сделать код менее читаемым и подверженным ошибкам, если их использовать неосторожно.

Пример на Python:

```
global_variable = 20  # global_variable - глобальная переменная

def another_function():
    print(global_variable)  # Можно использовать глобальную переменную

another_function()  # Вернет 20
print(global_variable)  # Вернет 20, так как она доступна глобально
```

результат:

```
20
20
```

3)Область видимости:

- Область видимости определяет, где можно обратиться к переменной и где она видима.
- В языках программирования существуют разные уровни областей видимости, такие как локальная, глобальная и вложенная области видимости.
- Если переменная имеет одно имя в разных областях видимости, то будет использоваться значение из наиболее близкой к месту обращения области видимости.

Пример на Python:

```
x = 10  # Глобальная переменная

def example_function():
    x = 5  # Локальная переменная с тем же именем, затеняющая глобальную
    print(x)  # Выводит 5, так как обращение идет к локальной переменной

example_function()
print(x)  # Выводит 10, так как обращение идет к глобальной переменной
```

результат:

```
5
10
```

Важно понимать, как работают локальные и глобальные переменные, чтобы правильно структурировать код и избежать ошибок в программе.

Вот пример задания, которое демонстрирует использование как локальных, так и глобальных переменных в Python:

Задание: Калькулятор(Способ 1)

Напишите программу для выполнения математических операций. Программа должна иметь следующие функции:

1)**add(a, b):** Функция принимает два аргумента a и b, выполняет сложение и выводит результат на экран.

2)subtract(a, b): Функция принимает два аргумента a и b, выполняет вычитание и выводит результат на экран.

3)multiply(a, b): Функция принимает два аргумента a и b, выполняет умножение и выводит результат на экран.

4)divide(a, b): Функция принимает два аргумента a и b, выполняет деление и выводит результат на экран. Проверьте, что b не равно нулю, чтобы избежать ошибки деления на ноль.

Глобальная переменная operation будет использоваться для хранения текущей выполняемой операции, такой как "сложение", "вычитание", "умножение" или "деление". Эта глобальная переменная будет установлена в соответствии с выбором пользователя, и локальные переменные будут использоваться для хранения аргументов a и b для каждой операции.

```python
# Глобальная переменная для хранения текущей операции
operation = ""

def add(a, b):
    global operation  # Используем глобальную переменную
    operation = "сложение"
    result = a + b
    print(f"Результат {operation}: {result}")

def subtract(a, b):
    global operation
    operation = "вычитание"
    result = a - b
    print(f"Результат {operation}: {result}")

def multiply(a, b):
    global operation
    operation = "умножение"
    result = a * b
    print(f"Результат {operation}: {result}")

def divide(a, b):
    global operation
    operation = "деление"
    if b != 0:
        result = a / b
        print(f"Результат {operation}: {result}")
    else:
        print("Ошибка: Деление на ноль!")

# Пример использования
add(5, 3)
subtract(10, 4)
multiply(7, 2)
divide(8, 0)  # Произойдет деление на ноль
```

результат:

```
Результат сложение: 8
Результат вычитание: 6
Результат умножение: 14
Ошибка: Деление на ноль!
```

В этом примере operation - глобальная переменная, а аргументы a и b - локальные переменные для каждой функции. Программа выполняет математические операции в зависимости от выбора пользователя и выводит результаты на экран.

Задание: Калькулятор(Способ 2)

Для создания простого калькулятора, использующего локальные и глобальные переменные для выполнения математических операций, вы можете написать программу на языке Python. Вот пример такой программы:

```
# Глобальная переменная для хранения результата
результат = 0
def сложить(число):
    global результат
    результат += число
def вычесть(число):
    global результат
    результат -= число

def умножить(число):
    global результат
    результат *= число

def разделить(число):
    global результат
    if число != 0:
        результат /= число
    else:
        print("Ошибка: деление на ноль!")
def главное_меню():
    global результат
    print("Текущий результат:", результат)
    print("1. Сложить")
    print("2. Вычесть")
    print("3. Умножить")
    print("4. Разделить")
    print("5. Выйти")
    выбор = input("Выберите операцию (1/2/3/4/5): ")
    if выбор == '1':
        число = float(input("Введите число для сложения: "))
        сложить(число)
    elif выбор == '2':
        число = float(input("Введите число для вычитания: "))
        вычесть(число)
    elif выбор == '3':
        число = float(input("Введите число для умножения: "))
        умножить(число)
    elif выбор == '4':
        число = float(input("Введите число для деления: "))
        разделить(число)
    elif выбор == '5':
        print("До свидания!")
        exit()
    else:
        print("Некорректный выбор операции.")
while True:
    главное_меню()
```

```
Текущий результат: 0
1. Сложить
2. Вычесть
3. Умножить
4. Разделить
5. Выйти
Выберите операцию (1/2/3/4/5): 1
Введите число для сложения: 56
Текущий результат: 56.0
1. Сложить
2. Вычесть
3. Умножить
4. Разделить
5. Выйти
Выберите операцию (1/2/3/4/5): 2
Введите число для вычитания: 45
Текущий результат: 11.0
1. Сложить
2. Вычесть
3. Умножить
4. Разделить
5. Выйти
Выберите операцию (1/2/3/4/5): 3
Введите число для умножения: 2
Текущий результат: 22.0
1. Сложить
2. Вычесть
3. Умножить
4. Разделить
5. Выйти
Выберите операцию (1/2/3/4/5): 4
Введите число для деления: 2
Текущий результат: 11.0
1. Сложить
2. Вычесть
3. Умножить
4. Разделить
5. Выйти
Выберите операцию (1/2/3/4/5): 5
```

В этой программе:

“результат” является глобальной переменной, в которой хранится текущий результат операций.

Функции сложить, вычесть, умножить и разделить принимают число и выполняют соответствующие математические операции, обновляя глобальную переменную результат.

Функция главное_меню отображает главное меню, предлагая выбрать операцию, и вызывает соответствующую функцию в зависимости от выбора пользователя.

Этот калькулятор позволяет пользователю выполнять операции сложения, вычитания, умножения и деления с текущим результатом.

Цикл **while True**, чтобы программа оставалась активной до тех пор, пока пользователь не решит выйти. Это распространенный подход для создания интерактивных консольных приложений. При желании пользователь может выйти из программы, выбрав соответствующую опцию.

Когда пользователь выбирает опцию "Выйти" в главном меню, программа вызывает exit(), что приводит к завершению программы. Это обычно используется в консольных приложениях для завершения работы программы.

Тест на тему- clck.ru/36DqQf

Тема №21.Модули в Python.Роль модулей в организации кода. Создание собственных модулей.Импортирование модулей в Python.

В программировании, модуль — это файл, который содержит Python-код, включая функции изменения, классы и функции. Модули предназначены для организации кода, чтобы обеспечить его чистоту, читаемость, повторное использование и структурированность.

Создание модулей - это один из фундаментальных принципов модульного программирования, который позволяет разбить большую программу на более мелкие и независимые части (модули). Это упрощает разработку, отладку и поддержку программного обеспечения. В зависимости от языка программирования, который вы используете, процесс создания модулей может немного различаться. Вот общие шаги по созданию модулей:

1)Определение функциональности модуля: Сначала определите, какая функциональность будет реализована в вашем модуле. Модуль должен выполнять конкретную задачу или предоставлять определенный набор функций.

2)Выбор имени модуля: Подумайте над именем для вашего модуля. Хорошие имена должны быть описательными и уникальными, чтобы избежать конфликтов с именами других модулей.

3)Создание файла модуля: В большинстве языков программирования модуль представляется файлом. Создайте файл с именем вашего модуля и добавьте расширение, соответствующее языку программирования (например, .py для Python или .js для JavaScript).

4)Определение интерфейса модуля: Определите интерфейс модуля, то есть список функций, классов или переменных, которые будут доступны извне модуля. Это включает в себя ключевые функции и данные, которые другие части программы смогут использовать.

5)Реализация функциональности: Напишите код, который реализует функциональность вашего модуля. Убедитесь, что код модуля хорошо структурирован и легко читаем.

6)Импортирование модуля: В других частях вашей программы импортируйте созданный модуль, чтобы использовать его функциональность. Способ импорта зависит от языка программирования.

7)Тестирование модуля: Протестируйте ваш модуль, чтобы убедиться, что он работает правильно. Создайте тестовые случаи для проверки разных аспектов функциональности модуля.

8)Документация: Добавьте документацию к вашему модулю. Опишите, как использовать интерфейс модуля, какие параметры принимают функции, и что они возвращают. Хорошая документация помогает другим разработчикам быстро разобраться в вашем модуле.

9)Обработка ошибок и исключений: Предусмотрите обработку ошибок и исключений в вашем модуле, чтобы улучшить его надежность.

10)Распространение модуля: Если ваш модуль предназначен для использования другими разработчиками, убедитесь, что он легко доступен для установки и использования. В случае Python, например, вы можете опубликовать модуль на Python Package Index (PyPI).

11)Обновление и поддержка: Поддерживайте ваш модуль, обновляя его при необходимости, и реагируйте на обратную связь от пользователей.

Это общие шаги, которые следует выполнять при создании модулей в программировании. Конкретные детали могут зависеть от языка программирования, платформы и целей вашего проекта.

Давайте рассмотрим пример создания модуля на Python. Создадим простой модуль, который содержит функцию для вычисления суммы двух чисел. Этот модуль можно использовать в других Python-программах.

1.Создайте файл с именем **my_module.py.** В этом файле мы определим наш модуль.

2.В файле my_module.py добавьте следующий код:

```
# my_module.py

def add_numbers(a, b):
    """Эта функция складывает два числа и возвращает результат."""
    return a + b
```

Этот код определяет модуль my_module с одной функцией add_numbers, которая принимает два аргумента a и b, складывает их и возвращает результат.

1.Теперь мы можем создать другую Python-программу и использовать наш модуль. Для этого создайте новый файл (например, main.py) и добавьте следующий код:

```
# main.py
import my_module

result = my_module.add_numbers(5, 7)
print("Результат сложения:", result)
```

Здесь мы импортируем модуль my_module с помощью оператора import и используем функцию add_numbers из этого модуля для сложения чисел 5 и 7. Результат будет выведен на экран.

Запустите файл main.py. Вы должны увидеть следующий вывод:

```
Результат сложения: 12
```

Библиотеки - это наборы функций, классов и ресурсов, предназначенные для решения определенных задач в программировании.

Они представляют собой уже написанный и протестированный код, который можно использовать в ваших собственных программных проектах для упрощения разработки и расширения функциональности.

Библиотеки могут быть написаны другими разработчиками и распространяться в виде отдельных пакетов или модулей, которые можно импортировать в ваши программы. Вот некоторые примеры библиотек:

1.Библиотеки стандартной библиотеки: Многие языки программирования, такие как Python, содержат стандартную библиотеку, которая включает в себя множество модулей и функций для общих задач, таких как работа с файлами, сетью, математические вычисления и т. д.

2.Библиотеки для работы с графикой: Например, библиотека Pygame для создания компьютерных игр или библиотеки для визуализации данных, такие как Matplotlib для Python.

3.Библиотеки для работы с базами данных: Например, SQLAlchemy для Python, Hibernate для Java.

4.Библиотеки для машинного обучения и искусственного интеллекта: Например, TensorFlow, PyTorch, scikit-learn для Python.

5.Библиотеки для веб-разработки: Например, Express.js для Node.js, Django и Flask для Python.

6.Библиотеки для обработки текста и работы с языками: Например, NLTK для обработки текста на естественных языках, регулярные выражения для обработки строк.

7.Библиотеки для работы с графами и сетями: Например, NetworkX для Python.

8.Библиотеки для разработки пользовательского интерфейса: Например, Qt для C++ или PyQt/PySide для Python.

Чтобы использовать библиотеку в своем проекте, вам обычно нужно выполнить следующие шаги:

Установка библиотеки: Если библиотека не включена в стандартную библиотеку вашего языка программирования, вам нужно будет установить ее с помощью инструмента управления пакетами, такого как “pip” для Python или “npm” для Node.js.

Импорт библиотеки: В вашем коде вы импортируете нужные модули или классы из библиотеки.

Использование функций и классов: Вы используете функции и классы из библиотеки для выполнения необходимых задач в вашем проекте.

Примеры:

```
# Импорт библиотеки
import math

# Использование функции из библиотеки
result = math.sqrt(25)  # Вычисление квадратного корня
print(result)
```

результат:

```
----
5.0
```

Библиотеки значительно упрощают разработку, так как они предоставляют готовый и проверенный код для широкого спектра задач.

Импорт модуля в Python осуществляется с использованием ключевого слова import.

Давайте рассмотрим несколько примеров импорта модулей в Python:

1.Импорт всего модуля:

Предположим, у нас есть файл math_operations.py с таким содержанием:

```
# math_operations.py

def add(a, b):
    return a + b

def subtract(a, b):
    return a - b
```

Теперь мы можем импортировать этот модуль и использовать его функции:

```
import math_operations

result = math_operations.add(5, 3)
print("Сумма:", result)

difference = math_operations.subtract(10, 4)
print("Разность:", difference)
```

Результат:

```
Сумма: 8
Разность: 6
```

2.Импорт конкретных функций из модуля:

Если нам нужны только определенные функции из модуля, мы можем импортировать их напрямую:

```
from math_operations import add, subtract

result = add(7, 2)
print("Сумма:", result)

difference = subtract(15, 6)
print("Разность:", difference)
```

результат:

```
Сумма: 9
Разность: 9
```

3.Импорт модуля с алиасом:

Мы также можем использовать алиас (псевдоним) при импорте:

```
import math_operations as mo
result = mo.add(8, 4)
print("Сумма:", result)
difference = mo.subtract(20, 7)
print("Разность:", difference)
```

результат:

```
Сумма: 12
Разность: 13
```

4.Импорт из стандартной библиотеки:

Python имеет множество встроенных модулей. Например, для работы с датами и временем мы можем импортировать модуль datetime следующим образом:

```
import datetime

current_time = datetime.datetime.now()
print("Текущее время:", current_time)
```

результат:

```
Текущее время: 2023-09-18 10:14:29.957550
```

Стандартные библиотеки в python.

Python имеет обширную стандартную библиотеку, которая включает в себя множество модулей и функций для решения различных задач. Вот некоторые из основных модулей и функций в стандартной библиотеке Python:

1.Модуль math:
math.sqrt(): Квадратный корень.
math.pow(): Возведение в степень.
math.sin(), math.cos(), math.tan(): Тригонометрические функции.
math.pi, math.e: Константы.
2.Модуль datetime:
datetime.datetime(): Работа с датами и временем.
datetime.timedelta(): Разница между двумя датами.
datetime.date(), datetime.time(): Работа с датой и временем отдельно.
3.Модуль os:
os.path: Работа с путями к файлам и директориям.
os.listdir(): Список файлов в директории.
os.mkdir(), os.makedirs(): Создание директорий.
os.remove(), os.unlink(): Удаление файлов.
4.Модуль json:
json.dumps(): Сериализация объекта Python в формат JSON.
json.loads(): Десериализация JSON в объект Python.
5.Модуль urllib:
urllib.request.urlopen(): Выполнение HTTP-запросов.
urllib.parse.urlencode(): Кодирование параметров URL.
6.Модуль collections:
collections.Counter(): Подсчет элементов в последовательности.
collections.defaultdict(): Словарь с значениями по умолчанию.
7.Модуль random:
random.random(): Генерация случайного числа от 0 до 1.
random.randint(): Генерация случайного целого числа в заданном

диапазоне.

8.Модуль csv:

csv.reader(): Чтение данных из CSV-файла.

csv.writer(): Запись данных в CSV-файл.

9.Модуль socket:

socket.socket(): Создание сокета для сетевого взаимодействия.

socket.connect(), socket.bind(), socket.listen(): Основные сетевые операции.

10.Модуль re: Регулярные выражения для поиска и замены текста в строках.

Тема №22. Продвинутые функциональные возможности.Анонимные функции (лямбда-выражения). Функции высшего порядка.Замыкания.

Расширенные возможности управления, такие как **анонимные функции (лямбда-выражения)**, функции высшего порядка и замыкания, являются обязательными инструментами Python для разработки более гибких и мощных программ. Давайте рассмотрим каждый из этих аспектов более подробно:

Анонимные функции (лямбда-выражения):

Анонимные функции, такие как лямбда-функции, представляют собой функцию без имени. Они обычно используются для создания дополнительных функций, которые могут быть переданы в функции высшего порядка. Пример:

```
add = lambda x, y: x + y
result = add(3, 5)
print(result)
# Ответ
8
```

Функции высшего порядка:

Функции высшего порядка — это функции, которые могут принимать другие функции в качестве аргументов или возвращать функции в качестве результата. Они позволяют абстрагировать операции и работать с механизмами более обобщенным образом. Пример:

```
def apply(func, x, y):
    return func(x, y)
def add(x, y):
    return x + y
result = apply(add, 3, 5)  # Результат равен 8
print(result)
# Ответ
8
```

Замыкания:

Замыкание направления, когда функция сохраняет переменную позицию из внешней области, даже после завершения выполнения функции. Они полезны для создания функций, которые могут запоминать состояние. Пример:

```
def make_counter():
    count = 0
    def counter():
        nonlocal count
        count += 1
        return count

    return counter
counter1 = make_counter()
print(counter1())
print(counter1())

counter2 = make_counter()
print(counter2())
#Ответ
1
2
1
```

Эти продвинутые возможности делают Python более мощным и позволяют разрабатывать более гибкие и выразительные программы. Они также полезны при работе с функциональными структурами данных и при использовании функционального программирования на Python.

Глава №6. Работа с графикой в Python.

Тема №23.Рисования с помощью модуля Turtle.

Для простого рисования в среде программирования Python используется модуль Turtle (черепашка), который подключается командой:

```
import turtle
turtle.reset()
```

Запустив программу, вы увидите окно для графики с пером (черепашкой) по центру. Теперь, задавая команды, мы можем строить рисунки на экране.

Программу, использующую модуль turtle надо запускать в графической оболочке. Все программы, работающие с черепашкой, должны начинаться с команд import turtle и turtle.reset(), а заканчиваться

строкой turtle._root.mainloop(). А между ними могут быть следующие команды для черепашки:

Команды перемещения черепашки

forward(n)	Проползти вперед n шагов (пикселей).
backward(n)	Проползти назад n шагов (пикселей).
left(angle)	Повернуться налево на angle градусов.
right(angle)	Повернуться направо на angle градусов.
circle(r)	Нарисовать окружность радиуса |r|, центр которой находится слева от черепашки, если r>0 и справа, если r<0.
circle(r,angle)	Нарисовать дугу радиуса |r| и градусной мерой angle. Дуга рисуется против часовой стрелки, если r>0 и по часовой стрелке, если r<0.
goto(x,y)	Переместить черепашку в точку с координатами (x,y).

Команды рисования

down()	Опустить перо. После этой команды черепашка начнет оставлять след при любом своем передвижении.
up()	Поднять перо.
width(n)	Установить ширину следа черепашки в n пикселей.
color(s)	Установить цвет следа черепашки в s. s должно быть текстовой строкой, заключенной в кавычки, с названием цвета (по-английски), например, "red", "yellow", "green" и т.д.
fill(f)	Используется для рисования закрашенных областей. Начиная рисовать закрашенную область, дайте команду turtle.fill(1), а закончив рисование области — turtle.fill(0).

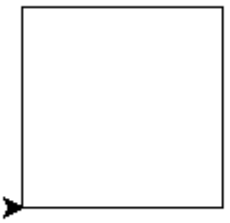
Прочие команды

reset()	Возврат черепашки в исходное состояние: очищается экран, сбрасываются все параметры, черепашка устанавливается в начало координат, глядя вправо.
clear()	Очистка экрана.
write(s)	Вывести текстовую строку s в точке нахождения черепашки.
radians()	Установить меру измерения углов (во всех командах черепашки) в радианы.
degrees()	Установить меру измерения углов (во всех командах черепашки) в градусы. Этот режим включен по умолчанию.
tracer(f)	Включить режим отладки (трассировки) программы черепашки, если значение f равно 1. Если значение f равно 0, отключить режим отладки. В режиме отладки черепашка перемещается медленнее и на экране нарисована сама черепашка. По умолчанию режим отладки включен.

На примере рассмотрим программы:

1. Для рисования квадрата:

```
import turtle
turtle.goto(100,0)
turtle.goto(100,100)
turtle.goto(0,100)
turtle.goto(0,0)
```

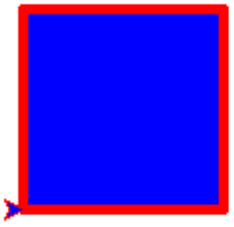


2.Установить ширину следа черепашки в 5 пикселей.

```
import turtle
turtle.width(5)
turtle.goto(100,0)
turtle.goto(100,100)
turtle.goto(0,100)
turtle.goto(0,0)
```

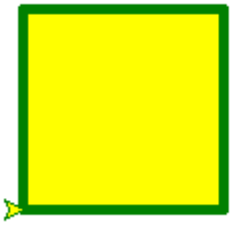
3. Рамку красным цветом и внутри синим заливаем:

```
import turtle
turtle.width(5)
turtle.color('red','blue')
turtle.begin_fill()
turtle.goto(100,0)
turtle.goto(100,100)
turtle.goto(0,100)
turtle.goto(0,0)
turtle.end_fill()
```



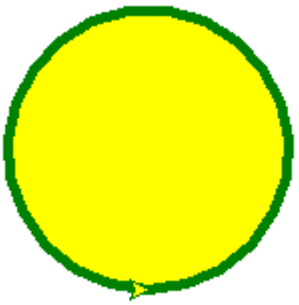
4. Каждый раз не повторять набор команды turtle, в начале задаем from turtle import*, и поменяйте цвет рамки и заливки, на другой какой хотите:

```
from turtle import*
width(5)
color('green','yellow')
begin_fill()
goto(100,0)
goto(100,100)
goto(0,100)
goto(0,0)
end_fill()
```



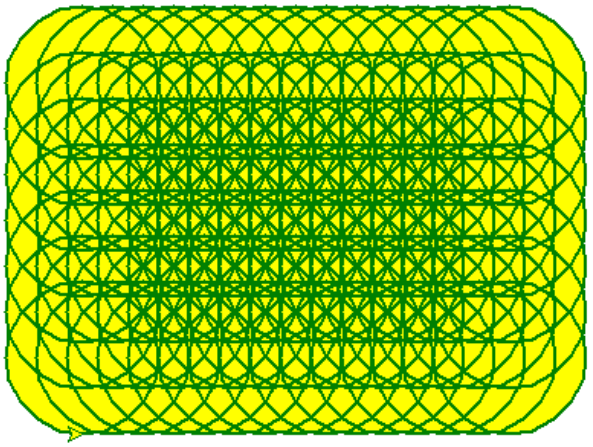
5. Теперь нарисуем круг:

```
from turtle import*
width(5)
color('green','yellow')
begin_fill()
circle(70)
end_fill()
```



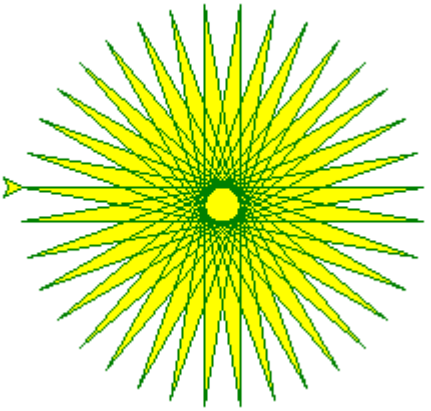
6.Используя цикл **for** нарисуем рисунок “ковер”:

```
from turtle import*
width(2)
color('green','yellow')
begin_fill()
tracer(3)
for x in range (100,-200,-20):
    for y in range(100,-100,-30):
        up()
        goto(x,y)
        down()
        circle(50)
end_fill()
```



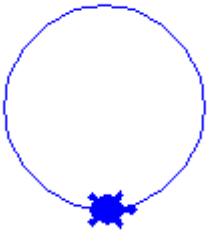
7. Используя цикл **while** нарисуем звезду:

```
from turtle import*
color('green','yellow')
begin_fill()
while True:
    forward(200)
    left(190)
    if abs(pos())<1:
        break
end_fill()
done()
```



8. В место курсора будет черепашка, после следующего кода:

```
from turtle import*
shape('turtle')
color('blue')
circle(50)
```



Тема №24.Использование функций в модуле Turtle.

Модуль Turtle на Python — это отличный инструмент для визуализации графиков. Он позволяет создавать удивительные рисунки, анимацию и игры с помощью простой команды. В этом уроке мы обсудим, как использовать функцию в модуле Черепаха для создания более сложных и красивых рисунков.

Основы модуля Черепаха

Перед тем как перейти к функциям, вспомним основы модуля Turtle:

```
import turtle

t = turtle.Turtle()
t.forward(100) # Переместить черепашку вперед на 100 пикселей
t.left(90)     # Поворот черепашки налево на 90 градусов
t.circle(50)   # Нарисовать окружность радиусом 50 пикселей
turtle.done()  # Завершить работу черепашки
```

Создание функций в модуле Turtle

Преимущества функций

Модульность кода: функции позволяют разбивать код на более мелкие, легко управляемые части.

Уменьшение повторения кода: функции помогают избежать повторения кода, что делает его более читаемым и легким для поддержки.

Улучшенная читаемость: использование функций делает код более понятным, так как их можно так называть, чтобы отразить их функциональность.

Пример создания функций

```
import turtle

def draw_square(t, size):
    for i in range(4):
        t.forward(size)
        t.left(90)

# Создание экземпляра черепашки
t = turtle.Turtle()

# Вызов функции для рисования квадрата
draw_square(t, 100)

# Завершение работы черепашки
turtle.done()
```

Передача параметров в функции

Параметры функции: Параметры могут передаваться в функцию для настройки ее поведения.

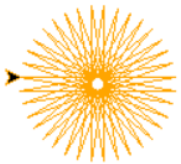
Возвращение результатов: функции также могут возвращать значения, которые могут использоваться в других программах.

Использование функций модуля Turtle помогает создавать более чистый, модульный и понятный код. Они также упрощают процесс создания сложных рисунков и графиков. При создании функций в модуле Turtle важно помнить об организации кода и параметрах включения для создания более гибкого и масштабируемого кода.

Рисование спирали

Попросите студентов создать функцию draw_spiral(t, length, angle)для рисования спирали с заданными параметрами. Затем спросите их использовать эту функцию для создания красочной спирали.

```
import turtle

def draw_spiral(t, length, angle):
    for _ in range(36):
        t.forward(length)
        t.right(angle)

# Создание экземпляра черепашки
t = turtle.Turtle()

# Рисование красочной спирали
colors = ["red", "blue", "green", "yellow", "orange"]
length = 100
angle = 170

for color in colors:
    t.pencolor(color)
    draw_spiral(t, length, angle)

# Завершение работы черепашки
turtle.done()
```



Глава №7. Работа с файлами в Python

Тема № 25. Введение в работу с файлами.Типы файлов (текстовые, бинарные).Основные операции: создание, открытие, чтение, запись и закрытие файлов.

Работа с файлами при программировании является важным фактором, поскольку позволяет сохранять программы и требует данных для дальнейшей обработки. Файлы могут быть разных типов, включая текстовые и бинарные. Различные операции с файлами включают

создание, открытие, чтение, запись и закрытие. Давайте рассмотрим каждую из этих операций подробнее.

Типы файлов:

Текстовые файлы: это файлы, содержащие текст, обычно в формате ASCII или Unicode. Они могут быть прочитаны и интерпретированы как обычный текст.

Бинарные файлы: это файлы, которые содержат данные в бинарном формате, то есть в виде последовательности байтов. Они могут сохранять изображения, видео, аудио или другие данные, которые нельзя интерпретировать как текст.

Основные операции с файлами:

Создание файла: Создание нового файла в файловой системе компьютера.

Открытие файла: Открытие существующего файла для выполнения операции чтения или записи.

Чтение файла: Чтение данных из файла, обычно в виде текста или байтов.

Запись файла: Запись данных в файл в виде текста, так и байтов.

Закрытие файла: Закрытие файла после завершения операций чтения или записи, чтобы открыть ресурс.

В разных языках программирования существуют разные методы работы с файлами, но общие концепции примерно одинаковы. Например, в языке Python можно использовать функции open(), read(), write()и close()для выполнения операций с файлами.

При работе с файлами необходимо помнить о безопасности и освобождении ресурсов после завершения операций. Кроме того, важны возможные ошибки использования, такие как отсутствие файла, проблемы с доступом к файлу и т.д.

На Python работа с файлами осуществляется с использованием встроенных функций и методов. Вот основные операции с файлами на Python:

Создание файла:

Вы можете создать новый файл с помощью функции **open()** с режимом записи **('w'):**

```
file = open('новый_файл.txt', 'w')
```

Это создаст новый файл с именем "новый_файл.txt" в современном рабочем каталоге.

Открытие файла:

Чтобы открыть существующий файл для чтения или записи, воспользуйтесь **open():**

```
file = open('существующий_файл.txt', 'r')  # Чтение файла
file = open('существующий_файл.txt', 'w')  # Запись в файл
```

Чтение файла:

Для чтения данных из файла вы можете использовать метод **read():**

```
data = file.read()
print(data)
```

Запись файла:

Для записи данных в файл воспользуйтесь методом **write():**

```
file.write("Некоторые данные для записи в файл.")
```

Закрытие файла:

Важно закрыть файл после завершения операций с ним, чтобы уменьшить ресурс. Для этого метода воспользуйтесь **close():**

```
file.close()
```

Закрытие файла с помощью конструкций with(контекстный менеджер):

Используйте контекстный менеджер для автоматического закрытия файла после завершения операций:

```
with open('существующий_файл.txt', 'r') as file:
    data = file.read()
    # Операции с данными
# Файл автоматически закроется при выходе из блока `with`
```

Эти операции помогают вам эффективно работать с файлами на Python. Помните также об исключениях, связанных с возможными ошибками при работе с файлами, такими как отсутствие файла или проблемы с доступом.

Тема № 26. Открытие и закрытие файлов.Использование функции open для открытия файлов.Чтение и запись текстовых файлов.

В Python открытие и закрытие файлов осуществляется с помощью функции **open()**, которая возвращает объект файла, с которым можно выполнять различные операции чтения и записи.

В Python для работы с файлами используется функция **open()**, которая открывает файл и возвращает объект файла, через который можно выполнять различные операции. Вот примеры открытия, чтения и записи текстовых файлов:

Открытие файла:

```
# Открытие файла для чтения
file = open('example.txt', 'r')
# Открытие файла для записи
file = open('example.txt', 'w')
# Открытие файла для добавления данных
file = open('example.txt', 'a')
```

Режимы открытия файлов включают:

'r' – чтение (по умолчанию)

'w' – запись, создает файл, если он не существует, усекает файл до нулевой длины, если он существует

'a' – добавление (добавляет данные в конец файла)

'r+' – чтение и запись

'b' – бинарный режим

't' – текстовый режим (по умолчанию)

Чтение текстового файла:

```
with open('example.txt', 'r') as file:
    data = file.read()  # Чтение всего файла
    # Можно также использовать file.readlines() для чтения построчно
    print(data)
```

Запись в текстовый файл:

```
with open('example.txt', 'w') as file:
    file.write('Это строка, которая будет записана в файл.')
```

Добавление в текстовый файл:

```
with open('example.txt', 'a') as file:
    file.write('\nЭто будет добавлено в конец файла.')
```

Полное чтение файла построчно:

```
with open('example.txt', 'r') as file:
    for line in file:
        print(line, end='')
```

Закрытие файла:

```
file.close()
```

Закрытие файла с использованием конструкции with:

```
with open('example.txt', 'r') as file:
    content = file.read()
    print(content)
# Файл автоматически закроется после выхода из блока with
```

При выполнении операций чтения и записи файлов важно учитывать правильное закрытие файлов. Также, использование конструкции with (контекстный менеджер) гарантирует автоматическое закрытие файла после окончания блока кода, даже если произошла ошибка, чтобы избежать утечек ресурсов. Кроме того, обработка исключений поможет учесть возможные ошибки, такие как отсутствие файла или проблемы с доступом.

Полный пример открытия, чтения и записи текстового файла на Python:

```python
# Открываем файл для записи
file_path = 'example.txt'
with open(file_path, 'w') as file:
    data = "Привет, мир! Это пример записи в файл.\nВторая строка."
    file.write(data)

# Открываем файл для чтения
with open(file_path, 'r') as file:
    content = file.read()
    print("Содержимое файла:")
    print(content)

# Работа с данными из файла
lines = content.split('\n')
print("Количество строк в файле:", len(lines))
print("Первая строка:", lines[0])
print("Вторая строка:", lines[1])
```

В этом примере:

1.Мы открываем файл "example.txt" для записи и записываем в него строку данных.

2.Затем мы снова открываем этот файл, но уже для чтения. Мы читаем содержимое файла и выводим его.

3.Мы разбиваем содержимое файла на строки, используя символ новой строки (\n) в качестве разделителя, и демонстрируем, как работать с данными из файла.

Важно отметить, что использование конструкции with автоматически закрывает файл после выполнения операций, что делает код более надежным и избегает забывания закрыть файл вручную.

Тема № 27. Работа с бинарными файлами .Чтение и запись бинарных файлов.Сериализация данных в бинарных файлах.

Преимущества и особенности бинарных файлов.

Бинарные файлы представляют собой файлы, которые хранят данные в бинарном формате, то есть в виде последовательности **нулей и единиц**. Это отличается от текстовых файлов, которые хранят данные в человекочитаемом формате. Работа с бинарными файлами в программировании позволяет сохранять и загружать различные типы данных, включая изображения, аудио, видео, структурированные записи и многое другое. Давайте рассмотрим некоторые аспекты работы с бинарными файлами, включая чтение, запись и сериализацию данных.

Чтение бинарных файлов:

Чтение бинарных файлов в программировании обычно выполняется с использованием байтовых операций. Это позволяет читать данные без изменения их структуры и типа. Для этого часто используется файловый поток, который открывает бинарный файл и позволяет поочередно считывать байты данных.

Пример на Python:

```
with open("binary_file.dat", "rb") as file:
    data = file.read()
    # Обработка данных
```

Запись в бинарные файлы:

Запись в бинарные файлы также производится с использованием байтовых операций. Вы можете записывать данные различных типов, такие как числа, строки, байтовые массивы и другие.

Пример на Python:

```
data = b'\x01\x02\x03\x04\x05'  # Пример байтовых данных
with open("binary_file.dat", "wb") as file:
    file.write(data)
```

Сериализация данных:

Сериализация - это процесс преобразования структурированных данных, таких как объекты, в формат, который можно сохранить в бинарном файле. Это удобно для сохранения и восстановления состояния объектов между запусками программы. В языке Python, модуль pickle предоставляет средства для сериализации и десериализации данных.

Пример на Python:

```
import pickle

data = {"name": "John", "age": 30}
with open("data.pkl", "wb") as file:
    pickle.dump(data, file)

with open("data.pkl", "rb") as file:
    loaded_data = pickle.load(file)
```

Преимущества и особенности бинарных файлов:

Эффективность: Бинарные файлы обычно занимают меньше места, чем текстовые файлы, так как они не содержат символы, которые используются только для человека читаемости.

Скорость: Чтение и запись бинарных файлов может быть быстрее, чем текстовых файлов, так как не требуется преобразование между текстовыми символами и байтами.

Поддержка различных типов данных: Бинарные файлы позволяют хранить практически любые данные, включая изображения, аудио, видео, структурированные записи и многое другое.

Гибкость: Бинарные файлы могут быть использованы для сохранения и восстановления структурированных данных, что делает их полезными для сериализации объектов в программировании.

Однако при работе с бинарными файлами важно учитывать, что они не читаемы для человека, и их формат может быть зависящим от платформы, поэтому необходимо быть осторожным при обмене

бинарными данными между разными системами.

При работе с бинарными файлами необходимо также учитывать вопросы безопасности, так как некорректная обработка бинарных данных может привести к уязвимостям и ошибкам в программе.

Методическая разработка аудиторных форм работы (Краткое содержание практических занятий)

Практическая работа 13,14

Программирование функции в Python

Цель работы: получение практических навыков программирования в создании и использовании функций.

Общие положения

Функция – особым образом сгруппированный набор команд, которые выполняются последовательно, но воспринимаются как единое целое. При этом функция может возвращать (или не возвращать) свой результат.

Функция это блок организованного, многократно используемого кода, который используется для выполнения конкретного задания. Функции обеспечивают лучшую модульность приложения и значительно повышают уровень повторного использования кода.

Функции – это такие участки кода, которые изолированы от остальной программы и выполняются только тогда, когда вызываются. Вы уже встречались с функциями sqrt(), len() и print(). Они все обладают общим свойством: они могут принимать параметры (ноль, один или несколько), и они могут возвращать значение (хотя могут и не возвращать). Например, функция sqrt() принимает один параметр и возвращает значение (корень числа). Функция print() принимает переменное число параметров и ничего не возвращает.

Создание функции

Для того чтобы использовать какую-нибудь собственную функцию, вначале необходимо ее объявить (создать).

Блок функции начинается с ключевого слова def, после которого следуют название функции и круглые скобки ( ). Любые аргументы, которые принимает функция, должны находиться внутри этих скобок. После скобок идет двоеточие и с новой строки с отступом начинается тело функции.

def <имя функции>(аргументы):

<тело функции>

Пример функции в Python:

```
def my_function(argument):
print (argument)
```

После функции до кода, который находится вне функции, необходимо делать отступ в две пустые строки для повышения читаемости кода. Если у вас есть несколько функций в одном файле, между кодом одной и сигнатурой другой функции тоже надо оставлять две пустые строки.

Вызов функции

После создания функции, ее можно исполнять, вызывая из другой функции или напрямую из оболочки Python. Для вызова функции следует ввести ее имя и добавить в скобках аргументы.

<имя функции>(аргументы)

Обращение к ранее объявленной функции с целью выполнения ее команд называется вызовом.

Имена функций должны состоять из маленьких букв, а слова разделяться символами подчеркивания. Аргументы (параметры) могут изменять поведение функции.

Аргументы функции в Python

Вызывая функцию, мы можем передавать ей следующие типы аргументов:

1. Обязательные аргументы (Required arguments)
2. Аргументы-ключевые слова (Keyword argument)
3. Аргументы по умолчанию (Default argument)
4. Аргументы произвольной длины (Variable-length argumens)

Обязательные аргументы функции. Если при создании функции мы указали количество передаваемых ей аргументов и их порядок, то и вызывать ее мы должны с тем же количеством аргументов, заданных в нужном порядке.

Например:

```
def bigger(a,b):#В описании функции указано, что она принимает 2 аргумен
    if a>b:
        print(a)
    else:
        print(b)


bigger(5,6)#Корректное использование функции
```

Аргументы - ключевые слова используются при вызове функции. Благодаря ключевым аргументам, вы можете задавать произвольный (то есть не такой, каким он описан при создании функции) порядок аргументов.

Например:

```
def person(name,age):
    print(name,"is",age,"years old")


person(name="Chika",age=23)
```

Аргументы, заданные по умолчанию. Аргумент по умолчанию, это аргумент, значение для которого задано изначально, при создании функции. Например:

```
def space(planet_name,center='Star'):
        print(planet_name,"is orbiting a",center)


space("Mars")#В результате получим: Mars is orbiting a Star
```

```
space("Mars","Black Hole")#В результате получим: Mars is orbiting a Black Hole
```

Аргументы произвольной длины. Иногда возникает ситуация, когда вы заранее не знаете, какое количество аргументов будет необходимо принять функции. В этом случае следует использовать аргументы произвольной длины. Они задаются произвольным именем переменной, перед которой ставится звездочка (*). Например:

```
def unknown(*args):
    for argument in args:
        print(argument)


unknown("hello","word")     # напечатает оба слова, каждое с новой строки
unknown(1,2,3,4,5)          # напечатает все числа, каждое с новой строки
unknown()                   # ничего не выведет
```

Ключевое слово return

Выражение **return** прекращает выполнение функции и возвращает указанное после выражения значение. Выражение **return** без аргументов это то же самое, что и выражение return None. Соответственно, теперь становится возможным, например, присваивать результат выполнения функции какой-либо переменной. Например:

```
def bigger(a,b):
    if a>b:
        return a    # Если а больше чем b, то возвращаем а и прекращаем выполнение функции
    return b   # Незачем использовать else. Если мы дошли до этой строки, то b, точно не меньше чем а


num = bigger(23,24) # присваиваем результат функции bigger переменной num
```

Пустая функция.

Чтобы создать пустую функцию, нужно в её теле использовать оператор заглушки pass. Тогда она будет существовать и не выполнять никаких действий. Такие функции могут использоваться для различных специфических задач, например, при работе с классами, асинхронной отправкой форм.

def example():

pass

Область видимости переменных

В Python две базовых области видимости переменных:

- локальные переменные - переменные, создаваемые внутри функций, недоступны извне и существуют только внутри функции.
- глобальные переменные – переменные, создаваемые вне функции, могут быть доступны из функций.

Это означает, что доступ к локальным переменным имеют только те функции, в которых они были объявлены, в то время как доступ к глобальным переменным можно получить по всей программе в любой функции.

Например:

```python
age = 44  #глобальная переменная age

def info():
    print(age)  # печатаем глобальную переменную age

def local_info():
    age = 22    #создаем локальную переменную age
    print(age)

info()          #напечатает 44
local_info()    #напечатает 22
```

Важно помнить, что для того чтобы получить доступ к глобальной переменной, достаточно лишь указать ее имя. Однако, если перед нами стоит задача изменить глобальную переменную внутри функции - необходимо использовать ключевое слово **global.**

Например:

```python
age = 13  #глобальная переменная age

def get_older():
    global age  # функция изменяющая глобальную переменную
    age+=1

print(age)  #напечатает 13
get_older() #увеличиваем age на 1
print(age)  #напечатает 14
```

Рекурсией называется процесс, когда функция вызывает саму себя, а сама функция называется рекурсивной.

Классическим примером рекурсии может послужить функция вычисления факториала числа.

Напомним, что факториалом числа, например, 5 является произведение всех натуральных (целых) чисел от 1 до 5.

То есть, 1 * 2 * 3 *4 * 5

```python
def f(num):
    if num==0:
        return 1    #факториал нуля равен единице
    return f(num-1)*num


print(f(5))         # выведет число 120
```

Рекурсию рекомендуется использовать только там, где это действительно необходимо. Интерпретатор Python автоматически выделяет память для выполняющейся функции, если вызовов самой себя будет слишком много, это приведёт к переполнению стека и аварийному завершению программы.

Функции высшего порядка

Функции, которые принимают или возвращают другие функции, называются **функциями высшего порядка.**

Функция filter

Функция **filter** принимает критерий отбора элементов, а затем сам список элементов. Возвращает она список из элементов, удовлетворяющих критерию.

Чтобы этой функцией воспользоваться, нужно сообщить функции **filter** критерий, который говорит, брать элемент в результирующий список или нет. Давайте напишем простую функцию, которая проверяет, что слово длиннее шести букв, и затем отберем с ее помощью длинные слова.

```
def is_word_long(word):
    return len(word)>6
words = ['В','новом','списке','останутся','только','длинные','слова']
for word in filter(is_word_long,words):
    print(word)
```

останутся
длинные

С методом filter вам не нужно вручную создавать и заполнять список, достаточно указать условие отбора.

Лямбда-функции

Часто в качестве аргумента для функций высшего порядка мы хотим использовать совсем простую функцию. Причем нередко такая функция нужна в программе только в одном месте, поэтому ей необязательно даже иметь имя.

Такие короткие безымянные (анонимные) функции можно создавать инструкцией

lambda <аргументы>: <выражение>

Такая инструкция создаст функцию, принимающую указанный список аргументов и возвращающую результат вычисления выражения.

В языке Python тело лямбда-функции имеет ровно одно выражение. Скобки вокруг аргументов не пишутся, аргументы от выражения отделяет двоеточие.

Теперь мы можем записать функцию, проверяющую длину слова, следующим образом:

```
lambda word: len(word)>6
words = ['В','новом','списке','останутся','только','длинные','слова']
long_words = list(filter(lambda word: len(word)>6,words))
print(long_words)
```

Лямбда-функция – полноценная функция. Ее можно использовать в составе любых конструкций. Например, созданную лямбда-функцию, можно присвоить какой-либо переменной:

```
add = lambda x, y:
    x + y
add(3, 5) # 8
add(1, add(2, 3)) # 6
```

Порядок выполнения работы

Задание 1. Напишите программу, которая вычислит значение выражения. Значение выражения вида $\sqrt{a^2 + b^2 + cos^3(ab)}$ вычислять с использованием функции.

$$L=\frac{\sqrt{x^2+z^2+cos^3(xz)}+\sqrt{y^2+x^2+cos^3(yx)}}{\sqrt{z^2+y^2+cos^3(zy)}}.$$

Задание 2. Напишите программу, которая вычислит значение выражения. Значение факториала вычислять с использованием функции (Факториалом числа является произведение всех натуральных (целых) чисел).

$$C = \frac{n!}{m!(n-m)!}$$

Задание 3. Составить программу, согласно полученному варианту задания, используя функции высшего порядка. Список чисел вводится пользователем.

Вариант	Задание
1	Напишите программу, которая подсчитает и выведет сумму кубов всех трехзначных чисел, делящихся на 8.
2	Напишите программу, которая подсчитывает и выводит сумму кубов отрицательных чисел списка
3	Напишите программу для печати четных чисел из заданного списка.
4	Напишите программу, которая извлекает из списка числа, делимые на 17
5	Напишите программу, которая извлекает из списка трехзначные четные числа, делимые на 6.
6	Напишите программу, которая извлекает из списка нечетные числа, делимые на 11.
7	Напишите программу, которая подсчитает и выведет произведение квадратов всех однозначных чисел, делящихся на 2.
8	Напишите программу для печати нечетных чисел из заданного списка
9	Напишите программу, которая подсчитает и выведет сумму квадратов всех двузначных чисел, делящихся на 7.
10	Напишите программу, которая подсчитает и выведет сумму квадратов положительных чисел списка.

Практическая работа №15

Создание модуля

Цель работы: получение практических навыков программирования в создании модуля.

Оборудование: ПЭВМ.

Контрольные вопросы:

1.Где находятся все модули Python?

2.Какие виды модулей есть в Python?

3.В чем суть модульного программирования?

4.Чем модуль отличается от подпрограмм?

5.Как использовать модули в Python?

Модули в Python - это просто файлы Python с расширением .py.

Модуль оформляется в виде отдельного файла с исходным кодом.

Имя модуля будет именем файла. Модуль Python может иметь набор функций, классов или переменных, определенных и реализованных.

Создание модуля

Чтобы создать свой модуль в Python достаточно сохранить код в файл с расширением.py Теперь он доступен в любом другом файле.

Например, создадим файл mymodule.py, в которой определим какие-нибудь функции:

```
def hello():
    print('Hello, world!')

def fib(n):
    a = b = 1
    for i in range(n - 2):
        a, b = b, a + b
    return b
```

Теперь в этой же папке создадим другой файл, например, main.py:

```
import mymodle
mymodle.hello()
print(mymodle.fib(10))
```

Выведет:

```
Hello, world!
55
```

Модуль нельзя именовать также, как и ключевое, также имена модулей нельзя начинать с цифры и не стоит называть модуль также, как какую-либо из встроенных функций.

Порядок выполнения работы

Задание 1. Создайте свой модуль, в котором определите какие-нибудь функции, одна из которых должна быть описанием вашего модуля. Затем создайте программу, которая будет использовать ваш модуль.
Задание 2. Составить программу, согласно полученному варианту задания.

Вариант	Задание
1	Список A содержит N чисел. Перепишите из списка A в список B только те элементы, значения которых не равны заданному значению K. Назначение функции: переписывание из одного списка в другой только тех элементов, которые не совпадают с заданным значением. Оформите созданную функцию в виде программного модуля и подключите его к основной программе.
2	Список A содержит N упорядоченных по возрастанию чисел. Вставьте в список некоторое значение K так, чтобы упорядоченность списка не нарушилась. Назначение функции: вставка заданного значения в упорядоченный список. Оформите созданную функцию в виде программного модуля и подключите его к основной программе.
3	Список A содержит N элементов, значения которых не определены. Организуйте запрос: «Сколько элементов списка следует заполнить?» Заполните список заданным числовым значением K. Назначение функции: заполнение заданного числа элементов заданным значением. Оформите созданную функцию в виде программного модуля и подключите его к основной программе.
4	Список A содержит N чисел. Найдите количество элементов списка, значения которых превышают заданное значение K. Назначение функции: подсчет количества элементов, превышающих заданное значение. Оформите созданную функцию в виде программного модуля и подключите его к основной программе.
5	Список A содержит N чисел. Найдите сумму значений элементов списка, меньших заданного значения K. Назначение функции: суммирование элементов, значения которых меньше заданного. Оформите созданную функцию в виде программного модуля и подключите его к основной программе.

6	Список А содержит N чисел. Найдите произведение значений элементов списка. Назначение функции: вычисление произведения элементов списка. Оформите созданную функцию в виде программного модуля и подключите его к основной программе.
7	Список А содержит N чисел. Удалите из списка значение с порядковым номером К. Назначение функции: удаление из списка значения с заданным порядковым номером. Оформите созданную функцию в виде программного модуля и подключите его к основной программе.
8	Список А содержит N чисел. Значения всех элементов списка, отличающихся от заданного значения К, замените другим заданным значением М. Назначение функции: замена значений элементов списка. Оформите созданную функцию в виде программного модуля и подключите его к основной программе.
9	Список А содержит N чисел. Найдите в нем элемент с наибольшим значением. Назначение функции: поиск в списке элемента с наибольшим значением. Оформите созданную функцию в виде программного модуля и подключите его к основной программе.
10	Список А содержит N чисел. Разработайте программу, с помощью которой можно определить количество наибольших элементов в нем. Назначение 1 функции: нахождение максимального элемента. Назначение 2 функции: подсчет количества максимальных элементов. Оформите созданные функции в виде программного модуля и подключите его к основной программе.

Практическая работа №16
Рисование с помощью модуля Turtle

Цель работы: Освоить основы рисования с использованием модуля Turtle в Python, понять принципы управления черепашкой на экране и создать простые графические изображения.

Вопросы контрольные:

1.Что такое модуль Turtle в Python и для чего он используется?

2.Как создать экземпляр черепашки для рисования?

3.Как передвигать черепашку на экране?
4.Как изменять цвет линии и её толщину?
5.Как создать простые графические изображения с использованием модуля Turtle?

Теоретические сведения:

Модуль Turtle в Python предоставляет простой способ рисования на графическом экране с помощью черепашки. Черепашка может перемещаться по экрану, рисуя след за собой. Основные команды для работы с черепашкой включают forward(), backward(), left(), right(), penup(), pendown(), color(), и другие.

Задание 1 (выполнение):

1.Импортируйте модуль Turtle:

```
import turtle
```

2.Создайте экземпляр черепашки:

```
t = turtle.Turtle()
```

3.Нарисуйте простую фигуру, например, квадрат:

```
for _ in range(4):
    t.forward(100)
    t.left(90)
```

4.Завершите рисование и отобразите результат:

```
turtle.done()
```

Задании для самостоятельной работы:

№	Варианты заданий:
1	Нарисуйте пятиугольник с разноцветными сторонами.
2	Нарисуйте множество квадратов разного размера и цвета, начиная с самого большого и уменьшая размер на каждом шаге.
3	Создайте анимацию, перемещая черепашку по экрану, рисуя путь звезды или другой геометрической фигуры.
4	Используйте циклы и условия, чтобы нарисовать сложную фигуру, например, звезду Давида или сердце.
5	Нарисуйте спираль с использованием черепашки. Сделайте

	спираль, которая уменьшается или увеличивается на каждом шаге.
6	Создайте графический паттерн, состоящий из повторяющихся геометрических фигур, таких как квадраты, круги или треугольники.
7	Напишите программу, которая рисует ваше имя на экране в графическом стиле. Каждая буква должна быть оформлена в виде графической фигуры.
8	Используйте модуль Turtle для создания анимации, представляющей простую сцену, например, движение мяча или машинки.
9	Нарисуйте свой собственный рисунок, представляющий что-то, что вас вдохновляет, например, природу, животных или абстрактное искусство.
10	Создайте интерактивное приложение с использованием черепашки. Например, программа может реагировать на перемещение мыши и выполнять действия, когда черепашка встречает определенные объекты на экране.

Практическая работа №17
Создание и запись в текстовый файл.Чтение текстового файла и обработка данных

Целью работы: является владение навыками создания, записи и чтения текстовых файлов на языке программирования, а также обработка данных, хранящихся в таких файлах.

Контрольные вопросы:

1.Как создать текстовый файл на Python и записать в него текст?

2.Как открыть существующий текстовый файл для чтения в Python?

3.Как считать критерием текстовый файл в переменную в Python?

4.Как записать данные в конец существующего текстового файла без удаления существующего изменения?

5.Как обработать каждый символ в текстовом файле и настроить режим работы?

Теоретические сведения:

Текстовый файл - это файл, который содержит текстовую информацию, представленную в виде символов последовательности.

Операции с текстовыми файлами включают создание файлов, запись данных в них, чтение данных из них и обработку информации, хранящейся в файлах.

Для работы с текстовыми файлами на языке программирования обычно используются стандартные библиотеки или модули, обеспечивающие соответствующие функции.

Открытие файлов: Для открытия файлов в Python используется функция **open().** Она принимает два основных аргумента: имя файла и режим доступа (например, «r» для чтения, «w» для записи, «a» для добавления данных в конец файла и другие). Пример: file = open("example.txt", "r").

Закрытие файлов: После завершения работы с файлом важно закрыть его с помощью метода close(), чтобы уменьшить ресурс и убедиться, что изменения сохранены. Пример: file.close().

Запись данных: Для записи данных в файл используется метод write(). Он позволяет записывать текстовую информацию в файл. Пример: file.write("Привет, мир!").

Чтение данных : Для чтения данных из файла можно использовать методы read(), readline(), или readlines(). read()считывает весь файл, readline()читает один текст, а readlines()читает все строки и возвращает их в виде списка.

Исключения : при работе с файлами важны возможные исключения, такие как FileNotFoundError(если файл не существует), PermissionError(если нет правого доступа) и другие. Рекомендуется использовать блоки try...excepтдля исключений.

Контекстные менеджеры : Для гарантированного закрытия файла после использования рекомендуется использовать контекстные менеджеры с оператором with.

Варианты заданий:

№	Задание
1	Создание и запись в файл: 1.Создайте текстовый файл с именем "example.txt". 2.Запишите в этот файл следующий текст: "Привет, мир!" 3.файл.
2	Чтение файла и вывод данных: 1.Откройте файл «example.txt» для чтения. 2.Создайте важный файл и выведите его на экран.

	3.Закрыть файл.
3	Обработка данных из файла: 1.Откройте файл «data.txt» для чтения. Файл содержит числа, разделенные пробелами. 2.Прочитайте числа из файла и вычислите их сумму. 3.Вы создаете структуру на экране. 4.Закрыть файл.
4	Запись данных в файл: 1.Создайте текстовый файл "names.txt". 2.Запросите у пользователя ввод нескольких имен (каждое имя в новой строке) и сохраните их в файле. 3.Закрыть файл.
5	Поиск слов в файле: 1.Откройте файл «text.txt» для чтения. Файл содержит текст. 2.Запросите у пользователя слово для поиска. 3.Проверьте, есть ли это слово в тексте файла, и вы введете сообщение.
6	Копирование файла статистики: 1.Откройте файл «source.txt» для чтения. 2.Откройте файл «destination.txt» для записей. 3.Скопируйте требования "source.txt" в "destination.txt". Закройте оба файла.
7	Подсчет слов в тексте: 1.Откройте файл «text.txt» для чтения. 2.Подпишите количество слов в тексте (слова разделяются пробелами или другими разделителями). 3.Вы вводите количество слов на экране. Закрыть файл.
8	Фильтрация данных в файл: 1.Откройте файл «data.txt» для чтения. Файл содержит числа, разделенные пробелами. 2.Прочтите числа из файла и создайте новый файл "filtered_data.txt", в котором будут только четные числа. 3.Закройте оба файла.
9	Замена текста в файле: 1.Откройте файл «text.txt» для чтения. 2.Замените все вхождения слова «старое» на «новое» и сохраните изменения в файле.

	3.Закрыть файл.
10	Объединение файлов изменений: 1.Откройте файлы «file1.txt» и «file2.txt» для чтения. 2.Считайте важные оба файла и внесите их в новый файл "merged.txt". 3.Закройте все файлы.

Практическая работа №18
Применение бинарных файлов в реальных приложениях (например, сохранение изображений).

Цель работы: состоит в создании Python-приложений, которые позволяют пользователям загружать, сохранять, обрабатывать и отображать изображения, используя двоичные файлы для хранения данных.

Контрольные вопросы:

1.Что такое бинарный файл и чем он отличается от текстового файла?
2.Как открыть и закрыть бинарный файл в Python?
3.Как записать данные в бинарный файл с помощью Python?
4.Как получить данные из бинарного файла с использованием Python?
5.Какие библиотеки Python часто используются для обработки изображений?

Теоретические сведения:

Бинарные файлы в Python: В Python для работы с бинарными файлами используются функции open(), read(), write(), и close(). Отличие бинарных файлов от текстовых заключается в том, что они обрабатывают данные как последовательность байтов, а не как символы.

Модуль PIL(Библиотека изображений Python): Модуль PIL, ныне известный как Pillow, позволяет работать с изображениями в форматах JPEG, PNG, BMP и многих других. Он предоставляет возможности для открытия, обработки и сохранения изображений.

Манипуляции с изображениями: Обработка изображений может включать в себя изменение размеров, фильтры, редактирование и другие операции.

Варианты заданий:

№	Задании
1	Задание на создание простого текстового редактора: Разработайте текстовый редактор на Python, который позволяет

	пользователю вводить текст, сохранять его в бинарном файле и загружать текст обратно из файла для редактирования.
2	Задание на обработку данных: создайте программу, которая принимает данные от пользователя, сохраняет их в бинарном файле, а затем извлекает и обрабатывает данные (например, вычисление некоторого количества чисел) при их включении.
3	Задание работы с изображениями: Разработайте простое приложение, которое позволяет пользователю загружать изображения, сохранять их в двоичных файлах и выводить их на экран. Это могут быть галереи изображений.
4	Задание создания простой системы хранения паролей: Разработайте программу для хранения паролей и логинов в бинарном файле. Пользователь должен иметь возможность добавлять, просматривать и удалять записи с помощью записей.
5	Задание на сохранение игры: создайте простую текстовую игру, которая позволит сохранить текущее состояние игры в бинарном файле и позже загрузить его для продолжения игры.
6	Задание для создания простого ежедневника: Разработайте приложение для управления ежедневником, позвольте пользователю вводить записи, сохранять их в бинарном файле и загружать записи из файла для просмотра.
7	Для задания управления списком задач: создайте программу для управления списком задач, разрешите пользователю удалить и решить задачу, сохраняя данные в бинарном файле.
8	Задание архивирования файлов: Разработайте программу для архивирования и извлечения файлов в бинарном формате. Пользователь должен иметь возможность создавать архивы из файлов и распаковывать их.

Учебно-методические материалы

Критерии баллов — рейтинговой оценки знаний и умений студентов

Деятельность студентов в течение семестра оценивается следующим образом: работа на семинарах (50%), самостоятельные работы и реферат (20%), активность (25%), посещение занятий (5%).

Работа на семинарах (50%)

Чтение текстов и участие в дискуссиях являются важными составляющими работы на семинарах. Приветствуются вопросы по структуре и содержанию текста, комментарии, помогающие уяснить значение основных категорий и т.п.

Пропущенные семинары необходимо отработать письменно. «Отработка» должна содержать основные моменты пропущенной темы занятия. Оценка за «отработки» не выставляется. Последний срок сдачи «отработок» - заключительное занятие по курсу (тем, кто не сможет присутствовать на заключительном занятии «отработку» необходимо принести заранее).

Неотработанные семинары являются основанием незачета по данному курсу.

Критерии оценки: регулярное присутствие и активное участие, уместность и глубина вопросов и комментариев, способность задавать живой импульс дискуссии и вовлекать других студентов в дебаты.

Оценки за активность на семинарах выставляются по 10-ти балльной шкале.

Критерии оценки работы студентов на семинарах следующие:

10 баллов – индивидуальный ответ, изложенный по существу структурно, логично, своими словами.

8-9 баллов – индивидуальный ответ, изложенный своими словами. Возможны мелкие проблемы с логикой изложения.

5-7 баллов – индивидуальный ответ, изложенный частично своими словами. Возможны мелкие проблемы с логикой изложения.

1-4 балла – индивидуальный ответ – уточнение (дополнение) по рассматриваемым вопросам семинарского занятия, задаваемые вопросы.

Самостоятельные работы и реферат (20%)

Самостоятельные работы выполняются на отдельном листочке письменно от руки. Указывается имя, фамилия, группа и дата сдачи работы.

Все письменные работы НЕ принимаются позже установленных сроков сдачи, за исключением документально подтвержденных случаев отсутствия вследствие болезни или форс-мажорных обстоятельств.

Критерии оценки письменных работ следующие:

10 – выдающаяся работа на высоком уровне, присутствует логика и оригинальность изложения, выдвинут и доказан тезис, видно уверенное владение освоенным материалом.

8-9 – очень хорошая работа, продемонстрированы не только усвоенные знания по курсу, но навыки анализа материала и самостоятельного

мышления. Возможны мелкие проблемы с логикой изложения.
6-7 – хорошая работа, продемонстрированы не только усвоение фактических знаний по курсу и основные навыки аргументации, но изложение не вполне закончено с точки зрения обоснования тезиса и раскрытия вопроса.
4-5 – средняя работа, неполное усвоение фактических знаний по курсу, слабая логика изложения и обоснования.
2-3 – плохая работа, отрывочные знания по курсу, слабая логика изложения и обоснования.
1 – отсутствие каких-либо знаний.
0 – доказанный случай плагиата.
Темы рефератов студенты выбирают согласно нумерации по учебному журналу.
Реферативная работа оформляется письменно от руки. Допускается печатное исполнение титульного листа, списка литературы, графических и табличных приложений.
Студенты, вовремя не сдавшие реферат, защищают свою работу на консультации или в дополнительно отведенное время.
Своевременное выполнение работ является предпосылкой к обоснованию возможности допуска студента к зачету (экзамену).
Проверка уровня усвоения лекционных занятий, включая теоретических СРС и СРСП, проводится тестированием по рейтинго-модульной системе. Каждый тест включает 15 вопросов, где правильный ответ на 1 вопрос оценивается на 1 балл.
Результаты практических работ, включая, практических СРС и СРСП принимаются в виде графических и контрольных работ, рефератов и собеседования.

Формы текущего и итогового контроля

№	Этапы проверки	Вид средства проверки	Баллы	Сроки
1	1 модуль	Тестирование	35	Согласно графику учебного процесса
2	2 модуль	Проверка заданий	35	Согласно графику учебного процесса
3	Практические СРС	Контрольные и графические работы, рефераты,	10	В течение семестра, до итогового контроля

		собеседование		
4	Поощрительные баллы за активность		7	В конце семестра, до итогового контроля
5	Посещение занятий		3	В течение семестра
6	Итоговый контроль	Письменный или устный	10	Согласно графику учебного процесса
	Итого:		**100**	

Вопросы к модулям

1.Что такое анонимные функции в Python и как они объявляются с использованием лямбда-выражений?
2.Что представляет собой функция высшего порядка, и какие примеры функций высшего порядка на Python вы можете привести?
3.Каковы основные различия между лямбда-выражениями и схожими функциями в Python?
4.Что такое заключение, и как они освоили Python? Какие преимущества они предоставляют?
5.Какие типы задач или сценариев можно эффективно решать с помощью функций анонимного использования, таких функций, как порядок и заключение на Python?
6.Как подключить модуль Turtle в свой Python-скрипт?
7.Как используется команда перемещения черепашек вперед на несколько шагов?
8.Какое использование команды для рисования закрашенных регионов?
9.Какое использование команды для установки цвета после черепашки?
10.Что используется для рисования круга?
11.Какие основные операции можно выполнять с файлами в Python?
12.Какие режимы открытия файлов доступны в функции open()? Как они отличаются друг от друга?
13.Почему важно закрывать файл после выполнения операций чтения или записи?
14.Какие преимущества предоставляет конструкция with при работе с файлами?
15.Как можно обработать возможные ошибки при работе с файлами, такие как отсутствие файла или проблемы с доступом?
16.Какие преимущества предоставляют бинарные файлы в сравнении с текстовыми файлами при хранении данных?
17.Какой метод используется для чтения данных из бинарного файла в языке программирования Python?
18.В чем состоит процесс сериализации данных, и зачем он полезен в программировании?

19.Как можно сериализовать данные с помощью модуля pickle в Python, и как их десериализовать?
20.Какой файловый режим следует использовать при открытии бинарного файла для записи в Python, и какие данные можно записывать в бинарный файл?

Темы для самостоятельной работы студентов

1.Разработка функций в Python: Основы и применение.
2.Область видимости переменных в функциях Python.
3.Рекурсия и ее роль в функциональном программировании.
4.Анонимные функции и лямбда-выражения в Python.
5.Функции высшего порядка и их использование в Python.
6.Замыкания и их использование в практике программирования.
7.Модули и пакеты в Python: Основы и структура.
8.Создание собственных модулей в Python и их применение.
9.Искусство и программирование: Исследование модуля Turtle в Python.
10.Применение модуля Turtle в образовании: анализ преимуществ использования Turtle для обучения программированию и геометрии.
11.Графическое программирование и дети: Влияние модуля Turtle на развитие логического мышления у детей.
12.Модуль Turtle и исследование алгоритмов: Анализ того, как модуль Turtle может быть использован для демонстрации и исследования различных алгоритмов.
13.Графическое программирование и творчество.
14.Применение графического программирования в развлекательных целях.
15.Обучение геометрии через программирование: Исследование использования модуля Turtle для обучения геометрии.
16.Работа с файлами в Python: Основные аспекты и методы.
17.Объяснение различий между текстовыми и бинарными файлами.
Какие данные могут храниться в каждом из них.
18.Основные операции с файлами.Подробное описание операций: создание, открытие, чтение, запись и закрытие файлов.
19.Роль бинарных файлов в хранении и обработке данных.
20.Методы чтения и записи бинарных файлов в программировании.
21.Сериализация данных: преимущества и применение.
22.Модуль pickle в Python: использование для сохранения и восстановления данных.
23.Основы создания функций и модулей в программировании.
24.Обработка ошибок при работе с файлами в Python.

25.Сравнительный анализ работы с текстовыми и бинарными файлами в Python.

Тестовые задания

На лямбда функции:

1.Что такое лямбда-выражение в Python?

A: Массив данных

B: Анонимная функция

C: Цикл программы

2.Каким образом можно включить функцию в качестве аргументов в другие функции Python?

A:С помощью циклов

B:С помощью списков

C: С помощью функций высшего порядка

3.Чем функции высшего порядка отличаются от обычных функций Python?

A: Они не могут принимать аргументы

B:Они могут вернуться только числа

C:Они могут использовать другие функции в качестве аргументов.

4.Как создать заключение на Python?

A: С помощью цикла для

B: С помощью функции print()

C:Путем включения функции в другую функцию

5.Какие преимущества имеют заключения в Python?

A: Ускорение выполнения программы

B: Возможность доступа к переменным во внешней области.

C: Упрощение синтаксиса программы

6.Как можно вызвать лямбда-функцию в Python?

A: С помощью оператора, если

B: С помощью оператора для

C:С помощью оператора вызвать функцию ()

7.Какая из следующих концепций не связана с функциональным программированием на Python?

A: Анонимные функции

B:Циклы

C:Замыкания

8.Как можно передать функцию в качестве аргумента в функцию?

A: Обернуть ее в кавычки

B:Просто укажите ее имя

C:Используйте функцию имени без скобок

9.Каковы преимущества анонимных функций Python?

A: Они могут быть очень длинными

B:Они могут использоваться внутри списков

C:Они обычно короткие и лаконичные

Правильный ответ: в) Обычно они короткие и лаконичные.

10.Какой из следующих вариантов приводит к заключению?

A: Вид цикла на Python

B: Возможность создания функций на Python

C:Функция «Сохранение состояния» внутри функции Python.

На графику Turtle:

1.Как используется команда перемещения черепашек вперед на несколько шагов?

A: назад(n)

B: вправо(угол)

C: вперед(n)

2.Какое использование команды для рисования закрашенных регионов?

A: down()

B: fill(f)

C: круг(r)

3.Какие средства используются для очистки экрана?

A: сброс()

B: очистка()

C: вверх()

4.Какие используются для установки консоли после черепашки?

A: цвет(a)

B: ширина(n)

C: запись(и)

5.Что используется для рисования круга?

A: вперед(н)

B: круг(r)

C: вправо(угол)

6.Какой цвет можно установить черепашке для рисования?

A: «черный»

B: «фиолетовый»

C: «белый»

7.Какие используются для рисования дуги?

A: goto(x,y)

B: круг(r,угол)

C: градусов()

8.Какие меры используются для установки угловых углов радианы?

A: радианы()

B: ширина(n)

C: трассировщик(f)

9.Какая команда используется для рисования рамок белого цвета?

A: заливка(e)

B: цвет(a)

C: вверх()

10.Какие используются для рисования текстовой строки?

A: запись(и)
B: вперед(п)
C: назад(п)

Тест на работу с файлами:

1.Какой режим открытия файла в Python используется для записи данных?
A: 'r'
B: 'w'
C: 'a'

2.Какой метод Python используется для чтения данных из файла?
A: readline()
B: readlines()
C: read()

3.Какой флаг открытия файла позволяет работать с файлом как с бинарным?
A: 't'
B: 'r'
C: 'b'

4.Как закрыть файл после работы с ним в Python?
A: file.close()
B: close(file)
C: with

5.Как обработать исключение FileNotFoundError при попытке открыть несуществующий файл?
A:Не нужно обрабатывать, Python автоматически обработает ошибку.
B: Использовать блок try...except
C: Использовать break

6.Какой метод Python используется для записи строки в файл?
A: file.append()
B: file.insert()
C: file.write()

7.Как получить количество символов в файле?

A: len(file)
B: file.size()
C: len(file.read())

8.Какой метод Python используется для чтения файла по строкам?
A: readline()
B: readlines()
C: for line in file:

9.Какие данные возвращает метод readlines() при чтении файла?
A: Строку с содержимым файла
B: Список строк
C: Количество символов в файле

10.Какой режим открытия файла используется, если вы хотите добавить данные в конец существующего файла?

A: 'r'
B: 'w'
C:'a'

Список литературы и учебно-методическая литература по дисциплине, разработанная преподавателями отделения

1. David Beazley и Brian K. Jones, "Python Cookbook", Издательство: O'Reilly Media,2013 г.,стр.756
2. Mark Lutz,"Learning Python", Издательство: O'Reilly Media,1999 г.,стр.1554
3. John M. Zelle,"Python Programming: An Introduction to Computer Science",Издательство: Franklin, Beedle & Associates,2004 г, стр.538
4. Wes McKinney, "Python for Data Analysis" , Издательство: O'Reilly Media, 2012 г.,стр.486
5. Al Sweigart,"Automate the Boring Stuff with Python", Издательство: No Starch Press,2015 г.,стр.565
6. Документация Python,На веб-сайте Python

Глоссарий

1.Модуль Turtle: Модуль в стандартной библиотеке Python, который предоставляет набор инструментов для создания графических рисунков и анимаций с использованием черепашки. Он позволяет программировать движение черепашки на экране.

2.Черепашка (Turtle): Графический объект, предоставляемый модулем Turtle, который можно перемещать и настраивать для рисования различных фигур.

3.turtle.Turtle(): Конструктор для создания экземпляра черепашки. Создание экземпляра черепашки позволяет управлять его движением и рисованием.

4.forward(distance): Метод черепашки, который перемещает ее вперед на указанное расстояние (пиксели).

5.left(angle): Метод черепашки, который поворачивает ее налево на указанный угол (градусы).

6.circle(radius): Метод черепашки, который рисует окружность с указанным радиусом.

7.fillcolor(color): Метод черепашки, который устанавливает цвет заливки для фигур.

8.begin_fill() и end_fill(): Методы черепашки, используемые для начала и завершения заливки фигуры. Между ними рисуются фигуры, которые будут заполнены указанным цветом.

9.Параметры функции: Значения, которые передаются в функцию,

чтобы настроить ее поведение. В заданиях использовались параметры, такие как t, size, и color, чтобы определить черепашку, размер и цвет фигур.

10.Цикл for: Конструкция в Python, которая выполняет определенный блок кода определенное количество раз. В задании 1 использовался цикл for для рисования нескольких квадратов разных цветов.

11.Список (List): В Python это упорядоченная коллекция элементов. В заданиях использовался список colors для хранения различных цветов.

12.Радиус (Radius): Расстояние от центра окружности до ее внешней грани. В задании 2 использовался параметр radius для определения радиуса окружности.

13.Геометрическая фигура: Замкнутая форма, такая как квадрат, круг или треугольник, которую можно нарисовать с помощью черепашки.

14.Бинарные файлы: Файлы, которые хранят данные в бинарном формате, состоящем из нулей и единиц.

15. Чтение бинарных файлов: Процесс считывания данных из бинарного файла.

16. Запись в бинарные файлы: Процесс записи данных в бинарный файл.

17.Сериализация данных: Процесс преобразования структурированных данных, таких как объекты, в формат, который можно сохранить в бинарном файле.

18. Десериализация данных: Процесс восстановления данных из бинарного формата, обратно в структурированные данные.

19. Модуль pickle (Python): Встроенный модуль в Python, предоставляющий средства для сериализации и десериализации данных.

20. Функции (в программировании): Фрагменты кода, которые выполняют конкретные задачи и могут быть вызваны из других частей программы.

21. Модули (в программировании): Файлы, содержащие набор функций, классов и переменных, предназначенных для организации и структурирования кода.

22. Файловый режим (в Python):

Параметр, указывающий режим открытия файла. Например, "rb" используется для чтения бинарного файла, "wb" для записи в бинарный файл и так далее.

23. Байты (bytes): Основная единица данных в бинарных файлах. Байты представляют собой последовательность восьми битов и могут содержать числовые данные, символы и многое другое.

24.Метод read(): Метод объекта файла, используемый для чтения данных из файла. Он возвращает содержимое файла в виде строки.

25.Метод write(): Метод объекта файла, используемый для записи данных в файл.