Встроенные типы объектов

`int float str bool list dict tuple set file`

Операции над числами

`+ - * / ** // %`

Приоритеты операций

слева на право убывает

в порядку убывания

```
()
**
* / // %
+ -
```

Встроенные операции

```
abs(value)
min(x1, x2, x3, ...)
max(x1, x2, x3, ...)
pow(x, y)
round(float, ndigits)
type(obg)
```

модуль math

ключевые слова

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

массовое присвоение

a=b=c=d=1

множественное присвоение

a, b = 2, 1

ввод данных

a=input('comment')

int(a)

int('4.5') ошибка

float('10') float('2.5')

a, b, c= map(int, input().split(" "))

print(a, sep=' - ', end='\n')

print('xxx %s gghj %s' % (a, b))

trunc отсечь floor округление вниз ceil округление вверх модуль math

операторы сравнения > < >= <= == != инверсия not конъюнкция and дизъюнкция or

приоритет not and or

строки ''' kjb ''' """bhvhjvj""" "klnkjb'bkhv'knjb" 'kn"lkm"lnbnjkb'

concatenate - сцепление + строк

'lkklk'*3

length()

'j' in 'bhvh'

'' == ''

'jbj'>'m'

ord('b') код символа

0< a < 10

a>0<b

методы строк

s.upper

s.lower

s.coutn(sub, start, end) количество вхождений

s.find(str) найти индекс подстроки или -1 не найдено

s.rfind искать с конца с права

s.index(str) искать индекс, исключение если не найдено

s.replace(old, new, count) замена в строке

s.isalpha() проверка состоит ли только из букв

s.isdigit() только из цифр

s.rjust(n, 's') расширить строку в право

s.ljust() влево

s.split() разделить строку

' '.join(l) список в строку с разделителем

s.strip() удалить справа слева все пробелы и служебные символы '\n'

s.rstrip() только справа

s.lstrip() только слева

isupper, islower, isalnum проверки

capitalize заглавный символ только первого слова

title заглавные первые символы слов

swapcase инверсия всех букв

startswith, endswith поиск в строке (соответствие на подстроку)

Служебные символы

`\newline` продолжение на новой строке

`\\` символ обратного слэша

`\'` апостроф

`\"` кавычка

`\a` звонок

`\b` забой

`\f` перевод формата

`\v` новая строка

`\r` возврат каретки

`\t` горизонтальная табуляция

`\v` вертикальная табуляция

r'string' сырая строка не воспринимает служебные символы

`r'''nnb`

`jb'''`

Форматирование строк format

```
""" kljjk {O} lkh bj {1} lnmn
hjkghh g{3} """.format(name, min, balance)
позиционное использование
```

```
""" kljjk {n} lkh bj {k} lnmn
hjkghh g{b} """.format(n=name, k=min, b=balance)
позиционное именное
```

F-строка

```
f""" hello {name} kgghf {d['ky']} """
```

Списки

bl = [] bl = list() пустой список

```
a = [True, 10, 'hello', 5.6, [4, 6, 5]] # пример списка
```

len(a) длинна списка

[12, 14] + ['1', 100] сцепление (сложение) списков

a += ['h', 5] расширение списка

['h', 'l']*5 дублирование списка

`4.5 in [True, 10.2, 5, 100, 4.5, 5.4]` проверка вхождения союзом in

max min sum для списков из чисел

sorted(list) сортировка по возрастанию

sorted(l, reverse=True) в обратном порядке

sorted(l, key=lambda x: (x[0], x[1], ...), reverse=True) сортировка по нескольким значениям сначала первое потов по второму при равных значениях первого и т.д.

[100, 3] > [34, 100, 2] сравнения списков (поэлементно)

[1, 2, 3]==[1, 2, 3]

a[0] первый элемент

a[-1] последний элемент

a[1:4] срез (последний индекс не включается)

a[2:-1] по пред последний

a[2:999]

a[2:] со второго и по конец

a[:4] с начало и до минус один

a[:] весь список

a[::2] с шагом

 a[::-1] реверс

a[2] = 10 список изменяемый объект

a[3:5] = 34, 23

a[2:5] = 23, 34 вставит значения и удалит недостающие до вставке

del a[2] удалить элемент списка

b = a[:] сохранить копию списка

 append

clear

copy

[]

count

extend

index

in

insert

pop

remove

reverse

sort reverse

listcomps, dictcomps, setcomps словарными включениями множественными включениями списковые включения

выражение-генератор

условия

if :

  print()

else:

  print()

if :

  print()

elif:

elif:

elif:

else:

  print()

циклы

while

break continue else

делитель - число на которое делиться делимое число без остатка

a%d == 0 d-делитель

НОД - наибольший общий делитель для двух чисел

алгоритм Евклида

Пока a != b

выбираем большее

уменьшаем на меньшее

вывод a или b

```
while a!=b:
    if a>b:
        a -= b
    else:
        b -= a
print(a)
```

если НОД(a, b) == 1 - взаимопростые числа

a > b > 0 a%b=s a = b b=s

```
while b>0:
    c=a%b
    a=b
    b=c
    # a, b = b, a%b
print(a)

math.gcd(30, 18)
```

НОК = наименьшее общее кратное

`a*b = НОД*НОК`

Итерируемые объекты

`range` формирует арифметическую прогрессию

`list(range(0, 5, 1))` 0-5 интервал 5 не входит в интервал

`range(10, 0, -1)`

`iterable` итерируемый объект (проход по всем элементам)

`v = iter(range(5)); next(v); v.__next__() StopIteration`

`iter(str) iter(list)`

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

`for <var, vari ...> in <iter obg>: <body>`

главная диагональ квадратной таблицы

i, j номер строки , столбца

главная диагональ i=j (из верхнего левого угла в нижний правый)

выше главной диагонали i<j

ниже главной диагонали i>j

https://www.python.org -> Documentation -> Library Reference

описание стандартных библиотек

`[ <выражение> for <val> in <iter> if <условие>]` генераторы списков

list comprehension генератор списка

`[(i, j) for i in 'abc' for j in [1, 2, 3] if True]` аналогично вложенным циклам. одна итерация первого цикла затем все итерации второго цикла ...

`[('a', 1), ('a', 2), ('a', 3), ('b', 1), ('b', 2), ('b', 3), ('c', 1), ('c', 2), ('c', 3)]`

```
from string import ascii_uppercase
```

```
__import__('string').ascii_uppercase[i]
```

```
# a <= b or a > b
[i**2 for i in range(a, b + 1)] or [i**3 for i in range(a, b - 1, -1)]  # в одном из будет пустой список а или вернет один

(2, 3)[a > b] # оброшению к кортэжу через индекс а True и False к типу инт 0 или 1
2+(a > b) -> 2 или 3
[<вырожение с n> for n in [x]]

(lambda n = 10: [i for i range(n)])() # ананимная функция с вызовом и параметром по умолчанию
или
(lambda n: print(n))(10)
```

`dictionary dict` словарь ассоциативный массив

`{key: value}  dict(key=value, ...)`

`bict([[key, val], [key, val], [key, val]])  или же tuple`

`dict.fromkeys(['a'],['b'],['c'], val)  a:val ...`

`{} dict()  пустой словаря`

ключ должен быть не изменяемый тип объекта

ключи уникальны

`d[key]` обращение к элементу словаря

`d[newkey] = val` создание элемента в словаре

`d[key] = val` словарь изменяемый объект

`del d[key]` удалить значение из словаря

`len(d)` сколько элементов в словаре

`key in d` проверка есть ли элемент с таким кличем

`for k in d: print(d[k])` перебор элементов словаря

`d.clear()` очистить словарь

`d.get(key, [def_val=None])` получить значение по ключу

`d.setdefault(key, def_val=None)` создает элемент если он не существует и возвращает значение, если элемент уже есть возвращает значение

`d.pop(key)` возвращает значение и удаляет элемент в словаре KeyError

`d.popitem()` удаляет случайный элемент и возвращает элемент (key, val)

`d.keys()` возвращает все ключи

`d.values()` все значения

`d.items()` все элементы пары ключ значение

`[print(f'{k}') for k in range(3)]`

`sorted(d, key=d.get, reverse=True)` d возвращает набор ключей ключи идут в функцию

`sorted(d)` сортировка массива по ключу возвращает список

`sorted(d.items(), key=lambda x: (-x[1], x[0]), reverse=False)` сортировка по числу в обратном порядке затем по алфавиту

можно использовать подряд две отдельно сортировки разных значений

```
Гермиона 4.0
Зина 4.0
```

```
for a in iter(input, 'конец'):  # sentinel='конец'  input -> obg.__call__()
    pass
```

Если передается аргумент `sentinel`, то ожидается, что первый аргумент `object`, поддерживает вызов `__call__()`. В этом случае, созданный итератор будет вызывать указанный объект с каждым обращением к своему `__next__()` и проверять полученное значение на равенство со значением, переданным в аргумент `sentinel`. Если полученное значение равно `sentinel`, то бросается исключение `StopIteration`, иначе возвращается полученное значение.

Если НЕТ аргумента `sentinel`, то первый аргумент `object` должен быть объектом-коллекцией, которая поддерживает протокол итераций метод `__iter__()` или он должен поддерживать протокол последовательности метод `__getitem__()` с целыми аргументами, начиная с `0`. Если он не поддерживает любой из этих протоколов, бросается исключение `TypeError`.

```
with open('mydata.txt') as fp:
    # читаем, пока не попадется пустая строка
    for line in iter(fp.readline, sentinel=''):
```

docs-python

min, max(d, key=d.get)

`d[i] = d.get(i, 0) +1` создаем или используем значения по ключу

разряженный список, не все элементы используются замена разряженных списков словарем

соответствие между объектами ключ не изменяемый объект и значение любой объект

`if k in d:` находится ли ключ в словаре

хранение данных об объекте вложенный словарь `d[k][k2]`

подсчет значений

tuple кортеж

`a = ()` `a = tuple([iter])` `a = 1, a=(1,)`

`x in a  not x in a` проверка вхождения или не вхождения элемента в кортеж

`a+b  a*n min max sum`

`a[0]`

кортеж не изменяемый объект

`index(x) count(x)`

в кортеже могут находиться изменяемые объекты

`a.__sieof__()`

кортеж в качестве ключа словаря

`list(a)` кортеж в список

`tuple(l)` список в кортеж

`for z in [ [morze[j] for j in i] for i in in_put ]` вложенные генераторы списков [[x, y, z ...], ... [a, b, ...i]]

from collections import Counter

генератор списков

key: value

`d = {i: i**2 for i in range(1, 11)}`

`{str(x): x+1 for x in [10, 22, 32] }`

`{key.title(): int(value) for key, value in data.items() }`

`d = {k: v for k, v in zip(range(len(phone_book)), phone_book)}`

множества set

неупорядоченная коллекция уникальных элементов не изменяемого типа (нет повторений)

`s = {1, '2' 32.25, (1, 'k')}`

`s=set('ghf hkjgk h')  set(iter_obj)`

`a = list(set(a))`

`a.add(x)`

`a.update(iterabl_obj)  a.update([5, 7, 6]) a.update({0, 10, 'h'}) a.update('jkbkjhf')` добавить поэлементно

`a.discard(x)` удалить элемент множества

`a.remove(x)` тоже самое но если нет то исключение генерируется

`a.pop()` удалить случайный элемент и вернуть его если пусто то исключение

`a.clear()` очистить множество

`len(a)  4 in a` длинна и вхождение ринадлежнасть

`a & b  a &= c  a.intersection(b) a.intersection_update(b)` пересечение (только элементы в обеих множествах)

`a | b  a |= b  a.union(b)` объединение элемента обоих списков

`a - b  b - a  a -= b` вычитание множеств из первого удаляются элементы второго

`a.difference_update(b) a.difference(b)` разность a-b

`a ^ b` симметрическая разность, все элементы кроме общих, исключение общих элементов

`a==b` сравнение множеств

`a<b  a<=b` сравнение множеств a подмножество b (a входит в b)

`for i in a:`

`print(len(set(input()) - set('{} ,')))`

функции

многократно используемый фрагмент кода

определение функции не может располагаться после вызова функции (определение до вызова)

функции решают проблему декомпозиции

```
varg #  global
def fname():
    varl #  local

def f():
    global varg
    varg = n  # изменяем состояние глобальной переменной иначе создается
локальная
    # глобальные переменные ведны внутри функции без global но не изменяемы
(создается лок. пер. с таким им.)
```

переопределение функций перезаписывают ранние определения

сначала поиск локальных переменных потом глобальных

функции могут быть вложенные

built in встроенное пространства имен

global глобальная область

local локальная область

```
def s():
    a = 100
    def q():
        nonlocal a
        a = 200
    print(a)
```

дефолтные значения аргументов функций должны быть не изменяемые что бы не создать замыкание

в функцию передаются ссылки

если нужно отвязать то передавать копии списка словаря объекта

f(1, 2, 3) позиционная передача параметров

f(b = 10, c = 20, a = 5) по имени

f(20, c=11, b=6) комбинированно

def f(a, b, c='nnn') аргументы по умолчанию (должны в конце быть)

произвольные значение и имена

a, b, *c = True, 7, 'hello', 9, '54', 1, 2, 3.0 -> print(a, b, c) -> True 7 ['hello', 9, '54', 1, 2, 3.0]

a, *b, c = ... *a, b, c = 'helow world'

a, b, *c = [1, 3] -> c == []

s = [1, 10]

list(range(*s)) распаковка списка

def f(a, b, c, d) -> x = ['h', True, 78, [3, 4, 5]] -> f(*x)

def f(*args) переменное количество не именованных входных параметров без значений по умолчанию

f(1, 2, 3, 4) передается как не изменяемый кортеж

f() -> args == () пустой кортеж

def f(**kwargs) именованные параметры (упаковываются в словарь) переменной длинны

f(a=0, b=2, c=3) -> kwargs['a'] kwargs['b'] kwargs['c'] представляется как словарь

d = {a: 0, b:1, c:1}

f(**d)

def f(*args, **kwargs) параметры переменной длинны комбинированные

print(*[2, 23, 5]) -> 2 23 5

рекурсивная функция стек вызовов в стек и из стека возвращается и продолжается выполнение

n! = (n-1)! * n 1! = 1 выход

```
def f(n):
    if n == 1: return 1
    return f(n-1) * n
```

```
def fibonacci(n):  # 0, 1, 1, 2, 3, 4  n > 2  n = n(i-1) + n(i-2)  n=1 -> 0 n=2
-> 1
    if n == 0: return 0 # условие выхода из рекурсии
    if n == 1: return 1
    # if n < 2: return n
    return fibonacci(n - 1) + fibonacci(n - 2)

# полиндром
def pal(s):
    if len(s) <= 1: return True
    if s[0] != s[-1]: return false
    return pal(s[1:-1])
```

rec = lambda n: rec(n-1) + rec(n-2) if n > 1 else n

per = lambda a, b, c: a+b+c ; p = per()

lambda x: 10 if x>0 else 20

def fib(n): return fib(n - 1) + fib(n - 2) if n > 2 else 1

enumerate(list, start_index)

lambda arg1, ... argn: expresion одно выражение с return

```
def lf(k, b): return lambda x: x*k+b
gr = lf(2, 5)
print(gr(4))
```

`os.listdir(path)`

`os.path.isdir(path)`

рекурсивный обход

os shutil

вложенные функции

def col():

 y = 'gr'

 x=10

 def pr_red():

  nonlocal x

  r = 'red'

  print(r)

  x = 20

 pr_red()

 print(x)

замыкания

def fun(y=None):

var = 20.5 if y is None else y

def inner_fun(x = 0):

print(var+x)

return fun

f = fun()

f2 = fun(354.54)

print(f(), f2(10))

```
def count():
    c = 0
    def inner_c():
        c += 1
        return c
    return inne_c
rn = count()
print(rn(), rn(), rn())
rn = count()
print(rn(), rn(), rn())

def aver_num():
    nums = []
    def inner(num):
        nums.apend(num); return sum(nums) / len(nums)
    return inner
af = aver_num()
af(10); af(12); af(20)

def aver_num():
    s = 0
    c = 0
    def inner(num):
        nonlocal s; nonlocal c;
        s += num; c += 1
    return s / c
af = aver_num()
af(10); af(12); af(20)

from time import perf_counter
pref_counter()
```

```
    def add(a, b): return a+b
    def mult(a, b, c): return a*b*c
    def coun(f):
        c = 0
        def ir(*args, **kwargs):
            nonlocal c; c+=1; print(f"{f.__name__} - {c}")
            return f(*args, **kwargs)
        return ir
    m = coun(add);  m(12, 10);  m(120, 100)
    m = coun(mult);  m(12, 10, 25);  m(120, 100, 3)
```

decorator

```
def dec_h(f):
    def ir():
        print('<h1>'); f*args, **kwargs print('<\h1>')
    return ir
def s():
    print(s.__name__)
ff = dec_h(s); ff(); ff = dec_h(ff); ff()

def dec_tb(f):
    def ir():
        print('<table>'); f(); print('<\table>')
    return ir
@dec_h
@dec_tb
def d():
    print('----')

d()

def d():
    print('***')

d = dec_h(dec_tb(d))
d()
```

```
def t(f):
    def fi(*a, **k):
        f(*a, **k)
    fi.__name__ = f.__name__; fi.__doc__ = f.__doc__

def s():
    pass
s = t(s)
s.__name__; s.__doc__; help(s)

from functools import wraps
def t(f):
    @wraps(f)
    def fi(*a, **k):
        f(*a, **k)
```

pprint pretty print

locals() словарь локальных переменных dir(m) пространство имен

Модули

pip freeze # список пакетов (модулей) установленных

`pip install <module name==ver>`

python3 -m pip install ...

`pip freeze > requirements.txt`

pypi.org

`pip install -r requirements.txt`

пространство имен модуля попадают все глобальные переменные

calendar.TextCalendar()

импорт модулей

`import <module_name>`

`<module_name>.var  <module_name>.fun_name()`

`import math as m` псевдоним

импорт определенных данных

`from <module_name> import (var1, var2 , f1, f2 as fff)`

по пэп8 импорт модулей вверху

если импортируется модуль в котором используется импорт то происходит расширение пространства имен

пространство имен модуля + импортированное пространство имен

при импорте 1 поиск указанного файла

sys.path - список путей где ищутся файлы

from dir.file import var, fn

sys.path.append('new path') добавить путь для поиска

при импорте модуля файл (этого модуля) запускается на выполнение при первом импорте

`import importlib`

`importlib.reload(<module_name>)` для перезагрузке модуля в коде

импорт с начало запускаться там где файл исполняемый

`__name__` имя модуля загружаемого. если модуль выполняется как основной то значение будет `__main__`

```
if __name__ == __main__:
```

```
  ....
```

пакеты - папка с модулями (файлами py) package

`import package.file`

`package.file.fn()`

`from package import file`

`from package.file import var`

`__init__.py` файл в пакете он исполняется когда импортируется

пакет с модулями. исполняется один раз даже если импортов много

относительное импортирование путь относительно чего импортируется

. текущий каталог

.. каталог выше

`__init__.py`

from . import file1, file2 ...

main.py

import package1

package1.file1

при импорте из модуля from .file1 import * в файле модуля можно указать перечень импортируемых имен `__all__ = ['var1', 'file1', ...]`

`from .. file1import var`

файлы

`file = open(r'.\test.txt', encoding='utf-8')`

`file.read([n_chars])` позиция курсора сдвигается

`file.seek(0)` установить позицию курсора для чтения

`file.readline()`

```
for row in file:
    print(row)  # вывод по строкам
    for ch in row:
        print(ch, end='')
```

`open('file.txt', 'rwa')  # r - read w - rite a - append b - binarry a+ read+write`

`f.write(s)`

`f.close()`

`whith` менеджер контекста

`JSON - JavaScript Object Notation`

`json -> obj python`

`import json   d = json.loads(str_json)`

`dict -> json  json.dumps(data, indent = 4)`

```
with open('j.json', 'w') as f:
    json.dump(dt, f, indent = 4)

with open('j.json', 'r') as f:
    dt = json.load(f)
```

Выражение генератор

i = (i**2 for i in range(6)) в круглых скобках возвращает генератор

next(i)

обойти for один раз плюсы не хранятся в памяти все вместе берутся на лету

`c = (i for i in range(1000000000)); for i in c: print(i)`

1_000_000 == 1000000

функция генератор yield

```
map(fun, *iterables) --> map object
```

```
filter([fun], iterable) --> filter object if fun = None return all items true
fun return True False
a = [10, 0, 25, 0, 1]
print(list(filter(None, a)))  #[10, 25, 1]
```

`list(zip(a, b, ...)) --> [(a[0], b[0]), (...), ... ]`

`r = zip(a, b, c)`

`c1, c2, c3  = zip(*r)`

вызываемые объекты

callable() -> True встроенные функции, встроенные методы, собственные функции, классы (при создании)

, объекты экземпляры класса с методом `__call__` , методы класса, функции генераторы yield

модуль collections

`from collections import Counter # подсчет элементов`

принимает итерируемый объект и подсчитывает одинаковые значения

результат в виде словаря + c.elements() + c.most_common([n]) возвращает список кортежей ключ значение n-количество наиболее встречаемых

c['key'] если key не существует то возвращает ноль можно также присваивать новые значения c['key']=new_val объекты можно складывать вычитать
в конструктор можно передавать ключ значение

ключи только не изменяемые объекты

`from collections import defaultdict` при обращении к несуществующему ключу создается по умолчанию значение типа класса переданное в конструктор класса d = defaultdic(int) d['s'] -> 0 int()

d.default_factory = callable можно и lambda

для подсчета и группировки элементов

`from collections import nmedtuple` именованный кортеж неизменяемый

в кортеже обращение по индексу в именованном кортеже обращение по имени

nmedtuple - фабричный класс

`point_d2 = namedtuple('Point2D', 'x y')`

`p1 = point_2d(2, 6); p1[0]; p1.x`

`p2 = point_2d(y=2, x=6); p2`

можно изменить значение через p._replace(key=vaue)

аннотация типов

`class Person(NamedTuple): nname: str; surname:str`

датаклассы