Python 数据分析与应用

第二章：Python 语言回顾

丁烨 教授
dingye@dgut.edu.cn
计算机科学与技术学院

2023-09-13

東莞理工學院
DONGGUAN UNIVERSITY OF TECHNOLOGY

目录

Python 简单回顾

基本语法

函数、类、模块

错误处理、日志处理

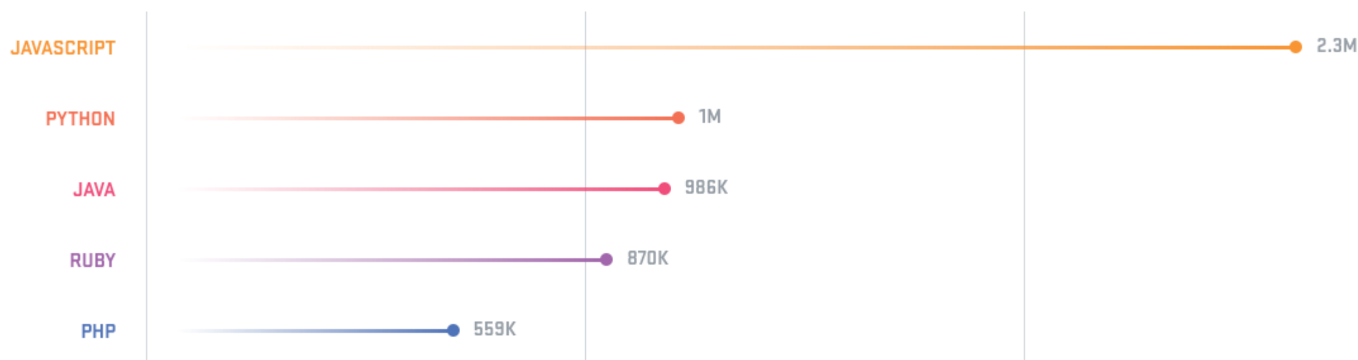
标准库

Python 简单回顾

Python 的作者

- 1991 年由 Guido van Rossum 发布
- 项目曾由 DARPA 资助研发
- https://gvanrossum.github.io/

- Computer Programming for Everybody
- https://www.python.org/doc/essays/cp4e/



Guido van Rossum ▸ Public Sep 19, 2013

Do **not** send me email like this:

""""

Hi Guido,

I came across your resume in a Google web search. You seem to have an awesome expertise on Python. I would be glad if you can reply my email and let me know your interest and availability.

....

Our client immediately needs a PYTHON Developers at its location in *, NJ. Below are the job details. If interested and available, kindly fwd me your updated resume along with the expected rate and the availability.

[...]

""""

I might reply like this:

""""

I'm not interested and not available.

""""

+1 1882 565

Python 简单回顾

Python 的设计特点

- Python 符合 UNIX 设计风格
- 最初版本是使用 Mac 开发的
- 用一种方法，最好是只有一种方法来做一件事
- Python 格言：`import this`

- Python 可以视为 LISP 的升级版
- 具备函数式编程的特点

- Python 使用缩进和空行分隔代码块

The Zen of Python

by *Tim Peters*

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

Python 简单回顾

Python 的设计特点

- Python 是一门解释器语言
- Python 其实是一种相当快的语言，但它并不像编译型语言那么快
- 官方解释器 Cpython 是用 C 语言编写的，是一个由社群驱动的自由软件

- 第三方 Python 编译器
- Nuitka：http://nuitka.net/
- PyPy：https://pypy.org/
- Cython：https://cython.org/

- PyPy 的编译结果是目前运行速度最快的，比官方解释器快 7.6 倍（via PyStone）
- `pypy3 pystone.py`

Python 简单回顾

使用 Python 解释器

- 在终端中执行:
- python3
- 或:
- python3.*

- 如果报错，可以查找已经安装的 Python:
- which python3
- 或:
- ls /usr/local/bin/python*

```
valency — valency@patriot: ~ — ssh patriot — 47×22
valency@patriot:~$ which python3
/usr/bin/python3
valency@patriot:~$ ls /usr/bin/python*
/usr/bin/python3
/usr/bin/python3-config
/usr/bin/python3-jsondiff
/usr/bin/python3-jsonpatch
/usr/bin/python3-jsonpointer
/usr/bin/python3-jsonschema
/usr/bin/python3.6
/usr/bin/python3.6-config
/usr/bin/python3.6m
/usr/bin/python3.6m-config
/usr/bin/python3.7
/usr/bin/python3.7-config
/usr/bin/python3.7m
/usr/bin/python3.7m-config
/usr/bin/python3m
/usr/bin/python3m-config
valency@patriot:~$
```

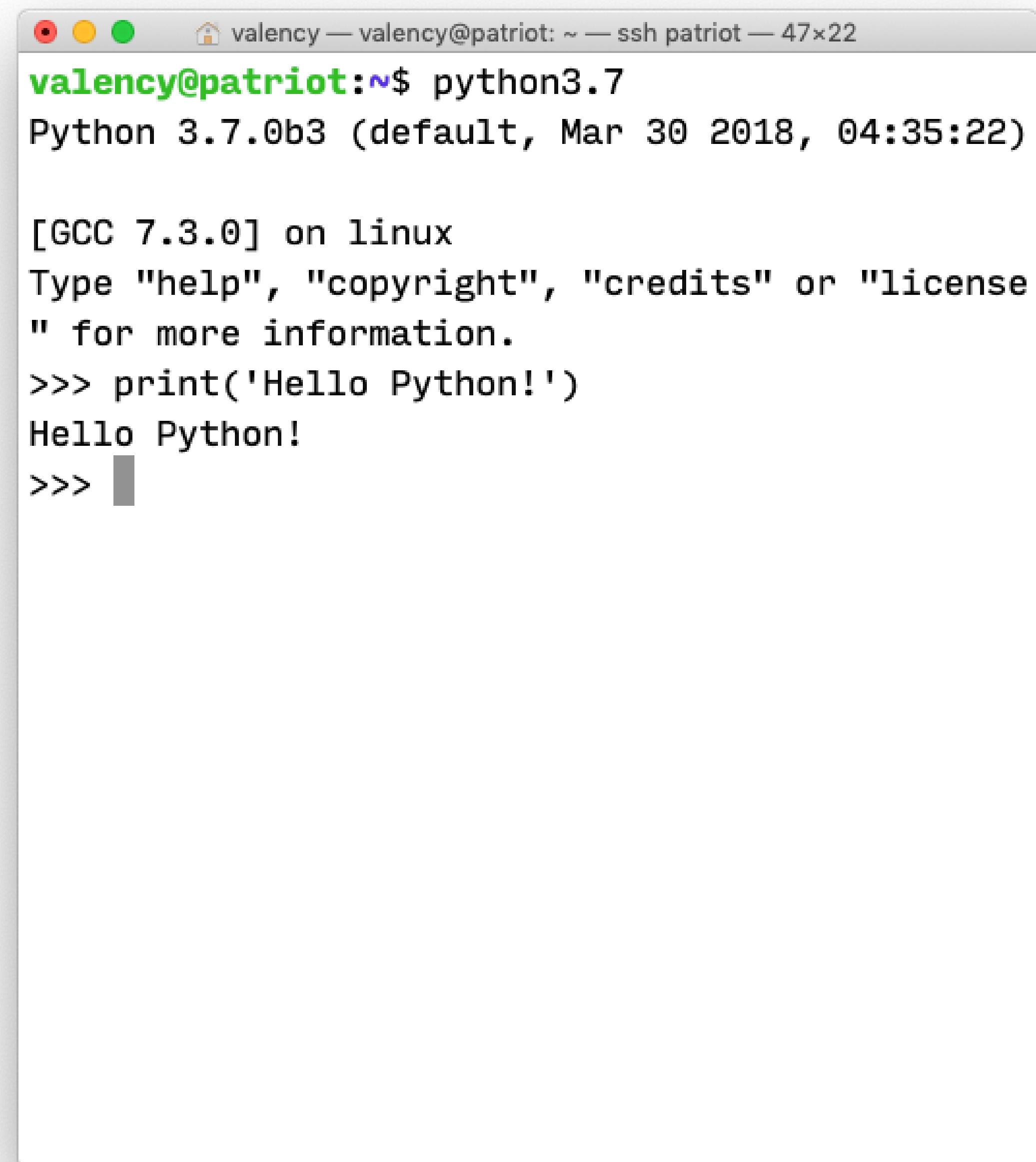
Python 简单回顾

使用 Python 解释器

- 在终端中执行：
- python3

- 直接输入 Python 代码：
- print('Hello Python!')

- 退出 Python 解释器交互模式：
- 按下 Ctrl+D 传入文件结束符（EOF），或：
- quit()



使用 Python 解释器

- 在终端中执行:
- vim test.py

- 录入 Python 代码（i）：
- print('Hello World!')

- 保存并退出（Esc 然后 :wq）：
- 在终端中执行:
- python3 test.py

❖ Vim

❖ https://www.vim.org/

- ❖ Vim 是从 vi 发展出来的一个文本编辑器
- ❖ 其代码补完、编译及错误跳转等方便编程的功能特别丰富，在程序员中被广泛使用
- ❖ 和 Emacs 并列成为类 UNIX 系统用户最喜欢的编辑器
- ❖ 对于大多数用户来说，Vim 有着一个比较陡峭的学习曲线
- ❖ 开始学习的时候可能会进展缓慢，但是一旦掌握之后，能大幅度提高编辑效率

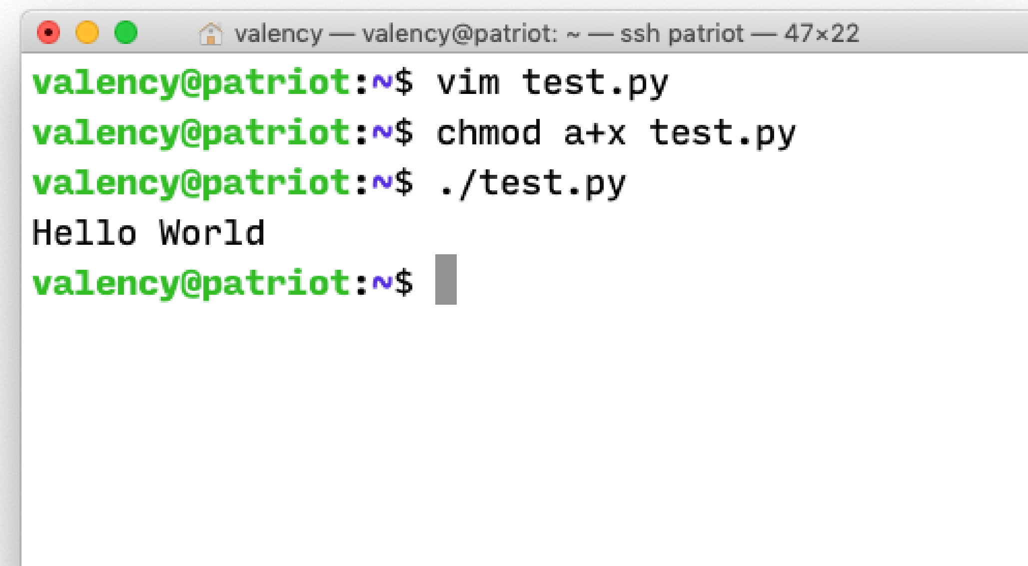
❖ Vim Cheat Sheet：https://devhints.io/vim

Python 简单回顾

使用 Python 解释器

- Shebang
- 添加 Shebang 可以让 .py 文件直接执行
- 在第一行添加 Shebang:
- #!/usr/bin/env python3

- 在终端中执行:
- chmod a+x test.py
- ./test.py



Python 简单回顾

使用 Python 解释器

- 源代码编码
- 默认情况下，Python 源文件是 UTF-8 编码
- 你也可以为源文件指定不同的字符编码

- 在 Shebang 行（首行）后插入:
- # -*- coding: utf-8 -*-

- 例如，在源代码中使用 Windows-1252 字符集:
- # -*- coding: cp-1252 -*-

valency — valency@patriot: ~ — ssh patriot — 47×22

```
#!/usr/bin/env python3.7
# -*- coding: utf-8 -*-

print('Hello World')

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
                        6,0-1         All
```

Python 简单回顾

使用 Python 解释器

- 参数传递
- 调用解释器时，脚本名和附加参数将被传入一个名为 sys.argv 的列表
- 获取这个列表：
- import sys
- sys.argv[0]：脚本名
- sys.argv[1] 以上：参数表（如有）

```
#!/usr/bin/env python3.7
# -*- coding: utf-8 -*-

import sys

print(sys.argv[0])
print('Hello', sys.argv[1])
```

```
valency@patriot:~$ ./test.py DGUT
./test.py
Hello DGUT
valency@patriot:~$ python3.7 ./test.py DGUT
./test.py
Hello DGUT
valency@patriot:~$
```

使用 Python 解释器

- 常用参数
- 模块模式：
- python3 -m test DGUT

- 禁用输出缓存（unbuffered）：
- python3 -u test.py DGUT

- 其他功能请参见：
- python3 -h

```
valency@patriot:~$ python3.7 -m test DGUT
/home/valency/test.py
Hello DGUT
valency@patriot:~$ python3.7 -u test.py DGUT
test.py
Hello DGUT
valency@patriot:~$ python3.7 -h
usage: python3.7 [option] ... [-c cmd | -m mod
| file | -] [arg] ...
Options and arguments (and corresponding enviro
nment variables):
-b     : issue warnings about str(bytes_instanc
e), str(bytearray_instance)
         and comparing bytes/bytearray with str
. (-bb: issue errors)
-B     : don't write .pyc files on import; also
 PYTHONDONTWRITEBYTECODE=x
-c cmd : program passed in as string (terminate
s option list)
-d     : debug output from parser; also PYTHOND
EBUG=x
-E     : ignore PYTHON* environment variables (
```

目录

Python 简单回顾

基本语法

函数、类、模块

错误处理、日志处理

标准库

- 基本数据类型
- 数值（Numbers）
- 字符串（String）
- 列表（List）
- 元组（Tuple）
- 字典（Dictionary）
- 集合（Set）

❖ 数值（Numbers）

❖ 包括整数、浮点数、复数

int	float	complex
10	0.0	3.14j
100	15.20	45.j
-786	-21.9	9.322e-36j
080	32.3+e18	.876j
-0490	-90.	-.6545+0J
-0x260	-32.54e100	3e+26J
0x69	70.2-E12	4.53e-7j

❖ 数值（Numbers）

❖ 数值运算

```
valency — valency@patriot: ~ — ssh -o ServerAliveInterval=60 -Y patriot — 53×24
valency@patriot:~$ python3.7
Python 3.7.0b3 (default, Mar 30 2018, 04:35:22)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for
more information.
>>> a = 10
>>> a * 0xa
100
>>> a + 1e-8
10.00000001
>>> a ** 2
100
>>>
```

Operation	Result
`x + y`	sum of *x* and *y*
`x - y`	difference of *x* and *y*
`x * y`	product of *x* and *y*
`x / y`	quotient of *x* and *y*
`x // y`	floored quotient of *x* and *y*
`x % y`	remainder of `x / y`
`-x`	*x* negated
`+x`	*x* unchanged
`abs(x)`	absolute value or magnitude of *x*
`int(x)`	*x* converted to integer
`float(x)`	*x* converted to floating point
`complex(re, im)`	a complex number with real part *re*, imaginary part *im*. *im* defaults to zero.
`c.conjugate()`	conjugate of the complex number *c*
`divmod(x, y)`	the pair `(x // y, x % y)`
`pow(x, y)`	*x* to the power *y*
`x ** y`	*x* to the power *y*

❖ 浮点数的精度

- ❖ 计算机无法准确表达无限位数的浮点数，浮点数是以二进制分数形式表达的，例如：
- ❖ 0.125 = 1/10 + 2/100 + 5/1000
- ❖ Python 的最高精度为 53 分位
- ❖ 在 Python（以及大部分程序语言）中使用浮点数比较两个数值大小一定要非常小心
- ❖ 数据（尤其是金融数据）分析应用中通常会采用整数 + 精度的方式记录浮点数，这样不仅运算速度快，而且可以避免精度问题

valency — valency@patriot: ~ — ssh -o ServerAliveInterval=60 -Y patriot — 53×24

```
>>> import math
>>> .1 + .1 + .1 == .3
False
>>> round(.1 + .1 + .1, 10) == round(.3, 10)
True
>>> round(.1, 1) + round(.1, 1) + round(.1, 1) == round(.3, 1)
False
>>> sum([0.1] * 10) == 1.0
False
>>> math.fsum([0.1] * 10) == 1.0
True
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
```

❖ 字符串（String）

- ❖ 可以用单引号或双引号标识，反斜杠转义
- ❖ 可以使用乘号快速生成字符串
- ❖ 可以直接当做字符的列表

valency — valency@patriot: ~ — ssh -o ServerAliveInterval=60 -Y patriot — 53×24

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> print('\some\name')
\some
ame
>>> print(r'\some\name')
\some\name
>>> 3 * 'un' + 'ium'
'unununium'
>>> word = 'Python'
>>> word[0]
'P'
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
```

- 列表（List）
- 声明空列表：
- a = list()
- 或：
- a = []

- 后者要快一些（via timeit）
  https://stackoverflow.com/questions/5790860/and-vs-list-and-dict-which-is-better

valency — valency@patriot: ~ — ssh -o ServerAliveInterval=60 -Y patriot — 53×24

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
>>> squares[0]
1
>>> squares[-1]
25
>>> squares[-3:]
[9, 16, 25]
>>> squares[:]
[1, 4, 9, 16, 25]
>>> squares + [36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>> squares[1] = 99
>>> squares
[1, 99, 9, 16, 25]
>>> squares.append(216)
>>> squares
[1, 99, 9, 16, 25, 216]
>>> squares[2:5] = [7, 6, 9]
>>> squares
[1, 99, 7, 6, 9, 216]
>>> [squares] * 2
[[1, 99, 7, 6, 9, 216], [1, 99, 7, 6, 9, 216]]
```

基本数据类型

- 元组（Tuple）
- 和列表类似，但是不可变（immutable）：不能给元组的一个独立的元素赋值
- 拆封赋值：类似列表，可以直接将元组直接赋值给多个变量，但不用写括号：
- p = 1, 2, 3
- a, b, c = p
- 函数的返回值通常采用拆封赋值的方式：
- a, b = f(0)

```
>>> t = 12345, 54321, 'hello!'
>>> t
(12345, 54321, 'hello!')
>>> t[0]
12345
>>> u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> x, y, z = t
>>> print(x, y, z)
12345 54321 hello!
>>> t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assig
nment
Error in sys.excepthook:
Traceback (most recent call last):
  File "/usr/lib/python3/dist-packages/apport_python_
hook.py", line 63, in apport_excepthook
    from apport.fileutils import likely_packaged, get
_recent_crashes
  File "/usr/lib/python3/dist-packages/apport/__init_
_.py", line 5, in <module>
```

❖ 字典（Dictionary）

- ❖ 以关键字（key）为索引，关键字可以是任意不可变类型，通常用字符串或数值
- ❖ 可以看做无序的键值对（key-value pair）集合，在同一个字典之内键必须是互不相同的

valency — valency@patriot: ~ — ssh -o ServerAliveInterval=60 -Y patriot — 53×24

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'jack': 4098, 'sape': 4139, 'guido': 4127}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel
{'jack': 4098, 'guido': 4127}
>>> list(tel.keys())
['jack', 'guido']
>>> sorted(tel.keys())
['guido', 'jack']
>>> 'guido' in tel
True
>>> dict([('sape', 4139), ('guido', 4127)])
{'sape': 4139, 'guido': 4127}
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>>
>>>
>>>
```

基本数据类型

- 集合（Set）
- 集合是一个无序不重复元素的列表
- 基本功能包括关系测试和消除重复元素
- 集合对象还支持 union（联合）、intersection（交），difference（差）和 symmetric difference（对称差集）等数学运算
- 大括号 {} 或 set( ) 函数可以用来创建集合
- 想要创建空集合，必须使用 set( )

valency — valency@patriot: ~ — ssh -o ServerAliveInterval=60 -Y patriot — 53×24

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'or
ange', 'banana'}
>>> basket
{'banana', 'pear', 'apple', 'orange'}
>>> 'orange' in basket
True
>>> set('abracadabra')
{'b', 'c', 'a', 'r', 'd'}
>>> {x for x in 'abracadabra' if x not in 'abc'}
{'d', 'r'}
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
```

流程控制

- 流程控制
- 条件判断
- 循环

❖ 条件判断

valency — valency@patriot: ~ — ssh -o ServerAliveInterval=60 -Y patriot — 125×16

```
>>> x = 3
>>> if x < 0:
...       x = 0
...       print('Negative changed to zero')
... elif x == 0:
...       print('Zero')
... elif x == 1:
...       print('Single')
... else:
...       print('More')
...
More
>>> y = x + 1 if x == 2 else x - 1
>>> y
2
>>>
```

❖ 条件判断

❖ 比较操作可以通过逻辑操作符 and 和 or 组合，比较的结果可以用 not 来取反义

❖ 比较操作可以传递：a < b == c

❖ 比较操作从左往右计算，一旦结果可以确定就会立刻停止：3 > 1 or int('a')

❖ 可以把比较或其它逻辑表达式的返回值赋给一个变量：a = '' or 'b' or 'c'

❖ 列表、元组、字符串也可以比较：

```
(1, 2, 3)              < (1, 2, 4)
[1, 2, 3]              < [1, 2, 4]
'ABC' < 'C' < 'Pascal' < 'Python'
(1, 2, 3, 4)           < (1, 2, 4)
(1, 2)                 < (1, 2, -1)
(1, 2, 3)             == (1.0, 2.0, 3.0)
(1, 2, ('aa', 'ab'))   < (1, 2, ('abc', 'a'), 4)
```

流程控制

- 循环
- 有 for 和 while 两种方法
- 可以使用 break 终止循环
- 可以使用 continue 跳过后续代码
- 可以使用 pass 执行空循环
- 可以使用列表循环赋值

valency — valency@patriot: ~ — ssh -o ServerAliveInterval=60 -Y patriot — 53×24

```
>>> for i in range(3):
...     print(i)
...
0
1
2
>>> for i in [0, 1, 2]:
...     print(i)
...
0
1
2
>>> a = [i * 3 for i in range(6)]
>>> a
[0, 3, 6, 9, 12, 15]
>>> i = 0
>>> while i < 3:
...     print(i)
...     i += 1
...
0
1
2
>>>
```

❖ 循环

❖ 在字典中循环时，关键字和对应的值可以使用 items() 方法同时解读出来

❖ 在列表中循环时，索引位置和对应值可以使用 enumerate() 函数同时得到

❖ 同时循环两个或更多的列表，可以使用 zip() 整体打包

```
valency — valency@patriot: ~ — ssh -o ServerAliveInterval=60 -Y patriot — 53×24
>>> knights = {'gallahad': 'pure', 'robin': 'brave'}
>>> for k, v in knights.items():
...     print(k, v)
...
gallahad pure
robin brave
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print(i, v)
...
0 tic
1 tac
2 toe
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print(q, '=>', a)
...
name => lancelot
quest => the holy grail
favorite color => blue
>>>
>>>
>>>
>>>
```

- 循环
- 需要逆向循环列表的话，先正向定位列表，然后调用 reversed() 函数
- 要按排序后的顺序循环列表的话，使用 sorted() 函数，它不改动原列表，而是生成一个新的已排序的列表
- 若要在循环内部修改正在遍历的列表（例如复制某些元素），建议首先制作源列表的副本
- 在列表上循环不会隐式地创建副本

```
valency — valency@patriot: ~ — ssh -o ServerAliveInterval=60 -Y patriot — 53×24
>>> for i in reversed(range(1, 10, 2)):
...     print(i)
...
9
7
5
3
1
>>> basket = ['apple', 'pear', 'orange', 'banana']
>>> for f in sorted(basket):
...     print(f)
...
apple
banana
orange
pear
>>> words = ['cat', 'window', 'defenestrate']
>>> for w in words[:]:
...     if len(w) > 6:
...         words.insert(0, w)
...
>>> words
['defenestrate', 'cat', 'window', 'defenestrate']
>>>
```

目录

Python 简单回顾

基本语法

函数、类、模块

错误处理、日志处理

标准库

函数、类、模块

函数

- 函数名指代的值（即函数体）有一个被 Python 解释器认定为用户自定义函数的类型
- 这个值可以赋予其他的名字（即变量名），然后它也可以被当做函数使用

- Python 只有函数，没有方法
- 没有 return 语句的函数会返回 None

```
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a + b
    print()
```

```
>>> f = fib
>>> f(100)
0 1 1 2 3 5 8 13 21 34 55 89
```

```
>>> print(fib(0))
None
```

函数

❖ 默认参数值

❖ Python 可以定义包含若干参数的函数

❖ 可以指定参数的类型（调用者可以不遵守）

❖ 可以指定参数的默认值

```
def ask(q: str, a='No'):
    print(q, '=>', a)
```

```
>>> ask('Do you love me?')
>>> ask('Are you sure?', 'Yes')
Do you love me? => No
Are you sure? => Yes
```

函数

- 默认参数值
- 重要警告：默认值只被赋值一次
- 当默认值是可变对象时会有所不同
- 例如：列表、字典或者大多数类的实例

```
def f(a, u=[]):
    u.append(a)
    return u
```

```
>>> print(f(1))
>>> print(f(2))
>>> print(f(3))
[1]
[1, 2]
[1, 2, 3]
```

```
def f(a, u=None):
    if u is None:
        u = []
    u.append(a)
    return u
```

❖ 关键字参数

❖ 函数可以通过关键字参数的形式来调用，形如 keyword = value

```
def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
    print("-- This parrot wouldn't", action, end=' ')
    print("if you put", voltage, "volts through it.")
    print("-- Lovely plumage, the", type)
    print("-- It's", state, "!")
```

```
parrot(1000)                                          # 1 positional argument
parrot(voltage=1000)                                  # 1 keyword argument
parrot(voltage=1000000, action='VOOOOOM')             # 2 keyword arguments
parrot(action='VOOOOOM', voltage=1000000)             # 2 keyword arguments
parrot('a million', 'bereft of life', 'jump')         # 3 positional arguments
parrot('a thousand', state='pushing up the daisies')  # 1 positional, 1 keyword
```

❖ 关键字参数

```
def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
    print("-- This parrot wouldn't", action, end=' ')
    print("if you put", voltage, "volts through it.")
    print("-- Lovely plumage, the", type)
    print("-- It's", state, "!")
```

❖ 以下几种调用是无效的：

```
parrot()                     # required argument missing
parrot(voltage=5.0, 'dead')  # non-keyword argument after a keyword argument
parrot(110, voltage=220)     # duplicate value for the same argument
parrot(actor='John Cleese')  # unknown keyword argument
```

- 匿名函数
- 源于 Lisp
- 通过 lambda 关键字创建短小的匿名函数
- Lambda 只能有一个单独的表达式

```
def make_incrementor(n):
    return lambda x: x + n
```

```
>>> f = make_incrementor(42)
>>> f(0)
42
>>> f(1)
43
```

```
>>> pairs = [(1, 'one'), (2,
'two'), (3, 'three'), (4, 'four')]
>>> pairs.sort(key=lambda pair:
pair[1]) >>> pairs
[(4, 'four'), (1, 'one'), (3,
'three'), (2, 'two')]
```

- 函数文档与注释
- 使用三个引号的代码块注释函数文档
- 使用井号（#）为代码注释

- PyCharm 中：
- 自动整理代码：Ctrl + Alt + L
- 自动整理引用：Ctrl + Alt + O
- 在函数名下方输入三个引号然后按下回车，会自动生成参数表和返回值说明

- Pylint：https://www.pylint.org/

```python
def f(a, b=0):
    """
    Calculate the sum of a and b
    :param a: the first number
    :param b: the second number
    :return: the sum of a and b
    """
    # Calculate the sum of a and b
    c = a + b
    # Return the result
    return c
```

❖ 类

- ❖ Python混合了 C++ 和 Modula-3 类的机制
- ❖ Python 的类并没有在用户和定义之间设立绝对的屏障
- ❖ 而是依赖于用户不去“强行闯入定义”的优雅

- ❖ 用 C++ 术语来讲，Python 中：
- ❖ 所有的类成员（包括数据成员）都是公有（ public ）的
- ❖ 所有的成员函数都是虚（ virtual ）的

类

❖ 类的继承

- ❖ 派生类定义的执行过程和基类是一样的
- ❖ 如果在类中找不到请求调用的属性，就搜索基类
- ❖ 如果基类是由别的类派生而来，这个规则会递归的应用上去

类

```python
class A:
    def __init__(self, k=0):
        self._a = k

    def __str__(self):
        return str(self._a)

    @property
    def a(self):
        return self._a

    def add(self, b):
        self._a += b
        return self._a

if __name__ == '__main__':
    a = A(1)
    a.add(1)
    print(a.a, a)
```

```python
class B(A):
    FACT_VALUE = -1

    def __init__(self, k=0):
        super().__init__(k)
        self._c = 0

    @staticmethod
    def fact():
        return B.FACT_VALUE

if __name__ == '__main__':
    b = B()
    print(b.a, b, b.fact())
```

```
2 2
0 0 -1
```

类

- 作用域和命名空间
- 命名空间是从命名到对象的映射
- 作用域就是一个 Python 程序可以直接访问命名空间的正文区域

- Python 的一个特别之处在于：
- 如果没有使用 `global` 语法，其赋值操作总是在最里层的作用域
- 赋值不会复制数据，只是将命名绑定到对象
- 删除也是如此：`del x` 只是从局部作用域的命名空间中删除命名 x

类

```python
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
```

函数、类、模块

类

- Python 解释器可以直接从文件中引用变量、函数和类
- 被引用文件被称为模块
- 文件中的变量、函数和类可以导入到模块中
- 模块的模块名（做为一个字符串）可以由全局变量 __name__ 得到

函数、类、模块

类

fibo.py

```python
# Fibonacci numbers module

def fib(n):
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a + b
    print()

def fib2(n):
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a + b
    return result
```

test.py

```python
import fibo

fibo.fib(1000)
fibo.fib2(100)
fibo.__name__
```

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
987
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
'fibo'
```

目录

Python 简单回顾

基本语法

函数、类、模块

错误处理、日志处理

标准库

错误和异常

- 语法错误（Syntax Error）
- 错误是由箭头前面的标记引起的（或者至少是这么检测的）

```
>>> while True print('Hello world')
  File "<stdin>", line 1, in ?
    while True print('Hello world')
                   ^
SyntaxError: invalid syntax
```

❖ 异常（Exception）

- ❖ 即使一条语句或表达式在语法上是正确的，当试图执行它时也可能会引发错误
- ❖ 运行期（runtime）检测到的错误被称为异常
- ❖ 程序不会无条件的崩溃，可以被处理（handle）

```
>>> 10 * (1 / 0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: int division or modulo by zero

>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: Can't convert 'int' object to str implicitly
```

错误处理、日志处理

错误和异常

❖ 错误处理按如下方式工作:

1. 首先，执行 try 子句
2. 如果没有异常发生， except 子句在 try 语句执行完毕后就被忽略了
3. 如果在 try 子句执行过程中发生了异常，那么该子句其余的部分就会被忽略
4. 如果异常匹配于 except 关键字后面指定的异常类型，就执行对应的 except 子句
5. 如果异常在 except 子句中没有与之匹配的分支，它就会传递到上一级 try 语句中
6. 如果最终仍找不到对应的处理语句，它就成为一个未处理异常，终止程序运行

```
for arg in sys.argv[1:]:
    try:
        f = open(arg, 'r')
    except IOError:
        print('Cannot open: ', arg)
    except:
        print(sys.exc_info()[0])
    else:
        print(len(f.readlines()))
        f.close()
```

错误处理、日志处理

日志处理

- 生产环境的代码中，要尽量避免使用 print
- print 无法直接指定日志级别
- print 无法直接将不同级别的日志输出到不同的文件中

- Python 的日志模块：logging
- 默认情况下：
- logging 模块将日志打印到屏幕上（stdout）
- 默认日志级别为 WARNING（即只有日志级别高于 WARNING 的日志信息才会输出）

```
import logging


logging.debug('debug message')
logging.info('info message')
logging.warn('warn message')
logging.error('error message')
logging.critical('critical message')
```

```
WARNING:root:warn message
ERROR:root:error message
CRITICAL:root:critical message
```

输入和输出

- 通常，你想要对输出做更多的格式控制，而不是简单的打印使用空格分隔的值
- 有两种方法可以格式化你的输出：
- 第一种方法是由你自己处理整个字符串，通过使用字符串切割和连接操作可以创建任何你想要的输出形式
- 第二种方法是使用 str.format() 方法
- 开发中应尽量使用第二种方法

```
>>> s = 'Hello, world.'
>>> str(s)
'Hello, world.'
>>> repr(s)
"'Hello, world.'"
>>> str(1/7)
'0.14285714285714285'
>>> x = 10 * 3.25
>>> y = 200 * 200
>>> s = 'The value of x is ' + repr(x)
>>> print(s)
The value of x is 32.5
>>> hello = 'hello, world\n'
>>> hellos = repr(hello)
>>> print(hellos)
'hello, world\n'
>>> repr((x, y, ('spam', 'eggs')))
"(32.5, 40000, ('spam', 'eggs'))"
```

```
>>> for x in range(1, 11):
...     print(repr(x).rjust(2), repr(x*x).rjust(3), end=' ')
...     print(repr(x*x*x).rjust(4))

>>> for x in range(1, 11):
...     print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))



 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

- 以上是一个 str.rjust() 方法的演示，它把字符串输出到一列，并通过向左侧填充空格来使其右对齐
- 类似的方法还有 str.ljust() 和 str.center()
- 还有另一个方法，str.zfill() 它用于向数值的字符串表达左侧填充 0（该函数可以正确理解正负号）
- 大括号和其中的字符会被替换成传入 str.format() 的参数。大括号中的数值指明使用传入 str.format() 方法的对象中的哪一个

```
>>> '12'.zfill(5)
'00012'
>>> '-3.14'.zfill(7)
'-003.14'
>>> '3.14159265359'.zfill(5)
'3.14159265359'

>>> print('{0} and {1}'.format('spam', 'eggs'))
spam and eggs
>>> print('{1} and {0}'.format('spam', 'eggs'))
eggs and spam
```

错误处理、日志处理

输入和输出

- 特别注意：使用 logging 的时候，不推荐用 str.format() 格式化字符串：
- logging.warning('Error: {}'.format('details'))

- 推荐使用参数表的形式格式化输出：
- logging.warning('Error: %s', 'details’)

- 因为这样可以避免多运行一个函数，效率更高

目录

Python 简单回顾

基本语法

函数、类、模块

错误处理、日志处理

标准库

- 函数 open( ) 返回文件对象
- 通常的用法需要两个参数：open(filename, mode)
- f = open('workfile', 'w')

- 第一个参数是一个含有文件名的字符串
- 第二个参数也是一个字符串，含有描述如何使用该文件的几个字符：
- 'r' 表示只是读取文件（默认）
- 'w' 表示只是写入文件（已经存在的同名文件将被删掉）
- 'a' 表示打开文件进行追加，写入到文件中的任何数据将自动添加到末尾
- 'r+' 表示打开文件同时进行读取和写入

文件读写

- Python 默认的字符串编码是 UTF-8，打开 GBK 编码的文件会乱码
- 文件模式可以追加一个 'b'，表示以二进制模式操作文件，例如：'rb'
- 二进制模式下，数据会以字节对象的形式读出和写入
- 在文本模式下，读取时默认会将平台有关的行结束符（UNIX 上是 \n , Windows上是 \r\n）转换为 \n
- 这种修改对文本文件没有问题，但会损坏 JPEG 或 EXE 这样的二进制文件中的数据
- 请正确选择读取文件的格式

❖ 文件对象方法

- ❖ f.read(size)
- ❖ 读取若干数量的数据并以字符串形式返回其内容
- ❖ size 是可选的数值，指定字符串长度
- ❖ 如果到了文件末尾，f.read() 会返回一个空字符串（''）

```
>>> f.read()
'This is the entire file.\n'
>>> f.read()
''
```

❖ 文件对象方法

- ❖ f.readline()
- ❖ 从文件中读取单独一行，字符串结尾会自动加上一个换行符（\n）
- ❖ 如果 f.readline() 返回一个空字符串，那就表示到达了文件末尾
- ❖ 如果是一个空行，就会描述为 '\n'，一个只包含换行符的字符串

```
>>> f.readline()
'This is the first line of the file.\n'
>>> f.readline()
'Second line of the file\n'
>>> f.readline()
''
```

❖ 文件对象方法

- ❖ f.write(string)
- ❖ 将 string 的内容写入文件
- ❖ 并返回写入字符的长度

```
>>> f.write('This is a test\n')
15
>>> value = ('the answer', 42)
>>> f.write(str(value))
18
>>> f.close()
```

标准库

文件读写

❖ 预定义清理行为

❖ 有些对象定义了标准的清理行为，无论对象操作是否成功，不再需要该对象的时候就会起作用

❖ 第一段代码的问题在于在代码执行完后没有立即关闭打开的文件，大量未关闭的文件会造成内存溢出

```python
for line in open("myfile.txt"):
    print(line)


with open("myfile.txt") as f:
    for line in f:
        print(line)
```

❖ with 语句可以确保对象能被及时清理

❖ 第二段代码执行后，文件 f 会被关闭

❖ 即使中途报错，文件 f 也会被关闭

❖ 不是所有的函数都具备这个功能，需要仔细确认

字符串正则匹配

- re 模块为高级字符串处理提供了正则表达式工具
- 对于复杂的匹配和处理，正则表达式提供了简洁、优化的解决方案

```
>>> 'tea for too'.replace('too', 'two')
'tea for two'

>>> import re
>>> re.findall(r'\bf[a-z]*', 'which foot or hand fell fastest')
['foot', 'fell', 'fastest']
>>> re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')
'cat in the hat’
```

数学

❖ math 模块为浮点运算提供了对底层 C 函数库的访问

```
>>> import math
>>> math.cos(math.pi / 4.0)
0.70710678118654757
>>> math.log(1024, 2)
10.0
```

❖ random 提供了生成随机数的工具

```
>>> import random
>>> random.choice(['apple', 'pear', 'banana'])
'apple'
>>> random.sample(range(100), 10)   # sampling without replacement
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]
>>> random.random()    # random float
0.17970987693706186
>>> random.randrange(6)    # random integer chosen from range(6)
4
```

标准库

互联网访问

- Python 标准库提供了多个用于访问互联网以及处理网络通信协议的模块
- 其中最简单的是用于处理从 URL 接收数据的 `urllib.request`

```
>>> from urllib.request import urlopen
>>> for line in urlopen('http://baidu.com'):
...     print(line)

b'<html>\n'
b'<meta http-equiv="refresh" content="0;url=http://www.baidu.com/">\n'
b'</html>\n'
```

日期和时间

- ❖ datetime 模块为日期和时间处理同时提供了简单和复杂的方法
- ❖ 支持日期和时间算法的同时，实现的重点放在更有效的处理和格式化输出
- ❖ 该模块还支持时区处理

```
>>> from datetime import date
>>> now = date.today()
>>> now
datetime.date(2003, 12, 2)
>>> now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d day of %B.")
'12-02-03. 02 Dec 2003 is a Tuesday on the 02 day of December.'

>>> birthday = date(1964, 7, 31)
>>> age = now - birthday
>>> age.days
14368
```

Why is subtracting these two times (in 1927) giving a strange result?

Ask Question

6318

1498

If I run the following program, which parses two date strings referencing times 1 second apart and compares them:

```
public static void main(String[] args) throws ParseException {
    SimpleDateFormat sf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    String str3 = "1927-12-31 23:54:07";
    String str4 = "1927-12-31 23:54:08";
    Date sDt3 = sf.parse(str3);
    Date sDt4 = sf.parse(str4);
    long ld3 = sDt3.getTime() /1000;
    long ld4 = sDt4.getTime() /1000;
    System.out.println(ld4-ld3);
}
```

The output is:

```
353
```

Why is `ld4-ld3` not `1` (as I would expect from the one-second difference in the times), but `353` ?

If I change the dates to times 1 second later:

```
String str3 = "1927-12-31 23:54:08";
String str4 = "1927-12-31 23:54:09";
```

Then `ld4-ld3` will be `1` .

asked 7 years, 7 months ago

viewed 586,518 times

active 24 days ago

UPCOMING EVENTS

2019 Community Moderator Election
ends Mar 19

FEATURED ON META

2019 Moderator Election Q&A - Questionnaire

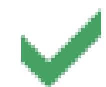
HOT META POSTS

30 2019 Community Moderator Election

16 Merge tags [google-play-services] and [google-services]

Jobs near you

10185



+1250

It's a time zone change on December 31st in Shanghai.

See this page for details of 1927 in Shanghai. Basically at midnight at the end of 1927, the clocks went back 5 minutes and 52 seconds. So "1927-12-31 23:54:08" actually happened twice, and it looks like Java is parsing it as the *later* possible instant for that local date/time - hence the difference.

Just another episode in the often weird and wonderful world of time zones.

EDIT: Stop press! History changes...

数据压缩

❖ 以下模块支持通用的数据打包和压缩格式：zlib，gzip，bz2，lzma，zipfile 以及 tarfile

```
>>> import zlib
>>> s = b'witch which has which witches wrist watch'
>>> len(s)
41
>>> t = zlib.compress(s)
>>> len(t)
37
>>> zlib.decompress(t)
b'witch which has which witches wrist watch'
>>> zlib.crc32(s)
226805979
```

性能度量

- 数据分析常常对解决同一问题的不同方法之间的性能差异很关注，Python 提供了一个度量工具，为这些问题提供了直接答案
- 例如，使用元组封装和拆封来交换元素看起来要比使用传统的方法要快一些，timeit 证明了这一点

```
>>> from timeit import Timer
>>> Timer('t=a; a=b; b=t', 'a=1; b=2').timeit()
0.57535828626024577
>>> Timer('a,b = b,a', 'a=1; b=2').timeit()
0.54962537085770791
```

十进制浮点数算法

❖ decimal 模块提供了一个 Decimal 数据类型用于更精确的浮点数计算

```
>>> from decimal import *
>>> round(Decimal('0.70') * Decimal('1.05'), 2)
Decimal('0.74')
>>> round(.70 * 1.05, 2)
0.73
>>> Decimal('1.00') % Decimal('.10')
Decimal('0.00')
>>> 1.00 % 0.10
0.09999999999999995
>>> sum([Decimal('0.1')]*10) == Decimal('1.0')
True
>>> sum([0.1]*10) == 1.0
False
>>> getcontext().prec = 36
>>> Decimal(1) / Decimal(7)
Decimal('0.142857142857142857142857142857142857')
```

❖ Python 不仅支持线程（Thread），还支持协程（Coroutine）

```
import threading, zipfile
class AsyncZip(threading.Thread):
    def __init__(self, infile, outfile):
        threading.Thread.__init__(self)
        self.infile = infile
        self.outfile = outfile
    def run(self):
        f = zipfile.ZipFile(self.outfile, 'w', zipfile.ZIP_DEFLATED)
        f.write(self.infile)
        f.close()
        print('Finished background zip of:', self.infile)
background = AsyncZip('mydata.txt', 'myarchive.zip')
background.start()
print('The main program continues to run in foreground.')
background.join()
print('Main program waited until background was done.')
```

Thanks!