Приложение 2.16.1к ОПОП по специальности
09.02.07 Информационные системы и программирование

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет им. Г.И. Носова»

Многопрофильный колледж

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ЛАБОРАТОРНЫХ ЗАНЯТИЙ УЧЕБНОЙ ДИСЦИПЛИНЫ

ДУП.03 Основы программирования на Python

для обучающихся специальности
09.02.07 Информационные системы и программирование

Квалификация: Программист

Магнитогорск, 2025

ОДОБРЕНО

Предметно-цикловой комиссией «Информатики и вычислительной техники»
Председатель Т.Б. Ремез
Протокол № 5 от «22» января 2025г

Методической комиссией МпК
Протокол № 3 от «19» февраля 2025г

Разработчик:
преподаватель отделения №2 «Информационных технологий и транспорта»
Многопрофильного колледжа ФГБОУ ВО «МГТУ им. Г.И. Носова»
Дувакин Андрей Андреевич

Методические указания по выполнению лабораторных работ разработаны на основе рабочей программы учебной дисциплины «Основы программирования на Python».

Содержание лабораторных работ ориентировано на подготовку обучающихся к освоению профессионального модуля программы подготовки специалистов среднего звена по специальности 09.02.07 Информационные системы и программирование и овладению профессиональными компетенциями.

СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ

Важную часть теоретической и профессиональной практической подготовки обучающихся составляют лабораторные занятия.

Состав и содержание лабораторных занятий направлены на реализацию Федерального государственного образовательного стандарта среднего профессионального образования.

Ведущей дидактической целью лабораторных занятий является экспериментальное подтверждение и проверка существенных теоретических положений.

В соответствии с рабочей программой учебной дисциплины «Основы программирования на Python» предусмотрено проведение лабораторных занятий.

В результате их выполнения, обучающийся должен:

уметь:

ПРб1 Устанавливать и настраивать Python и среду разработки, а также управлять установкой внешних модулей через pip.

ПРб2 Создавать и использовать переменные различных типов данных, выполнять операции со строками и числами, применять условные выражения.

ПРб3 Применять условные операторы и циклы для управления логикой программы, а также создавать и использовать функции, включая лямбда-функции.

ПРб4 Разрабатывать программы с использованием объектно-ориентированного программирования, включая классы, объекты, инкапсуляцию, наследование и перегрузку методов.

ПРб5 Работать с различными структурами данных, использовать списочные выражения и конструкции pattern matching.

ПРб6 Обрабатывать файлы, работать с датами, временем, архивами, а также использовать модули для решения специализированных задач.

Содержание практических и лабораторных занятий ориентировано на подготовку обучающихся к освоению профессионального модуля программы подготовки специалистов среднего звена по специальности и овладению ***профессиональными компетенциями:***

ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием

А также формированию ***общих компетенций***:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях

Выполнение обучающихся практических и/или лабораторных работ по учебной дисциплине «Основы программирования на Python» направлено на:

- обобщение, систематизацию, углубление, закрепление, развитие и детализацию полученных теоретических знаний по конкретным темам учебной дисциплины;
- формирование умений применять полученные знания на практике, реализацию единства интеллектуальной и практической деятельности;
- выработку при решении поставленных задач профессионально значимых качеств, таких как самостоятельность, ответственность, точность, творческая инициатива.

Лабораторные занятия проводятся в рамках соответствующей темы, после освоения дидактических единиц, которые обеспечивают наличие знаний, необходимых для ее выполнения.

2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Тема 1.1. Установка Python и базовые конструкции программирования

Лабораторное занятие №1
Установка Python и настройка среды разработки

Цель: Научиться устанавливать Python и настраивать среду разработки для написания программ.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях;

ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

Материальное обеспечение:

MS Windows 10, MS Office 2016, 7 Zip, Visual Studio Code, Python.

Задание:

1. Установить Python и настроить среду разработки (VS Code или PyCharm).
2. Установить внешний модуль через pip.
3. Написать и запустить простую программу с выводом текста.

Порядок выполнения работы:

1. Ознакомьтесь с инструкцией по установке Python.
2. Установите Python и проверьте версию через командную строку.
3. Настройте VS Code для работы с Python.
4. Установите модуль requests через pip.
5. Напишите программу, выводящую текст "Hello, Python!".

Ход работы:

Ознакомление с инструкцией по работе с переменными и строками

Перейдите к изучению материалов, где описаны основные типы данных в Python, включая целые числа, числа с плавающей точкой, строки и булевы значения. Особое внимание уделите операциям со строками, таким как конкатенация, форматирование (f-строки, метод .format()), а также основным строковым методам, например .upper(), .lower(), .strip(). Убедитесь, что вы понимаете, как создавать переменные и присваивать им значения разных типов данных. Эти знания необходимы для выполнения практических заданий.

Создание переменных разных типов данных

Откройте Visual Studio Code и создайте новый файл с именем variables.py. В этом файле создайте переменные различных типов: целое число (int), число с плавающей точкой (float), строка (str) и булево значение (bool). Например, создайте переменную

```
age = 20
height = 1.75
 name = "Alice"
is_student = True.
```

Проверьте тип каждой переменной с помощью функции type(). Для этого напишите код:

```
print(type(age))
print(type(height))
```

И так далее. Запустите программу, чтобы убедиться, что типы данных отображаются корректно (например, <class 'int'> для age). Сохраните файл и сделайте скриншот вывода в терминале.

Выполнение операций со строками

В том же файле variables.py выполните операции со строками. Создайте две строки, например,

```
first_name = "Alice"
```

и

```
last_name = "Smith".
```

Выполните конкатенацию строк с помощью оператора +:

```
full_name = first_name + " " + last_name
```

Затем используйте f-строку для форматирования:

```
greeting = f"Hello, {full_name}!"
```

Примените метод .upper() к строке greeting, чтобы преобразовать её в верхний регистр, и выведите результат:

```
print(greeting.upper())
```

Ожидаемый результат: `HELLO, ALICE SMITH!`. Также попробуйте метод .strip() для удаления пробелов из строки, например,

```
text = " Python "; print(text.strip())
```

Сохраните изменения и запустите программу, чтобы проверить вывод.

Создание программы с использованием строковых методов

Напишите программу, которая запрашивает у пользователя его имя через функцию input():

```
user_name = input("Введите ваше имя: ")
```

Затем создайте строку, которая использует введённое имя, например,

```
message = f"Привет, {user_name.capitalize()}!"
```

Метод .capitalize() приведёт первую букву имени к верхнему регистру. Выведите результат с помощью print(message). Для примера, если пользователь введёт alice, программа выведет Привет, Alice!. Дополнительно примените метод .replace() для замены части строки, например, замените слово "Привет" на "Здравствуй":

```
print(message.replace("Привет", "Здравствуй"))
```

Запустите программу, введите тестовое имя и сделайте скриншот вывода в терминале.

Проверка и сохранение результатов

Убедитесь, что программа работает корректно: все переменные созданы, операции со строками выполняются без ошибок, а вывод соответствует ожидаемому результату. Проверьте, что файл variables.py сохранён. Сделайте скриншот окна Visual Studio Code с открытым кодом и терминала с результатами выполнения программы. Этот скриншот будет представлен как результат лабораторной работы. Если возникают ошибки, проверьте синтаксис кода и убедитесь, что интерпретатор Python настроен правильно в VS Code.

Форма представления результата:

Работа представляется в виде скриншота настроенной среды и выполненной программы.

Критерии оценки:

Оценка «отлично» выставляется при выполнении 90-100% лабораторной работы.

Оценка «хорошо» выставляется при выполнении 80-89% лабораторной работы.

Оценка «удовлетворительно» выставляется при выполнении 70-79% лабораторной работы.

Оценка «неудовлетворительно» выставляется при выполнении менее 70% лабораторной работы.

Тема 1.1. Установка Python и базовые конструкции программирования

Лабораторное занятие №2
Работа с переменными, типами данных и строками

Цель: Освоить создание и использование переменных, работу со строками.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях;

ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

Материальное обеспечение:

MS Windows 10, MS Office 2016, 7 Zip, Visual Studio Code, Python.

Задание:

1. Создать переменные разных типов данных.
2. Выполнить конкатенацию и форматирование строк.
3. Написать программу с использованием строковых методов.

Порядок выполнения работы:

1. Ознакомьтесь с типами данных в Python.
2. Создайте переменные: int, float, str.
3. Выполните операции со строками: конкатенацию, метод .upper().
4. Напишите программу для форматирования строки с использованием f-строк.

Ознакомление с типами данных в Python

Для выполнения заданий необходимо изучить основные типы данных в Python, включая целые числа (int), числа с плавающей точкой (float), строки (str) и булевы значения (bool). Особое внимание уделите строкам, так как они будут использоваться для операций конкатенации и форматирования. Изучите встроенные мтоды строк, такие как .upper(), .lower(), .strip(), .replace(), и о способах форматирования строк, включая f-строки и метод .format(). Эти знания помогут правильно выполнить задания и написать код без ошибок.

Создание переменных разных типов данных

Откройте Visual Studio Code и создайте новый файл с именем variables.py. В этом файле определите переменные различных типов данных. Например, создайте переменную для целого числа: age = 20, для числа с плавающей точкой:

```
height = 1.75
```

для строки:

```
name = "Alice"
```

и для булевого значения:

```
is_student = True.
```

Чтобы убедиться, что переменные созданы корректно, выведите их типы с помощью функции type(). Напишите код:

```
print(type(age))
print(type(height))
print(type(name))
print(type(is_student))
```

Запустите программу, нажав правой кнопкой мыши на файл и выбрав "Run Python File in Terminal" или выполнив в терминале команду python variables.py. Ожидаемый результат в терминале: <class 'int'>, <class 'float'>, <class 'str'>, <class 'bool'>. Сохраните файл и сделайте скриншот вывода.

Выполнение операций со строками

В файле variables.py добавьте код для выполнения операций со строками. Создайте две строковые переменные, например:

```
first_name = "Alice"
```

и

```
last_name = "Smith".
```

Выполните конкатенацию строк с помощью оператора +:

```
full_name = first_name + " " + last_name
```

Выведите результат:

```
print(full_name)
```

Ожидаемый результат: Alice Smith. Затем примените метод .upper() к переменной full_name, чтобы преобразовать строку в верхний регистр:

```
print(full_name.upper())
```

Ожидаемый результат: ALICE SMITH. Для дополнительной проверки используйте метод .lower():

```
print(full_name.lower())
```

Чтобы получить alice smith. Сохраните изменения и запустите программу, чтобы проверить корректность вывода. Если возникают ошибки, убедитесь, что синтаксис правильный, и проверьте кодировку файла (должна быть UTF-8).

Написание программы для форматирования строки с использованием f-строк

В том же файле variables.py создайте программу, которая использует f-строки для форматирования текста. Добавьте код:

```
greeting = f"Hello, {full_name}! You are {age} years old."
```

и выведите его:

```
print(greeting).
```

Ожидаемый результат: Hello, Alice Smith! You are 20 years old.. Затем модифицируйте программу, чтобы она запрашивала данные у пользователя с помощью функции input(). Например:

```
user_name = input("Введите ваше имя: ")
```

и

```
user_age = int(input("Введите ваш возраст: "))
```

Создайте f-строку:

```
message = f"Привет, {user_name.capitalize()}! Тебе {user_age} лет."
```

Метод .capitalize() преобразует первую букву имени в верхний регистр. Выведите результат:

```
print(message)
```

Для теста введите имя, например, bob, и возраст, например, 25. Ожидаемый результат: Привет, Bob! Тебе 25 лет.. Дополнительно примените метод .replace() для замены слова "Привет" на "Здравствуй":

```
print(message.replace("Привет", "Здравствуй")).
```

Запустите программу и сделайте скриншот вывода.

Проверка и сохранение результатов

Проверьте, что все части программы работают корректно: переменные созданы, операции со строками выполняются без ошибок, а вывод соответствует ожидаемому. Убедитесь, что файл variables.py сохранён. Запустите программу несколько раз, вводя разные данные через input(), чтобы проверить её универсальность. Сделайте скриншот окна Visual Studio Code с открытым

кодом variables.py и терминала с результатами выполнения (включая вывод типов данных, результаты конкатенации, методов строк и f-строк). Этот скриншот будет представлен как результат лабораторной работы. Если программа выдаёт ошибки, проверьте синтаксис, корректность введённых данных (например, преобразование строки в число с помощью int()) и настройки интерпретатора Python в VS Code.

Форма представления результата:
Код программы и скриншот вывода.

Критерии оценки:
Оценка «отлично» выставляется при выполнении 90-100% лабораторной работы.
Оценка «хорошо» выставляется при выполнении 80-89% лабораторной работы.
Оценка «удовлетворительно» выставляется при выполнении 70-79% лабораторной работы.
Оценка «неудовлетворительно» выставляется при выполнении менее 70% лабораторной работы.

Тема 1.1. Установка Python и базовые конструкции программирования

Лабораторное занятие №3
Операции с числами и условные выражения

Цель: Научиться выполнять операции с числами и использовать условные выражения.
Выполнение лабораторной работы способствует формированию:
ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;
ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях;
ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.
Материальное обеспечение:
MS Windows 10, MS Office 2016, 7 Zip, Visual Studio Code, Python.

Задание:
1. Выполнить арифметические операции с числами.
2. Написать программу с условным выражением.

Порядок выполнения работы:
1. Ознакомьтесь с арифметическими операциями.
2. Создайте программу для вычисления площади прямоугольника.
3. Напишите программу, проверяющую, является ли число четным.

Ознакомление с арифметическими операциями
Перед выполнением заданий изучите основные арифметические операции в Python, включая сложение (+), вычитание (-), умножение (*), деление (/), целочисленное деление (//), остаток от деления (%) и возведение в степень (**). Обратите внимание на то, как Python обрабатывает числа с плавающей точкой, чтобы избежать ошибок округления. Эти знания необходимы для корректного вычисления в программе и понимания, как использовать операторы в условных выражениях.

Создание программы для вычисления площади прямоугольника

Откройте Visual Studio Code и создайте новый файл с именем operations.py. В этом файле напишите программу, которая запрашивает у пользователя длину и ширину прямоугольника с помощью функции input(), преобразует их в числа с плавающей точкой и вычисляет площадь. Например, добавьте код:

```
length = float(input("Введите длину прямоугольника: "))
```

и

```
width = float(input("Введите ширину прямоугольника: ")).
```

Затем вычислите площадь с помощью умножения:

```
area = length * width.
```

Выведите результат:

```
print(f"Площадь прямоугольника: {area}")
```

Для демонстрации других операций добавьте примеры в программе, такие как периметр:

```
perimeter = 2 * (length + width)
```

и выведите его:

```
print(f"Периметр прямоугольника: {perimeter}").
```

Запустите программу, введя тестовые значения, например, длину 5 и ширину 3. Ожидаемый результат: Площадь прямоугольника: 15.0 и Периметр прямоугольника: 16.0. Если ввод некорректный (например, текст вместо числа), программа выдаст ошибку; в этом случае проверьте преобразование типов. Сохраните файл и сделайте скриншот вывода в терминале.

Написание программы, проверяющей, является ли число четным

В том же файле operations.py добавьте код для второй части задания. Запросите у пользователя целое число с помощью

```
number = int(input("Введите число: "))
```

Используйте условное выражение для проверки четности: примените оператор остатка от деления %. Напишите:

```
if number % 2 == 0:
    print(f"{number} является четным числом.")
else:
    print(f"{number} является нечетным числом.")
```

Это демонстрирует базовое условное выражение с использованием оператора if-else. Для полноты добавьте пример с несколькими операторами: elif не требуется здесь, но вы можете расширить программу, чтобы проверить, положительное ли число, например:

```
if number < 0:
    print("Число отрицательное.")
elif number == 0:
    print("Число ноль.")
else:
    print("Число положительное.")
```

Запустите программу, введя тестовые значения, такие как 4 (четное), 7 (нечетное) и 0 (четное). Ожидаемый результат: для 4 — 4 является четным числом. и Число положительное.; для 7 — 7 является нечетным числом. и Число положительное.. Убедитесь, что программа обрабатывает ввод корректно, и сохраните изменения. Сделайте скриншот вывода для разных входных данных.

Проверка и сохранение результатов

Проверьте работу всей программы: убедитесь, что арифметические операции выполняются точно, условные выражения работают без ошибок, и вывод соответствует введённым данным. Запустите operations.py несколько раз с разными значениями, чтобы протестировать устойчивость к различным вводам (например, дробные числа для площади или отрицательные для четности). Если возникают ошибки, такие как ValueError при неверном вводе, добавьте обработку исключений в будущем, но на этом этапе сосредоточьтесь на базовой логике. Сохраните файл и

сделайте скриншот окна Visual Studio Code с открытым кодом operations.py и терминала с результатами выполнения (включая вычисления площади и проверку четности). Этот скриншот будет представлен как результат лабораторной работы.

Форма представления результата:
Код программы и скриншот вывода.

Критерии оценки:
Оценка «отлично» выставляется при выполнении 90-100% лабораторной работы.
Оценка «хорошо» выставляется при выполнении 80-89% лабораторной работы.
Оценка «удовлетворительно» выставляется при выполнении 70-79% лабораторной работы.
Оценка «неудовлетворительно» выставляется при выполнении менее 70% лабораторной работы.

Тема 1.2. Управляющие конструкции и функции

Лабораторное занятие №4
Создание программ с использованием условных операторов и циклов

Цель: Освоить условные операторы и циклы для управления логикой программы.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях;

ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

Материальное обеспечение:

MS Windows 10, MS Office 2016, 7 Zip, Visual Studio Code, Python.

Задание:

1. Написать программу с использованием условных операторов.
2. Создать программу с циклами for и while.

Порядок выполнения работы:

1. Ознакомьтесь с условными операторами и циклами.
2. Напишите программу для определения оценки по баллам.
3. Напишите программу для вывода чисел от 1 до 10 с помощью цикла.

Ознакомление с условными операторами и циклами

Перед началом выполнения заданий изучите условные операторы (if, elif, else) и циклы (for, while) в Python. Убедитесь, что вы понимаете, как использовать условные операторы для управления потоком программы, включая вложенные условия и логические операторы (and, or, not). Изучите циклы: for для итерации по последовательностям и while для выполнения действий до выполнения определённого условия. Обратите внимание на операторы break и continue, которые позволяют прерывать или пропускать итерации цикла. Эти знания помогут вам правильно реализовать логику программ и избежать ошибок, таких как бесконечные циклы.

Создание программы для определения оценки по баллам

Откройте Visual Studio Code и создайте новый файл с именем control_flow.py. Напишите программу, которая запрашивает у пользователя количество баллов (от 0 до 100) с помощью функции input() и преобразует ввод в целое число:

```
score = int(input("Введите количество баллов (0-100): ")).
```

Используйте условные операторы для определения оценки по следующей шкале: 90–100 — «A», 80–89 — «B», 70–79 — «C», 60–69 — «D», менее 60 — «F». Реализуйте это с помощью конструкции if-elif-else. Пример кода:

```
if score >= 90:
     print("Оценка: A")
elif score >= 80:
     print("Оценка: B")
elif score >= 70:
     print("Оценка: C")
elif score >= 60:
     print("Оценка: D")
else:
```

```
    print("Оценка: F").
```

Добавьте проверку на корректность ввода: если баллы вне диапазона 0–100, выведите сообщение об ошибке:

```
if score < 0 or score > 100:
print("Ошибка: баллы должны быть в диапазоне 0-100").
```

Запустите программу, введя тестовые значения, например, 95, 85, 65 и -5. Ожидаемые результаты: Оценка: A для 95, Оценка: B для 85, Оценка: D для 65 и Ошибка: баллы должны быть в диапазоне 0-100 для -5. Сохраните файл и сделайте скриншот вывода в терминале.

Создание программы для вывода чисел от 1 до 10 с помощью циклов

В том же файле control_flow.py добавьте код для вывода чисел от 1 до 10 с использованием циклов for и while. Для цикла for напишите:

```
print("Числа от 1 до 10 (цикл for):")
```

и

```
for i in range(1, 11):
    print(i, end=" ").
```

Аргумент end=" " в функции print позволяет вывести числа в одной строке с пробелом между ними. Ожидаемый результат: Числа от 1 до 10 (цикл for): 1 2 3 4 5 6 7 8 9 10. Для цикла while добавьте:

```
print("\nЧисла от 1 до 10 (цикл while):")
i = 1
while i <= 10:
    print(i, end=" ")
    i += 1.
```

Переменная i увеличивается на 1 на каждой итерации, а \n в print добавляет перенос строки перед выводом. Ожидаемый результат аналогичен: Числа от 1 до 10 (цикл while): 1 2 3 4 5 6 7 8 9 10. Для дополнительной практики модифицируйте цикл for, чтобы выводить только чётные числа:

```
for i in range(2, 11, 2):
    print(i, end=" ").
```

Ожидаемый результат: 2 4 6 8 10. Запустите программу и проверьте, что оба цикла работают корректно. Сделайте скриншот вывода.

Проверка и сохранение результатов

Убедитесь, что программа control_flow.py выполняет все задачи: корректно определяет оценку по баллам и выводит числа с помощью обоих циклов. Проверьте устойчивость программы, вводя разные значения для оценки, включая некорректные (например, 150 или отрицательные числа). Если возникают ошибки, например, ValueError при вводе нечисловых данных, убедитесь, что вы используете int() для преобразования ввода, и в будущем можно добавить обработку исключений. Сохраните файл и запустите программу несколько раз с разными входными данными. Сделайте скриншот окна Visual Studio Code с открытым кодом и терминала с результатами выполнения (включая вывод оценок и чисел от 1 до 10). Этот скриншот будет представлен как результат лабораторной работы.

Форма представления результата:

Код программы и скриншот вывода.

Критерии оценки:

Оценка «отлично» выставляется при выполнении 90-100% лабораторной работы.

Оценка «хорошо» выставляется при выполнении 80-89% лабораторной работы.

Оценка «удовлетворительно» выставляется при выполнении 70-79% лабораторной работы.

Оценка «неудовлетворительно» выставляется при выполнении менее 70% лабораторной работы.

Тема 1.2. Управляющие конструкции и функции

Лабораторное занятие №5
Разработка функций и лямбда-функций

Цель: Научиться создавать и использовать функции, включая лямбда-функции.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях;

ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

Материальное обеспечение:

MS Windows 10, MS Office 2016, 7 Zip, Visual Studio Code, Python.

Задание:

1. Создать функцию для вычисления факториала.
2. Написать лямбда-функцию для умножения двух чисел.

Порядок выполнения работы:

1. Ознакомьтесь с функциями.
2. Напишите функцию для вычисления факториала числа.
3. Создайте лямбда-функцию для умножения.

Ознакомление с функциями

Перед началом работы изучите основы работы с функциями в Python. Убедитесь, что вы понимаете, как определять функции с помощью ключевого слова def, задавать параметры, возвращать значения с помощью return и вызывать функции. Изучите особенности лямбда-функций, которые представляют собой анонимные функции, создаваемые с использованием ключевого слова lambda. Обратите внимание на различия между обычными функциями и лямбда-функциями, включая их синтаксис и ограничения (например, лямбда-функции обычно используются для коротких операций). Эти знания помогут вам правильно реализовать задания и избежать ошибок при написании кода.

Написание функции для вычисления факториала числа

Откройте Visual Studio Code и создайте новый файл с именем functions.py. Напишите функцию для вычисления факториала числа, используя рекурсивный подход. Факториал числа n (обозначается n!) — это произведение всех целых чисел от 1 до n. Определите функцию:

```
def factorial(n):
    return 1 if n == 0 else n * factorial(n - 1).
```

Эта функция проверяет, равно ли входное число 0 (базовый случай, где факториал равен 1), и в противном случае вызывает саму себя с уменьшенным аргументом. Для проверки добавьте код, который запрашивает у пользователя целое число:

```
number = int(input("Введите число для вычисления факториала: ")).
```

Убедитесь, что число неотрицательное, добавив проверку:

```
if number < 0:
    print("Факториал не определён для отрицательных чисел")
else:
    print(f"Факториал {number} равен {factorial(number)}").
```

Запустите программу, введя тестовые значения, например, 5, 0 и -1. Ожидаемые результаты: для 5 — Факториал 5 равен 120, для 0 — Факториал 0 равен 1, для -1 — Факториал не определён для отрицательных чисел. Сохраните файл и сделайте скриншот вывода в терминале. Если возникает ошибка ValueError, проверьте, что пользователь вводит целое число.

Создание лямбда-функции для умножения двух чисел

В том же файле functions.py добавьте код для создания лямбда-функции, выполняющей умножение двух чисел. Определите лямбда-функцию:

```
multiply = lambda x, y: x * y.
```

Эта функция принимает два аргумента x и y и возвращает их произведение. Для проверки вызовите лямбда-функцию с тестовыми значениями: `print("Произведение 3 и 4 равно", multiply(3, 4))`. Ожидаемый результат: Произведение 3 и 4 равно 12. Затем расширьте программу, чтобы пользователь мог ввести два числа:

```
a = float(input("Введите первое число: "))
```

и

```
b = float(input("Введите второе число: ")).
```

Выведите результат:

```
print(f"Произведение {a} и {b} равно {multiply(a, b)}")
```

Использование float вместо int позволяет обрабатывать как целые, так и дробные числа. Запустите программу с тестовыми значениями, например, 2.5 и 4. Ожидаемый результат: Произведение 2.5 и 4.0 равно 10.0. Для дополнительной практики создайте ещё одну лямбда-функцию, например, для возведения в квадрат:

```
square = lambda x: x ** 2
```

и протестируйте её:

```
print("Квадрат 5 равен", square(5)).
```

Ожидаемый результат: Квадрат 5 равен 25. Сохраните изменения и сделайте скриншот вывода.

Проверка и сохранение результатов

Убедитесь, что программа functions.py корректно выполняет обе задачи: вычисляет факториал и умножает числа с помощью лямбда-функции. Проверьте устойчивость программы, вводя разные значения, включая пограничные случаи (0 для факториала, отрицательные числа, дробные числа для умножения). Если возникают ошибки, например, RecursionError при большом числе для факториала, это может указывать на ограничения рекурсии; в таком случае можно переписать функцию факториала с использованием цикла for. Сохраните файл и запустите программу несколько раз с разными входными данными. Сделайте скриншот окна Visual Studio Code с открытым кодом functions.py и терминала с результатами выполнения (включая вывод факториала и результатов лямбда-функции). Этот скриншот будет представлен как результат лабораторной работы.

Форма представления результата:

Код программы и скриншот вывода.

Критерии оценки:

Оценка «отлично» выставляется при выполнении 90-100% лабораторной работы.

Оценка «хорошо» выставляется при выполнении 80-89% лабораторной работы.

Оценка «удовлетворительно» выставляется при выполнении 70-79% лабораторной работы.

Оценка «неудовлетворительно» выставляется при выполнении менее 70% лабораторной работы.

Тема 1.2. Управляющие конструкции и функции

Лабораторное занятие №6
Разработка функций и лямбда-функций

Цель: Изучение области видимости переменных, декораторов и замыканий.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях;

ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

Материальное обеспечение:

MS Windows 10, MS Office 2016, 7 Zip, Visual Studio Code, Python.

Задание:

1. Создать программу с локальной и глобальной переменной.
2. Реализовать простой декоратор.

Порядок выполнения работы:

1. Ознакомьтесь с областью видимости и декораторами.
2. Напишите программу с глобальной и локальной переменной.
3. Создайте декоратор для подсчета времени выполнения функции.

Ознакомление с областью видимости и декораторами

Перед выполнением заданий изучите область видимости переменных и декораторы в Python. Убедитесь, что вы понимаете разницу между локальными и глобальными переменными: локальные переменные создаются внутри функции и доступны только в ней, тогда как глобальные переменные определены вне функций и доступны повсеместно, если не перекрыты локальными. Изучите ключевое слово global для изменения глобальных переменных внутри функций. Декораторы — это функции, которые оборачивают другие функции, добавляя дополнительную функциональность, например, измерение времени выполнения или логирование. Прочитайте о синтаксисе декораторов с использованием символа @ и о замыканиях, которые позволяют функции запоминать значения переменных из внешней области видимости. Эти знания необходимы для корректной реализации заданий.

Написание программы с глобальной и локальной переменной

Откройте Visual Studio Code и создайте новый файл с именем scope_decorators.py. Определите глобальную переменную, например, counter = 0, которая будет отслеживать количество вызовов функции. Создайте функцию, которая использует как локальную, так и глобальную переменную. Например, напишите:

```
def increment_counter():
     global counter
     local_var = 10
     counter += 1
     print(f"Локальная переменная: {local_var}, Глобальная переменная:
     {counter}").
```

Ключевое слово global позволяет изменять значение глобальной переменной counter внутри функции. Вызовите функцию несколько раз:

```
increment_counter()
```

```
increment_counter()
```

Ожидаемый результат: Локальная переменная: 10, Глобальная переменная: 1 при первом вызове и Локальная переменная: 10, Глобальная переменная: 2 при втором. Для демонстрации области видимости попробуйте вывести local_var вне функции, добавив print(local_var) после вызовов функции — это вызовет ошибку NameError, так как local_var доступна только внутри функции. Запустите программу, чтобы проверить поведение переменных, и сохраните файл. Сделайте скриншот вывода в терминале, включая сообщения об ошибке, если они возникли, для демонстрации области видимости.

Создание декоратора для подсчёта времени выполнения функции

В том же файле scope_decorators.py создайте декоратор, который измеряет время выполнения функции. Для этого используйте модуль time. Определите декоратор:

```
import time
def timing_decorator(func):
     def wrapper():
          start_time = time.time()
          func()
          end_time = time.time()
          print(f"Время выполнения {func.__name__}: {end_time -
          start_time:.4f} секунд")
          return wrapper.
```

Этот декоратор записывает время перед и после вызова функции, вычисляет разницу и выводит её. Создайте тестовую функцию, например, вычисляющую сумму чисел от 1 до 1000000:

```
@timing_decorator
def calculate_sum():
     total = sum(range(1000000))
     print(f"Сумма: {total}").
```

Символ @timing_decorator автоматически применяет декоратор к функции calculate_sum. Вызовите функцию: calculate_sum(). Ожидаемый результат: сначала выводится сумма (Сумма: 499999500000), затем время выполнения, например, Время выполнения calculate_sum: 0.0156 секунд. Для дополнительной практики создайте ещё одну функцию с декоратором, например,

```
def slow_function():
     time.sleep(1); print("Медленная функция выполнена")
```

И примените к ней декоратор:

```
@timing_decorator
slow_function()
```

Ожидаемый результат: Медленная функция выполнена и время выполнения slow_function: 1.0023 секунд. Запустите программу и проверьте вывод. Сохраните изменения и сделайте скриншот вывода.

Проверка и сохранение результатов

Убедитесь, что программа scope_decorators.py корректно демонстрирует работу с глобальными и локальными переменными и декоратором. Проверьте, что глобальная переменная counter увеличивается при каждом вызове функции, а попытка доступа к локальной переменной вне функции вызывает ошибку. Убедитесь, что декоратор правильно измеряет время выполнения обеих функций, а вывод времени отображается с точностью до четырёх знаков после запятой. Запустите программу несколько раз, чтобы проверить стабильность результатов. Если возникают ошибки, например, NameError при неправильном использовании переменных или проблемы с модулем time, убедитесь, что вы импортировали необходимые модули и правильно определили декоратор. Сохраните файл и сделайте скриншот окна Visual Studio Code с открытым кодом scope_decorators.py и терминала с результатами выполнения (включая вывод переменных и

времени выполнения функций). Этот скриншот будет представлен как результат лабораторной работы.

Форма представления результата:
Код программы и скриншот вывода.

Критерии оценки:
Оценка «отлично» выставляется при выполнении 90-100% лабораторной работы.
Оценка «хорошо» выставляется при выполнении 80-89% лабораторной работы.
Оценка «удовлетворительно» выставляется при выполнении 70-79% лабораторной работы.
Оценка «неудовлетворительно» выставляется при выполнении менее 70% лабораторной работы.

Тема 2.1. Объектно-ориентированное программирование

Лабораторное занятие №7
Создание классов и объектов, применение инкапсуляции

Цель: Научиться создавать классы, объекты и применять инкапсуляцию.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях;

ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

Материальное обеспечение:

MS Windows 10, MS Office 2016, 7 Zip, Visual Studio Code, Python.

Задание:

1. Создать класс с атрибутами и методами.
2. Реализовать приватный атрибут.

Порядок выполнения работы:

1. Ознакомьтесь с ООП.
2. Создайте класс Student с атрибутами name и _grade.
3. Реализуйте метод для получения приватного атрибута.

Ознакомление с объектно-ориентированным программированием

Для успешного выполнения заданий изучите основы объектно-ориентированного программирования (ООП) в Python. Убедитесь, что вы понимаете концепции классов, объектов, атрибутов и методов. Классы определяют шаблон для создания объектов, которые являются экземплярами класса. Атрибуты — это данные, связанные с объектом, а методы — функции, которые работают с этими данными. Особое внимание уделите инкапсуляции, которая позволяет скрывать данные с помощью приватных атрибутов (используя префикс _ или __). Изучите, как создавать методы для доступа к приватным атрибутам (геттеры) и их модификации (сеттеры). Эти знания помогут корректно реализовать задания и избежать ошибок при работе с классами.

Создание класса Student с атрибутами и методами

Откройте Visual Studio Code и создайте новый файл с именем student.py. Определите класс Student, который будет содержать атрибуты name (имя студента) и _grade (оценка, приватный атрибут). В конструкторе __init__ задайте эти атрибуты. Добавьте метод для вывода информации о студенте:

```
class Student:
    def __init__(self, name, grade):
        self.name = name
        self._grade = grade

    def display_info(self):
        return f"Студент: {self.name}, Оценка: {self._grade}"
```

Создайте объект класса Student, например,

```
student1 = Student("Alice", 85)
```

И вызовите метод display_info:

```
print(student1.display_info())
```

Ожидаемый результат: Студент: Alice, Оценка: 85. Для проверки создайте ещё один объект, например:

```
student2 = Student("Bob", 92)
```

И вызовите метод:

```
print(student2.display_info())
```

Запустите программу, чтобы убедиться, что атрибуты и метод работают корректно. Сохраните файл и сделайте скриншот вывода в терминале.

Реализация приватного атрибута и метода для его получения

В том же файле student.py модифицируйте класс Student, чтобы атрибут _grade был защищённым, и добавьте метод-геттер для доступа к нему. Также добавьте метод-сеттер для изменения оценки с проверкой корректности (например, оценка должна быть в диапазоне 0–100). Обновите код класса:

```
class Student:
    def __init__(self, name, grade):
        self.name = name
        self._grade = grade

    def get_grade(self):
        return self._grade

    def set_grade(self, grade):
        if 0 <= grade <= 100:
            self._grade = grade
            return f"Оценка для {self.name} обновлена: {self._grade}"
        else:
            return "Ошибка: оценка должна быть в диапазоне 0-100"

    def display_info(self):
        return f"Студент: {self.name}, Оценка: {self._grade}"
```

Добавьте код для тестирования приватного атрибута и методов:

```
student1 = Student("Alice", 85)
print(student1.display_info())  # Вывод информации
print(student1.get_grade())    # Получение оценки
print(student1.set_grade(95))  # Установка новой оценки
print(student1.display_info()) # Проверка обновления
print(student1.set_grade(150)) # Проверка некорректной оценки
```

Запустите программу. Ожидаемый результат:

```
Студент: Alice, Оценка: 85
85
Оценка для Alice обновлена: 95
Студент: Alice, Оценка: 95
Ошибка: оценка должна быть в диапазоне 0-100
```

Попробуйте напрямую обратиться к _grade, например,

```
print(student1._grade)
```

Это сработает, так как Python использует соглашение об именовании для приватности, а не строгую защиту. Однако добавьте комментарий в коде, что прямой доступ к _grade не рекомендуется, так как нарушает инкапсуляцию. Сохраните изменения и сделайте скриншот вывода.

Проверка и сохранение результатов

Проверьте, что программа student.py корректно создаёт объекты класса, отображает информацию, позволяет получать и изменять приватный атрибут _grade через методы-геттер и -сеттер. Убедитесь, что метод set_grade правильно обрабатывает некорректные значения (например, отрицательные или больше 100). Запустите программу с разными тестовыми данными, создав несколько объектов с разными именами и оценками, например,

```
student2 = Student("Bob", 70)
```

И протестируйте методы. Если возникают ошибки, проверьте синтаксис, правильность вызова методов и диапазон входных данных. Сохраните файл и сделайте скриншот окна Visual Studio Code с открытым кодом student.py и терминала с результатами выполнения (включая вывод информации, получение оценки и результат установки новой оценки). Этот скриншот будет представлен как результат лабораторной работы.

Форма представления результата:

Код программы и скриншот вывода.

Критерии оценки:

Оценка «отлично» выставляется при выполнении 90-100% лабораторной работы.

Оценка «хорошо» выставляется при выполнении 80-89% лабораторной работы.

Оценка «удовлетворительно» выставляется при выполнении 70-79% лабораторной работы.

Оценка «неудовлетворительно» выставляется при выполнении менее 70% лабораторной работы.

Тема 2.1. Объектно-ориентированное программирование

Лабораторное занятие №8
Реализация наследования и перегрузки методов

Цель: Освоить наследование и перегрузку методов в Python.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях;

ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

Материальное обеспечение:

MS Windows 10, MS Office 2016, 7 Zip, Visual Studio Code, Python.

Задание:

1. Создать базовый и производный класс.
2. Переопределить метод базового класса.

Порядок выполнения работы:

1. Ознакомьтесь с наследованием.
2. Создайте класс Animal и класс Dog, наследующий от Animal.
3. Переопределите метод в классе Dog.

Ознакомление с наследованием

Для успешного выполнения заданий изучите основы наследования в Python. Наследование позволяет создавать производные (дочерние) классы, которые наследуют атрибуты и методы базового (родительского) класса, с возможностью их переопределения или расширения. Убедитесь, что вы понимаете, как указывать базовый класс в определении производного класса с помощью синтаксиса class DerivedClass(BaseClass):. Изучите процесс переопределения методов, когда метод в дочернем классе заменяет метод базового класса с тем же именем. Также обратите внимание на использование функции super() для вызова методов базового класса из производного. Эти знания помогут корректно реализовать задания и избежать ошибок при создании иерархии классов.

Создание базового класса Animal и производного класса Dog

Откройте Visual Studio Code и создайте новый файл с именем inheritance.py. Определите базовый класс Animal, который будет содержать атрибут name и метод sound для вывода звука животного. В конструкторе __init__ задайте атрибут name. Добавьте метод description для вывода информации о животном. Затем создайте класс Dog, который наследуется от Animal:

```
class Animal:
    def __init__(self, name):
        self.name = name

    def sound(self):
        return "Какой-то звук"

    def description(self):
        return f"Животное: {self.name}, издаёт звук: {self.sound()}"
```

```
class Dog(Animal):
    pass
```

Создайте объект класса Animal, например:

```
animal = Animal("Неизвестное животное")
```

И объект класса Dog, например:

```
dog = Dog("Рекс")
```

Вызовите методы для обоих объектов:

```
print(animal.description())
```

и

```
print(dog.description()).
```

Ожидаемый результат: Животное: Неизвестное животное, издаёт звук: Какой-то звук и аналогичный вывод для dog, так как метод sound унаследован. Запустите программу, чтобы убедиться, что наследование работает корректно. Сохраните файл и сделайте скриншот вывода в терминале.

Переопределение метода в классе Dog

В файле inheritance.py модифицируйте класс Dog, чтобы переопределить метод sound, задав специфический звук для собаки. Обновите класс Dog:

```
class Dog(Animal):
    def sound(self):
        return "Гав-гав!"
```

Теперь метод sound в классе Dog переопределяет метод базового класса Animal. Для демонстрации возможности вызова метода базового класса добавьте новый метод в Dog, который использует super() для вызова оригинального метода sound. Например:

```
class Dog(Animal):
    def sound(self):
        return "Гав-гав!"

    def original_sound(self):
        return super().sound()
```

Добавьте код для тестирования:

```
animal = Animal("Неизвестное животное")
dog = Dog("Рекс")
print(animal.description())          # Вывод для базового класса
print(dog.description())            # Вывод с переопределённым методом
print(f"Оригинальный звук для {dog.name}: {dog.original_sound()}")   #
Вызов метода базового класса
```

Запустите программу. Ожидаемый результат:

```
Животное: Неизвестное животное, издаёт звук: Какой-то звук
Животное: Рекс, издаёт звук: Гав-гав!
Оригинальный звук для Рекс: Какой-то звук
```

Для дополнительной практики создайте ещё один производный класс, например, Cat, с переопределённым методом sound:

```
def sound(self):
   return "Мяу!"
```

Создайте объект

```
cat = Cat("Мурка")
```

и вызовите

```
print(cat.description())
```

Ожидаемый результат: `Животное: Мурка, издаёт звук: Мяу!`. Сохраните изменения и сделайте скриншот вывода.

Проверка и сохранение результатов

Проверьте, что программа inheritance.py корректно демонстрирует наследование и переопределение методов. Убедитесь, что объекты классов Animal, Dog и (при наличии) Cat создаются правильно, метод sound переопределяется в производных классах, а метод original_sound в классе Dog вызывает метод базового класса через super(). Запустите программу несколько раз с разными именами животных, чтобы проверить универсальность кода. Если возникают ошибки, проверьте синтаксис определения классов, правильность использования super() и вызов методов. Сохраните файл и сделайте скриншот окна Visual Studio Code с открытым кодом inheritance.py и терминала с результатами выполнения (включая вывод для всех объектов и методов). Этот скриншот будет представлен как результат лабораторной работы.

Форма представления результата:

Код программы и скриншот вывода.

Критерии оценки:

Оценка «отлично» выставляется при выполнении 90-100% лабораторной работы.

Оценка «хорошо» выставляется при выполнении 80-89% лабораторной работы.

Оценка «удовлетворительно» выставляется при выполнении 70-79% лабораторной работы.

Оценка «неудовлетворительно» выставляется при выполнении менее 70% лабораторной работы.

Тема 2.1. Объектно-ориентированное программирование

Лабораторное занятие №9
Работа с абстрактными классами и классом object

Цель: Научиться использовать абстрактные классы и класс object.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях;

ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

Материальное обеспечение:

MS Windows 10, MS Office 2016, 7 Zip, Visual Studio Code, Python.

Задание:

1. Создать абстрактный класс.
2. Переопределить метод str.

Порядок выполнения работы:

1. Ознакомьтесь с модулем abc.
2. Создайте абстрактный класс Shape с методом area.
3. Реализуйте метод str для класса.

Ознакомление с модулем abc и классом object

Для успешного выполнения заданий изучите абстрактные классы и их реализация в Python с помощью модуля abc. Абстрактные классы используются для определения шаблонов, которые не могут быть инстанцированы напрямую, а только через наследование в конкретных классах. Убедитесь, что вы понимаете, как использовать класс ABC и декоратор @abstractmethod для создания абстрактных методов, которые должны быть реализованы в дочерних классах. Также изучите класс object, который является базовым для всех классов в Python, и метод __str__, который позволяет задавать строковое представление объекта. Эти знания помогут корректно реализовать абстрактный класс и переопределить метод __str__.

Создание абстрактного класса Shape с методом area

Откройте Visual Studio Code и создайте новый файл с именем abstract.py. Импортируйте модуль abc и определите абстрактный класс Shape, который наследуется от ABC. В этом классе объявите абстрактный метод area, который должен быть реализован в дочерних классах. Пример кода:

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass
```

Попробуйте создать объект класса Shape:

```
shape = Shape()
```

Это вызовет ошибку TypeError, так как абстрактный класс нельзя инстанцировать. Создайте конкретный класс Rectangle, который наследуется от Shape и реализует метод area:

```
class Rectangle(Shape):
```

```
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height
```

Создайте объект

```
rect = Rectangle(5, 3)
```

И вызовите метод area:

```
print(f"Площадь прямоугольника: {rect.area()}")
```

Ожидаемый результат: Площадь прямоугольника: 15. Запустите программу, чтобы убедиться, что метод работает корректно, и сохраните файл.

Реализация метода __str__ для класса

В файле abstract.py модифицируйте классы Shape и Rectangle, чтобы добавить метод __str__ для строкового представления объектов. В абстрактном классе Shape можно определить базовый метод __str__, который будет переопределяться в дочерних классах. Обновите код:

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

    def __str__(self):
        return "Фигура неизвестного типа"

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

    def __str__(self):
        return f"Прямоугольник с шириной {self.width} и высотой {self.height}"
```

Для дополнительной практики создайте ещё один класс, например, Circle, также наследующийся от Shape:

```
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14159 * self.radius ** 2

    def __str__(self):
        return f"Круг с радиусом {self.radius}"
```

Добавьте код для тестирования:

```
rect = Rectangle(5, 3)
circle = Circle(4)
```

```
print(rect)                # Вывод строки для прямоугольника
print(f"Площадь: {rect.area()}")
print(circle)              # Вывод строки для круга
print(f"Площадь: {circle.area()}")
```

Запустите программу. Ожидаемый результат:

```
Прямоугольник с шириной 5 и высотой 3
Площадь: 15
Круг с радиусом 4
Площадь: 50.26544
```

Убедитесь, что метод __str__ возвращает читаемое описание объекта, а метод area корректно вычисляет площадь. Сохраните изменения и сделайте скриншот вывода.

Проверка и сохранение результатов

Проверьте, что программа abstract.py корректно демонстрирует использование абстрактного класса Shape и его дочерних классов Rectangle и Circle. Убедитесь, что попытка создать объект Shape вызывает ошибку, а дочерние классы реализуют метод area и переопределяют __str__. Запустите программу с разными значениями для Rectangle (например, width=2, height=6) и Circle (например, radius=2), чтобы проверить универсальность кода. Если возникают ошибки, проверьте, импортирован ли модуль abc, правильно ли определены абстрактные методы и корректно ли вызывается __str__. Сохраните файл и сделайте скриншот окна Visual Studio Code с открытым кодом abstract.py и терминала с результатами выполнения (включая строковое представление объектов и площади). Этот скриншот будет представлен как результат лабораторной работы.

Форма представления результата:

Код программы и скриншот вывода.

Критерии оценки:

Оценка «отлично» выставляется при выполнении 90-100% лабораторной работы.

Оценка «хорошо» выставляется при выполнении 80-89% лабораторной работы.

Оценка «удовлетворительно» выставляется при выполнении 70-79% лабораторной работы.

Оценка «неудовлетворительно» выставляется при выполнении менее 70% лабораторной работы.

Тема 2.2. Структуры данных и pattern matching

Лабораторное занятие №10
Работа со списками, кортежами и словарями

Цель: Освоить работу со списками, кортежами и словарями.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях;

ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

Материальное обеспечение:

MS Windows 10, MS Office 2016, 7 Zip, Visual Studio Code, Python.

Задание:

1. Создать список и кортеж, выполнить операции.
2. Создать словарь и получить доступ к элементам.

Порядок выполнения работы:

1. Ознакомьтесь со структурами данных.
2. Создайте список чисел и выполните сортировку.
3. Создайте словарь с данными студента.

Ознакомление со структурами данных

Перед выполнением заданий изучите основные структуры данных в Python: списки (списки — изменяемые последовательности), кортежи (кортежи — неизменяемые последовательности) и словари (словари — неупорядоченные коллекции пар ключ-значение). Убедитесь, что вы понимаете различия: списки и кортежи индексируются по порядку, словари — по ключам. Изучите операции для списков (добавление элементов с append, удаление с remove, сортировка с sort), кортежей (индексация и срезы, но без изменения) и словарей (доступ по ключу с [] или get, итерация по ключам с keys() и значениям с values()). Эти знания помогут эффективно работать с данными и избежать ошибок, таких как попытка изменить кортеж или обратиться к несуществующему ключу в словаре.

Создание списка и кортежа, выполнение операций

Откройте Visual Studio Code и создайте новый файл с именем data_structures.py. Создайте список чисел, например, numbers_list = [5, 2, 8, 1, 9], и кортеж, например:

```
numbers_tuple = (5, 2, 8, 1, 9)
```

Демонстрируйте операции: для списка выполните индексацию:

```
print(numbers_list[0])
```

Ожидаемый результат: 5. срез

```
print(numbers_list[1:3])
```

Ожидаемый результат: [2, 8]. добавьте элемент:

```
numbers_list.append(10)
```

И удалите:

```
numbers_list.remove(2)
```

Затем отсортируйте список:

```
numbers_list.sort()
```

и выведите

```
print(numbers_list)
```

Ожидаемый результат: [1, 5, 8, 9, 10]. Для кортежа покажите индексацию

```
print(numbers_tuple[2])
```

Ожидаемый результат: 8. И срез

```
print(numbers_tuple[1:3])
```

Ожидаемый результат: (2, 8)), но отметьте, что кортеж неизменяем — попытка

```
numbers_tuple.append(10)
```

Вызовет ошибку AttributeError. Пример полного кода для этого пункта:

```
numbers_list = [5, 2, 8, 1, 9]
numbers_tuple = (5, 2, 8, 1, 9)

print("Список:", numbers_list)
print("Кортеж:", numbers_tuple)
print("Первый элемент списка:", numbers_list[0])
print("Срез кортежа:", numbers_tuple[1:3])

numbers_list.append(10)
numbers_list.remove(2)
numbers_list.sort()
print("Отсортированный список:", numbers_list)
```

Запустите программу, чтобы проверить вывод, и сохраните файл. Сделайте скриншот вывода в терминале.

Создание словаря и получение доступа к элементам

В том же файле data_structures.py создайте словарь с данными студента, например:

```
student = {"name": "Alice", "age": 20, "grade": 85, "city": "Moscow"}
```

Получите доступ к элементам: выведите значение по ключу:

```
print(student["name"])
```

Ожидаемый результат: Alice. Используйте метод get для безопасного доступа:

```
print(student.get("grade", "Неизвестно"))
```

Ожидаемый результат: 85. Добавьте новый элемент:

```
student["major"] = "Computer Science"
```

И удалите:

```
del student["city"]
```

Выведите ключи:

```
print("Ключи:", list(student.keys()))
```

Ожидаемый результат: ['name', 'age', 'grade', 'major']. И значения:

```
print("Значения:", list(student.values())
```

Ожидаемый результат: ['Alice', 20, 85, 'Computer Science']. Пример кода:

```
student = {"name": "Alice", "age": 20, "grade": 85, "city": "Moscow"}

print("Словарь студента:", student)
print("Имя:", student["name"])
print("Оценка (get):", student.get("grade", "Неизвестно"))

student["major"] = "Computer Science"
del student["city"]
print("Обновлённый словарь:", student)
print("Ключи:", list(student.keys()))
print("Значения:", list(student.values()))
```

Запустите программу и убедитесь, что доступ к элементам работает корректно (если ключ отсутствует, get вернёт значение по умолчанию). Сохраните изменения и сделайте скриншот вывода.

Проверка и сохранение результатов

Проверьте, что программа data_structures.py корректно создаёт и манипулирует списками, кортежами и словарями: список изменяется и сортируется, кортеж остаётся неизменным, словарь позволяет доступ и модификацию элементов. Запустите программу несколько раз, экспериментируя с разными данными, например, другим списком чисел или данными другого студента. Если возникают ошибки, такие как KeyError в словаре или IndexError в списке/кортеже, проверьте индексы и ключи. Убедитесь, что все операции выполняются без сбоев. Сохраните файл и сделайте скриншот окна Visual Studio Code с открытым кодом data_structures.py и терминала с результатами выполнения (включая вывод списков, кортежей, словарей и их операций). Этот скриншот будет представлен как результат лабораторной работы.

Форма представления результата:

Код программы и скриншот вывода.

Критерии оценки:

Оценка «отлично» выставляется при выполнении 90-100% лабораторной работы.

Оценка «хорошо» выставляется при выполнении 80-89% лабораторной работы.

Оценка «удовлетворительно» выставляется при выполнении 70-79% лабораторной работы.

Оценка «неудовлетворительно» выставляется при выполнении менее 70% лабораторной работы.

Тема 2.2. Структуры данных и pattern matching

Лабораторное занятие №11
Использование множеств и списочных выражений

Цель: Научиться работать с множествами и списочными выражениями.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях;

ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

Материальное обеспечение:

MS Windows 10, MS Office 2016, 7 Zip, Visual Studio Code, Python.

Задание:

1. Создать множества и выполнить операции.
2. Написать списочное выражение.

Порядок выполнения работы:

1. Ознакомьтесь с множествами.
2. Создайте два множества и выполните объединение.
3. Напишите списочное выражение для квадратов чисел.

Ход работы:

Ознакомление с множествами и списочными выражениями

Для успешного выполнения заданий изучите множества и списочные выражения в Python. Множества (set) — это неупорядоченные коллекции уникальных элементов, поддерживающие операции объединения (|), пересечения (&), разности (-) и симметричной разности (^). Убедитесь, что вы понимаете, как создавать множества с помощью {} или функции set() и как выполнять операции добавления и удаления элементов. Списочные выражения (list comprehension) позволяют компактно создавать списки, заменяя циклы for. Изучите синтаксис [выражение for элемент in последовательность if условие]. Эти знания помогут корректно реализовать задания и избежать ошибок, таких как добавление дубликатов в множество или неправильное использование условий в списочных выражениях.

Создание двух множеств и выполнение объединения

Откройте Visual Studio Code и создайте новый файл с именем sets_comprehension.py. Создайте два множества с числами, например:

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
```

Выведите множества для проверки:

```
print("Множество 1:", set1)
print("Множество 2:", set2)
```

Выполните операции объединения, пересечения, разности и симметричной разности:

```
print("Объединение:", set1 | set2)
print("Пересечение:", set1 & set2)
print("Разность (set1 - set2):", set1 - set2)
print("Симметричная разность:", set1 ^ set2)
```

Для демонстрации модификации множеств добавьте элемент в set1 и удалите элемент из set2:

```
set1.add(7)
set2.discard(5)
print("После добавления 7 в set1:", set1)
print("После удаления 5 из set2:", set2)
```

Запустите программу. Ожидаемый результат:

```
Множество 1: {1, 2, 3, 4}
Множество 2: {3, 4, 5, 6}
Объединение: {1, 2, 3, 4, 5, 6}
Пересечение: {3, 4}
Разность (set1 - set2): {1, 2}
Симметричная разность: {1, 2, 5, 6}
После добавления 7 в set1: {1, 2, 3, 4, 7}
После удаления 5 из set2: {3, 4, 6}
```

Использование discard вместо remove предотвращает ошибку, если элемент отсутствует. Сохраните файл и сделайте скриншот вывода в терминале.

Написание списочного выражения для квадратов чисел

В том же файле sets_comprehension.py создайте списочное выражение для генерации списка квадратов чисел от 1 до 10:

```
squares = [x**2 for x in range(1, 11)]
print("Квадраты чисел от 1 до 10:", squares)
```

Ожидаемый результат:

```
Квадраты чисел от 1 до 10: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Для демонстрации условий создайте список квадратов только чётных чисел:

```
even_squares = [x**2 for x in range(1, 11) if x % 2 == 0]
print("Квадраты чётных чисел:", even_squares)
```

Ожидаемый результат:

```
Квадраты чётных чисел: [4, 16, 36, 64, 100]
```

Для дополнительной практики создайте список кубов чисел, кратных 3:

```
cube_multiples_of_three = [x**3 for x in range(1, 11) if x % 3 == 0]
print("Кубы чисел, кратных 3:", cube_multiples_of_three)
```

Ожидаемый результат:

```
Кубы чисел, кратных 3: [27, 216, 729]
```

Запустите программу и проверьте корректность вывода списков. Сохраните изменения и сделайте скриншот вывода.

Проверка и сохранение результатов

Проверьте, что программа sets_comprehension.py корректно выполняет операции с множествами (объединение, пересечение, разность, симметричная разность, добавление и удаление элементов) и создаёт списки с помощью списочных выражений. Запустите программу с разными множествами (например, множества строк или других чисел) и списочными выражениями (например, другие математические операции). Если возникают ошибки, такие как KeyError при использовании remove для несуществующего элемента, убедитесь, что используется discard. Проверьте синтаксис списочных выражений, особенно условий. Сохраните файл и сделайте скриншот окна Visual Studio Code с открытым кодом sets_comprehension.py и терминала с результатами выполнения (включая вывод множеств и списков). Этот скриншот будет представлен как результат лабораторной работы.

Форма представления результата:

Код программы и скриншот вывода.

Критерии оценки:

Оценка «отлично» выставляется при выполнении 90-100% лабораторной работы.

Оценка «хорошо» выставляется при выполнении 80-89% лабораторной работы.

Оценка «удовлетворительно» выставляется при выполнении 70-79% лабораторной работы.

Оценка «неудовлетворительно» выставляется при выполнении менее 70% лабораторной работы.

Тема 2.2. Структуры данных и pattern matching

Лабораторное занятие №12
Применение конструкции match и pattern matching для кортежей и массивов

Цель: Освоить конструкцию match и pattern matching.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях;

ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

Материальное обеспечение:

MS Windows 10, MS Office 2016, 7 Zip, Visual Studio Code, Python.

Задание:

1. Написать программу с использованием match.
2. Применить pattern matching для кортежа.

Порядок выполнения работы:

1. Ознакомьтесь с конструкцией match.
2. Напишите программу для обработки команды с помощью match.
3. Реализуйте pattern matching для кортежа.

Ознакомление с конструкцией match и pattern matching

Для успешного выполнения заданий изучите конструкцию match, введённую в Python 3.10 для pattern matching. Эта конструкция позволяет сопоставлять значения с различными шаблонами, такими как строки, списки, кортежи или словари, и выполнять соответствующие действия. Убедитесь, что вы понимаете синтаксис match value: case pattern: action, включая использование подстановочных знаков (_) и условий в шаблонах. Особое внимание уделите сопоставлению кортежей, например, case (x, y):, и работе с переменным количеством элементов, например, case [x, *rest]:. Эти знания помогут корректно реализовать задания и избежать ошибок при использовании match.

Написание программы для обработки команды с помощью match

Откройте Visual Studio Code и создайте новый файл с именем pattern_matching.py. Напишите программу, которая запрашивает у пользователя команду (например, "start", "stop", "pause") и обрабатывает её с помощью конструкции match. Запросите ввод команды:

```
command = input("Введите команду (start/stop/pause): ")
```

Реализуйте обработку команды с помощью match, игнорируя регистр ввода:

```
match command.lower():
    case "start":
        print("Программа запущена")
    case "stop":
        print("Программа остановлена")
    case "pause":
        print("Программа приостановлена")
    case _:
        print("Неизвестная команда")
```

Запустите программу с тестовыми командами, такими как "start", "STOP", "invalid". Ожидаемый результат:

```
Программа запущена
Программа остановлена
Неизвестная команда
```

Убедитесь, что метод .lower() корректно обрабатывает ввод в любом регистре. Сохраните файл и сделайте скриншот вывода в терминале.

Реализация pattern matching для кортежа

В том же файле pattern_matching.py добавьте код для обработки кортежа с координатами точки (x, y) с использованием match. Запросите у пользователя две координаты и создайте кортеж:

```
x = float(input("Введите координату x: "))
y = float(input("Введите координату y: "))
point = (x, y)
```

Используйте match для определения положения точки относительно начала координат:

```
match point:
    case (0, 0):
        print("Точка в начале координат")
    case (0, y):
        print(f"Точка на оси Y с координатой y={y}")
    case (x, 0):
        print(f"Точка на оси X с координатой x={x}")
    case (x, y) if x > 0 and y > 0:
        print(f"Точка в первой четверти: ({x}, {y})")
    case (x, y) if x < 0 and y > 0:
        print(f"Точка во второй четверти: ({x}, {y})")
    case (x, y) if x < 0 and y < 0:
        print(f"Точка в третьей четверти: ({x}, {y})")
    case (x, y) if x > 0 and y < 0:
        print(f"Точка в четвёртой четверти: ({x}, {y})")
    case _:
        print("Непредвиденный случай")
```

Запустите программу с тестовыми значениями, например, (0, 0), (5, 0), (-2, 3), (4, -1). Ожидаемый результат:

```
Точка в начале координат
Точка на оси X с координатой x=5.0
Точка во второй четверти: (-2.0, 3.0)
Точка в четвёртой четверти: (4.0, -1.0)
```

Для дополнительной практики добавьте обработку списка с помощью match. Запросите список чисел:

```
numbers = [int(x) for x in input("Введите числа через пробел: ").split()]
```

Реализуйте сопоставление:

```
match numbers:
    case []:
        print("Список пуст")
    case [single]:
        print(f"Один элемент: {single}")
    case [first, second, *rest]:
        print(f"Первые два элемента: {first}, {second}; Остальные: {rest}")
    case _:
        print("Список не соответствует шаблонам")
```

Протестируйте с вводами "", "5", "1 2 3 4". Ожидаемый результат:

```
Список пуст
Один элемент: 5
Первые два элемента: 1, 2; Остальные: [3, 4]
```

Сохраните изменения и сделайте скриншот вывода.

Проверка и сохранение результатов

Проверьте, что программа pattern_matching.py корректно обрабатывает команды с помощью match и выполняет pattern matching для кортежей и списков. Запустите программу с различными входными данными: командами в разном регистре, координатами (включая пограничные случаи, такие как (0, 5)) и списками чисел (пустой, с одним или несколькими элементами). Если возникают ошибки, например, ValueError при вводе нечисловых значений, проверьте преобразование с помощью float или int. Убедитесь, что используется Python версии 3.10 или выше, так как match не поддерживается в более ранних версиях. Сохраните файл и сделайте скриншот окна Visual Studio Code с открытым кодом pattern_matching.py и терминала с результатами выполнения (включая вывод для команд, координат и списков). Этот скриншот будет представлен как результат лабораторной работы.

Форма представления результата:

Код программы и скриншот вывода.

Критерии оценки:

Оценка «отлично» выставляется при выполнении 90-100% лабораторной работы.

Оценка «хорошо» выставляется при выполнении 80-89% лабораторной работы.

Оценка «удовлетворительно» выставляется при выполнении 70-79% лабораторной работы.

Оценка «неудовлетворительно» выставляется при выполнении менее 70% лабораторной работы.

Тема 2.3. Работа с файлами и модулями

Лабораторное занятие №13
Чтение и запись текстовых и CSV-файлов

Цель: Научиться работать с текстовыми и CSV-файлами.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях;

ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

Материальное обеспечение:

MS Windows 10, MS Office 2016, 7 Zip, Visual Studio Code, Python.

Задание:

1. Написать программу для чтения и записи текстового файла.
2. Создать CSV-файл и прочитать его.

Порядок выполнения работы:

1. Ознакомьтесь с модулем csv.
2. Напишите программу для записи текста в файл.
3. Создайте CSV-файл с данными и прочитайте его.

Ознакомление с модулем csv и работой с файлами

Для успешного выполнения заданий изучите основы работы с файлами в Python, включая чтение и запись текстовых файлов с использованием функций open(), read(), write() и контекстного менеджера with. Убедитесь, что вы понимаете режимы открытия файлов: "r" для чтения, "w" для записи, "a" для добавления. Изучите модуль csv, который позволяет работать с файлами в формате CSV (Comma-Separated Values), включая чтение с помощью csv.reader и запись с помощью csv.writer. Обратите внимание на обработку строк и списков при работе с CSV-файлами. Эти знания помогут корректно реализовать задания и избежать ошибок, таких как неправильный режим открытия файла или некорректная обработка данных в CSV.

Написание программы для записи текста в файл

Откройте Visual Studio Code и создайте новый файл с именем file_operations.py. Напишите программу, которая запрашивает у пользователя текст и записывает его в текстовый файл output.txt. Используйте контекстный менеджер with для безопасной работы с файлом. Затем прочитайте содержимое файла и выведите его на экранuser_text = input("Введите текст для записи в файл: ")

```
with open("output.txt", "w", encoding="utf-8") as file:
    file.write(user_text + "\n")

with open("output.txt", "r", encoding="utf-8") as file:
    content = file.read()
    print("Содержимое файла:", content):
```

Параметр encoding="utf-8" обеспечивает корректную работу с русским текстом. Запустите программу, введя, например, "Привет, это тест!". Ожидаемый результат:

```
Содержимое файла: Привет, это тест!
```

Проверьте, что файл output.txt создан в той же папке, где находится file_operations.py. Для дополнительной практики добавьте возможность записи нескольких строк:

```
lines = []
print("Введите строки (пустая строка для завершения):")
while True:
    line = input()
    if line == "":
        break
    lines.append(line)

with open("output.txt", "a", encoding="utf-8") as file:
    for line in lines:
        file.write(line + "\n")
```

Сохраните файл и сделайте скриншот вывода в терминале, а также проверьте содержимое output.txt.

Создание и чтение CSV-файла

В том же файле file_operations.py добавьте код для создания CSV-файла с данными студентов (имя, возраст, оценка) и последующего чтения его содержимого. Импортируйте модуль csv и создайте CSV-файл students.csv с несколькими записями. Затем прочитайте файл и выведите данные:

```
import csv

students = [
    ["Name", "Age", "Grade"],
    ["Alice", 20, 85],
    ["Bob", 22, 90],
    ["Charlie", 19, 78]
]

with open("students.csv", "w", newline="", encoding="utf-8") as file:
    writer = csv.writer(file)
    writer.writerows(students)

with open("students.csv", "r", encoding="utf-8") as file:
    reader = csv.reader(file)
    print("Данные из CSV-файла:")
    for row in reader:
        print(row)
```

Параметр newline="" предотвращает лишние пустые строки в CSV-файле на Windows. Запустите программу. Ожидаемый результат:

```
Данные из CSV-файла:
['Name', 'Age', 'Grade']
['Alice', '20', '85']
['Bob', '22', '90']
['Charlie', '19', '78']
```

Проверьте, что файл students.csv создан и содержит указанные данные. Для дополнительной практики добавьте код для чтения только оценок студентов (пропуская заголовок):

```
with open("students.csv", "r", encoding="utf-8") as file:
    reader = csv.reader(file)
    next(reader)  # Пропуск заголовка
    grades = [int(row[2]) for row in reader]
    print("Оценки студентов:", grades)
```

Ожидаемый результат:

```
Оценки студентов: [85, 90, 78]
```

Сохраните изменения и сделайте скриншот вывода.

Проверка и сохранение результатов

Проверьте, что программа file_operations.py корректно записывает и читает текстовый файл output.txt и создаёт и читает CSV-файл students.csv. Убедитесь, что текстовый файл содержит введённый текст и добавленные строки, а CSV-файл корректно отображает данные студентов, включая заголовок и значения. Запустите программу с разными входными данными, например, разным текстом и дополнительными записями в CSV. Если возникают ошибки, такие как UnicodeDecodeError, проверьте параметр encoding="utf-8". Убедитесь, что файлы создаются в правильной директории и доступны для чтения. Сохраните файл и сделайте скриншот окна Visual Studio Code с открытым кодом file_operations.py и терминала с результатами выполнения (включая вывод текстового файла и данных CSV). Этот скриншот будет представлен как результат лабораторной работы.

Форма представления результата:

Код программы и скриншот вывода.

Критерии оценки:

Оценка «отлично» выставляется при выполнении 90-100% лабораторной работы.

Оценка «хорошо» выставляется при выполнении 80-89% лабораторной работы.

Оценка «удовлетворительно» выставляется при выполнении 70-79% лабораторной работы.

Оценка «неудовлетворительно» выставляется при выполнении менее 70% лабораторной работы.

Тема 2.3. Работа с файлами и модулями

Лабораторное занятие №14
Работа с бинарными файлами и модулем shelve

Цель: Освоить работу с бинарными файлами и модулем shelve.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях;

ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

Материальное обеспечение:

MS Windows 10, MS Office 2016, 7 Zip, Visual Studio Code, Python.

Задание:

1. Написать программу для работы с бинарным файлом.
2. Использовать модуль shelve для хранения данных.

Порядок выполнения работы:

1. Ознакомьтесь с модулем shelve.
2. Запишите данные в бинарный файл.
3. Сохраните словарь в shelve.

Ознакомление с модулем shelve и работой с бинарными файлами

Для успешного выполнения заданий изучите работу с бинарными файлами и модулем shelve в Python. Бинарные файлы хранят данные в формате, отличном от текстового, и открываются в режиме "rb" (чтение) или "wb" (запись). Модуль shelve позволяет сохранять Python-объекты (например, словари, списки) в бинарный файл в виде ключ-значение, обеспечивая удобный доступ. Убедитесь, что вы понимаете, как использовать контекстный менеджер with для работы с файлами и shelve.open() для создания или открытия хранилища. Изучите особенности shelve, такие как необходимость закрытия хранилища и ограничения на типы данных (объекты должны быть сериализуемыми). Эти знания помогут корректно реализовать задания и избежать ошибок, таких как неправильный режим открытия файла или повреждение данных в хранилище.

Запись данных в бинарный файл

Откройте Visual Studio Code и создайте новый файл с именем binary_shelve.py. Напишите программу, которая записывает данные в бинарный файл data.bin. Например, запишите строку, преобразованную в байты, и затем прочитайте её. Используйте контекстный менеджер with для безопасной работы с файлом:

```
text = "Привет, это бинарный файл!"
binary_data = text.encode("utf-8")

with open("data.bin", "wb") as file:
    file.write(binary_data)

with open("data.bin", "rb") as file:
    content = file.read()
    decoded_content = content.decode("utf-8")
    print("Содержимое бинарного файла:", decoded_content)
```

Параметр encode("utf-8") преобразует строку в байты, а decode("utf-8") — обратно в строку. Запустите программу. Ожидаемый результат:

```
Содержимое бинарного файла: Привет, это бинарный файл!
```

Проверьте, что файл data.bin создан в той же папке, где находится binary_shelve.py. Для дополнительной практики запишите список чисел в бинарный файл, используя модуль struct для преобразования данных:

```
import struct

numbers = [1, 2, 3, 4, 5]
with open("numbers.bin", "wb") as file:
    for num in numbers:
        file.write(struct.pack("i", num))

with open("numbers.bin", "rb") as file:
    data = file.read()
    unpacked_numbers = [struct.unpack("i", data[i:i+4])[0] for i in
range(0, len(data), 4)]
    print("Числа из бинарного файла:", unpacked_numbers)
```

Ожидаемый результат:

```
Числа из бинарного файла: [1, 2, 3, 4, 5]
```

Сохраните файл и сделайте скриншот вывода в терминале.

Сохранение словаря в shelve

В том же файле binary_shelve.py добавьте код для работы с модулем shelve. Создайте словарь с данными студентов и сохраните его в хранилище students_shelf. Затем прочитайте данные и выведите их:

```
import shelve

students = {
    "Alice": {"age": 20, "grade": 85},
    "Bob": {"age": 22, "grade": 90},
    "Charlie": {"age": 19, "grade": 78}
}

with shelve.open("students_shelf") as shelf:
    for key, value in students.items():
        shelf[key] = value

with shelve.open("students_shelf") as shelf:
    print("Данные из хранилища shelve:")
    for key in shelf:
        print(f"{key}: {shelf[key]}")
```

Запустите программу. Ожидаемый результат:

```
Данные из хранилища shelve:
Alice: {'age': 20, 'grade': 85}
Bob: {'age': 22, 'grade': 90}
Charlie: {'age': 19, 'grade': 78}
```

Проверьте, что в директории созданы файлы хранилища (например, students_shelf.dat или аналогичные, в зависимости от системы). Для дополнительной практики добавьте код для обновления данных в хранилище:

```
with shelve.open("students_shelf") as shelf:
    shelf["Alice"]["grade"] = 88
    print("Обновлённая оценка Alice:", shelf["Alice"])
```

Сохраните изменения и сделайте скриншот вывода.

Проверка и сохранение результатов

Проверьте, что программа binary_shelve.py корректно записывает и читает данные из бинарного файла data.bin и numbers.bin, а также сохраняет и извлекает данные из хранилища shelve. Убедитесь, что бинарные файлы содержат корректные данные, а хранилище students_shelf позволяет добавлять и обновлять записи. Запустите программу с разными входными данными, например, другим текстом для бинарного файла или дополнительными записями в shelve. Если возникают ошибки, такие как UnicodeDecodeError или проблемы с доступом к файлам, проверьте кодировку и режимы открытия файлов. Убедитесь, что файлы создаются в правильной директории. Сохраните файл и сделайте скриншот окна Visual Studio Code с открытым кодом binary_shelve.py и терминала с результатами выполнения (включая вывод содержимого бинарных файлов и данных из shelve). Этот скриншот будет представлен как результат лабораторной работы.

Форма представления результата:

Код программы и скриншот вывода.

Критерии оценки:

Оценка «отлично» выставляется при выполнении 90-100% лабораторной работы.

Оценка «хорошо» выставляется при выполнении 80-89% лабораторной работы.

Оценка «удовлетворительно» выставляется при выполнении 70-79% лабораторной работы.

Оценка «неудовлетворительно» выставляется при выполнении менее 70% лабораторной работы.

Тема 2.3. Работа с файлами и модулями

Лабораторное занятие №15
Использование модулей random, math, locale и decimals

Цель: Научиться использовать модули random, math, locale и decimals.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях;

ПК 1.1 Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

Материальное обеспечение:

MS Windows 10, MS Office 2016, 7 Zip, Visual Studio Code, Python.

Задание:

1. Написать программу с использованием модуля random.
2. Использовать модули math и decimals для вычислений.

Порядок выполнения работы:

1. Ознакомьтесь с модулями random, math .
2. Напишите программу для генерации случайного числа.
3. Выполните вычисления с использованием math и decimals.

Ознакомление с модулями random, math, locale и decimal

Для успешного выполнения заданий изучите модули random, math, locale и decimal. Модуль random используется для генерации случайных чисел и выбора элементов, включая функции randint(), random() и choice(). Модуль math предоставляет математические функции, такие как sin(), cos(), sqrt() и константы, например, pi. Модуль decimal обеспечивает точные вычисления с десятичными числами, избегая ошибок округления, характерных для чисел с плавающей точкой. Модуль locale позволяет форматировать числа и строки в соответствии с локальными настройками, но в этой лабораторной работе он не требуется, так как задание сосредоточено на random, math и decimal. Убедитесь, что вы понимаете, как импортировать модули и использовать их функции. Эти знания помогут корректно реализовать задания и избежать ошибок, таких как неправильное использование диапазонов в random или неточности в вычислениях.

Написание программы для генерации случайного числа

Откройте Visual Studio Code и создайте новый файл с именем modules.py. Напишите программу, которая использует модуль random для генерации случайных чисел. Импортируйте модуль и сгенерируйте случайное целое число в диапазоне от 1 до 100, а также случайный элемент из списка:

```
import random

random_number = random.randint(1, 100)
print("Случайное целое число от 1 до 100:", random_number)

items = ["яблоко", "банан", "груша", "апельсин"]
random_item = random.choice(items)
print("Случайный элемент из списка:", random_item)
```

Запустите программу. Ожидаемый результат (значения будут варьироваться из-за случайности):

```
Случайное целое число от 1 до 100: 42
Случайный элемент из списка: груша
```

Для дополнительной практики сгенерируйте случайное число с плавающей точкой в диапазоне от 0 до 1:

```
random_float = random.random()
print("Случайное число с плавающей точкой (0-1):", random_float)
```

Ожидаемый результат (пример):

```
Случайное число с плавающей точкой (0-1): 0.7319452837
```

Сохраните файл и сделайте скриншот вывода в терминале.

Выполнение вычислений с использованием math и decimal

В том же файле modules.py добавьте код для выполнения математических вычислений с использованием модулей math и decimal. Например, вычислите площадь круга с заданным радиусом, используя math.pi, и выполните точные вычисления с дробями, используя decimal.Decimal:

```
import math
from decimal import Decimal, getcontext

radius = float(input("Введите радиус круга: "))
area = math.pi * math.pow(radius, 2)
print(f"Площадь круга с радиусом {radius}: {area:.4f}")

getcontext().prec = 10
price = Decimal('19.99')
tax_rate = Decimal('0.13')
total_price = price + (price * tax_rate)
print("Цена с налогом:", total_price)
```

Параметр getcontext().prec = 10 задаёт точность вычислений в модуле decimal. Запустите программу с радиусом, например, 5. Ожидаемый результат:

```
Площадь круга с радиусом 5.0: 78.5398
Цена с налогом: 22.5897
```

Для дополнительной практики вычислите синус угла и корень числа с помощью math:

```
angle_degrees = 45
angle_radians = math.radians(angle_degrees)
print(f"Синус угла {angle_degrees} градусов:",
math.sin(angle_radians))

number = 16
print(f"Квадратный корень из {number}:", math.sqrt(number))
```

Ожидаемый результат:

```
Синус угла 45 градусов: 0.7071067811865475
Квадратный корень из 16: 4.0
```

Сохраните изменения и сделайте скриншот вывода.

Проверка и сохранение результатов

Проверьте, что программа modules.py корректно генерирует случайные числа с помощью random, выбирает случайные элементы из списка и выполняет математические вычисления с использованием math и decimal. Запустите программу несколько раз, чтобы убедиться, что случайные числа и элементы варьируются, и протестируйте с разными радиусами (например, 3, 10) и другими входными данными. Если возникают ошибки, такие как ValueError при вводе нечисловых значений, проверьте преобразование ввода с помощью float. Убедитесь, что

импортированы все необходимые модули и что вычисления с decimal выполняются с заданной точностью. Сохраните файл и сделайте скриншот окна Visual Studio Code с открытым кодом modules.py и терминала с результатами выполнения (включая вывод случайных чисел, площади круга, цены с налогом и других вычислений). Этот скриншот будет представлен как результат лабораторной работы.

Форма представления результата:
Код программы и скриншот вывода.

Критерии оценки:
Оценка «отлично» выставляется при выполнении 90-100% лабораторной работы.
Оценка «хорошо» выставляется при выполнении 80-89% лабораторной работы.
Оценка «удовлетворительно» выставляется при выполнении 70-79% лабораторной работы.
Оценка «неудовлетворительно» выставляется при выполнении менее 70% лабораторной работы.