КАЛУЖСКИЙ ФИЛИАЛ
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО
ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ Н.Э. БАУМАНА
(национальный исследовательский университет)»

Факультет «Информатика и управление»

Кафедра «Программное обеспечение ЭВМ, информационные технологии»

Высокоуровневое программирование

Лекция №4. «List, dict, set comprehensions»

Калуга - 2020

Циклы - for

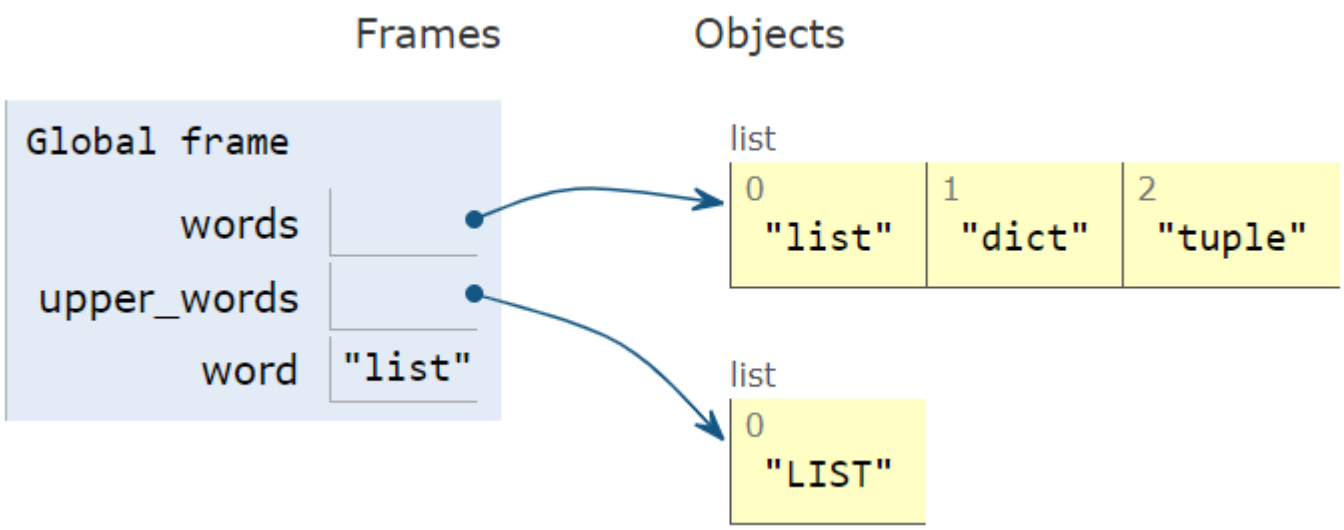
Примеры последовательностей элементов, по которым может проходиться цикл **for**:

- строка
- список
- словарь
- функция range
- любой итерируемый объект (см. выше + кортежи, файлы)

Циклы - for

```
1 words = ['list', 'dict', 'tuple']
2 upper_words = []
3 for word in words:
4     upper_words.append(word.upper())
5 upper_words
```

```
['LIST', 'DICT', 'TUPLE']
```



Циклы - for

```
# функция range возвращает неизменяемую последовательность
# чисел в виде объекта range
range(stop)
range(start, stop[, step])
# start - с какого числа начинается последовательность. По умолчанию - 0
# stop  - до какого числа продолжается последовательность чисел.
#         Указанное число не включается в диапазон
# step  - с каким шагом растут числа. По умолчанию 1
```

```
for i in range(10):
    print('number is {}'.format(i))
```

```
number is 0
number is 1
number is 2
number is 3
number is 4
number is 5
number is 6
number is 7
number is 8
number is 9
```

Циклы – for и словари

```
1  d = {
2      'Moscow': 'Russian Federation',
3      'London': 'England',
4      'Madrid': 'Spain',
5      'Paris': 'France'
6  }
7  for k in d:
8      print(k)
```

```
Moscow
London
Madrid
Paris
```

```
1  for key in d:
2      print(key + ' => ' + d[key])
```

```
Moscow => Russian Federation
London => England
Madrid => Spain
Paris => France
```

```
1  for key, value in d.items():
2      print(key + ' => ' + value)
```

```
Moscow => Russian Federation
London => England
Madrid => Spain
Paris => France
```

Циклы – while

```
choice = "y"

while choice.lower() == "y":
    print("Привет")
    choice = input("Для продолжения нажмите Y, а для
выхода любую другую клавишу: ")
print("Работа программы завешена")
```

```
1  a = 1234
2
3  while a:
4      print(a % 10, end=' ')
5      a //= 10
```

```
4 3 2 1
```

Оператор break

Оператор **break** позволяет досрочно прервать цикл:

- **break** прерывает текущий цикл и продолжает выполнение следующих выражений
- если используется несколько вложенных циклов, **break** прерывает внутренний цикл и продолжает выполнять выражения, следующие за блоком **break** может использоваться в циклах **for** и **while**

Оператор break

```
1  for num in range(10):
2      if num < 7:
3          print(num)
4      else:
5          break
```

```
0
1
2
3
4
5
6
```

```
1  i = 0
2  while i < 10:
3      if i == 5:
4          break
5      else:
6          print(i)
7          i += 1
```

```
0
1
2
3
4
```

Оператор continue

Оператор **continue** возвращает управление в начало цикла. То есть, **continue** позволяет «перепрыгнуть» оставшиеся выражения в цикле и перейти к следующей итерации.

```
1  for num in range(5):
2      if num == 3:
3          continue
4      else:
5          print(num)
```

```
0
1
2
4
```

```
1  i = 0
2  while i < 6:
3      i += 1
4      if i == 3:
5          print("Пропускаем 3")
6          continue
7          print("Это никто не увидит")
8      else:
9          print("Текущее значение: ", i)
```

```
Текущее значение:  1
Текущее значение:  2
Пропускаем 3
Текущее значение:  4
Текущее значение:  5
Текущее значение:  6
```

Оператор pass

- Оператор **pass** ничего не делает. Фактически, это такая заглушка для объектов.
- Например, **pass** может помочь в ситуации, когда нужно прописать структуру скрипта. Его можно ставить в циклах, функциях, классах. И это не будет вли[illegible]

```
1 for num in range(5):
2     if num < 3:
3         pass
4     else:
5         print(num)
```

```
3
4
```

List, dict, set comprehensions

Python поддерживает специальные выражения, которые позволяют компактно создавать списки, словари и множества.

На английском эти выражения называются, соответственно:

- List comprehensions
- Dict comprehensions
- Set comprehensions

Эти выражения не только позволяют более компактно создавать соответствующие объекты, но и создают их быстрее.

List comprehensions
(генераторы списков)

```
1  numbers = ['num {}'.format(num) for num in range(10,16)]
2  print(numbers)
```

```
['num 10', 'num 11', 'num 12', 'num 13', 'num 14', 'num 15']
```

```
1  numbers = []
2  for num in range(10,16):
3      numbers.append('num {}'.format(num))
4  print(numbers)
```

```
['num 10', 'num 11', 'num 12', 'num 13', 'num 14', 'num 15']
```

List comprehensions
(генераторы списков)

```python
# В list comprehensions можно использовать выражение if.
# Таким образом можно добавлять в список только некоторые объекты.
items = ['10', '20', 'a', '30', 'b', '40']
only_digits = []
for item in items:
    if item.isdigit():
        only_digits.append(int(item))
print(only_digits)
```

```
[10, 20, 30, 40]
```

```python
items = ['10', '20', 'a', '30', 'b', '40']
only_digits = [int(item) for item in items if item.isdigit()]
print(only_digits)
```

```
[10, 20, 30, 40]
```

List comprehensions
(генераторы списков)

```
[expression for item1 in iterable1 if condition1
            for item2 in iterable2 if condition2
            ...
            for itemN in iterableN if conditionN ]
```

Это значит, можно использовать несколько for в выражении.

List comprehensions
(генераторы списков)

```python
numbers = [[10, 21, 35], [101, 115, 150], [111, 40, 50]]
result = []
for num_list in numbers:
    for num in num_list:
        result.append(num)
print(result)
```

```
[10, 21, 35, 101, 115, 150, 111, 40, 50]
```

```python
numbers = [[10,21,35], [101, 115, 150], [111, 40, 50]]

result = [num for num_list in numbers for num in num_list]

print(result)
```

```
[10, 21, 35, 101, 115, 150, 111, 40, 50]
```

List comprehensions
(генераторы списков) и функция zip

```python
numbers = [[10,21,35], [101, 115, 150], [111, 40, 50]]

result = [num for num_list in numbers for num in num_list]

print(result)
```

```
[10, 21, 35, 101, 115, 150, 111, 40, 50]
```

```python
numbers = [100, 110, 150, 200]

names = ['A', 'B', 'C', 'D']

result = [f'number {num}\n name {name}' for num, name in zip(numbers, names)]

print('\n'.join(result))
```

```
number 100
 name A
number 110
 name B
number 150
 name C
number 200
 name D
```

Функция zip

- В **Pyhon** функция **zip** позволяет пройтись одновременно по нескольким итерируемым объектам (спискам и др.)
- Здесь выражение **zip(a, b)** создает объект-итератор, из которого при каждом обороте цикла извлекается кортеж, состоящий из двух элементов. Первый берется из списка **a**, второй - из **b**.

```
1 result = zip(numbers, names)
2 print(result)
```

```
<zip object at 0x005A95E8>
```

Dict comprehensions (генераторы словарей)

- Генераторы словарей аналогичны генераторам списков, но они используются для создания словарей.

```
d = {}

for num in range(1,11):
    d[num] = num**2

print(d)
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

```
d = {num: num**2 for num in range(1,11)}

print(d)
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

Dict comprehensions (генераторы словарей)

```python
d = {
    'Moscow': 'Russian Federation',
    'London': 'England',
    'Madrid': 'Spain',
    'Paris': 'France'
}

d_upper = {str.upper(key): str.lower(value) for key, value in d.items()}
print(d_upper)
```

```
{'MOSCOW': 'russian federation', 'LONDON': 'england', 'MADRID': 'spain', 'PARIS': 'france'}
```

Set comprehensions (генераторы множеств)

- Генераторы множеств в целом аналогичны генераторам списков.
- Например, надо получить множество с уникальными числами (решение без генератора и с ним):

```
1  numbers = [100, 110, 100, 10, 150, 200, 150]
2  s = set()
3  for num in numbers:
4      s.add(int(num))
5  print(s)
```

```
{100, 200, 10, 110, 150}
```

```
1  numbers = [100, 110, 100, 10, 150, 200, 150]
2  s = {num for num in numbers}
3  print(s)
```

```
{100, 200, 10, 110, 150}
```

Задача №1 для самостоятельного решения

- Напишите программу, которая выводит часть последовательности 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 ... (число повторяется столько раз, чему равно). На вход программе передаётся неотрицательное целое число n — столько элементов последовательности должна отобразить программа. На выходе ожидается последовательность чисел, записанных через пробел в одну строку.

- Например, если n = 7, то программа должна вывести 1 2 2 3 3 3 4.

Задача №2 для самостоятельного решения

- Напишите программу, которая считывает список чисел lst из первой строки и число x из второй строки, которая выводит все позиции, на которых встречается число x в переданном списке lst.

- Позиции нумеруются с нуля, если число x не встречается в списке, вывести строку "Отсутствует"