Изучение базовых правил синтаксиса Python для написания корректного кода.

Python — один из самых популярных языков программирования в мире благодаря своему простому и читаемому синтаксису. В этом руководстве мы рассмотрим основные элементы синтаксиса Python, которые необходимо знать каждому начинающему программисту.

Комментарии в Python

Комментарии в Python начинаются с символа # и используются для добавления пояснений к коду. Интерпретатор Python игнорирует комментарии при выполнении программы. Существует два типа комментариев:

```
Однострочные комментарии
# print(data)
```

```
Многострочный комментарий

 """
Это многострочный комментарий
(документационная строка)
Используется для описания функций, классов и модулей

 """
```

```
Многострочный комментарий
'''
Альтернативный способ создания
многострочного комментария
'''
```

Отступы в Python

Отступы — это фундаментальная особенность Python, которая определяет структуру кода. В отличие от других языков программирования, где используются фигурные скобки, Python использует отступы для группировки блоков кода.

Правила отступов:

- Стандартный отступ: 4 пробела
- Все строки в одном блоке должны иметь одинаковый отступ
- Не смешивайте пробелы и табуляцию

```
    Многострочный комментарий

if условие:
    # Блок кода с отступом в 4 пробела
    выполнить_действие()
    другое_действие()
else:
    # Другой блок кода с тем же уровнем отступа
    выполнить_другое_действие()
```

Переменные и типы данных

Python — язык с динамической типизацией, что означает, что тип переменной определяется автоматически при присваивании значения.

Основные типы данных:

```
    Многострочный комментарий

if условие:
    # Блок кода с отступом в 4 пробела
    выполнить_действие()
    другое_действие()
else:
    # Другой блок кода с тем же уровнем отступа
    выполнить_другое_действие()
```

Примеры объявления переменных

```
x = 5              # int
y = 3.14           # float
name = "Alice"     # str
is_student = True  # bool
grades = [85, 90, 78]  # list
point = (10, 20)   # tuple
student_info = {"name": "Bob", "age": 20}  # dict
```

Операторы в Python

Python поддерживает различные типы операторов для выполнения математических и логических операций. Подробнее об операторах можно ознакомиться в «Лекции 7 – Изучение операторов и выражений в Python»

Арифметические операторы:

```
x = 10
y = 3

сумма = x + y         # 13 - сложение
разность = x - y      # 7 - вычитание
произведение = x * y  # 30 - умножение
частное = x / y       # 3.333... - деление
целое_деление = x // y # 3 - целочисленное деление
остаток = x % y       # 1 - остаток от деления
степень = x ** y      # 1000 - возведение в степень
Операторы сравнения:
равно = x == y        # False - равно
не_равно = x != y     # True - не равно
больше = x > y        # True - больше
меньше = x < y        # False - меньше
больше_равно = x >= y # True - больше или равно
меньше_равно = x <= y # False - меньше или равно
Логические операторы:
a = True
b = False

и = a and b      # False - логическое И
```

```
или = a or b      # True - логическое ИЛИ
не = not a        # False - логическое НЕ
```

Условные выражения

Условные конструкции позволяют выполнять различные блоки кода в зависимости от условий.

```
age = 18

if age >= 18:
    print("Вы совершеннолетний")
elif age >= 13:
    print("Вы подросток")
else:
    print("Вы ребенок")
```

Тернарный оператор:

Краткая форма условного выражения

```
status = "взрослый" if age >= 18 else "несовершеннолетний"
```

Циклы в Python

Циклы позволяют выполнять повторяющиеся операции.

Цикл for:

```
# Перебор чисел
for i in range(5):
    print(f"Итерация {i}")

# Перебор элементов списка
fruits = ["яблоко", "банан", "апельсин"]
for fruit in fruits:
    print(fruit)

# Перебор с индексом
for index, fruit in enumerate(fruits):
    print(f"{index}: {fruit}")
Цикл while:
count = 0
while count < 5:
    print(f"Счетчик: {count}")
    count += 1
Операторы break и continue:
for i in range(10):
    if i == 3:
        continue  # Пропустить итерацию
    if i == 7:
        break     # Выйти из цикла
    print(i)
```

Функции

Функции позволяют группировать код для повторного использования и лучшей организации программы.

```
def приветствие(имя, возраст=None):
    """
    Функция для приветствия пользователя

    Args:
        имя (str): Имя пользователя
```

```
        возраст (int, optional): Возраст пользователя

    Returns:
        str: Приветственное сообщение
    """
    if возраст:
        return f"Привет, {имя}! Тебе {возраст} лет."
    else:
        return f"Привет, {имя}!"
```

Вызов функции

```
сообщение = приветствие("Анна", 25)
print(сообщение)
```

Лямбда-функции:

Анонимная функция

```
квадрат = lambda x: x ** 2
print(квадрат(5))  # 25
```

Многострочные инструкции

Python предоставляет несколько способов записи многострочных инструкций.

Использование скобок:

```
# Длинное условие
if (температура > 25 and влажность < 60 and
    давление > 1000 and скорость_ветра < 10):
    print("Отличная погода!")

# Длинный список
продукты = [
    "хлеб", "молоко", "яйца",
    "сыр", "масло", "мясо"
]
```

Использование обратного слеша:

```
общая_сумма = цена_товара1 + цена_товара2 + \
              цена_товара3 + цена_товара4
```

Вложенные инструкции

Python поддерживает вложенные конструкции для создания сложной логики.

```
# Вложенные условия
if погода == "солнечно":
    if температура > 20:
        if есть_время:
            print("Идем на пикник!")
        else:
            print("Нет времени для пикника")
    else:
        print("Слишком холодно")
else:
    print("Погода не подходит")

# Вложенные циклы
for i in range(3):
    for j in range(3):
        print(f"i={i}, j={j}")
```

Импорт модулей

Система импорта позволяет использовать дополнительные функции и библиотеки.

```
# Импорт всего модуля
import math
результат = math.sqrt(16)

# Импорт конкретной функции
from math import sqrt, pi
результат = sqrt(16)

# Импорт с псевдонимом
import numpy as np
массив = np.array([1, 2, 3])
```

Обработка исключений

Обработка ошибок — важная часть написания надежного кода.

```
try:
    число = int(input("Введите число: "))
    результат = 10 / число
    print(f"Результат: {результат}")
except ValueError:
    print("Ошибка: введено не число")
except ZeroDivisionError:
    print("Ошибка: деление на ноль")
except Exception as e:
    print(f"Неожиданная ошибка: {e}")
finally:
    print("Блок finally выполняется всегда")
```

Работа со строками

Строки — один из самых важных типов данных в Python.

```
# Создание строк
имя = "Python"
описание = 'Язык программирования'
многострочная = """
Это многострочная
строка
"""

# Форматирование строк
возраст = 25
# f-строки (рекомендуемый способ)
сообщение = f"Меня зовут {имя}, мне {возраст} лет"

# Методы строк
текст = "python programming"
print(текст.upper())        # PYTHON PROGRAMMING
print(текст.capitalize())   # Python programming
print(текст.split())        # ['python', 'programming']
Списки и их методы
```

```
Списки — это упорядоченные изменяемые коллекции элементов.
# Создание списка
числа = [1, 2, 3, 4, 5]
смешанный = [1, "текст", 3.14, True]

# Методы списков
числа.append(6)         # Добавить элемент
числа.insert(0, 0)      # Вставить элемент по индексу
числа.remove(3)         # Удалить элемент по значению
последний = числа.pop() # Удалить и вернуть последний элемент

# Срезы списков
первые_три = числа[:3]
последние_два = числа[-2:]
каждый_второй = числа[::2]
```

Словари

Словари хранят пары ключ-значение и обеспечивают быстрый доступ к данным.

```
# Создание словаря
студент = {
    "имя": "Иван",
    "возраст": 20,
    "курс": 2,
    "оценки": [4, 5, 4, 5]
}

# Работа со словарями
print(студент["имя"])           # Иван
студент["специальность"] = "ИТ" # Добавить новый ключ
студент.update({"город": "Москва"}) # Обновить несколько ключей

# Методы словарей
ключи = студент.keys()
значения = студент.values()
элементы = студент.items()
```