# ПМ3 Разработка модулей ПО.

## РО 3.1 Понимать и применять принципы объектно-ориентированного и асинхронного программирования.

# Тема 12. Модули. Импорты.

# Цель занятия:

Сформировать представление о модулях в Python, объяснить назначение модульной структуры программ, научить подключать модули различными способами и использовать стандартные и пользовательские модули.

# Учебные вопросы:

1. Понятие модуля.
2. Стандартные модули Python.
3. Импорт модулей.

# 1. Понятие модуля

Модуль в Python — это отдельный файл с расширением **.py**, содержащий:

- функции,
- переменные,
- классы,
- исполняемый код,

который может быть подключён и использован в других программах.

# Назначение модулей

Модули используются для:

- логического разделения программы на части;
- повторного использования кода;
- упрощения поддержки и развития программ;
- предотвращения дублирования кода.

# Модульность программы

Большие программы удобно разбивать на модули, каждый из которых отвечает за свою часть логики.

Пример структуры программы:

```
project/
├── main.py
├── math_utils.py
└── data_utils.py
```

# Пример простого модуля

Файл math_utils.py:

```
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b
```

# Использование модуля

В другом файле можно использовать функции из модуля:

```
import math_utils

result = math_utils.add(5, 3)
print(result)
```

# 2. Стандартные модули Python.

Стандартная библиотека Python — это набор модулей, которые:

- поставляются вместе с Python;
- не требуют дополнительной установки;
- предназначены для решения типовых задач.

Эти модули охватывают работу с:

- математикой,
- датами и временем,
- файлами и операционной системой,
- аргументами командной строки,
- случайными величинами и др.

# Модуль math

Модуль math предоставляет функции и константы для математических вычислений.

Примеры:

```
import math

print(math.sqrt(16))    # квадратный корень
print(math.pi)          # число π
print(math.factorial(5))
```

Основные возможности:

- тригонометрические функции;
- степень, корень, логарифмы;
- математические константы (pi, e).

# Модуль random

Модуль random используется для работы со случайными числами.

Примеры:

```
import random

print(random.randint(1, 10))      # случайное целое число
print(random.choice([1, 2, 3]))   # случайный элемент списка
```

Основные функции:

- генерация случайных чисел;
- выбор случайных элементов;
- перемешивание последовательностей.

# Модуль datetime

Модуль datetime предназначен для работы с датой и временем.

Примеры:

```
from datetime import datetime


now = datetime.now()
print(now)
print(now.year, now.month, now.day)
```

Возможности:

- получение текущей даты и времени;
- работа с датами;
- форматирование дат.

# Модуль os

Модуль os предоставляет средства взаимодействия с операционной системой.

Примеры:

```
import os

print(os.getcwd())          # текущий каталог
print(os.listdir())         # список файлов
```

Основные задачи:

- работа с файлами и каталогами;
- получение переменных окружения;
- запуск команд операционной системы.

# Модуль sys

Модуль sys предоставляет доступ к параметрам и функциям интерпретатора Python.

Примеры:

```
import sys

print(sys.argv)     # аргументы командной строки
print(sys.path)     # пути поиска модулей
```

Назначение:

- работа с аргументами командной строки;
- управление завершением программы;
- доступ к настройкам интерпретатора.

# 3. Импорт модулей.

Стандартная библиотека Python — это набор модулей, которые:

- поставляются вместе с Python;
- не требуют дополнительной установки;
- предназначены для решения типовых задач.

# Оператор import

Для подключения модуля в Python используется оператор **import**.

После импорта становятся доступны функции, переменные и классы модуля.

```
import math
```

# Импорт всего модуля

При импорте всего модуля необходимо обращаться к его элементам через имя модуля.

```
import math


print(math.sqrt(25))
print(math.pi)
```

После импорта модуля его элементы становятся доступны через точечную нотацию:

**module_name.element**

где element — это функция, переменная или константа, определённая в модуле.

Пример с модулем math:

```
import math

result = math.sqrt(36)
print(result)
```

Здесь:

math — имя модуля;

sqrt — функция из модуля math.

Преимущества:

- понятное происхождение функций;
- отсутствие конфликтов имён.

Недостаток:

- необходимость писать имя модуля перед каждым обращением.

# Импорт отдельных объектов из модуля

Можно импортировать только нужные элементы с помощью конструкции:

```
from ... import ....
```

```
from math import sqrt, pi

print(sqrt(25))
print(pi)
```

Если используется from ... import ..., имя модуля не указывается.

```
from math import pi, sqrt


print(pi)
print(sqrt(49))
```

Преимущества:

- краткая запись;
- удобно при частом использовании функций.

Недостатки:

- возможны конфликты имён;
- сложнее определить источник функции.

# Импорт всех объектов из модуля

Конструкция:

**from module import ***

импортирует все публичные имена (функции, переменные, классы) из модуля напрямую в текущее пространство имён.

После такого импорта к элементам модуля можно обращаться без имени модуля.

Пример:

```
from math import *

print(sqrt(16))
print(pi)
```

**import *** считается плохой практикой.

Невозможно понять, из какого модуля пришла функция или переменная.

## Импорт с псевдонимом (as)

Можно задать псевдоним для модуля или объекта.

Псевдоним для модуля:

```
import math as m
print(m.sqrt(16))
```

Псевдоним для объекта

```
from math import sqrt as square_root
print(square_root(16))
```

Когда используется **as**:

- для сокращения длинных имён;
- при совпадении имён из разных модулей;
- для повышения читаемости кода.

Вывод:

- **import module** — импорт всего модуля.
- **from module import object** — импорт отдельных элементов.
- **as** — задание псевдонима.
- Выбор способа импорта зависит от задачи и требований к читаемости кода.

# 4. Пользовательские модули.

Пользовательский модуль — это файл с расширением **.py**, созданный программистом, который содержит функции, переменные или классы и предназначен для повторного использования в других программах.

## Создание собственного модуля.

Для создания модуля достаточно:

- создать файл с расширением .py;
- написать в нём код;
- сохранить файл в доступной директории.

Пример пользовательского модуля

Файл **utils.py**:

```
def add(a, b):
    return a + b

def multiply(a, b):
    return a * b
```

# Структура файла модуля

Типичная структура пользовательского модуля может включать:

- импорты других модулей;
- функции и переменные;
- docstring модуля;
- блок **if __name__ == "__main__"** (опционально).

Подключение пользовательского модуля.

Импорт всего модуля.

```
import utils

result = utils.add(3, 4)
print(result)
```

Импорт отдельных элементов

```
from utils import multiply

print(multiply(5, 6))
```

# Импорт с псевдонимом.

```
import utils as ut

print(ut.add(2, 3))
```

Где должен находиться файл модуля?

Файл модуля должен:

- находиться в той же директории, что и основной файл программы;
- или быть в одной из директорий, указанных в sys.path.

# 5. Переменная __name__ и точка входа программы.

Назначение переменной **__name__**

**__name__** — это служебная переменная Python, которая автоматически создаётся для каждого модуля.

Её значение зависит от того:

- как запускается файл;
- или как он используется (импортируется).

Значения переменной **__name__**

Если файл запускается напрямую, то:

**__name__ == "__main__"**

Это означает, что файл является точкой входа программы.

Импорт файла как модуля

Если файл импортируется в другую программу:

**__name__ == "имя_модуля"**

То есть **__name__** будет равно имени файла (без .py).

Конструкция **if __name__ == "__main__"**:

Эта конструкция позволяет:

- определить код, который должен выполняться только при прямом запуске файла;
- предотвратить выполнение этого кода при импорте модуля.

Общий вид

```
if __name__ == "__main__":
    # код, выполняемый только при запуске файла
```

# Практический пример

Файл **utils.py**:

```python
def add(a, b):  2 usages
    return a + b


def main():  1 usage
    print(add( a: 2,  b: 3))


if __name__ == "__main__":
    main()
```

Файл **main.py**:

```python
import utils


result = utils.add( a: 20,  b: 30)
print(result)
```

Зачем нужна точка входа программы:

- отделяет тестовый или демонстрационный код от логики модуля;
- делает модуль безопасным для импорта;
- повышает читаемость и переиспользуемость кода.

# Контрольные вопросы:

- Что такое модуль в Python?
- В чём преимущество использования модулей?
- Чем отличается import module от from module import ...?
- Для чего используется псевдоним as?
- Что такое стандартная библиотека Python?
- Как создать и подключить собственный модуль?
- Зачем используется конструкция if __name__ == "__main__"?

# Домашнее задание:

https://ru.hexlet.io/courses/js-asynchronous-programming

## Материалы лекций:

https://github.com/ShViktor72/Education

## Обратная связь:

colledge20education23@gmail.com