Теоретический материал

Установка Python на различных операционных системах

Установка Python на Windows

Перейдите на официальный сайт Python и скачайте последнюю версию установщика для Windows. После этого апустите скачанный файл установщика.

В окне установщика отметьте галочкой пункт "Add Python to PATH" (Добавить Python в переменную окружения PATH). Это позволит использовать Python из любого места в командной строке.

Нажмите "Install Now" и дождитесь завершения установки.

Чтобы проверить, что Python установлен успешно, откройте командную строку Windows и введите команду python --version. Вы должны увидеть выведенную версию Python.

Установка Python на macOS

Перейдите на официальный сайт Python и скачайте последнюю версию установщика для macOS.

Запустите скачанный файл установщика и следуйте инструкциям на экране.

По умолчанию Python будет установлен в /usr/local/bin/python3. Чтобы проверить версию, откройте терминал и введите команду python3 --version.

Установка Python на Linux

Способы установки Python на Linux различаются в зависимости от используемого дистрибутива. Рассмотрим установку на Ubuntu:

Откройте терминал и выполните команду:

```
sudo apt-get update
sudo apt-get install python3
```

Чтобы проверить версию Python, введите в терминале:

```
python3 --version
```

Настройка среды разработки

Для эффективной работы с Python вам потребуется установить и настроить несколько инструментов. Рассмотрим основные шаги:

Шаг 1: Выбор редактора кода
Для написания и редактирования Python-кода вам понадобится текстовый редактор или интегрированная среда разработки (IDE). Популярны варианты:

Visual Studio Code - бесплатный, кроссплатформенный редактор с широким набором расширений.

PyCharm - мощная IDE от JetBrains, доступна в бесплатной и платной версиях.

Sublime Text - легкий и быстрый текстовый редактор с поддержкой множества языков.

Выберите редактор, который наиболее удобен для вас и установите его.

Шаг 2: Настройка виртуальной среды
Виртуальные среды позволяют изолировать зависимости проекта от системных библиотек Python. Для создания виртуальной среды вы можете использовать встроенный модуль venv (в Python 3.3 и новее) или сторонние инструменты, такие как virtualenv. Создайте виртуальную среду для каждого проекта, чтобы избежать конфликтов между пакетами. Активируйте виртуальную среду перед началом работы над проектом.

Шаг 3: Отладка и тестирование
Большинство IDE предоставляют встроенные инструменты для отладки кода, такие как точки останова, пошаговое выполнение и просмотр переменных.

Используйте модуль unittest или более мощные фреймворки, такие как pytest, для написания и запуска автоматических тестов.

Переменные и типы данных

Что такое переменная?

Переменная - это именованная ячейка в памяти компьютера, предназначенная для хранения данных. Переменные позволяют нам сохранять, изменять и использовать данные в нашей программе.

Как объявить переменную в Python?

Для объявления переменной в Python достаточно присвоить ей значение. Вот пример:

```
name = "John"
age = 25
is_student = True
```

В этом примере мы объявили три переменные: name, age и is_student. Имена переменных должны быть осмысленными и описывать то, что они хранят.

Типы данных в Python

Python является языком с динамической типизацией, что означает, что тип данных переменной определяется во время выполнения программы. Рассмотрим основные типы данных в Python:

1. Числовые типы:

- Целые числа (int): 42, -10
- Числа с плавающей точкой (float): 3.14, -2.5
- Комплексные числа (complex): 2+3j, -4-2j

2. Строковый тип (str): "Hello, world!", 'Python is fun'

3. Логический тип (bool): True, False

4. Последовательные типы:

- Списки (list): [1, 2, 3, "four"]

- Кортежи (tuple): (1, 2, 3, "four")
- Диапазоны (range): range(1, 11)

5. Словарный тип (dict): {"name": "John", "age": 25}

6. Множественный тип (set): {1, 2, 3, 4, 5}

Мы можем проверить тип данных переменной, используя функцию type():

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

Операции с переменными

Мы можем выполнять различные операции с переменными, например, арифметические:

```
a = 10
b = 3
print(a + b)
print(a - b)
print(a * b)
print(a / b)
print(a // b)
print(a % b)
```

Также мы можем присваивать новые значения переменным:

```
name = "Alice"
name = "Bob"
print(name)
```

Кроме того, мы можем использовать переменные в строковых литералах с помощью f-строк:

```
age = 25
print(f"My name is {name} and I'm {age} years old.")
```

Операторы и выражения

Операторы в Python - это символы или ключевые слова, которые используются для выполнения различных операций над данными. Существует несколько основных типов операторов:

1. Арифметические операторы:

- сложение (+);
- вычитание (-);
- умножение (*);
- деление (/);
- целочисленное деление (//);
- возведение в степень (**);
- остаток от деления (%).

Пример:

```
a = 10
b = 3
print(a + b)
print(a - b)
print(a * b)
print(a / b)
print(a // b)
print(a % b)
```

2. Операторы сравнения:

- Равно (==);
- Не равно (!=);
- Больше (>);
- Меньше (<);
- Больше или равно (>=);
- Меньше или равно (<=);

Пример:

```
a = 10
b = 20
print(a == b) # Вывод: False
```

```
print(a != b) # Вывод: True
print(a > b) # Вывод: False
print(a < b) # Вывод: True
print(a >= 10) # Вывод: True
print(a <= b) # Вывод: True
```

3. Логические операторы:

- И (and);
- Или (or);
- Не (not);

Пример:

```
a = True
b = False
print(a and b) # Вывод: False
print(a or b) # Вывод: True
print(not a) # Вывод: False
```

4. Операторы присваивания:

- Простое присваивание (=)
- Составные операторы присваивания (+=, -=, *=, /=, //=, **=, %=)

Пример:

```
a = 10
a += 5 # Эквивалентно a = a + 5
print(a) # Вывод: 15
```

Выражения в Python - это комбинация операндов (переменные, литералы, функции) и операторов, которые вычисляют значение. Выражения могут быть простыми или сложными, в зависимости от количества и типа операций.

Пример:

```
x = 5
y = 3
z = (x + y) * (x - y)
print(z) # Вывод: 32
```

В этом примере выражение (x + y) * (x - y) вычисляется следующим образом:
1. Сначала вычисляется x + y, результат - 8.
2. Затем вычисляется x - y, результат - 2.
3. И, наконец, выполняется умножение 8 * 2, результат - 16.

Ввод и вывод

Ввод и вывод данных - это важная часть любой программы. В Python существует несколько способов работать с вводом и выводом.

1. Вывод данных:

- Функция print() используется для вывода данных в консоль.
- Можно выводить строки, числа, переменные и другие объекты.
- Несколько объектов можно вывести, разделяя их запятыми.
- Можно форматировать вывод с помощью f-строк или метода .format().

Пример:

```
name = "Alice"
age = 25
print("My name is", name, "and I am", age, "years old.")
print(f"My name is {name} and I am {age} years old.")
print("My name is {} and I am {} years old.".format(name, age))
```

2. Ввод данных:

- Функция input() используется для получения данных от пользователя.
- По умолчанию input() возвращает введенные данные в виде строки.
- Для преобразования типов данных можно использовать функции int(), float(), bool() и др.

Пример:

```
name = input("What is your name? ")
age = int(input("How old are you? "))
print(f"Hello, {name}! You are {age} years old.")
```

3. Форматирование вывода:

- Можно использовать f-строки для вставки переменных в строку.
- Метод .format() позволяет более гибко форматировать вывод.
- Можно использовать различные спецификаторы форматирования, такие как {:d}, {:.2f}, {:10s} и др.

Пример:

```
pi = 3.14159
print(f"The value of pi is approximately {pi:.2f}")
print("The value of pi is approximately {0:.4f}".format(pi))
print("The number is {:d}".format(42))
print("The string is {:10s}".format("hello"))
```

4. Файловый ввод-вывод:

- Для работы с файлами используется функция open().
- Можно открывать файлы в различных режимах: чтение, запись, дозапись и т.д.
- Для чтения из файла используются методы read(), readline() и readlines().
- Для записи в файл используется метод write().

Пример:

```
# Чтение из файла
with open("file.txt", "r") as file:
content = file.read()
print(content)


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

Управляющие конструкции

Управляющие конструкции в Python позволяют управлять потоком выполнения программы. Рассмотрим основные типы управляющих конструкций:

1. Условные операторы:

- if-elif-else
- Используются для принятия решений в зависимости от условий

Пример:

```
x = 10
if x > 0:
print("Positive number")
elif x < 0:
print("Negative number")
else:
print("Zero")
```

2. Циклы:

- for - используется для перебора элементов итерируемых объектов (списки, кортежи, строки, и т.д.)
- while - используется для выполнения блока кода, пока условие истинно

Пример:

```
# Цикл for
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
print(fruit)


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

3. Операторы выбора:

- match-case (с Python 3.10)

- Используются для выполнения различных действий в зависимости от значения выражения
Пример:

```
day = 3
match day:
case 1:
print("Monday")
case 2:
print("Tuesday")
case 3:
print("Wednesday")
case _:
print("Invalid day")
```

4. Операторы управления циклом:

- break - прерывает выполнение цикла
- continue - пропускает текущую итерацию цикла и переходит к следующей
Пример:

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


for j in range(10):
if j % 2 == 0:
continue
print(j)
```

5. Функция pass

- Используется как заполнитель, когда в блоке кода не требуется никаких действий
Пример:

```
def my_function():
pass
```

Определение и вызов функции

Функции в Python - это именованные блоки кода, которые можно вызывать в различных частях программы. Они позволяют разбивать код на модульные части, повторно использовать код, а также улучшать читаемость и структуру программы.

1. Определение функции:

- Используется ключевое слово def для создания функции.
- За ним следует имя функции, параметры (если есть) и двоеточие.
- Внутри функции находится блок кода, который будет выполняться при ее вызове.
- Функция может возвращать значение с помощью ключевого слова return.

Пример:

```
def greet(name):
print(f"Hello, {name}!")


def add_numbers(a, b):
result = a + b
return result
```

2. Вызов функции:

- Функция вызывается по ее имени, с необходимыми аргументами (если есть).
- Аргументы передаются в скобках, разделенные запятыми.
- Результат, возвращаемый функцией, можно сохранить в переменной.

Пример:

```
greet("Alice") # Вывод: Hello, Alice!


sum_of_numbers = add_numbers(5, 3)
print(sum_of_numbers) # Вывод: 8
```

Аргументы и возвращаемые значения

Аргументы и возвращаемые значения являются ключевыми концепциями, связанными с функциями в Python. Они позволяют передавать данные в функцию, а также получать результаты ее выполнения.

1. Аргументы функции:

- Аргументы - это значения, которые передаются в функцию при ее вызове.
- Функции могут принимать как обязательные, так и необязательные аргументы.
- Необязательные аргументы могут иметь значения по умолчанию.
- Аргументы передаются в функцию в том же порядке, в котором они определены.

Пример:

```
def greet(name, greeting="Hello"):
print(f"{greeting}, {name}!")


greet("Alice") # Вывод: Hello, Alice!
greet("Bob", "Hi") # Вывод: Hi, Bob!
```

2. Возвращаемые значения:

- Функции могут возвращать одно или несколько значений с помощью ключевого слова return.
- Возвращаемые значения могут быть любых типов данных (числа, строки, списки, словари и т.д.).
- Если функция не возвращает значение явно, она возвращает значение None по умолчанию.

Пример:

```
def calculate_area(length, width):
area = length * width
return area


rectangle_area = calculate_area(5, 10)
```

```
print(rectangle_area) # Вывод: 50
```

Встроенные функции

Python предоставляет множество встроенных функций, которые можно использовать без дополнительной установки. Эти функции охватывают различные области, такие как работа с числами, строками, коллекциями, и многое другое.

Математические функции:
- abs(x) - возвращает абсолютное значение числа.
- round(x, [n]) - округляет число до n-го знака после запятой (по умолчанию 0).
- min(x1, x2, ...) - возвращает наименьшее значение из переданных аргументов.
- max(x1, x2, ...) - возвращает наибольшее значение из переданных аргументов.
- sum(iterable) - возвращает сумму элементов итерируемого объекта.

Пример:
```
print(abs(-5)) # Вывод: 5
print(round(3.14159, 2)) # Вывод: 3.14
print(min(1, 3, -2, 5)) # Вывод: -2
print(max(1, 3, -2, 5)) # Вывод: 5
print(sum([1, 2, 3, 4, 5])) # Вывод: 15
```

Использование стандартных библиотек

Python поставляется с большим количеством стандартных библиотек, которые предоставляют широкий спектр функциональности. Эти библиотеки включают в себя модули для работы с операционной системой, математические вычисления, обработку данных, сетевое взаимодействие и многое другое.

Импорт библиотек:
- Библиотеки импортируются с помощью ключевого слова import.
- Можно импортировать весь модуль или только отдельные функции/классы из модуля.

Пример:

```
import math
from math import sqrt, pi
```

Подключение и импортирование внешних модулей

В дополнение к встроенным стандартным библиотекам, Python предоставляет возможность использовать внешние модули, которые расширяют его функциональность. Эти модули могут быть установлены с помощью менеджера пакетов pip.

Установка и импорт внешних модулей:
- Внешние модули устанавливаются с помощью команды pip install <имя_модуля>.
- Для использования модуля, его необходимо импортировать в ваш Python-скрипт.

Пример:
```
# Установка модуля
pip install numpy


# Импорт модуля
import numpy as np
```

Создание собственных модулей

В дополнение к использованию стандартных и сторонних библиотек, Python также позволяет создавать собственные модули. Это позволяет организовать код, повторно использовать функциональность и улучшить модульность вашего приложения.

1. Создание модуля:
- Модуль - это просто Python-файл с расширением .py.
- Модуль может содержать функции, классы, константы и другие определения.

Пример файла my_module.py:

```
def greet(name):
print(f"Hello, {name}!")


PI = 3.14159


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


def introduce(self):
print(f"My name is {self.name} and I'm
{self.age} years old.")
```

2. Импорт модуля:
- Модуль импортируется с помощью ключевого слова import.
- Можно импортировать модуль целиком или отдельные элементы из него.

Пример импорта модуля my_module.py:

```
import my_module


my_module.greet("Alice") # Вывод: Hello, Alice!
print(my_module.PI) # Вывод: 3.14159


from my_module import Person
person = Person("Bob", 30)
```

```
person.introduce() # Вывод: My name is Bob and I'm 30 years old.
```

Исключения и их обработка

В Python исключения являются важным механизмом для обработки ошибок и нештатных ситуаций, возникающих во время выполнения программы.

Понимание исключений:
- Исключение - это объект, представляющий нештатную ситуацию, возникшую во время выполнения программы.
- Стандартные исключения в Python, такие как ZeroDivisionError, TypeError, FileNotFoundError и многие другие.

Пример:
```
print(10 / 0) # Возникает исключение
ZeroDivisionError
```

Пользовательские исключения

В Python существуют множество встроенных исключений, таких как ValueError, TypeError, FileNotFoundError и другие. Однако, иногда вам может потребоваться определить собственные исключения, которые лучше подходят для вашего приложения. Это называется созданием пользовательских исключений.

Основы создания пользовательских исключений:
- Пользовательские исключения определяются как классы, которые наследуются от базового класса Exception или одного из его подклассов.
- Создание пользовательского исключения позволяет более точно описывать и обрабатывать нештатные ситуации в вашем приложении.

Пример определения пользовательского исключения:
```
class InvalidInputError(Exception):
pass
```