

# 十五分钟让你了解Python套路



Mr\_\_C (/u/926ad592492b) [+ 关注](#)  
2016.06.07 23:18\* 字数 6082 阅读 10978 评论 52 喜欢 532  
(/u/926ad592492b)

Mr.C/文

古人云：书山有路勤为径 学海无涯苦作舟。

注：一般标榜着“XX天学会XX”、“XX分钟了解XX”的文章都不会是让你增长功力的文章,如本文。

随着互联网发展，全球在快速数字化，编程这件事也从数十年前只有科研人研在实验室才能做的事变得越来越让人们所熟悉。但是作为平时甚少接触编程的小伙伴，是不是也会对计算机世界拥有一点点好奇心呢（比如电影《黑客帝国》、《源代码》）？

## Q1: "怎么进入编程世界?"

首先，你要学会一门编程语言！

而我要推荐的计算机语言，答案已在标题中被出卖：Python!

Python是一门十分容易上手，但是又被IT业界广泛使用的编程语言（Scratch这种图形化编程虽然简单，但基本只能用于教学）。2016年5月份Python在全球最受欢迎编程语言中排第五。

编程语言排行榜 TOP 20 榜单：

May 2016	May 2015	Change	Programming Language	Ratings	Change
1	1		Java	20.956%	+4.09%
2	2		C	13.223%	-3.62%
3	3		C++	6.698%	-1.18%
4	5	▲	C#	4.481%	-0.76%
5	6	▲	Python	3.789%	+0.06%
6	9	▲	PHP	2.992%	+0.27%
7	7		JavaScript	2.340%	-0.79%
8	15	▲	Ruby	2.338%	+1.07%
9	11	▲	Perl	2.326%	+0.51%
10	8	▼	Visual Basic .NET	2.325%	-0.64%
11	13	▲	Delphi/Object Pascal	2.008%	+0.71%
12	22	▲	Assembly language	1.883%	+1.12%
13	10	▼	Visual Basic	1.828%	-0.07%
14	4	▼	Objective-C	1.597%	-3.80%
15	18	▲	Swift	1.593%	+0.48%
16	12	▼	R	1.334%	-0.11%
17	38	▲	Groovy	1.288%	+0.90%
18	14	▼	MATLAB	1.287%	+0.00%
19	17	▼	PL/SQL	1.208%	+0.08%
20	30	▲	D	0.975%	+0.39%

2016年5月份 TIOBE 排行榜

## Q2: “Python如何学？”



如果你是一位从未接触过编程语言的新手，请移步 [这里](http://learnpythonthehardway.org/book/intro.html)

(<http://learnpythonthehardway.org/book/intro.html>)。此书作者是一位程序员大叔，大叔说了：“学编程，首先，得会打字！！不会打字您就别学了哈”。大叔的在书提出的观点个人觉得也是值得学习的，无论用于编程还是其他事情，如：

As you study this book, and continue with programming, remember that anything worth doing is difficult at first. Maybe you are the kind of person who is afraid of failure so you give up at the first sign of difficulty. Maybe you never learned self-discipline so you can't do anything that's "boring." Maybe you were told that you are "gifted" so you never attempt anything that might make you seem stupid or not a prodigy. Maybe you are competitive and unfairly compare yourself to someone like me who's been programming for more than 20 years.

Whatever your reason for wanting to quit, keep at it. Force yourself. If you run into a Study Drill you can't do, or a lesson you just do not understand, then skip it and come back to it later. Just keep going because with programming there's this very odd thing that happens. At first, you will not understand anything. It'll be weird, just like with learning any human language. You will struggle with words, and not know what symbols are what, and it'll all be very confusing. Then one day BANG your brain will snap and you will suddenly "get it." If you keep doing the exercises and keep trying to understand them, you will get it. You might not be a master coder, but you will at least understand how programming works."

万事开头难，但自己选择的路，即使跪着也要走完。也许你是那种一碰到困难就想到放弃的人；也许你从未掌握“自律”这件事导致你无法做任何“枯燥”的事情；也许你一直被认为是“天才”，所以你不会傻到尝试去做那些有可能让你看起来是一个笨蛋或者至少不是“神童”的事情；也许你觉得和我这种已经编程超过20年的人比较编程这件事是一件逗逼的事情。

无论是什么原因使你想要退出，不要管它！要管住自己（译者注：论“自虐”能力的重要性。）！如果某一个练习你不会做，或者有一课你听不明白，那就跳过它，然后再回来重新学习（译者注：子曰：温故而知新）。编程这事说起来是件奇怪的事情，当你没有领悟真谛的时候你只需要坚持。像学习任何一门自然语言一样，刚开始时可能总会感觉怪怪的。那些奇怪的单词，那些你不知道的符号，也许会让你感到十分困惑。但只要你坚持不懈，坚持做本书中的那些练习并且尝试着去理解它们。有一天，你的脑子会突然闪出“哇哈~！原来是这样！”的顿悟时刻。也许你不会成为一个高级码农，但是至少你将会知道编程到底是什么鬼。

言归正传，本文面向对象为具有一丁点编程经验的小伙伴，旨在快速了解Python的基本语法和部分特性。

## 前言

```
# Python中单行注释请用'#'  
""" Python中多行注释  
    请用"""，我写不了那么  
    多字，随便凑个样板。  
    """
```

## 1. 基本类型和运算符



```

# 定义了一个数字 3
3 # => 3

# 基本计算
1 + 1 # => 2
8 - 1 # => 7
10 * 2 # => 20
35 / 5 # => 7

# 当除数和被除数都为整型时，除 这个操作只求整数
# ( python2.x语法。经测试，Python3.x 已经全部当做浮点数处理，还会计算小数)
5 / 2 # => 2
10/-3 #python3的结果为-3.3333333333333335 python2 结果为-4
10/3 #python3的结果为3.3333333333333335 python2 结果为3
#由上面两个结果也可以看出，在Python2中，如结果有小数，则会取最近最小整数

# 如果我们除数和被除数为浮点型，则Python会自动把结果保存为浮点数
2.0 # 这是浮点数
11.0 / 4.0 # 这个时候结果就是2.75啦！是不是很神奇？

# 当用'/'进行计算时，python3不会全部单做浮点数处理。
5 // 3 # => 1
5.0 // 3.0 # => 1.0
-5 // 3 # => -2
-5.0 // 3.0 # => -2.0

from __future__ import division # 注可以在通过 __future__ 关键字
# 在python2中引入python3 特性

11/4 # => 2.75 ... 标准除法
11//4 # => 2 ... 除后取整

# 求余数操作
7 % 3 # => 1

# 幂操作 2的4次方
2**4 # => 16

# 先乘除，后加减，括号优先
(1 + 3) * 2 # => 8

# 布尔值操作
# 注：or 和 and 两个关键字是大小写敏感的
True and False #=> 返回False
False or True #=> 返回True

# 布尔值和整形的关系，除了0外，其他都为真
0 and 2 #=> 0
-5 or 0 #=> -5
0 == False #=> True
2 == True #=> False
1 == True #=> True

# not 操作
not True # => False
not False # => True

#等值比较 "=="，相等返回值为True，不相等返回False
1 == 1 # => True
2 == 1 # => False

# 非等比较 "!="，如果两个数不相等返回True，相等返回False
1 != 1 # => False
2 != 1 # => True

# 大于/小于 和等于的组合比较
1 < 10 # => True
1 > 10 # => False
2 <= 2 # => True
2 >= 2 # => True

# Python可以支持多数值进行组合比较，
#但只要一个等值为False，则结果为False
1 < 2 < 3 # => True
2 < 3 < 2 # => False

# 可以通过 " 或者 '来创建字符串
"This is a string."
'This is also a string.'

# 字符串间可以通过 + 号进行相加，是不是简单到爆？
"Hello " + "world!" # => "Hello world!"
# 甚至不使用 '+'号，也可以把字符串进行连接
"Hello " "world!" # => "Hello world!"

```



```

#可以通过 * 号,对字符串进行复制,比如 ;
importantNote = "重要的事情说三遍\n" * 3
print (importantNote)
""" 结果为:
重要的事情说三遍
重要的事情说三遍
重要的事情说三遍
"""

"Hello" * 3 # => "HelloHelloHello"

# 字符串可以在任意位置被打断
"This is a string"[0] # => 'T'

#字符串可以用 %连接,并且可以打印出变量值
# (和C/C++ 一样%d 表示整数,%s表示字符串,
#但python可以自己进行判断,我们无需太担心这个问题)
x = 'apple'
y = 'lemon'
z = "The items in the basket are %s and %s" % (x,y)

# 一个新的更好的字符串连接方式是通过.format()函数,推荐使用该方式
"{ } is a { }".format("This", "placeholder")
"{0} can be {1}".format("strings", "formatted")
# You can use keywords if you don't want to count.
"{name} wants to eat {food}".format(name="Bob", food="lasagna")

# None是一个对象,None就是None,它是一个特殊的变量
None # => None

# 在和None进行比较时,不要用"=="操作符,用 "is"
"etc" is None # => False
None is None # => True

#"is"操作符用于对象之间的比较,
#对于底层类型进行比较时
#不建议用"is",但对于对象之间的比较,用"is"是最合适的
# bool可以用于对任何对象进行判断
# 以下这些值是非真的
# - None
# - 各类数值型的0 (e.g., 0, 0L, 0.0, 0j)
# - 空元组、空列表 (e.g., '', (), [])
# - 空字典、空集合 (e.g., {}, set())
# - 其他值请参考:
#   https://docs.python.org/2/reference/datamodel.html#object.__nonzero__
#
# All other values are truthy (using the bool() function on them returns True).
bool(0) # => False
bool("") # => False

```

## 2. 变量和集合

```

# 打印 print()
print ("I'm Python. Nice to meet you!") # => I'm Python. Nice to meet you!

# 从控制台中获取输入
input_string_var = raw_input("Enter some data: ") # 返回字符串类型
input_var = input("Enter some data: ") # python会判断类型如果是字符串 则输入时要加""
# 注意:在 python 3中, input() 由 raw_input() 代替

# 在Python中不需要设定变量类型,python会自动根据值进行判断
some_var = 5
some_var # => 5

# if 可以作为表达时被使用,下句可以这样理解 "输出'yahoo!'如果3大于2的话,不然输出2"
"yahoo!" if 3 > 2 else 2 # => "yahoo!"

```

## 列表



```

# python中的列表定义
li = []
# 也可以通过初始化时内置列表的值
other_li = [4, 5, 6]

# append函数可以在列表中插入值
li.append(1)    # li is now [1]
li.append(2)    # li is now [1, 2]
li.append(4)    # li is now [1, 2, 4]
li.append(3)    # li is now [1, 2, 4, 3]
# pop函数从列表末移除值
li.pop()        # => 3 and li is now [1, 2, 4]
# 移除后通过append接回
li.append(3)    # li is now [1, 2, 4, 3] again.

# 通过[]的方式可以提取任何列表中的任意值
# (前提, index不大于列表总数)
li[0] # => 1
# 也可以通过[]下标的方式直接给列表赋值
li[0] = 42
li[0] # => 42
# 如果[]小标的值为负数, 则表示以逆序获取列表中的值
li[-1] # => 3

# 查询的值不可以超出列表个数, 否则报错。
# 但是利用insert()插入时可以, 超出范围的值会被插入到列表最末
li[4] # Raises an IndexError

# 可以通过[:], 获取列表中指定范围的值
# (It's a closed/open range for you mathy types.)
# 这是半开取值法, 比如li[1:3], 取的是列表中index为1、2的两个值,
# 该法则适用于以下所有通过[]取值的方式
li[1:3] # => [2, 4]
# 如果一边不去值, 则表示取所有该边的值。
li[2:] # => [4, 3]
li[:3] # => [1, 2, 4]

# [:2]表示选择从[0]开始, 步长为2上的值
li[:2] # => [1, 4]
# [::-1]表示反向选择, -可以理解为 反向选择, 而1表示步长, 步长1则包含了列表中的所有元素
li[::-1] # => [3, 4, 2, 1]
# []规则完整版表示方法[开始:结束:步长]
# li[start:end:step]

# "del"关键字可以直接删除列表中的值
del li[2] # li is now [1, 2, 3]

# 可以通过"+"操作符对列表进行操作, 注: 列表只有 + 操作, 而集合 (set) 有+ 和 -
li + other_li # => [1, 2, 3, 4, 5, 6]

# 也可以 "extend()"方法对列表进行扩展
li.extend(other_li) # Now li is [1, 2, 3, 4, 5, 6]

# Remove 方法和 del 类似, 但remove的直接是数值, 而不是index
li.remove(2) # li is now [1, 3, 4, 5, 6]
li.remove(2) # 如果remove的值不存在列表中, 则会报错

# 在指定位置插入数值, 上面已经提过, 如果index值超过的话, 会直接插到列表末
li.insert(1, 2) # li is now [1, 2, 3, 4, 5, 6] again

# 获取某个值的index
li.index(2) # => 1
li.index(7) # 如果

# "in"可以直接查看某个值是否存在于列表中
1 in li # => True

# "len()"函数可以检测队列的数量
len(li) # => 6

```

## 元组



```
# Tuples (元组) 是一个类似数列的数据结构，但是元组是不可修改的
tup = (1, 2, 3)
tup[0] # => 1
tup[0] = 3 # 一修改就会报错

# 数列中的方法在元组也可以使用（除了 修改）
len(tup) # => 3
tup + (4, 5, 6) # => (1, 2, 3, 4, 5, 6)
tup[:2] # => (1, 2)
2 in tup # => True

# 可以一次性赋值几个变量
a, b, c = (1, 2, 3) # a 为1, b为2, c为3
d, e, f = 4, 5, 6 # 元组赋值也可以不用括号
# 同样元组不用括号也同样可以创建
g = 4, 5, 6 # => (4, 5, 6)
# Python中的数据交换十分简单：只要在赋值时互调位置即可
e, d = d, e # d is now 5 and e is now 4
```

## 字典

```
# Python中的字典定义
empty_dict = {}
# 也可以通过定义时赋值给字典
filled_dict = {"one": 1, "two": 2, "three": 3}

# 可以通过[]的key方式查询字典中的值
filled_dict["one"] # => 1

# 可以通过"keys()"方法获取字典中的所有key值
filled_dict.keys() # => ["three", "two", "one"]
# Note - 返回的keys并不一定按照顺序排列的。
# 所以测试结果可能和上述结果不一致

# 通过 "values()"的方式可以获取字典中所有值，
# 同样他们返回的结果也不一定按照顺序排列
filled_dict.values() # => [3, 2, 1]

# 可以通过 "in"方式获取查询某个键值是否存在字典中，但是数值不可以
"one" in filled_dict # => True
1 in filled_dict # => False

# 查找不存在的key值时，Python会报错
filled_dict["four"] # KeyError

# 用 "get()" 方法可以避免键值错误的产生
filled_dict.get("one") # => 1
filled_dict.get("four") # => None
# 当键值不存在的时候，get方法可以通过返回默认值，
# 但是并没有对值字典进行赋值
filled_dict.get("one", 4) # => 1
filled_dict.get("four", 4) # => 4

# 字典中设置值的方式和列表类似，通过[]方式可以设置
filled_dict["four"] = 4 # now, filled_dict["four"] => 4

# "setdefault()" 可以设置字典中的值
# 但是注意：只有当该键值之前未存在的时候，setdefault() 函数才生效
filled_dict.setdefault("five", 5) # filled_dict["five"] is set to 5
filled_dict.setdefault("five", 6) # filled_dict["five"] is still 5
```

## 集合



```
empty_set = set()
# 初始化set的方式可以通过 set()来实现
some_set = set([1, 2, 2, 3, 4]) # some_set is now set([1, 2, 3, 4])

# 集合的排列是无序的！集合的排列是无序的！集合的排列是无序的！
another_set = set([4, 3, 2, 2, 1]) # another_set is now set([1, 2, 3, 4])

# Python2.7以后，{}可以用于被定义集合
filled_set = {1, 2, 2, 3, 4} # => {1, 2, 3, 4}

# Add方法可用于增加集合成员
filled_set.add(5) # filled_set is now {1, 2, 3, 4, 5}

#集合可通过 &操作符取交集
other_set = {3, 4, 5, 6}
filled_set & other_set # => {3, 4, 5}

# 通过|操作符取并集
filled_set | other_set # => {1, 2, 3, 4, 5, 6}

# 通过 - 操作符取差集
{1, 2, 3, 4} - {2, 3, 5} # => {1, 4}

# 通过 ^ 操作符取非集
{1, 2, 3, 4} ^ {2, 3, 5} # => {1, 4, 5}

# 通过 >= 判断左边集合是否是右边集合的超集
{1, 2} >= {1, 2, 3} # => False

# 通过 <= 判断左边集合是否是右边集合的子集
{1, 2} <= {1, 2, 3} # => True

# 通过 in 可以判断元素是否在集合中
2 in filled_set # => True
10 in filled_set # => False
```

## Python数据集合类型总结

- 列表 定义方式 li = [1,2,3,4, "Hello World"]（列表可以包含任意基本类型）
- 元组 定义方式 tup = (1,2,3,4)（和列表类似，但 **元组不可更改**）
- 字典 定义方式 dic = {"one": 2, "tow": 3, "three": 0}（字典，就是字典嘛。以 **key:value** 方式存在）
- 集合 定义方式 set=set ( 1 , 2 , 3 , 4 ) or set = {1 , 2 , 3 , 4}（集合里的元素是唯一的，集合支持 & | ^ + -操作）

## 3. Python 逻辑运算符



```

# 创建一个变量
some_var = 5

# 通过if进行逻辑判断
if some_var > 10:
    print "some_var is totally bigger than 10."
elif some_var < 10: # This elif clause is optional.
    print "some_var is smaller than 10."
else: # This is optional too.
    print "some_var is indeed 10."

"""
通过for...in...进行循环打印：
dog is a mammal
cat is a mammal
mouse is a mammal
"""
for animal in ["dog", "cat", "mouse"]:
    # You can use {} to interpolate formatted strings. (See above.)
    print "{0} is a mammal".format(animal)

"""
通过"range()" 方式，控制for的循环次数
prints:
0
1
2
3
"""
for i in range(4):
    print i

"""
"range(lower, upper)" 返回 lower 到 upper的值，
注意：range左边必须小于右边参数
prints:
4
5
6
7
"""
for i in range(4, 8):
    print i

"""
while 循环
prints:
0
1
2
3
"""
x = 0
while x < 4:
    print x
    x += 1 # Shorthand for x = x + 1

# Python支持 try/except 语法

# Python2.6以上的版本，支持try...except...:
try:
    # raise显示地引发异常。一旦执行了raise语句，raise后面的语句将不能执行。
    raise IndexError("This is an index error")
except IndexError as e:
    pass # pass 空语句，跳过处理
except (TypeError, NameError):
    pass # python 支持同时检测多个错误
else: # Python必须要处理所有情况，这里是其他未定义的情况
    print "All good!"
finally: # finally无论有没有异常都会执行
    print "We can clean up resources here"

#通过with函数，可以替代try...except...函数 [with详解](http://www.ibm.com/developer)
with open("myfile.txt") as f:
    for line in f:
        print line

```

## 4. Functions





```

# def 关键字定义函数
def add(x, y):
    print "x is {0} and y is {1}".format(x, y)
    return x + y    #可以直接return结果

# 函数调用参数
add(5, 6)    # => prints out "x is 5 and y is 6" and returns 11

# Python支持参数互换, 只需要在调用函数时加上形参
add(y=6, x=5)    # Keyword arguments can arrive in any order.

# Python函数支持可变参数
# 在定义函数时通过*号表示可变长参数
def varargs(*args):
    return args

varargs(1, 2, 3)    # => (1, 2, 3)

# 可以通过**的方式定义Key可变长参数查找字典中的关键词

def keyword_args(**kwargs):
    return kwargs

# 当函数参数是**类型的时候, Python可以通过该函数定义字典
keyword_args(big="foot", loch="ness")    # => {"big": "foot", "loch": "ness"}

#同时支持函数和字典类型参数, 具体事例如下:
def all_the_args(*args, **kwargs):
    print args
    print kwargs
"""
all_the_args(1, 2, a=3, b=4) prints:
(1, 2)
{"a": 3, "b": 4}
"""

# 在调用函数时, 可以同时赋值, 文字难以表达, 例子如下:
args = (1, 2, 3, 4)
kwargs = {"a": 3, "b": 4}
all_the_args(*args)    # equivalent to foo(1, 2, 3, 4)
all_the_args(**kwargs)    # equivalent to foo(a=3, b=4)
all_the_args(*args, **kwargs)    # equivalent to foo(1, 2, 3, 4, a=3, b=4)

# 在函数中也可以通过单独处理* 或者 **的方式, 增加函数的健壮性
def pass_all_the_args(*args, **kwargs):
    all_the_args(*args, **kwargs)
    print varargs(*args)
    print keyword_args(**kwargs)

# 全局变量 x
x = 5

def set_x(num):
    # 当在函数里面改变变量时, 如果没有加global关键字, 则改变的是局部变量
    x = num    # => 43
    print x    # => 43

def set_global_x(num):
    global x
    print x    # => 5
    x = num    # 加了global关键字后, 即可在函数内操作全局变量
    print x    # => 6

set_x(43)
set_global_x(6)

# 返回函数指针方式定义函数/*换个说法, 匿名函数*/
def create_adder(x):
    def adder(y):
        return x + y
    return adder

add_10 = create_adder(10)
add_10(3)    # => 13

# Lambda 关键字定义的匿名函数
(lambda x: x > 2)(3)    # => True
(lambda x, y: x ** 2 + y ** 2)(2, 1)    # => 5

# map方式也可以调用函数并传入参数
map(add_10, [1, 2, 3])    # => [11, 12, 13]
map(max, [1, 2, 3], [4, 2, 1])    # => [4, 2, 3]

filter(lambda x: x > 5, [3, 4, 5, 6, 7])    # => [6, 7]

```



```
# 可以通过这两种方式结合调用，下面的函数解析：  
#add_10(i) 是映射了for...in...函数的返回值，返回值作为参数传进。  
[add_10(i) for i in [1, 2, 3]] # => [11, 12, 13]  
[x for x in [3, 4, 5, 6, 7] if x > 5] # => [6, 7]
```

## 5. Python中的类



```

# 下面代码是定义了一个Human类，继承自object类
# Python类可以继承自多个类，如class Human(object,orangOutang)
class Human(object):

    # 类变量
    species = "H. sapiens"类接口
    __species = "Other.sapiens" #内部结构，无法被外部直接访问

    # __init__(),初始化函数，python中在对类进行处理时，会先处理以下函数，
    # 其实就是系统默认定义了接口，而这个接口是开放给用户去实现的，具体如下：
    # __init__ 构造函数，在生成对象时调用
    # __del__ 析构函数，释放对象时使用
    # __repr__ 打印，转换
    # __setitem__按照索引赋值
    # __getitem__按照索引获取值
    # __len__获得长度
    # __cmp__比较运算
    # __call__函数调用
    # __add__加运算
    # __sub__减运算
    # __mul__乘运算
    # __div__除运算
    # __mod__求余运算
    # __pow__称方

    def __init__(self, name):
        #声明类中的属性，并初始化，在初始化的时候同时
        #就是定义了变量类型
        self.name = name
        self.age = 0

    # 在类中所有函数都必须把self作为第一个参数
    # (下面定义的类方法和静态方法除外)
    def say(self, msg):
        return "{0}: {1}".format(self.name, msg)

    # 类方法
    @classmethod
    def get_species(cls):
        return cls.species

    # 静态方法，
    @staticmethod
    def grunt():
        return "*grunt*"

    # A property is just like a getter.
    # It turns the method age() into an read-only attribute
    # of the same name.
    #property属性，相当于getter
    @property
    def age(self):
        return self._age

    # This allows the property to be set
    @age.setter
    def age(self, age):
        self._age = age

    # This allows the property to be deleted
    @age.deleter
    def age(self):
        del self._age

#类实例化
i = Human(name="Ian")
print i.say("hi") # prints out "Ian: hi"

j = Human("Joel")
print j.say("hello") # prints out "Joel: hello"

#调用实例方法用"."
i.get_species() # => "H. sapiens"

# 改变类变量
Human.species = "H. neanderthalensis"
i.get_species() # => "H. neanderthalensis"
j.get_species() # => "H. neanderthalensis"

# 调用静态方法
Human.grunt() # => "*grunt*"

# 给age赋值
i.age = 42

```



```
# 获取age值
i.age # => 42

# 删除age
del i.age
i.age # => raises an AttributeError
```

## 6. Python的模块（库）

```
# Python中的一个*.py文件就是一个模块
import math
print math.sqrt(16) # => 4

# 可以只引入模块中的某些类/方法
from math import ceil, floor
print ceil(3.7) # => 4.0
print floor(3.7) # => 3.0

# 也可以通过*引入全部方法
# Warning: this is not recommended
from math import *

#math库的缩写可以为m
math.sqrt(16) == m.sqrt(16) # => True
# 可以直接引入sqrt库
from math import sqrt
math.sqrt == m.sqrt == sqrt # => True

#python的库就只是文件
import math
dir(math)

# If you have a Python script named math.py in the same
# folder as your current script, the file math.py will
# be loaded instead of the built-in Python module.
# This happens because the local folder has priority
# over Python's built-in libraries.

#如果你在当前目录下有一个Python脚本的名字也叫math.py
#当前目录下的math.py会替换掉内置的Python模块
#因为在Python中当前目录的优先级会高于内置模块的优先级
```

## 7. Python中的高级特性（生成器、装饰器:wraps）



```

# Generators ,生成器函数在Python中与迭代器协议的概念联系在一起。
# 简而言之,包含yield语句的函数会被特地编译成生成器。
# 当函数被调用时,他们返回一个生成器对象,这个对象支持迭代器接口。函数
#也许会有个return语句,但它的作用是用来yield产生值的。
for i in iterable:
    yield i + i

xrange_ = xrange(1, 900000000)

for i in double_numbers(xrange_):
    print i
    if i >= 30:
        break

# 装饰器wraps,wraps可以包装
# Beg will call say. If say_please is True then it will change the returned
# message
from functools import wraps

def beg(target_function):
    @wraps(target_function)
    def wrapper(*args, **kwargs):
        msg, say_please = target_function(*args, **kwargs)
        if say_please:
            return "{} {}".format(msg, "Please! I am poor :(")
        return msg

    return wrapper

@beg
def say(say_please=False):
    msg = "Can you buy me a beer?"
    return msg, say_please

print say() # Can you buy me a beer?
print say(say_please=True) # Can you buy me a beer? Please! I am poor :

```

[1]learnxinyminutes (<https://learnxinyminutes.com/docs/python/>),本文代码大部分取于该网站,在此对该网站作者表示感谢!

📖 修行 (/nb/4058487)

举报文章 © 著作权归作者所有



Mr\_C (/u/926ad592492b) ♂

写了 29640 字,被 297 人关注,获得了 815 个喜欢  
(/u/926ad592492b)

+ 关注

但行好事,莫问前程

♡ 喜欢 (/sign\_in?utm\_source=desktop&utm\_medium=not-signed-in-like-button) | 532



更多分享

(<http://cwb.assets.jianshu.io/notes/images/4284553>)



登录 (/sign\_in?utm\_source=desktop&utm\_medium=not-signed-in-comment-form)

52条评论 只看作者

按喜欢排序 按时间正序 按时间倒序



如海 (/u/0846f3827474)

14楼 · 2016.06.13 08:10

(/u/0846f3827474)

已经入门好长时间了,不知道学完基础接下来干嘛,基础已经忘光了!

👍 5人赞 🗨 回复



Mr\_\_C (/u/926ad592492b)

@如海 (/users/0846f3827474) 可以参考这个知乎问题 ( https://www.zhihu.com/question/29372574 (https://www.zhihu.com/question/29372574) ) , 找项目实现。


2016.06.13 09:53 回复

如海 (/u/0846f3827474)

@Mr\_\_C (/users/926ad592492b) 哈哈 我看了下 不错的项目 由易到难 什么时候安排练手, 先谢谢了

2016.06.17 10:22 回复

添加新评论

 于龙君 (/u/bc24268ee41f)

2楼 · 2016.06.11 21:45

(/u/bc24268ee41f)

逗, 扫一遍都不止15分钟。十五分钟连基本语法都学不会, 别说函数和类了。装饰器, 生成器, 迭代器, 哪个不得理解很长时间?

赞 回复

Mr\_\_C (/u/926ad592492b)

@于龙君 (/users/bc24268ee41f) 嗯, 所以本文开头已声明, 是针对有编程基础的同学。就是对函数、类、装饰器、生成器这些已经有了解的, 本文只是叙述上述概念在Python的实现。

2016.06.11 21:57 回复

添加新评论

 剑鱼 (/u/b1a4bbffb013)

3楼 · 2016.06.13 06:28

(/u/b1a4bbffb013)

挺清楚

赞 回复

 hoptop (/u/9ea40b5f607a)

4楼 · 2016.06.13 06:32

(/u/9ea40b5f607a)

总结的很到位

赞 回复


 苏流云 (/u/78b494f2a248)

6楼 · 2016.06.13 06:44

(/u/78b494f2a248)

很不错, 收藏

赞 回复

 木棉花芯糖 (/u/fe2e934e1b0a)

8楼 · 2016.06.13 07:11

(/u/fe2e934e1b0a)

满满的基础。

赞 回复

 简匡 (/u/6a8d639ce4af)

9楼 · 2016.06.13 07:14

(/u/6a8d639ce4af)

学习ue

赞 回复

 马尔克聊数据 (/u/160c0051a3d8)


11楼 · 2016.06.13 07:40

(/u/160c0051a3d8)




先顶再看

👍 赞    💬 回复

 Yno糖 (/u/71a113ff2549)  
12楼 · 2016.06.13 07:41  
(/u/71a113ff2549)  
入门基础，值得收藏


👍 赞    💬 回复

 TCJ (/u/691dc22d277a)  
13楼 · 2016.06.13 07:42  
(/u/691dc22d277a)  
Ruby程序员默默路过


👍 赞    💬 回复

好简单的书 (/u/f50170c0cd7d) : @TCJ (/users/691dc22d277a) 你也写个ruby  
2016.06.13 10:26    💬 回复


✍️ 添加新评论

 小小燕 (/u/ff3ded95ee46)  
15楼 · 2016.06.13 08:14  
(/u/ff3ded95ee46)  
好

👍 赞    💬 回复

 RainStar\_ (/u/b4757fcd89e9)  
16楼 · 2016.06.13 08:14  
(/u/b4757fcd89e9)  
赞一个，收藏了！


👍 赞    💬 回复

 一缕白衣 (/u/ba041e340cb6)  
17楼 · 2016.06.13 09:30  
(/u/ba041e340cb6)  
先收藏，楼主写的很好。一直想接触python，好像openstack就是python写的


👍 赞    💬 回复

Mr\_\_C (/u/926ad592492b) : @一缕白衣 (/users/ba041e340cb6) 是的，Openstack是用Python实现的，而且目前Openstack社区也只接受Python的项目。  
2016.06.13 09:48    💬 回复

✍️ 添加新评论

 车大侠、 (/u/0d08add828cc)  
18楼 · 2016.06.13 09:32  
(/u/0d08add828cc)  
m

👍 赞    💬 回复

 墨上花开\_半夏 (/u/5552e25b4d6b)  
19楼 · 2016.06.13 09:35  
(/u/5552e25b4d6b)  
归纳的很好...

👍 赞    💬 回复



被以下专题收入，发现更多相似内容

-  首页投稿 (/c/bDHhpK?utm\_source=desktop&utm\_medium=notes-included-collection)
-  程序员 (/c/NEt52a?utm\_source=desktop&utm\_medium=notes-included-collection)
-  技能提升 (/c/84d84d88aaae?utm\_source=desktop&utm\_medium=notes-included-collection)
-  电商后台数据产品笔记 (/c/d109943b4ef3?utm\_source=desktop&utm\_medium=notes-included-collection)
-  web开发 (/c/a4d42d6c3062?utm\_source=desktop&utm\_medium=notes-included-collection)
-  IT在线课程 (/c/bb233a70a20e?utm\_source=desktop&utm\_medium=notes-included-collection)
-  Python (/c/59151e432e95?utm\_source=desktop&utm\_medium=notes-included-collection)

展开更多

