ЛЕКЦИЯ №4 ПО ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Москва, 2020

Почему Python

- Автоматизация простых рутинных задач
- Научные приложения
- Обычные desktop-приложения
- Android
- Web (Django и Flask фреймворки)
- Машинное обучение (библиотека Tenser Flow)

Почему Python

- Краткий и элегантный синтаксис
- Множество доступных бесплатных библиотек (open source)
- Легко изучаем
- Невероятно популярен среди серьёзных компаний

Python технически

- Язык программирования общего назначения
- Строго типизированный
- Динамический
- Интерпретируемый
  (предварительно компилируется в байт-код)

- Мультипарадигмальный
- CPython – основная реализация Python

Питон 3

- Python 3 это будущее
- Python 2 и Python 3 частично несовместимы
- Поддержка Python 2 полностью прекратится в 2020 году
- Знаешь Python 3? Быстро перейдёшь и на Python 2!
- Python 3 работает с Unicode по умолчанию
- Большинство библиотек переписаны на Python 3

Notebook-оболочка

- Поддерживает markdown и визуализации
- Легко создавать документированный код
- Удобный режим REPL (read-evaluate-print loop)
- Идеально для изучения ЯП (языка программирования)
- Jupyter Notebook – самый популярный
- Расширение файлов - .ipynb

Типы данных

Имя	Тип	Описание
Целые числа	int	1, 2, 3
Числа с плавающей точкой	float	1.1, 2.4, 3.0
Строки	str	последовательность символов: "Hello", 'hello', "42"
Списки	list	последовательность объектов: `[1, 2.0, "hello"]`
Словари	dict	Список пар «ключ-значение»: `{"john" : "+1-23-45", "bob" : "+1-32-65"}`
Кортежи	tuple	Последовательность неизменяемых объектов: `(1, 2.0, "hello")`
Множества	set	последовательность уникальных объектов: `{"a", "b"}`
Булевые значения	bool	логические значения: True или False

Len and Count

```
In [6]: x="hello, my name is Elias"

In [7]: len(x)
Out[7]: 23

In [8]: x[len(x)-1]
Out[8]: 's'

In [9]: x.count('l')
Out[9]: 3
```

Casing

In [10]:
```
x.capitalize()
```

Out[10]:
```
'Hello, my name is elias'
```

In [14]:
```
upper_cased=x.upper()
print(upper_cased)
```

```
HELLO, MY NAME IS ELIAS
```

In [18]:
```
lower_cased=upper_cased.lower()
print(lower_cased)
```

```
hello, my name is elias
```

In [22]:
```
print(upper_cased.isupper())
print(lower_cased.islower())
print(x.isupper())
print(x.islower())
```

In [1]:
```
print("My name is {}".format("Elias"))
```

```
My name is Elias
```

In [2]:
```
print("My name is {} and I'm {}".format("Elias", "30"))
```

```
Me name is Elias and I'm 30
```

In [3]:
```
print("My name is {0} and I'm {1}".format("Elias", "30"))
```

```
Me name is Elias and I'm 30
```

In [4]:
```
print("My name is {name} and I'm {age}".format(name="Elias", age="30"))
```

```
Me name is Elias and I'm 30
```

In [6]:
```
pi=3.1415
pi
```

Out[6]:
```
3.1415
```

In [13]:
```
print("Pi equals {pi:1.2f}".format(pi=pi))
```

```
Pi equals 3.14
```

2. Инструкции управления циклом

После тела цикла можно написать слово `else:` и после него блок операций, который будет выполнен *один раз* после окончания цикла, когда проверяемое условие станет неверно:

запустить выполнить пошагово

```
1  i = 1
2  while i <= 10:
3      print(i)
4      i += 1
5  else:
6      print('Цикл окончен, i =', i)
7
```

Казалось бы, никакого смысла в этом нет, ведь эту же инструкцию можно просто написать *после* окончания цикла. Смысл появляется только вместе с инструкцией `break`. Если во время выполнения Питон встречает инструкцию `break` внутри цикла, то он сразу же прекращает выполнение этого цикла и выходит из него. При этом ветка `else` исполняться не будет. Разумеется, инструкцию `break` осмысленно вызывать только внутри инструкции `if`, то есть она должна выполняться только при выполнении какого-то особенного условия.

```
a = int(input())
while a != 0:
    if a < 0:
        print('Встретилось отрицательное число', a)
        break
    a = int(input())
else:
    print('Ни одного отрицательного числа не встретилось')

```

Во втором варианте программы сначала на вход подается количество элементов последовательности, а затем и сами элементы. В таком случае удобно воспользоваться циклом `for`. Цикл `for` также может иметь ветку `else` и содержать инструкции `break` внутри себя.

запустить выполнить пошагово

```
n = int(input())
for i in range(n):
    a = int(input())
    if a < 0:
        print('Встретилось отрицательное число', a)
        break
else:
    print('Ни одного отрицательного числа не встретилось')

```

```
a = int(input())
while a != 0:
    if a < 0:
        print('Встретилось отрицательное число', a)
        break
    a = int(input())
else:
    print('Ни одного отрицательного числа не встретилось')

```

Во втором варианте программы сначала на вход подается количество элементов последовательности, а затем и сами элементы. В таком случае удобно воспользоваться циклом `for`. Цикл `for` также может иметь ветку `else` и содержать инструкции `break` внутри себя.

запустить выполнить пошагово

```
n = int(input())
for i in range(n):
    a = int(input())
    if a < 0:
        print('Встретилось отрицательное число', a)
        break
else:
    print('Ни одного отрицательного числа не встретилось')

```