WEB-ПРОГРАММИРОВАНИЕ
ЛЕКЦИЯ 4. PYTHON 3 (ЧАСТЬ 1)

АСИСТ. КАФ. 308 ТРУТНЕВА НАДЕЖДА ВЛАДИМИРОВНА

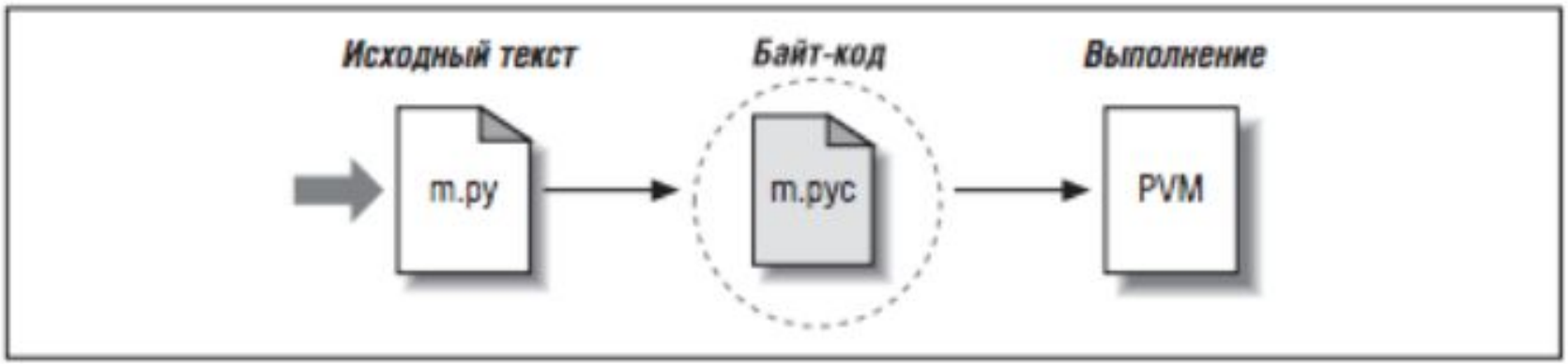
ТЕЛ: **8-926-880-12-76**

ПОЧТА: **NTRUTN@GMAIL.COM**

PYTHON 3

Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода.

Python – интерпретируемый язык программирования.



ПРОГРАММА

Программа на языке Python может состоять из одного или нескольких модулей. Каждый модуль представляет собой текстовый файл в кодировке, совместимой с 7-битной кодировкой ASCII. Для кодировок, использующих старший бит, необходимо явно указывать название кодировки. Например, модуль, комментарии или строковые литералы которого записаны в кодировке KOI8-R, должен иметь в первой или второй строке следующую спецификацию:

```
# -*- coding: utf8 -*-
```

Программа на Python, с точки зрения интерпретатора, состоит из логических строк. Одна логическая строка, как правило, располагается в одной физической, но длинные логические строки можно явно (с помощью обратной косой черты) или неявно (внутри скобок) разбить на несколько физических:
(синтаксис python 2.7)

```
print a,  " - очень длинная строка, которая не помещается в", \
   80, "знакоместах"
```

ИМЕНА

Имя может начинаться с латинской буквы (любого регистра) или подчеркивания, а дальше допустимо использование цифр. В качестве идентификаторов нельзя применять ключевые слова языка и нежелательно переопределять встроенные имена.

```
>>> import keyword
>>> keyword.kwlist
['and', 'assert', 'break', 'class', 'continue', 'def', 'del',
'elif', 'else', 'except', 'exec', 'finally', 'for', 'from',
'global', 'if', 'import', 'in', 'is', 'lambda', 'not', 'or',
'pass', 'print', 'raise', 'return', 'try', 'while', 'yield']
```

Имена, начинающиеся с подчеркивания или двух подчеркиваний, имеют особый смысл. Одиночное подчеркивание говорит программисту о том, что имя имеет местное применение, и не должно использоваться за пределами модуля. Двойным подчеркиванием в начале и в конце обычно наделяются специальные имена.

ИМЕНА

В каждой точке программы интерпретатор "видит" три пространства имен:

- локальное,
- глобальное
- встроенное

Пространство имен - отображение из имен в объекты.

Локальные имена - имена, которым присвоено значение в данном блоке кода. Глобальные имена - имена, определяемые на уровне блока кода определения модуля или те, которые явно заданы в операторе global. Встроенные имена - имена из специального словаря __ builtins __.

Области видимости имен могут быть вложенными друг в друга, например, внутри вызванной функции видны имена, определенные в вызывающем коде. Переменные, которые используются в блоке кода, но связаны со значением вне кода, называются свободными переменными.

Так как переменную можно связать с объектом в любом месте блока, важно, чтобы это произошло до ее использования, иначе будет возбуждено исключение NameError. Связывание имен со значениями происходит в операторах присваивания, from, import, в формальных аргументах функций, при определении функции или класса, во втором параметре части except оператора try-except.

ЦЕЛЫЕ ЧИСЛА

- обычное целое int
- целое произвольной точности long
- логический bool

x + y	Сложение
x - y	Вычитание
x * y	Умножение
x / y	Деление
x // y	Получение целой части от деления
x % y	Остаток от деления
-x	Смена знака числа
abs(x)	Модуль числа
divmod(x, y)	Пара (x // y, x % y)
x ** y	Возведение в степень
pow(x, y[, z])	x^y по модулю (если модуль задан)

ЦЕЛЫЕ ЧИСЛА

Битовые операции

Над целыми числами также можно производить битовые операции

x | y	Побитовое *или*
x ^ y	Побитовое *исключающее или*
x & y	Побитовое *и*
x << n	Битовый сдвиг влево
x >> y	Битовый сдвиг вправо
~x	Инверсия битов

СИСТЕМЫ СЧИСЛЕНИЯ

• int([object], [основание системы счисления]) - преобразование к целому числу в деся-тичной системе счисления. По умолчанию система счисления десятичная, но можно задать любое основание от 2 до 36 включительно.
• bin(x) - преобразование целого числа в двоичную строку.
• hex(x) - преобразование целого числа в шестнадцатеричную строку.
• oct(x) - преобразование целого числа в восьмеричную строку

СИСТЕМЫ СЧИСЛЕНИЯ

```
>>> a = int('19') # Переводим строку в число
>>> b = int('19.5')  # Строка не является целым числом
Traceback (most recent call last):
  File "", line 1, in
ValueError: invalid literal for int() with base 10: '19.5'
>>> c = int(19.5)  # Применённая к числу с плавающей точкой, отсекает дробную часть
>>> print(a, c)
19 19
>>> bin(19)
'0b10011'
>>> oct(19)
'0o23'
>>> hex(19)
'0x13'
>>> 0b10011  # Так тоже можно записывать числовые константы
19
>>> int('10011', 2)
```

ВЕЩЕСТВЕННЫЕ И КОМПЛЕКСНЫЕ ЧИСЛА

Тип float

```
>>> pi = 3.1415926535897931
>>> pi ** 40
7.6912142205156999e+19
```

Кроме арифметических операций, можно использовать операции из модуля math.

Тип complex

Литерал мнимой части задается добавлением j в качестве суффикса (перемножаются мнимые единицы):

```
>>> -1j * -1j
(-1-0j)
```

Тип реализован на базе вещественного. Кроме арифметических операций, можно использовать операции из модуля cmath.

ПОСЛЕДОВАТЕЛЬНОСТИ

Последовательности бывают изменчивыми и неизменчивыми.

Синтаксис	Семантика
`len(s)`	Длина последовательности `s`
`x in s`	Проверка принадлежности элемента последовательности. В новых версиях Python можно проверять принадлежность подстроки строке. Возвращает `True` или `False`
`x not in s`	`= not x in s`
`s + s1`	Конкатенация последовательностей
`s*n` или `n*s`	Последовательность из `n` раз повторенной `s`. Если `n < 0`, возвращается пустая последовательность.
`s[i]`	Возвращает `i` -й элемент `s` или `len(s)+i` -й, если `i < 0`
`s[i:j:d]`	Срез из последовательности `s` от `i` до `j` с шагом `d` будет рассматриваться ниже
`min(s)`	Наименьший элемент `s`
`max(s)`	Наибольший элемент `s`

Дополнительные конструкции для изменчивых последовательностей:

`s[i] = x`	`i` -й элемент списка `s` заменяется на `x`
`s[i:j:d] = t`	Срез от `i` до `j` (с шагом `d`) заменяется на (список) `t`
`del s[i:j:d]`	Удаление элементов среза из последовательности

ПОСЛЕДОВАТЕЛЬНОСТИ (СРЕЗЫ)

последовательность[нач:кон:шаг]

где нач - промежуток начала среза, кон - конца среза, шаг - шаг. По умолчанию нач=0, кон=len(последовательность), шаг=1, если шаг не указан, второе двоеточие можно опустить.

```
>>> s = range(10)
>>> s
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> s[0:3]
[0, 1, 2]
>>> s[-1:]
[9]
>>> s[::3]
[0, 3, 6, 9]
>>> s[0:0] = [-1, -1, -1]
>>> s
[-1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> del s[:3]
>>> s
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

СТРОКИ

S = 'spam"s'

S = "spam's"

- Конкатенация (сложение)

```
>>> S1 = 'spam'
>>> S2 = 'eggs'
>>> print(S1 + S2)
'spameggs'
```

- Дублирование строки

```
>>> print('spam' * 3)
spamspamspam
```

- Длина строки (функция len)

```
>>> len('spam')
4
```

- Доступ по индексу

```
>>> S = 'spam'
>>> S[0]
's'
>>> S[2]
'a'
>>> S[-2]
'a'
```

СТРОКИ (МЕТОД **FORMAT**)

```
>>> 'Hello, {}!'.format('Vasya')
'Hello, Vasya!'
```

```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> '{}, {}, {}'.format('a', 'b', 'c')
'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')
'c, b, a'
>>> '{2}, {1}, {0}'.format(*'abc')
'c, b, a'
>>> '{0}{1}{0}'.format('abra', 'cad')
'abracadabra'
>>> 'Coordinates: {latitude}, {longitude}'.format(latitude='37.24N', longitude='-
↪115.81W')
'Coordinates: 37.24N, -115.81W'
>>> coord = {'latitude': '37.24N', 'longitude': '-115.81W'}
>>> 'Coordinates: {latitude}, {longitude}'.format(**coord)
'Coordinates: 37.24N, -115.81W'
```

СПИСОК **(LIST)**

```
>>> list('список')
['с', 'п', 'и', 'с', 'о', 'к']
```

```
>>> s = []  # Пустой список
>>> l = ['s', 'p', ['isok'], 2]
>>> s
[]
>>> l
['s', 'p', ['isok'], 2]
```

Создание списка с помощью генератора списков:

```
>>> c = [c * 3 for c in 'list']
>>> c
['lll', 'iii', 'sss', 'ttt']
```

```
>>> c = [c * 3 for c in 'list' if c != 'i']
>>> c
['lll', 'sss', 'ttt']
>>> c = [c + d for c in 'list' if c != 'i' for d in 'spam' if d != 'a']
>>> c
['ls', 'lp', 'lm', 'ss', 'sp', 'sm', 'ts', 'tp', 'tm']
```

МЕТОДЫ СПИСКОВ

Метод	Описание
`.append(x)`	Добавляет элемент в конец последовательности
`.count(x)`	Считает количество элементов, равных `x`
`.extend(s)`	Добавляет к концу последовательности последовательность `s`
`.index(x)`	Возвращает наименьшее `i`, такое, что `s[i] == x`. Возбуждает исключение `ValueError`, если `x` не найден в `s`
`.insert(i, x)`	Вставляет элемент `x` в `i` -й промежуток
`.pop(i)`	Возвращает `i` -й элемент, удаляя его из последовательности
`.reverse(s)`	Меняет порядок элементов `s` на обратный
`.sort([cmpfunc])`	Сортирует элементы `s`. Может быть указана своя *функция сравнения* `cmpfunc`

КОРТЕЖ (TUPLE)

Для представления константной последовательности (разнородных) объектов используется тип кортеж. Литерал кортежа обычно записывается в круглых скобках, но можно, если не возникают неоднозначности, писать и без них.

```
p = (1.2, 3.4, 0.9)   # точка в трехмерном пространстве
for s in "one", "two", "three":  # цикл по значениям кортежа
  print s
one_item = (1,)
empty = ()
p1 = 1, 3, 9   # без скобок
p2 = 3, 8, 5,  # запятая в конце игнорируется
```

Использовать синтаксис кортежей можно и в левой части оператора присваивания. В этом случае на основе вычисленных справа значений формируется кортеж и связывается один в один с именами в левой части.

```
a, b = b, a
```

СЛОВАРЬ (DICT)

Словарь (хэш, ассоциативный массив) - это изменчивая структура данных для хранения пар ключ-значение, где значение однозначно определяется ключом. В качестве ключа может выступать неизменчивый тип данных (число, строка, кортеж и т.п.). Порядок пар ключ-значение произволен.

```
d = {1: 'one', 2: 'two', 3: 'three', 4: 'four'}
d0 = {0: 'zero'}
print d[1]        # берется значение по ключу
d0[0] = 0          # присваивается значение по ключу
del d0[0]          # удаляется пара ключ-значение с данным ключом
print d
for key, val in d.items():   # цикл по всему словарю
  print key, val
for key in d.keys():   # цикл по ключам словаря
  print key, d[key]
for val in d.values():   # цикл по значениям словаря
  print val
d.update(d0)  # пополняется словарь из другого
print len(d)  # количество пар в словаре
```

СЛОВАРЬ (DICT)

Генератор словарей

```
>>> d = {a: a ** 2 for a in range(7)}
>>> d
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

МНОЖЕСТВО (SET)

Множество в python - "контейнер", содержащий не повторяющиеся элементы в случайном порядке.

```
>>> a = set()
>>> a
set()
>>> a = set('hello')
>>> a
{'h', 'o', 'l', 'e'}
>>> a = {'a', 'b', 'c', 'd'}
>>> a
{'b', 'c', 'a', 'd'}
>>> a = {i ** 2 for i in range(10)} # генератор множеств
>>> a
{0, 1, 4, 81, 64, 9, 16, 49, 25, 36}
>>> a = {}  # А так нельзя!
>>> type(a)
<class 'dict'>
```

МНОЖЕСТВО **(SET)**

```
>>> words = ['hello', 'daddy', 'hello', 'mum']
>>> set(words)
{'hello', 'daddy', 'mum'}
```

ФАЙЛЫ

Объекты этого типа предназначены для работы с внешними данными. В простом случае - это файл на диске. Файловые объекты должны поддерживать основные методы: read(), write(), readline(), readlines(), seek(), tell(), close() и т.п.

Следующий пример показывает копирование файла:

```
f1 = open("file1.txt", "r")
f2 = open("file2.txt", "w")
for line in f1.readlines():
  f2.write(line)
f2.close()
f1.close()
```

Чаще используют библиотеку urllib

УСЛОВИЯ

```
if test1:
    state1
elif test2:
    state2
else:
    state3
```

```
a = int(input())
if a < -5:
    print('Low')
elif -5 <= a <= 5:
    print('Mid')
else:
    print('High')
```

УСЛОВИЯ

Проверка истинности в Python

- Любое число, не равное 0, или непустой объект - истина.
- Числа, равные 0, пустые объекты и значение None - ложь
- Операции сравнения применяются к структурам данных рекурсивно
- Операции сравнения возвращают True или False
- Логические операторы and и or возвращают истинный или ложный объект-операнд

Логические операторы:

```
X and Y
```

Истина, если оба значения X и Y истинны.

```
X or Y
```

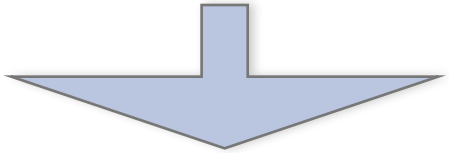
Истина, если хотя бы одно из значений X или Y истинно.

```
not X
```

Истина, если X ложно.

УСЛОВИЯ

```
if X:
    A = Y
else:
    A = Z
```



```
A = Y if X else Z
```

```
>>> A = 't' if 'spam' else 'f'
>>> A
't'
```

ЦИКЛ **WHILE**

While - один из самых универсальных циклов в Python, поэтому довольно медленный. Выполняет тело цикла до тех пор, пока условие цикла истинно.

```
>>> i = 5
>>> while i < 15:
...     print(i)
...     i = i + 2
...
5
7
9
11
13
```

```
f = open("file.txt", "r")
while f:
  l = f.readline()
  if not l:
    break
  if len(l) > 5:
    print l
f.close()
```

ЦИКЛ **FOR**

```
>>> for i in 'hello world':
...     print(i * 2, end='')
...
hheellllloo  wwoorrlldd
```

```
for i in range(1, 10):
  for j in range(1, 10):
    print("%2i" % (i*j))
```

ПРЕРЫВАНИЕ

Оператор continue

Оператор continue начинает следующий проход цикла, минуя оставшееся тело цикла (for или while)

```
>>> for i in 'hello world':
...     if i == 'o':
...         continue
...     print(i * 2, end='')
...
hheellll  wwrrlldd
```

Оператор break

Оператор break досрочно прерывает цикл.

```
>>> for i in 'hello world':
...     if i == 'o':
...         break
...     print(i * 2, end='')
...
hheellll
```

ПРЕРЫВАНИЕ

Волшебное слово else

Слово else, примененное в цикле for или while, проверяет, был ли произведен выход из цикла инструкцией break, или же “естественным” образом. Блок инструкций внутри else выполнится только в том случае, если выход из цикла произошел без помощи break.

```
>>> for i in 'hello world':
...     if i == 'a':
...         break
... else:
...     print('Буквы a в строке нет')
...
Буквы a в строке нет
```

СПАСИБО ЗА ВНИМАНИЕ !

ВОПРОСЫ???