Программирование на Python

лекция 3

Владимир Юрьевич Полищук

Инженерная школа
информационных технологий и
робототехники,
Томский политехнический университет

pvy3@tpu.ru

Содержание

- инструкции языка Python
- инструкции циклов *while* и *for*
- инструкции *break* и *continue*
- функции *range*, *zip* и *enumerate*

Структура программы на языке Python

1. Программы делятся на модули.
2. Модули содержат инструкции.
3. *Инструкции состоят из выражений.*
4. Выражения создают и обрабатывают объекты.

Условная инструкция

Условная инструкция на С-подобном языке:

```
if (x > y) {
    x = 1;
    y = 2;
}
```

Условная инструкция на Python:

```
if x > y:
     x = 1
     y = 2
```

Что добавляет язык Python

Основная инструкция:

Вложенный блок инструкций

Что Python устраняет

Круглые скобки не обязательны: *if x < y*

Конец строки является концом инструкции

x = 1

Конец отступа – это конец блока

```
if x > y:
    x = 1
    y = 2
```

Синтаксическое правило:
все инструкции в пределах одного блока должны иметь один и тот же отступ от левого края.

Специальные случаи оформления инструкций

```
a = 1; b = 2; print(a + b) # Три инструкции на одной строке

list = [111,
        222,
         333]


 X = (A + B +
      C + D)
```

Специальные случаи оформления инструкций

```
if (A == 1 and
    B == 2 and
    C == 3):
        print('word' * 3)

X = (A + B + \  # Альтернативный способ, который
        C + D)  может быть источником ошибок
```

Специальный случай оформления блока

```
if x > y: print(x)
```

Размещайте инструкции в отдельных строках и всегда оформляйте отступы для вложенных блоков и ваш программный код будет проще читать и подвергать изменениям.

Инструкции присваивания

- Инструкция присваивания создает ссылку на объект.
- Переменные создаются при первом присваивании.
- Прежде чем переменную можно будет использовать, ей должно быть присвоено значение.
- Некоторые инструкции неявно тоже выполняют операцию присваивания.

Формы инструкции присваивания

1. word = ‘word’ # Каноническая форма
2. word, upword = 'lord', 'LORD' # Присваивание кортежей (позиционное)
3. [word, upword] = ['lord','LORD'] # Присваивание списков (позиционное)
4. a, b, c, d = ‘word’ #Присваивание последовательностей, обобщенное
5. a, *b = ‘word’ # Расширенная операция распаковывания последовательностей
6. word = upword = ‘apple' # Групповое присваивание одного значения
7. words += 42 # Комбинированная инструкция присваивания (эквивалентно инструкции words = words + 42)

Присваивание последовательностей

```
>>> value = 1
>>> number = 2
>>> A, B = value, number # Присваивание кортежей
                        # Что равносильно A = value; B = number
>>> A, B
 (1, 2)
>>> [C, D] = [value, number] # Присваивание списков
>>> C, D
(1, 2)
```

Присваивание последовательностей

```
>>> value = 1
>>> number = 2
>>> value, number = number, value # Кортежи: обмен значениями, то же, что и (T = value; value = number; number = T)
>>> value, number
 (2, 1)
```

Присваивание последовательностей

```
>>> [a, b, c] = (1, 2, 3) # Кортеж значений
                          присваивается списку
                          переменных
>>> a, c
(1, 3)
>>> (a, b, c) = "ABC" # Строка символов
                      присваивается кортежу
                      переменных
>>> a, c
('A', 'C')
```

Дополнительные варианты инструкции присваивания последовательностей

```
>>> string = 'word'
>>> a,b,c,d = string # Одинаковое число элементов с обеих сторон
>>> a,d
('w', 'd')
>>> a,b,c = string #  В противном случае выводится сообщение об ошибке
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    a,b,c = string
ValueError: too many values to unpack (expected 3)
```

Дополнительные варианты инструкции присваивания последовательностей

```
>>> string = 'word'
>>> a, b, c = string[0], string[1], string[2:] # Элементы и срез
>>> a, b, c
('w', ‘o', 'rd')
>>> a, b, c = list(string[:2]) + [string[2:]] # Срезы и конкатенация
>>> a, b, c
('w’, 'o', 'rd')

>>> a, b = string[:2]                    # То же самое, только проще
>>> c = string[2: ]
>>> a, b, c
('w', ‘o’, 'rd')

>>> ((a,b),c) = ('wo','rd') # связывание производится в соответствии с формой и местоположением
>>> a,b,c
('w', 'o', 'rd')
```

Дополнительные варианты инструкции присваивания последовательностей

```
>>> red, green, blue = range(3)
>>> red, green, blue
(0, 1, 2)
>>> list(range(3))
[0, 1, 2]
```

```
>>> L = [1, 2, 3, 4]
>>> while L:
        first, L=L[0],L[1:]
        print(first,L)
1 [2, 3, 4]
2 [3, 4]
3 [4]
4 []
```

Расширенная операция распаковывания последовательностей

```
>>> seq = [1, 2, 3, 4]
>>> a, b, c, d = seq
>>> print(a, b, c, d)
1 2 3 4
>>> a,b =seq
Traceback (most recent call last):
  File "<pyshell#86>", line 1, in <module>
    a,b =seq
ValueError: too many values to unpack (expected 2)
```

Расширенная операция распаковывания последовательностей

```
>>> a, *b = seq
>>> a
1
>>> b
 [2, 3, 4]
```

```
>>> *a, b = seq
>>> a
[1, 2, 3]
 >>> b
4
```

```
 >>> a, *b, c = seq
>>> a
1
 >>> b
 [2, 3]
>>> c
4
```

Расширенная операция распаковывания последовательностей

```
>>> a, b, *c = seq
>>> a
1
>>> b
2
>>> c
 [3, 4]
```

```
>>> a, *b = ‘word’
>>> a, b
(‘w’, [‘o’, ‘r’, ‘d’])
>>> a, *b, c = ‘word’
>>> a, b, c
(‘w’, [‘o’, ‘r’], ‘d’)
```

Расширенная операция распаковывания последовательностей

```
>>> S = 'word'
>>> S[0], S[1:] # Тип среза зависит от типа исходной
                                последовательности,
('w', 'ord')  # расширенная операция распаковывания
                всегда возвращает список
>>> S[0], S[1:3], S[3]
('w', 'or', 'd')
```

Расширенная операция распаковывания последовательностей

```
>>> L = [1, 2, 3, 4]
>>> while L:
        first, L=L[0],L[1:]
        print(first,L)
1 [2, 3, 4]
2 [3, 4]
3 [4]
4 []
```

```
>>> L = [1, 2, 3, 4]
>>> while L:
        first, *L = L # Получить первый и остальные  элементы
        print(first, L) # без операции извлечения среза
 1 [2, 3, 4]
2 [3, 4]
 3 [4]
4 []
```

Граничные случаи

```
>>> seq
[1, 2, 3, 4]
>>> a, b, c, *d = seq
>>> print(a, b, c, d)
1 2 3 [4]
```

Граничные случаи

```
>>> a, b, c, d, *e = seq
>>> print(a, b, c, d, e)
1 2 3 4 []
>>> a, b, *e, c, d = seq
>>> print(a, b, c, d, e)
1 2 3 4 []
```

Граничные случаи

```
>>> a,*b,c,*d = seq
SyntaxError: two starred expressions in assignment
>>> a,b = seq
Traceback (most recent call last):
  File "<pyshell#105>", line 1, in <module>
    a,b =seq
ValueError: too many values to unpack (expected 2)

>>> *a = seq
SyntaxError: starred assignment target must be in a list or tuple
>>> *a, = seq
>>> a
[1, 2, 3, 4]
```

Полезное удобство

```
>>> seq
[1, 2, 3, 4]
>>>a, *b = seq
>>>a, b # Первый, остаток
(1, [2, 3, 4])
>>>a, b = seq[0], seq[1:]
>>>a, b # Первый, остаток:
        традиционная реализация
(1, [2, 3, 4])
```

Полезное удобство

```
>>> *a, b = seq # Остаток, последний
>>> a, b
 ([1, 2, 3], 4)
>>> a, b = seq[:-1], seq[-1] # Остаток,
        последний: традиционная реализация
>>> a, b
([1, 2, 3], 4)
```

Групповое присваивание и разделяемые ссылки

```
>>> a = b = 0
>>> b = b + 1
>>> a, b
 (0, 1)

>>> a = b = []
>>> b.append(42)
>>> a, b
([42], [42])
```

```
>>> a = []
>>> b = []
>>> b.append(42)
>>> a, b
([], [42])
```

Цикл while

```
while<test>:          # Условное выражение test
    <statements1>  # Тело цикла
else:                 # Необязательная часть else
    <statements2> # Выполняется, если выход из
    цикла производится не инструкцией break
```

Цикл while

```
>>> while True:
        print('Нажми Ctrl-C чтобы остановить меня!')

>>> x = 'word'
>>>while x: # Пока x не пустая строка
                    # эквивалент while x != '':
            print(x, end=' ')
            x = x[1:] # Вырезать первый символ из x
word ord rd d
```

Цикл while

```
>>> a=0; b=10
>>> while a < b:      # Один из способов
                      организации циклов перечисления
        print(a, end=' ')
        a += 1        # Или, a = a + 1
0 1 2 3 4 5 6 7 8 9
```

break, continue, pass u else

break

Производит переход за пределы объемлющего цикла, всей инструкции цикла.

continue

Производит переход в начало цикла, в строку заголовка.

pass

Ничего не делает: это пустая инструкция, используемая как заполнитель.

else

Выполняется, только если цикл завершился обычным образом, без использования инструкции break.

Общий формат цикла

while <test1>:

    <statements1>

    if <test2>: **break** # Выйти из цикла, пропустив часть else

    if<test3>:**continue** # Перейти в начало цикла, к выражению test1

else:

    <statements2> # Выполняется, если не была использована инструкция ‘break’

while 1: **pass** # Нажмите Ctrl-C, чтобы прервать цикл! *pass* в мире инструкций – это то же, что *None* в мире объектов, – явное ничто

def *func1():* ***pass*** # Реализация функции будет добавлена позже

!!

```
def func1():
        ...     # Альтернатива инструкции pass (Python 3)
        «TBD» (To Be Done – подлежит реализации)
func1() # При вызове не выполняет никаких действий

>>> X = ... # Альтернатива объекту None
>>> X
Ellipsis
```

Инструкция continue

```
x = 10
while x:
        x = x-1 # Или, x -= 1
        if x % 2 !=0:  continue # Нечетное? – пропустить вывод
         print(x, end=’ ‘)
```

8 6 4 2 0

Инструкция break

```
>>> while 1:
        name = input('Enter name:')
        if name == 'stop': break
```

Инструкция break и else

```
>>> x = y // 2    # Для значений y > 1
>>> while x > 1:
            if y % x == 0:  # Остаток
                print(y, ‘ имеет делитель’, x)
                break  # Перешагнуть блок else
            x -= 1
    else:  # Нормальное завершение цикла
            print(y, ‘простое число’)
```

Цикл for

```
for <target> in <object>: # Связывает элементы
                            объекта с переменной цикл
    <statements> # Повторяющееся тело цикла:
                   использует переменную цикла
else:
    <statements> # Если не попали на инструкцию
                   'break'
```

Цикл for

```
for <target> in <object>: # Присваивает элементы
                            объекта с переменной цикла
    <statements>
    if <test>: break      # Выход из цикла, минуя блок else
    if <test>: continue   # Переход в начало цикла
else:
    <statements>          # Если не была вызвана инструкция
                            'break'
```

Цикл for

```
>>> for x in [‘word’, ‘string’, ‘text’]:
        print(x, end=’ ‘)
word string text

>>> sum = 0
>>> for x in [1, 2, 3, 4]:
      sum = sum + x
>>> sum
10

>>> prod = 1
>>> for item in [1, 2, 3, 4]:
        prod *= item
>>> prod
24
```

Цикл for

```
>>> S = ‘string’
>>>for x in S: print(x, end=’ ‘) # Обход строки
s t r i n g

>>> T = {‘one’, ‘two’, ‘three’}
>>>for x in T: print(x, end=’ ‘)   # Обход элементов кортежа
one two three
```

Цикл for

```
>>> T = [(1, 2), (3, 4), (5, 6)]
>>>for (a, b) in T: # Операция присваивания кортежа
        print(a, b)  # в действии
1 2
3 4
5 6
```

Цикл for

```
>>> D = {‘a’: 1, ‘b’: 2, ‘c’: 3}
>>> for key in D:
        print(key, ‘=>’, D[key]) # Используется
                                 итератор словаря и
                                 операция индексирования
a => 1
c => 3
b => 2
```

Цикл for

```
>>> list(D.items())
[('a', 1), ('c', 3), ('b', 2)]
>>> for (key, value) in D.items():
        print(key, '=>', value) # Обход ключей и
                                значений одновременно
a => 1
c => 3
b => 2
```

Цикл for

```
>>> T
[(1, 2), (3, 4), (5, 6)]
>>> for both in T:
        a, b = both # Эквивалентный вариант с
        print(a, b)  # присваиванием вручную
1 2
3 4
5 6
```

Цикл for

```
>>> ((a, b), c) = ((1, 2), 3) # Вложенные структуры также могут использоваться
>>> a, b, c
 (1, 2, 3)
>>> for ((a, b), c) in [((1, 2), 3), ((4, 5), 6)]:
        print(a, b, c)
1 2 3
4 5 6

>>> for ((a, b), c) in [([1, 2], 3), ['XY', 6]]:
        print(a, b, c)
1 2 3
X Y 6
```

Расширенная операция присваивания последовательностей

```
>>> a, *b, c = (1, 2, 3, 4) # Расширенная инструкция распаковывания последовательностей
>>> a, b, c
(1, [2, 3], 4)

>>> for (a, *b, c) in [(1, 2, 3, 4), (5, 6, 7, 8)]:
        print(a, b, c)
1 [2, 3] 4
5 [6, 7] 8
```

Вложенные циклы for

```
>>> items = [‘aaa’, 111, (4, 5), 2.01] # Множество объектов
>>> tests = [(4, 5), 3.14] # Ключи, которые требуется отыскать
>>> for key in tests: # Для всех ключей
        for item in items: # Для всех элементов
            if item == key: # Проверить совпадение
                print(key, ‘нашелся’)
                break
        else:
            print(key, ‘не найден’)
(4, 5) нашелся
3.14 не найден
```

Вложенные циклы for

```
>>> for key in tests: # Для всех ключей позволить интерпретатору отыскать совпадения
        if key in items:
            print(key, ‘нашелся’)
        else:
            print(key, ‘не найден’)
(4, 5) нашелся
3.14 не найден
```

Вложенные циклы for

```
>>> seq1 = ‘spam’
>>> seq2 = ‘scam’
>>> res = [] # Изначально список пуст
>>> for x in seq1: # Выполнить обход первой последовательности
        if x in seq2: # Общий элемент?
            res.append(x) # Добавить в конец результата
>>>res
[‘s’, ‘a’, ‘m’]
```

Чтение файлов

```
file = open('test.txt', 'r') # Прочитать содержимое файла в строку
print(file.read())

file = open('test.txt')
while True:
    char = file.read(1) # Читать по одному символу
    if not char: break
    print(char)

for char in open('test.txt').read():
    print(char)
```

Чтение файлов

```
file = open('test.txt')
while True:
    line = file.readline() # Читать строку за строкой
    if not line: break
    print(line, end=' ')

file = open('test.txt', 'rb')
while True:
    chunk = file.read(10) # Читать блоками по 10 байтов
    if not chunk: break
    print(chunk)
```

Чтение файлов

```
for line in open('test.txt').readlines():
    print(line, end='')

for line in open('test.txt'): # Использование итератора: лучший способ чтения текста
    print(line, end='')
```

Приемы программирования циклов

- Встроенная функция *range* возвращает непрерывную последовательность увеличивающихся целых чисел, которые можно использовать в качестве индексов внутри цикла *for*.

- Встроенная функция *zip* возвращает список кортежей, составленных из элементов входных списков с одинаковыми индексами, который может использоваться для одновременного обхода нескольких последовательностей в цикле *for*.

Счетные циклы: while и range

```
>>> list(range(5)), list(range(2, 5)), list(range(0, 10, 2))
([0, 1, 2, 3, 4], [2, 3, 4], [0, 2, 4, 6, 8])

>>> list(range(-5, 5))
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]
>>> list(range(5, -5, -1))
[5, 4, 3, 2, 1, 0, -1, -2, -3, -4]
```

Счетные циклы: while и range

```
>>> for i in range(3):
        print(i, 'Pythons')
0 Pythons
1 Pythons
2 Pythons

>>> X = 'word'
>>> for item in X: print(item, end=' ') # Простейший цикл
w o r d
```

Счетные циклы: while и range

```
>>> i = 0
>>>while i < len(X): # Обход с помощью цикла while
        print(X[i], end=' ')
        i += 1
w o r d
```

Счетные циклы: while и range

```
>>> X = ‘word’
>>>len(X) # Длина строки
4
>>>list(range(len(X))) # Все допустимые смещения в X
[0, 1, 2, 3]

>>> for i in range(len(X)): print(X[i], end=’ ‘) #
Извлечение элементов вручную
w o r d


>>> for item in X: print(item) # Простейшая итерация
```

Обход части последовательности: range и срезы

```
>>> S = ‘abcdefghijk’
>>> list(range(0, len(S), 2))
[0, 2, 4, 6, 8, 10]
>>> for i in range(0, len(S), 2): print(S[i], end=’ ‘)
a c e g i k

>>> S = ‘abcdefghijk’
>>> for c in S[::2]: print(c, end=’ ‘)
a c e g i k
```

Параллельный обход: zip

```
>>> L1 = [1,2,3,4]
 >>> L2 = [5,6,7,8]

>>> zip(L1, L2)
<zip object at 0x026523C8>

>>>list(zip(L1,L2))
[(1, 5), (2, 6), (3, 7), (4, 8)]
```

Параллельный обход: zip

```
>>> for (x, y) in zip(L1, L2):
        print(x, y, '--', x + y)
1 5 -- 6
2 6 -- 8
3 7 -- 10
4 8 -- 12
```

Функция zip

```
>>> T1, T2, T3 = (1,2,3), (4,5,6), (7,8,9)
>>> T3 (7, 8, 9)
>>> list(zip(T1,T2,T3))
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]

>>> S1 = ‘abc’
>>> S2 = ‘xyz123’
>>> list(zip(S1, S2))
 [(‘a’, ‘x’), (‘b’, ‘y’), (‘c’, ‘z’)]
```

Конструирование словаря с помощью функции zip

```
>>> keys = [‘spam’, ‘eggs’, ‘toast’]
>>> vals = [1, 3, 5]

>>> list(zip(keys, vals))
 [(‘spam’, 1), (‘eggs’, 3), (‘toast’, 5)]
>>> D2 = {}
>>> for (k, v) in zip(keys, vals): D2[k] = v
>>> D2
{‘toast’: 5, ‘eggs’: 3, ‘spam’: 1}
```

Конструирование словаря с помощью функции zip

```
>>> keys = [‘spam’, ‘eggs’, ‘toast’]
>>> vals = [1, 3, 5]

>>> D3 = dict(zip(keys, vals))
>>> D3 {‘toast’: 5, ‘eggs’: 3, ‘spam’: 1}
```

Генерирование индексов и элементов: enumerate

```
>>> S = ‘spam’
>>> offset = 0
>>> for item in S:
        print(item, ‘имеет смещение’, offset)
        offset += 1
```

s имеет смещение 0
p имеет смещение 1
a имеет смещение 2
m имеет смещение 3

Генерирование индексов и элементов: enumerate

```
>>> S = ‘spam’
>>> for (offset, item) in enumerate(S):
        print(item, ‘имеет смещение’, offset)
```

```
s имеет смещение 0
p имеет смещение 1
a имеет смещение 2
m имеет смещение 3
```

Генерирование индексов и элементов: enumerate

```
>>> E = enumerate(S)
>>> E
 <enumerate object at 0x02765AA8>
>>> next(E)
 (0, 's')
>>> next(E)
 (1, 'p')
>>> next(E)
 (2, 'a')

>>> [c * i for (i, c) in enumerate(S)]
['', 'p', 'aa', 'mmm']
```