Числа, переменные и функции

```
x = 2
y = x + 2
```

Арифметические операции:

```
    + -      *      /
```

// - деление нацело
% - остаток от деления
** - возведение в степень

Сокращенные операторы присвоения:

```
x = x + 1   ⇔   x += 1
x = x * 2   ⇔   x *= 2
```

и т. д.

Вывод на экран с помощью вызова функции print():

```
print(x, y)
```

Напишем собственную функцию:

```
def sqrt(x):
      return x ** 0.5

print(sqrt(2))
>>> 1.4142135623730951
```

Аргументы могут быть порядковыми и именованными.
Именованным можно задавать значения по умолчанию:

```
def my_func(x, y, text = 'x + y ='):
      print(text, x + y)
```

Границы контекста в python определяются отступами.

Строки

```
s = 'test'   или   s = "test"
```

Строки можно складывать и умножать на целое число:

```
s *= 2
print(s)
>>> testtest
```

Отдельные символы доступны по индексу,
но только для чтения:

```
print(s[0])
```

Функция len:

```
print(len('test'))
>>> 4
```

Помимо этого, для работы со строками есть множество *методов*, например:

```
s = 'кит'
s = s.replace('и', 'о')
print(s)
>>> 'кот'
```

Метод - это функция, определенная внутри типа данных.
Эти записи эквивалентны:

```
s = s.replace('и', 'о')
s = str.replace(s, 'и', 'о')
```

Преобразование типов и логический тип:

```
int(1.41) == 1
float(2) == 2.0
str(123) == '123'
int('123') == 123
bool('') == False
bool(0) == False
```

Логические операторы:

```
and, or, not
```

Условный оператор:

```
if x > 2:
     print(x, ' > 2')
else:
     print(x, ' <= 2')
```

Сравнение:

```
==    !=    >     <     >=    <=    is
```

Проверка типа:

```
type(x) is int
isinstance(x, int)
```

Список (list)

Список в python может хранить любые типы данных, даже вперемешку:

```
l = [1, 2, 3, 'test']
```

Список может быть пустым:

```
l = []
```

Элементы списка можно как читать, так и изменять по индексу:

```
l[0] == 1
l[0] = 10
```

Оператор in (работает и со строками):

```
print('test' in l)
>>> True

print('t' not in 'test')
>>> False
```

Методы списков:

Добавить в конец списка:

```
l.append(4)
```

Объединить списки:

```
l.extend([5, 6, 7])
```

Список можно использовать как LIFO стек:

```
x = l.pop()
print (x, l)
>>> 7 [10, 2, 3, 'test', 4, 5, 6]
```

Сортировка:

```
sorted(l) # возвращает новый список
l.sort()  # изменяет исходный список
```

(Работает, если все данные сравнимого типа.)

Обратный порядок:

```
sorted(l, reverse=True)
```

Циклы и функция range:

```
for i in range(10):
    print(i)
```

Итерация по элементам:

По индексу:

```
for i in range(len(l)):
     print(l[i])
```

По значениям:

```
for n in l:
     print(n)
```

С помощью функции enumerate:

```
for i, n in enumerate(l):
    print(i, n)
```

Кортеж (tuple)

Очень похож на список, но неизменяем:

```
t = (3, 1, 2)
print(sorted(t))
>>> [1, 2, 3]
```

Срезы

```
l = [1, 2, 3, 4, 5]
print(l[1:4])
>>> [2, 3, 4]

print(l[:-2])
>>> [1, 2, 3]

print(l[::-1])
>>> [5, 4, 3, 2, 1]
```

Распаковка

```
t = (1, 2)
x, y = t

t = (1, 2, 3, 4, 5)
x, y, *rest = t
print(rest)
>>> [3, 4, 5]
```

Словарь (dict)

```
d = {'name':'Max', 'age':22, 'height':183}
```

Записи доступны по ключу для чтения и изменения:

```
d['age'] = 23
```

Безопасная работа со словарями:

```
if 'city' in d:
      city = d['city']
else:
      city = 'Unknown'
```

эквивалентно:

```
city = d.get('city', 'Unknown')
```

```
if 'city' not in d:
      d[city] = 'Toronto'
city = d['city']
```

эквивалентно:

```
city = d.setdefault('city', 'Toronto')
```

Полезные итераторы:

```
d.keys()    d.values()  d.items()
```

```
for key, value in d.items():
      print(key, ':', value)
```

Включения (comprehensions)

Заполнение списка в цикле

```
 l = []
 for i in range(10):
      if i % 2 == 0:
            l.append(i ** 2)
      else:
            l.append(i)
```

Заменяется одной строчкой:

```
 l = [i ** 2 if i % 2 == 0 else i for i in range(10)]
```

Аналогично со словарем:

```
 d = {}
 for i in range(10):
      if i % 2 == 0:
            d[i] = i ** 2
      else:
            d[i] = i

 d = {i: i ** 2 if i % 2 == 0 else i for i in range(10)}
```

Lamdba-функции

Простую функцию тоже можно определить в одну строку:

```
def sqrt(x):
      return  x ** 0.5

sqrt = lambda x: x ** 0.5
```

Генераторы

Map

```
t = (1, 2, 3, 4)
for i in map(sqrt, t):
    print(i)

>>> 1.0
1.4142135623730951
1.7320508075688772
2.0

print(list(map(sqrt, t)))

>>> [1.0, 1.4142135623730951, 1.7320508075688772, 2.0,
2.23606797749979]
```

Filter

```
for i in filter(lambda x: x % 2 == 0, t):
    print(i)
>>> 2
4
```