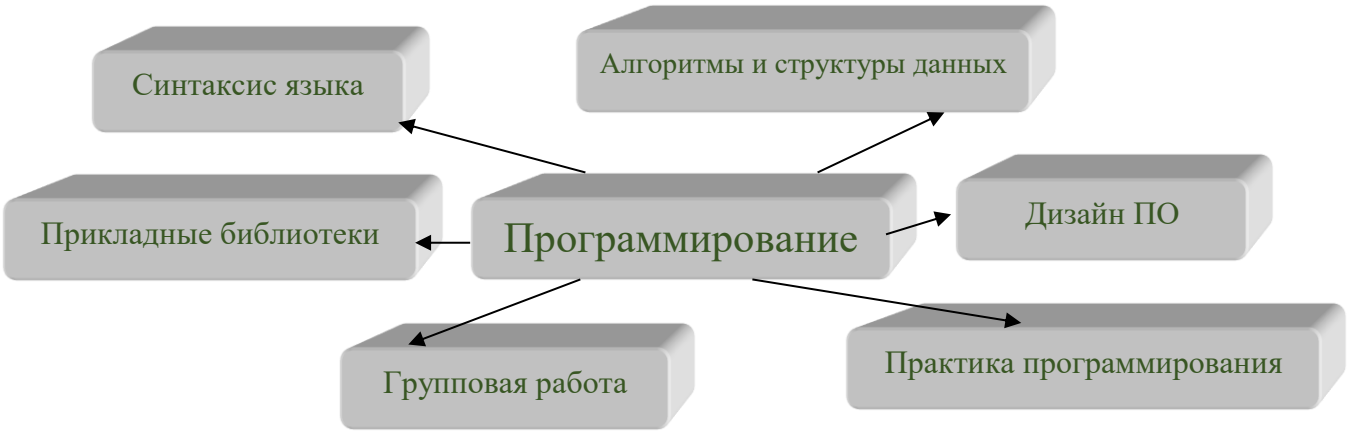
Раздаточный материл № 1



Раздаточный материал № 2

Java: int a = 1;
Python: a = 1

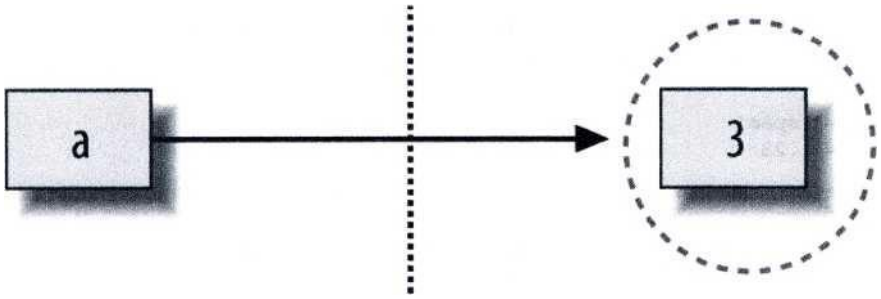
Раздаточный материл № 3

```
import keyword
…
print(keyword.iskeyword("NumberInt"))
```

False

Раздаточный материл № 4

a = 3



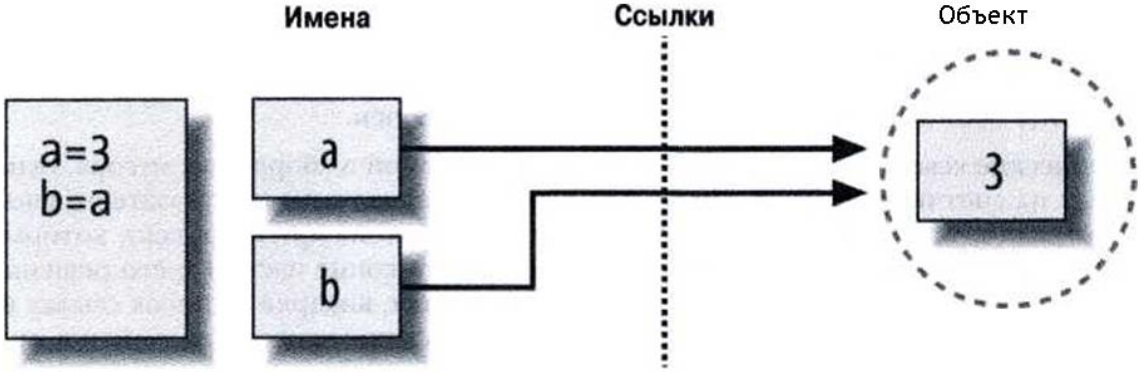
Раздаточный материл № 5

#1

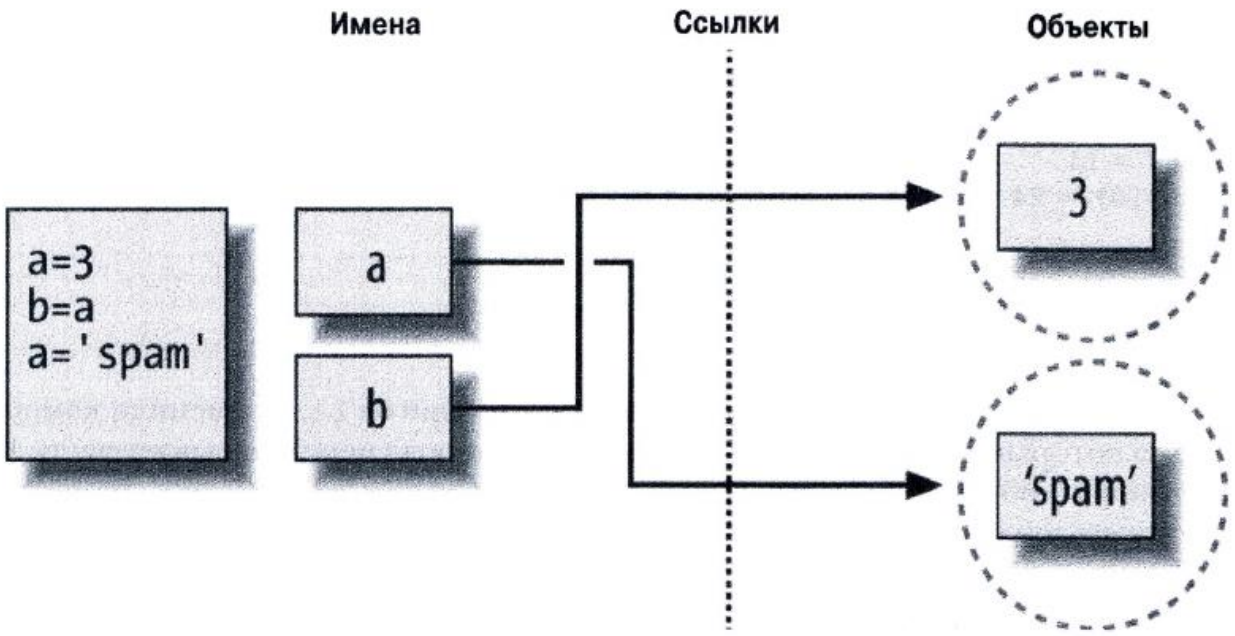
```
print(id(UserName))
```

11083840

Раздаточный материл № 6



Раздаточный материал № 7



Раздаточный материал № 8

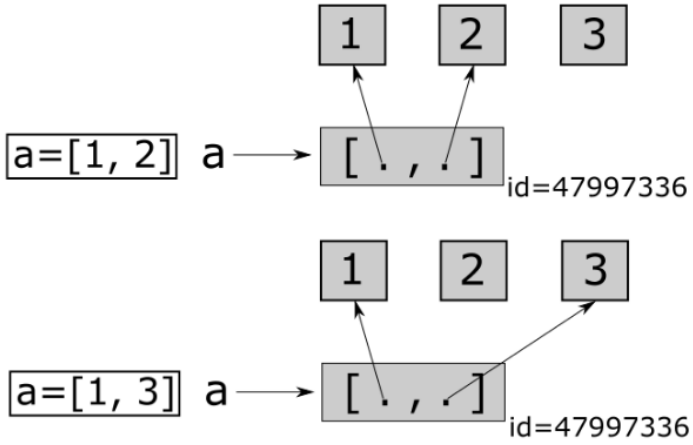
```
r = 15
print(id(r))→ 2012981392
r +=5.1
print(id(r))→ 1719872
```

Раздаточный материал № 9

создадим список [1, 2], а потом заменим второй элемент на 3.

```
>>> a = [1, 2]
>>> id(a)
47997336
>>> a[1] = 3
>>> a
[1, 3]
>>> id(a)
47997336
```

Объект, на который ссылается переменная a, был изменен. Это можно проиллюстрировать следующим рисунком.



Раздаточный материал № 10

Символ	Операция
==	Равно
!=	Не равно
<	Меньше
>	Больше
>=	Больше или равно
<=	Меньше или равно

Например,
x < y; a + b >= c/d; abs (m — n) <= 1.

Примеры вычисления значений отношений:

Отношение	Результат
12 >= 12	True
56 > 10	True
11 <= 6	False

Раздаточный материал № 11

and («и») - логическое умножение,
or («или») - логическое сложение,
not («не») - логическое отрицание.
Действие логических операций and, or и not определяется с помощью таблиц истинности.

Раздаточный материал № 12

1. Вычислить значения следующих логических выражений:

а) K % 7 == K // 5 - 1 при K = 15;
б) t And (P % 3 == 0) при t = True, P = 10101;
в) (x * y != 0) And (y > x) при x = 2, y = 1;
г) a Or Not b при a = False, b = True.

2. Написать оператор присваивания, в результате выполнения которого логическая переменная t получит значение True, если следующее утверждение истинно, и значение False — в противном случае:

а) из чисел x, y, z только два равны между собой;
б) x— положительное число;
в) каждое из чисел x, y, z положительное;
г) только одно из чисел x, y, z положительное;
д) p делится без остатка на q;
е) цифра 5 входит в десятичную запись трехзначного целого числа k.

Раздаточный материал № 13

1. None (неопределенное значение переменной)
2. Логические переменные (Boolean Type)
3. Числа (Numeric Type)
   - int – целое число
   - float – число с плавающей точкой
   - complex – комплексное число
4. Списки (Sequence Type)
   - list – список
   - tuple – кортеж
   - range – диапазон
5. Строки (Text Sequence Type)
   - str
6. Бинарные списки (Binary Sequence Types)
   - bytes – байты
   - bytearray – массивы байт
   - memoryview – специальные объекты для доступа к внутренним данным объекта через protocol buffer
7. Множества (Set Types)
   - set – множество
   - frozenset – неизменяемое множество

8. Словари (Mapping Types)
   dict – словарь
9. Типы программных единиц
   функции
   модули
   классы
10. Типы, связанные с реализацией
   скомпилированный код
   трассировка стека

Раздаточный материал № 14

- целые числа (тип int) – положительные и отрицательные целые числа, а также 0(например, 4, 687, 45, 0).

– числа с плавающей точкой (тип float) – дробные, они же вещественные, числа (например, 1.45, 3.789654, 0.00453). Примечание: для разделения целой и дробной частей здесь используется точка, а не запятая.

Раздаточный материал № 15 (справочно)

Встроенные математические функции Python доступные без подключения модулей.

abs(x) - возвращает модуль числа. Аргумент x может быть целым (int) или вещественным (float) числом.

pow(base, exp[, mod]) - возвращает base в степени exp. Допустима отрицательная и вещественная степень. Если указан третий аргумент mod, функция вернёт остаток по модулю.

divmod(a, b) - для целых аргументов возвращается кортеж с целочисленным результатом деления и остатком от деления.

round(number[, ndigits]) - возвращает число округлённое с точностью ndigits знаков после запятой. Если ndigits пропущено или равно None, функция возвращает ближайшее к number целое число.

oct(x) - конвертирует целое число в строку с восьмеричным числом с префиксом "0o".

bin(x) - конвертирует целое число в строку с двоичным числом с префиксом "0b".

hex(x) - конвертирует целое число в строку с шестнадцатеричным числом с префиксом "0x".

Раздаточный материал № 16

```
# Пример 1
a,b = 5,7 # позиционное присваивание кортежей
>>> a,b
(5, 7)
>>> a,b = b,a # обмен значениями
>>> a,b
(7, 5)

# Пример 2
x=[0,1,2,3]
>>> i=0
>>> i,x[i]=2,6
>>> x
[0, 1, 6, 3]
```

Раздаточный материал № 18

```
UserName = input('Введите имя')
```

Раздаточный материал № 19

```
NumberInt = int(input('Введите первое число:   '))

NumberFloat = float(input('Введите второе число:   '))
```

Раздаточный материал № 20

```
print ( [object, …] [,sep='  '] [, end='\n’][,  file=sys .stdout] [, flush=False] )
```

Раздаточный материал № 21

```
#1
>>> print("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun", sep="-")
Mon-Tue-Wed-Thu-Fri-Sat-Sun
>>> print(1, 2, 3, sep="//")
1//2//3

#2
>>> print(10, end='\n\n')
10

>>>

#3
# \n перенесет каждое слово на новую строку
print('лекция', 'по', 'функции', 'print()', sep='\n')

лекция
по
функции
print()

#4
print('лекция', 'по', 'функции', 'print()', sep=',')

лекция,по,функции,print()

#5
print('лекция', 'по', 'функции', 'print()', sep=',+')

лекция,+по,+функции,+print()
```

Раздаточный материал № 22

```
#1
if n < 100:
        b = n + a #отступ является обязательным, т.к. формирует тело условного оператора
print(b) # оператор print не является телом условного оператора

#2
tovar1 = 50
tovar2 = 32
if tovar1 + tovar2 > 9 :
        print("99 рублей недостаточно")
else:
        print("Чек оплачен")
```

#3

```
a = 5 > 0 # подвыражение 5 > 0 выполнится первым, после чего его результат будет присвоен
            переменной a
if a:
        print(a)

if a > 0 and a < b:
        print(b - a)

if 0 < a < b:
        print(b - a)
```

Раздаточный материал № 23

```
old = int(input('Ваш возраст: '))
print('Рекомендовано:', end=' ')
if 3 <= old < 6:
        print('"Заяц в лабиринте"')
elif 6 <= old < 12:
        print('"Марсианин"')
elif 12 <= old < 16:
        print('"Загадочный остров"')
elif 16 <= old:
        print('"Поток сознания"')
```

Раздаточный материал № 24

```
a if условие else b
```

Раздаточный материал № 25

```
a = 50
b = 100
c = 40
max = a if a > b else b
max = c if c > max else max
print(max)
```

Раздаточный материал № 26

```
# Дано целое число. Если оно является положительным,
# то прибавить к нему 20, в противном случае вычесть из него 5/
# Результат не сохраняется

c = int(input('Введи число:     '))
print('Результат =  ', c + 20 if c >= 0 else c - 5)

# Дано целое число. Если оно является положительным,
# то прибавить к нему 20, в противном случае вычесть из него 5/
# Результат сохраняется

c = int(input('Введи число:     '))
f = c + 20 if c >= 0 else c - 5
print('Результат =  ', f)
```

Раздаточный материал № 27

```
while проверка:
        операторы
        if проверка: break          # Выход из цикла с пропуском else, если есть
        if проверка: continue       # Переход на проверку в начале цикла
else:
        операторы                   # Выполняется, если не было break
```

Блок else цикла выполняется тогда и только тогда, когда происходит нормальный выход из цикла (т.е. без выполнения оператора break).

Раздаточный материал № 28

```
def fund () : pass      # Позже поместить сюда реальный код
def func2(): pass
```

Раздаточный материал № 29

```
def func1(): ...
func1()             # При вызове ничего не делает
```

Раздаточный материал № 30

```
t = 10
while t:
    t -= 1
    if t % 2 != 0: continue # пропуск нечетных чисел
    print(t, end=' ')
```

Раздаточный материал № 31

```
while True:
    name = input('Enter name: ')
    if name == 'stop': break # при вводе stop – выход из цикла
    age = input ('Enter age: ')
    print('Hello', name, '=>', int(age) ** 2)
```

Раздаточный материал № 32

```
try:
    n = int(input("Введите целое число: "))
    print("Удачно")
except:
    print("Что-то пошло не так")
```

Раздаточный материал № 33

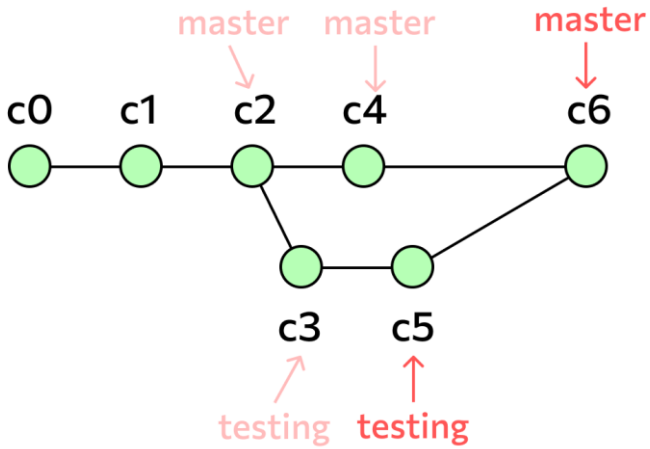
```
try:
    n = int(input("Введите целое число: "))
    print("Удачно")
except ValueError:
    print("Что-то пошло не так")
```

Раздаточный материал № 34

```
n = input("Введите целое число: ")
while type(n) != int:
    try:
        n = int(n)
    except ValueError:
      print("Неправильно ввели!")
      n = input("Введите целое число: ")

if  not(math.fmod(n, 2)) :
        print("Четное")
else:
        print("Нечетное")
```

Раздаточный материал № 35



Раздаточный материал № 36

```
def countFish():
        a = int(input())
        b = int(input())
        print("Всего", a+b, "шт.")
```

Раздаточный материал № 37

```
        # программа выводит SOS

        def CharS():
            print('S', end='')


        def CharO():
            print('O', end='')


        CharS()
        CharO()
        CharS()
```

Раздаточный материал № 38

```
def rectangle():
    a = float(input("Ширина %s: " % figure)) # обращение к глобальной
    b = float(input("Высота %s: " % figure)) # переменной figure
    print("Площадь: %.2f" % (a*b))
def triangle():
    a = float(input("Основание %s: " % figure))
    h = float(input("Высота %s: " % figure))
    print("Площадь: %.2f" % (0.5 * a * h))
figure = input("1-прямоугольник, 2-треугольник: ")
if figure == '1':
    rectangle()
elif figure == '2':
    triangle()
```

Раздаточный материал № 39

```
# В основной ветке программы вызывается функция cylinder(), которая вычисляет
площадь
# цилиндра. В теле cylinder() определена функция circle(), вычисляющая площадь
круга по
# формуле πr2. В теле cylinder() у пользователя спрашивается, хочет ли он получить
только
# площадь боковой поверхности цилиндра, которая вычисляется по формуле 2πrh, или
полную
# площадь цилиндра. В последнем случае к площади боковой поверхности цилиндра
должен
# добавляться удвоенный результат вычислений функции circle().

SC = 0
SQ = 0


def cylinder():
    r = float(input('Введи радиус:  '))

    def circle():
        SC = 3.14 * r * 2
        return SC

    c = input('1 - площадь боковой поверхности цилиндра, 2 - полная площадь
цилиндра:  ')
    if c == '1':
        print(circle())
    elif c == '2':
        h = float(input('Введи высоту:  '))
        SQ = 2 * 3.14 * r * h + 2 * circle()
        print(SQ)


cylinder()
```

Раздаточный материал № 40

Port = `cylinder()`

Раздаточный материал № 41

```
def duble():
    width = float(input('Введи ширину:    '))
    height = float(input('Введи высоту:    '))
    ploch = width * height
    perim  = 2 * (width + height)
    return ploch, perim

g_ploch, g_perim = duble()
print('Полощадь прямоугольника:  ', g_ploch)
print('Периметр прямоугольника:  ', g_perim)
```

Раздаточный материал № 42

```
print(duble())

Введи ширину:   10
Введи высоту:   20
(200.0, 60.0) – скобки говорят, что выводится кортеж
```

Раздаточный материал № 43

```
def duble(a, b):
    ploch = a * b
    perim  = 2 * (a + b)
    return ploch, perim

width = float(input('Введи ширину:    '))
height = float(input('Введи высоту:    '))
g_ploch, g_perim = duble(width, height)
print('Площадь прямоугольника:  ', g_ploch)
print('Периметр прямоугольника:  ', g_perim)
```

Раздаточный материал № 44

```
# 1
def duble(a, b=20):
    ploch = a * b
    perim  = 2 * (a + b)
    return ploch, perim

width = float(input('Введи ширину:    '))
g_ploch, g_perim = duble(width)
print('Площадь прямоугольника:  ', g_ploch)
print('Периметр прямоугольника:  ', g_perim)

# 2
def duble(a, b=20):
    ploch = a * b
    perim  = 2 * (a + b)
    return ploch, perim

width = float(input('Введи ширину:    '))
height = float(input('Введи высоту:    '))
g_ploch, g_perim = duble(width, height)
print('Площадь прямоугольника:  ', g_ploch)
print('Периметр прямоугольника:  ', g_perim)
```

Раздаточный материал № 45

```
def duble(a, b):
    ploch = a * b
    perim = 2 * (a + b)
    return ploch, perim
```

```
g_ploch, g_perim = duble(b=20, a=10)
print('Площадь прямоугольника:  ', g_ploch)
print('Периметр прямоугольника:  ', g_perim)
```

Раздаточный материал № 46

```
def oneOrMany(*a):
    print(a)

oneOrMany(1)
oneOrMany('1', 1, 2, 'abc')
oneOrMany()
```

Результат:
(1,)
('1', 1, 2, 'abc')
()

Раздаточный материал № 47 (справочно)

abs() - возвращает абсолютное значение числа. Если это комплексное число, то абсолютным значением будет величина целой и мнимой частей.

chr() - возвращает строку, представляющую символ Unicode для переданного числа. Она является противоположностью ord(), которая принимает символ и возвращает его числовой код.

callable() - сообщает, является ли объект вызываемым. Если да, то возвращает True, а в противном случае — False. Вызываемый объект — это объект, который можно вызвать.

```
>>> callable(5)
False
```

round() - округляет вещественное число до определенного знака после запятой. Если второй аргумент не задан, то округление идет до целого числа. Второй аргумент может быть отрицательным числом. В этом случае округляться начинают единицы, десятки, сотни и т. д., то есть целая часть.

```
>>> a = 10/3
>>> a
3.3333333333333335
>>> round(a,2)
3.33
>>> round(a)
3

>>> round(5321, -1)
5320
>>> round(5321, -3)
5000
>>> round(5321, -4)
10000
```

divmod() - выполняет одновременно деление нацело и нахождение остатка от деления. Возвращает кортеж.

```
>>> divmod(10, 3)
(3, 1)
>>> divmod(20, 7)
(2, 6)
```

pow() - возводит в степень. Первое число – основание, второе – показатель. Может принимать третий необязательный аргумент - это число, на которое делится по модулю результат возведения в степень.

```
>>> pow(3, 2)
9
>>> pow(2, 4)
16

>>> pow(2, 4, 4)
0
>>> 2**4 % 4
0
```

dict() - используется для создания словарей. Это же можно делать и вручную, но функция предоставляет большую гибкость и дополнительные возможности. Например, ей в качестве параметра можно передать несколько словарей, объединив их в один большой.

```
>>> dict({"a":1, "b":2}, c = 3)
{'a': 1, 'b': 2, 'c': 3}

>>> list = [["a",1],["b",2]]
>>> dict(list)
{'a': 1, 'b': 2}
```

dir() - получает список всех атрибутов и методов объекта. Если объект не передать, то функция вернет все имена модулей в локальном пространстве имен.

```
>>> x = ["Яблоко", "Апельсин", "Гранат"]
>>> print(dir(x))
['__add__', '__class__', '__contains__',....]
```

enumerate() - в качестве параметра эта функция принимает последовательность. После этого она перебирает каждый элемент и возвращает его вместе со счетчиком в виде перечисляемого объекта. Основная особенность таких объектов — возможность размещать их в цикле для перебора.

```
>>> x = "Строка"
>>> list(enumerate(x))
[(0, 'С'), (1, 'т'), (2, 'р'), (3, 'о'), (4, 'к'), (5, 'а')]
```

eval() - обрабатывает переданное в нее выражение и исполняет его как выражение Python. После этого возвращается значение. Чаще всего эта функция используется для выполнения математических функций.

```
>>> eval('2+2')
4
>>> eval('2*7')
14
>>> eval('5/2')
2.5
```

filter() - функция используется для перебора итерируемых объектов и последовательностей, таких как списки, кортежи и словари. Но перед ее использованием нужно также иметь подходящую функцию, которая бы проверяла каждый элемент на валидность. Если элемент подходит, он будет возвращаться в вывод.

```
list1 = [3, 5, 4, 8, 6, 33, 22, 18, 76, 1]
result = list(filter(lambda x: (x%2 != 0) , list1))
```

```
print(result)
```

float() - конвертирует число или строку в число с плавающей точкой и возвращает результат. Если из-за некорректного ввода конвертация не проходит, возвращаются ValueError или TypeError.

hash() - у большинства объектов в Python есть хэш-номер. Функция hash() возвращает значение хэша переданного объекта. Объекты с __hash__() — это те, у которых есть соответствующее значение.

```
>>> hash('Hello World')
-2864993036154377761
>>> hash(True)
1
```

help() - предоставляет простой способ получения доступа к документации Python без интернета для любой функции, ключевого слова или модуля.

```
>>> help(print)
Help on built-in function print in module builtins:
```

int() - функция возвращает целое число из объекта, переданного в параметра. Она может конвертировать числа с разным основанием (шестнадцатеричные, двоичные и так далее) в целые.

```
>>> int(5.6)
5

>>> int('0101', 2)
5
```

iter() - принимает объект и возвращает итерируемый объект. Сам по себе он бесполезен, но оказывается крайне эффективным при использовании в циклах for и while. Благодаря этому объект можно перебирать по одному свойству за раз.

```
>>> lis = ['a', 'b', 'c', 'd', 'e']
>>> x = iter(lis)
>>> next(x)
'a'
>>> next(x)
'b'
>>> next(x)
'c'
>>> next(x)
'd'
```

max() - функция используется для нахождения «максимального» значения в последовательности, итерируемом объекте и так далее. В параметрах можно менять способ вычисления максимального значения.

```
>>> max('a', 'A')
'a'

>>> x = [5, 7, 8, 2, 5]
>>> max(x)
8

>>> x = ["Яблоко", "Апельсин", "Автомобиль"]
>>> max(x, key = len)
'Яблоко'
```

min() - функция используется для нахождения «минимального» значения в последовательности, итерируемом объекте и так далее. В параметрах можно менять способ вычисления минимального значения.

```
>>> min('a','A')
'A'

>>> x = [5, 7, 8, 2, 5]
>>> min(x)
2

>>> x = ["Виноград", "Манго", "Фрукты", "Клубника"]
>>> min(x)
'Виноград'
```

len() - функция используется для вычисления длины последовательности или итерируемого объекта.

```
>>> x = (2, 3, 1, 6, 7)
>>> len(x)
5
>>> len("Строка")
6
```

list() - в качестве параметра функция list() принимает итерируемый объект и возвращает список. Она обеспечивает большие гибкость и скорость при создании списков по сравнению с обычным способом.

```
>>> list("Привет")
['П', 'р', 'и', 'в', 'е', 'т']

>>> list({1:"a", 2:"b", 3:"c"})
[1, 2, 3]
```

map() - используется для применения определенной функции к итерируемому объекту. Она возвращает результат в виде итерируемого объекта (списки, кортежи, множества). Можно передать и несколько объектов, но в таком случае нужно будет и соответствующее количество функций.

```
>>> def inc(x):
        x = x + 1
        return x

>>> lis = [1,2,3,4,5]
>>> result = map(inc,lis)

>>> for x in result:
        print(x)

2
3
4
5
6
```

next() - используется для итерируемых объектов. Умеет получать следующий (next) элемент в последовательности. Добравшись до конца, выводит значение по умолчанию.

```
>>> lis = ['a', 'b', 'c', 'd', 'e']
>>> x = iter(lis)
>>> next(x)
'a'
>>> next(x)
'b'
>>> next(x)
'c'
>>> next(x)
'd'
```

ord() - принимает один символ или строку длиной в один символ и возвращает соответствующее значение Unicode. Например, ord("a") вернет 97, а 97 — a.

```
>>> ord('a')
97

>>> ord('A')
65
```

reversed() - предоставляет простой и быстрый способ развернуть порядок элементов в последовательности. В качестве параметра она принимает валидную последовательность, например, список, а возвращает итерируемый объект.

```
>>> x = [3,4,5]
>>> b = reversed(x)
>>> list(b)
[5, 4, 3]
```

range() - используется для создания последовательности чисел с заданными значениями от и до, а также интервалом. Такая последовательность часто используется в циклах, особенно в цикле for.

```
>>> list(range(10,20,2))
[10, 12, 14, 16, 18]
```

reduce() - выполняет переданную в качестве аргумента функцию для каждого элемента последовательности. Она является частью functools, поэтому перед ее использованием соответствующий модуль нужно импортировать.

```
>>> list1 = [2, 5, 3, 1, 8]
>>> functools.reduce(operator.add,list1)
19

>>> list1 = [2, 5, 3, 1, 8]
>>> functools.reduce(operator.mul,list1)
240

>>> list1 = [2, 5, 3, 1, 8]
>>> functools.reduce(operator.truediv,list1)
0.016666666666666666
```

sorted() - используется для сортировки последовательностей значений разных типов. Например, может отсортировать список строк в алфавитном порядке или список числовых значений по возрастанию или убыванию.

```
>>> X = [4, 5, 7, 3, 1]
```

```
>>> sorted(X)
[1, 3, 4, 5, 7]
```

str() - используется для создания строковых представлений объектов, но не меняет сам объект, а возвращает новый. У нее есть встроенные механизмы кодировки и обработки ошибок, которые помогают при конвертации.

```
>>> str(5)
'5'

>>> X = [5,6,7]
>>> str(X)
'[5, 6, 7]'
```

set() - используется для создания наборов данных, которые передаются в качестве параметра. Обычно это последовательность, например, строка или список, которая затем преобразуется в множество уникальных значений.

```
>>> set()
set()

>>> set("Hello")
{'e', 'l', 'o', 'H'}

>>> set((1,2,3,4,5))
{1, 2, 3, 4, 5}
sum() - автоматически суммирует все элементы и возвращает сумму.
>>> x = [1, 2, 5, 3, 6, 7]
>>> sum(x)
24
```

tuple() - принимает один аргумент (итерируемый объект), которым может быть, например, список или словарь, последовательность или итератор и возвращает его в форме кортежа. Если не передать объект, то вернется пустой кортеж.

```
>>> tuple("Привет")
('П', 'р', 'и', 'в', 'е', 'т')

>>> tuple([1, 2, 3, 4, 5])
(1, 2, 3, 4, 5)
```

type() - применяется в двух сценариях. Если передать один параметр, то она вернет тип этого объекта. Если же передать три параметра, то можно создать объект type.

```
>>> type(5)
<class 'int'>

>>> type([5])
<class 'list'>
```

Раздаточный материал № 48

```
>>> import math
```

Раздаточный материал № 49

```
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor',
'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite',
'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf',
'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

Раздаточный материал № 50

```
>>> math.pow(2, 2)
4.0
>>> math.pi
3.141592653589793
```

Раздаточный материал № 51

```
>>> help(math.gcd)
```

Раздаточный материал № 52

```
>>> from math import gcd, sqrt, hypot
```

Раздаточный материал № 53

```
>>> from math import *
```

Раздаточный материал № 54

```
>>> pi = 3.14
>>> from math import pi
>>> pi
3.141592653589793
```

Раздаточный материал № 55

```
>>> from math import pi as P
>>> P
3.141592653589793
>>> pi
3.14
```

Раздаточный материал № 56 (справочно)

Модуль Math

math.ceil(X) – округление до ближайшего большего числа.

math.copysign(X, Y) - возвращает число, имеющее модуль такой же, как и у числа X, а знак - как у числа Y.

math.fabs(X) - модуль X.

math.factorial(X) - факториал числа X.

math.floor(X) - округление вниз.

math.fmod(X, Y) - остаток от деления X на Y.

math.frexp(X) - возвращает мантиссу и экспоненту числа.

math.ldexp(X, I) - X * 2i. Функция, обратная функции math.frexp().

math.fsum(последовательность) - сумма всех членов последовательности. Эквивалент встроенной функции sum(), но math.fsum() более точна для чисел с плавающей точкой.

math.isfinite(X) - является ли X числом.

math.isinf(X) - является ли X бесконечностью.
math.isnan(X) - является ли X NaN (Not a Number - не число).
math.modf(X) - возвращает дробную и целую часть числа X. Оба числа имеют тот же знак, что и X.
math.trunc(X) - усекает значение X до целого.
math.exp(X) - eX.
math.expm1(X) - eX - 1. При X → 0 точнее, чем math.exp(X)-1.
math.log(X, [base]) - логарифм X по основанию base. Если base не указан, вычисляется натуральный логарифм.
math.log1p(X) - натуральный логарифм (1 + X). При X → 0 точнее, чем math.log(1+X).
math.log10(X) - логарифм X по основанию 10.
math.log2(X) - логарифм X по основанию 2. Новое в Python 3.3.
math.pow(X, Y) - X^Y.
math.sqrt(X) - квадратный корень из X.
math.acos(X) - арккосинус X. В радианах.
math.asin(X) - арксинус X. В радианах.
math.atan(X) - арктангенс X. В радианах.
math.atan2(Y, X) - арктангенс Y/X. В радианах. С учетом четверти, в которой находится точка (X, Y).
math.cos(X) - косинус X (X указывается в радианах).
math.sin(X) - синус X (X указывается в радианах).
math.tan(X) - тангенс X (X указывается в радианах).
math.hypot(X, Y) - вычисляет гипотенузу треугольника с катетами X и Y (math.sqrt(x * x + y * y)).
math.degrees(X) - конвертирует радианы в градусы.
math.radians(X) - конвертирует градусы в радианы.
math.cosh(X) - вычисляет гиперболический косинус.
math.sinh(X) - вычисляет гиперболический синус.
math.tanh(X) - вычисляет гиперболический тангенс.
math.acosh(X) - вычисляет обратный гиперболический косинус.
math.asinh(X) - вычисляет обратный гиперболический синус.
math.atanh(X) - вычисляет обратный гиперболический тангенс.
math.erf(X) - функция ошибок.
math.erfc(X) - дополнительная функция ошибок (1 - math.erf(X)).
math.gamma(X) - гамма-функция X.
math.lgamma(X) - натуральный логарифм гамма-функции X.
math.pi - pi = 3,1415926...
math.e - e = 2,718281...

Раздаточный материал № 57

```
>>> import random
>>> from random import random, randrange, randint
```

Раздаточный материал № 58

```
>>> random.randint(0, 10)
10
```
или (если импортировались отдельные функции):
```
>>> randint(-100, 200)
-10
```

Раздаточный материал № 59

randrange(10, 20, 3) → "случайное" число будет выбираться из чисел 10, 13, 16, 19

Раздаточный материал № 60

```
>>> random.random()
0.17855729241927576
```
или
```
>>> random()
0.025328854415995194
```

Для округления результата
```
>>> round(random.random(), 3)
0.629
```

Раздаточный материал № 61

```
>>> random.random() * 10
2.510618091637596
>>> random.random() * (1 + 1) - 1
-0.673382618351051
```

Раздаточный материал № 62

```
>>> a = [12, 3.85, "black", -4]

>>> a

[12, 3.85, 'black', -4]
```

Раздаточный материал № 63

```
>>> a[0]
12
>>> a[3]
-4
```

Раздаточный материал № 64

```
>>> a[0:2]
[12, 3.85]
```

Раздаточный материал № 65

```
>>> a[:3]
[12, 3.85, 'black']
>>> a[2:]
['black', -4]
>>> a[:]
[12, 3.85, 'black', -4]
```

Раздаточный материал № 66

```
>>> a[1] = 4
>>> a
[12, 4, 'black', -4]
```

Раздаточный материал № 67

```
>>> a = [1, 3, 5, 7]
```

```
>>> b = a[:]
>>> print(a)
[1, 3, 5, 7]
>>> print(b)
[1, 3, 5, 7]
```

Раздаточный материал № 68 (справочно)

list.append(значение) – добавление нового значения в конец списка.
list.insert(позиция, значение) - добавление нового значения в указанную позицию списка.
list.remove(значение) – удаляет указанное значение из списка, не привязываясь к индексу.
list.pop() – удаляет последний элемент списка и возвращает значение.
list.pop(индекс) – удаляет элемент списка с указанным индексом и возвращает значение.
del a[индекс] - удаляет элемент списка с указанным индексом.
del a[индекс : индекс] - удаляет срез элементов списка с указанными индексами.
list.clear() – удаляет все элементы из списка.
list.index - Возвращает индекс элемента
list.count(x) Возвращает количество вхождений элемента x в список
list.sort(key=None, reverse=False) - сортирует элементы в списке по возрастанию. Для сортировки в обратном порядке используйте флаг reverse=True.
ist.reverse() - изменяет порядок расположения элементов в списке на обратный.
list.copy() - возвращает копию списка.

Раздаточный материал № 69

```
for цель in объект:          # Присваивает цели элементы объекта
        операторы            # Повторяемое тело цикла: использует цель
else:                        # Необязательная часть else
        операторы            # Если не встречался оператор break
```

```
spisok = [10, 40, 20, 30]
>>> for element in spisok:
        print(element + 2)
12
42
22
32
```

Раздаточный материал № 70

```
for цель in объект:                    #Присваивает цели элементы объекта
        операторы if проверка: break   #Выход из цикла с пропуском else
        if проверка: continue                  #Переход в начало цикла
else:
        операторы                      #Если не встречался оператор break
```

Раздаточный материал № 71

```
>>> s = "Hello, World!"
>>> s[0]
'H'
>>> s[7:]
'World!'
>>> s[::2]     # здесь извлечение идет с шагом = 2
'Hlo ol!'
```

Раздаточный материал № 72

```
>>> s = s[0:-1] + '.'
>>> s
'Hello, World.'         # старое значение s теряется
```

Раздаточный материал № 73

obj.foo(<args>)

Этот код вызывает метод .foo() объекта obj. <args> — аргументы, передаваемые методу (если есть).

Раздаточный материал № 74 (справочно)

Строковые операторы	

+ - конкатенация строк	>>> s = 'py'
	>>> t = 'th'
	>>> u = 'on'
	>>> s + t + u
	'python'
	>>> print('Привет, ' + 'Мир!')

* - умножение строк. Значение множителя должно быть целым положительным числом	>>> s = 'py.'
	>>> s * 4
	'py.py.py.py.'

in - оператор принадлежности подстроки, возвращает True, если подстрока входит в строку, и False, если нет.
Есть также оператор not in, у которого обратная логика
	>>> s = 'Python'
	>>> s in 'I love Python.'
	True
	>>> s in 'I love Java.'
	False

Встроенные функции строк	

split() позволяет разбить строку по пробелам. В результате получается список слов.
Может принимать необязательный аргумент-строку, указывающей по какому символу или подстроке следует выполнить разделение
	>>> s = input()
	red blue orange white
	>>> s
	'red blue orange white'
	>>> sl = s.split()
	>>> sl
	['red', 'blue', 'orange', 'white']
	>>> s
	'red blue orange white'
	
	>>> s.split('e')
	['r', 'd blu', ' orang', ' whit', '']
	>>> '40030023'.split('00')
	['4', '3', '23']

Метод строк join() выполняет обратное действие. Он формирует из списка строку. Поскольку это метод строки, то впереди ставится строка-разделитель, а в скобках — передается список.
Если разделитель не нужен, то метод применяется к пустой строке
	>>> '-'.join(sl)
	'red-blue-orange-white'
	
	>>> ''.join(sl)
	'redblueorangewhite'

find() ищет подстроку в строке и возвращает индекс первого элемента найденной подстроки. Если подстрока не найдена, то возвращает -1. Поиск может производиться не во всей строке, а лишь на каком-то ее отрезке. В этом случае указывается первый и последний индексы отрезка. Если последний не указан, то ищется	>>> s
	'red blue orange white'
	>>> s.find('blue')
	4
	>>> s.find('green')
	-1

до конца строки. Метод find() возвращает только первое вхождение.	>>> letters = 'ABCDACFDA'
	>>> letters.find('A', 3)
	4
	>>> letters.find('DA', 0, 6)
	3
	# Поиск идет с третьего индекса и до конца, а также с первого и до шестого

replace() заменяет одну подстроку на другую	>>> letters.replace('DA', 'NET')
	'ABCNETCFNET'
	
	Исходная строка не меняется:
	>>> letters
	'ABCDACFDA'
	
	если результат надо сохранить, то его надо присвоить переменной
	>>> new_letters = letters.replace('DA', 'NET')
	>>> new_letters
	'ABCNETCFNET'

ord(c) возвращает числовое значение для заданного символа	>>> ord('a')
	97
	>>> ord('#')
	35

chr(n) возвращает символьное значение для данного целого числа.	>>> chr(8364)
	'€'
	>>> chr(8721)
	'∑'

len(s) возвращает длину строки	>>> s = 'Простая строка.'
	>>> len(s)
	15

str(obj) возвращает строковое представление объекта	>>> str(49.2)
	'49.2'
	>>> str(3+4j)
	'(3+4j)'
	>>> str(3 + 29)
	'32'
	>>> str('py')
	'py'

Встроенные методы строк	

string.capitalize() приводит первую букву в верхний регистр, остальные в нижний.	>>> s = 'everyTHing yoU Can IMaGine is rEAl'
	>>> s.capitalize()
	'Everything you can imagine is real'

string.lower() преобразует все буквенные символы в строчные.	>>> 'everyTHing yoU Can IMaGine is rEAl'.lower() 'everything you can imagine is real'

string.swapcase() меняет регистр буквенных символов на противоположный.	>>> 'the sun also rises'.title()
	'The Sun Also Rises'
	
	>>> 'follow us @PYTHON'.title()
	'Follow Us @Python'

string.upper() преобразует все буквенные символы в заглавные.	>>> 'follow us @PYTHON'.upper() 'FOLLOW US @PYTHON'
string.count(<sub>[, <start>[, <end>]]) подсчитывает количество вхождений подстроки в строку. s.count(<sub>) возвращает количество точных вхождений подстроки <sub> в s:	>>> 'foo goo moo'.count('oo') 3

Количество вхождений изменится, если указать <start> и <end>	>>> 'foo goo moo'.count('oo', 0, 8) 2

string.endswith(<suffix>[, <start>[, <end>]])
определяет, заканчивается ли строка заданной подстрокой
s.endswith(<suffix>) возвращает, True если s заканчивается указанным <suffix> и False если нет
Сравнение ограничено подстрокой, между <start> и <end>, если они указаны
	>>> 'python'.endswith('on')
	True
	>>> 'python'.endswith('or')
	False
	>>> 'python'.endswith('yt', 0, 4)
	True
	>>> 'python'.endswith('yt', 2, 4)
	False

string.find(<sub>[, <start>[, <end>]]) ищет в строке заданную подстроку
s.find(<sub>) возвращает первый индекс в s который соответствует началу строки <sub>
Этот метод возвращает, -1 если указанная подстрока не найдена
Поиск в строке ограничивается подстрокой, между <start> и <end>, если они указаны
	>>> 'Follow Us @Python'.find('Us')
	7
	>>> 'Follow Us @Python'.find('you')
	-1
	>>> 'Follow Us @Python'.find('Us', 4)
	7
	>>> 'Follow Us @Python'.find('Us', 4, 7)
	-1

s.rfind(<sub>) возвращает индекс последнего вхождения подстроки <sub> в s, который соответствует началу <sub>. Как и в .find(), если подстрока не найдена, возвращается -1. Поиск в строке ограничивается подстрокой, между <start> и <end>, если они указаны.	>>> 'Follow Us @Python'.rfind('o') 15

string.isalnum() определяет, состоит ли строка из букв и цифр, возвращает True, если строка s не пустая, а все ее символы буквенно-цифровые (либо буква, либо цифра). В другом случае False	>>> 'abc123'.isalnum()
	True
	>>> 'abc$123'.isalnum()
	False
	>>> ''.isalnum()
	False

string.isalpha() определяет, состоит ли строка только из букв, возвращает True, если строка s не пустая, а все ее символы буквенные. В другом случае False	>>> 'ABCabc'.isalpha()
	True
	>>> 'abc123'.isalpha()
	False

string.isdigit() определяет, состоит ли строка из цифр (проверка на число), возвращает True когда строка s не пустая и все ее символы являются цифрами, а в False если нет	>>> '123'.isdigit()
	True
	>>> '123abc'.isdigit()
	False

string.isidentifier() определяет, является ли строка допустимым идентификатором Python, возвращает True, если s валидный идентификатор (название переменной, функции, класса и т.д.) python, а в False если нет. Вернет True для строки, которая соответствует зарезервированному ключевому слову python, даже если его нельзя использовать	>>> 'foo32'.isidentifier()
	True
	>>> '32foo'.isidentifier()
	False
	>>> 'foo$32'.isidentifier()
	False

string.islower() определяет, являются ли буквенные символы строки строчными, возвращает True, если строка s не пустая, и все содержащиеся в нем буквенные символы строчные, а False если нет. Не алфавитные символы игнорируются	>>> 'abc'.islower()
	True
	>>> 'abc1$d'.islower()
	True
	>>> 'Abc1$D'.islower()
	False

string.isprintable() определяет, состоит ли строка только из печатаемых символов, возвращает, True если строка s пустая или все буквенные символы которые она содержит можно вывести на экран. Возвращает, False если s содержит хотя бы один специальный символ. Не алфавитные символы игнорируются. Это единственный метод, который возвращает True, если s пустая строка. Все остальные возвращаются False	>>> 'a\tb'.isprintable() # \t - символ табуляции
	False
	>>> 'a b'.isprintable()
	True
	>>> ''.isprintable()
	True
	>>> 'a\nb'.isprintable() # \n - символ перевода строки
	False

string.isspace() определяет, состоит ли строка только из пробельных символов, возвращает True, если s не пустая строка, и все символы являются пробельными, а False, если нет. Наиболее часто встречающиеся пробельные символы — это пробел ' ', табуляция '\t' и новая строка '\n'	>>> ' \t \n '.isspace()
	True
	>>> ' a '.isspace()
	False

string.istitle() определяет, начинаются ли слова строки с заглавной буквы, возвращает True когда s не пустая строка и первый алфавитный символ каждого слова в верхнем регистре, а все остальные буквенные символы в каждом слове строчные. Возвращает False, если нет	>>> 'This Is A Title'.istitle()
	True
	>>> 'This is a title'.istitle()
	False
	>>> 'Give Me The #$#@ Ball!'.istitle()
	True

string.isupper() определяет, являются ли буквенные символы строки заглавными, возвращает True, если строка s не пустая, и все содержащиеся в ней буквенные символы являются заглавными, и в False, если нет. Не алфавитные символы игнорируются	>>> 'ABC'.isupper()
	True
	>>> 'ABC1$D'.isupper()
	True
	>>> 'Abc1$D'.isupper()
	False

Раздаточный материал № 75

```
storm_1 = ('Lightning')
Union = (' and ')
storm_2 = ('Thunder')
print(storm_1 + Union + storm_2)
```
Результат: Lightning and Thunder

```
dog_do = ('woof!',)
print(dog_do * 3)
```
Результат: ('woof!', 'woof!', 'woof!')

Раздаточный материал № 76

```
>>> a[3]
89
>>> a[1:3]
(2.13, 'square')
```

Раздаточный материал № 77

```
>>> a = ()
>>> print(type(a))
<class 'tuple'>
```

Раздаточный материал № 78

```
>>> a = (1, 2, 3, 4, 5)
>>> print(type(a))
<class 'tuple'>
>>> print(a)
(1, 2, 3, 4, 5)
>>> print(*a)
1 2 3 4 5
>>> a = tuple('hello, world!')
>>> a
('h', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!')
```

Раздаточный материал № 79

```
>>> a = tuple((1, 2, 3, 4))
>>> print(a)
(1, 2, 3, 4)
```

Раздаточный материал № 80

```
>>> a = (1, 2, 3, 4, 5)
>>> print(a[0])
1
>>> print(a[1:3])
(2, 3)
```

Раздаточный материал № 81

```
>>> del a
>>> print(a)
Traceback (most recent call last):
File "<pyshell#28>", line 1, in <module>
print(a)
NameError: name 'a' is not defined
```

Раздаточный материал № 82

```
>>> lst = [1, 2, 3, 4, 5]
>>> print(type(lst))
<class 'list'>
>>> print(lst)
[1, 2, 3, 4, 5]
>>> tpl = tuple(lst)
>>> print(type(tpl))
<class 'tuple'>
>>> print(tpl)
(1, 2, 3, 4, 5)
```

Обратная операция также является корректной:

```
>>> tpl = (2, 4, 6, 8, 10)
>>> print(type(tpl))
<class 'tuple'>
>>> print(tpl)
(2, 4, 6, 8, 10)
>>> lst = list(tpl)
>>> print(type(lst))
<class 'list'>
>>> print(lst)
[2, 4, 6, 8, 10]
```

Раздаточный материал № 83

```
>>> nested = (1, "do", ["param", 10, 20])
```

Список внутри кортежа изменить можно

Раздаточный материал № xx

```
>>> nested[2][1] = 15
>>> nested
(1, 'do', ['param', 15, 20])
```

Выражения типа nested[2][1] используются для обращения к вложенным объектам. Первый индекс указывает на позицию вложенного объекта, второй – индекс элемента внутри вложенного объекта.

Раздаточный материал № 84

```
>>> T = (1, 2, 3, 2, 4, 2)          # Методы кортежей в Python 2.6, 3.0
                                     # и последующих версиях
>>> T.index(2)                      # Смещение первого появления элемента 2
1
>>> T.index(2, 2)                   # Смещение появления элемента 2 после смещения 2
3
>>> T.count(2)                      # Сколько всего элементов 2?
3
```

Раздаточный материал № 85

```
tuplex = (4, 6, 2, 8, 3, 1)
a, b, *c = tuplex
Результат: 4 6 [2, 8, 3, 1, 9]
```

Переменные a и b содержат целочисленные переменные, в с помещается список. Исходный кортеж tuplex остается неизменным.

Раздаточный материал № 86

```
>>> d1 = dict()
>>> print(type(d1))
<class 'dict'>

>>> d2 = {}
>>> print(type(d2))
<class 'dict'>
```

Раздаточный материал № 87

```
>>> d1 = dict(Ivan="менеджер", Mark="инженер")
>>> print(d1)
{'Mark': ‘инженер’, 'Ivan': ‘менеджер’}

>>> d2 = {"A1":"123", "A2":"456"}
>>> print(d2)
{'A2': '456', 'A1': '123'}

>>> a = {'cat': 'кошка', 'dog': 'собака', 'bird': 'птица', 'mouse': 'мышь'}
```

Создание словаря через вложенный список

```
a = [['cat','кошка'], ['dog','собака'], ['bird','птица'], ['mouse','мышь']]
s = dict(a)
print(s)
{'cat': 'кошка', 'dog': 'собака', 'bird': 'птица', 'mouse': 'мышь'}
```

Раздаточный материал № 88

```
>>> a['cat']
'кошка'

>>> a['bird']
'птица'
```

Раздаточный материал № 89

```
>>> a['elephant'] = 'бегемот'        # добавляем
>>> a['table'] = 'стол'              # добавляем
>>> a
{'dog': 'собака', 'cat': 'кошка', 'mouse': 'мышь', 'bird': 'птица', 'table': 'стол', 'elephant': 'бегемот'}

>>> a['elephant'] = 'слон'           # изменяем
>>> del a['table']                   # удаляем
>>> a
{'dog': 'собака', 'cat': 'кошка', 'mouse': 'мышь', 'bird': 'птица', 'elephant':'слон'}
```

Раздаточный материал № 90

```
d2 = {"A1":"123", "A2":"456"}
 "A1" in d2
True
"A3" in d2
False
```

Раздаточный материал № 91

```
nums = {1: 'one', 2: 'two', 3: 'three'}
person = {'name': 'Tom', 1: [30, 15, 16], 2: 2.34, ('ab', 100): 'no'}
```

Раздаточный материал № 92

```
# извлекаются ключи
for i in nums:
        print(i)
```

Результат

```
1
2
3
```

```
# извлекаются значения:
for i in nums:
        print(nums[i])
```

Результат

```
one
two
three
```

Раздаточный материал № 93 (справочно)

clear() -удаляет все элементы словаря, но не удаляет сам словарь	>>> d2 = {"A1":"123", "A2":"456"}
	>>> print(d2)
	{'A2': '456', 'A1': '123'}
	>>> d2.clear()
	>>> print(d2)
	{}

copy() - создает новую копию словаря	>>> d2 = {"A1":"123", "A2":"456"}
	>>> d3 = d2.copy()
	>>> print(d3)
	{'A1': '123', 'A2': '456'}
	>>> d3["A1"]="789"
	>>> print(d2)
	{'A2': '456', 'A1': '123'}
	>>> print(d3)
	{'A1': '789', 'A2': '456'}

fromkeys(seq[, value]) - создает новый словарь с ключами из seq и значениями из value. По умолчанию value присваивается значение None.	`t = dict.fromkeys(['a','b','c'],15)`
	`print(t)`
	{'a': 15, 'b': 15, 'c': 15}

get(key) - возвращает значение из словаря по ключу key	>>> d = {"A1":"123", "A2":"456"}
	>>> d.get("A1")
	'123'

items() - возвращает (в виде кортежа) элементы словаря (ключ, значение) в отформатированном виде	>>> d = {"A1":"123", "A2":"456"}
	>>> d.items()
	dict_items([('A2', '456'), ('A1', '123')])

keys() - возвращает ключи словаря	>>> d = {"A1":"123", "A2":"456"}
	>>> d.keys()
	dict_keys(['A2', 'A1'])

pop(key[, default]) - если ключ key есть в словаре, то данный элемент удаляется из словаря и возвращается значение по этому ключу, иначе будет возвращено значение default. Если default не указан и запрашиваемый ключ отсутствует в словаре, то будет вызвано исключение KeyError	>>> d = {"A1":"123", "A2":"456"}
	>>> d.pop("A1")
	'123'
	>>> print(d)
	{'A2': '456'}

popitem() - удаляет и возвращает последнюю пару (ключ, значение) из словаря. Если словарь пуст, то будет вызвано исключение KeyError	>>> d = {"A1":"123", "A2":"456"}
	>>> d.popitem()
	('A2', '456')
	>>> print(d)
	{'A1': '123'}

setdefault(key[, default]) - если ключ key есть в словаре, то возвращается значение по ключу. Если такого ключа нет, то в словарь вставляется элемент с ключом key и значением default, если default не определен, то по умолчанию присваивается None	>>> d = {"A1":"123", "A2":"456"}
	>>> d.setdefault("A3", "777")
	'777'
	>>> print(d)
	{'A2': '456', 'A3': '777', 'A1': '123'}
	>>> d.setdefault("A1")
	'123'
	>>> print(d)
	{'A2': '456', 'A3': '777', 'A1': '123'}

update([other]) - обновляет словарь парами (key/value) из other, если ключи уже существуют, то обновляет их значения	>>> d = {"A1":"123", "A2":"456"}
	>>> d.update({"A1":"333", "A3":"789"})
	>>> print(d)
	{'A2': '456', 'A3': '789', 'A1': '333'}

values() - возвращает значения элементов словаря	>>> d = {"A1":"123", "A2":"456"}

	>>> d.values() dict_values(['456', '123'])

Раздаточный материал № 94

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

Результат

1 is one
2 is two
3 is three

Раздаточный материал № 95

```
 # целые числа
b = {1, 3, 5, 7, 1, 3, 5, 7}
print(b)
```

{1, 3, 5, 7}

```
# строки
c = {'ok', 'no', 'yes', 'ok', 'no', 'yes'}
print(c)
```

{'ok', 'no', 'yes'}

Раздаточный материал № 96

```
 d = set('множество')
print(d)
```

{'о', 'ж', 'т', 'н', 'с', 'м', 'е', 'в'}

Раздаточный материал № 97

```
 f = set([11, 12, 13, 14, 12, 13])
print(f)
```

{11, 12, 13, 14}

```
e = set(['list', 'set', 'and', 'set'])
print(e)
```

{'and', 'list', 'set'}

Раздаточный материал № 98

```
q = set()
```

Раздаточный материал № 99

```
 w = {3, 4, 9}
w.add(11)
print(w)
```

{11, 9, 3, 4}

Раздаточный материал № 100

```
 t = {5, 6, 10}
t.update([12, 15, 17])
print(t)
```

{5, 6, 10, 12, 15, 17}

Раздаточный материал № 101

```
y = {20, 21, 22}
y.discard(21)
print(y)
{20, 22}
```

Метод remove делает тоже самое.

```
p = {27, 28, 29}
p.remove(27)
print(p)
{28, 29}
```

При попытке удаления несуществующего элемента методом discard никакой ошибки не будет. А при удалении с помощью метода remove, возникнет ошибка.

Раздаточный материал № 102

```
g = {27, 28, 29}
g.pop()
print(g)
{28, 29}
```

Раздаточный материал № 103

```
h = {31, 32, 33}
h.clear()
print(h)
set()
```

Раздаточный материал № 104

```
k = {34, 35, 36, 37}
print(len(k))
4
```

Раздаточный материал № 105

```
l = {38, 39, 40, 41}
z = {42, 39, 40, 43}
print(l & z)
{40, 39}

x = {44, 45, 46, 47}
c = {48, 49, 50, 51}
print(x & c)
set()
```

Раздаточный материал № 106

```
v = {52, 53, 54, 55}
b = {55, 56, 57, 58}
print(v | b)
{52, 53, 54, 55, 56, 57, 58}
```

Метод union является аналогичным способом объединения множеств.

```
 n = {59, 60, 61}
m = {62, 63, 64}
print (n.union(m))
{64, 59, 60, 61, 62, 63}
```

Раздаточный материал № 107

```
one = {71, 72, 73}
two = {71, 72, 75}
print(one - two)
{73}
```

Раздаточный материал № 108

```
one = {71, 72, 73}
two = {71, 72, 73}
print(one == two)
True

q = {65, 66, 67}
z = {68, 69, 70}
print(q == z)
False
```

Раздаточный материал № 109

```
colors = {"red", "green", "blue"}
for color in colors:
    print(color)

Результат (порядок может быть другим):
red
green
blue
```

Раздаточный материал № 110

```
f1 = open('new_file_1.txt')
print('Читаем и выводим на экран первые 10 байт или символов')
print(f1.read(10))
f1.close()

f1 = open('new_file_1.txt')
print('Читаем и выводим на экран весь файл')
print(f1.read())
f1.close()

f1 = open('new_file_1.txt')
print('Читаем и выводим на экран весь файл с помощью for')
for line in f1:
    print(line, end='')
print(type(f1.read()))
f1.close()
```

Раздаточный материал № 111

Метод	Описание
readline()	Чтение файла построчно
readlines()	Считывает сразу все строки и создает список

Раздаточный материал № 112

```
print('Запишем в файл структуру данных - список')
l = [' tree', ' four']
f2 = open('data.txt', 'w')
f2.write('one')
f2.write(' two')
f2.writelines(l)
f2.close()
f2 = open('data.txt')
print(f2.read())
print(type(f2.read())) # получаем тип - строка
f2.close()
```

Раздаточный материал № 113

```
# содержимое файла data_2.txt:
# зима
# весна
# лето
# осень

nums = []
for i in open('data_2.txt', encoding='UTF-8'):
    nums.append(i[:-1])
print(nums)
print('получаем тип', type(nums))
```

Результат
['зима', 'весна', 'лето', 'осень']

Раздаточный материал № 114

lambda <аргумент(ы)>: <выражение>

Лямбда функции могут иметь сколько угодно аргументов или не иметь их вовсе, но обязательно должны содержать лишь одно выражение.

Лямбда функции лучше использовать в связке с обычными функциями, например, для работы с итерируемыми объектами (map(), reduce(), zip(), filter()).

map() — это встроенная функция Python, принимающая в качестве аргумента функцию и последовательность. Она работает так, что применяет переданную функцию к каждому элементу.

Предположим, есть список целых чисел, которые нужно возвести в квадрат с помощью map.

Раздаточный материал № 115

```
# список целых чисел, которые нужно возвести в квадрат
L = [1, 2, 3, 4]
print(list(map(lambda x: x**2, L)))
```

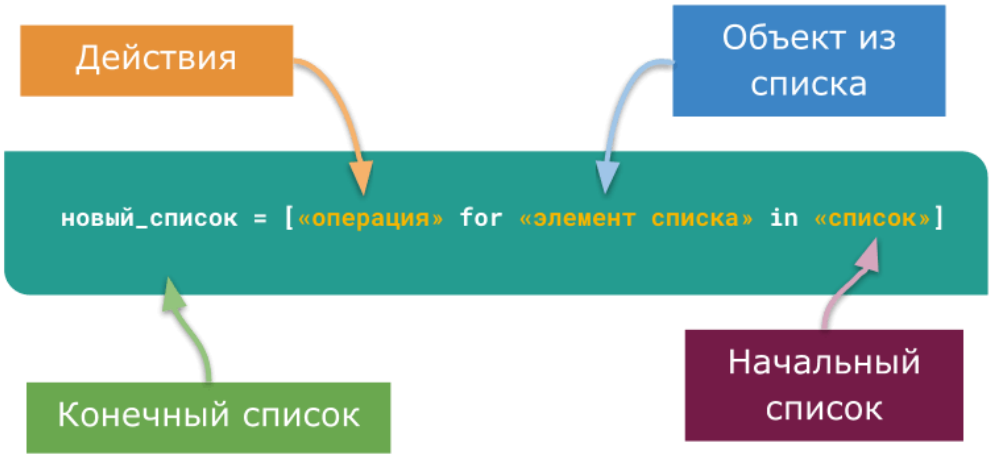
Раздаточный материал № 116

print(list(filter(lambda x: x % 2 == 0, [1, 3, 2, 5, 20, 21])))

Раздаточный материал № 117

```
fromfunctools import reduce
print(reduce(lambda x,y: y-x, L)) # работа reduce
                                  # 3 - 1 = 2
                                  # 2 - 2 = 0
                                  # 5 - 0 = 5
                                  # 20 - 5 = 15
                                  # 21 - 15 = 6
```

Раздаточный материал № 118



Раздаточный материал № 119
#1

```
# в интернет-магазине сегодня 10% скидка на ряд товаров

price = [500, 1200, 800, 600, 150]
price_new = [n * (1 - 0.1) forninprice]

print('Старыйпрайс',price)
print('Новыйпрайс', price_new)
```

#2
>>>nums = [n for n in range(1,6)]
>>> print(nums)
[1, 2, 3, 4, 5]

Раздаточный материал № 120

новый_список = [«операция» for «элемент списка» in «список» if «условие»]
Важно: здесь невозможно использовать elif, else или другие if

```
price_new1 = [n * (1 - 0.1) for n in price if n <1000]
print('Новыйпрайссостоимостьюменее 1 тыс. руб.', price_new1)
```

Раздаточный материал № 121

новый_список = [«операция» if «условие» for «элемент списка» in «список»]
Важно: условие может дополняться вариантом else (но elif невозможен).

Раздаточный материал № 122

```
# Дана строка, в которой могут присутствовать буквы любых алфавитов.
# Составить новый список, где напротив каждой буквы будет отмечено,
# является ли она английской или нет

from string import ascii_letters #string импортирован объект ascii_letters,
                                 # в котором содержатся только буквы английского
алфавита

letters = 'ыtфтгцзqп'  # набор букв из разных алфавитов

# Разграничиваем буквы на английские и не английские
is_eng = [f'{letter}-ДА' if letter in ascii_letters else f'{letter}-НЕТ' for letter in
letters]
print(is_eng)
```

Раздаточный материал № 123

```
# Генерация таблицы умножения от 3 до 7
table = [
    [x * y for x in range(3, 8)]
for y in range(3, 8)]
print(table)
```

Раздаточный материал № 124

```
# выбрать все гласные буквы из исходной фразы
fraza = "я изучаю язык Питон"
new_fraza = {i for i in fraza if i in 'аеёиоуэюя'}
print(new_fraza)
```

```
#в словаре в качестве значения ключа поместить его квадрат
squares = {i: i * i for i in range(10)}
print(squares)
```

Раздаточный материал № 125

```
id = [1, 2, 3, 4]
name = ['Меркурий', 'Венера', 'Земля', 'Марс']

rec = zip(id, name) # объединение для двух списков
print(list(rec))

radius = [2439, 6051, 3678, 3376]
rec1 = zip(id, name, radius) # объединение для трех списков
print(list(rec1))

radius_1 = [2439, 6051, 3678]
rec2 = zip(id, name, radius_1) # объединение для трех списков по длине наименьшего
print(list(rec2))

name_dict_1 = {i: nd for i, nd in zip(id, name)} # создание словаря с использованием
dict comprehension
print(name_dict_1)

name_dict_2 = dict(zip(id, name)) # создание словаря с использованием dict
comprehension
print(name_dict_2)

# добавим в словарь новые значения

new_id = [5]
new_name = ['Юпитер']
name_dict_2.update(zip(new_id, new_name))
print(name_dict_2)

# zip и выполнение расчетов
diff = [a-b for a, b in zip(radius, radius[1:])]
print(diff)
```

Раздаточный материал № 126

```
>>> lst = [1, 6, 8, 10, 20, 2, 5]
>>> type(lst)
<class 'list'>
>>> lst_it = iter(lst)
>>> type(lst_it)
<class 'list_iterator'>
>>> dir(lst)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format_
>>> dir(lst_it)
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__'
>>> next(lst_it)
1
>>> next(lst_it)
6
>>> next(lst_it)
8
>>> next(lst_it)
10
>>> next(lst_it)
20
>>> next(lst_it)
2
>>> next(lst_it)
5
>>> next(lst_it)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
StopIteration
```

Раздаточный материал № 127

numbers = [6, 57, 4, 7, 68, 95]

sq = (n**2 for n in numbers)

Раздаточный материал № 128

```
# Вариант 1
defsq_all(numbers):
for n in numbers:
yield n ** 2

numbers = [6, 57, 4, 7, 68, 95]
squares = sq_all(numbers)

for i in squares:
print(i)
# Вариант 2
def sq_all(numbers):
# оператор for убирается как самостоятельная конструкция
yield from [n ** 2 for n in numbers]


numbers = [6, 57, 4, 7, 68, 95]
squares = sq_all(numbers)

for i in squares:
print(i)
```

Раздаточный материал № 129

```
# В заданной строке найти все прописные буквы, посчитать их количество.
# Использовать библиотеку string

from string import ascii_uppercase

string_new = 'In PyCharm, you can specify third-party standalone applications and run
them as External Tools'
str_1 = [i for i in string_new if i in ascii_uppercase]
print(len(str_1))
print(str_1)
```

Раздаточный материал № 130 Константы библиотеки srting (справочно)

string.ascii_letters
Объединение констант ascii_lowercase и ascii_uppercase описано ниже. Значение не зависит от языкового стандарта.

string.ascii_lowercase
Строчные буквы 'abcdefghijklmnopqrstuvwxyz'.

string.ascii_uppercase
Заглавные буквы 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'.

string.digits
Строка '0123456789'.

string.hexdigits
Строка '0123456789abcdefABCDEF'.

string.octdigits

Строка '01234567'.

string.punctuation

Строка символов ASCII, которые считаются символами пунктуации: !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~.

string.printable

Строка символов ASCII, которые считаются печатаемыми. Комбинация digits, ascii_letters, punctuation и whitespace.

string.whitespace

Строка, содержащая все символы ASCII, считающиеся пробелами. Включает в себя пространство символов, табуляцию, перевод строки, возврат, перевод страницы и вертикальную табуляцию.

Раздаточный материал № 131

. ^ (читается «карет»)$*+?{ }[ ] \ | ( )

Раздаточный материал № 132

[09] — соответствует числу 0 или 9;
[0-9] — соответствует любому числу от 0 до 9;
[абв] — соответствует буквам «а», «б» и «в»;
[а-г] — соответствует буквам «а», «б», «в» и «г»;
[а-яё] — соответствует любой букве от «а» до «я»;
[АБВ] — соответствует буквам «А», «Б» и «В»;
[А-ЯЁ] — соответствует любой букве от «А» до «Я»;
[а-яА-ЯёЁ] — соответствует любой русской букве в любом регистре;
[0-9а-яА-ЯёЁa-zA-Z] — любая цифра и любая буква независимо от регистра и языка.
Буква «ё» не входит в диапазон [а-я], а буква «Ё» — в диапазон [А-Я].

[^09] - не цифра 0 или 9;
[^0-9] - не цифра от 0 до 9;
[^а-яА-ЯёЁa-zA-Z] - не буква.

Раздаточный материал № 133

\d — соответствует любой цифре. При указании флага A (ASCII) эквивалентно [0-9];
\w — соответствует любой букве, цифре или символу подчеркивания. При указании флага A (ASCII) эквивалентно [a-zA-Z0-9_];
\s — любой пробельный символ. При указании флага A (ASCII) эквивалентно [\t\n\r\f\v];
\D — не цифра. При указании флага A (ASCII) эквивалентно [^0-9];
\W — не буква, не цифра и не символ подчеркивания. При указании флага A (ASCII) эквивалентно [^a-zA-Z0-9_];
\S — не пробельный символ. При указании флага A (ASCII) эквивалентно [^\t\n\r\f\v].

Раздаточный материал № 134

^ Начало текста
$ Конец текста
\b — привязка к началу слова (началом слова считается пробел или любой символ, не являющийся буквой, цифрой или знаком подчеркивания);
\B — привязка к позиции, не являющейся началом слова.

Раздаточный материал № 135

<шаблон> = re.compile(<регулярное выражение> [, <флаг>])

Раздаточный материал № 136

p = re.compile (r"^\w+$")
нужно было бы записать так:
p = re.compile("^\\w+$")

Раздаточный материал № 137

```
import re           # Подключаем модуль
d = "29,12.2009"    # Вместо точки указана запятая
p = re.compile(r"^[0-3][0-9].[01][0-9].[12][09][0-9][0-9]$")
# Символ "\" не указан перед точкой
if p.search(d):
print("Дата введена правильно")
else:
print("Дата введена неправильно")
#  Так как точка означает любой символ,
#  выведет: Дата введена правильно

p = re.compile(r"^[0-3][0-9]\.[01][0-9]\.[12][09][0-9][0-9]$")
#  Символ "\" указан перед точкой
if p.search(d):
print("Дата введена правильно")
else:
print("Дата введена неправильно")
#  Так как перед точкой указан символ
#  выведет: Дата введена неправильно

p = re.compile(r"^[0-3][0-9][.][01][0-9][.][12][09][0-9][0-9]$")
#  Точка внутри квадратных скобок
if p.search(d):
print("Дата введена правильно")
else:
print("Дата введена неправильно")
#  Выведет: Дата введена неправильно
```

Раздаточный материал № 138

```
Import re
p = re.compile(r"^.+$")  # Точка не соответствует \n
print(p.findall("str1\nstr2\nstr3"))      # Ничего не найдено []

p = re.compile(r"^.+$", re.S)             # Теперь точка соответствует \n
print(p.findall("str1\nstr2\nstr3"))      # Строка полностью
соответствует['str1\nstr2\nstr3']

p = re.compile(r"^.+$", re.M)             # Многострочный режим
print(p.findall("str1\nstr2\nstr3"))      # Получили каждую подстроку ['str1', 'str2',
'str3']
```

Раздаточный материал № 139

```
import re# Подключаем модуль
p = re.compile(r"^[0-9]+$", re.S)
if p.search("245"):
print("Число")                  # Выведет: Число
else:
print("Не число")

if p.search("Строка245"):
print("Число")
else:
print("Не число")               # Выведет: Не число
```

Раздаточный материал № 140

```
import re# Подключаем модуль
p = re.compile(r"[0-9]+",  re.S)
if p.search("Строка245"):
print("Число")                  # Выведет: Число
else:
print("Не число")
```

Раздаточный материал № 141

```
# Привязка к началу и концу строки

import re
p = re.compile(r"[0-9]+$", re.S)
if p.search("Строка245"):
print("Есть число в конце строки")
else:
print("Нет числа в конце строки")
#  Выведет: Есть число в конце строки

p = re.compile(r"^[0-9]+", re.S)
if p.search("Строка245"):
print("Есть число в начале строки")
else:
print("Нет числа в начале строки")
#  Выведет: Нет числа в начале строки
```

Раздаточный материал № 142

```
import re

p = re.compile(r"\bpython\b")
print ("Найдено" if p.search ("python") else "Нет")
# выдаст Найдено

print ("Найдено" if p.search("pythonware") else "Нет")
# выдаст Нет

p = re.compile(r"\Bth\B")
print ("Найдено" if p.search("python") else "Нет")
# выдаст Найдено

print ("Найдено" if p.search("this") else "Нет")
# выдаст Нет
```

Раздаточный материал № 143

```
import re

p = re.compile(r"красн((ая)|(ое))")
print("Найдено" if p.search("красная") else "Нет")
# выдаст Найдено

print("Найдено" if p.search("красное") else "Нет")
# выдаст Найдено

print("Найдено" if p.search("красный") else "Нет")
# выдаст Нет
```

Раздаточный материал № 144

{n} — n вхождений символа в строку. Например, шаблон r"^[0-9]{2}$" соответствует двум вхождениям любой цифры;

(n,} — n или более вхождений символа в строку. Например, шаблон r"^[0-9][2, }$" соответствует двум и более вхождениям любой цифры;

{n,m} — не менее n и не более m вхождений символа в строку. Числа указываются через запятую без пробела. Например, шаблон r"^[0-9]{2,4}$" соответствует от двух до четырех вхождений любой цифры;

* — ноль или большее число вхождений символа в строку. Эквивалентно комбинации {0,};

+ — одно или большее число вхождений символа в строку. Эквивалентно комбинации {1,};

? — ни одного или одно вхождение символа в строку. Эквивалентно комбинации {0,1}.

Раздаточный материал № 145

Получим содержимое всех тегов <b> вместе с тегами:

```
importre

s = "<b>Text1</b>Text2<b>Text3</b>"
p = re.compile(r"<b>.*</b>", re.S)
print(p.findall(s))

# выдаст ['<b>Text1</b>Text2<b>Text3</b>']
# ожидалось['<b>Textl</b>', '<b>Text3</b>']
```

Раздаточный материал № 146

```
p = re.compile(r"<b>.*?</b>", re.S)
print(p.findall(s))

# выдаст ['<b>Text1</b>’, ‘<b>Text3</b>']
```

Раздаточный материал № 147

```
p = re.compile(r"<b>(.*?)</b>", re.S)
print(p.findall(s))

# выдаст['Text1', 'Text3']
```

Раздаточный материал № 148

match (<Строка>[, <Начальная позиция> [, <Конечная позиция»]])

```
import re

p = re.compile(r"[0-9]+")
print("Найдено" if p.match("str123") else "Нет")
# выдаст Нет

print("Найдено" if p.match("str123", 3) else "Нет")
# выдаст Найдено

print("Найдено" if p.match("123str") else "Нет")
# выдаст Найдено
```

Раздаточный материал № 149

re.match(<Шаблон>, <Строка>[,' <Модификатор>])

```
p = r"[0-9]+"
print("Найдено" if re.match(p, "str123") else "Нет")
# выдастНет
```

Раздаточный материал № 150
search(<Строка>[, <Начальная позиция>[, <Конечная позиция»]])

```
p = re.compile(r"[0-9]+")
print ("Найдено" if p.search("str123") else "Нет")
# выдаст Найдено

print ("Найдено" if p.search("123str") else "Нет")
# выдаст Найдено

print ("Найдено" if p.search("123str", 3) else "Нет")
# выдаст Нет
```

Раздаточный материал № 151
re. search (<Шаблон>, <Строка>[, <модификатор>])

```
p = r"[0-9]+"
print("Найдено" if re.search(p, "str123") else "Нет")
# выдаст Найдено
```

Раздаточный материал № 152
fullmatch (<Строка>[, <Начальная позиция>[, <Конечная позиция>]])

```
p = re.compile("[Pp]ython")
print("Найдено" if p.fullmatch("Python") else "Нет")
# выдаст Найдено

print("Найдено" if p.fullmatch("py") else "Нет")
# выдаст Нет

print("Найдено" if p.fullmatch("PythonWare") else "Нет")
# выдаст Нет

print("Найдено" if p.fullmatch("PythonWare", 0, 6) else "Нет")
# выдаст Найдено
```

Раздаточный материал № 153
re.fullmatch(<Шаблон>, <Строка>[, <Модификатор>])

Раздаточный материал № 154

findall(<Строка>[, <Начальная позиция>[, <Конечная позиция>]])

```
import re

p = re.compile(r"[0-9]+")
print(p.findall ("2007, 2008, 2009, 2010, 2011"))
# выдаст ['2007', '2008', '2009', '2010', '2011']

p = re.compile(r"[a-z]+")
print(p.findall("2007, 2008, 2009, 2010, 2011"))
# выдаст []
```

Раздаточный материал № 155

re.findall(<Шаблон>, <Строка>[, <Модификатор>])

Раздаточный материал № 156

finditer(<Строка>[, <Начальная позиция>[, <Конечная позиция>]])

Раздаточный материал № 156 (справочно)

♦ I или IGNORECASE — поиск без учета регистра:

```
import re
p = re.compile (r"^[а-яе]+$", re.I | re.U)
print ("Найдено" ifp.search("АБВГДЕЕ") else "Нет")
    Найдено
p = re.compile (r"^[а-яе]+$", re.U)
print ("Найдено" if p.search ("АБВГДЕЕ") else "Нет”)
    Нет
```

♦ M или MULTILINE— поиск в строке, состоящей из нескольких подстрок, разделенных символом новой строки ("\n"). Символ ^ соответствует привязке к началу каждой подстроки, а символ $ — позиции перед символом перевода строки;

♦ S или DOTALL — метасимвол «точка» по умолчанию соответствует любому символу, кроме символа перевода строки (\n). Символу перевода строки метасимвол «точка» будет соответствовать в присутствии дополнительного модификатора. Символ ^ соответствует привязке к началу всей строки, а символ $ — привязке к концу всей строки:

```
p = re.compile (r"^.$")
print ("Найдено" if p.search ("\n") else "Нет")
    Нет

p = re.compile(r"^.$", re.M)
print ("Найдено" if p.search("\n") else "Нет")
Нет

p = re.compile(r"^.$", re.S)
print ("Найдено" if p.search("\n") else "Нет")
    Найдено
```

♦ X или VERBOSE — если флаг указан, то пробелы и символы перевода строки будут проигнорированы. Внутри регулярного выражения можно использовать и комментарии:

```
p = re.compile(r"""^ # Привязка к началу строки
    [0-9]+ # Строка должна содержать одну цифру (или более)
    $      # Привязка к концу строки
    """, re.X | re.S)
print("Найдено" if p.search("1234567890") else "Нет")
    Найдено

print("Найдено" if p.search("abcdl23") else "Нет")
    Нет
```

♦ A или ASCII — классы \w, \W, \b, \B, \d, \D, \s и \S будут соответствовать символам в кодировке ASCII (по умолчанию указанные классы соответствуют Unicode-символам);

Флаги и иUNICODE, включающие режим соответствия Unicode-символам классов \w, \w, \b, \в, \d, \D, \s и \s, сохранены в Python 3 лишь для совместимости с ранними версиями этого языка и никакого влияния на обработку регулярных выражений не оказывают.

♦ L или LOCALE— учитываются настройки текущей локали. Начиная с Python 3.6, могут быть использованы только в том случае, когда регулярное выражение задается в виде значения типов bytes или bytearray.

Раздаточный материал № 157

sub(<Новый фрагмент или ссылка на функцию>, <Строка для замены>
[, <Максимальное количество замен>])

```
import re

p = "Это самый сложный урок"
print(re.sub("сложный", "не сложный", p))
# выдаст Это самый не сложный урок
```

Раздаточный материал № 158
subn(<Новый фрагмент или ссылка на функцию>, <Строка для замены>
[, <Максимальное количество замен>])

```
#Заменим все числа в строке на 0:
p = re.compile(r"[0-9]+")
print(p.subn("0", "2008, 2009, 2010, 2011"))
# выдаст ('0, 0, 0, 0', 4)
```

Раздаточный материал № 159

split(<Исходная строка>[, <Лимит>])

Раздаточный материал № 160

Содержимое файла for_split.txt

```
Этот файл
создан для демонстрации;
работы функции split. В результате должен
получиться; список
```

```
Import re

p = re.compile(r'[\n;,]+')
with open('for_split.txt', 'r', encoding='utf-8') as file:
    text = file.read()
    reg_name = re.split(p, text)
print(reg_name)

# выдаст ['Этот файл', 'создан для демонстрации', 'работы функции split.В результате
должен', 'получиться', ' список']
```

Раздаточный материал № 161

1 from tkinter import * 2 from tkinter import ttk 3	В результате импортирования в пространстве имён программы (скрипта) появляются имена, встроенные в tkinter, к которым можно обращаться непосредственно. Массовое импортирование имён может привести к их конфликту. Кроме того, для интерпретатора требуется больше времени, чтобы в списке доступных имён найти нужное. Основные виджеты- **Раздаточный материал № 162** Подмодуль ttk предоставляет доступ к множеству стилизуемых виджетов tk, а так же предоставляет дополнительные виджеты. Импорт ttkдолжен следовать за импортом tk
4 root = Tk() 5 root.title("Привет мир!") 6 root.geometry('300x40') 7	команда создаёт корневое (root) окно программы команда меняет заголовок окна команда устанавливает размеры окна

8 def button_clicked(): 9 print("Hello World!") 10 11 def close(): 12 root.destroy() 13 root.quit()	определение функции-обработчика события «нажата кнопка мыши» Функция-обработчик события «закрытие главного окна». Она останавливает главный цикл приложения и разрушает главное окно. Без неё закрыть программу можно, лишь если завершить процесс интерпретатора Python. Поскольку функция использует глобальную переменную root, объявление самой функции должно следовать после объявления переменной root.
15 button = ttk.Button(root, 16 text="Press Me", 17 command=button_clicked)	Создание кнопки с текстом «Press Me» и привязка её к функции-обработчику. rootможно опускать, т.к. это значение по умолчанию.
18 button.pack(fill=BOTH)	«Упаковываем» созданную кнопку с помощью менеджера компоновки pack. fill=BOTH (также можно fill="both") указывает кнопке занимать все доступное пространство (по ширине и высоте) на родительском виджете root
19 20 root.protocol('WM_DELETE_WINDOW', close)	Привязываем событие закрытия главного окна с функцией-обработчиком close
22 root.mainloop()	Запускаем главный цикл приложения

Раздаточный материал № 162 - Основные виджеты (справочно)

Toplevel/root

Toplevel–окноверхнегоуровня.Обычно используется для создания многооконных программ, а также для диалоговых окон.

Методы виджета

title - заголовок окна

overrideredirect - указание оконному менеджеру игнорировать это окно. Аргументом является True или False. В случае, если аргумент не указан - получаем текущее значение. Если аргумент равен True, то такое окно будет показано оконным менеджером без обрамления (без заголовка и бордюра). Может быть использовано, например, для создания splashscreen при старте программы.

iconify / deiconify - свернуть / развернуть окно

withdraw - "спрятать" (сделать невидимым) окно. Для того, чтобы снова показать его, надо использовать метод deiconify.

minsize и maxsize - минимальный / максимальный размер окна. Методы принимают два аргумента - ширина и высота окна. Если аргументы не указаны - возвращают текущее значение.

state - получить текущее значение состояния окна. Может возвращать следующие значения: normal (нормальное состояние), icon (показано в виде иконки), iconic (свёрнуто), withdrawn (не показано), zoomed (развёрнуто на полный экран, только для Windows и Mac OS X)

resizable - может ли пользователь изменять размер окна. Принимает два аргумента - возможность изменения размера по горизонтали и по вертикали. Без аргументов возвращает текущее значение.

geometry - устанавливает геометрию окна в формате ширинахвысота+x+y (пример: geometry("600x400+40+80") - поместить окно в точку с координатам 40,80 и установить размер в 600x400). Размер или координаты могут быть опущены (geometry("600x400") - только изменить размер, geometry("+40+80") - только переместить окно).

transient - сделать окно зависимым от другого окна, указанного в аргументе. Будет сворачиваться вместе с указанным окном. Без аргументов возвращает текущее значение.

protocol - получает два аргумента: название события и функцию, которая будет вызываться при наступлении указанного события. События могут называться WM_TAKE_FOCUS (получение фокуса), WM_SAVE_YOURSELF (необходимо сохраниться, в настоящий момент является устаревшим), WM_DELETE_WINDOW (удаление окна).

tkraise (синоним lift) и lower - поднимает (размещает поверх всех других окон) или опускает окно. Методы могут принимать один необязательный аргумент: над/под каким окном разместить текущее.

grab_set - устанавливает фокус на окно, даже при наличии открытых других окон
grab_release - снимает монопольное владение фокусом ввода с окна
bg – цвет фона
bd – ширина рамки

Пример:

```
from Tkinter import *
def window_deleted():
print u'Окно закрыто'
   root.quit() # явное указание на выход из программы
root=Tk()
root.title(u'Пример приложения')
root.geometry('500x400+300+200') # ширина=500, высота=400, x=300, y=200
root.protocol('WM_DELETE_WINDOW', window_deleted) # обработчик закрытия окна
root.resizable(True, False) # размер окна может быть изменён только по горизонтали
root.mainloop()
```

Таким способом можно предотвратить закрытие окна (например, если закрытие окна приведёт к потере введённых пользователем данных).

Button

Виджет Button - самая обыкновенная кнопка, которая используется в тысячах программ. Примеркода:

```
from Tkinter import *
root=Tk()
button1=Button(root,text='ok',width=25,height=5,bg='black',fg='red',font='arial 14')
button1.pack()
root.mainloop()
```

За создание, собственно, окна, отвечает класс Tk(), и первым делом нужно создать экземпляр этого класса. Этот экземпляр принято называть root, хотя вы можете назвать его как угодно. Далее создаётся кнопка, при этом мы указываем её свойства (начинать нужно с указания окна, в примере - root). Здесь перечислены некоторые из них:

text - какой текст будет отображён на кнопке (в примере - ок)
width,height - соответственно, ширина и длина кнопки.
bg - цвет кнопки (сокращенно от background, в примере цвет - чёрный)
fg - цвет текста на кнопке (сокращённо от foreground, в примере цвет - красный)
font - шрифт и его размер (в примере - arial, размер - 14)
compound – определяет ориентацию текста по отношению к изображению
image–добавляет изображение на кнопку

Label

Label - это виджет, предназначенный для отображения какой-либо надписи без возможности редактирования пользователем. Имеет те же свойства, что и перечисленные свойства кнопки.

Entry

Entry - это виджет, позволяющий пользователю ввести одну строку текста. Имеет дополнительное свойство bd (сокращённо от borderwidth), позволяющее регулировать ширину границы.

borderwidth - ширина бордюра элемента
bd - сокращение от borderwidth
width - задаёт длину элемента в знакоместах.
show - задает отображаемый символ.

Text

Text - это виджет, который позволяет пользователю ввести любое количество текста. Имеет дополнительное свойство wrap, отвечающее за перенос (чтобы, например, переносить по словам, нужно использовать значение WORD).Например,

```
from Tkinter import *
root=Tk()
text1=Text(root,height=7,width=7,font='Arial 14',wrap=WORD)
text1.pack()
root.mainloop()
```

Методыinsert, deleteигetдобавляют, удаляютилиизвлекаюттекст. Первыйаргумент - местовставкиввиде 'x.y', где x – этострока, а y – столбец. Например,

```
text1.insert(1.0,'Добавить Текст\n\ в начало первой строки')
text1.delete('1.0', END)   # Удалитьвсе
text1.get('1.0', END)      # Извлечьвсе
```

Listbox

Listbox - это виджет, который представляет собой список, из элементов которого пользователь может выбирать один или несколько пунктов. Имеет дополнительное свойство selectmode, которое, при значении SINGLE, позволяет пользователю выбрать только один элемент списка, а при значении EXTENDED - любое количество. Пример:

```
from Tkinter import *
root=Tk()
listbox1=Listbox(root,height=5,width=15,selectmode=EXTENDED)
listbox2=Listbox(root,height=5,width=15,selectmode=SINGLE)
list1=[u"Москва",u"Санкт-Петербург",u"Саратов",u"Омск"]
list2=[u"Канберра",u"Сидней",u"Мельбурн",u"Аделаида"]
for i in list1:
listbox1.insert(END,i)
for i in list2:
listbox2.insert(END,i)
listbox1.pack()
listbox2.pack()
root.mainloop()
```

Frame

Виджет Frame (рамка) предназначен для организации виджетов внутри окна. Рассмотримпример:

```
from tkinter import *
root=Tk()
frame1=Frame(root,bg='green',bd=5)
frame2=Frame(root,bg='red',bd=5)
button1=Button(frame1,text=u'Перваякнопка')
```

```
button2=Button(frame2,text=u'Вторая кнопка')
frame1.pack()
frame2.pack()
button1.pack()
button2.pack()
root.mainloop()
```

Свойство bd отвечает за толщину края рамки.

Checkbutton

Checkbutton - это виджет, который позволяет отметить „галочкой“ определенный пункт в окне. При использовании нескольких пунктов нужно каждому присвоить свою переменную. Разберем пример:

```
from tkinter import *
root=Tk()
var1=IntVar()
var2=IntVar()
check1=Checkbutton(root,text=u'1 пункт',variable=var1,onvalue=1,offvalue=0)
check2=Checkbutton(root,text=u'2 пункт',variable=var2,onvalue=1,offvalue=0)
check1.pack()
check2.pack()
root.mainloop()
```

IntVar() - специальный класс библиотеки для работы с целыми числами. variable - свойство, отвечающее за прикрепление к виджету переменной. onvalue, offvalue - свойства, которые присваивают прикреплённой к виджету переменной значение, которое зависит от состояния(onvalue - при выбранном пункте, offvalue - при невыбранном пункте).

Radiobutton

Виджет Radiobutton выполняет функцию, схожую с функцией виджета Checkbutton. Разница в том, что в виджете Radiobutton пользователь может выбрать лишь один из пунктов. Реализация этого виджета несколько иная, чем виджета Checkbutton:

```
from tkinter import *
root=Tk()
var=IntVar()
rbutton1=Radiobutton(root,text='1',variable=var,value=1)
rbutton2=Radiobutton(root,text='2',variable=var,value=2)
rbutton3=Radiobutton(root,text='3',variable=var,value=3)
rbutton1.pack()
rbutton2.pack()
rbutton3.pack()
root.mainloop()
```

В этом виджете используется уже одна переменная. В зависимости от того, какой пункт выбран, она меняет своё значение. Если присвоить этой переменной какое-либо значение, поменяется и выбранный виджет.

Scale

Scale (шкала) - это виджет, позволяющий выбрать какое-либо значение из заданного диапазона. Свойства:

orient - как расположена шкала на окне. Возможные значения: HORIZONTAL, VERTICAL (горизонтально, вертикально).

length - длина шкалы.

from_ - с какого значения начинается шкала.

to - каким значением заканчивается шкала.
tickinterval - интервал, через который отображаются метки шкалы.
resolution - шаг передвижения (минимальная длина, на которую можно передвинуть движок)
Примеркода:

```
from tkinter import *
root = Tk()
def getV(root):
   a = scale1.get()
print "Значение", a
scale1 = Scale(root,orient=HORIZONTAL,length=300,from_=50,to=80,tickinterval=5,
resolution=5)
button1 = Button(root,text=u"Получитьзначение")
scale1.pack()
button1.pack()
button1.bind("<Button-1>",getV)
root.mainloop()
```

Здесь используется специальный метод get(), который позволяет снять с виджета определенное значение, и используется не только в Scale.

Scrollbar

Этот виджет даёт возможность пользователю "прокрутить" другой виджет (например, текстовое поле). Необходимо сделать две привязки: command полосы прокрутки привязываем к методу xview/yview виджета, а xscrollcommand/yscrollcommand виджета привязываем к методу set полосы прокрутки.

Пример:

```
from tkinter import *
root = Tk()
text = Text(root, height=3, width=60)
text.pack(side='left')
scrollbar = Scrollbar(root)
scrollbar.pack(side='left')
# перваяпривязка
scrollbar['command'] = text.yview
# вторыяпривязка
text['yscrollcommand'] = scrollbar.set
root.mainloop()
```

Раздаточный материал № 163 - Упаковщики (справочно)

pack()

Упаковщик pack() является самым интеллектуальным (и самым непредсказуемым). При использовании этого упаковщика с помощью свойства side нужно указать к какой стороне родительского виджета он должен примыкать. Как правило этот упаковщик используют для размещения виджетов друг за другом (слева направо или сверху вниз). Пример:

```
from tkinter import *
root=Tk()
button1 = Button(text="1")
button2 = Button(text="2")
button3 = Button(text="3")
button4 = Button(text="4")
button5 = Button(text="5")
button1.pack(side='left')
```

```
button2.pack(side='top')
button3.pack(side='left')
button4.pack(side='bottom')
button5.pack(side='right')
root.mainloop()
```

Для создания сложной структуры с использованием этого упаковщика обычно используют Frame, вложенные друг в друга.

При применении этого упаковщика можно указать следующие аргументы:

side ("left"/"right"/"top"/"bottom") - к какой стороне должен примыкать размещаемый виджет.
fill (None/"x"/"y"/"both") - необходимо ли расширять пространство предоставляемое виджету.
expand (True/False) - необходимо ли расширять сам виджет, чтобы он занял всё предоставляемое ему пространство.
in_ - явное указание в какой родительский виджет должен быть помещён.

Дополнительные функции
pack_configure - синонимдлярack.
pack_slaves (синоним slaves) - возвращает список всех дочерних упакованных виджетов.
pack_info - возвращает информацию о конфигурации упаковки.
pack_propagate (синоним propagate) (True/False) - включает/отключает распространении информации о геометрии дочерних виджетов. По умолчанию виджет изменяет свой размер в соответствии с размером своих потомков. Этот метод может отключить такое поведение (pack_propagate(False)). Это может быть полезно, если необходимо, чтобы виджет имел фиксированный размер и не изменял его по прихоти потомков.[7]
pack_forget (синоним forget) - удаляет виджет и всю информацию о его расположении из упаковщика. Позднее этот виджет может быть снова размещён.

grid()

Этот упаковщик представляет собой таблицу с ячейками, в которые помещаются виджеты.

Аргументы
row - номер строки, в который помещаем виджет.
rowspan - сколько строк занимает виджет
column - номер столбца, в который помещаем виджет.
columnspan - сколько столбцов занимает виджет.
padx / pady - размер внешней границы (бордюра) по горизонтали и вертикали.
ipadx / ipady - размер внутренней границы (бордюра) по горизонтали и вертикали. Разница между pad и ipad в том, что при указании pad расширяется свободное пространство, а при ipad расширяется помещаемый виджет.
sticky ("n", "s", "e", "w" или их комбинация) - указывает к какой границе "приклеивать" виджет. Позволяет расширять виджет в указанном направлении. Границы названы в соответствии со сторонами света. "n" (север) - верхняя граница, "s" (юг) - нижняя, "w" (запад) - левая, "e" (восток) - правая.
in_ - явное указание в какой родительский виджет должен быть помещён.
Для каждого виджета указываем, в какой он находится строке, и в каком столбце. Если нужно, указываем, сколько ячеек он занимает (если, например, нам нужно разместить три виджета под одним, необходимо "растянуть" верхний на три ячейки). Пример:

```
entry1.grid(row=0,column=0,columnspan=3)
button1.grid(row=1,column=0)
button2.grid(row=1,column=1)
button3.grid(row=1,column=2)
```

Дополнительные функции
grid_configure - синонимдлягrid.

grid_slaves (синоним slaves) - см. pack_slaves.

grid_info - см.pack_info.

grid_propagate (синоним propagate) - см. pack_propagate.

grid_forget (синоним forget) - см. pack_forget.

grid_remove - удаляет виджет из-под управления упаковщиком, но сохраняет информацию об упаковке. Этот метод удобно использовать для временного удаления виджета.

grid_bbox (синоним bbox) - возвращает координаты (в пикселях) указанных столбцов и строк.

grid_location (синоним location) - принимает два аргумента: x и y (в пикселях). Возвращает номер строки и столбца в которые попадают указанные координаты, либо -1 если координаты попали вне виджета.

grid_size (синоним size) - возвращает размер таблицы в строках и столбцах.

grid_columnconfigure (синоним columnconfigure) и grid_rowconfigure (синоним rowconfigure) - важные функции для конфигурирования упаковщика. Методы принимают номер строки/столбца и аргументы конфигурации. Список возможных аргументов:

minsize - минимальная ширина/высота строки/столбца.

weight - "вес" строки/столбца при увеличении размера виджета. 0 означает, что строка/столбец не будет расширяться. Строка/столбец с "весом" равным 2 будет расширяться вдвое быстрее, чем с весом 1.

uniform - объединение строк/столбцов в группы. Строки/столбцы имеющие одинаковый параметр uniform будут расширяться строго в соответствии со своим весом.

pad - размер бордюра. Указывает, сколько пространства будет добавлено к самому большому виджету в строке/столбце.

Пример, текстовый виджет с двумя полосами прокрутки:

```
from tkinter import *
root=Tk()
text = Text(wrap=NONE)
vscrollbar = Scrollbar(orient='vert', command=text.yview)
text['yscrollcommand'] = vscrollbar.set
hscrollbar = Scrollbar(orient='hor', command=text.xview)
text['xscrollcommand'] = hscrollbar.set
# размещаемвиджеты
text.grid(row=0, column=0, sticky='nsew')
vscrollbar.grid(row=0, column=1, sticky='ns')
hscrollbar.grid(row=1, column=0, sticky='ew')
# конфигурируем упаковщик, чтобы текстовый виджет расширялся
root.rowconfigure(0, weight=1)
root.columnconfigure(0, weight=1)
root.mainloop()
```

place()

place представляет собой простой упаковщик, позволяющий размещать виджет в фиксированном месте с фиксированным размером. Также он позволяет указывать координаты размещения в относительных единицах для реализации "резинового" размещения. При использовании этого упаковщика, нам необходимо указывать координаты каждого виджета. Например:

```
button1.place(x=0,y=0)
```

Этот упаковщик, хоть и кажется неудобным, предоставляет полную свободу в размещении виджетов на окне.

Аргументы

anchor ("n", "s", "e", "w", "ne", "nw", "se", "sw" или "center") - какой угол или сторона размещаемого виджета будет указана в аргументах x/y/relx/rely. По умолчанию "nw" - левый верхний

bordermode ("inside", "outside", "ignore") - определяет в какой степени будут учитываться границы при размещении виджета.

in_ - явное указание в какой родительский виджет должен быть помещён.

x и y - абсолютные координаты (в пикселях) размещения виджета.
width и height - абсолютные ширина и высота виджета.
relx и rely - относительные координаты (от 0.0 до 1.0) размещения виджета.
relwidth и relheight - относительные ширина и высота виджета.

Относительные и абсолютные координаты (а также ширину и высоту) можно комбинировать. Так например, relx=0.5, x=-2 означает размещение виджета в двух пикселях слева от центра родительского виджета, relheight=1.0, height=-2 - высота виджета на два пикселя меньше высоты родительского виджета.

Дополнительные функции
place_slaves, place_forget, place_info - см. описание аналогичных методов упаковщика pack.

Раздаточный материал № 164

<Button-1> – клик левой кнопкой мыши
<Button-2> – клик средней кнопкой мыши
<Button-3> – клик правой кнопкой мыши
<Double-Button-1> – двойной клик левой кнопкой мыши
<Motion> – движение мыши
и т. д.

Раздаточный материал № 165 (справочно)

Однострочные
Нет пустых строк перед или после документации.
Используйте тройные кавычки, даже если документация умещается на одной строке. Потом будет проще её дополнить.
Закрывающие кавычки на той же строке. Это смотрится лучше.
Нет пустых строк перед или после документации.
Однострочная строка документации не должна быть "подписью" параметров функции / метода (которые могут быть получены с помощью интроспекции).
Вставляйте пустую строку до и после всех строк документации (однострочных или многострочных), которые документируют класс - вообще говоря, методы класса разделены друг от друга одной пустой строкой, а строка документации должна быть смещена от первого метода пустой строкой; для симметрии, поставьте пустую строку между заголовком класса и строкой документации. Строки документации функций и методов, как правило, не имеют этого требования.

Раздаточный материал № 166

```
def rectangle():
"""Вычисление площади прямоугольника"""
a = float(input("Ширина %s: " % figure)) # обращение к глобальной
b = float(input("Высота %s: " % figure)) # переменной figure
print("Площадь: %.2f" % (a*b))
```

Раздаточный материал № 167

```
def triangle():
"""Вычисление площади треугольника

    Используется общепринятая формула

    """
a = float(input("Основание %s: " % figure))
    h = float(input("Высота %s: " % figure))
print("Площадь: %.2f" % (0.5 * a * h))
```

Раздаточный материал № 168

```
"""Это описание модуля"""

def rectangle():
"""Вычисление площади прямоугольника"""
pass

def triangle():
"""Вычисление площади треугольника

    Используется общепринятая формула

    """
pass

print(rectangle.__doc__)
print(triangle.__doc__)
```

Раздаточный материал № 169

>>>import sys
>>>sys.path
['C:\\Program Files\\JetBrains\\PyCharm Community Edition 2021.1.2\\plugins\\python-ce\\helpers\\pydev', 'C:\\Program Files\\JetBrains\\PyCharm Community Edition 2021.1.2\\plugins\\python-ce\\helpers\\third_party\\thriftpy', 'C:\\Program Files\\JetBrains\\PyCharm Community Edition 2021.1.2\\plugins\\python-ce\\helpers\\pydev', 'C:\\Users\\OLGA\\AppData\\Local\\Programs\\Python\\Python38-32\\python38.zip', 'C:\\Users\\OLGA\\AppData\\Local\\Programs\\Python\\Python38-32\\DLLs', 'C:\\Users\\OLGA\\AppData\\Local\\Programs\\Python\\Python38-32\\lib', 'C:\\Users\\OLGA\\AppData\\Local\\Programs\\Python\\Python38-32', 'C:\\Users\\OLGA\\AppData\\Local\\Programs\\Python\\Python38-32\\lib\\site-packages', 'C:\\PythonProjects\\zab', 'C:/PythonProjects/zab']

Раздаточный материал № 170

>>>import main
>>>dir(main)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', …]

Раздаточный материал № 171

```
if __name__ == '__main__':
print_hi('PyCharm')
```

Раздаточный материал № 172

Существует файл mod.py:

a = [10, 20, 30]
print('a =', a)

>>>importmod
a = [100, 200, 300]
>>>importmod
>>>importmod

Оператор print() не выполняется при последующем импорте.

Раздаточный материал № 173

```
>>>import importlib
>>>importlib.reload(Doc.mod)
a =  [10, 20, 30]
<module 'Doc.mod' from 'C:\\PythonProjects\\zab\\Doc\\mod.py'>
```