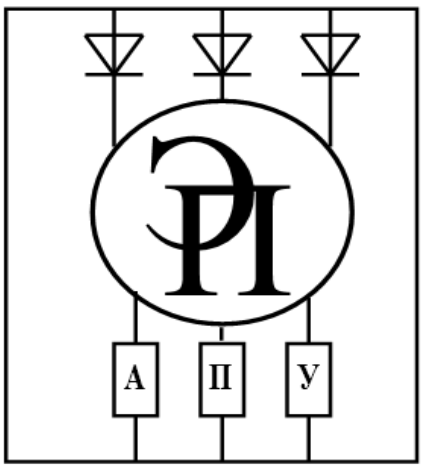
МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Электропривод и автоматизация промышленных установок»

РАЗРАБОТКА ПРОФЕССИОНАЛЬНЫХ РЕШЕНИЙ

Методические рекомендации к лабораторным работам для студентов направления подготовки 13.03.02 «Электроэнергетика и электротехника» очной формы обучения



Могилев 2023

УДК 658.26
ББК 31.19
Р75

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Электропривод и автоматизация промышленных установок» «2» октября 2023 г., протокол № 2

Составитель ст. преподаватель Т. С. Ларькина

Рецензент канд. техн. наук, доц. С. В. Болотов

Методические рекомендации к лабораторным работам предназначены для студентов направления подготовки 13.03.02 «Электроэнергетика и электротехника» очной формы обучения.

Учебное издание

РАЗРАБОТКА ПРОФЕССИОНАЛЬНЫХ РЕШЕНИЙ

Ответственный за выпуск	А. С. Коваль
Корректор	И. В. Голубцова
Компьютерная верстка	Е. В. Ковалевская

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 36 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Целью учебной дисциплины «Моделирование в электроприводе» является представление о принципах разработки профессиональных приложений на современном языке программирования для решения задач профессиональной деятельности (ПД).

В результате освоения учебной дисциплины студент

познает:

– процессы, методы поиска, сбора, хранения, обработки, предоставления, распространения информации и способы осуществления таких процессов и методов (информационные технологии);

– логику построения и принципы функционирования современных языков программирования, сред разработки информационных систем и технологий, принципы разработки алгоритмов и компьютерных программ;

– современные языки программирования, среды разработки информационных систем и технологий;

научится:

– выбирать языки программирования, среды разработки информационных систем и технологий исходя из имеющихся задач;

– применять современные языки программирования для разработки оригинальных алгоритмов и компьютерных программ, пригодных для практического применения;

– применять современные программные среды разработки информационных систем и технологий;

– понимать исходные коды программных продуктов, написанных на освоенных языках программирования, и вносить требуемые изменения;

– анализировать профессиональные задачи, разрабатывать подходящие ИТ-решения;

– самостоятельно осваивать новые для себя современные языки программирования, среды разработки информационных технологий и систем;

овладеет:

– навыками разработки профессиональных приложений, пригодных для практического применения;

– навыками отладки и тестирования прототипов программно-технических комплексов.

Знания, полученные в результате изучения курса, необходимы для научной и практической деятельности инженера и будут применены при прохождении учебной и производственных практик, а также при подготовке выпускной квалификационной работы и в дальнейшей профессиональной деятельности.

Правила техники безопасности при работе в компьютерном классе

Общие положения

1 К работе в компьютерном классе допускаются только студенты, прошедшие инструктаж по технике безопасности, соблюдающие указания преподавателя, расписавшиеся в журнале регистрации инструктажа.

2 Работа студентов в компьютерном классе разрешается только в присутствии преподавателя.

3 Во время занятий посторонние лица могут находиться в классе только с разрешения преподавателя.

4 Во время перемен между занятиями проводится обязательное проветривание компьютерного кабинета с обязательным выходом студентов из класса.

5 Каждый студент в ответе за состояние своего рабочего места и сохранность размещенного на нем оборудования.

6 Необходимо неукоснительно соблюдать правила по технике безопасности, т. к. нарушение этих правил может привести к поражению электрическим током, вызвать возгорание и навредить здоровью.

При эксплуатации оборудования необходимо остерегаться:

– поражения электрическим током;

– механических повреждений, травм.

***Перед началом работы необходимо*:**

– убедиться в отсутствии видимых повреждений на рабочем месте;

– разместить на столе тетради, учебные пособия так, чтобы они не мешали работе на компьютере;

– принять правильную рабочую позу;

– посмотреть на индикатор монитора и системного блока и определить, включен или выключен компьютер. Переместить мышь, если компьютер находится в энергосберегающем состоянии, или включить монитор, если он был выключен.

***При работе в компьютерном классе категорически запрещается*:**

– находиться в классе в верхней или во влажной одежде;

– класть одежду и сумки на столы;

– находиться в классе с напитками и едой;

– располагаться сбоку или сзади от включенного монитора;

– присоединять или отсоединять кабели, трогать разъемы, провода и розетки;

– передвигать компьютеры и мониторы;

– открывать системный блок;

– прикасаться к экрану монитора;

– включать и выключать компьютеры самостоятельно;

– пытаться самостоятельно устранять неисправности в работе аппаратуры;

– перекрывать вентиляционные отверстия на системном блоке и мониторе;

– ударять по клавиатуре, бесцельно нажимать на клавиши;

– класть книги, тетради и другие вещи на клавиатуру, монитор и системный блок;

– удалять и перемещать чужие файлы;

– использовать дискеты, CD-, DVD-диски, USB-флэш-диски без разрешения преподавателя. Если такое разрешение получено, то перед работой необходимо проверить их на наличие вредоносного ПО с помощью антивирусных программ;

– приносить и запускать компьютерные игры;

– работать при плохом самочувствии;

– вставать без разрешения преподавателя со своих мест, когда входят посетители;

– мешать работе других студентов и преподавателя.

***Находясь в компьютерном классе, студенты обязаны*:**

– соблюдать тишину и порядок;

– выполнять требования преподавателя;

– находясь в сети, работать только под своим именем и паролем;

– соблюдать режим работы (согласно п. 9.4.2 Санитарных правил и норм);

– при появлении рези в глазах, резком ухудшении видимости, невозможности сфокусировать взгляд или навести его на резкость, появлении боли в пальцах и кистях рук, усилении сердцебиения немедленно покинуть рабочее место, сообщить о происшедшем преподавателю и обратиться к врачу;

– после окончания работы завершить все активные программы и корректно выключить компьютер;

– оставить рабочее место чистым, в определенном порядке, принятом в компьютерном классе.

***Работая за компьютером, необходимо соблюдать следующие правила*:**

– расстояние от экрана до глаз – 70…80 см (расстояние вытянутой руки);

– вертикально прямая спина;

– плечи опущены и расслаблены;

– ноги на полу и не скрещены;

– локти, запястья и кисти рук на одном уровне;

– локтевые, тазобедренные, коленные, голеностопные суставы под прямым углом.

***Во время работы*:**

– строго выполняйте все указанные выше правила, а также текущие указания преподавателя;

– следите за исправностью аппаратуры и немедленно прекращайте работу при появлении необычного звука или самопроизвольного отключения аппаратуры. Немедленно докладывайте об этом преподавателю;

– плавно нажимайте на клавиши, не допуская резких ударов;

– не пользуйтесь клавиатурой, если не подключено напряжение;

– работайте на клавиатуре чистыми руками;

– никогда не пытайтесь самостоятельно устранить неисправность в работе аппаратуры;

– не вставайте со своих мест, когда в кабинет входят посетители.

***По окончании работы*:** закройте все приложения.

***Требования безопасности в аварийных ситуациях*:**

– при появлении программных ошибок или сбоях оборудования учащийся должен немедленно обратиться к преподавателю;

– при появлении запаха гари, необычного звука немедленно прекратить работу и сообщить преподавателю.

1 Лабораторная работа № 1. Программирование структур в приложении

Цель работы: освоить принципы и навыки программирования структур в приложении с использованием языка программирования Python.

1.1 Порядок выполнения работы

1.1.1 Подготовить окружение для разработки (установить программы, необходимые для программирования на языке Python).

1.1.2 Создать и разработать классы и их методы.

1.1.3 Скомпилировать и протестировать приложение.

1.1.4 Сделать выводы.

Важно! *Возможно обойтись без необходимости установки дополнительных программ на компьютер*, *предпочитая воспользоваться веб-интерфейсом онлайн-среды разработки для Python.*

1.2 Основные теоретические положения

Python – это высокоуровневый язык программирования, известный своей простотой и интуитивностью. В данной лабораторной работе фокусируемся на разработке и реализации методов классов, что позволяет эффективно манипулировать данными в созданном на Python приложении. Методы классов позволяют организовать и автоматизировать работу с информацией в структуре данных приложения.

Определение классов и их атрибутов.

Классы представляют абстракцию данных и функциональности, которые будут использоваться в приложении. Для примера рассмотрим класс «Студент», который будет иметь атрибуты «имя», «возраст» и «средний балл»:

```
1: class Student:
2:     def __init__(self, name, age, average_grade):
3:         self.name = name
4:         self.age = age
5:         self.average_grade = average_grade
```

Методы и их реализация.

Методы классов определяют поведение объектов данного класса. Например, добавим метод «получить_информацию», который будет выводить информацию о студенте:

```
class Student:
    # ... (код из предыдущего примера)

    def get_info(self):
        return f»Имя: {self.name}, Возраст: {self.age}, Средний балл:
{self.average_grade}»
```

Создание экземпляров класса и вызов методов.

Для работы с данными структуры необходимо создать экземпляр класса. Например:

```
class Student:
student1 = Student(«Иванов», 20, 4.5)
student2 = Student(«Петров», 19, 3.8)

info_student1 = student1.get_info()
info_student2 = student2.get_info()

print(info_student1)
print(info_student2)
```

Этот код создает описание двух студентов и выводит их информацию.

Обновление данных через методы.

Для обновления данных в структуре используются методы класса. Например, добавим метод «изменить_средний_балл»:

```
class Student:
    # ... (код из предыдущих примеров)

    def change_grade(self, new_grade):
        self.average_grade = new_grade
```

Использование:

```
student1.change_grade(4.8)
new_info_student1 = student1.get_info()
print(new_info_student1)
```

Этот код изменяет средний балл первого студента и выводит обновленную информацию.

Разработка класса и его методов.

Создайте файл с описанием класса в отдельном модуле. Например, student.py. Определите класс и его методы в этом файле. Методы класса будут реализовывать функциональность работы с данными.

Пример разработки класса:

```
# student.py

class Student:
    def __init__(self, name, age, average_grade):
        self.name = name
        self.age = age
        self.average_grade = average_grade

    def get_info(self):
        return f»Имя: {self.name}, Возраст: {self.age}, Средний балл:
{self.average_grade}»
```

Создание основного скрипта приложения.

Создайте основной скрипт, который будет взаимодействовать с пользователем через консоль. В этом скрипте можно создавать объекты классов, вызывать их методы и обрабатывать ввод пользователя.

Пример основного скрипта (main.py):

```
# main.py

from student import Student

# Создаем объекты класса Student
student1 = Student(«Иванов», 20, 4.5)
student2 = Student(«Петров», 19, 3.8)

# Выводим информацию о студентах
print(student1.get_info())
print(student2.get_info())
```

Запуск приложения и тестирование.

Запустите основной скрипт (main.py) с помощью интерпретатора Python. Увидите результаты выполнения программы в консоли.

```
$ python main.py
```

Приложение должно успешно создавать объекты класса, вызывать их методы и выводить результаты в консоль.

Содержание отчета

Отчет по лабораторной работе оформляется согласно ГОСТ 2.105–2019 на листах формата А4 и должен содержать:

1) титульный лист;

2) цель работы;

3) введение: описание предметной области, цель работы, краткое описание разрабатываемого приложения;

4) реализация структуры данных: описание созданных классов, их атрибутов и методов;

5) программирование функциональности: обзор разработанных методов классов для работы с данными структур;

6) разработка консольного интерфейса: описание созданных элементов пользовательского интерфейса и их функциональности;

7) вывод.

Отчет оформляется на персональном компьютере в текстовом редакторе.

Контрольные вопросы

1 Что такое классы и объекты в Python?

2 Какие основные операции можно выполнять с данными структуры класса?

3 Каким образом можно связать интерфейс пользователя с Python-кодом?

4 Как обеспечить проверку корректности вводимых данных пользователем в Python-приложении?

5 Какие принципы следует соблюдать при разработке интерфейса в Python-приложении?

2 Лабораторная работа № 2. Программирование обработки исключений в исключительных ситуациях

Цель работы: освоить принципы обработки исключений в Python для эффективного управления ошибками в программах.

2.1 Порядок выполнения работы

2.1.1 Изучить основные концепции обработки исключений в Python. Изучить синтаксис команд try, except, finally, что является ключевым аспектом данной лабораторной работы.

2.1.2 Научиться создавать исключения, которые соответствуют специфическим ситуациям в программе. Это может понадобиться для точной обработки определенных типов ошибок.

2.1.3 Рассмотреть различные виды исключений в Python, такие как ValueError, TypeError, FileNotFoundError и т. д., и ознакомиться с тем, как правильно реагировать на каждый из них.

2.1.4 Изучить блок finally и его роль в обеспечении завершающих действий независимо от того, произошло исключение или нет.

2.1.5 Разработать несколько простых программ, в которых потребуется обрабатывать различные типы исключений. В этих сценариях необходимо акцентировать внимание на правильности обработки ошибок.

Исходные данные для выполнения данной лабораторной работы студент получает от преподавателя.

2.2 Основные теоретические положения

Типы распространенных исключений.

В Python существует множество различных типов исключений. Ниже приведены некоторые из наиболее распространённых:

– ValueError: возникает, когда функция получает аргумент правильного типа, но с некорректным значением;

– TypeError: возникает, когда операция применяется к объекту несоответствующего типа;

– FileNotFoundError: возникает, когда производится попытка открытия несуществующего файла;

– ZeroDivisionError: возникает, когда производится попытка деления на ноль.

Пример обработки исключения ValueError:

```
try:
    num = int(input(«Введите число: «))
except ValueError as e:
    print(f»Ошибка: {e}»)
```

В Python ключевое слово *as* используется для присваивания имени определённой переменной или объекта. Это может быть полезно, когда необходимо использовать объект с другим именем для избежания конфликта имен.

Например, при обработке исключений ключевое слово *as* позволяет присвоить ошибку (объект исключения) определённой переменной для последующей работы с ней.

Пример:

```
try:
    num = int(input(«Введите число: «))
except ValueError as e:
    print(f»Произошла ошибка: {e}»)
```

В этом примере *as e* означает, что объект ошибки будет доступен под именем *e*. Это позволяет обращаться к информации об ошибке (например, текст сообщения об ошибке) с использованием этого имени (*e*).

Использование нескольких блоков except.

Можно использовать несколько блоков except, чтобы обрабатывать разные типы исключений по-разному.

Пример с использованием except для двух разных типов исключений:

```
try:
    num = int(input(«Введите число: «))
except ValueError as e:
    print(f»Ошибка: {e}»)
```

Использование блока else.

Блок else позволяет выполнить код, если в блоке try не возникло исключений.

Пример использования else:

```
try:
    num = int(input(«Введите число: «))
    result = 10 / num
except ValueError as e:
    print(f»Ошибка: {e}»)
except ZeroDivisionError as e:
    print(f»Деление на ноль: {e}»)
else:
    print(f»Результат: {result}»)
```

Использование блока finally.

Блок finally позволяет выполнить код независимо от того, произошло исключение или нет.

Пример использования finally:

```
try:
    file = open(«example.txt», «r»)
    content = file.read()
except FileNotFoundError as e:
    print(f»Файл не найден: {e}»)
finally:
    file.close()
```

Содержание отчета

Отчет по лабораторной работе оформляется согласно ГОСТ 2.105–2019 на листах формата А4 и должен содержать:

1) титульный лист;
2) цель работы;
3) введение: описание темы лабораторной работы, её актуальность и цели;
4) теоретический обзор: обзор основных понятий и методов работы с исключениями в Python;
5) разработка программ: подробное описание разработанных программ с примерами кода;
6) вывод.

Отчет оформляется на персональном компьютере в текстовом редакторе.

Контрольные вопросы

1 Что такое исключение в Python?
2 Какой синтаксис используется для обработки исключений?
3 В чем разница между блоками try, except и finally?
4 Как создать собственное исключение в Python?
5 Какие основные типы исключений в Python вам известны?

3 Лабораторная работа № 3. Программирование библиотеки классов приложения

Цель работы: освоить принципы создания и использования библиотеки классов в Python для эффективной организации и повторного использования кода.

3.1 Порядок выполнения работы

3.1.1 Изучить концепцию создания базовых классов. Создать несколько базовых классов с основными атрибутами и методами, соответствующими назначению библиотеки.

3.1.2 Научиться расширять базовые классы, добавляя новые атрибуты и методы. Это позволит адаптировать базовые классы под конкретные потребности при разработке приложения.

3.1.3 Разработать несколько подклассов, унаследованных от базовых классов. Это даст возможность создавать специализированные объекты с уникальной функциональностью.

3.1.4 Создать тестовый скрипт, в котором будут созданы объекты с использованием разработанных классов. Проверить их работу, удостоверившись, что атрибуты и методы возвращают ожидаемые результаты.

Задания для выполнения данной лабораторной работы студент получает от преподавателя.

3.2 Основные теоретические положения

В этом разделе лабораторной работы описывается концепция наследования и создания подклассов. Подклассы предоставляют возможность расширения функциональности базовых классов с целью создания специализированных объектов.

Наследование от базового класса.

Необходимо начать с создания подкласса, который унаследует атрибуты и методы базового класса. Это делается с использованием следующей синтаксической конструкции:

```
class BaseClass:
    def __init__(self, attribute):
        self.attribute = attribute

    def method(self):
        return f»Метод базового класса, атрибут: {self.attribute}»

class SubClass(BaseClass):
    def new_method(self):
        return «Новый метод подкласса»
```

Добавление новых атрибутов и методов.

Расширить функциональность подкласса можно, добавив новые атрибуты и методы. Это позволяет создавать объекты, обладающие дополнительными свойствами и действиями.

```
class SubClass(BaseClass):
    def __init__(self, attribute, new_attribute):
        super().__init__(attribute)
        self.new_attribute = new_attribute

    def new_method(self):
        return f»Новый метод подкласса, новый атрибут: {self.new_attribute}»
```

Переопределение методов базового класса.

Иногда требуется изменить поведение методов, унаследованных от базового класса. Это делается путем переопределения метода в подклассе.

```
class SubClass(BaseClass):
    def method(self):
        return «Переопределенный метод подкласса»
```

Создание нескольких уровней наследования.

Python позволяет создавать цепочки наследования с несколькими уровнями подклассов.

```
class BaseClass:
    pass

class SubClass(BaseClass):
    pass

class SubSubClass(SubClass):
    pass
```

Содержание отчета

Отчет по лабораторной работе оформляется согласно ГОСТ 2.105–2019 на листах формата А4 и должен содержать:

1) титульный лист;
2) цель работы;
3) описание и выполнение хода работы по пп. 3.1.1–3.1.4;
4) вывод.

Отчет оформляется на персональном компьютере в текстовом редакторе.

Контрольные вопросы

1 Что такое базовый класс в Python?
2 Каким образом можно расширить базовый класс?
3 Что представляет собой подкласс?
4 Как осуществляется наследование атрибутов и методов в Python?
5 Какие преимущества предоставляет создание библиотеки классов при разработке приложений?

4 Лабораторная работа № 4. Программирование динамических структур (коллекций)

Цель работы: овладеть навыками работы с динамическими структурами данных в Python, такими как списки, кортежи, множества и словари.

4.1 Порядок выполнения работы

4.1.1 Изучить базовые операции со списками, такие как добавление, удаление элементов, индексирование и срезы. Создать несколько примеров работы с этой динамической структурой данных.

4.1.2 Ознакомиться с особенностями кортежей и их применением в Python. Разработать несколько примеров, иллюстрирующих работу с кортежами.

4.1.3 Изучить особенности множеств и операций, доступных для работы с ними. Создать примеры, демонстрирующие применение множеств в Python.

4.1.4 Ознакомиться с принципами работы со словарями, включая добавление, удаление элементов и получение значений по ключу. Разработать несколько примеров работы с этой динамической структурой данных.

Задание для выполнения данной лабораторной работы студент получает от преподавателя.

4.2 Основные теоретические положения

В данном разделе рассматриваются основные динамические структуры данных в Python и варианты их использования.

Списки (Lists).

Списки представляют собой упорядоченные коллекции элементов, каждому из которых присваивается индекс. Они позволяют хранить объекты разных типов, включая другие списки. Важно отметить, что списки являются изменяемыми структурами данных, т. е. их можно изменять после создания.

Примеры операций со списками:

```
# Создание списка
my_list = [1, 2, 3, 'hello', True]

# Добавление элемента в конец списка
my_list.append(4)

# Изменение элемента по индексу
my_list[2] = 5

# Удаление элемента по значению
my_list.remove('hello')

# Получение подсписка (среза)
subset = my_list[1:3]

# Длина списка
length = len(my_list)
```

Кортежи (Tuples).

Кортежи похожи на списки, но они являются неизменяемыми. Это значит, что после создания кортежа его элементы нельзя изменить.

Примеры работы с кортежами:

```
# Создание кортежа
my_tuple = (1, 2, 3, 'hello', True)

# Доступ к элементам по индексу
element = my_tuple[2]

# Попытка изменить элемент (приведет к ошибке)
# my_tuple[0] = 4
```

Множества (Sets).

Множества представляют собой коллекции уникальных элементов без определенного порядка. Они поддерживают операции над множествами, такие как объединение, пересечение и разность.

Примеры работы с множествами:

```
# Создание множества
my_set = {1, 2, 3, 4, 5}

# Добавление элемента
my_set.add(6)

# Удаление элемента
my_set.remove(3)

# Проверка наличия элемента
contains = 4 in my_set

# Объединение множеств
union_set = my_set.union({7, 8, 9})

# Пересечение множеств
intersection_set = my_set.intersection({4, 5, 6})
```

Словари (Dictionaries).

Словари представляют собой коллекции пар «ключ – значение». Они позволяют быстро получать значение по ключу. Ключи должны быть уникальными.

Примеры работы со словарями:

```
# Создание словаря
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}

# Доступ к значению по ключу
name = my_dict['name']

# Изменение значения по ключу
my_dict['age'] = 31

# Добавление новой пары ключ-значение
my_dict['gender'] = 'male'

# Удаление элемента по ключу
del my_dict['city']
```

Содержание отчета

Отчет по лабораторной работе оформляется согласно ГОСТ 2.105–2019 на листах формата А4 и должен содержать:

1) титульный лист;
2) цель работы;
3) описание и выполнение хода работы по пп. 4.1.1–4.1.4;
4) вывод.

Отчет оформляется на персональном компьютере в текстовом редакторе.

Контрольные вопросы

1 Что представляют собой динамические структуры данных в Python?

2 Какие операции доступны при работе с динамическими структурами данных?

3 В чем различие между списками и кортежами?

4 Какие особенности характеризуют множества в Python?

5 Как создать и работать со словарем в Python?

5 Лабораторная работа № 5. Программирование обработки графических файлов

Цель работы: изучить основы программирования обработки графических файлов в Python с использованием библиотеки PIL (Pillow).

5.1 Порядок выполнения работы

5.1.1 Установить и импортировать библиотеки Pillow.

5.1.2 Открыть изображения с использованием функции Image.open( ).

5.1.3 Изучить основные операции с изображениями: изменение размера, поворот, зеркальное отражение и другие манипуляции.

5.1.4 Применить фильтры и эффекты к изображениям с использованием встроенных возможностей библиотеки Pillow.

Исходные данные для выполнения данной лабораторной работы студент получает от преподавателя.

5.2 Основные теоретические положения

Для начала работы с Pillow, необходимо установить библиотеку при помощи pip:

```
python pip install pillow
```

После установки импортируем модуль Image:

```
from PIL import Image
```

Библиотека Pillow (Python Imaging Library, PIL) представляет собой мощный инструмент для работы с изображениями в Python. Она предоставляет широкий набор функций для открытия, редактирования и сохранения различных форматов графических файлов. Рассмотрим некоторые ключевые аспекты и возможности библиотеки Pillow.

Открытие и сохранение изображений.

Основной класс библиотеки – Image. Для открытия изображения используется метод open( ). Сохранение изображения осуществляется с помощью метода save( ).

Пример открытия и сохранения изображения:

```
from PIL import Image

# Открытие изображения
img = Image.open('example.jpg')

# Сохранение изображения
img.save('output.jpg')
length = len(my_list)
```

Простейшие операции с изображениями.

Библиотека Pillow предоставляет широкий набор методов для простейших манипуляций с изображениями. К ним относятся изменение размера, поворот, зеркальное отражение и многие другие.

Пример изменения размера и поворота изображения:

```
# Изменение размера
resized_img = img.resize((800, 600))

# Поворот на 90 градусов
rotated_img = img.rotate(90)
```

Фильтры и эффекты.

Библиотека предоставляет возможность применения различных фильтров и эффектов к изображениям. Например, можно преобразовать изображение в ч/б, улучшить контрастность, применить размытие и другие эффекты.

Пример применения фильтров:

```
# Преобразование в ч/б
bw_img = img.convert('L')

# Увеличение контрастности
from PIL import ImageEnhance

enhancer = ImageEnhance.Contrast(img)
high_contrast_img = enhancer.enhance(2.0)
```

Содержание отчета

Отчет по лабораторной работе оформляется согласно ГОСТ 2.105–2019 на листах формата А4 и должен содержать:

1) титульный лист;
2) цель работы;
3) описание и выполнение хода работы по пп. 5.1.1–5.1.4;
4) вывод.

Отчет оформляется на персональном компьютере в текстовом редакторе.

Контрольные вопросы

1 Какие возможности предоставляет библиотека Pillow для работы с графическими файлами?

2 Как открыть изображение с использованием Pillow?

3 Какие операции можно выполнять с изображением с помощью Pillow?

4 Какие фильтры и эффекты можно применять к изображению с использованием Pillow?

6 Лабораторная работа № 6. Программирование обработки файлов текстовых форматов

Цель работы: изучить основы программирования для обработки файлов в текстовых форматах с использованием Python.

6.1 Порядок выполнения работы

6.1.1 Научиться открывать, читать и записывать текстовые файлы с использованием встроенных средств Python.

6.1.2 Разработать программу для обработки данных, считанных из текстового файла, включая сортировку, фильтрацию, анализ и другие манипуляции.

6.1.3 Изучить работу с различными форматами данных в текстовых файлах, такими как CSV, JSON, XML. Научиться считывать и записывать данные в этих форматах.

Задания для выполнения данной лабораторной работы студент получает от преподавателя.

6.2 Основные теоретические положения

Работа с текстовыми файлами в Python предполагает не только открытие и чтение данных, но и обработку этих данных в соответствии с поставленной задачей.

Открытие и сохранение изображений.

Для начала работы с файлами используется функция open(). Она принимает два аргумента: имя файла и режим. Режим 'r' означает чтение, 'w' – запись, 'a' – добавление. После завершения работы с файлом его следует закрыть с помощью метода close().

Пример чтения и записи файла:

```
# Чтение файла
with open('example.txt', 'r') as file:
    content = file.read()

# Запись в файл
with open('output.txt', 'w') as file:
    file.write('Hello, World!')
```

Простейшие операции с изображениями.

Считанные из файла данные могут быть обработаны различными способами. Это может включать в себя сортировку, фильтрацию, анализ и другие манипуляции в зависимости от конкретной задачи.

Пример сортировки чисел из файла:

```
with open('numbers.txt', 'r') as file:
    numbers = [int(line) for line in file]

sorted_numbers = sorted(numbers)
```

Фильтры и эффекты.

В Python существует несколько популярных форматов данных, которые используются для хранения и обмена информацией. Рассмотрим основные.

CSV (Comma Separated Values) – этот формат предназначен для представления табличных данных в текстовом виде. Модуль csv предоставляет функционал для работы с CSV-файлами.

Пример чтения данных из CSV-файла:

```
import csv

with open('data.csv', 'r') as file:
    reader = csv.reader(file)
    data = list(reader)
```

JSON (JavaScript Object Notation) – представляет собой удобный формат для передачи данных между приложениями. В Python есть модуль json для работы с этим форматом.

Пример чтения и записи JSON-файла:

```
import json

# Чтение JSON файла
with open('data.json', 'r') as file:
    data = json.load(file)

# Запись в JSON файл
with open('output.json', 'w') as file:
    json.dump(data, file)
```

XML (Extensible Markup Language) – используется для представления и обмена структурированными данными. В Python есть модуль xml.etree.ElementTree для работы с XML-файлами.

Пример чтения данных из XML-файла:

```
import xml.etree.ElementTree as ET

tree = ET.parse('data.xml')
root = tree.getroot()

for child in root:
    print(child.tag, child.attrib)
```

Содержание отчета

Отчет по лабораторной работе оформляется согласно ГОСТ 2.105–2019 на листах формата А4 и должен содержать:

1) титульный лист;
2) цель работы;
3) описание и выполнение хода работы по пп. 6.1.1–6.1.3;
4) вывод.

Отчет оформляется на персональном компьютере в текстовом редакторе.

Контрольные вопросы

1 Как открыть файл для чтения в Python? И для записи?
2Как читать данные из файла?
3 Как записать данные в файл?
4 Какой модуль используется для работы с CSV-файлами?
5 Какой формат данных представляет JSON?
6 Какой модуль используется для работы с XML-файлами?

7 Лабораторная работа № 7. Программирование асинхронного обмена данными

Цель работы: изучить основы программирования асинхронных операций для эффективного обмена данными в Python.

7.1 Порядок выполнения работы

7.1.1 Изучить принципы асинхронного программирования и понятия, связанные с ним (корутины, событийный цикл, await/async).

7.1.2 Разработать асинхронную программу, использующую асинхронные функции и корутины для обмена данными между несколькими компонентами.

7.1.3 Реализовать асинхронный обмен данными с использованием сетевого соединения (например, сокеты) или симуляции асинхронного ввода/вывода.

7.1.4 Провести тестирование асинхронного обмена данными и убедиться в корректности работы программы.

7.1.5 Сравнить эффективность асинхронной программы с синхронной реализацией той же задачи.

7.1.6 Проанализировать преимущества и недостатки асинхронного программирования в данной задаче.

Задания для выполнения данной лабораторной работы студент получает от преподавателя.

7.2 Основные теоретические положения

Асинхронное программирование в Python является методом организации работы программы таким образом, чтобы она не блокировала свое выполнение в ожидании завершения некоторых операций. Вместо этого программа может продолжать выполнять другие задачи, в то время как определённые операции выполняются в фоновом режиме.

Данный подход особенно полезен в ситуациях, когда программе необходимо работать с операциями ввода/вывода, которые могут занять значительное время (например, чтение большого файла, отправка данных по сети и т. д.). Благодаря асинхронному программированию вместо того, чтобы ожидать завершения этих операций, программа может продолжать выполнять другие задачи или операции.

Применение асинхронного программирования особенно актуально в приложениях, где требуется обрабатывать большое количество параллельных операций. Это может включать в себя веб-сервера, сетевые приложения, ботов для социальных сетей, обработку больших данных и многие другие сценарии.

Для реализации асинхронного программирования в Python используется модуль asyncio. Этот модуль предоставляет набор инструментов для создания асинхронных функций и эффективного управления выполнением различных задач. Он позволяет программистам асинхронно ожидать завершения определенных операций, в то время как другие задачи могут продолжать свою работу.

В итоге асинхронное программирование позволяет сделать программу более эффективной и отзывчивой, особенно в ситуациях с высокой нагрузкой на операции ввода/вывода или параллельной обработки множества задач.

Функции из библиотеки asyncio, используемые в лабораторной работе.

1 async def: определение асинхронной функции. Этот ключевой момент позволяет функции работать асинхронно, т. е. не блокировать выполнение программы в ожидании завершения операций.

```
async def my_async_function():
    # Код асинхронной функции
```

2 await: ожидание завершения асинхронной операции. Это ключевое слово позволяет приостановить выполнение текущей асинхронной функции до завершения указанной асинхронной операции.

```
async def my_async_function():
    result = await some_async_operation()
```

3 asyncio.get_event_loop(): получение цикла событий. Цикл событий управляет асинхронными операциями в программе.

```
loop = asyncio.get_event_loop()
```

4 loop.run_until_complete(): запуск асинхронной задачи. Эта функция запускает асинхронную задачу и блокирует выполнение программы до её завершения.

```
loop.run_until_complete(my_async_function())
```

5 loop.close(): закрытие цикла событий. После завершения работы с асинхронными задачами цикл событий должен быть закрыт.

```
loop.close()
```

Эти функции позволяют создавать и управлять асинхронными задачами в Python с использованием библиотеки asyncio. Они позволяют эффективно работать с асинхронными операциями ввода/вывода и другими параллельными задачами, что особенно важно в сценариях с высокой нагрузкой на операции ввода/вывода или параллельной обработки множества задач.

Пример 1 – Загрузка файлов из сети.

Допустим, есть список URL-адресов файлов, которые необходимо скачать. Используя асинхронное программирование, можно одновременно начать загрузку всех файлов, вместо того чтобы ждать, пока каждый файл будет загружен поочередно.

```
import asyncio
import aiohttp

async def download_file(url):
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as response:
            content = await response.read()
            with open(f'file_{url[-5:]}.txt', 'wb') as file:
                file.write(content)

async def main():
    urls = [
        'https://example.com/file1.txt',
        'https://example.com/file2.txt',
        'https://example.com/file3.txt'
    ]
    await asyncio.gather(*[download_file(url) for url in urls])

asyncio.run(main())
```

В этом примере используется библиотека aiohttp для асинхронной загрузки файлов из сети. Создается асинхронная функция download_file, которая принимает URL и скачивает файл. Функция main запускает все задачи параллельно с помощью asyncio.gather.

Пример 2 – Параллельные вычисления.

Предположим, есть несколько задач, которые можно выполнить параллельно, например обработка данных из нескольких источников. Используя асинхронные функции, можно запустить все эти задачи параллельно и дождаться их завершения.

```
import asyncio
async def task1():
    await asyncio.sleep(2)
    return «Результат задачи 1»
async def task2():
    await asyncio.sleep(3)
    return «Результат задачи 2»
async def task3():
    await asyncio.sleep(1)
    return «Результат задачи 3»
async def main():
    results = await asyncio.gather(task1(), task2(), task3())
    print(results)
asyncio.run(main())
```

В этом примере имеется три задачи (task1, task2, task3), которые могут быть выполнены параллельно. Функция main использует asyncio.gather, чтобы запустить все задачи одновременно. Как только все задачи завершатся, получим результаты.

Пример 3 – Параллельные запросы к API.

В этом примере используется библиотека aiohttp для выполнения параллельных запросов к API. Есть список URL-адресов, по которым необходимо получить данные. Создаем асинхронную функцию fetch_data, которая отправляет запрос к каждому URL и возвращает результат.

```
import asyncio
import aiohttp

async def fetch_data(url):
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as response:
            data = await response.json()
            return data

async def main():
    urls = [
        'https://api.example.com/data1',
        'https://api.example.com/data2',
        'https://api.example.com/data3'
    ]
    results = await asyncio.gather(*[fetch_data(url) for url in urls])
    print(results)

asyncio.run(main())
```

В этом примере нужно получить данные с нескольких URL-адресов API. Создаётся асинхронная функция fetch_data, которая использует библиотеку aiohttp для отправки асинхронных HTTP-запросов. Затем в функции main используется asyncio.gather, чтобы запустить все запросы параллельно и дождаться их завершения.

Такой подход особенно полезен, когда требуется получить данные из нескольких источников (например, различные эндпоинты API) и обработать их параллельно. Это позволяет сократить общее время ожидания, что может существенно улучшить производительность программы.

Асинхронное программирование помогает ускорить выполнение программы в ситуациях, когда есть задачи, которые можно выполнять параллельно. Это особенно важно при работе с сетью, базами данных и другими операциями ввода/вывода, которые могут занимать много времени.

Содержание отчета

Отчет по лабораторной работе оформляется согласно ГОСТ 2.105–2019 на листах формата А4 и должен содержать:

1) титульный лист;
2) цель работы;
3) описание и выполнение хода работы по пп. 7.1.1–7.1.6;
4) вывод.

Отчет оформляется на персональном компьютере в текстовом редакторе.

Контрольные вопросы

1 Что такое асинхронное программирование?
2 Как создать асинхронную функцию в Python?
3 Как ожидать асинхронную операцию внутри асинхронной функции?
4 Как запустить асинхронную задачу?
5 Какой модуль используется для асинхронного программирования в Python?

8 Лабораторная работа № 8. Использование переопределения операторов в приложениях

Цель работы: ознакомиться с принципами переопределения операторов в Python и их применением в приложениях.

8.1 Порядок выполнения работы

8.1.1 Разработать программу, использующую переопределение переменных.
8.1.2 Реализовать асинхронный обмен данными с использованием сетевого соединения (например, сокеты) или симуляции асинхронного ввода/вывода.
8.1.3 Провести тестирование разработанной программы и убедиться в корректности её работы.

Задания для выполнения данной лабораторной работы студент получает от преподавателя.

8.2 Основные теоретические положения

Переопределение операторов в Python предоставляет возможность пользовательским классам изменять стандартное поведение операторов так, чтобы оно соответствовало специфике данных класса. Это позволяет делать работу с объектами данного класса более естественной и интуитивно понятной.

1 Базовое переопределение операторов:

- __add__ (сложение);
- __sub__ (вычитание);
- __mul__ (умножение);
- __truediv__ (деление).

2 Расширенное переопределение операторов:

- __eq__ (равенство);
- __ne__ (неравенство);
- __lt__ (меньше чем);
- __le__ (меньше или равно);
- __gt__ (больше чем);
- __ge__ (больше или равно);
- __iadd__ (расширенное сложение);
- __isub__ (расширенное вычитание).
- и другие.

Вот некоторые из наиболее используемых методов переопределения.

1 __init__(self, ...): метод инициализации объекта. Вызывается при создании нового объекта класса. Здесь определяются начальные значения атрибутов.

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y
vec = Vector(3, 4)
```

2 __str__(self): метод, возвращающий строковое представление объекта. Вызывается функцией str( ) или при передаче объекта в print( ).

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f'({self.x}, {self.y})'

vec = Vector(3, 4)
print(vec)  # Выведет: (3, 4)
```

3 __add__(self, other): метод для операции сложения (+). Определяет поведение при использовании оператора +.

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)

vec1 = Vector(3, 4)
vec2 = Vector(1, 2)
result = vec1 + vec2
print(result)  # Выведет: (4, 6)
```

4 __eq__(self, other): метод для операции сравнения на равенство (==). Определяет поведение при использовании оператора ==.

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

vec1 = Vector(3, 4)
vec2 = Vector(3, 4)
print(vec1 == vec2)  # Выведет: True
```

5 __lt__(self, other): метод для операции сравнения «меньше чем» (<). Определяет поведение при использовании оператора <.

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __lt__(self, other):
        return self.x < other.x and self.y < other.y

vec1 = Vector(3, 4)
vec2 = Vector(5, 6)
print(vec1 < vec2)  # Выведет: True
```

6 __len__(self): метод, возвращающий длину объекта. Вызывается функцией len( ).

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __len__(self):
        return 2

vec = Vector(3, 4)
print(len(vec))  # Выведет: 2
```

Эти методы представляют лишь малую часть возможностей по переопределению операторов в Python. Однако они обеспечивают основу для создания более сложного и интуитивно понятного поведения объектов пользовательских классов.

Содержание отчета

Отчет по лабораторной работе оформляется согласно ГОСТ 2.105–2019 на листах формата А4 и должен содержать:

1) титульный лист;
2) цель работы;
3) описание и выполнение хода работы по пп. 8.1.1–8.1.3;
4) вывод.

Отчёт оформляется на персональном компьютере в текстовом редакторе.

Контрольные вопросы

1 Что такое переопределение операторов в Python?

2 Какой метод класса используется для переопределения оператора сложения?

3 Какой метод класса используется для представления объекта в виде строки?

4 Какой метод класса позволяет сравнивать объекты на равенство?

5 Для чего используется метод __init__ в контексте переопределения операторов?

6 Как переопределить оператор сравнения «больше чем» (>) в пользовательском классе?

7 В каких ситуациях полезно использовать переопределение операторов?

8 Какие операторы можно переопределить в пользовательских классах?

9 Лабораторная работа № 9. Программирование данных времени и даты

Цель работы: освоить работу с модулем datetime и научиться выполнять различные операции с данными времени и даты в Python.

9.1 Порядок выполнения работы

9.1.1 Научиться создавать объекты datetime, timedelta для представления текущей даты и времени.

9.1.2 Выполнить арифметические операции с временными интервалами (сложение, вычитание).

9.1.3 Выполнить парсинг строки в объект datetime с заданным форматом.

Задания для выполнения данной лабораторной работы студент получает от преподавателя.

9.2 Основные теоретические положения

Модуль datetime в Python предоставляет функционал для работы с датами и временем. В нем содержатся классы datetime, date, time и timedelta. Объекты класса datetime представляют собой комбинацию даты и времени, в то время как объекты класса date представляют только дату, а объекты класса time – только время.

Пример создания объекта datetime:

```
1: from datetime import datetime
2: now = datetime.now()
```

Для получения отдельных компонентов (год, месяц, день, час, минута, секунда) используются атрибуты: year, month, day, hour, minute, second.

Работа с временными интервалами.

Класс timedelta представляет собой временной интервал. Он может использоваться для вычисления разницы между двумя датами или для прибавления/вычитания времени к дате.

Пример создания объекта timedelta:

```
1: from datetime import timedelta
2: delta = timedelta(days=5, hours=3)
```

Операции с timedelta позволяют складывать, вычитать и умножать его на число.

Форматирование и парсинг дат.

Для преобразования объекта datetime в строку с определенным форматом используется метод strftime(format). Для обратной операции – преобразования строки в объект datetime – используется метод strptime(date_string, format).

Пример форматирования и парсинга:

```
formatted_date = now.strftime('%Y-%m-%d %H:%M:%S')
parsed_date = datetime.strptime('2023-09-26 14:30:00', '%Y-%m-%d %H:%M:%S')
```

Работа с часовыми поясами.

Модуль datetime позволяет работать с часовыми поясами. Для этого используется класс timezone. Метод astimezone(timezone) позволяет преобразовать дату и время из одного часового пояса в другой.

Пример работы с часовыми поясами:

```
from datetime import timezone, timedelta
new_timezone = timezone(timedelta(hours=-5))  # UTC-5
new_time = now.astimezone(new_timezone)
```

Работа с календарем.

Модуль datetime также предоставляет функционал для работы с календарем. Например, метод weekday() возвращает номер дня недели (0 – понедельник, 6 – воскресенье), а метод isoweekday() возвращает номер дня недели (1 – понедельник, 7 – воскресенье).

Пример использования:

```
day_of_week = now.weekday()
isoweekday = now.isoweekday()
```

Эти основные теоретические положения помогут студентам начать работу с модулем datetime и освоить основные операции с датами и временем в Python.

Содержание отчета

Отчет по лабораторной работе оформляется согласно ГОСТ 2.105–2019 на листах формата А4 и должен содержать:

1) титульный лист;
2) цель работы;
3) описание и выполнение хода работы по пп. 9.1.1–9.1.3;
4) вывод.

Отчет оформляется на персональном компьютере в текстовом редакторе.

Контрольные вопросы

1 Какие операторы можно переопределить в пользовательских классах? Чем отличаются объекты datetime и date?

2 Как получить текущую дату и время с использованием модуля datetime?

3 Как выделить год, месяц и день из объекта datetime?

4 В чем разница между timedelta и datetime?

5 Как преобразовать строку в объект datetime?

6 Как вычислить следующий понедельник?

7 Какое значение будет иметь объект datetime после операции + timedelta(days=5)?

8 Как узнать, является ли год високосным с использованием модуля datetime?

10 Лабораторная работа № 10. Программирование многопоточных приложений

Цель работы: ознакомиться с принципами многопоточного программирования в Python с использованием модуля threading. Изучить основные компоненты многопоточности и научиться создавать эффективные и отзывчивые многопоточные приложения.

10.1 Порядок выполнения работы

10.1.1 Научиться создавать потоки, работать с блокировками, очередями, семафорами, замками с использованием встроенных средств Python.

10.1.2 Разработать программу для обработки данных с использованием многопоточности. Также аналогичную программу разработать классическими средствами.

10.1.3 Сравнить две программы, проанализировать преимущества и недостатки.

Задания для выполнения данной лабораторной работы студент получает от преподавателя.

10.2 Основные теоретические положения

Многопоточное программирование в Python позволяет выполнять несколько задач параллельно, что повышает эффективность использования процессора и позволяет улучшить отзывчивость программы.

В Python для работы с многопоточностью используется модуль threading. Он предоставляет классы и функции для создания и управления потоками.

Основные компоненты многопоточного программирования.

Потоки (Threads).

Поток представляет собой единицу исполнения внутри процесса. Он позволяет выполнить некоторый блок кода независимо от остальных потоков. В Python потоки создаются с использованием класса Thread.

Пример создания потока:

```
from threading import Thread

def print_numbers():
    for i in range(5):
        print(i)

thread = Thread(target=print_numbers)
thread.start()
```

В этом примере импортируется модуль threading, создается функцию print_numbers, которая выводит числа от 0 до 4. Затем создается объект потока thread и запускается с помощью метода start(). В результате на экране поочередно появляются числа.

Блокировки (Locks).

Блокировка предназначена для синхронизации доступа к общим данным из разных потоков. Она предотвращает конфликты при попытках одновременного доступа к ресурсу.

Пример использования блокировки:

```
from threading import Lock

lock = Lock()

def increment_counter():
    global counter
    lock.acquire()
    counter += 1
    lock.release()
```

Здесь создается объект блокировки lock с помощью Lock(). Функция increment_counter увеличивает глобальную переменную counter на 1. Перед началом операции инкрементации захватывается блокировка с помощью lock.acquire(), а после завершения операции она освобождается с помощью lock.release(). Это предотвращает гонки данных при одновременном доступе к counter из разных потоков.

Очереди (Queues).

Очередь позволяет безопасно обмениваться данными между потоками. Она предоставляет методы для добавления и извлечения элементов, обеспечивая при этом атомарность операций.

Пример использования очереди:

```
from queue import Queue

def producer(queue):
    for i in range(5):
        queue.put(i)

def consumer(queue):
    while True:
        item = queue.get()
        if item is None:
            break
        print(item)
```

В этом примере используется модуль queue. Создается объект очереди queue с помощью Queue(). Функция producer добавляет числа от 0 до 4 в очередь с помощью queue.put(i), а функция consumer извлекает и выводит эти числа с помощью item = queue.get().

Семафоры (Semaphores).

Семафор используется для управления доступом к общему ресурсу. Он позволяет контролировать количество потоков, имеющих одновременный доступ.

Пример использования семафора:

```
from threading import Semaphore

semaphore = Semaphore(value=3)  # Максимум 3 потока могут захватить семафор

def access_shared_resource():
    with semaphore:
        # Критическая секция
        pass
```

Здесь создается семафор semaphore с максимальным значением 3. Функция access_shared_resource представляет критическую секцию. Перед входом в критическую секцию захватывается семафор (with semaphore:), а после выхода из него он освобождается. Таким образом, не более трех потоков могут одновременно находиться в критической секции.

Замки (Event).

Замки используются для синхронизации между потоками, когда один поток должен дождаться события, произошедшего в другом потоке.

Пример использования замка:

```
from threading import Event

event = Event()

def wait_for_event():
    print(«Waiting for event»)
    event.wait()
    print(«Event received»)

def set_event():
    print(«Event is set»)
    event.set()
```

В этом примере создается объект события event с помощью Event(). Функция wait_for_event ждет, пока событие не будет установлено с помощью event.wait(), а затем выводит сообщение о получении события. Функция set_event устанавливает событие с помощью event.set().

Работа с многопоточностью требует аккуратности и внимательности из-за потенциальных проблем с синхронизацией и гонками данных. Всегда следует правильно организовывать доступ к общим ресурсам из разных потоков.

Содержание отчета

Отчет по лабораторной работе оформляется согласно ГОСТ 2.105–2019 на листах формата А4 и должен содержать:

1) титульный лист;
2) цель работы;
3) описание и выполнение хода работы по пп. 10.1.1–10.1.3;
4) вывод.

Отчет оформляется на персональном компьютере в текстовом редакторе.

Контрольные вопросы

1 Что такое поток в программировании?

2 Как создать поток в Python с использованием модуля threading?

3 Какие методы блокировки (Lock) используются для захвата и освобождения?

4 В чем отличие между блокировкой и семафором?

5 Как работают методы put() и get() для работы с очередью (Queue)?

6 Зачем нужны замки (Event)? Как они используются?

7 Какие основные принципы синхронизации потоков?

8 Как избежать гонок данных при работе с общими ресурсами в многопоточном приложении?

11 Лабораторная работа № 11. Использование рефлексии при проектировании приложений

Цель работы: ознакомиться с принципами использования рефлексии в Python для анализа и модификации программы во время её выполнения. Изучить основные методы и приемы работы с рефлексией и применять их при проектировании приложений.

11.1 Порядок выполнения работы

11.1.1 Научиться открывать, читать и записывать текстовые файлы с использованием встроенных средств Python.

11.1.2 Разработать программу для обработки данных, считанных из текстового файла, включая в себя сортировку, фильтрацию, анализ и другие манипуляции.

11.1.3 Изучить работу с различными форматами данных в текстовых файлах, такими как CSV, JSON, XML. Научиться считывать и записывать данные в этих форматах.

Задания для выполнения данной лабораторной работы студент получает от преподавателя.

11.2 Основные теоретические положения

Рефлексия представляет собой способность программы к анализу своей структуры и поведения во время выполнения. В Python рефлексия позволяет получать информацию о классах, объектах, атрибутах и методах, а также динамически изменять их.

В Python тип объекта можно получить с помощью встроенной функции type(). Пример:

```
1: class MyClass:
2:     pass
3:
4: obj = MyClass()
5: print(type(obj))  # Выведет <class '__main__.MyClass'>
```

С использованием функции type() можно динамически создавать классы. Пример:

```
MyDynamicClass = type('MyDynamicClass', (), {'attribute': 42})
obj = MyDynamicClass()
print(obj.attribute)  # Выведет 42
```

Через рефлексию можно получить доступ к атрибутам и методам объекта. Пример:

```
class MyClass:
    def __init__(self):
        self.attribute = 42

    def method(self):
        print(«Hello!»)

obj = MyClass()
print(obj.attribute)  # Выведет 42
obj.method()  # Выведет «Hello!»
```

Аннотации и документационные строки доступны через атрибуты __annotations__ и __doc__. Пример:

```
def my_function(arg1: int, arg2: str) -> float:
    ««« 
    This is the docstring of my_function.
    ««« 
    return 3.14

print(my_function.__annotations__)  # Выведет {'arg1': <class 'int'>, 'arg2': <class 'str'>, 'return': <class 'float'>}
print(my_function.__doc__)  # Выведет «This is the docstring of my_function.»
```

Примеры применения рефлексии

Автоматическое создание объектов на основе данных.

Рефлексия позволяет создавать объекты и загружать данные в них динамически на основе внешних источников, таких как файлы или базы данных.

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

data = {'name': 'Alice', 'age': 30}
user = User(**data)
print(user.name, user.age)  # Выведет «Alice 30»
```

В этом примере создаётся класс User с атрибутами name и age. С использованием рефлексии, можно динамически создать объект этого класса и заполнить его данными из словаря data. Это особенно полезно, когда есть внешние источники данных, такие как файлы или базы данных, и нужно создать объекты на основе этих данных.

Интроспекция объектов.

Рефлексия позволяет получать информацию о классах и объектах во время выполнения программы, что может быть полезно для отладки или создания универсальных функций.

```
class MyClass:
    def method(self):
        pass

obj = MyClass()
print(hasattr(obj, 'method'))  # Выведет «True»
print(callable(getattr(obj, 'method')))  # Выведет «True»
```

В этом примере создан класс MyClass с методом method. С использованием рефлексии, проверяется, есть ли у объекта obj атрибут method с помощью функции hasattr(). Затем проверяется, можно ли вызвать атрибут method с помощью функции callable().

Эта возможность позволяет динамически анализировать объекты во время выполнения программы. Это может быть полезно для отладки или для создания универсальных функций, которые работают с разными типами объектов.

Динамическое расширение функционала.

Рефлексия позволяет добавлять или заменять атрибуты и методы объектов во время выполнения программы.

```
class MyClass:
    def original_method(self):
        print(«Original method»)

def new_method():
    print(«New method»)

obj = MyClass()
setattr(obj, 'new_method', new_method)
obj.new_method()  # Выведет «New method»
```

В этом примере есть класс MyClass с методом original_method. С использованием рефлексии, добавив новый метод new_method к объекту obj, можно вызвать new_method как обычный метод объекта.

Это демонстрирует, как рефлексия позволяет нам динамически изменять функционал объектов во время выполнения программы. Это может быть полезно, если нужно добавить или заменить атрибуты и методы объектов в зависимости от текущих условий.

Содержание отчета

Отчет по лабораторной работе оформляется согласно ГОСТ 2.105–2019 на листах формата А4 и должен содержать:

1) титульный лист;

2) цель работы;

3) описание и выполнение хода работы по пп. 11.1.1–11.1.3;

4) вывод.

Отчет оформляется на персональном компьютере в текстовом редакторе.

Контрольные вопросы

1 Что такое рефлексия в программировании?

2 В чем основное применение рефлексии?

3 Как получить информацию о типе объекта в Python?

4 Как можно динамически создавать классы?

5 Каким образом можно получить список атрибутов и методов объекта через рефлексию?

6 Как осуществляется вызов методов динамически?

7 Как можно получить информацию о типах аргументов и возвращаемых значений функций через рефлексию?

12 Лабораторная работа № 12. Использование в приложениях иных языков программирования

Цель работы: ознакомиться с механизмами интеграции различных языков программирования в приложения на Python.

12.1 Порядок выполнения работы

12.1.1 Вспомогательные языки программирования (например, C, C++, Java) предоставляют интерфейсы для интеграции с Python. Используя один из таких языков, необходимо создать программу, которая выполняет вычисления и возвращает результат Python-скрипту.

12.1.2 Написать Python-скрипт, который вызывает функции, реализованные на выбранном вспомогательном языке. Обработать результат вычислений в Python-скрипте.

12.1.3 Изучить работу с различными форматами данных в текстовых файлах, такими как CSV, JSON, XML. Научиться считывать и записывать данные в этих форматах.

Задания для выполнения данной лабораторной работы студент получает от преподавателя.

12.2 Основные теоретические положения

Интеграция различных языков программирования в приложения на Python предоставляет возможность использовать преимущества каждого языка для решения конкретных задач. Это особенно важно, когда требуется объединить высокоуровневые возможности Python с низкоуровневыми операциями или библиотеками, реализованными на других языках. В данной лабораторной

работе рассматриваются механизмы интеграции с вспомогательными языками программирования, такими как C, C++, Java и др.

Механизмы интеграции.

1 Использование ctypes.

Модуль ctypes в Python предоставляет механизм для вызова функций и работы с библиотеками, написанными на C. Этот подход позволяет взаимодействовать с кодом, реализованным на низкоуровневом языке, непосредственно из Python.

2 Использование Cython.

Cython – это компилятор для Python, который позволяет писать расширения на Python с синтаксическими элементами C. Это позволяет ускорить выполнение кода, а также вызывать функции из существующих C-библиотек.

3 Использование JNI (Java Native Interface).

Если требуется взаимодействовать с Java-кодом, можно использовать Java Native Interface, позволяющий вызывать функции на Java из кода на C/C++.

Преимущества интеграции различных языков.

1 Увеличение производительности: возможность использовать низкоуровневые языки программирования позволяет увеличить производительность в критических участках кода.

2 Использование существующих библиотек и ресурсов: интеграция с другими языками позволяет использовать уже существующие библиотеки и решения, написанные на этих языках.

3 Доступ к аппаратным ресурсам: некоторые задачи, такие как работа с системными вызовами, могут требовать использования низкоуровневых возможностей, которые могут быть легче реализовать на C/C++.

Применение.

Вычислительные приложения: использование C/C++ для вычислительных задач может значительно ускорить выполнение программы.

Работа с библиотеками, написанными на других языках: взаимодействие с библиотеками, доступными только на определённом языке программирования, может потребовать интеграции.

Работа с аппаратурой: для взаимодействия с аппаратными устройствами, драйверами и системными вызовами часто используется *C/C++*.

Интеграция различных языков программирования позволяет создавать мощные и эффективные приложения, используя преимущества каждого языка в соответствующих областях. Однако следует помнить, что такой подход требует внимательного контроля и тестирования для обеспечения правильной работы взаимодействия между компонентами на разных языках.

Пример – Использование ctypes для работы с библиотекой C.

1 Создание библиотеки на *C* (код на языке python).

```c
// Пример библиотеки на C (math_operations.c)
#include <stdio.h>

int add(int a, int b) {
    return a + b;
}

int subtract(int a, int b) {
    return a - b;
}

int multiply(int a, int b) {
    return a * b;
}

int divide(int a, int b) {
    if (b != 0) {
        return a / b;
    } else {
        printf(«Error: Division by zero!\n»);
        return -1;
    }
}
```

2 Компиляция библиотеки на C в команду строке.

```
gcc -shared -o math_operations.so -fPIC math_operations.c
```

3 Использование в Python с ctypes.

```python
# Импорт ctypes
import ctypes

# Загрузка библиотеки
math_lib = ctypes.CDLL('./math_operations.so')

# Вызов функций
result_add = math_lib.add(3, 5)
result_subtract = math_lib.subtract(8, 2)
result_multiply = math_lib.multiply(4, 7)
result_divide = math_lib.divide(10, 2)

print(f»Addition: {result_add}»)
print(f»Subtraction: {result_subtract}»)
print(f»Multiplication: {result_multiply}»)
print(f»Division: {result_divide}»)
```

В данном примере создается библиотека на C, которая содержит четыре математические функции: add, subtract, multiply, divide.

С использованием ctypes библиотека загружается в Python, и функции из нее вызываются.

Содержание отчета

Отчет по лабораторной работе оформляется согласно ГОСТ 2.105–2019 на листах формата А4 и должен содержать:

1) титульный лист;
2) цель работы;
3) описание и выполнение хода работы по пп. 12.1.1–12.1.3;
4) вывод.

Отчет оформляется на персональном компьютере в текстовом редакторе.

Контрольные вопросы

1 В чем заключаются преимущества использования различных языков программирования в одном приложении?

2 Какие механизмы интеграции могут быть использованы для взаимодействия между Python и другими языками программирования?

3 Как можно передавать данные между Python и вспомогательным языком программирования?

4 Как обрабатывать результаты выполнения программы на вспомогательном языке в Python-скрипте?

5 В каких сценариях может быть особенно полезной интеграция Python с другими языками программирования?

Список литературы

1 **Агальцов, В. П.** Базы данных: учебник: в 2 кн. Кн. 1: Локальные базы данных / В. П. Агальцов. – Москва: ФОРУМ; ИНФРА-М, 2021. – 352 с.: ил.

2 **Агальцов, В. П.** Базы данных: в 2 кн. Кн 2: Распределенные и удаленные базы данных: учебник / В. П. Агальцов. – Москва: ФОРУМ; ИНФРА-М, 2021. – 271 с.

3 **Гвоздева, В. А.** Информатика, автоматизированные информационные технологии и системы: учебник / В. А. Гвоздева. – Москва: ФОРУМ; ИНФРА-М, 2021. – 542 с.

4 **Гагарина, Л. Г.** Технология разработки программного обеспечения: учебное пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Сидорова-Виснадул; под ред. Л. Г. Гагариной. – Москва: ФОРУМ; ИНФРА-М, 2022. – 400 с.

5 **Дадян, Э. Г.** Современные технологии программирования. Язык C#: учебник: в 2 т. Т. 1: Для начинающих пользователей / Э. Г. Дадян. – Москва: ИНФРА-М, 2021. – 312 с.

6 **Мартишин, С. А.** Базы данных. Практическое применение СУБД SQL и NoSQL-типа для проектирования информационных систем: учебное пособие / С. А. Мартишин, В. Л. Симонов, М. В. Храпченко. – Москва: ФОРУМ; ИНФРА-М, 2021. – 368 с.

7 **Хорев, П. Б.** Объектно-ориентированное программирование с примерами на C#: учебное пособие / П. Б. Хорев. – Москва: ФОРУМ; ИНФРА-М, 2020. – 200 с.

8 **Черников, Б. В.** Оценка качества программного обеспечения. Практикум: учебное пособие / Б. В. Черников, Б. Е. Поклонов; под ред. Б. В. Черникова. – Москва: ФОРУМ; ИНФРА-М, 2022. – 400 с.: ил.