Python.
Сложные типы данных

Строки

Что это?

Строка – **неизменяемая** упорядоченная последовательность символов.

Что значит неизменяемая?

Так можно?
s=‘Hello, world’
s=‘Чмоки, чмоки всем’

А так?
s=‘Hello, world’
s[1]=‘Ы’

В неизменяемых типах данных нельзя поменять элементы (в данном случае символы) из которых состоит объект, но можно перезаписать весь объект (например, строку) целиком. Это особенность связана с хранением данных в памяти ПК.

Как с ними работать?

Как и в Pascal, строки можно складывать, сравнивать и обращаться к символам.

S[i] – обращение к символу
S1+S2 – сложение строк
S*3 – дублирование строки

Стандартные функции

Функция	**Назначение**	**Пример**
len(S)	Длина строки	n=len(s)
ord(символ)	ASCII код символа	print(ord(‘a’))
chr(число)	Символ по коду	print(chr(121))

Функции и методы

И функции и методы выполняют действия. Разница между ними заключается в способе вызова.

Функция принимает данные в качестве аргументов (которые записываются в скобках). Функции не привязаны к объектам и могут быть вызваны просто по имени.

переменная=функция(параметры)
dlina=len(s)

Метод – это действие с конкретным объектом (строкой, списком, библиотекой, виджетом и т.п.). Для вызова необходимо сначала указать объект (например, строку), а потом через точку нужный метод. Методы также могут получать аргументы (в скобках).

переменная=объект.метод(параметры)
number=s.find('Ы')

Срезы

Срез – извлечение фрагмента строки

S[start:stop:step]
S – строка
start – начальный символ
stop – конечный символ (символ с номером stop в срез не входит)
step - шаг

S=“Hello, World!”
S[7:12] #World
S[:5] #Hello
S[7:] #World!
S[2:-2] #llo, Worl
S[2:11:2] #lo ol
S[::3] #Hl r!
S[::-1] - ???

Обращение к одному символу – это тоже срез

Методы строк

Метод	Назначение	Пример
S.find(str)	Поиск подстроки str в строке S. Возвращает номер первого вхождения или -1	s='Hello' n=s.find('l') #n=2

Списки

Что это?

Список – **изменяемая** упорядоченная последовательность данных разного типа.

```
L=[] #пустой список
L=list() #преобразовать в список (или создать пустой)
L=[1,5,”Hello, world”,42] #список со значениями
```

Также, как и для строк, используются: сложение, дублирование, функция len() и срезы.

Генератор списка

```
L = [значение for параметр in диапазон]
L= [0 for i in range(5)] #список заполнят 5 элементов со значением 0
```

В чём подвох?

Задача:

Задать список из 10 случайных чисел в диапазоне от 10 до 100.

Как это делалось на Pascal?

```
from random import randint
L=list()
for i in range(10):
    L[i]=randint(10,100)
```

В пустом списке нет элементов. При попытке обратиться к элементу L[i], которого не существует будет выдаваться ошибка. Поэтому добавление элементов происходит следующим образом:

```
L=list()
for i in range(10):
    L.append(randint(10,100))
```

Или можно использовать генератор:

```
L=[randint(10,100) for i in range(10)]
```

Методы списков

Метод	**Назначение**	**Пример**
L.append(x)	Добавляет элемент в конец списка	x=int(input()) L.append(x)

Функции списков

Функция	Назначение	Пример
min(L)	Возвращает минимальное значение из L	L=[1,2,1,2,3] m=min(L) #m=1
max(L)	Возвращает максимальное значение из L	L=[1,2,1,2,3] m=max(L) #m=3
sum(L)	Возвращает сумму элементов L. Работает только со списками, значения которых числа.	L=[1,2,1,2,3] s=sum(L) #s=9

Также наличие элемента в списке можно проверить с помощью оператора in.

```
L=[1,1,2,4,1]
X=1
while X in L:
    L.remove(X)
```

Множества

Что это?

Множество – неупорядоченная последовательность уникальных элементов. Может содержать только данные неизменяемых типов.

A=set() #пустое множество или преобразовать в множество

A= set(“hello”) #будет множество {‘h’,’o’,’l’,’e’}

A={44, 22,55} #множество со значениями

A={} – а вот так нельзя, получится dict

Можно использовать функции len(), max() и min().

… И генератор множества (как в списках)

```
a = {i ** 2 for i in range(10)}
# {0, 1, 4, 81, 64, 9, 16, 49, 25, 36}
```

Если использовать случайную генерацию, то значение записывается в множество только один раз и при совпадении не дублируется.

```
a = {random.randint(10) for i in range(5)}
# может быть {2, 5, 9} или {1, 2, 7, 5}
```

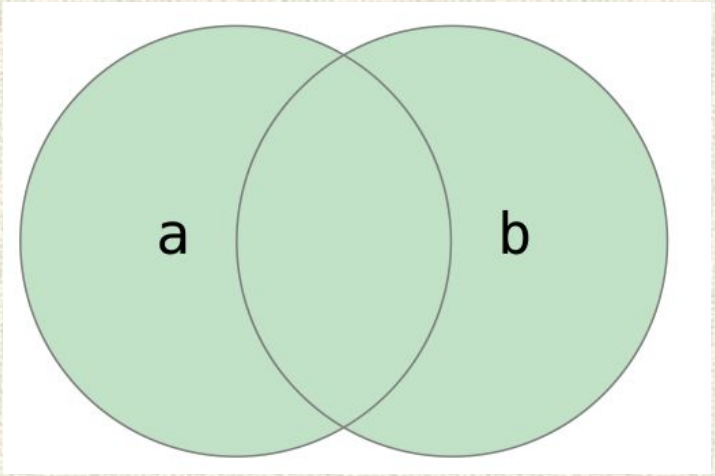
Операции с множествами

1) Объединение

Возвращает множество, содержащее неповторяющиеся элементы первого и второго множеств.

c=a | b a|=b

c=a.union(b) a.update(b)



2) Пересечение

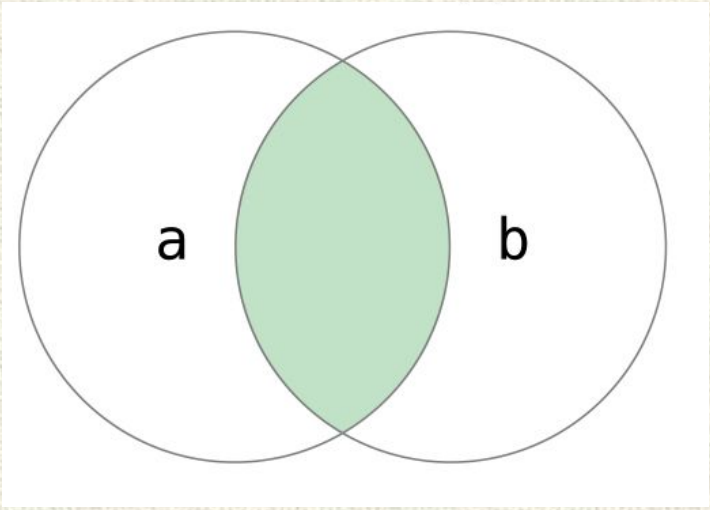
Возвращает множество, содержащее элементы, которые есть в первом и втором множествах.

c=a & b

c=a.intersection(b)

a&=b

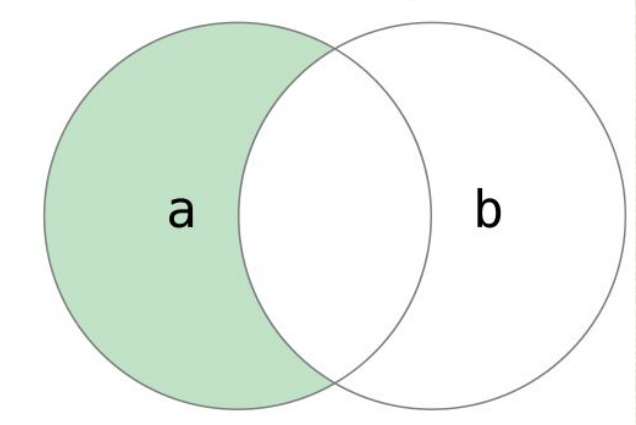
a.intersection_update(b)



Операции с множествами

3) Разность

Возвращает множество, содержащее элементы первого множества, которых нет во втором.



c=a - b

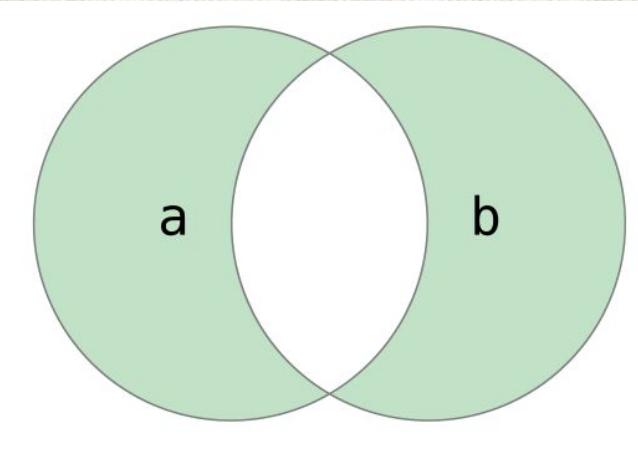
a-=b

c=a.difference(b)

a.difference_update(b)

4) Симметрическая разность

Возвращает множество, содержащее элементы, входящие в первое или второе множества, но не в оба одновременно.



c=a ^ b

c=a.symmetric_difference(b)

a^=b

a.symmetric_difference_update(b)

Методы множеств

Метод	Назначение	Пример
A.add(elem)	Добавление элемента	A.add(5)

Оператор **in** - возвращает значение True если элемент находится в множестве.

print(3 in {3,5,7}) #true
print (‘a’ in {‘s’, ‘qwer’, ‘q’}) #false

Также существует обратный оператор **not in**.

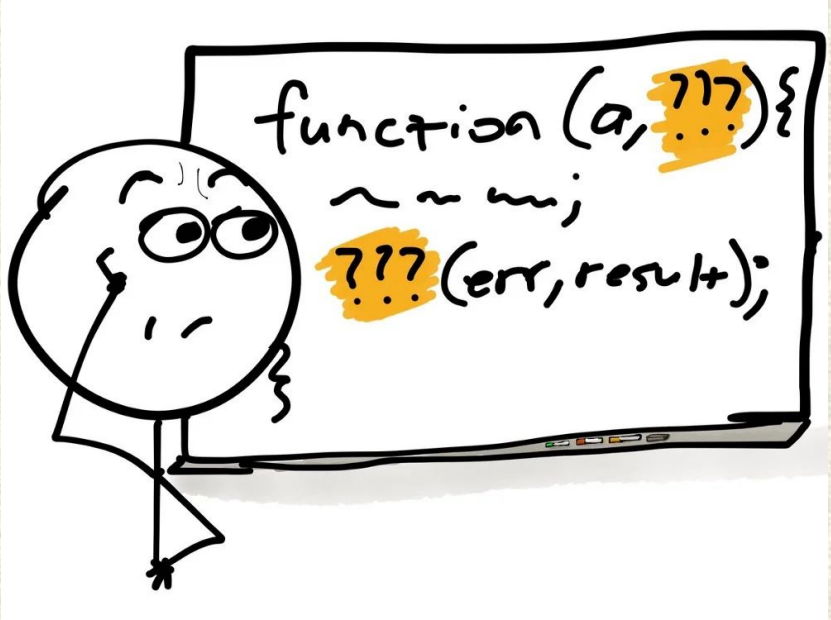
Функции

Что это?

Функция – группа команд, объединенных одним именем, возвращающая в точку вызова какое-либо значение.

Предназначены для:

1) Структурирования (разбиения основного алгоритма на части) программы.
2) Оптимизации однотипных нестандартных действий.



Как работает?

Параметры – величины, от которых зависит выполнение процедуры или функции.

Формальные – используются при описании работы функции, конкретных значений не имеют.

Фактические – передаются в функцию в точке вызова, имеют конкретные значения.

Количество и тип формальных и фактических параметров должны совпадать.

Функции

```
def Имя_Функции(параметры, через, запятую):
    тело функции
```

Переменные, объявленные внутри функции всегда (при отсутствии оператора global) являются локальными. Поэтому данные, используемые для работы функции лучше передавать через параметры, а возвращаемое значение передавать через оператор **return**.

```
def Имя_Функции(параметры, через, запятую):
    тело функции
    return значение
```

Если необходимо вернуть серию данных – используйте сложные типы данных или реализуйте алгоритм несколькими функциями.

Параметры

```
def summa():
    a=int(input())
    b=int(input())
    return  a + b

def proizv():
    a=int(input())
    b=int(input())
    return  a * b

print(summa())
print(proizv())
#не работает!
```

```
def summa(a,b):
    return  a + b

def proizv(c,d):
    return  c*d

a=int(input()) #глобальные переменные
b=int(input())

print(summa(a,b))
print(proizv(a,b)) #а так можно????
```

Если при создании функции вы указали количество передаваемых ей параметров и их порядок, то и при вызове должно быть тоже количество параметров, заданных в нужном порядке.

Если вы передали недостаточно, или слишком много параметров для функции, вы получите ошибку TypeError.

Например: **print(summa(2,3,4))**

Параметры-ключи

```
def person(name, age):
    print (name, "is", age, "years old“)

person(age=23, name="John")
```

Параметры в виде ключевых слов (Keyword argument) позволяют вводить данные в функцию в произвольном порядке (т.е. не так, как описано в функции).

```
def summa(a,b):
    return  a + b

print(summa(a=5,b=4)) #9
print(summa(b=5,a=4)) # и снова 9
print(summa(c=5,d=4)) # не работает, т.к. таких ключей нет
```

Значения по умолчанию

Также параметры могут принимать значения по умолчанию.

def Имя_Функции(параметр=значение):

```
def summa(a=5,b=3):
    return  a + b

print(summa(2,2))  # 4
print(summa(2))    # 5
print(summa())     # 8
```

При объявлении и вызове функций возможны различные сочетания параметров.

```
def summa(a,b=5):
    return  a + b

print(summa(4,4)) # 8
print(summa(5)) # 10
```

Нужно больше параметров!

В функцию можно ввести любое количество значений. Для этого используется параметр *args.

```
def fun(*args):
    s=0
    for i in args:
        s+=i
    return s
print(fun(1,3,5,7)) #16
```

```
def many(*args, **kwargs):
    print( args )
    print( kwargs )
many(1, 2, 3, name="Mike", job="proger")
# (1, 2, 3)
# {'job': 'proger', 'name': 'Mike'}
```

После введения данных *args становится кортежем (неизменяемый список) и может быть использован внутри функции.

Для хранения произвольного количества параметров-ключей используется параметр **kvargs, который становится словарём.

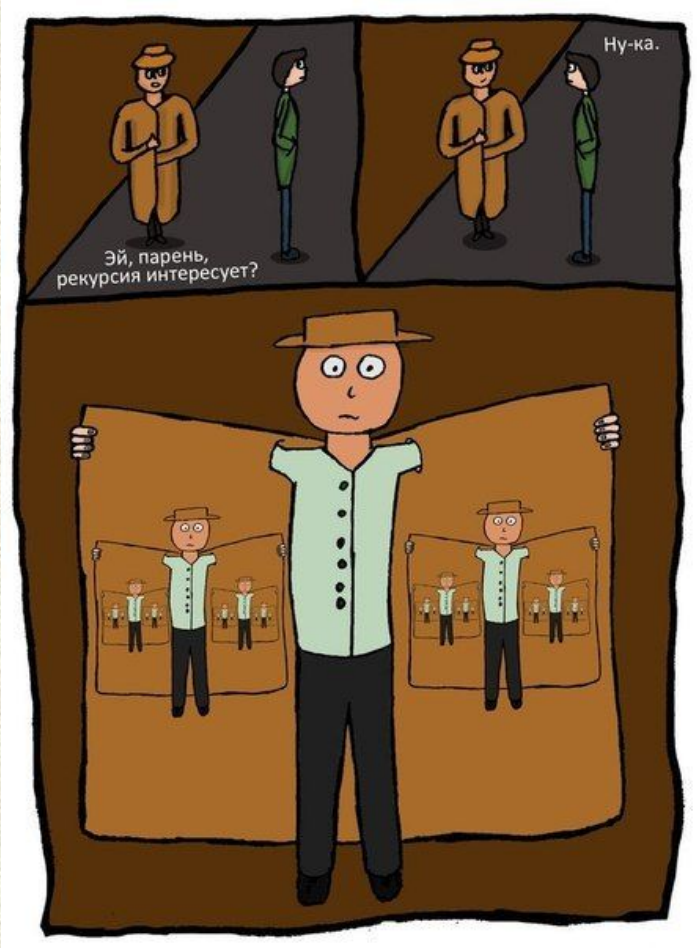
На самом деле эти параметры можно называть, как угодно (например, *lol и **omg), но принято использовать *args и **kvargs.

Функции

Рекурсия – функция, вызывающая в теле сама себя.

```
def Имя_Функции(параметры):
    тело функции
    условие выхода
    вызов себя
```

```
def factorial(n):
    if n <= 0:
        return 1
    else:
        return n * factorial(n - 1)
```



Анонимные функции

Анонимная функция (или лямбда-функция) - это короткая однострочная функция, которая содержит только одну команду. Такая функция может принимать любое количество параметров и не имеет названия.

lambda аргументы: действие

```
sum = lambda x, y: x + y
print(sum(1, 2)) #3
```

Чаще всего используются в ситуациях, когда функция является параметром другой функции или метода.

Например, в качестве ключа при сортировке:

```
a = [[1, 2], [4, 1], [9, 10], [13, -3]]
a.sort(key=lambda x: x[1])
print(a)  # [[13, -3], [4, 1], [1, 2], [9, 10]]
```

Файлы

Что это?

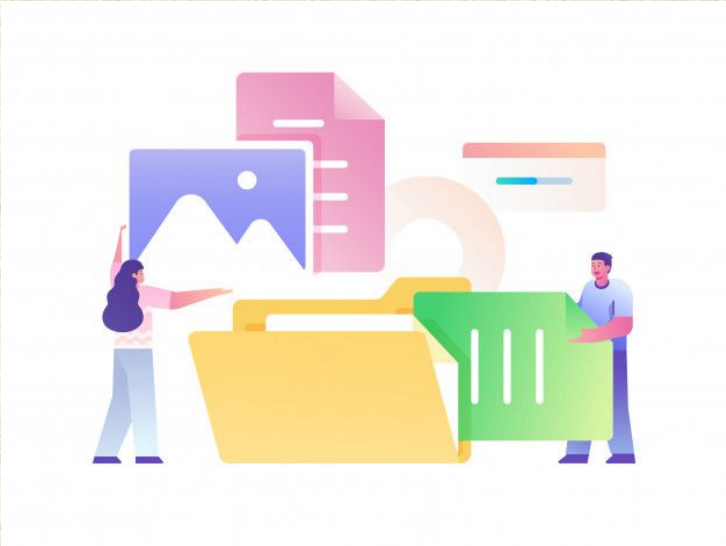
Файл – поименованная область на носителе информации, содержащая данные определенного типа. Предназначены для долговременного хранения данных.

Типы файлов в Python:

1)Текстовые – хранят данные любого вида в текстовом формате. Могут иметь расширение .txt или .rtf.
2)Бинарные – хранят последовательность битов (расширение .bin).

Алгоритм работы с файлом:

1)Связать файл и файловую переменную
2)Открыть файл
3)Чтение/запись
4)Закрыть файл



Связь и открытие

Связь с файловой переменной и открытие файла происходят при помощи функции open().

Ф.п.=open(‘имя_файла’, ‘способ_открытия’)

f=open(‘myfile.txt’, ‘r’)

Режим	Описание
'r'	Только для чтения.
‘w’	Только для записи. Создаст новый файл, если его не существует.
‘a’	Только для дозаписи. Создаст новый файл, если его не существует.
‘x’	Только для записи, если файла не существует. Иначе исключение
‘t'	В текстовом режиме.
‘b'	В бинарном режиме.
‘+'	Для чтения и записи.

Режимы можно комбинировать. Например, для записи бинарного файла используем **‘wb’**, а для одновременного чтения и записи текстового – **‘r+’**.

По умолчанию выбран текстовый файл, а режим – чтение.

Чтение

Метод	Назначение
f.readline([i])	Чтение одной строки из файла (i – номер строки)
f.readlines()	Построчное чтение файла в список (элемент – строка)
f.read([i])	Прочитает весь файл (или i символов) в строку

Файл является порядковым типом, поэтому:

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

Для чтения или записи в нужном участке файла используют указатель.

Метод	Назначение
f.tell()	Возвращает позицию указателя в файле
f.seek(x[,y])	Перемещает указатель к позиции x[,y]

Запись

Метод	Назначение
f.write(s)	Записывает строку s в файл
f.writelines(L)	Записывает порядковый тип (например, список) в одной строке файла.

Записываемые данные должны быть строкового типа. Если они не строки, то их нужно предварительно преобразовать.

При записи в файл не происходит перевода курсора и данные пишутся в одной строке (для перевода можно использовать ‘\n’).

А нельзя, как в Pascal построчно писать в файл?

Для построчной записи без предварительного преобразования можно использовать print() с аргументом file.

```
L=[1,2,3,4]
f=open(‘myfile.txt’, ‘w’)
for i in L:
    print(i, file=f)
```

Оператор with

Для закрытия файла используется метод .close().

В результате, например, чтение из файла выполняется следующим образом:.

```
f=open('myfile.txt', 'r')
for i in f:
    s=i
    print(s[1])
f.close()
```

```
with open('myfile.txt', 'r') as f:
    for i in f:
        s=i
        print(s[1])
```

Оператор with … as создает изолированный контейнер для работы с файлом. Это самый корректный способ работы с файлом.

With, например, позволяет не закрывать файл (он закроется автоматически после окончания with). Также, если имя файла указано некорректно, то команды внутри with просто не выполнятся и аварийной остановки программы не произойдет.