

*Петербургский государственный
университет путей сообщения
Императора Александра I*

Введение в Python

Ведет: аспирант 2 года **Волков Егор Алексеевич**

gole00201@gmail.com

ауд. 11 - 304

Санкт-Петербург 2024

Что такое модуль?

- **Модуль** — это файл Python с расширением `.py`, содержащий код (функции, переменные, классы).
- Модули помогают организовать код, делают его более читаемым и позволяют повторно использовать функции и классы.

Пример: `math`, `random`, `os`

Зачем нужны модули?

- Разделяют код на логические блоки, повышая его читаемость.
- Способствуют **повторному использованию кода**.
- Упрощают **тестирование и поддержку**.
- Позволяют **импортировать** код только по необходимости.

Импортирование модулей

Основные способы

```
import math  
print(math.sqrt(16)) # Вывод: 4.0
```

```
import module_name as alias
```

```
import math as m  
print(m.sqrt(16)) # Вывод: 4.0
```

```
from module_name import object_name
```

```
from math import sqrt  
print(sqrt(16)) # Вывод: 4.0
```

```
from module_name import *
```

```
from math import *  
print(sqrt(16)) # Вывод: 4.0
```

Как создать свой модуль

Создай файл .py со своими функциями и переменными.
Импортируй его в другой файл.

Пример модуля my_module.py:

```
def greet(name):  
    return f"Hello, {name}!"  
  
pi = 3.14159
```

Импорт модуля:

```
import my_module  
print(my_module.greet("Егор")) # Вывод: Hello, Егор!
```

Пакеты и файл `__init__.py`

Пакет — это каталог, содержащий модули и файл `__init__.py`.
`__init__.py` позволяет Python распознавать каталог как пакет.
Пакеты полезны для организации большого количества модулей.

Пример структуры пакета:

```
my_package/  
|-- __init__.py  
|-- module1.py  
|-- module2.py
```

Импорт пакета:

```
from my_package import module1
```

- Работа с модулем `sys`

Модуль `sys` предоставляет доступ к переменным и функциям, взаимодействующим с интерпретатором Python.

Примеры использования *sys*

```
import sys
print(sys.version)    # Версия Python
print(sys.path)       # Пути поиска модулей
```

- Управление путями поиска модулей

`sys.path` — список директорий, которые Python просматривает при поиске модулей.

Можно добавить новый путь с помощью `sys.path.append(path)`.

```
import sys  
sys.path.append('/path/to/module')
```

Установка сторонних модулей

Для установки сторонних модулей используется pip (Python Package Installer).
Установка: `pip install module_name`

Пример:

```
pip install requests
```

```
import requests  
response = requests.get("https://api.example.com")
```

Модуль `argparse` для работы с аргументами командной строки

- `argparse` — это стандартный модуль Python для работы с аргументами командной строки.
- Облегчает **обработку аргументов и флагов** командной строки, позволяет определять **обязательные и необязательные** аргументы.

Основы работы с argparse

Простой пример

```
import argparse

# Создаем парсер
parser = argparse.ArgumentParser(description="Пример программы с argparse")

# Добавляем аргумент
parser.add_argument("name", help="Имя пользователя")

# Разбираем аргументы
args = parser.parse_args()

print(f"Привет, {args.name}!")
```

Запуск:

```
python script.py Егор  
# Вывод: Привет, Егор!
```


Добавление необязательных аргументов и флагов

- Пример с необязательным аргументом

```
parser.add_argument("-a", "--age", type=int, help="Возраст пользователя")
```

- Пример с флагом

```
parser.add_argument("-v", "--verbose", action="store_true", help="Подробный вывод")
```

- Пример скрипта с несколькими аргументами

```
import argparse

# Инициализация парсера
parser = argparse.ArgumentParser(description="Расчет площади прямоугольника")

# Обязательные аргументы
parser.add_argument("width", type=float, help="Ширина прямоугольника")
parser.add_argument("height", type=float, help="Высота прямоугольника")

# Необязательный флаг
parser.add_argument("-u", "--unit", choices=["m", "cm", "mm"], default="cm", help="Единицы измерения")

args = parser.parse_args()

area = args.width * args.height
print(f"Площадь: {area} {args.unit}^2")
```

Запуск:

```
python script.py 5 10 --unit m  
# Вывод: Площадь: 50.0 m^2
```

Файл `setup.py` для упаковки и распространения

- `setup.py` — это скрипт конфигурации, используемый для упаковки и распространения Python-проектов.
- С помощью `setup.py` можно описать:
 - Метаданные пакета (имя, версия, автор).
 - Зависимости пакета.
 - Инструкции для установки и сборки.

Базовая структура setup.py

```
from setuptools import setup, find_packages

setup(
    name="my_project",
    version="0.1",
    author="Егор Волков",
    author_email="your_email@example.com",
    description="Описание проекта",
    packages=find_packages(),
    install_requires=[
        "requests>=2.20.0", # Пример зависимости
        "numpy>=1.18.0"
    ],
    entry_points={
        "console_scripts": [
            "my_project=my_project.main:main", # Создание командной строки
        ],
    },
)
```

- `name`: Название пакета.
- `version`: Версия пакета, помогает отслеживать изменения.
- `author` и `author_email`: Информация об авторе.
- `description`: Краткое описание пакета.
- `packages`: Определяет, какие модули и пакеты включать.
 - `find_packages()` автоматически находит пакеты в проекте.
- `install_requires`: Список зависимостей пакета, устанавливаемых автоматически.
- `entry_points`: Позволяет создать скрипт командной строки.

Создание команды для установки пакета

Сначала убедись, что у тебя установлен `setuptools` и `wheel`:

```
pip install setuptools wheel
```

Для сборки пакета:

```
python setup.py sdist bdist_wheel
```

Установка пакета локально:

```
pip install .
```



```
my_project/
├── my_project/
│   ├── __init__.py
│   ├── main.py
│   ├── utils.py
│   └── submodule/
│       ├── __init__.py
│       └── helper.py
├── tests/
│   ├── __init__.py
│   └── test_main.py
├── docs/
│   └── index.md
├── setup.py
├── README.md
├── LICENSE
└── requirements.txt
```

Основная директория с исходным кодом пакета
Делает каталог "my_project" пакетом Python
Основной модуль или точка входа
Пример модуля с вспомогательными функциями
Подпакет для дополнительной логики

Пример файла в подпакете

Тесты для пакета

Пример файла с тестами

Документация проекта
Основной файл документации

Файл конфигурации для установки пакета
Описание проекта
Лицензия проекта
Зависимости проекта для установки через pip

Лабораторная работа №6

- Создать свой собственный пакет содержащий в себе функции для решения **Лабораторной работы №2**
- Пакет должен быть устанавливаемый через `pip install .`
- Пакет должен содержать всю необходимую метайнформацию в `setup.py`