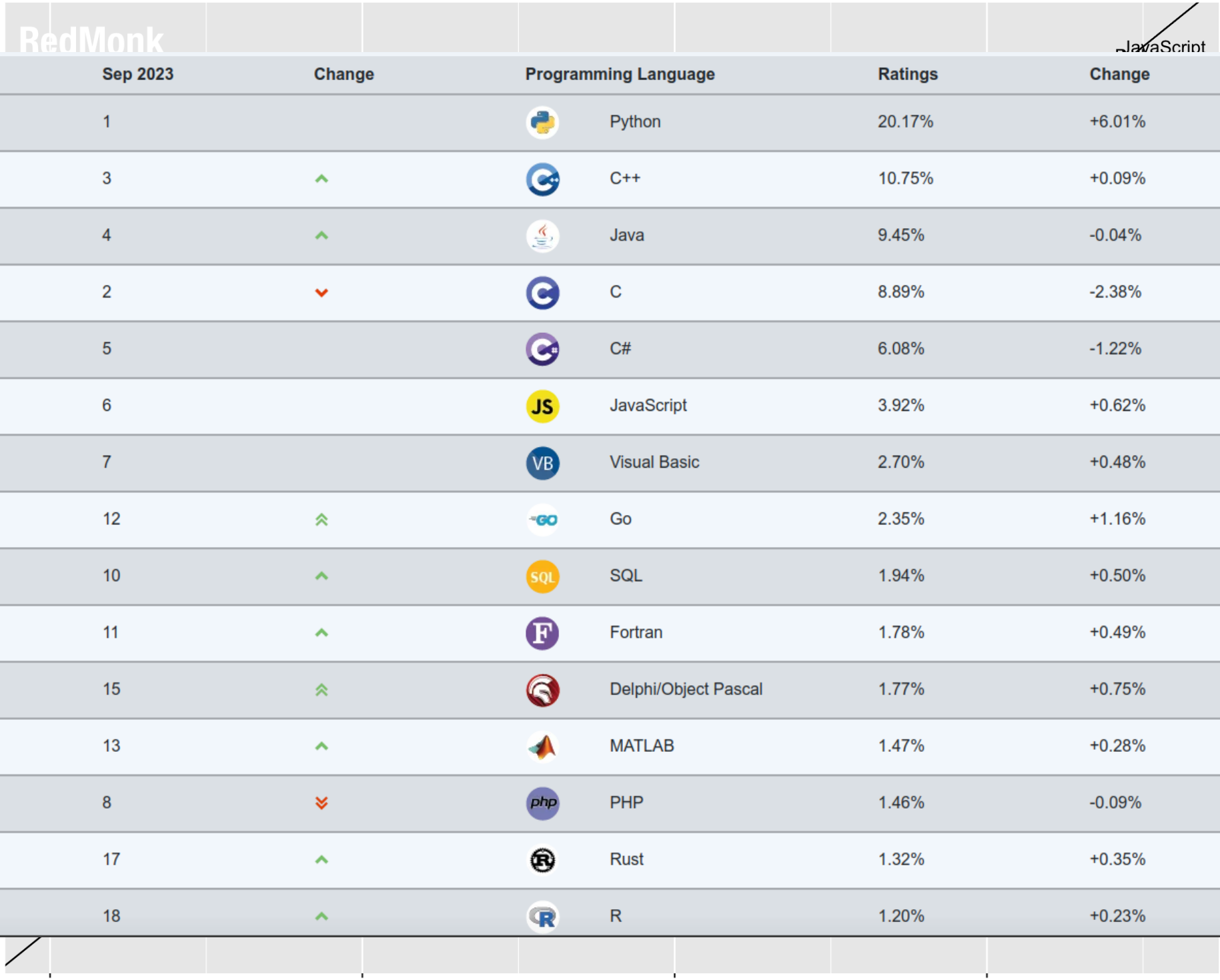
Обработка и визуализация данных в науках о Земле

RedMonk Q124 Programming Language Rankings



Sep 2024	Sep 2023	Change	Programming Language	Ratings	Change
1	1		Python	20.17%	+6.01%
2	3	↑	C++	10.75%	+0.09%
3	4	↑	Java	9.45%	-0.04%
4	2	↓	C	8.89%	-2.38%
5	5		C#	6.08%	-1.22%
6	6		JavaScript	3.92%	+0.62%
7	7		Visual Basic	2.70%	+0.48%
8	12	⇈	Go	2.35%	+1.16%
9	10	↑	SQL	1.94%	+0.50%
10	11	↑	Fortran	1.78%	+0.49%
11	15	⇈	Delphi/Object Pascal	1.77%	+0.75%
12	13	↑	MATLAB	1.47%	+0.28%
13	8	⇊	PHP	1.46%	-0.09%
14	17	↑	Rust	1.32%	+0.35%
15	18	↑	R	1.20%	+0.23%

Python“Pros”

- Скриптовость
- Кроссплатформенность
- Динамическая типизация
- Автоуправление памятью
- Open Source (PSF License Agreement)
- Обширная встроенная библиотека модулей (“батарейки в комплекте”)
- Централизованный каталог дополнительных модулей
- Огромное количество информации

Дистрибутивы, документация

python.org

docs.python.org

www.anaconda.com/distribution/

Учебники, статьи и прочее...

python.ru

pythonworld.ru

docs.python-guide.org/scenarios/scientific/

www.python-course.eu

realpython.com

book.pythontips.com

Альтернативы

- Matlab
- Octave (octave.org) – свободная альтернатива Matlab
- Julia (julialang.org) – производительность
- R (www.r-project.org) – статистика

Программа курса

- Синтаксис языка Python
- Обзор стандартной библиотеки
- Вычисления и обработка данных: Nympy, Scipy, PANDAS
- Визуализация, графики: Matplotlib, Seaborn, Basemap, Cartopy ….
- Специализированные модули: OBSPy ….

ocean.phys.msu.ru/courses/geodata/

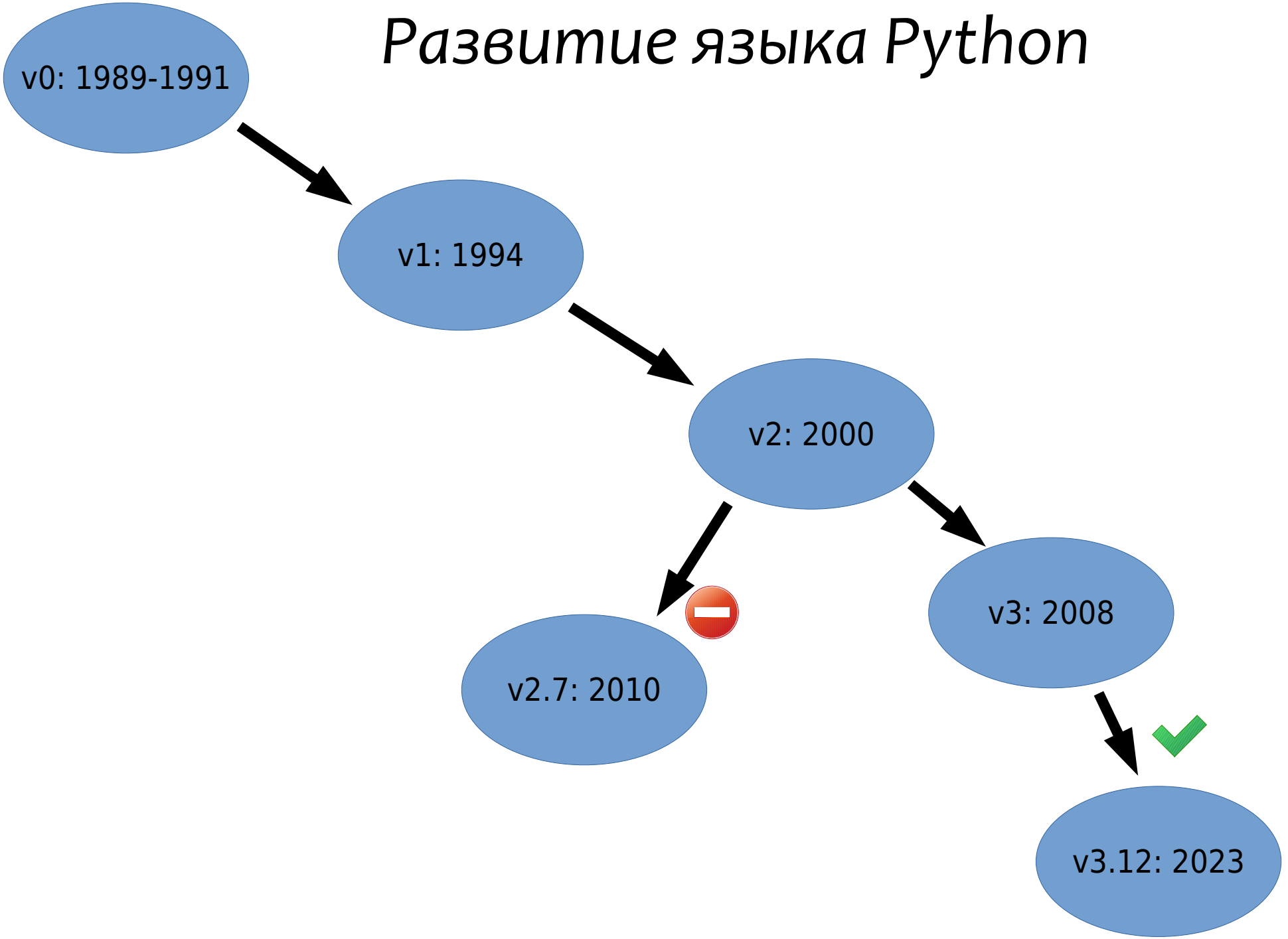
python.org

Guido van Rossum

Философские постулаты (The Zen)

- Красивое лучше, чем уродливое.
- Явное лучше, чем неявное.
- Простое лучше, чем сложное.
- Сложное лучше, чем запутанное.
- Плоское лучше, чем вложенное.
- Разреженное лучше, чем плотное.
- Читаемость имеет значение.
- Особые случаи не настолько особые, чтобы нарушать правила.
- При этом практичность важнее безупречности.
- Ошибки никогда не должны замалчиваться.
- Если не замалчиваются явно.
- Встретив двусмысленность, отбрось искушение угадать.
- Должен существовать один — и, желательно, только один — очевидный способ сделать это.
- Хотя он поначалу может быть и не очевиден, если вы не голландец.
- Сейчас лучше, чем никогда.
- Хотя никогда зачастую лучше, чем прямо сейчас.
- Если реализацию сложно объяснить — идея плоха.
- Если реализацию легко объяснить — идея, возможно, хороша.
- Пространства имён — отличная вещь! Давайте будем делать их больше!

Развитие языка Python



Пример программы

```
import datetime as dt

def make_string(*args):
    '''
    comment
    '''
    return  ' '.join(args)

a = 'Hello'
b = 'world'
c = dt.date.today()

print(make_string(a, b, str(c), '!'))
```

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       await
break      except     in         raise      async
match      case
```

Built-in Functions in Python						
abs()	classmethod()	filter()	id()	max()	property()	str()
all()	compile()	float()	input()	memoryview()	range()	sum()
any()	complex()	format()	int()	min()	repr()	super()
ascii()	delattr()	frozenset()	isinstance()	next()	reversed()	tuple()
bin()	dict()	getattr()	issubclass()	object()	round()	type()
bool()	dir()	globals()	iter()	oct()	set()	vars()
bytearray()	divmod()	hasattr()	len()	open()	setattr()	zip()
bytes()	enumerate()	hash()	list()	ord()	slice()	__import__()
callable()	eval()	help()	locals()	pow()	sorted()	
chr()	exce()	hex()	map()	print()	staticmethod()	

Типы данных

- None
- Числа
- Строки
- Байты
- Списки (lists)
- Кортежи (tuples)
- Словари (dictionaries)
- Множества (sets)

Типы данных: None

```
null_variable = None
not_null_variable = 'Hello There!'

# The is keyword
if null_variable is None:
    print('null_variable is None')
else:
    print('null_variable is not None')

if not_null_variable is None:
    print('not_null_variable is None')
else:
    print('not_null_variable is not None')


# The == operator
if null_variable == None:
    print('null_variable is None')
else:
    print('null_variable is not None')

if not_null_variable == None:
    print('not_null_variable is None')
else:
    print('not_null_variable is not None')
```

Типы данных: числа

int 25 0x19 0o31 0b10101

float 25. 2.5e+1

complex 25.j 2.5e+1j

x + y	x – y	x * y	x / y	x // y	x % y	-x
abs(x)	int(x)	float(x)	complex(re, im)		c.conjugate()	
divmod(x, y)		pow(x,y)		x ** y		
x | y	x ^ y	x & y	x << n	x >> n	~x	

Типы данных: строки

```
'allows embedded "double" quotes'
"allows embedded 'single' quotes".
'''Three single quotes''', """Three double quotes"""
```

s.capitalize	s.format_map	s.isprintable	s.partition	s.splitlines
s.casefold	s.index	s.isspace	s.replace	s.startswith
s.center	s.isalnum	s.istitle	s.rfind	s.strip
s.count	s.isalpha	s.isupper	s.rindex	s.swapcase
s.encode	s.isdecimal	s.join	s.rjust	s.title
s.endswith	s.isdigit	s.ljust	s.rpartition	s.translate
s.expandtabs	s.isidentifier	s.lower	s.rsplit	s.upper
s.find	s.islower	s.lstrip	s.rstrip	s.zfill
s.format	s.isnumeric	s.maketrans	s.split	

Типы данных: строки

```
>>> name = "Fred"
>>> f"He said his name is {name}."
"He said his name is Fred."

>>> width = 10
>>> precision = 4
>>> value = decimal.Decimal("12.34567")
>>> f"result: {value:{width}.{precision}}"
'result:      12.35'

>>> '_'.join(('1','f','qwerty'))
'1_f_qwerty'

>>> '1_2_44'.split('_')
['1', '2', '44']

>>> '1_2_44'.replace('_', ' ')
'1 2 44'
```

Типы данных: списки

```
[], [a], [a, b, c], [x for x in iterable], list(), list(iterable)
```

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

>>> list[0]
abcd
>>> list[1:3]
[786, 2.23]
>>> list[2:]
[2.23, 'john', 70.200000000000003]
>>> list[-2]
john
```

s[i] = x	s[i:j] = t	del s[i:j]	s[i:j:k] = t	del s[i:j:k]
s.append(x)	s.clear()	s.copy()	s.extend(t) s += t	s *= n
s.insert(i, x)	s.pop([i])	s.remove(x)	s.reverse()	

Особенности использования памяти

```
>>> a = []

>>> id(a)
139677037319872

>>> b = a

>>> id(b)
139677037319872

>>> c = a.copy()

>>> id(c)
139676949584128

>>> b.append(1)

>>> a
[1]

>>> b
[1]
```

Списки a и b используют “общую память”.
Для создания самостоятельной копии надо использовать .copy()

Особенности использования памяти

```
>>> c=2

>>> d=c

>>> id(c)
140188934422800

>>> id(d)
140188934422800

>>> d=3

>>> c
2

>>> d
3

>>> id(d)
140188934422832
```

На самом деле, объектами являются даже не переменные, а их значения!

Типы данных: кортежи

```
(), (a), (a, b, c), (x for x in iterable), tuple(), tuple(iterable)

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

>>> tuple[0]
abcd
>>> tuple[1:3]
(786, 2.23)
>>> tuple[2:]
(2.23, 'john', 70.200000000000003)
>>> tuple[-2]
john
```

Генераторы

(x for x in iterable)

```
>>> gen = (2**i for i in range(1,5))
>>> next(gen)
2
>>> next(gen)
4
>>> next(gen)
8
>>> next(gen)
16
>>> next(gen)
---------------------------------------------------------------------------
StopIteration
```

```
>>> gen = (2**i for i in range(1,5))

>>> for i in gen:
...     print(i)

2
4
8
16
```

Типы данных: словари

```
dict(one=1, two=2, three=3)
{'one': 1, 'two': 2, 'three': 3}
dict(zip(['one', 'two', 'three'], [1, 2, 3]))
dict([('two', 2), ('one', 1), ('three', 3)])
dict({'three': 3, 'one': 1, 'two': 2})
```

a.clear	a.get	a.pop	a.update
a.copy	a.items	a.popitem	a.values
a.fromkeys	a.keys	a.setdefault	

Типы данных: словари

```
>>> dishes = {'eggs': 2, 'sausage': 1, 'bacon': 1, 'spam': 500}
>>> keys = dishes.keys()
>>> values = dishes.values()

>>> # iteration
>>> n = 0
>>> for val in values:
...     n += val
>>> print(n)
504


>>> list(keys)
['eggs', 'bacon', 'sausage', 'spam']
>>> list(values)
[2, 1, 1, 500]


>>> del dishes['eggs']
>>> del dishes['sausage']
>>> list(keys)
['spam', 'bacon']
```

Типы данных: множества

```
{}, {a}, {a, b, c}, {x for x in iterable}, set(), set(iterable)
```

```
>>>s = {'a','e','i','o'}

>>>set('some string') & s
 {'e', 'i', 'o'}

>>>set('some string') ^ s
{' ', 'a', 'g', 'm', 'n', 'r', 's', 't'}
```

s.add	s.difference_update		s.isdisjoint		s.remove
s.clear	s.discard	s.issubset	s.symmetric_difference		
s.copy	s.intersection	s.issuperset	s.symmetric_difference_update		
s.difference	s.intersection_update		s.pop	s.union	s.update

Условия

```
if x < 0:
     x = 0
     print('Negative changed to zero')
elif x == 0:
     print('Zero')
elif x == 1:
     print('Single')
else:
     print('More')
```

```
A = Y if X else Z
```

Циклы

```
i = 5
while i < 15:
    print(i)
    i = i + 2
```

```
for i in 'hello world':
    if i == 'o':
        break
    print(i, end='')
```

```
for i in 'hello world':
    if i == 'a':
        break
    else:
        print('Буквы а в строке нет')
```

Ввод / вывод

```
fo = open("foo.txt", "wb")
print ("Name of the file: ", fo.name)
print ("Closed or not : ", fo.closed)
print ("Opening mode : ", fo.mode)
fo.close()
```

fo.buffer	fo.errors	fo.mode	fo.readline	fo.truncate
fo.close	fo.fileno	fo.name	fo.readlines	fo.writable
fo.closed	fo.flush	fo.newlines	fo.seek	fo.write
fo.detach	fo.isatty	fo.read	fo.seekable	fo.writelines
fo.encoding	fo.line_buffering		fo.readable	fo.tell

Ввод / вывод

```
# Write a file
with open("test.txt", "wt") as out_file:
    out_file.write("This Text is going to out file!")

# Read a file
with open("test.txt", "rt") as in_file:
    text = in_file.read()

print(text)
```

```
>>> a=input()
1234

>>> a
Out[33]: '1234'
```

Функции

```
def functionname( parameters ):
   "function_docstring"
   code
   return [expression]
```

```
def changeme(mylist):
   "This changes a passed list into this function"
   print ("Values inside the function before change: ", mylist)
   mylist[2]=50
   print ("Values inside the function after change: ", mylist)
   return


mylist = [10,20,30]
changeme(mylist)
print ("Values outside the function: ", mylist)
```

Функции

```
def printinfo( name="Alex", age=35 ):
   "This prints a passed info into this function"
   print ("Name: ", name)
   print ("Age ", age)


printinfo(age = 50, name = "miki")
printinfo(name = "miki")
printinfo(age = 50)
printinfo(50, "miki")
```

Функции

```
def printinfo( arg1, *vartuple, **kwarg ):
    "This prints a variable passed arguments"
    print("Output is: ")
    print('arg1', arg1)
    for var in vartuple:
       print(var)
    for i in kwarg:
        print(i, kwarg[i])
    return


printinfo(10, q=15)
printinfo(70, 60, 50 ,e=3, r=0)
```

```
arg1 10
q 15

arg1 70
60
50
e 3
r 0
```

Лямбда-функции

```
x = lambda a : a + 10
```

```
x = lambda a, b : a * b
```

Функции-генераторы

```
>>> def gen(a, b):
...     for i in range(a):
...         yield 2**i / b


>>> gf = gen(7, 5)

>>> for i in gf:
...     print(i)

0.2
0.4
0.8
1.6
3.2
6.4
12.8
```

```
>>> gf = gen(2, 2)

>>> next(gf)
0.5

>>> next(gf)
1.0

>>> next(gf)
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
```

Область видимости переменных

```
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

```
After local assignment: test spam
After nonlocal assignment: nonlocal spam
After global assignment: nonlocal spam
In global scope: global spam
```

Классы

```
class MyClass:
    """A simple example class"""
    i = 12345

    def __init__(self, A):
        self.A = A

    def f(self):
        return self.i, self.A


q = MyClass(0)

print(q.f())
```

isinstance	issubclass	getattr	setattr	delattr

Исключения

```
try:
    some_code
except (A, B) as e:
    code_1
except C:
    code_2
finally:
    common_code
```

```
while True:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print("Oops!  That was no valid number. Try again…")
    except KeyboardInterrupt:
        print('Bye!')
        break
```

Исключения

```
with A() as a, B() as b:
    code
```

```
>>> try:
...     raise NameError('HiThere')
... except NameError:
...     print('An exception flew by!')
...     raise
...
An exception flew by!
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
NameError: HiThere
```

Исключения

```
class B(Exception):
    pass

class C(B):
    pass

class D(C):
    pass

for cls in [B, C, D]:
    try:
        raise cls()
    except D:
        print("D")
    except C:
        print("C")
    except B:
        print("B")
```

Python 3.8

Операция присваивания внутри выражений :=

```
if (n := len(a)) > 10:
    print(f"List is too long ({n} elements, expected <= 10)")
```

Можно указать, какие параметры функций можно передавать через синтаксис именованных аргументов, а какие нет

```
def f(a, b, /, c, d, *, e, f):
    print(a, b, c, d, e, f)

f(10, 20, 30, d=40, e=50, f=60) # OK
f(10, b=20, c=30, d=40, e=50, f=60)  # ошибка, `b` не может быть
                                     # именованным аргументом
f(10, 20, 30, 40, 50, f=60)          # ошибка, `e` обязан быть
                                     # именованным аргументом
```

Python 3.10

Новый оператор сопоставления с образцом

```
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 401|403|404:
            return "Not allowed"
        case 418:
            return "I'm a teapot"
        case _:
            return "Something else"
```