шпаргалка по Python

✦ переменные

имена переменных

```python
# не могут начинаться с цифр, а также специальных символов
# не могут совпадать с ключевыми словами языка

# примеры подходящих имен переменных:
a_7
toto2
summa

# примеры неподходящих имен переменных:
8h
sum
_yat
```

ключевые слова

```python
False, True, None, and, with / as, assert, break, class,
continue, def, del, elif, else, except, finally, for, from,
global, if, import, in, is, lambda, nonlocal, not, or, pass,
raise, return, try, while, yield и др.
```

так называть переменные нельзя

присвоение переменных

имя переменной = значение или вычислимое значение

```python
a = 3
b = 35 + 86
c, d, e = 'Hi', -8.9, 1
```

✦ типы данных

базовые типы данных

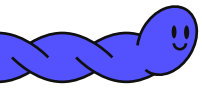
Python
```python
# int - целое число, дробная часть отбрасывается
b = 3
# float - число с плавающей точкой/дробное число
c = 3.5
# bool - True/False - логический тип (Булевый)
e = True
```

контейнерные типы данных

Python
```python
# строка (str)
# "контейнер" - набор символов
# возможно использование как двойных, так и одинарных кавычек
s = '' # пустая строка

a = 'hello'
b = "Webium"

r = a + b # r = 'helloWebium'
d = a * 3 # d = 'hellohellohello'

e = str(645) # e = '645'
```

функции split() и join()

Python
```python
# a.split(x) - возвращает список из элементов строки a,
# строку разделяют по символу x(если не указать x, по
# умолчанию разделение по пробелу)
s = '1+3+5+78+6'
b = s.split('+')  # b = ['1', '3', '5', '78', '6']

# x.join(a) - возвращает строку из элементов списка a,
# соединенных символом x
b = ['3', '57', '86', '7']
s = '*'.join(b)  # s = '3*57*86*7'
```



```python
# список (list) - упорядоченный набор объектов
z = []  # пустой список

a = [1, 'Hi', 17.0, True]
b = [2, 2.5]

d = a + b  # d = [1, 'Hi', 17.0, True, 2, 2.5]
f = b * 3  # f = [2, 2.5, 2, 2.5, 2, 2.5]

p = list("abc")  # p = ['a', 'b', 'c']
```

приведение типов

```python
x = 1000.7
int_x = int(x) # 1000
float_x = float(x) # 1000.7
str_x = str(x) # '1000.7'
bool_x = bool(x) # True

list_a = [1, 2, 1, 3]
set_a = set(list_a) # {1, 2, 3}
list_a = list(set_a) # [1, 2, 3]
```

✦ операции с контейнерными типами данных

операции для всех контейнерных типов

Python

```python
a = [1, 2, 0, 6, 4]

# len(v) - возвращает количество элементов в v
e = len(a) # e = 5

# min(v) - возвращает минимальное значение из v
e = min(a) # e = 0

# max(v) - возвращает максимальное значение из v
e = max(a) # e = 6

# sum(v) - возвращает сумму элементов v
e = sum(a) # e = 13

# sorted(v) - возвращает копию v с отсортированными по возрастанию
# элементами
e = sorted(a) # e = [0, 1, 2, 4, 6]
# sorted(v, reverse = True) - по убыванию
e = sorted(a, reverse = True)  # e = [6, 4, 2, 1, 0]

# val in v - возвращает True, если значение val содержится в v, и
# False в противоположном случае
e = (9 in a)  # e = False
```

операции только для списков

```python
a = [4, 5, 8, 1]

# p.append(val) - добавляет элемент val в конец p
a.append(9)  # a = [4, 5, 8, 1, 9]

# p.insert(ind, val) - вставляет значение val в p по индексу
# ind (элементы сдвигаются вправо от ind)
a.insert(1, 6)  # a = [4, 6, 5, 8, 1]

# p.remove(val) - удаляет первое вхождение val в p
a.remove(5)  # a = [4, 8, 1]

# p.pop(ind) - возвращает значение элемента из p по индексу
# ind + удаляет его из p
s = a.pop(2)  # s = 8; a = [4, 5, 1]

# p.sort() - сортирует p по возрастанию
a.sort()  # a = [1, 4, 5, 8]

# p.reverse() - устанавлиевает расположение элементов p в
# обратном порядке
a.reverse()  # a = [1, 8, 5, 4]
```

операции только для списков и строк

```python
a = [4, 7, 8, 8, 1]

#z.index(val) - возвращает копию z с обратным порядком элементов
c = a.index(7)  # c = 1

# z.count(val) - возвращет количество вхождений val в z
c = a.count(8)  # c = 2
```

```python
# множество (set) - набор уникальных элементов

j = set()  # пустое множество

s = {1, 5, 5, 7, 4, 1}  # s = {1, 5, 7, 4}
```

```python
# словарь (dict) набор пар "ключ:значение"

y = {}  # пустой словарь

k = {1: 'January', 2: 'February', 3: 'March'}
```

операции только для словарей

```python
a = {1:'A', 2:'B', 3:'C'}

# d[key] - возвращает значение d по ключу key
s = a[1] # s = 'A'

# d.keys() - возвращает список ключей d
s = a.keys() # s = [1, 2, 3]

# d.values() - возвращает список значений d
s = a.values() # s = ['A', 'B', 'C']

# d.items() - возвращает список пар (ключ, значение) d
s = a.items() # s = [(1, 'A'), (2, 'B'), (3, 'C')]

# d.pop[key] - возвращает значение из d по ключу key и удаляет эту
# пару
s = a.pop[2] # s = 'B'; a={1:'A', 3:'C'} (очередность элементов
# может быть иной)
```

операции только для множеств

```python
a = {1, 2, 3}

# g.add(val) - добавляет элемент val в g
a.add(4) # a = {1, 4, 2, 3} (очередность элементов может быть
# иной)

# g.remove(val) - удаляет элемент val из g
a.remove(2) # a = {1, 3} (очередность элементов может быть иной)
```

операции только для строк

```python
a = 'hellohello'

# h.replace(s, n, c) - возвращает строку, где произошла замена
подстроки s на подстроку n c раз
m = a.replace('l','r',3) # m = 'herroherlo'
```

✦ доступ к элементам списков и строк

индексы

11 - это и 0й элмент и -4
65 - это 1й и -3

```python
#      0    1     2      3    #индексы элементов
#     -4   -3    -2     -1    #отрицательные индексы
a = [11, 65, 'ab', 78.0]

b = a[0]  # b = 11
c = a[-1]  # c = 78.0
d = a[2]  # d = 'ab'
```

срезы

```python
# s[start:stop:step] - срез элементов, начная от элемента с
# индексом start до элемента с индексом stop(невключительно) и
# шагом step
# по умолчанию 1 параметр - нулевой индекс, 2 параметр - конец
# строки/списка, 3 параметр - единица

a = 'helloWebium'

b = a[:-1] # b = 'helloWebiu'
c = a[1:-1] # c = 'elloWebiu'
d = a[::2] # d = 'hloeim'
e = a[:] # e = 'helloWebium'
f = a[::-1] # f = 'muibeWolleh'
```

✦ ВВОД/ВЫВОД

функция ввода

```python
# Считывает значение, введенное с клавиатуры
# Считанное значение по умолчанию имеет тип str - используйте
# приведение к другим типам при необходимости

x = input(y) # На экран выводится строка y, после чего
# происходит считывание данных, которые записываются в x
```

функция вывода

```python
# Выводит на экран литералы, переменные, выражения

# print(<>, sep=k, end=g)

# sep - строка, которая будет использована в качестве разделителя
параметров(по умолчанию это пробел)
# end - строка, который будет замыкать вывод(по умолчанию это \n
# - перенос строки)

a = 2
b = 56

print(b,'hello', b - a, sep='**', end='!!!') # 56**hello**54!!!
```

✦ математические операции

операции

+ сложение
- вычитание
* умножение
/ деление
// целочисленное деление (целя часть от деления)
% деление по модулю (остаток от деления)
** возведение в степень

функции

Python

```python
# abs(x) - значение числа x по модулю
a = -4
b = abs(a) # b = 4

# round(x,y) - округление числа x до y знаков после запятой
a = 3.57
b = round(a,1) # b = 3.6
```

✦ булевая логика

сравнения

< меньше
> больше
<= меньше или равно
>= больше или равно
== равно
!= не равно

ВОМ

```python
# True - истина
# False - ложь

x = 100
y = 142
# x and y - логическое и, возвращет True, если оба значения x и y
# истинны
a = (x > y) # False
b = (x != y) # True
c = (a and b) # c = False

# x or y - логическое или, возвращает True, если хотя бы одно
# значение x или y истинно
a = (x <= y) # True
b = (x == y) # False
c = (a or b) # c = True

# not x - возвращает True, если значение x ложно
a = (x > y)
c = (not a) # c = False
```

✦ условный оператор → if

Блок инструкций выполняется, только если условие истинно

```python
if <условие 0>:
 # блок инструкций, если <условие 0> == True
elif <условие 1>:
 # блок инструкций, если <условие 0> == False,
 # а <условие 1> == True
# блоков elif может быть сколько угодно
# выполнится первый, для которого его условие == True
else:
 # блок выполнится, если не выполнено ни одно из условий в if
 # и elif
 print('nope')
```

Пример:

Python

```python
if x == 42:
 # блок выполнится, если значение x равно 42
 print('yeah')
elif x > 0:
 # блок выполнится, если значение x не равно 42, но больше 0
else:
 # блок выполнится, если не выполнено ни одно из условий в if и
 # elif
 print('nope')
```

✦ циклы → while и for

циклы → while и for

Блок инструкций выполняется до тех пор, пока условие истинно

```python
while <условие>:

# блок инструкций/операций
```

Важно: избегайте бесконечных циклов!

Пример:

```python
s = 0
i = 1
while i < = 100:
 # блок выполнится до тех пора, пока i меньше или равно 100
 s = s + i ** 2
 i = i + 1 # изменение переменной из условия цикла
```

цикл перебора → for

Блок инструкций выполняется для всех элементов в <последовательности>.
На каждой итерации цикла следующее значение записывается в <переменная>.

```python
for <переменная> in <последовательность>:
# блок инструкций/операций
```

Пример 1:

```python
# Проход по элементам последовательности

s = 'text'
cnt = 0
for c in s:
 # блок будет выполняться для каждого элемента в s
 cnt = cnt + 1
```

Пример 2:

```python
# range(start, stop, step) - генерация последовательности чисел от
# [start;stop) с шагом step
# по умолчанию значение start равно 0, значение step равно 1

a = [11, 18, 9, 12, 23]
summa = 0
for ind in range(len(a)):
 # блок будет выполняться для каждого числа из [0; len(a)),
 summa += a[ind]
```

управление циклом

`break` - немедленный выход из цикла

`continue` - переход к следующей итерации цикла

✦ продвинутое объявление списков и словарей

```python
a = [i for i in range(1, 10)] # a = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Эквивалентно
a = []
for i in range(1, 10):
 a.append(i)

b = [el for el in a if el % 2 == 0] # создание списка b на основе
# a с использованием условия

# Эквивалентно
b = []
for el in a:
 if el % 2 == 0:
  b.append(el)
```

```python
# создание словаря с парами код ASCII:символ по этому коду
# chr(i) - возращаает символ по коду i
a = {chr(i):i for i in range(65, 91) if i % 2! = 1}
# Эквивалентно
a = {}
for i in range(65, 91):
 a[chr(i)] = i
```

✦ работа с файлами

```python
# open() - Открытие файла на чтение
a = open('путь к файлу + его имя')

# Важно: при указании пути слэши необходимо изменить на обратные
# слэши

# f.read(x) - считывание из файла f первые x символов (если не
# указан x, будет считан весь файл)
t = a.read()

# f.readline() - считывание следующей строки
t = a.readline()

# f.readlines() - считыавание все строк, в f - список из строк
# файла
t = a.readlines()
```

✦ функции

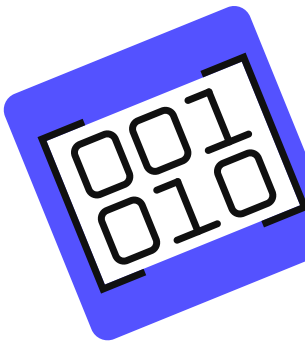
определение функции

```Python
def <имя функции> (<параметры, переданные в функцию>):
 # блок инструкций/операций
 return <результат> # конец работы и возрвращение значения
 # функции, по умолчанию - None
```

Параметры функции и блок инструкций существуют только во время выполнения функции

вызов функции

```Python
#определение функции
def our_sum(a, b):
 res = a+b
 return res

x = 9
y = 10
# вызов функции
g = our_sum(x,y) # g = 19
```



✦ библиотека itertools

Библиотека для упрощения работы с последовательностями символов и их генерациями

Для работы с библиотекой:

`import itertools` - в начале программы

функция products

```python
# Создает список всевозможных комбинаций элементов s, каждая
# комбинация имеет длину x
# Каждый элемент s может встречаться любое количество раз в каждой
# из комбинаций

# itertools.product(s,repeat=x)

a = '12'
b = list(itertools.product(a, repeat=3))

# в b список из элементов:
# ['1','1','1']
# ['1','1','2']
# ['1','2','1']
# ['1','2','2']
# ['2','1','1']
# ['2','1','2']
# ['2','2','1']
# ['2','2','2']
```

функция permutations

Python

```python
# Создает список всевозможных перестановок элементов s, каждая
# перестановка имеет длину x
# x не может быть больше длины s, если не указан x, длина
# перестановок равна длине s
# Каждый элемент s может встречаться в перестановках столько же раз,
# сколько и изначально в s или меньше

# itertools.permutations(s, x)

a = '123'
b = list(itertools.permutations(a))

# в b список из элементов:
# ['1','2','3']
# ['1','3','2']
# ['2','1','3']
# ['2','3','1']
# ['3','1','2']
# ['3','2','1']
```