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 №3. «Наполнение Python: списки, словари, кортежи и множества. Циклы»

Калуга - 2020

Форматирование строк
с помощью f-строк

- F-строки позволяют не только подставлять какие-то значения в шаблон, но и позволяют выполнять вызовы функций, методов и т.п.
- F-строки — это литерал строки с буквой f перед ним. Внутри f-строки в паре фигурных скобок указываются имена переменных, которые надо подставить:

```
1 country = "Russian Federation"
2 city = "Kaluga"
3 print(f"I'm from {country} and live in {city}")
```

```
I'm from Russian Federation and live in Kaluga
```

- Очень важное отличие f-строк от format: f-строки — это выражение, которое выполняется, а не просто строка.

Форматирование строк с помощью f-строк

```
1  street = "Lenina"
2  house = 20
3  appartment = 30
4  print(f'''
5      I'm from {country.upper()} and live in {city.lower()}.
6      My address is:
7      Street:{street:>15} House#{house:>10} Appartment#{appartment:>10}''')
```

```
I'm from RUSSIAN FEDERATION and live in kaluga.
My address is:
Street:          Lenina House#         20 Appartment#         30
```

Объединение литералов строк

```
1  s = ('Test' 'String')
2  s
```

```
'TestString'
```

```
1  s1 = 'Test' 'String'
2  s1
```

```
'TestString'
```

```
1  print(id('Test'))
2  print(id('String'))
3  print(id(s))
4  print(id(s1))
```

```
1817472
47910080
89710944
89710944
```

```
1  s = ('New'
2       'Test'
3       'String')
4  s
```

```
'NewTestString'
```

Тернарное выражение (Ternary expression)

- Иногда удобнее использовать тернарный оператор, нежели развернутую форму

```
<expression on true> if <predicate>
else <expression on false>
```

```
1  s = [1, 2, 3, 4]
2  result = True if len(s) > 5 else False
3  result
```

```
False
```

```
1  number = int(input('Enter n: '))
2  n = number if number >= 0 else -number
3  n
```

- Этим лучше не злоупотреблять, но в простых выражениях такая запись может быть полезной

Список (List)

- последовательность элементов, разделенных между собой запятой и заключенных в квадратные скобки изменяемый упорядоченный тип данных

```
1 l1 = [10, 20, 30]
2 l2 = ['a', 'b', 'c']
3 l3 = [1, 'd', 2.3]
4 print(l1)
5 print(l2)
6 print(l3)
```

```
[10, 20, 30]
['a', 'b', 'c']
[1, 'd', 2.3]
```

```
1 l4 = list('string')
2 print(l4)
```

```
['s', 't', 'r', 'i', 'n', 'g']
```

```
1 print(f'{l3[0]}\t{l3[-1]}\t{l3[::2]}\t{l3[::-1]}')
```

```
1       2.3     [1, 2.3]        [2.3, 'd', 1]
```

Список (List)

```
1 print(l3)
2 l3.reverse()
3 print(l3)
```

```
['100', 'd', 2.3]
[2.3, 'd', '100']
```

```
1 l3[0] = '100'
2 l3
```

```
['100', 'd', 2.3]
```

```
1 l4 = [l1, l2, l3]
2 l4
```

```
[[10, 20, 30], ['a', 'b', 'c'], ['100', 'd', 2.3]]
```

```
1 l2[1] = '***'
2 l4
```

```
[[10, 20, 30], ['a', '***', 'c'], ['100', 'd', 2.3]]
```

Список (List)

```
1  len(l3)
```

```
3
```

```
1  l5 = ['Ivan', 'Oleg', 'Pavel', 'Artem']
2  sorted(l5)
```

```
['Artem', 'Ivan', 'Oleg', 'Pavel']
```

```
1  # Метод join() собирает список строк в одну строку
2  # с разделителем, который указан перед join:
3  ' , '.join(l5)
```

```
'Ivan , Oleg , Pavel , Artem'
```

```
1  # Метод append() добавляет в конец списка указанный элемент:
2  l5.append('Nikita')
3  l5
```

```
['Ivan', 'Oleg', 'Pavel', 'Artem', 'Nikita']
```

Список (List)

```
# Если нужно объединить два списка, то можно использовать два способа:
# метод extend() и операцию сложения.
# У этих способов есть важное отличие - extend меняет список,
# к которому применен метод, а суммирование возвращает новый список,
# который состоит из двух.
l5.extend(l3)
l5
```

```
['Ivan', 'Oleg', 'Pavel', 'Artem', 'Nikita', 2.3, 'd', '100']
```

```
l6 = l5 + l3
l6
```

```
['Ivan', 'Oleg', 'Pavel', 'Artem', 'Nikita', 2.3, 'd', '100', 2.3, 'd', '100']
```

```
l6.pop(-6)
```

```
2.3
```

```
l6
```

```
['Ivan', 'Oleg', 'Pavel', 'Artem', 'Nikita', 'd', '100', 2.3, 'd', '100']
```

Список (List)

```
1  l6
```

```
['Ivan', 'Oleg', 'Pavel', 'Artem', 'Nikita', 'd', 1, '100', 'd', 1]
```

```
1  l6.remove('d')
2  l6
```

```
['Ivan', 'Oleg', 'Pavel', 'Artem', 'Nikita', 1, '100', 'd', 1]
```

```
1  l6.remove(-1)
2  l6
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-38-299af08808d3> in <module>
----> 1 l6.remove(-1)
      2 l6

ValueError: list.remove(x): x not in list
```

```
1  l6.index('100')
```

```
6
```

Список (List)

```
1 l6.insert(2, '10')
2 l6
```

```
['Ivan', 'Oleg', '10', 'Pavel', 'Artem', 'Nikita', 1, '100', 'd', 1]
```

```
1 sorted(l6)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-51-3f53467fab92> in <module>
----> 1 sorted(l6)

TypeError: '<' not supported between instances of 'int' and 'str'
```

```
1 l6.remove(1)
2 l6
```

```
['Ivan', 'Oleg', '10', 'Pavel', 'Artem', 'Nikita', '100', 'd']
```

```
1 sorted(l6)
```

```
['10', '100', 'Artem', 'Ivan', 'Nikita', 'Oleg', 'Pavel', 'd']
```

Список (List)

```
1 l6
```

```
['Ivan', 'Oleg', '10', 'Pavel', 'Artem', 'Nikita', '100', 'd']
```

```
1 l6.sort()
2 l6
```

```
['10', '100', 'Artem', 'Ivan', 'Nikita', 'Oleg', 'Pavel', 'd']
```

```
1 l6.sort(reverse=True)
2 l6
```

```
['d', 'Pavel', 'Oleg', 'Nikita', 'Ivan', 'Artem', '100', '10']
```

```
1 s = "aaaa  a   aaa    aaaaaaaaaaaaaa "
2 lst = s.split()
3 lst
```

```
['aaaa', 'a', 'aaa', 'aaaaaaaaaaaaaa']
```

```
1 lst = s.split(' ')
2 lst
```

```
['aaaa', '', 'a', '', '', 'aaa', '', '', '', 'aaaaaaaaaaaaaa', '']
```

Словарь (Dictionary)

Словари – это изменяемый упорядоченный тип данных:

- данные в словаре – это пары ключ: значение
- доступ к значениям осуществляется по ключу, а не по номеру, как в списках
- данные в словаре упорядочены по порядку добавления элементов
- так как словари изменяемы, то элементы словаря можно менять, добавлять, удалять
- ключ должен быть объектом неизменяемого типа: число, строка, кортеж
- значение может быть данными любого типа

Словарь (Dictionary)

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

```
'England'
```

```
d['Istambul'] = 'Turkey'
d
```

```
{'Moscow': 'Russian Federation',
 'London': 'England',
 'Madrid': 'Spain',
 'Paris': 'France',
 'Istambul': 'Turkey'}
```

```
sorted(d)
```

```
['Istambul', 'London', 'Madrid', 'Moscow', 'Paris']
```

Словарь (Dictionary)

```
1  # метод copy() позволяет создать полную копию словаря
2  d_copy = d.copy()
3  d_copy
```

```
{'Moscow': 'Russian Federation',
 'London': 'England',
 'Madrid': 'Spain',
 'Paris': 'France',
 'Istambul': 'Turkey'}
```

```
1  # метод clear() позволяет очистить словарь
2  d.clear()
3  d
```

```
{}
```

```
1  capitals = d_copy
```

```
1  # метод get() запрашивает ключ, и если его нет,
2  # вместо ошибки возвращает None
3  print(capitals.get('Tokyo'))
```

```
None
```

Словарь (Dictionary)

```
1  # метод setdefault() ищет ключ, и если его нет,
2  # вместо ошибки создает ключ со значением None
3  capitals.setdefault('Tokyo')
4  capitals
```

```
{'Moscow': 'Russian Federation',
 'London': 'England',
 'Madrid': 'Spain',
 'Paris': 'France',
 'Istambul': 'Turkey',
 'Tokyo': None}
```

```
1  capitals.setdefault('Athens', 'no_country')
2  capitals
```

```
{'Moscow': 'Russian Federation',
 'London': 'England',
 'Madrid': 'Spain',
 'Paris': 'France',
 'Istambul': 'Turkey',
 'Tokyo': None,
 'Athens': 'no_country'}
```

Словарь (Dictionary)

```
1 capitals.keys()
```

```
dict_keys(['Moscow', 'London', 'Madrid', 'Paris', 'Istambul', 'Tokyo', 'Athens'])
```

```
1 capitals.values()
```

```
dict_values(['Russian Federation', 'England', 'Spain', 'France', 'Turkey', None, 'no_country'])
```

```
1 capitals.items()
```

```
dict_items([('Moscow', 'Russian Federation'), ('London', 'England'), ('Madrid', 'Spain'), ('Paris', 'France'), ('Istambul', 'Tu
rkey'), ('Tokyo', None), ('Athens', 'no_country')])
```

```
1 del capitals['Tokyo']
2 capitals
```

```
{'Moscow': 'Russian Federation',
 'London': 'England',
 'Madrid': 'Spain',
 'Paris': 'France',
 'Istambul': 'Turkey',
 'Athens': 'no_country'}
```

Словарь (Dictionary)

```
1  capitals.update({'Athens':'Greece'})
2  capitals
```

```
{'Moscow': 'Russian Federation',
 'London': 'England',
 'Madrid': 'Spain',
 'Paris': 'France',
 'Istambul': 'Turkey',
 'Athens': 'Greece'}
```

```
1  for key, value in capitals.items():
2      if key == 'Moscow':
3          print(value)
```

```
Russian Federation
```

```
1  d_keys = capitals.keys()
2  empty_capitals = dict.fromkeys(d_keys)
3  empty_capitals
```

```
{'Moscow': None,
 'London': None,
 'Madrid': None,
 'Paris': None,
 'Istambul': None,
 'Athens': None}
```

```
1  if 'Paris' in capitals:
2      print(capitals['Paris'])
3  else:
4      print('error')
```

```
France
```

Кортеж (Tuple)

Кортеж в Python это:

- последовательность элементов, которые разделены между собой запятой и заключены в скобки
- неизменяемый упорядоченный тип данных
- Грубо говоря, кортеж – это список, который нельзя изменить. То есть, в кортеже есть только права на чтение. Это может быть защитой от случайных изменений.

Кортеж (Tuple)

```
1  t1 = tuple()
2  t1
```

```
()
```

```
1  t2 = ('password')
2  t2
```

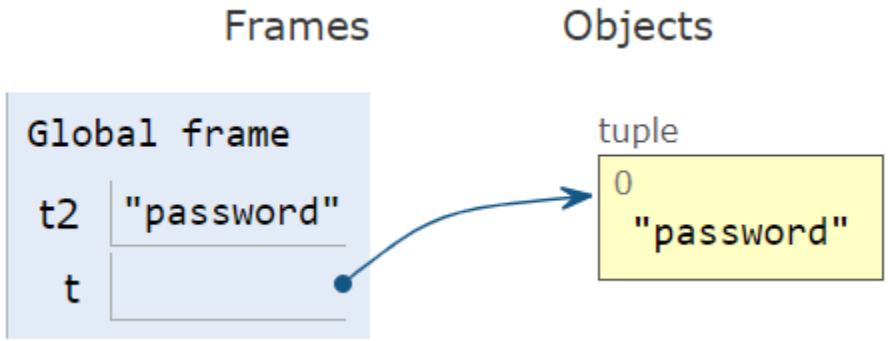
```
'password'
```

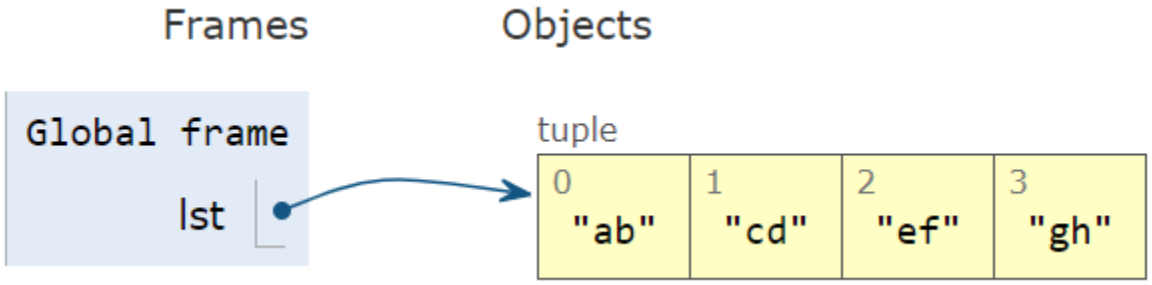
```
1  t = ('password', )
2  t
```

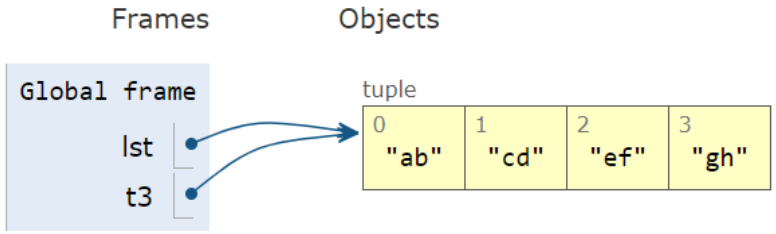
```
('password',)
```

```
1  lst = ('ab', 'cd', 'ef', 'gh')
2  t3 = tuple(lst)
3  t3
```

```
('ab', 'cd', 'ef', 'gh')
```







Кортеж (Tuple)

```
1 t3[1]
```

```
'cd'
```

```
1 t3[1] = 'ij'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-b7b097182ca1> in <module>
----> 1 t3[1] = 'ij'

TypeError: 'tuple' object does not support item assignment
```

```
1 t4 = sorted(t3, reverse=True)
2 t4
```

```
['gh', 'ef', 'cd', 'ab']
```

Особенности кортежей

- Кортежи занимают меньше места
- Вы не сможете изменить/удалить элемент кортежа по ошибке
- Вы сможете использовать кортежи как ключ словаря
- Аргументы функции передаются как кортежи

Множество (Set)

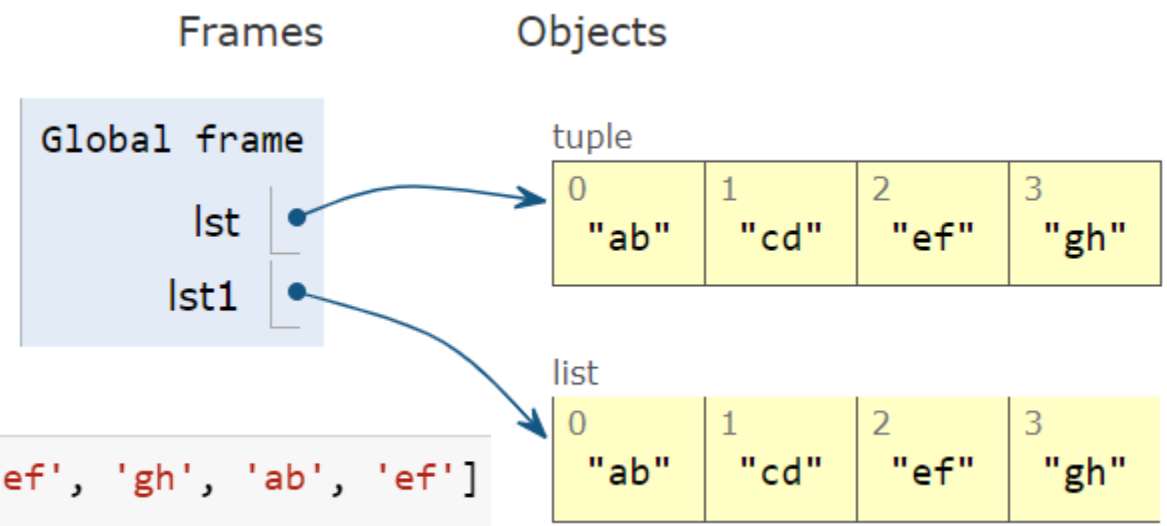
- Множество – это изменяемый неупорядоченный тип данных. В множестве всегда содержатся только уникальные элементы.

- Множество в Python – это последовательность элементов, которые разделены между собой запятой и заключены в фигурные скобки.

- С помощью множества можно легко убрать повторяющиеся элементы.

Множество (Set) – создание

```
1  lst = ('ab', 'cd', 'ef', 'gh')
2  lst1 = ['ab', 'cd', 'ef', 'gh']
```




```
1  lst1 = ['ab', 'cd', 'ef', 'gh', 'ab', 'ef']
2  lst1
```

```
['ab', 'cd', 'ef', 'gh', 'ab', 'ef']
```

```
1  s = set(lst1)
2  s
```

```
{'ab', 'cd', 'ef', 'gh'}
```

Множество (Set) – методы

```
1  # add добавляет элемент во множество
2  s.add('ij')
3  s
```

```
{10, 'ab', 'cd', 'ef', 'gh', 'ij'}
```

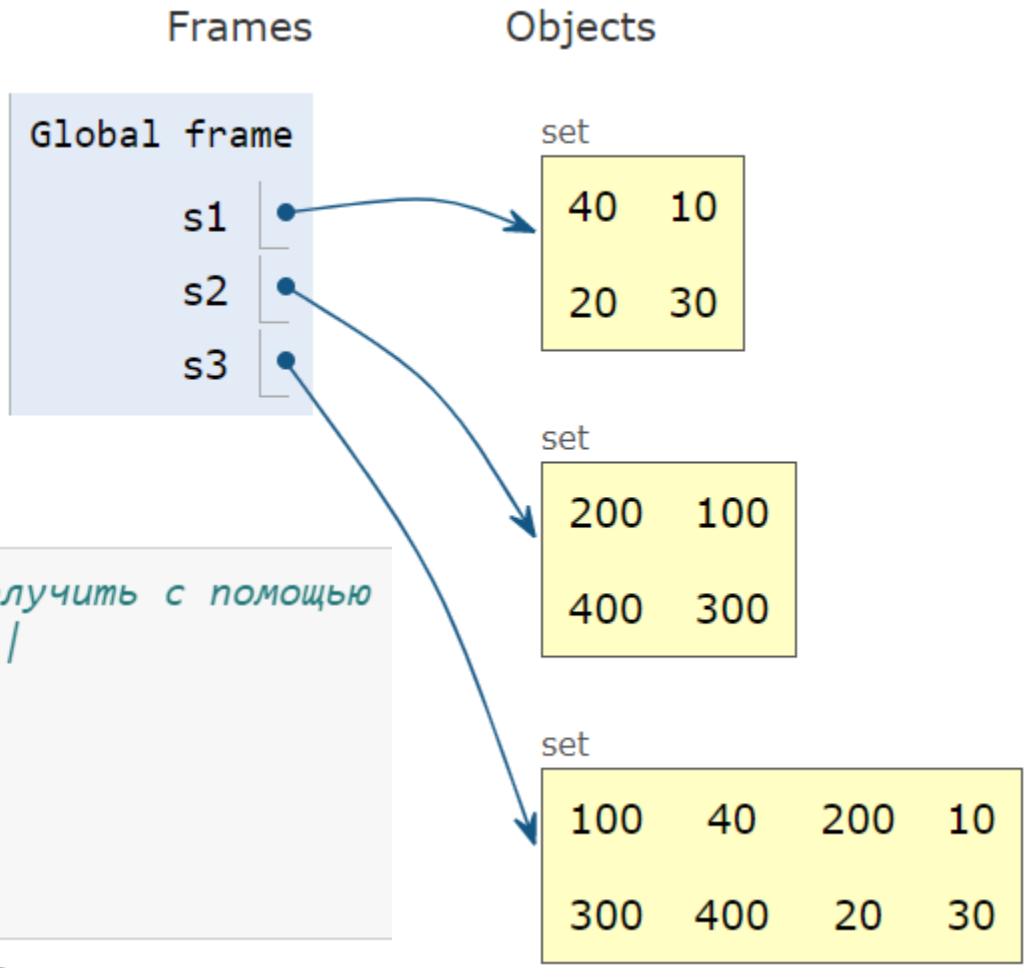
```
1  # discard() позволяет удалять элементы, не выдавая ошибку,
2  # если элемента в множестве нет
3  s.discard(20)
4  s.discard(10)
5  s
```

```
{'ab', 'cd', 'ef', 'gh', 'ij'}
```

```
1  # clear() очищает множество
2  s.clear()
3  s
```

```
set()
```

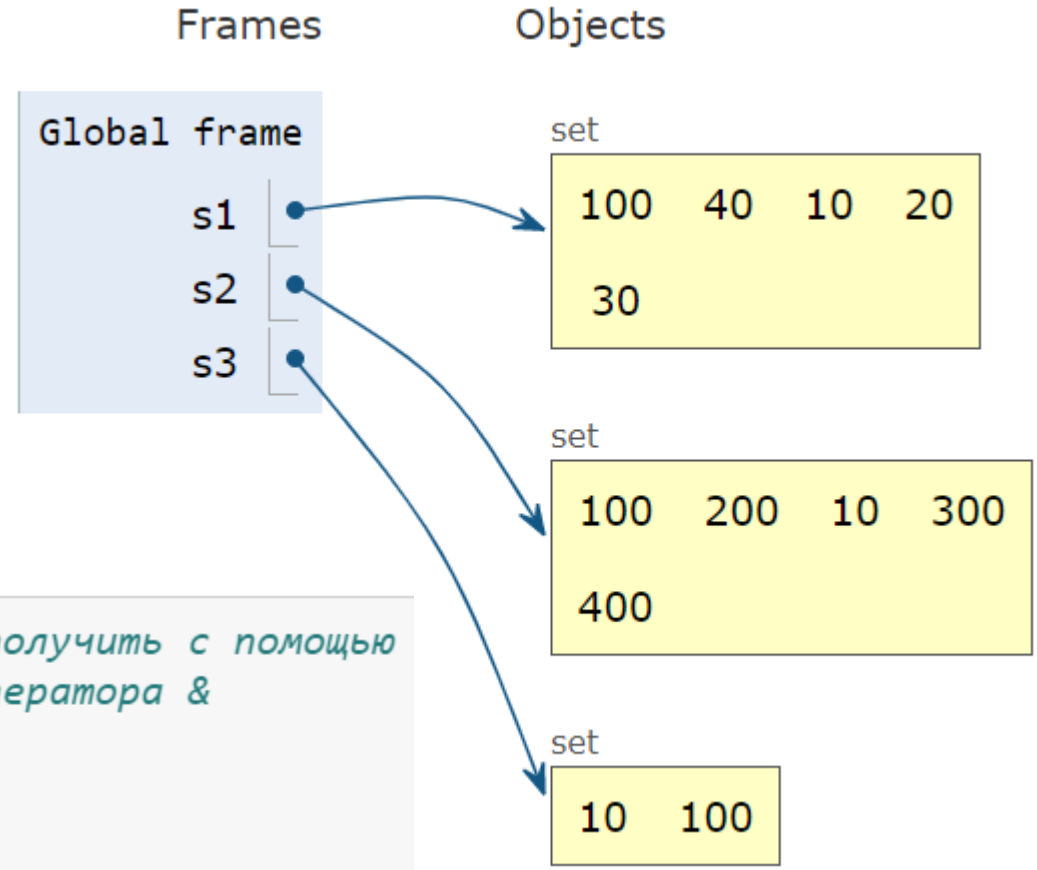
Множество (Set) – операции




```
1  # объединение множеств можно получить с помощью
2  # метода union() или оператора |
3  s1 = {10, 20, 30, 40}
4  s2 = {100, 200, 300, 400}
5  s3 = s1.union(s2)
6  s3 = s1 | s2
7  s3
```

```
{10, 20, 30, 40, 100, 200, 300, 400}
```

Множество (Set) – операции




```
1  # пересечение множеств можно получить с помощью
2  # метода intersection() или оператора &
3  s1 = {10, 20, 30, 40, 100}
4  s2 = {100, 200, 300, 400, 10}
5  s3 = s1.intersection(s2)
6  s3 = s1 & s2
7  s3
```

```
{10, 100}
```

Множество (Set) – способы создания

```
1 set1 = {}
2 type(set1)
```

```
dict
```

```
1 set2 = set()
2 type(set2)
```

```
set
```

```
1 set3 = set('long long sting')
2 set3
```

```
{' ', 'g', 'i', 'l', 'n', 'o', 's', 't'}
```

```
1 set4 = set([10, 20, 30, 10, 40, 20, 50])
2 set4
```

```
{10, 20, 30, 40, 50}
```

Циклы - for

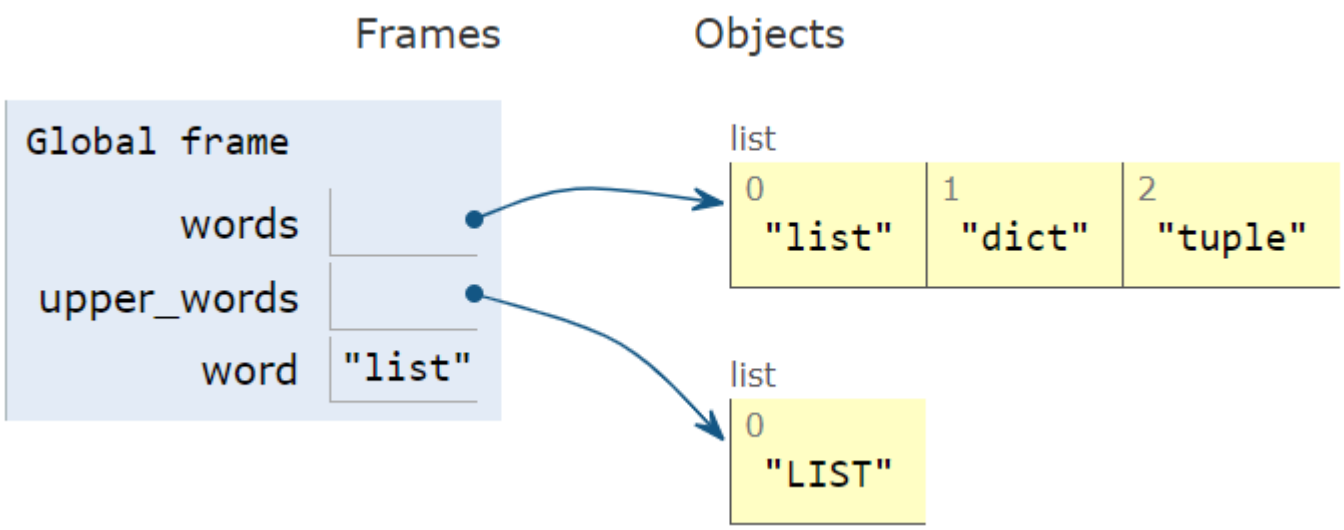
Примеры последовательностей элементов, по которым может проходиться цикл **for**:

- 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
```