Списки (Lists)

Список – это упорядоченная изменяемая коллекция объектов произвольных типов

Список – изменяемый тип данных, т.е. его можно модифицировать. Элементами списка могут быть любые объекты

Создание списков

```
a = [2,5,7]
a = [] # empty list
type(a) # list
a = list('list') # ['l','i','s','t']
a = ['s', 'p', ['isok'], 2]
a = [2,5,7]
b = a+[3,3] # [2,5,7,3,3]
b = [1,2]*3 # [1,2,1,2,1,2]
```

Доступ к элементам списка. Срезы

```
a = [1,3,8,7]
a[0] # 1
a[10] # error
a[:] # [1,3,8,7]
a[1:] # [3,8,7]
a[:3] # [1,3,8]
a[::2] # [1,8]
a[::-1] # [7,8,3,1]
a[:-2] # [1,3]
a[-2::-1] # [8,3,1]
a[1:4:-1] # []
a[10:20] # []
a[i:j:step]
```

Функции списков

```
a = [1,3,8,7]
len(a) # 4
min(a) # 1
max(a) # 8
sum(a) # 19
sorted(a) # [1,3,7,8]
del a[:-2] # [8,7]

a = [1,3,8,7]
a[1:3] = [0,0,0,0]
# [1,0,0,0,0,7]

if 7 in a:
    print('Element is in list')
```

Методы списков

```
a = [1,8,7,3]
a.sort() # a = [1,3,7,8]

a = [1,8,7,7,3]
a.count(7) # 2
a.count(27) # 0
a.index(8) # 1
a.append([15,20]) # [1,8,7,7,3,[15,20]]
a.extend([15,20]) # [1,8,7,7,3,[15,20],15,20]

a = [1,8,7,3]
a.reverse() # [3,7,8,1]
a.remove(7) # [3,8,1]
a.clear() # []
...
```

Псевдонимы (Aliases) в Python

```
a = [1,8,7,3]
b = a # reference to a
b[0] = 15
print(a) # [15,8,7,3]
```

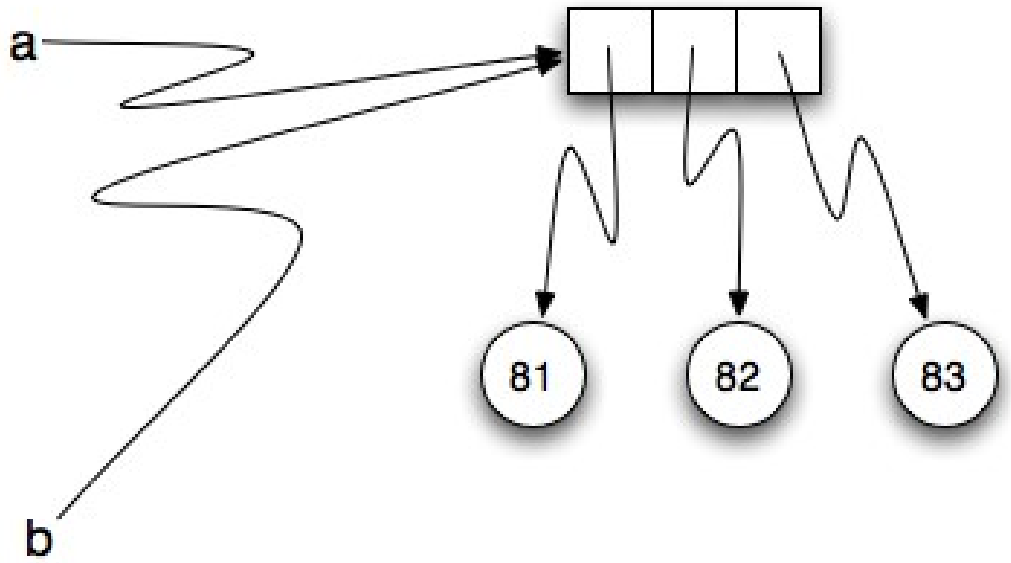
В Python две переменные называются псевдонимами, если они ссылаются на одинаковые адреса памяти

Как проверить, ссылаются ли переменные на один и тот же объект?

```
a = [1,8,7,3]
b = a
c = b is a # True
b = [1,8,7,3]
c = b is a # False
```

Псевдонимы. Иллюстрация

```
a = [81,82,83]
b = a
b == a # True
b is a # True
id(b) == id(a) # True
```



Копирование списков

- **Поверхностное копирование (shallow copy)**
  Создается новый объект, но он будет заполнен ссылками на элементы, которые содержались в оригинале

```
a = [4,3,[2,1]]
b = a[:] # b is a copy
b is a # False
b[2][0]=-100
print(a) # [4,3,[-100,1]]
```

- **Глубокое копирование (deep copy)**
  Создается новый объект и рекурсивно создаются копии всех объектов, содержащихся в оригинале

```
import copy
a = [4,3,[2,1]]
b = copy.deepcopy(a)
b[2][0]=-100
print(a) # [4,3,[2,1]]
```

Списки и строки

```
s = 'hello'
a = list(s) # ['h','e','l','l','o']
''.join(a) # 'hello'
' '.join(a) # 'h e l l o'
'**'.join(a) # 'h**e**l**l**o'
'h*e*l*l*o'.split('*') # ['h','e','l','l','o']
list(str(123)) # ['1','2','3']
```

Итерирование строк и списков

```
num = [0.8, 7.0, 6.8, -6]
for i in num:
    print(i)
s = 'hello'
for ch in s:
    print(ch)
```

Интернирование

```
a = 'abc'
b = a # not a copy, just reference to a
b = a[:]; b is a # True
c = str(a); c is a # True
```

Копирования объектов b и c не происходит

Та же ситуация с другими неизменяемыми типами: int, float, bytes, frozenset и др.

```
a = 'abc'; b = 'abc'
b is a # True
```

Создание нового объекта b не происходит, вместо этого создаётся ссылка на объект a

Интернирование (interning) – механизм оптимизации кода, при котором хранится лишь одна копия из множества одинаковых объектов

Диапазоны (Ranges)

Диапазон – последовательность целых чисел

```
r = range(i, j) # i , i+1, i+2, ..., j-1
r = range(i, j, step)
r = range(2, 20, 2) # 2 4 6 8 10 12 14 16 18
r = range(20, 2, -2) # 20 18 16 14 12 10 8 6 4
type(r) # range
```

Итерирование диапазонов

```
r = range(5) # 0 1 2 3 4
for i in r:
    print(i)
a = list(r) # [0,1,2,3,4]
for i in range(len(a)):
    print(a[i])
```

Итерируемые классы и итераторы

Инструкция **for–in** используется для итерирования по элементам последовательностей либо других типов, поддерживающих итерирование

Классы, поддерживающие итерирование, (iterables) реализуют метод __iter__ , возвращающий итератор

Классы-итераторы (iterators) реализуют метод __next__

```
s = 'abc' # s is iterable
for c in s:
    print(c)

# user iteration
it = iter(s) # get iterator (type str_iterator)
for i in range(len(s)):
    print(next(it))
```

Генераторы списков (List Comprehensions)

Генераторы списков служат для создания новых списков на основе существующих

```
a = [c*3 for c in 'list']
# ['lll','iii','sss','ttt']
a = [c*3 for c in 'list' if c != 'i']
# ['lll','sss','ttt']
a = [i**2 for i in range(1,5)]
# [1,4,9,16]
b = [i+2 for i in a]
# [3,6,11,18]
b = [(i+2 if i%2==0 else 0) for i in a]
# [0,6,0,18]
# value = i+2 if i%2==0 else 0
# b = [value for i in a]
```

Функции-генераторы

range() – генератор последовательности целых чисел

Генераторы не хранят значения в памяти, а генерируют их на лету

Функции-генераторы возвращают значение, используя yield

```
def my_range(first=0.0, last=1.0, step=0.1):
    number = first
    while number < last:
        yield number
        number += step

for x in my_range():
    print(x)
# 0.0 0.1 ... 0.9
```

Ключевое слово `yield`

При использовании ключевого слова `yield` функция возвращает значение без уничтожения локальных переменных, кроме того, при каждом последующем вызове функция начинает своё выполнение с оператора `yield`

```
type(my_range) # function
gen = my_range() # type(gen) — generator
for x in gen:
    print(x)
# 0.0 0.1 ... 0.9
for x in gen:
    print(x)
# Empty
```

Генераторы применяются в тех случаях, когда нет необходимости сохранять всю последовательность и промежуточные значения в памяти

Итерирование генератора

Для итерирования генератора служит функция **next**

```
gen = my_range()
next(gen) # 0.0
next(gen) # 0.1
...
next(gen) # 0.9
next(gen) # error
```

Функция-генератор может быть создана при помощи круглых скобок ()

```
gen = (x for x in [1,2,3])
gen = (x+2 for x in my_range(0,0.5))
for i in gen:
    print(i)
# 2.0 2.1 2.2 2.3 2.4
```

Кортежи (Tuples)

Кортеж – это упорядоченная неизменяемая коллекция объектов произвольных типов

Кортеж = неизменяемый список

Создание кортежей

```
a = (2,5,7)
a = 2,5,7 # parentheses are unnecesary
a = tuple() # empty tuple
a = () # empty tuple
type(a) # tuple
a = tuple('tuple') # ('t','u','p','l','e')
a = ('t', 'u', ['ple'], 2)
a = (2) # 2 (type int)
a = (2,) # (2,) (type tuple)
a = (1,2)*3 # (1,2,1,2,1,2)
```

Зачем нужны кортежи?

- Кортежи используются в тех случаях, когда набор значений не должен изменяться
- Кортежи занимают меньший объём памяти, чем списки

С помощью кортежей можно присваивать значения одновременно нескольким переменным:

```
(a,b,c) = (2,10,7)
# a=2, b=10, c=7
```

Если кортеж содержит изменяемые объекты, то их можно изменить:

```
a = (1,[5,7],'a')
a[1][0] = 12
# a = (1,[12,7],'a')
```

Функции и методы кортежей

Функции и методы кортежей аналогичны функциям и методам списков

```
a = (1,3,8,7)
len(a) # 4
min(a) # 1
max(a) # 8
sum(a) # 19
sorted(a) # (1,3,7,8)
del a[1] # error: tuple is immutable
a[1:3] # (3,8)
if 3 in a:
    print('Element is in tuple')
a.count(7) # 1
a.index(7) # 3
```

Множества (Sets)

Множество – это изменяемая неупорядоченная коллекция неизменяемых уникальных объектов

Создание множеств

```
a = {2,'abc',7,(2,3)}
a = {2,'abc',7,[2,3]} # error: list is mutable
a = set() # empty set
type(a) # set
a = set('hello') # {'h','e','l','o'}
a = set([2,5,2,7]) # {2,5,7}
a = set( (2,5,2,7) ) # {2,5,7}
a = set(range(5)) # {0,1,2,3,4}

# remove duplicates from tuple/list
tuple(set( (3,6,3,5) )) # (3,5,6)
list(set([3,6,3,5])) # [3,5,6]
```

Операции над множествами

```
a = {2,'abc',7}
a.add(5) # {2,5,7,'abc'}
a.remove('abc') # {2,5,7}
a.remove('qwe') # error
a.discard('qwe') # do nothing if element doesn't exist

a = {2,5,7}
b = {5,7,12}
a.intersection(b) # a&b = {5,7}
b.intersection(a) # b&a = {5,7}
a.union(b) # a|b = {2,5,7,12}
a.difference(b) # a\b={2}
b.difference(a) # a\b={12}
if 5 in a: print('Element belongs to set')
```

Неизменяемые множества (Frozen sets)

Set – изменяемое множество

Frozen set – неизменяемое множество

```
a = {2,'abc',7,(2,3)}
b = frozenset(a)
type(b) # frozenset
c = set(b)
type(c) # set
c.add(12) # ok
b.add(12) # error: frozenset is immutable
b.remove(2) # error: frozenset is immutable
a == b # True
```

Отличие между Set и Frozen set такое же, как и между List and Tuple

Словари (Dictionaries)

Словарь – неупорядоченная изменяемая коллекция с произвольными ключами неизменяемого типа
Ключами словаря могут быть произвольные неизменяемые объекты (не обязательно одного типа)

```
a = {'key1':12, 'key2':15}
a = {101:12, 102:15}
a = {('s',5):12, (8,5):15}
a = dict() # empty dict
a = {} # empty dict
type(a) # dict
a = dict(key1=12, key2=15) # {'key1':12, 'key2':15}
# create from collection of tuples (key,value)
a = dict([('key1',12), ('key2',15)])
a = dict(( ('key1',12), ('key2',15) ))
a = dict({('key1',12), ('key2',15)})
```

Генераторы словарей (Dictionary Comprehensions)

Генераторы словарей служат для создания новых словарей на основе существующих списков

```
a = {c:c*3 for c in 'list'}
# {'i':'iii', 'l':'lll', 's':'sss', 't':'ttt'}
a = {c:c*3 for c in 'list' if c != 'i'}
# {'l':'lll', 's':'sss', 't':'ttt'}
a = {i:i**2 for i in range(1,5)}
# {1:1, 2:4, 3:9, 4:16}
b = ['abc', 'qwe', 'str']
a = {2*i:b[i] for i in range(len(b))}
# {0:'abc', 2:'qwe', 4:'str'}
b = [('key1', 12), ('key2', 15)]
a = {k:v for (k,v) in b}
# {'key1':12, 'key2':15}
```

Функции словарей

```
a = {'key1':12, 'key2':15, (1,2):18}
a['key1'] # 12
a[1] # error: key doesn't exist

len(a) # 3
del a[(1,2)]
# a={'key1':12, 'key2':15}

a['key1'] = [2,3]
# a={'key1':[2,3], 'key2':15}

if 'key1' in a:
    print('Key is in dict')
```

Методы словарей

```
a = {'key1':12, 'key2':15, (1,2): 18}
a.get('key1') # 12
a.get('s') # None (default)
a.keys() # 'key1', 'key2', (1,2)
a.values() # 12, 15, 18
a.items() # 'key1':12, 'key2':15, (1,2):18
b = a # b is a reference to a
b = a.copy() # b is a shallow copy of a
a.pop( (1,2) ) # 18, a= {'key1': 12, 'key2': 15}
a.clear() # {}
```

Итерирование словарей:

```
for (k,v) in a.items():
    print(k,':',v)
```

Функции с переменным числом параметров

```
def total(initial=5, *numbers, **keywords):
    count = initial
    for number in numbers:
        count += number
    for (k,v) in keywords.items():
        count += v
    return count

print(total(10, 1, 2, 3, vegetables=50, fruits=100))
```

Неключевые аргументы объединяются в кортеж

Ключевые аргументы объединяются в словарь

```
# numbers = (1,2,3)
# keywords = {'vegetables':50, 'fruits':100}
```

Функция zip

Функция **zip** упаковывает последовательности одинаковой длины в кортежи

```
a = [2,5,7]
s = 'qwe'
z = zip(a, s) # type(z) - zip
next(z) # (2,'q')
next(z) # (5,'w')
next(z) # (7,'w')
next(z) # error
for (i,c) in zip(a,s):
    print(i,':',c)

z = zip((1,2), 'ab', ['q','w'])
next(z) # (1,'a','q')
next(z) # (2,'b','w')
```

Оператор *

Оператор $*$ распаковывает последовательность на отдельные элементы

```
a = [(1,'a'), (2,'b'), (3,'c')] # list
z = zip(*a) # zip
next(z) # (1,2,3)
next(z) # ('a','b','c')
```

```
def add3(a,b,c):
    return a+b+c

add3([1,4,5]) # error: missing arguments b,c
add3(*[1,4,5]) # 10
add3(*(1,4,5)) # 10
add3(*{1,4,5}) # 10
```

Оператор **

Оператор ** распаковывает словарь на отдельные элементы key:value

```
def add3(a,b,c):
    return a+b+c

d = {'a':1, 'b':2, 'c':3} # dict
add3(*d) # 'abc'
add3(**d) # 6
```

Пример использования:

```
def f(x, *args, **kwargs):
    kwargs['newArg'] = 5
    g(x, *args, **kwargs)
```