Python

Почему python?

- Прост в изучении
- Большое количество модулей
- Один из самых популярных языков
- Стоит знать яп с динамической типизацией (об этом во второй части курса)

Когда использовать?

- Когда эффективность разработки важнее
- Исследовательское программирование (ML и т.п.)
- Прототипы
- Вспомогательные программы

Медленно?

- 90% времени выполнения тратится на 10% кода
- 90% кода не нужно оптимизировать, но отлаживать все равно нужно

Что нужно знать на текущем этапе

- Интерпретируемый (с предварительной компиляцией)
- Динамическая, но строгая типизация
- Автоматическое управление памятью
- Механизм обработки исключений

Интерпретаторы

- CPython
- IronPython (.NET)
- Jython JVM
- PyPy

Будем использовать CPython. В курсе подразумевается работа с версией Python 3.7+

Где взять Python?

- Если нужно все и сразу: anaconda
- По-умолчанию можно: CPython
- Для домашек по ML: jupyter

Типы данных

`int`, `float`, `complex`, `bool`, `NoneType` (единственный экземпляр — `None`)

Булевы операции

- `X or Y`
- `X and Y`
- `not X`

Типы данных

Числовые операции

- `x + y`, `x - y`, `x * y`, `x / y`, `-x`, `+x`, `x // y`, `x % y`
- `pow(x, y)`, `x ** y`
- операции с присваиванием: `+=`, `-=`, `*=`, `**=` и пр.
- `x << y`, `x >> y`, `x & y`, `x | y`, `~ x`, `x ^ y`
- `ceil`, `floor` и др.
- `<`, `<=`, `>`, `>=`, `==`, `!=`

Коллекции. list

```
In : a = ['one', 'two', 'three']

In : a[2] + ' people'
Out: 'three people'

In : a + [2, 3]
Out: ['one', 'two', 'three', 2, 3]

In : a.append(True)

In : a
Out: ['one', 'two', 'three', True]
```

Коллекции. list

```
In : a.extend([4])

In : a
Out: ['one', 'two', 'three', True, 4]

In : a.insert(0, 'zero') # Почему плохо?

In : a
Out: ['zero', 'one', 'two', 'three', True, 4]
```

Коллекции. list

```
In : a = ['one', 'two', 'three', 'for']

In : a[1:3:2]
Out: ['two']

In : a[::2]
Out: ['one', 'three']
In : a[0:2] = []

In : a
Out[20]: ['three', 'for']
```

Коллекции. list

```
In : a = ['one', 'two', 'three', 'for']

In : a.remove(152)
------------------
ValueError          Traceback (most recent call last)
----> 1 a.remove(152)
ValueError: list.remove(x): x not in list

In : a.index('one')
Out: 0

In : 'two' in a
Out: True

In : len(a)
Out: 4
```

Коллекции. tuple

```
In : a = ('one', 'two', 'three', 'for')

In : a[1]
Out: 'two'

In : a[::2]
Out: ('one', 'three')

In : a[0] = 50
---------------
TypeError                 Traceback (most recent call last)
----> 1 a[0] = 50
TypeError: 'tuple' object does not support item assignment
```

Коллекции. dict

```
In : dict = {}

In : circus = {}

In : circus = {'lion': 130, 'giraffe': 800, 'hippo': 1300}

In : circus['lion']
Out: 130

In : circus['bear']
--------------
KeyError                  Traceback (most recent call last)
----> 1 circus['bear']
KeyError: 'bear'
```

Коллекции. dict

```
In : circus['panda'] = (70, 100)
In : circus
Out: {'lion': 130, 'giraffe': 800, 'hippo': 1300, 'panda': (70,
100)}

In : circus.get('bear', (80, 600))
Out: (80, 600)
```

Коллекции. dict

```
In : circus.keys()
Out: dict_keys(['lion', 'giraffe', 'hippo', 'panda'])
In : circus.keys()[0]
--------------------
TypeError          Traceback (most recent call last)
----> 1 circus.keys()[0]

TypeError: 'dict_keys' object is not subscriptable

In : circus.values()
In : circus.items()



In : 'giraffe' in circus.keys()
Out: True

In : 'giraffe' in circus # делайте так!
Out: True
```

Коллекции. set

```
In : basket = {'apple', 'orange', 'apple', 'pear', 'orange',
'banana'}
In : print(basket)
Out: {'orange', 'banana', 'pear', 'apple'}

In : 'orange' in basket # O(1)
Out: True

In : 'crabgrass' in basket
Out: False
```

Коллекции. set

```
In : a = set('abracadabra')
In : b = set('alacazam')
In : a
Out: {'a', 'r', 'b', 'c', 'd'}

In : a - b # разница
Out: {'r', 'd', 'b'}
In : a | b # объединение
Out: {'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
In : a & b # пересечение
Out: {'a', 'c'}
In : a ^ b # симметрическая разность
Out: {'r', 'd', 'b', 'm', 'z', 'l'}
```

Строки.

```
In : 'String'
Out: 'String'

In : "String"
Out: 'String'

In : "String" == 'String'
Out: True

In : """Multiline
       : string"""
Out: 'Multiline\nstring'

In : "Lecture".find("reg")
Out: -1

In : "    boo    ".strip()
Out: 'boo'
```

Строки.

```
In : a = "8-301-30-2-20-30".split("-")
In : a
Out: ['8', '301', '30', '2', '20', '30']

In : " ".join(a)
Out: '8 301 30 2 20 30'

In : name = "John"
In : surname = "Doe"

In : f'Welcome back, {name} {surname}'
Out: 'Welcome back, John Doe'
```

Изменяемые и неизменяемые объекты

Объекты некоторых встроенных типов нельзя изменять после создания: `bool`, `int`, `float`, `str`, `tuple`.

Поэтому их ещё называют **неизменямыми/immutable** объектами.

А объекты других типов, например, `dict` или `list`, соответственно, **изменяемыми/mutable** объектами.

Термин весьма условный и средствами языка выражается только через определение магических и обычных методов таким образом, что они не могут изменить объект.

Изменяемые и неизменяемые объекты

```
In : a = 5
In : id(a)
Out: 139877966367152

In : a += 1
In : id(a)
Out: 139877966367184

In : a = (1, 2)
In : a[0]
Out: 1
In: a[1] = 50
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Хешируемые и нехешируемые объекты

Объект называется **хешируемым**, если для него можно вычислить хеш-значение, которое не меняется в течение жизни объекта.

Вычисляется хеш вызовом специального магического метода (`__hash__()`).

Все встроенные неизменяемые объекты хешируемые.

Ключи словарей и значения множества обязательно должны быть хешируемыми (почему?).

Хешируемые и нехешируемые объекты

```
dict_with_wrong_key = { [1, 2, 3]: 10 }
# TypeError                  Traceback (most recent call last)
# ----> 1 {[1, 2, 3]: 10}
#
# TypeError: unhashable type: 'list'
```

Коллекции. Распаковка

```
In : c = "abcd" # Работает не только со строками
In : a, b, e, d = c
In : a
Out: 'a'

In : a, *mid, d = c
In: mid
Out: ['b', 'c']
```

Коллекции. Распаковка

```
In : [*range(4), 4]
Out: [0, 1, 2, 3, 4]

In: {'x': 1, **{'y': 2}}
Out:{'x': 1, 'y': 2}
```

Выражения. if

```
y = int(input())

if y > 20: # пробелы вокруг условия не ставятся
    print(y) # обязательные отступы в 4 пробела
elif y == 0:
    print("c:")
else:
    print(":c")

x = 5 if p > q else 3
print(x)
```

Выражения. while. for

```
a, b = 0, 1
while b < 60:
    print(b)
    a, b = b, a + b

vals = ['It', 'is an', 'interesting', 'lecture']
for x in vals:
    print(x, end=" ")
print()

for i, v in enumerate(vals): # если нужно пронумеровать
    print(f'{i}: {v}')

for i in range(10):  # диапазон range(0, 10, 3) # задаем шаг
    print(i)
```

Выражения. while. for

```
circus = {"lion" : 4, "hippo" : "foo", "giraffe" : 2}

for key, value in circus.items():
    pass

for key in circus.keys():
    pass
```

Функции.

```
def dostuff(names):
    """This docstring for pretty useless function dostuff.
    Function takes list of names and always appends list `[1, 2,
3]` to the end
    """
    names.append([1, 2, 3])
    return names

names = ['Katya', 'Kolya', 'Vitya']
names.append('Sveta')
print(names)

dostuff(names)
print(names)
```

Анонимные функции

```
lambda arguments: expression
# аналогично
def <lambda>(arguments):
    return expression

In : (lambda x, y: x - y)(5, 10)
Out: -5
```

Выражения. Особенности

Есть понятие ложно- и истинноподобных значений.

Ложными считаются:

- `None`
- числа, равные `0`: `0`, `0.`, `0+0j`
- пустые строки: '', ""
- пустые коллекции: `[]`, `()`, `{}`, `set()`

Истинными

- ненулевые числа
- непустые строки и коллекции

Выражения. Особенности

```
def print_first_element(elements):
    if elements:
        print(f'Passed elems: {elements}')

def book(dates=None):
    dates = dates or ["10 January", "20 March"]
    ...
```

Выражения. Особенности

```
a = 4
b = 9
for i in range(2, min(a, b) + 1):
    print('Testing', i)
    if a % i == 0 and b % i == 0:
        print('Not coprime')
        break
else:
    print('Coprime')
```

Выражения. Особенности

```
def coprime(a, b):
    for i in range(min(a, b) + 1):
        if a % i == 0 and b % i == 0:
            return True
    return False

a = 4
b = 9
if not coprime(a, b):
    print('Not coprime')
else:
    print('Coprime')
```

Выражения. Особенности

```
animals = {'lion': 130, 'giraffe': 800, 'hippo': 1300}
name = input()

weight = animals.get(name, 0)
if weight > 500:
    print(weight)
```

Выражения. Особенности

```
animals = {'lion': 130, 'giraffe': 800, 'hippo': 1300}
name = input()

if (weight := animals.get(name, 0)) > 500: # с версии 3.8
    print(weight)
```