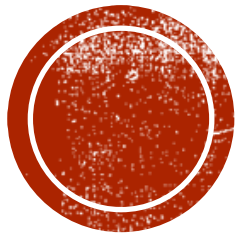
ЯЗЫКИ ПРИКЛАДНОГО ПРОГРАММИРОВАНИЯ

Лекция 12

```python
a = "str"
match a:
    case "шаблон_1":
        print("действие_1")
    case "шаблон_2":
        print("действие_2")

    #...............

    case "шаблон_N":
        print("действие_N")
    case _:
        print("действие_по_умолчанию")

def print_hello(language):
    match language:
        case "russian":
            print("Привет")
        case "american english" | "british english" | "english":
            print("Hello")
        case _:
            print("Undefined")
```

PATTERN MATCHING

Аналог **switch-case**, появился в версии 3.10, подробный разбор оператора можно почитать по ссылке.



ООП в Python

ОСНОВНОЙ СИНТАКСИС

```python
class class_name:

    static_field = 42 # аналог static из .NET/C++ доступно без создания класса
    __private_static_field = 42 # "приватне" поле, но доступ все равно можно получить
    #  _class_name__private_static_field

    @staticmethod # декоратор обозначающий что метод статический
    def static_method(): # статический метод
        print("static method")

    def __init__(self): # конструктор объекта
        # self - аналог this, должен быть в каждом методе класса
        self.class_field = "some str" # публичное поле класса
        self.__private_field = "secret" # приватное поле класса

    def method(self, param): # метод класса
        return self.__private_field + " " + param


print(class_name.static_field) # печать статического поля
class_name.static_method() # вызов статического метода

obj = class_name() # создание класса
print(obj.class_field) # доступ к полю класса
obj.method("some value") # вызов метода
```

СВОЙСТВА

```python
# пример применения свойств:
class Rectangle:

    def __init__(self, width, height):
        self.__width = width
        self.__height = height

    @property
    def width(self):
        return self.__width

    @width.setter
    def width(self, w):
        if w > 0:
            self.__width = w
        else:
            raise ValueError

    @property
    def height(self):
        return self.__height

    @height.setter
    def height(self, h):
        if h > 0:
            self.__height = h
        else:
            raise ValueError

    def area(self):
        return self.__width * self.__height

rect = Rectangle(10, 20)
print(rect.width)
print(rect.height)
```

НАСЛЕДОВАНИЕ

- Доступно множественное наследование
- Все классы унаследованы от **object**, явно это указывать не нужно
- Синтаксис создания класса наследника:
  - **class имя_класса(имя_родителя1, [имя_родителя2,..., имя_родителя_n])**
- super –ключевое слово, для обращения к родительскому классу из наследника
- **isinstance(object, type)** – проверка принадлежит ли объект типу (точное совпадение)

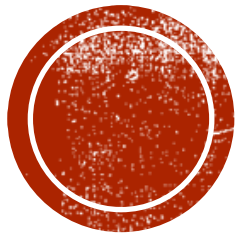
ПРИМЕР

```python
class Figure:
    def __init__(self, color):
        self.color = color

    # пример переопределения метода (переопределяется метод object)
    # __str__ отвечает за строковое представление метода
    def __str__(self):
        print("Figure")
        print("Color: " + self.__color)


class Rectangle(Figure):
    def __init__(self, width, height, color):
        super().__init__(color) # вызов базового конструктора
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

    # пример переопределения метода (переопределяется метод object)
    def __str__(self):
        print("Rectangle")
        print("Color: " + self.color)
        print("Width: " + str(self.width))
        print("Height: " + str(self.height))
        print("Area: " + str(self.area()))
```

ИТЕРАТОРЫ

- Итератор – объект который реализует метод **__next__** и позволяет обходить коллекцию
- Окончание обозначается при помощи бросания исключения **StopIteration**
- Чтобы по классу можно было итерироваться необходим метод **__iter__** возвращающий итератор
  - *(в примере объект сам себе итератор)*

```
class SimpleIterator:
    def __iter__(self):
        return self

    def __init__(self, limit):
        self.limit = limit
        self.counter = 0

    def __next__(self):
        if self.counter < self.limit:
            self.counter += 1
            return self.counter
        else:
            raise StopIteration


s_iter = SimpleIterator(5)
for i in s_iter:
    print(i)
```



ПОДРОБНЕЕ О ФУНКЦИЯХ

ЧТО ТАКОЕ ФУНКЦИЯ В PYTHON

- Функция в **python** это объект с методом **def __call__(self)**
- Функция может создаваться в другой функции
- Функции могут передаваться в другие функции как параметры

LAMBDA

- Lambda-функция – это безымянная функция с произвольным числом аргументов и вычисляющая одно выражение. Тело такой функции не может содержать более одной инструкции (или выражения).

```
l = [1, 2, 3, 4, 5, 6, 7]
f = lambda x: x**3
print(list(map(f, l)))

## [1, 8, 27, 64, 125, 216, 343]
```

ЗАМЫКАНИЯ

- Замыкание - функция, которая находится внутри другой функции и ссылается на переменные объявленные в теле внешней функции.

```
# пример функции которая возвращает функцию (замыкание) умножения на переданный параметр a
def mul(a):
    def helper(b):
        return a * b
    return helper

two_mul = mul(2)
print(two_mul(5)) # 10
```

ДЕКОРАТОРЫ

- Декоратор – функция, аргументом которой является другая функция.
- Декоратор предназначен для добавления дополнительного функционала к функции без ее изменения.

```
# декоратор для функции с одной переменной
def print_fn_info(fn):
    def wrapper(arg):
        print("Run function: " + str(fn.__name__) + "(), with param: " + str(arg))
        return fn(arg)
    return wrapper
```

ДЕКОРАТОРЫ

- Декоратор – функция, аргументом которой является другая функция.
- Декоратор предназначен для добавления дополнительного функционала к функции без ее изменения.

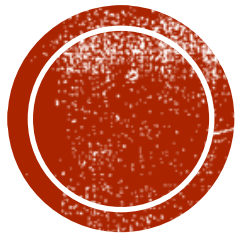
```
# декоратор для функции с одной переменной
def print_fn_info(fn):
    def wrapper(arg):
        print("Run function: " + str(fn.__name__) + "(), with param: " + str(arg))
        return fn(arg)
    return wrapper
```

```python
def pow2(x):
    return x * x

print(pow2(2))

weapped_pow2 = print_fn_info(pow2)
pow2 = weapped_pow2

print(weapped_pow2(2)) # Run function: pow2(), with param: 2
print(pow2(2)) # Run function: pow2(), with param: 2

# чтобы не писать постоянно

# weapped_pow2 = print_fn_info(pow2)
# pow2 = weapped_pow2

# используется следующий синтаксис:

@print_fn_info
def pow2(x):
    return x * x
```

ПРИМЕРЫ

ПРИМЕРЫ

```python
# декоратор для метода класса
def method_decor(fn):
    def wrapper(self):
        print("Run method: " + str(fn.__name__))
        fn(self)
    return wrapper

class Vector():
    def __init__(self, px = 0, py = 0):
        self.px = px
        self.py = py

    @method_decor
    def norm(self):
        print((self.px**2 + self.py**2)**0.5)

vc = Vector(px=10, py=5)
print(vc.norm())
# Run method: norm
# 11.180339887498949
```



МОДУЛИ И ПАКЕТЫ

МОДУЛИ

- Модуль – файл с расширением **.py***

- Подключение модуля:
  - *import <module_name1>, <module_name2>, <module_name3>*
  - *import <module_name1> as <new_name>*
  - *from <module_name> import <name1>, <name2>*
  - *from <module_name> import **
  - *from <module_name> import <name> as <new_name>*

- ** - Модуль может быть написан не только на **python***

```
from math import factorial as f
f(4)

from math import cos, sin, pi
cos(pi/3)
sin(pi/3)

import math as m
m.sin(m.pi/3)

from math import *
cos(pi/2)
sin(pi/4)
```

ПАКЕТЫ

- Пакеты – аналог **namespace** из **.NET**
- Пакет – каталог, содержащий файл __init__.py.
- Пакеты используются для формирования пространства имен, что позволяет работать с модулями через указание уровня вложенности (через точку).
- Пакет может включать в себя модули и другие пакеты.
- Для импортирования пакетов используется тот-же синтаксис, что и для импорта модулей.

ПАКЕТЫ

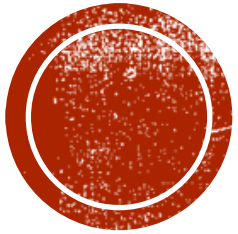
- Файл __init__.py может быть пустым или содержать переменную *__all__*, хранящую список модулей, который импортируется при загрузке через конструкцию.
- Пример – см. репозиторий (**Python/my_package**)

УСТАНОВКА СТОРОННИХ ПАКЕТОВ

- Сторонние пакеты устанавливаются при помощи утилиты **pip** (**pip3** если у вас установлен **python2**)

- **Pip install** _имя_пакета_

- *Для изоляции версий могут быть использованы различные виртуальные окружения (подробности по ссылке)*



СТАНДАРТНАЯ БИБЛИОТЕКА

Полная документация

Версия на русском *(вероятно неполная)*

МОДУЛИ ДЛЯ РЕАЛИЗАЦИИ ИНТЕРФЕЙСА И ЛОГГИРОВАНИЯ

- curses - графический интерфейс в терминале.
- tkinter - стандартный кроссплатформенный графический интерфейс.
- logging – модуль для логирования.
- gettext – модуль для добавления локализации.

МОДУЛИ ОБЩЕГО НАЗНАЧЕНИЯ

- calendar - работа с календарем.
- datetime - работа с датой и временем (чтение, парсинг, операции).
- difflib - сравнение последовательностей.
- hashlib - модуль с набором хеш функций: SHA1, SHA224, SHA256, SHA384, SHA512, MD5...
- decimal - функции для быстрого преобразования чисел с плавающей точкой.
- math - функции для работы с математикой.
- random - генератор псевдо-случайных чисел.
- re – работа с регулярными выражениями синтаксиса perl.
- string - функции для работы со строками.

МОДУЛИ ДЛЯ РАБОТЫ С РАЗЛИЧНЫМИ ФОРМАТАМИ

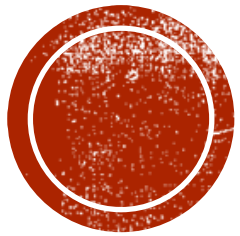
- html – модуль для разбора html страниц.
- json - модуль для работы с форматом json.
- xml - модуль для работы с форматом xml.
- csv - модуль для работы с форматом csv.
- gzip, zlib - модули для работы со сжатыми данными.
- email - модуль для разбора структуры email сообщений, проверки email и т.д.
- configparser - чтение содержимого конфигурационных файлов формата ini.

МОДУЛИ ВВОДА/ВЫВОДА И ВЗАИМОДЕЙСТВИЯ С ОС

- io - основные функции для работы с потоками ввода/вывода (текст, бинарные…)
- os - взаимодействие с операционной системой.
- pathlib - работа с путями в файловой системе.
- threading – модуль для работы с многопоточностью.

МОДУЛИ ДЛЯ РАБОТЫ С СЕТЬЮ

- http - работа с интернет ресурсами по протоколу HTTP.
- ssl - работа с ssl сертификатами, используется для получения html страниц по протоколу https.
- socket - работа с сокетами напрямую.
- urllib - простая работа с URL.



СТОРОННИЕ МОДУЛИ

АННОТАЦИЯ ТИПОВ

Согласно PEP 3107 могут быть следующие варианты использования аннотаций:

- проверка типов;
- расширение функционала IDE в части предоставления информации об ожидаемых типах аргументов и типе возвращаемого значения у функций;
- перегрузка функций и работа с дженериками;
- взаимодействие с другими языками;
- использование в предикатных логических функциях;
- маппинг запросов в базах данных;
- маршалинг параметров в RPC (удаленный вызов процедур).

- Подробности по ссылке

АННОТАЦИЯ ПРИМЕР

```
name = "John" # type: str
print(name)
name = 10
print(name)

# python -m pip install mypy - установка пакета аннотаций
# python -m mypy test_type.py

# test_type.py:3: error: Incompatible types in assignment (expression has type “int”, variable has type “str”)
```

JUPITER NOTEBOOK

- Интерактивный «блокнот» поддерживающий **python** и некоторые другие языки.
- Веб-среда разработки => может запускаться локально или с удаленного сервера
- В основном применяется в **ML, DS** и т.п.

- Установка:
  - **Pip install jupyter**
  - Или скачать и установить **Anaconda** (пакет **python** + набор распространенных сторонних пакетов)
- Запуск:
  - **jupyter notebook**

ПАКЕТЫ ДЛЯ DS

- **NumPy –** быстрые вычисления, математические структуры данных (многомерные массивы), множество готовых математических функций
- **SciPy** – основан на **NumPy,** набор подпакетов, в которых реализованы различные вычислительные механизмы (быстрое преобразование Фурье, решение дифференциальных уравнений, механизмы линейной алгебры…)
- **Pandas –** использует **NumPy** и **SciPy**, предназначен для анализа данных в том числе больших объемов.
- **StatsModels** – использует **NumPy** и **SciPy,** также используется для анализа данных и построения статистических моделей.

ПАКЕТЫ ДЛЯ ОТОБРАЖЕНИЯ ДАННЫХ

- **Matplotlib** – пакет для визуализации данных с богатым функционалом построения графиков
- **Seaborn** – расширение для Matplotlib, направленное на то, чтобы сделать графики Matplotlib привлекательнее и упростить создание сложных визуализаций.
- **Plotly** - интерактивные графики, позволяющие исследовать взаимоотношения переменных. Продвинутый функционал по построению трехмерных графиков

- **Scikit-Learn**
- **Keras**

ML

- **TensorFlow** – библиотека для машинного обучения (не только для **python**)
- **Keras** – надстройка над **TensorFlow**
- **Scikit-Learn**