Лекция №1 - вводный курс в Python. Синтаксис.

- Где взять: https://www.python.org/
- Документация: https://docs.python.org/3/
- PEP: https://www.python.org/dev/peps/

Еловских Михаил, wronglink@yandex.ru

Синтаксис

```
#!/usr/bin/env python
import random

def roll_dice(min, max):
    """
    Функция, имитирующая бросок кубика. Возвращает
    случайное число на одной из его сторон.
    """
    value = random.randint(min, max)
    return value

if __name__ == '__main__':
    min, max = 1, 6

    # Бросим кубик 10 раз и выведем результат
    for i in range(10):
        value = roll_dice(min, max)
        print(value)
```

Python

- Блоки выделяются отступами (принято 4 пробела)

```
def a():
    for i in range(10):
        print(i)
```

- Динамически типизированный

```
a = 1
b = 2
```

- Регистро зависимый (`var != Var`)

Кавычки, комментарии

```
a = "Привет"
b = 'Мир'
c = """
Привет
Мир
"""
d = '''
Привет
Мир
'''
```

Переменные и идентификаторы

1. Начинаются на `a-zA-Z_`
2. Могут содержать `a-zA-Z0-9_`
3. Чувствительны к регистру
4. Имена классов `CamelCase`
5. Имена аттрибутов и функций - `snake_case`
6. `_my_attribute` - "приватный" аттрибут
7. `__my_attribute` - очень "приватный" аттрибут

Типы:

- Числа: `float()` и `int()` (и `long()`)
- Коллекции: `list()`, `tuple()`, `dict()`, `set()`
- Строки: `str()`
-

Условные ветвления

```
a = 1

if a > 1:
    print("a > 1")
elif a < 1:
    print("a < 1")
else:
    print("a == 1")
```

Однострочники:

```
a = 'Четное' if i % 2 = 0 else 'Нечетное'
```

Циклы

```
numbers = [1, 2, 3, 4, 5]
for number in numbers:
    print(number)
```

```
numbers = range(10)
while numbers:
    print(numbers.pop())
```

- `break` - прервать цикл
- `continue` - следующая итерация
- `else` - если цикл не был прерван

Циклы

```
def is_spammer(messages):
    # messages = ['Привет', 'Как дела?', ...]

    # Проверим, спам-фильтром
    for message in messages:
        if is_spam(message):
            return True
    else:
        return False
```

Полезности:

- `enumerate(iterable)` - возвращает `i, value`
- `range()` - возвращает итератор значений
- `itertools` - модуль для работы с итераторами

List comprehansions и другие comprehansions

```
my_list = []

for i in range(5):
    if i % 2 == 0:
        my_list.append(i + 1)

my_list = [i + 1 for i in range(5) if i % 2 == 0]
```

Словари и множества:

```
my_dict = {i: i for i in range(10)}
my_set = {i for i in range(10)}
```

Исключения

1. Наследуются от базового `Exception`
2. Существуют отдельные `KeyboardInterrupt` и `SyntaxError`
3. Выбрасываются `raise MyException`
4. Список встроенных исключений.

Обработка исключений

Обработка:

```
try:
    1 / 0
except ZeroDivisionError as exception:
    # Если был ZeroDivisionError
except ValueError:
    # Если был ValueError
except:
    # Если было любое другое исключение
    print("Unhandled error")
    raise
else:
    # Не было исключений
finally:
    # Выполнится в любом случае
```

Контекстные менеджеры

```
try:
    file = open('file.txt')
    # работаем с файлом
finally:
    file.close()
```

```
with open('file.txt') as file:
    # работаем с файлом
```

- Работа с файлами
- Взятие локов
- Изменение окружения

Функции

```
def my_function(param1, param2=123, *args, **kwargs):
    print(param1, param2, args, kwargs)
```

- `param1` - обязатетльный параметр
- `param2` - опциональный параметр
- `args`, `kwargs` - доп. параметры

```
my_function(12, 123, 1234, 12345)
my_function(param2=12, param1=123,
            param3=1234, param4=12345)
```

Функции

Функция - first class object.

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

def caller(func, *args):
    return func(*args)

another_add = add
caller(another_add, 1, 2)
```

Анонимные функции

```
a = lambda x, y: x + y
```

Декораторы

```
def my_decorator(func):
    def wrapper():
        print("Before func() is called")
        func()
        print("After func() is called")
    return wrapper

def my_function():
    print("Hello world!")

my_function = my_decorator(my_function)
my_function()
```

Декораторы

```
def my_decorator(func):
    def wrapper():
        print("Before func() is called")
        func()
        print("After func() is called")
    return wrapper

@my_decorator
def my_function():
    print("Hello world!")
```

- Тайминг
- Логгирование
- Добавление логики
- Регистрация handler-ов

Классы

```
class MyClass():
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def add(self):
        return self.a + self.b
instance = MyClass(1, 2)
instance.add()
```

Классы

- Множественное наследование
- Миксины
- `type(instance)` - получить класс инстанса
- `isinstance()`, `issubclass()` - работа с наследованием
- `__method__` - магические методы

Итераторы и генераторы

```
iterator = iter(obj)`
next(iterator)
next(iterator)
```

- Класс реализует метод `__iter__`
- Итератор реализует метод `__next__`
- По окончании возвращает `StopIteration`

```
def reverse(data):
    for index in range(len(data)-1, -1, -1):
        yield data[index]
```