Рубежный контроль № 3: конспект по скриптовому языку

20 Января 2025г.

Александр Яннаев, ИУ9-11Б

Введение

Python — это высокоуровневый, интерпретируемый язык программирования, который стал основным инструментом в области прикладной информатики благодаря своей универсальности, простоте и мощным библиотекам. С момента своего создания в конце 1980-х годов Гвидо ван Россумом, Python зарекомендовал себя как язык, способный решать сложные задачи в различных областях, включая научные исследования, анализ данных, машинное обучение и веб-разработку.

- Python выделяется своей читаемостью, что делает его идеальным для разработки сложных систем. Синтаксис языка минималистичен и интуитивно понятен, что позволяет разработчикам сосредоточиться на решении задач, а не на синтаксических нюансах. Использование отступов для обозначения блоков кода способствует лучшему восприятию структуры программы.
- Python поддерживает несколько парадигм программирования, включая объектно-ориентированное, функциональное и процедурное программирование. Это делает его подходящим для разработки как небольших скриптов, так и крупных программных систем. Возможность интеграции с другими языками, такими как C и C++, расширяет его функциональность и производительность.
- Python обладает богатой экосистемой библиотек и фреймворков, что позволяет быстро разрабатывать приложения для различных задач. Библиотеки, такие как `NumPy` и `pandas`, обеспечивают мощные инструменты для анализа данных, в то время как `TensorFlow` и `PyTorch` предоставляют средства для разработки и обучения моделей машинного обучения. Это делает Python стандартом в области научных вычислений и анализа данных.
- Python является кроссплатформенным языком, что позволяет разработчикам создавать приложения, которые могут работать на различных операционных системах без необходимости изменения кода. Это особенно важно в

контексте распределенных систем и облачных вычислений, где приложения должны быть доступны на разных платформах.

- Как интерпретируемый язык, Python позволяет разработчикам выполнять код построчно, что упрощает отладку и тестирование. Это свойство делает Python идеальным для прототипирования и быстрой разработки, позволяя разработчикам быстро вносить изменения и тестировать их.

Python находит широкое применение в различных областях прикладной информатики:

- **Веб-разработка:** Использование фреймворков, таких как `Django` и `Flask`, позволяет быстро создавать масштабируемые веб-приложения с богатым функционалом.
- **Научные вычисления и анализ данных:** Библиотеки, такие как `SciPy` и `Matplotlib`, делают Python стандартом в области научных исследований, позволяя эффективно обрабатывать и визуализировать большие объемы данных.
- **Искусственный интеллект и машинное обучение:** Python стал основным языком для разработки алгоритмов машинного обучения благодаря библиотекам, таким как `scikit-learn` и `Keras`, которые упрощают процесс создания и обучения моделей.
- **Автоматизация и администрирование**: Python часто используется для написания скриптов, автоматизирующих рутинные задачи, что значительно повышает производительность и снижает вероятность ошибок.

Python представляет собой мощный инструмент для решения сложных задач. Его простота, обширная экосистема и активное сообщество делают его идеальным выбором для студентов и профессионалов, стремящихся развивать свои навыки и применять их в реальных проектах.

1. Типизация и система типов языка

Описание типизации в Python

Python является языком с динамической типизацией, что означает, что тип переменной определяется во время выполнения программы, а не при компиляции. Это делает Python более гибким, но также может привести к ошибкам, если типы не обрабатываются должным образом.

Основные типы данных в Python

В Python существует несколько встроенных типов данных. Вот таблица с некоторыми из них:

Тип данных	Описание	Пример
`int`	Целочисленный тип	`x = 5`
`float`	Число с плавающей запятой	`y = 3.14`
`str`	Строковый тип	`s = "Hello, World!"`
`list`	Список (упорядоченная изменяемая коллекция)	`my_list = [1, 2, 3]`
`tuple`	Кортеж (неизменяемая последовательность)	`my_tuple = (1, 2, 3)`
`dict`	Словарь (неупорядоченная коллекция пар ключ-значение)	`my_dict = {'a': 1, 'b': 2}`
`set`	Множество (неупорядоченная коллекция уникальных элементов)	`my_set = {1, 2, 3}`

Преобразование типов

Python поддерживает преобразование типов, что позволяет преобразовывать данные из одного типа в другой.

Примеры преобразования:

```
# Преобразование строки в целое число
num_str = "10"
num_int = int(num_str)  # num_int будет 10

# Преобразование целого числа в строку
num_new_str = str(num_int)  # num_new_str будет "10"

# Преобразование списка в кортеж
my_list = [1, 2, 3]
my_tuple = tuple(my_list)  # my_tuple будет (1, 2, 3)
```

Проверка типов

Можно использовать функцию type() для проверки типа переменной или оператор isinstance() для проверки, является ли объект экземпляром определенного типа.

Пример проверки:

```
x = 10
if isinstance(x, int):
    print("x - это целое число")
else:
    print("x - не целое число")
```

Неявная и явная типизация

В Python можно использовать как неявную, так и явную типизацию. - Неявная типизация: происходит автоматически при присваивании значения переменной. - Явная типизация: производится с помощью функций преобразования.

Пример неявной и явной типизации:

```
# Неявная типизация
a = 5           # a имеет тип int
a = "Hello"     # теперь a имеет тип str

# Явная типизация
b = str(10)     # b теперь строка "10"
```

Примечания по типизации

Python позволяет использование переменных без предварительного объявления их типа. Возможны ошибки во время выполнения программы, если неправильные типы используются с операциями, которые не поддерживают такие типы. # 2. Основные управляющие конструкции

Условные операторы

В Python для выполнения различных действий в зависимости от условий используются условные операторы. Основные из них: `if`, `elif` и `else`.

Пример:

```
x = 10
if x > 0:
    print("x положительное")
elif x < 0:
    print("x отрицательное")
else:
    print("x равно нулю")
```

Циклы

Циклы позволяют выполнять блок кода несколько раз. В Python есть два основных типа циклов: `for` и `while`.

- **Цикл `for`**

Цикл `for` используется для итерации по элементам последовательности (например, списку, строке или диапазону).

Пример:

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

- **Цикл `while`**

Цикл `while` выполняет блок кода, пока условие истинно.

Пример:

```
count = 0
while count < 5:
    print(count)
    count += 1
```

Операторы прерывания

В Python есть операторы `break` и `continue`, которые управляют выполнением циклов.

- **Оператор `break`**

Останавливает выполнение цикла.

Пример:

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

- **Оператор `continue`**

Пропускает текущую итерацию и переходит к следующей.

Пример:

```
for i in range(5):
    if i == 2:
        continue
    print(i)
```

3. Подмножество языка для функционального программирования

Основы функционального программирования в Python

Функциональное программирование — это парадигма, в которой функции рассматриваются как объекты первого класса. В Python есть множество возможностей для работы с функциональным программированием, включая использование функций высшего порядка, замыкания и лямбда-выражений.

Функции высшего порядка

Функции высшего порядка — это функции, которые принимают другие функции в качестве аргументов или возвращают их как результаты.

Пример функции высшего порядка:

```
def apply_function(f, value):
    return f(value)

def square(x):
    return x * x

result = apply_function(square, 5)  # result будет 25
```

Лямбда-выражения

Лямбда-выражения позволяют создавать анонимные функции в Python. Они используются для небольших функций, которые могут быть определены в одной строке.

Пример лямбда-выражения:

```
add = lambda x, y: x + y
result = add(2, 3)  # result будет 5
```

Замыкания

Замыкание — это функция, которая имеет доступ к переменным своей внешней функции даже после завершения ее выполнения.

Пример замыкания:

```
def outer_function(x):
    def inner_function(y):
        return x + y
    return inner_function

closure = outer_function(10)
result = closure(5)  # result будет 15
```

Использование встроенных функций для функционального программирования

Python предоставляет встроенные функции, такие как map(), filter() и reduce(), которые поддерживают функциональный стиль программирования.

- **Функция map()**

Функция `map()` применяет функцию к каждому элементу последовательности.

Пример использования `map()`:

```
numbers = [1, 2, 3, 4]
squared_numbers = list(map(lambda x: x  2, numbers))  # [1, 4, 9, 16]
```

- **Функция filter()**

Функция `filter()` возвращает элементы последовательности, которые удовлетворяют условию.

Пример использования `filter()`:

```
numbers = [1, 2, 3, 4, 5]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))  # [2, 4]
```

- **Функция reduce()**

Функция `reduce()` применяет функцию к данным в последовательности, сводя их к одному значению. Для использования `reduce()` необходимо импортировать ее из модуля `functools`.

Пример использования `reduce()`:

```
from functools import reduce
numbers = [1, 2, 3, 4]
product = reduce(lambda x, y: x * y, numbers)  # product будет 24
```

Примечания по функциональному программированию

- Функциональный подход может повысить читаемость и структуру кода.
- Важно помнить о производительности при использовании функциональных конструкций, так как они могут быть менее эффективны, чем императивный подход в некоторых случаях.

4. Объектно-ориентированное программирование (ООП) в Python

Объектно-ориентированное программирование (ООП) — это парадигма программирования, которая основывается на использовании объектов и классов. Python поддерживает ООП и предоставляет множество инструментов для работы с классами и объектами.

Основные понятия ООП

- Класс — это шаблон или чертеж для создания объектов. Он определяет свойства и методы, которые будут доступны для созданных объектов.
- Объект — это экземпляр класса. Он содержит данные (атрибуты) и функциональность (методы), определенные в классе.
- Инкапсуляция — это механизм, который скрывает внутреннее представление объекта от внешнего мира. Он позволяет управлять доступом к данным с помощью модификаторов доступа.

- Наследование — это способ создания нового класса на основе существующего, что позволяет повторно использовать уже написанный код.
- Полиморфизм — это возможность использовать один и тот же интерфейс для объектов разных типов.

Определение класса и создание объекта

Чтобы определить класс в Python, используется ключевое слово class. Пример определения простого класса:

Пример:

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

    def bark(self):
        return f"{self.name} говорит Гав!"

#Создание объекта класса Dog
dog = Dog("Шарик")
print(dog.bark())
```

Наследование

Наследование позволяет создавать новый класс, который наследует атрибуты и методы от родительского класса.

Пример:

```
class Animal:
    def speak(self):
        return "Издает звук"

class Cat(Animal):
    def speak(self):
        return "Мяу"

#Создание объекта класса Cat
cat = Cat()
print(cat.speak())  # Вывод: Мяу
```

Инкапсуляция

Инкапсуляция позволяет скрывать внутренние детали реализации. В Python для инкапсуляции используют именование с одним предшествующим символом подчеркивания.

Пример:

```
class BankAccount:
    def __init__(self, balance):
        self.__balance = balance  # Приватный атрибут

    def get_balance(self):
        return self.__balance

#Создание объекта класса BankAccount
account = BankAccount(1000)
print(account.get_balance())  # Вывод: 1000
```

Полиморфизм

Полиморфизм позволяет использовать одни и те же методы для различных классов.

Пример:

```
class Bird:
    def fly(self):
        return "Летит"

class Airplane:
    def fly(self):
        return "Самолет в воздухе"

def let_it_fly(obj):
    print(obj.fly())

bird = Bird()
airplane = Airplane()

let_it_fly(bird)      # Вывод: Летит
let_it_fly(airplane)  # Вывод: Самолет в воздухе
```

Объектно-ориентированное программирование в Python предоставляет мощные инструменты для организации и структурирования кода. Понимание классов, объектов, инкапсуляции, наследования и полиморфизма является основным для написания эффективного и поддерживаемого кода.

5. Важнейшие функции для работы с потоками ввода/вывода

Файл-объект

В Python взаимодействие с файлами осуществляется через файловые объекты, которые представляют собой абстракцию для работы с данными на диске. Для открытия файла используется функция `open()`, которая возвращает файл-объект. Этот объект предоставляет методы для чтения и записи данных, а также управления состоянием файла.

Пример создания файла-объекта:

```
file = open('example.txt', 'r')  # Открытие файла в режиме чтения
```

Чтение и запись файлов

Основные методы работы с файловыми объектами включают: - `read(size)`: считывает данные из файла. Если параметр `size` не указан, считываются все данные. - `readline()`: считывает одну строку из файла. - `readlines()`: считывает все строки из файла и возвращает их в виде списка. - `write(data)`: записывает данные в файл. - `writelines(lines)`: записывает последовательность строк в файл.

Пример чтения и записи:

```
with open('example.txt', 'w') as file:
    file.write('Hello, World!')

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

Работа со строками

Обработка строк является неотъемлемой частью работы с файлами. В Python для этого доступны различные методы, которые предоставляют функциональность для манипуляции, замены и форматирования строк.

Основные функции:

- `strip()`: удаляет начальные и конечные пробелы из строки.
- `split(separator)`: делит строку на подстроки на основе указанного разделителя.
- `join(iterable)`: объединяет все элементы переданного итерируемого объекта в одну строку, используя заданный разделитель.

Пример работы со строками:

```
line = "  Hello, World!  "
cleaned_line = line.strip()  # 'Hello, World!'
words = cleaned_line.split(', ')  # ['Hello', 'World!']
```

Регулярные выражения (модуль re)

Регулярные выражения представляют собой мощный способ поиска и манипуляции строками. Модуль `re` предоставляет функциональность для выполнения операций поиска, замены и разделения строк.

Основные операции:

- **Поиск**: `re.search(pattern, string)`: ищет первое совпадение паттерна в строке и возвращает объект-результат или None.
- **Замена**: `re.sub(pattern, replacement, string)`: заменяет все вхождения паттерна в строке на указанный текст.
- **Разделение**: `re.split(pattern, string)`: разбивает строку на части на основе паттерна.

Пример использования регулярных выражений:

```
import re
text = "Привет, мир! Программирование на Python."
pattern = r'\bП\w+'  # Слова, начинающиеся с "П"
matches = re.findall(pattern, text)  # ['Привет', 'Программирование']
modified_text = re.sub(r'Привет', 'Здравствуйте', text)
```

Таблица функций работы с регулярными выражениями

Функция	Описание
`re.search()`	Поиск первого совпадения с паттерном в строке
`re.findall()`	Поиск всех вхождений паттерна в строке
`re.sub()`	Замена вхождений паттерна на указанный текст
`re.split()`	Разбиение строки на части на основе паттерна
`re.compile()`	Компиляция регулярного выражения для последующего использования

Заключение

Python, безусловно, занимает уникальное место в экосистеме языков программирования, и его популярность объясняется множеством факторов, которые отличают его от других языков. В этом разделе мы рассмотрим ключевые аспекты, в которых Python выделяется на фоне других языков, таких как Java, C++ и R, и сделаем вывод о его значении в области прикладной информатики.

Python обладает одним из самых активных сообществ разработчиков, что обеспечивает постоянное обновление библиотек и фреймворков, а также доступ к обширной документации и ресурсам. Это делает его особенно привлекательным для студентов и профессионалов, стремящихся развивать свои навыки и находить решения для сложных задач.

Хотя Python предлагает множество преимуществ, его производительность может быть ниже по сравнению с языками, такими как C++ или Java, которые компилируются в машинный код и обеспечивают более высокую скорость выполнения. Однако для многих приложений, особенно в области анализа данных и машинного обучения, удобство разработки и скорость итерации, которые предоставляет Python, часто перевешивают недостатки производительности.

Ссылки и источники

- Статьи про Python на Хабре
- Wikipedia
- Документация Python
- Курс по Python
- Книги по Python
- Python for Everybody