

【Python/八股文系列】：100个Python的面试/笔试高频考点

原创

AlphaGu...
20:10:46 修改


阅读量
3w

★收藏
916

点赞数
120

版权

分类专栏：[# Python面试笔试八股文](#) 文章标签：[python](#) [面试](#) [开发语言](#)

 Python面试笔试八股文 专栏
19 订阅 1 篇文章 [订阅专栏](#)

Python的100个面试/笔试高频考点

本文主要整理了关于Python的面试/笔试的一些考点，可用于查漏补缺。

涉及到的一些 [Python进阶](#) 知识，可以查看专栏学习：《[Python进阶](#)》

1. 解释型和编译型语言的区别

- 编译型语言：把做好的源程序全部编译成**二进制的**可运行程序。然后，可直接运行这个程序。如：C，C++ ；
- 解释型语言：把做好的源程序翻译一句，然后执行一句，直至结束！如：Python。

注意：Java 有些特殊，java程序也需要编译，但是没有直接编译成为 **机器语言**，而是编译称为**字节码**，然后用**解释方式**执行字节码。

2. 简述下 Python 中的字符串、列表、元组和字典

- 字符串 (str)：字符串是用引号括起来的任意文本，是编程语言中最常用的数据类型。
- 列表 (list)：列表是**有序**的集合，可以向其中添加或删除元素。
- 元组 (tuple)：元组也是**有序**集合，元组中的数无法修改。即**元组是不可变的**。
- 字典 (dict)：字典是**无序**的集合，是由**键值对** (key-value) 组成的。
- 集合 (set)：是一组 key 的集合，每个元素都是**唯一**，**不重复且无序**的。

3. 简述上述数据类型的常用方法

字符串：

1. 切片： `'luobodazahui'[1:3]`
2. `format`： `"welcome to luobodazahui, dear {name}"format(name="baby")`
3. `join`：可以用来连接字符串，将字符串、元组、列表中的元素以指定的字符(分隔符)连接生成一个新的字符串。 `'-'.join(['luo', 'bo', 'da', 'za', 'hui'])`
4. `String.replace(old,new,count)`：将字符串中的 `old` 字符替换为 `New` 字符，`count` 为替换的个数 `'luobodazahui-haha'.replace('haha', 'good')`
5. `split`：切割字符串，得到一个列表

```
1 >>> mystr5 = 'luobo,dazahui good'
2
3 >>> print(mystr5.split()) # 默认以空格分割
4 ['luobo,dazahui', 'good']
5
6 >>> print(mystr5.split('h')) # 以h分割
7 ['luobo,daza', 'ui good']
8
9 >>> print(mystr5.split(',')) # 以逗号分割
10 ['luobo', 'dazahui good']
```

列表：

1. 切片，同字符串
2. `append` 和 `extend` 向列表中添加元素

```
1 >>> mylist1 = [1, 2]
2 >>> mylist2 = [3, 4]
3
```

```
3 >>> mylist3 = [1, 2]
4
5 >>> mylist1.append(mylist2)
6 >>> print(mylist1)
7 [1, 2, [3, 4]]
8
9 >>> mylist3.extend(mylist2)
10 >>> print(mylist3)
11 [1, 2, 3, 4]
```

删除元素 `del`：

1. 根据下标进行删除 `pop`：删除最后一个元素
2. `remove`：根据元素的值进行删除

```
1 >>> mylist4 = ['a', 'b', 'c', 'd']
2
3 >>> del mylist4[0]
4 >>> print(mylist4)
5 ['b', 'c', 'd']
6
7 >>> mylist4.pop()
8 >>> print(mylist4)
9 ['b', 'c']
10
11 >>> mylist4.remove('c')
12 >>> print(mylist4)
13 ['b']
```

4. 元素排序 `sort`：是将 `list` 按特定顺序重新排列，默认为由小到大，参数 `reverse=True` 可改为倒序，由大到小。

```
1 >>> mylist5 = [1, 5, 2, 3, 4]
2 >>> mylist5.sort()
3 >>> print(mylist5)
4 [1, 2, 3, 4, 5]
5 >>> mylist5.reverse()
6 >>> print(mylist5)
7 [5, 4, 3, 2, 1]
```

5. `reverse`：是将 `list` 逆置。

字典：

1. 清空字典 `dict.clear()`

```
1 >>> dict1 = {'key1':1, 'key2':2}
2 >>> dict1.clear()
3 >>> dict1
4 {}
```

2. 指定删除：使用 `pop` 方法来指定删除字典中的某一项（随机的）。

```
1 >>> dict1 = {'key1':1, 'key2':2}
2 >>> d1 = dict1.pop('key1')
3 >>> dict1
4 {'key2': 2}
5 >>> d1
6 1
```

3. 遍历字典

```
1 >>> dict2 = {'key1':1, 'key2':2}
2 >>> mykey = [key for key in dict2] # ['key1', 'key2']
3 >>> mykey
4 ['key1', 'key2']
5 >>> myvalue = [value for value in dict2.values()]
6 >>> myvalue
7 [1, 2]
8 >>> key_value = [(k, v) for k, v in dict2.items()]
9 >>> key_value
10 [('key1', 1), ('key2', 2)]
```

4. `fromkeys` 用于创建一个新字典，以序列中元素做字典的键，`value` 为字典所有键对应的初始值。

```
1 >>> keys = ['zhangfei', 'guanyu', 'liubei', 'zhaoyun']
2 >>> dict.fromkeys(keys, 0)
3 {'zhangfei': 0, 'guanyu': 0, 'liubei': 0, 'zhaoyun': 0}
```

4. 简述 Python 中的字符串编码

计算机在最初的设计中，采用了8个比特（bit）作为一个字节（byte）的方式。一个字节能表示的最大的整数就是255，如果要表示更大的整数，就必须用更多的字节。最早，计算机只有 ASCII 编码，即只包含大小写英文字母、数字和一些符号，这些对于其他语言，如中文，日文显然是不够用的。后来又发明了Unicode，Unicode把所有语言都统一到一套编码里，这样就不会再有乱码问题了。当需要保存硬盘或者需要传输的时候，就转换为UTF-8编码。**UTF-8 是隶属于 Unicode 的可变长的编码方式。**

在 Python 中，以 Unicode 方式编码的字符串，可以使用 `encode()` 方法来编码成指定的 `bytes`，也可以通过 `decode()` 方法来把 `bytes` 编码成字符串。

```
1 >>> "你好".encode('utf-8')
2 b'\xe4\xbd\xa0\xe5\xa5\xbd'
3 >>> b'\xe4\xb8\xad\xe6\x96\x87'.decode('utf-8')
4 "你好"
```

5. 一行代码实现数值交换

```
1 >>> a, b = 1, 2
2 >>> a, b = b, a
3 >>> print(a, b)
```

6. is 和 == 的区别

`==` 是比较操作符，只是判断对象的值（value）是否一致，而 `is` 则判断的是对象之间的身份（内存地址）是否一致。对象的身份，可以通过 `id()` 方法来查看。

```
1 >>> c = d = [1, 2]
2 >>> e = [1, 2]
3
4 >>> print(c is d)
5 True
6
7 >>> print(c == d)
8 True
9
10 >>> print(c is e)
11 False
12
13
14
```

```
>>> print(c == e)
```

只有 `id` 一致时，`is` 比较才会返回 `True`，而当 `value` 一致时，`==` 比较就会返回 `True`。

7. Python 函数中的参数类型

位置参数，默认参数，可变参数，关键字参数。

8. *arg 和 **kwargs 作用

允许我们在调用函数的时候传入多个实参

```
1 >>> def test(*arg, **kwargs):
2     ...     if arg:
3     ...         print("arg:", arg)
4     ...     if kwargs:
5     ...         print("kwargs:", kwargs)
6     ...
7 >>> test('ni', 'hao', key='world')
8 arg: ('ni', 'hao')
9 kwargs: {'key': 'world'}
```

可以看出，`*arg` 会把位置参数转化为 `tuple`，`**kwargs` 会把关键字参数转化为 `dict`。

9. 获取当前时间

```
1 >>> import time
2 >>> import datetime
3 >>> print(datetime.datetime.now())
4 2022-09-12 19:51:24.314335
5 >>> print(time.strftime('%Y-%m-%d %H:%M:%S'))
6 2022-09-12 19:51:24
```

10. PEP8 规范

简单列举10条：

1. 尽量以单独使用小写字母' l ', 大写字母' O ', 以及大写字母' I '等容易混淆的字母。
2. **函数命名使用全部小写的方式**，可以使用下划线。
3. **常量命名使用全部大写的方式**，可以使用下划线。
4. 使用 `has` 或 `is` 前缀命名布尔元素，如： `is_connect = True`； `has_member = False`。
5. 不要在行尾加分号，也不要加分号将两条命令放在同一行。
6. 不要使用反斜杠连接行。
7. **方法定义之间空1行，顶级定义之间空两行。**
8. 如果一个类不继承自其它类，就显式的从 `object` 继承。
9. 内部使用的类、方法或变量前，需加前缀 `_` 表明此为内部使用的。
10. 要用断言来实现静态类型检测。

11. Python 的深浅拷贝 (❤❤)

1. 浅拷贝

```
1 >>> import copy
2 >>> list1 = [1, 2, 3, [1, 2]]
3 >>> list2 = copy.copy(list1)
4 >>> list2.append('a')
5 >>> list2[3].append('a')
6 >>> list1
7 [1, 2, 3, [1, 2, 'a']]
8 >>> list2
9 [1, 2, 3, [1, 2, 'a'], 'a']
```

浅拷贝只成功“独立”拷贝了列表的外层，而列表的内层列表，还是共享的。（划重点!!!）

2. 深拷贝

```
1 >>> import copy
2 >>> list1 = [1, 2, 3, [1, 2]]
```

```
3 >>> list3 = copy.deepcopy(list1)
4 >>> list3.append('a')
5 >>> list3[3].append('a')
6 >>> list1
7 [1, 2, 3, [1, 2]]
8 >>> list3
9 [1, 2, 3, [1, 2, 'a'], 'a']
```

深拷贝使得两个列表完全独立开来，每一个列表的操作，都不会影响到另一个。

12. `[lambda x:i*x for i in range(4)]` ♥♥

这是一道非常经典的题目了。

```
1 >>> def num():
2 ...     return [lambda x:i*x for i in range(4)]
3 ...
4 >>> [m(1) for m in num()]
5 [3, 3, 3, 3]
```

why?

详细看链接：[《Python面试题目：\[lambda x: x*i for i in range\(4\)\]》](#)

13. 打印九九乘法表

```
1 >>> for i in range(1, 10):
2 ...     for j in range(1, i + 1):
3 ...         print(f"{i}*{j}={i * j}", end=" ")
4 ...     print()
5 ...
6 1*1=1
7 2*1=2 2*2=4
8 3*1=3 3*2=6 3*3=9
9 4*1=4 4*2=8 4*3=12 4*4=16
10 5*1=5 5*2=10 5*3=15 5*4=20 5*5=25
11 6*1=6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36
12 7*1=7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49
13 8*1=8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64
14 9*1=9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```


知识点： `print` 函数默认是会换行的，其有一个默认参数 `end`。

14. `filter`、`map`、`reduce` 的作用

1. `filter` 函数用于**过滤序列**，它接收一个函数和一个序列，把函数作用在序列的每个元素上，然后根据返回值是 `True` 还是 `False` 决定保留还是丢弃该元素。

```
1 >>> mylist = list(range(10))
2 >>> list(filter(lambda x: x % 2 == 1, mylist))
3 [1, 3, 5, 7, 9]
```

2. `map` 函数传入一个函数和一个序列，并把函数作用到序列的每个元素上，返回一个可迭代对象。

```
1 >>> list(map(lambda x: x % 2, mylist))
2 [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
3 >>> list(map(lambda x: x * 2, mylist))
4 [2, 4, 6, 8, 10, 12, 14, 16, 18]
```

3. `reduce` 函数用于**递归计算**，同样需要传入一个函数和一个序列，并把函数和序列元素的计算结果与下一个元素进行计算。

`reduce()` 将一个数据集合（链表，元组等）中的**所有数据**进行下列操作：用传给 `reduce` 中的 `function` 函数（有两个参数）先对集合中的第 1、2 个元素进行操作，得到的结果再与第三个数据用 `function` 函数运算，最后得到一个结果。

```
1 >>> from functools import reduce
2 >>> reduce(lambda x, y: x + y, range(101))
3 5050
```

`reduce()` 的应用：

- 一行代码搞定计算所有元素的积：

```
1 reduce(lambda x, y: x * y, [1, 2, 3, 4])
```

- 计算列表中所有元素的最大值：

```
1 | reduce(lambda x, y: x if x > y else y, [1, 2, 3, 4])
```

- 有一个列表：[3, 5, 8, 1] 对应的是3581的每一个数字，要从这个列表计算出原来的数，可以这样做：

```
1 | reduce(lambda x, y: x * 10 + y, [3, 5, 8, 1])
```

可以看出，上面的三个函数与匿名函数相结合使用，可以写出强大简洁的代码。

15. 为什么不建议函数的默认参数传入可变对象

例如：

```
1 | >>> def test(L=[]):
2 | ...     L.append('test')
3 | ...     print(L)
4 | ...
5 | >>> test()
6 | ['test']
7 | >>> test()
8 | ['test', 'test']
```

默认参数是一个列表，是可变对象 `[]`，Python在函数定义的时候，默认参数 `L` 的值就被计算出来了，是 `[]`，每次调用函数，如果 `L` 的值变了，那么下次调用时，默认参数的值就已经不再是 `[]` 了。

16. 面向对象中 `__new__` 和 `__init__` 区别 (❤❤)

- `__new__` 是在实例创建之前被调用的，因为它的任务就是创建实例然后返回该实例对象，是个静态方法。
 - `__init__` 是当实例对象创建完成后被调用的，然后设置对象属性的一些初始值，通常用在初始化一个类实例的时候，是一个实例方法。
1. `__new__` 至少要有个参数 `cls`，代表当前类，此参数在实例化时由 Python 解释器自动识别。

2. `__new__` 必须要有返回值，返回实例化出来的实例，这点在自己实现 `__new__` 时要特别注意，可以 `return` 父类（通过 `super(当前类名, cls)`）`__new__` 出来的实例，或者是 `object` 的 `__new__` 出来的实例。
3. `__init__` 有一个参数 `self`，就是这个 `__new__` 返回的实例，`__init__` 在 `__new__` 的基础上可以完成一些其它初始化的动作，`__init__` 不需要返回值。
4. 如果 `__new__` 创建的是当前类的实例，会自动调用 `__init__` 函数，通过 `return` 语句里面调用的 `__new__` 函数的第一个参数是 `**cls**` 来保证是当前类实例，如果是其他类的类名，那么实际创建返回的就是其他类的实例，其实就不会调用当前类的 `__init__` 函数，也不会调用其他类的 `__init__` 函数。

17. 三元运算规则

看下面的例子：如果 `a > b` 成立 就输出 `a - b` 否则 `a + b`。

```
1 >>> a, b = 1, 2
2 >>> h = a - b if a > b else a + b
3 >>> h
4 3
```

18. 生成随机数

```
1 >>> import random
2 >>> random.random()
3 0.7571910055209727
4 >>> random.randint(1, 100)
5 23
6 >>> random.uniform(1, 5)
7 3.0640732831151687
```

用 `np.random` 也可以！

19. zip 函数用法

`zip()` 函数将可迭代的对象作为参数，将对象中对应的元素打包成一个元组，然后返回由这些元组组成的列表。

```
1 >>> list1 = ['zhangfei', 'guanyu', 'liubei', 'zhaoyun']
2 >>> list2 = [0, 3, 2, 4]
3 >>> list(zip(list1, list2))
4 [('zhangfei', 0), ('guanyu', 3), ('liubei', 2), ('zhaoyun', 4)]
```

20. range 和 xrange 的区别

只有Python2中才有 `xrange()` 和 `range()`，Python3中的 `range()` 其实就是Python2中 `xrange()`。

- `range([start,] stop[, step])`，根据 `start` 与 `stop` 指定的范围以及 `step` 设定的步长，生成一个序列；
- `xrange()` 生成一个生成器，可以很大的节约内存。

21. with 方法打开文件的作用

打开文件在进行读写的时候可能会出现一些异常状况，如果按照常规的 `f.open()` 写法，我们需要 `try`，`except`，`finally`，做异常判断，并且文件最终不管遇到什么情况，都要执行 `f.close()` 关闭文件，`with` 方法帮我们实现了 `finally` 中 `f.close()`。

```
1 with open("hello.txt", "a") as f:
2     f.write("hello world!")
```

22. 字符串转列表

```
1 >>> s = "1,2,3,4,5,6,7,8,9"
2 >>> s.split(",")
3 ['1', '2', '3', '4', '5', '6', '7', '8', '9']
```

23. 字符串转整数

```
1 >>> s = "1,2,3,4,5,6,7,8,9"
2 >>> list(map(lambda x: int(x), s.split(",")))
3 [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

24. 删除列表中的重复值

利用集合去重：

```
1 >>> mylist = [1, 2, 3, 4, 5, 5, 6, 7, 4, 3]
2 >>> list(set(mylist))
3 [1, 2, 3, 4, 5, 6, 7]
```

25. 字符串单词统计

统计字符串中字母个数：

```
1 >>> from collections import Counter
2 >>> mystr = 'sdfsfsfsdsd,were,hrhrgege.sdfwe!sfsdfs'
3 >>> Counter(mystr)
4 Counter({'s': 9,
5         'd': 5,
6         'f': 7,
7         ',': 2,
8         'w': 2,
9         'e': 5,
10        'r': 3,
11        'h': 2,
12        'g': 2,
13        '.': 1,
14        '!': 1})
```

统计字符串中单词个数

```
1 >>> mystr2 = "hello, Nice to meet you!"
2 >>> len(mystr2.split(" "))
3 5
```

26. 列表推导，求奇偶数

```
1 >>> [x for x in range(20) if x % 2 == 1]
2 [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

27. 一行代码展开列表

```
1 >>> list1 = [[1, 2], [3, 4], [5, 6]]
2 >>> [j for i in list1 for j in i]
3 [1, 2, 3, 4, 5, 6]
```

28. 实现二分法查找函数

二分查找算法也称折半查找，基本思想就是折半，对比大小后再折半查找，必须是有序序列才可以使用二分查找。

非递归算法：

```
1 def binary_search(data, item):
2     left = 0
3     right = len(data) - 1
4     while left <= right:
5         mid = (left + right) // 2
6         if data[mid] == item:
7             return True
8         elif data[mid] < item:
9             left = mid + 1
10        else:
11            right = mid - 1
12    return False
```

递归算法：

```
1 def binary_search(data, item):
2     n = len(data)
3     if n > 0:
4         mid = n // 2
```

```
5     if data[mid] == item:
6         return True
7     elif data[mid] > item:
8         return binary_search(data[:mid], item)
9     else:
10        return binary_search(data[mid + 1:], item)
11    return False
```

29. 合并两个元组到字典

如: ("zhangfei", "guanyu"), (66, 80) -> {'zhangfei': 66, 'guanyu': 80}。

```
1 >>> a = ("zhangfei", "guanyu")
2 >>> b = (66, 80)
3 >>> dict(zip(a, b))
4 {'zhangfei': 66, 'guanyu': 80}
```

30. 给出如下代码的输入，并简单解释

例子1:

```
1 >>> a = (1, 2, 3, [4, 5, 6, 7], 8)
2 >>> a[3] = 2
```

报错: TypeError: 'tuple' object does not support item assignment.

原因: **元组不能修改**! tuple 是不可变类型, 不能改变 tuple 里的元素。

例子2:

```
1 >>> a = (1, 2, 3, [4, 5, 6, 7], 8)
2 >>> a[3][2] = 2
3 >>> a
4 (1, 2, 3, [4, 5, 2, 7], 8)
```

list 是可变类型, 改变其元素是允许的。

31. 字典和 json 转换

```
1 dict1 = {'zhangfei':1, "liubei":2, "guanyu": 4, "zhaoyun":3}
2 myjson = json.dumps(dict1) # 字典转JSON
3 mydict = json.loads(myjson) # JSON转字典
```

32. 列表推导式、字典推导式和生成器

列表推导式：返回一个列表。

```
1 >>> td_list = [i for i in range(10)]
2 >>> td_list
3 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
4 >>> type(td_list)
5 list
```

生成器：

```
1 >>> ge_list = (i for i in range(20))
2 >>> ge_list
3 <generator object <genexpr> at 0x000001C3C127AAC8>
4 >>> type(ge_list)
5 generator
```

字典推导式：

```
1 >>> dic = {k: 2 for k in ["a", "b", "c", "d"]}
2 >>> dic
3 {'a': 2, 'b': 2, 'c': 2, 'd': 2}
4 >>> type(dic)
5 dict
```

33. 简述 read、readline、readlines 的区别

- `read()` 读取整个文件
- `readline()` 读取下一行,使用生成器方法
- `readlines()` 读取整个文件到一个迭代器以供我们遍历

34. 打乱一个列表

```
1 >>> import random
2 >>> list = list(range(1, 10))
3 >>> random.shuffle(list)
4 >>> list
5 [3, 9, 1, 4, 6, 2, 8, 7, 5]
```

35. 反转字符串

```
1 >>> 'luobodazahui'[::-1]
2 'iuhazadoboul'
```

36. 单下划线和双下划线的作用

- `__foo__`：一种约定，Python内部的名字，用来区别其他用户自定义的命名，以防冲突，就是例如 `__init__()`，`__del__()`，`__call__()` 些特殊方法。
- `_foo`：一种约定，用来指定变量**私有**。不能用 `from module import *` 导入，其他方面和公有变量一样访问。
- `__foo`：这个有真正的意义：解析器用 `__classname__foo` 来代替这个名字，以区别和其他类相同的命名，它无法直接像公有成员一样随便访问，通过 `对象名._类名__xxx` 这样的方式可以访问。

37. 新式类和旧式类

- 在 python 里凡是继承了 `object` 的类，都是新式类
- Python3 里只有新式类
- Python2 里面继承 `object` 的是新式类，没有写父类的是经典类
- 经典类目前在 Python 里基本没有应用

38. Python 面向对象中的继承有什么特点

- 同时支持单继承与多继承，当只有一个父类时为单继承，当存在多个父类时为多继承；
- 子类会继承父类所有的属性和方法，子类也可以覆盖父类同名的变量和方法；
- 在继承中基类的构造（`__init__()`）方法不会被自动调用，它需要在其派生类的构造中专门调用；
- 在调用基类的方法时，需要加上基类的类名前缀，且需要带上 `self` 参数变量。区别于在类中调用普通函数时并不需要带上 `self` 参数。

39. super 函数的作用

`super()` 函数是用于调用父类(超类)的一个方法

```
1 class A():
2     def funcA(self):
3         print("this is func A")
4
5 class B(A):
6     def funcA_in_B(self):
7         super(B, self).funcA()
8
9     def funcC(self):
10        print("this is func C")
11
12 >>> ins = B()
13 >>> ins.funcA_in_B()
14 this is func A
15 >>> ins.funcC()
```

40. 类中的各种函数

主要分为实例方法、类方法和静态方法。

1. 实例方法

定义：第一个参数必须是**实例对象**，该参数名一般约定为“`self`”，通过它来传递实例的属性和方法（也可以传类的属性和方法）。

调用：只能由实例对象调用。

2. 类方法

定义：使用装饰器 `@classmethod`。第一个参数必须是当前类对象，该参数名一般约定为“`cls`”，通过它来传递类的属性和方法（不能传实例的属性和方法）。

调用：实例对象和类对象都可以调用。

3. 静态方法

定义：使用装饰器 `@staticmethod`。参数随意，没有“`self`”和“`cls`”参数，但是方法体中不能使用类或实例的任何属性和方法。

调用：实例对象和类对象都可以调用。

静态方法是类中的函数，不需要实例。静态方法主要是用来**存放逻辑性**的代码，主要是一些逻辑属于类，但是和类本身没有交互。即在静态方法中，不会涉及到类中的方法和属性的操作。可以理解为**将静态方法存在此类的名称空间中**。

类方法是**将类本身作为对象进行操作**的方法。他和静态方法的区别在于：不管这个方式是从实例调用还是从类调用，它都用第一个参数把类传递过来。

41. 如何判断是函数还是方法

- 与类和实例无绑定关系的 `function` 都属于函数（function）
- 与类和实例有绑定关系的 `function` 都属于方法（method）

普通函数：

```
1 def func1():
2     pass
3 >>> print(func1)
4 <function func1 at 0x000001C3C12A4438>
```

类中的函数：

```
1 class People(object):
2     def func2(self):
3         pass
4
```

```
5     @staticmethod
6     def func3():
7         pass
8
9     @classmethod
10    def func4(cls):
11        pass
12
13    >>> people = People()
14
15    >>> print(people.func2)
```

42. `isinstance` 的作用以及与 `type()` 的区别 (❤❤)

`isinstance()` 函数来判断一个对象是否是一个已知的类型，类似 `type()`。

区别：

- `type()` 不会认为子类是一种父类类型，不考虑继承关系；
- `isinstance()` 会认为子类是一种父类类型，考虑继承关系。

```
1 class A(object):
2     pass
3
4 class B(A):
5     pass
6
7 >>> a = A()
8 >>> b = B()
9
10 >>> print(isinstance(a, A))
11 True
12 >>> print(type(a) == A)
13 True
14
15 >>> print(isinstance(b, A))
```

43. 单例模式与工厂模式

- 单例模式：主要目的是确保某一个类只有一个实例存在；

- 工厂模式：包涵一个超类，这个超类提供一个抽象化的接口来创建一个特定类型的对象，而不是决定哪个对象可以被创建。

单例模式：

这种模式涉及到一个单一的类，该类负责创建自己的对象，同时确保只有单个对象被创建。这个类提供了一种访问其唯一的对象的方式，可以直接访问，不需要实例化该类的对象。

意图：**保证一个类仅有一个实例**，并提供一个访问它的全局访问点。

主要解决：**一个全局使用的类频繁地创建与销毁。**

举例：操作一个**文件**，应该用一个唯一的实例去操作。

工厂模式：

工厂类模式设计的核心是：**让“生产”和“产品”解耦。**

工厂模式的主要问题：将原来分布在各个地方的对象创建过程单独抽离出来，交给工厂类负责创建。其他地方想要使用对象直接找工厂（即调用工厂的方法）获取对象。

优点：

- 一个调用者想创建一个对象，只要知道其名称就可以了。
- 扩展性高，如果想增加一个产品，只要扩展一个工厂类就可以。
- 屏蔽产品的具体实现，调用者只关心产品的接口。

44. 查看目录下的所有文件

```
1 >>> import os
2 >>> print(os.listdir('.'))
```

45. 计算由1到5组成的互不重复的三位数

```
1 for i in range(1, 6):
2     for j in range(1, 6):
3         for k in range(1, 6):
4             if (i != j) and i != k and j != k:
5                 print(f"{i}{j}{k}")
```

46. 去除字符串首尾空格

```
1 >>> "    hello world    ".strip()
2 'hello world'
```

47. 去除字符串中间的空格

方法一：

```
1 >>> "hello you are good".replace(" ", "")
2 'helloyouaregood'
```

方法二：

```
1 >>> "".join("hello you are good".split(" "))
2 'helloyouaregood'
```

48. 字符串格式化方式

方法一：使用 `%` 操作符

```
1 print("This is for %s" % "Python")
2 print("This is for %s, and %s" %("Python", "You"))
```

方法二： `str.format` （在 Python3 中，引入了这个新的字符串格式化方法）

```
1 print("This is my {}".format("chat"))
2 print("This is {name}, hope you can {do}".format(name="zhouluob", do=''))
```

方法三： `f-strings` （在 Python3-6 中，引入了这个新的字符串格式化方法）

```
1 name = "luobodazahui"
2 print(f"hello {name}")
```

一个复杂些的例子：

```
1 def mytest(name, age):
2     return f"hello {name}, you are {age} years old!"
3
4 >>> people = mytest("luobo", 20)
5 >>> print(people)
6 hello luobo, you are 20 years old!
```

49. 将"hello world"转换为首字母大写"Hello World"

`title()` 函数：

```
1 >>> str1 = "hello world"
2 >>> str1.title()
3 'Hello World'
```

不使用 `title()` 函数

```
1 >>> str1 = "hello world"
2 >>> " ".join(list(map(lambda x: x.capitalize(), str1.split(" "))))
3 'Hello World'
4
5 >>> " ".join(list(map(lambda x: x[0].upper() + x[1:], str1.split(" "))))
6 'Hello World'
```

50. 一行代码转换列表中的整数为字符串

如: [1, 2, 3] -> ["1", "2", "3"]

```
1 >>> list1 = [1, 2, 3]
2 >>> list(map(lambda x: str(x), list1))
3 ['1', '2', '3']
```

51. Python 中的反射 (❤️❤️)

反射就是通过字符串的形式，导入模块；通过字符串的形式，去模块寻找指定函数，并执行。利用字符串的形式去对象（模块）中操作（查找/获取/删除/添加）成员，一种

基于字符串的事件驱动!

简单理解就是用来判断某个字符串是什么，是变量还是方法。

例子：先定义一个类：

```
1 class NewClass(object):
2     def __init__(self, name, male):
3         self.name = name
4         self.male = male
5
6     def myname(self):
7         print(f'My name is {self.name}')
8
9     def mymale(self):
10        print(f'I am a {self.male}')
```

```
1 >>> people = NewClass('luobo', 'boy')
2 >>> print(hasattr(people, 'name'))
3 True
4 >>> print(getattr(people, 'name'))
5 luobo
6 >>> setattr(people, 'male', 'girl')
7 >>> print(getattr(people, 'male'))
8 girl
```

`getattr`，`hasattr`，`setattr`，`delattr` 对模块的修改都在内存中进行，并不会影响文件中真实内容。

详情请看博客：[《Python进阶系列》十八：详解Python中的反射——通过字符串的形式操作对象](#)

52. metaclass 元类 (❤️❤️)

类与实例：首先定义类以后，就可以根据这个类创建出实例，所以：先定义类，然后创建实例；

类与元类：先定义元类，根据 metaclass 创建出类，所以：先定义 metaclass，然后创建类。

```
1 class MyMetaclass(type):
2     def __new__(cls, class_name, class_parents, class_attr):
3         class_attr['print'] = "this is my metaclass's subclass %s" %c
4         return type.__new__(cls, class_name, class_parents, class_attr)
5
6
```



```
0         class MyNewclass(object, metaclass=MyMetaclass):
7             pass
8
9         >>> myinstance = MyNewclass()
10        >>> myinstance.print
11        "this is my metaclass's subclass MyNewclass"
```

详细内容见博客：[《Python进阶系列》十五：详解Python中的元类（metaclass）](#)

53. sort 和 sorted 的区别

`sort()` 是可变对象列表（list）的方法，无参数，无返回值。`sort()` 会改变可变对象。

```
1 >>> list1 = [2, 1, 3]
2 >>> print(list1.sort())
3 None
4 >>> list1
5 [1, 2, 3]
6 >>> dict1.sort()
7 Traceback (most recent call last):
8   File "<stdin>", line 1, in <module>
9   AttributeError: 'dict' object has no attribute 'sort'
```

`sorted()` 是产生一个新的对象。`sorted(L)` 返回一个排序后的 `L`，不改变原始的 `L`，`sorted()` 适用于任何可迭代容器。

```
1 >>> dict1 = {'test1':1, 'test2':2}
2 >>> list1 = [2, 1, 3]
3 >>> print(sorted(dict1))
4 ['test1', 'test2']
5 >>> print(sorted(list1))
6 [1, 2, 3]
```

54. Python 中的 GIL

GIL 是 Python 的**全局解释器锁**，同一进程中假如有多个线程运行，一个线程在运行 Python 程序的时候会占用 Python 解释器（加了一把锁即 GIL），使该进程内的其他线程无法运行，等该线程运行完后其他线程才能运行。如果线程运行过程中遇到耗时

操作，则解释器锁解开，使其他线程运行。所以在多线程中，线程的运行仍是有先后顺序的，并不是同时进行。

55. 产生8位随机密码 (❤️❤️)

string 模块用法：

1、将大写的ASCII字符列表和数字组合起来

```
1 >>> string.ascii_uppercase + string.digits
2 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
```

2、string.printable 输出被视为可打印符号的 ASCII 字符组成的字符串。

```
1 >>> string.printable
2 '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!\"#$%&'
```

string.printable[:-7] 去除被视为空白符号的 ASCII 字符组成的字符串。

使用python的random模块，使用其中的 choice 方法，从给定的字符序列中随机选择字符组合。

```
1 >>> import random
2 >>> import string
3 >>> "".join(random.choice(string.printable[:-7]) for i in range(8))
```

这就产生了8位随机密码。

56. 输出原始字符

python提供了输出原始字符串“r”的方法。

```
1 >>> print('hello\nworld')
2 hello
3 world
4 >>> print(b'hello\nworld')
5 b'hello\nworld'
6 >>> print(r'hello\nworld')
7 hello\nworld
```

57. 简述 `any()` 和 `all()` 方法

`all`：如果存在 `0`，`Null`，`False` 返回 `False`，否则返回 `True`；

`any`：如果都是 `0`，`None`，`False` 时，返回 `False`。

例子：

```
1 >>> all([1, 2, 3, 0])
2 False
3 >>> all([1, 2, 3])
4 True
5 >>> any([1, 2, 3])
6 True
7 >>> any([0, None, False])
8 False
```

58. 反转整数

首先判断是否是整数，再判断是否是一位数字，最后再判断是不是负数。

```
1 def reverse_int(x):
2     if not isinstance(x, int):
3         return False
4     if -10 < x < 10:
5         return x
6     tmp = str(x)
7     if tmp[0] != '-':
8         tmp = tmp[::-1]
9         return int(tmp)
10    else:
11        tmp = tmp[1:][::-1]
12        x = int(tmp)
13        return -x
```

```
1 >>> reverse_int(-23837)
2 -73832
```

59. 函数式编程 (❤️❤️)

函数式编程是一种抽象程度很高的编程范式，**纯粹的函数式编程语言编写的函数没有变量**，因此，任意一个函数，只要输入是确定的，输出就是确定的，这种纯函数称之为没有副作用。而允许使用变量的程序设计语言，由于函数内部的变量状态不确定，同样的输入，可能得到不同的输出，因此，这种函数是有副作用的。由于 Python 允许使用变量，因此，**Python 不是纯函数式编程语言**。

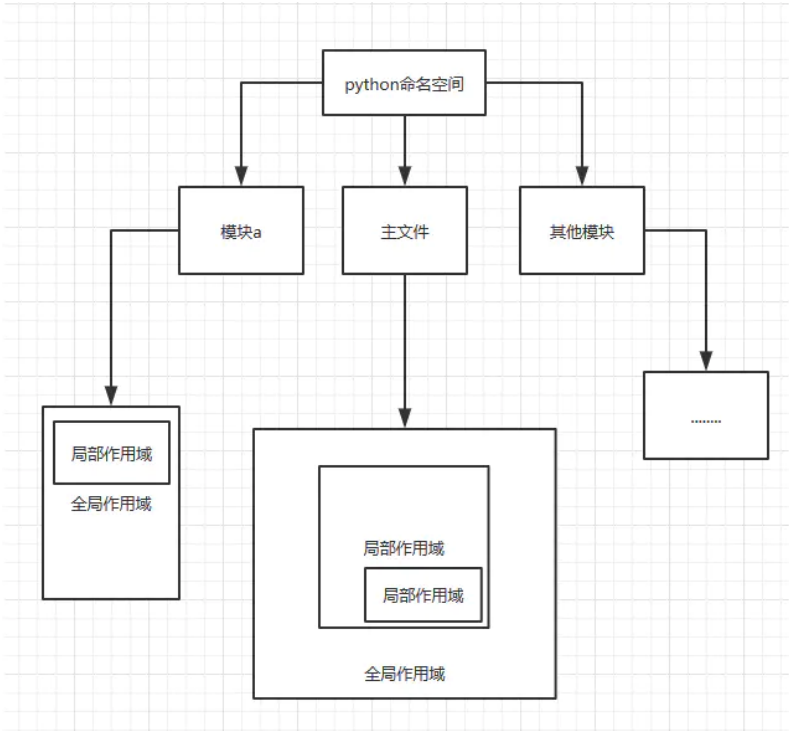
函数式编程的一个特点就是，**允许把函数本身作为参数传入另一个函数，还允许返回一个函数**！

函数作为返回值例子：

```
1  def sum(*args):
2      def inner_sum():
3          tmp = 0
4          for i in args:
5              tmp += i
6          return tmp
7      return inner_sum
8
9  >>> mysum = sum(2, 4, 6)
10 >>> print(type(mysum))
11 <class 'function'>
12 >>> mysum()
13 12
```

60. 简述闭包 (❤️❤️)

如果在一个内部函数里，对在外部的作用域（但不是在全局作用域）的变量进行引用，那么内部函数就被认为是闭包（closure）。



闭包特点：

- 必须有一个内嵌函数；
- 内嵌函数必须引用外部函数中的变量；
- 外部函数的返回值必须是内嵌函数。

详细看链接： [《Python面试题目： \[lambda x: x*i for i in range\(4\)\]》](#)

61. 简述装饰器 (❤❤)

装饰器是一种特殊的闭包，就是在闭包的基础上传递了一个函数，然后覆盖原来函数的执行入口，以后调用这个函数的时候，就可以额外实现一些功能了。

一个打印 log 的例子：

```
1 | import time
2 | def log(func):
3 |     def inner_log(*args, **kw):
```

```
4         print("Call: {}".format(func.__name__))
5         return func(*args, **kw)
6     return inner_log
7
8
9 @log
10 def timer():
11     print(time.time())
12
13 timer()
14 # Call: timer
15 # 1560171403.5128365
```

本质上，decorator就是一个返回函数的高阶函数。

详细看链接：[《Python进阶系列》八：装饰器的用法](#)

62. 协程的优点

协程的优点：

1. 最大优势就是协程极高的**执行效率**。因为子程序切换不是线程切换，而是由程序自身控制，因此，没有线程切换的开销，和多线程比，线程数量越多，协程的性能优势就越明显。
2. **不需要多线程的锁机制**，因为只有一个线程，也不存在同时写变量冲突，在协程中控制共享资源不加锁，只需要判断状态就好了，所以执行效率比多线程高很多。

因为协程是一个线程执行，那怎么利用多核CPU呢？最简单的方法是**多进程+协程**，既充分利用多核，又充分发挥协程的高效率，可获得极高的性能。

其他一些重要的点：

- 协程并没有增加线程数量，只是在线程的基础之上通过**分时复用**的方式运行多个协程，而且协程的切换在用户态完成，切换的代价比线程从用户态到内核态的代价小很多。
- 因此在协程调用阻塞IO操作的时候，操作系统会让线程进入阻塞状态，当前的协程和其它绑定在该线程之上的协程都会陷入阻塞而得不到调度，这往往是不能接受的。
- 因此在协程中不能调用导致线程阻塞的操作。也就是说，协程只有和异步IO结合起来，才能发挥最大的威力。

- 协程对计算密集型的任务没有太大的好处，计算密集型的任务本身不需要大量的线程切换，因为协程主要解决以往线程或者进程上下文切换的开销问题，所以协程主要对那些I/O密集型应用更好。

协程只有和异步IO结合起来才能发挥出最大的威力。

63. 实现斐波那契数列

斐波那契数列：又称黄金分割数列，指的是这样一个数列：1、1、2、3、5、8、13、21、34、.....在数学上，斐波纳契数列以如下递归的方法定义：

$$F(1) = 1, F(2) = 1, F(n) = F(n-1) + F(n-2) (n \geq 2, n \in N^*)$$

生成器法：

```
1 def fib(n):
2     if n == 0:
3         return False
4     if not isinstance(n, int) or (abs(n) != n):
5         return False
6     a, b = 0, 1
7     while n:
8         a, b = b, a + b
9         n -= 1
10        yield a
11
12 >>> fib(10)
13 <generator object fib at 0x000001AF2F2395C8>
14 >>> [i for i in fib(10)]
15 [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

递归法：

```
1 def fib(n):
2     if n == 0:
3         return False
4     if not isinstance(n, int) or (n < 0):
5         return False
6     if n <= 1:
7         return 1
8     return fib(n - 1) + fib(n - 2)
9
10 >>> fib(10)
11 55
12
```

```
13 | >>> [fib(i) for i in range(1, 11)]  
    | [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

64. 正则切分字符串

```
1 | >>> import re  
2 | >>> str1 = "hello world:luobo dazahui"  
3 | >>> re.split(r":| ", str1)  
4 | ['hello', 'world', 'luobo', 'dazahui']
```

65. yield 用法

yield 是用来生成迭代器的语法，在函数中，如果包含了 **yield**，那么这个函数就是一个迭代器。当代码执行至 **yield** 时，就会中断代码执行，直到程序调用 **next()** 函数时，才会在上次 **yield** 的地方继续执行。

详情见：《Python中的迭代器和生成器》。

67. 冒泡排序

```
1 | def Bubble(data, reversed):  
2 |     for i in range(len(data) - 1):  
3 |         for j in range(len(data) - i - 1):  
4 |             if data[j] > data[j + 1]:  
5 |                 data[j], data[j + 1] = data[j + 1], data[j]  
6 |     if reversed:  
7 |         data.reverse()  
8 |     return data  
9 |  
10 | >>> Bubble(list1, True)  
11 | [11, 9, 8, 5, 3, 2]  
12 | >>> Bubble(list1, False)  
13 | [2, 3, 5, 8, 9, 11]
```


68. 快速排序

- **思想**：首先任意选取一个数据（通常选用数组的第一个数）作为关键数据，然后将所有比它小的数都放到它前面，所有比它大的数都放到它后面，这个过程称为一趟快速排序，之后再递归排序两边的数据。
- **挑选基准值**：从数列中挑出一个元素，称为“基准”（pivot）。
- **分割**：重新排序数列，所有比基准值小的元素摆放在基准前面，所有比基准值大的元素摆在基准后面（与基准值相等的数可以到任何一边）。在这个分割结束之后，对基准值的排序就已经完成。
- **递归排序子序列**：递归地将小于基准值元素的子序列和大于基准值元素的子序列排序。

```
1 def partition(arr, low, high):
2     pivot = arr[low]
3     while low < high:
4         while low < high and arr[high] >= pivot:
5             high -= 1
6         arr[low], arr[high] = arr[high], arr[low]
7         while low < high and arr[low] <= pivot:
8             low += 1
9         arr[low], arr[high] = arr[high], arr[low]
10    return low
11
12
13 def quickSort(arr, low, high):
14     if low < high:
15         pi = partition(arr, low, high)
```

70. requests 简介

该库是发起 HTTP 请求的强大类库，调用简单，功能强大。

```
1 >>> import requests
2 >>> url = "http://www.baidu.com"
3 >>> response = requests.get(url) # 获得请求
4 >>> response.encoding = "utf-8" # 改变其编码
5 >>> html = response.text # 获得网页内容
6 >>> binary_content = response.content # 获得二进制数据
7 >>> raw = requests.get(url, stream=True) # 获得原始响应内容
8 >>> headers = {'user-agent': 'my-test/0.1.1'} # 定制请求头
9 >>> r = requests.get(url, headers=headers)
10
```

```
11 | >>> cookies = {"cookie": "# your cookie"} # cookie的使用
    >>> r = requests.get(url, cookies=cookies)
```

71. 比较两个 json 数据是否相等

方法一：

1. json数据转换成字典；
2. 将两个字典按 **key** 排好序，然后使用 **zip()** 函数将两个字典对应的元素打包成元组。比较对应的元素的 **value** 是否相等；
3. **zip()** 函数用于将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组。

```
1 | >>> file_text2 = '{"name":"john","age":22,"sex":"woman","address":"US"
2 | >>> file_text1 = '{"name":"john","age":22,"sex":"man","address":"USA"
3 | >>> dict1 = json.loads(file_text1)
4 | >>> dict2 = json.loads(file_text2)
5 | >>> for s1, s2 in zip(sorted(dict1), sorted(dict2)):
6 |     if str(dict1[s1]) != str(dict2[s2]):
7 |         print(s1 + ":" + dict1[s1] + "!=" + s2 + ":" + dict2[s2])
8 | sex:man!=sex:woman
```

方法二：引入第三方模块 **jsonpatch**。

```
1 | >>> import jsonpatch
2 | >>> src = {'numbers': [1, 3, 4, 8], 'foo': 'bar'}
3 | >>> dst = {'foo': 'bar', 'numbers': [1, 3, 8]}
4 | >>> patch = jsonpatch.JsonPatch.from_diff(src, dst)
5 | >>> print(patch)
6 | [{"op": "remove", "path": "/numbers/2"}]
```

说明：两个 **json** 对象，**src** 要变成 **dst**，需要移除 **numbers** 下索引是 **2** 的元素。

```
1 | >>> src = {'numbers': [1, 3, 4, 8], 'foo': 'bar'}
2 | >>> dst = {'foo': 'bar', 'numbers': [1, 3, 4, 8]}
3 | >>> patch = jsonpatch.JsonPatch.from_diff(src, dst)
4 | >>> print(patch)
5 | []
```

若相等的话直接返回空列表。

有了这个模块，判断json串是否相等不用遍历内部属性，直接判断 `patch` 长度即可。

72. 读取键盘输入

```
1 def forinput():
2     input_text = input()
3     print("your input text is: ", input_text)
4
5 >>> forinput()
6 >? 2
7 your input text is:  2
```

73. `enumerate()` 的用法

`enumerate()` 是 Python 内置函数。`enumerate()` 函数用于将一个可遍历的数据对象（如列表、元组或字符串）组合为一个索引序列，同时列出数据和数据下标，一般用在 `for` 循环当中。

它允许我们遍历数据并自动计数，用法如下：

```
1 for counter, value in enumerate(some_list):
2     print(counter, value)
```

不只如此，`enumerate()` 也接受一些可选参数，这使它更有用。

```
1 >>> my_list = ['apple', 'banana', 'grapes', 'pear']
2 >>> for c, value in enumerate(my_list, 1):
3 ...     print(c, value)
4 ...
5 1 apple
6 2 banana
7 3 grapes
8 4 pear
```

上面这个可选参数允许我们定制从哪个数字开始枚举。

此外，还可以用来创建包含索引的元组列表，例如：

```
1 >>> my_list = ['apple', 'banana', 'grapes', 'pear']
2 >>> counter_list = list(enumerate(my_list, 1))
3 >>> print(counter_list)
4 [(1, 'apple'), (2, 'banana'), (3, 'grapes'), (4, 'pear')]
```

74. pass 语句

`pass` 是空语句，是为了保持程序结构的完整性。`pass` 不做任何事情，一般用做占位语句。

```
1 def forpass(n):
2     if n == 1:
3         pass
4     else:
5         print('not 1')
6
7 >>> forpass(1)
```

75. 正则匹配邮箱

```
1 >>> import re
2 >>> email_list = ["test01@163.com", "test02@163.123", ".test03g@qq.com"]
3 >>> for email in email_list:
4     ret = re.match("[\w]{4,20}@(\.)*\.com$", email)
5     if ret:
6         print("%s 是符合规定的邮件地址，匹配后结果是:%s" % (email, ret.group(1)))
7     else:
8         print("%s 不符合要求" % email)
9
10 test01@163.com 是符合规定的邮件地址，匹配后结果是:test01@163.com
11 test02@163.123 不符合要求
12 .test03g@qq.com 不符合要求
13 test04@gmail.com 是符合规定的邮件地址，匹配后结果是:test04@gmail.com
```

76. 统计字符串中大写字母的数量

```
1 >>> str2 = 'werrQWSDDiWuW'
2 >>> counter = 0
3 >>> for i in str2:
4     if i.isupper():
5         counter += 1
6 >>> counter
7 6
```

77. json 序列化时保留中文

普通序列化：

```
1 >>> import json
2 >>> dict1 = {'name': '萝卜', 'age': 18}
3 >>> dict1_new = json.dumps(dict1)
4 >>> print(dict1_new)
5 {"name": "\u841d\u535c", "age": 18}
```

保留中文：

```
1 >>> import json
2 >>> dict1 = {'name': '萝卜', 'age': 18}
3 >>> dict1_new = json.dumps(dict1, ensure_ascii=False)
4 >>> print(dict1_new)
5 {"name": "萝卜", "age": 18}
```

78. 简述继承

一个类继承自另一个类，也可以说是一个孩子类/派生类/子类，继承自父类/基类/超类，同时获取所有的类成员（属性和方法）。继承使我们可以重用代码，并且还可以更方便地创建和维护代码。

Python 支持以下类型的继承：

- 单继承：一个子类类继承自单个基类
- 多重继承：一个子类继承自多个基类
- 多级继承：一个子类继承自一个基类，而基类继承自另一个基类
- 分层继承：多个子类继承自同一个基类

- 混合继承：两种或两种以上继承类型的组合

79. 什么是猴子补丁 (❤️❤️)

猴子补丁是指在运行时**动态修改类和模块**。

猴子补丁主要有以下几个用处：

- 在运行时替换方法、属性等；
- 在不修改第三方代码的情况下增加原来不支持的功能；
- 在运行时为内存中的对象增加 patch 而不是在磁盘的源代码中增加。

详细看链接：[《Python进阶系列》十七：详解Python中的猴子补丁——允许在运行时更改对象的行为](#)

80. `help()` 函数和 `dir()` 函数

`help()` 函数返回帮助文档和参数说明

```
1 >>> help(dict)
2 Help on class dict in module builtins:
3 class dict(object)
4 |     dict() -> new empty dictionary
5 |     dict(mapping) -> new dictionary initialized from a mapping object
6 |         (key, value) pairs
7 |     dict(iterable) -> new dictionary initialized as if via:
8 |         d = {}
9 |         for k, v in iterable:
10 |             d[k] = v
11 |     dict(**kwargs) -> new dictionary initialized with the name=value |
12 |         in the keyword argument list.  For example:  dict(one=1, two=
13 |
14 |     Methods defined here:
15 |     .....
```

`dir()` 函数返回对象中的所有成员 (任何类型)

```
1 >>> dir(dict)
2 ['__class__',
3  '__contains__',
```

```
4 | '__delattr__',
5 | '__delitem__',
6 | '__dir__',
7 | '__doc__',
8 | '__eq__',
9 | '__format__',
10 | '__ge__',
11 | '__getattribute__',
12 | .....
```

81. 解释 Python 中的 `//` , `%` 和 `**` 运算符

- `//` 运算符执行地板除法，返回结果的整数部分 (向下取整)。
- `%` 是取模符号，返回除法后的余数。
- `**` 符号表示取幂。 `a**b` 返回 `a` 的 `b` 次方。

82. 主动抛出异常

使用 `raise` :

```
1 | def test_raise(n):
2 |     if not isinstance(n, int):
3 |         raise Exception("not a int type")
4 |     else:
5 |         print("good")
6 |
7 | >>> test_raise(8.9)
8 | Traceback (most recent call last):
9 |   File "D:\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py",
10 |     exec(code_obj, self.user_global_ns, self.user_ns)
11 |   File "<ipython-input-106-27962090d274>", line 1, in <module>
12 |     test_raise(8.9)
13 |   File "<ipython-input-105-8095f403c7e2>", line 3, in test_raise
14 |     raise Exception("not a int type")
15 | Exception: not a int type
```

83. tuple 和 list 转换

```
1 >>> tuple1 = (1, 2, 3, 4)
2 >>> list1 = list(tuple1)
3 >>> print(list1)
4 >>> tuple2 = tuple(list1)
5 >>> print(tuple2)
```

84. 简述断言

Python 的断言就是检测一个条件，如果条件为真，它什么都不做；反之它触发一个带可选错误信息的 `AssertionError`。

```
1 def testassert(n):
2     assert n == 2, "n is not 2"
3     print("n is 2")
4
5 >>> testassert(2)
6 n is 2
7 >>> testassert(3)
8 Traceback (most recent call last):
9   File "D:\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py",
10     exec(code_obj, self.user_global_ns, self.user_ns)
11   File "<ipython-input-109-9b7ba30779d0>", line 1, in <module>
12     testassert(3)
13   File "<ipython-input-107-ecbc45f8f4b4>", line 2, in testassert
14     assert n == 2, "n is not 2"
15 AssertionError: n is not 2
```

85. 什么是异步非阻塞

同步异步指的是调用者与被调用者之间的关系。

- 所谓同步，就是在发出一个功能调用时，在没有得到结果之前，该调用就不会返回，一旦调用返回，就得到了返回值；
- 异步的概念和同步相对，调用在发出之后，这个调用就直接返回了，所以没有返回结果。当该异步功能完成后，被调用者可以通过状态、通知或回调来通知调用者。

阻塞非阻塞是线程或进程之间的关系。

- 阻塞调用是指调用结果返回之前，当前线程会被挂起（如遇到io操作）。调用线程只有在得到结果之后才会返回。函数只有在得到结果之后才会将阻塞的线程激活
- 非阻塞和阻塞的概念相对应，非阻塞调用指在不能立刻得到结果之前也会立刻返回，同时该函数不会阻塞当前线程

86. 什么是负索引

Python 中的序列是有索引的，它由正数和负数组成。正的数字使用'0'作为第一个索引，'1'作为第二个索引，以此类推。负数的索引从'-1'开始，表示序列中的最后一个索引，'-2'作为倒数第二个索引，依次类推。

87. 退出 Python 后，内存是否全部释放

不是的，那些具有对象循环引用或者全局命名空间引用的变量，在 Python 退出时往往不会被释放，
另外不会释放 C 库保留的部分内容。

88. Flask 和 Django 的异同

- Flask 是 “microframework”，主要用来编写小型应用程序，不过随着 Python 的普及，很多大型程序也在使用 Flask。同时，在 Flask 中，我们必须使用外部库。
- Django 适用于大型应用程序。它提供了灵活性，以及完整的程序框架和快速的项目生成方法。可以选择不同的数据库，URL结构，模板样式等。

89. 创建删除操作系统上的文件

```
1 >>> f = open('test.txt', 'w')
2 >>> f.close()
3 >>> os.listdir()
4 ['.idea',
5  'test.txt',
6  '__pycache__']
7
```

```
8  
9 >>> os.remove('test.txt')  
10 >>> os.listdir()  
11 ['.idea',  
    '__pycache__']
```

90. 简述 logging 模块

logging 模块是 Python 内置的标准模块，主要用于输出运行日志，可以设置输出日志的等级、日志保存路径、日志文件回滚等；相比 print，具备如下优点：

- 可以通过设置不同的日志等级，在 release 版本中只输出重要信息，而不必显示大量的调试信息
- print 将所有信息都输出到标准输出中，严重影响开发者从标准输出中查看其它数据；logging 则可以由开发者决定将信息输出到什么地方，以及怎么输出。

简单配置：

```
1 >>> import logging  
2 >>> logging.debug("debug log")  
3 >>> logging.info("info log")  
4 >>> logging.warning("warning log")  
5 >>> logging.error("error log")  
6 >>> logging.critical("critica log")
```

默认情况下，只显示了大于等于WARNING级别的日志。`logging.basicConfig()` 函数调整日志级别、输出格式等。

详细看链接：[《Python进阶系列》九：别再用print来打印了，试试logging模块](#)

91. 统计字符串中字符出现次数

```
1 >>> from collections import Counter  
2 >>> str1 = "nihsasehndcismewetpxc"  
3 >>> print(Counter(str1))  
4 Counter({'s': 3, 'e': 3, 'n': 2, 'i': 2, 'h': 2, 'c': 2, 'a': 1, 'd':
```

92. 正则 re.compile 的作用

`re.compile` 是将正则表达式编译成一个对象，加快速度，并重复使用。

93. try except else finally 的意义

- `try..except..else` 没有捕获到异常，执行 `else` 语句
- `try..except..finally` 不管是否捕获到异常，都执行 `finally` 语句

94. 反转列表

第一种方法：使用切片

```
1 >>> list1 = list(range(10))
2 >>> list1[::-1]
3 [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

第二种方法：使用 `reverse()`

```
1 >>> list1 = list(range(10))
2 >>> list1.reverse()
3 >>> list1
4 [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

需要注意的是：这两种方法都可以反转列表，但内置函数 `reverse()` 会更改原始列表，而切片方法会创建一个新列表。内置函数 `reverse()` 比列表切片方法更快！

95. 字符串中数字替换

使用 `re` 正则替换：

```
1 >>> import re
2 >>> str1 = '我是周萝卜，今年18岁'
3
```

```
4 >>> re.sub(r"\d+", "20", str1)
'我是周萝卜, 今年20岁'
```

96. 读取大文件 (❤️❤️)

现要处理一个大小为10G的文件，但是内存只有4G，如果在只修改get_lines 函数而其他代码保持不

变的情况下，应该如何实现？需要考虑的问题都有哪些？

```
1 def get_lines():
2     with open('file.txt', 'rb') as f:
3         return f.readlines()
4
5 if name == ' main ':
6     for e in get_lines():
7         process(e) # 处理每一行数据
```

方法一：readlines() 函数在文件过大时并不适用，应添加参数，限制读取的字节数，并使用生成器。

```
1 def get_lines():
2     l = []
3     with open('file.txt', 'rb') as f:
4         data = f.readlines(60000)
5         l.append(data)
6     yield l
```

方法二：使用mmap

```
1 from mmap import mmap
2
3 def get_lines(fp):
4     with open(fp, "r+") as f:
5         m = mmap(f.fileno(), 0)
6         tmp = 0
7         for i, char in enumerate(m):
8             if char==b"\n":
9                 yield m[tmp:i + 1].decode()
10                tmp = i + 1
11
12 if name == " main ":
13     for i in get_lines("fp_some_huge_file"):
14         print(i)
```

详细看链接： [《Python进阶系列》二十八：mmap模块（处理大文本）](#)

97. 输入日期，判断这一天是这一年的第几天

```
1 import datetime
2
3 def dayofyear():
4     year = input("请输入年份：")
5     month = input("请输入月份：")
6     day = input("请输入天：")
7     date1 = datetime.date(year=int(year), month=int(month), day=int(d
8     date2 = datetime.date(year=int(year), month=1, day=1)
9     return (date1 - date2).days + 1
10
11 >>> dayofyear()
12 请输入年份：>? 2022
13 请输入月份：>? 5
14 请输入天：>? 23
15 143
```

98. 排序

根据字典按值排序

现有字典 `d= {'a':24, 'g':52, 'i':12, 'k':33}` 请按 `value` 值进行排序？

```
1 >>> d = {'a': 24, 'g': 52, 'i': 12, 'k': 33}
2 >>> d.items()
3 dict_items([('a', 24), ('g', 52), ('i', 12), ('k', 33)])
4
5 >>> sorted(d.items(), key=lambda x: x[1])
6 [('i', 12), ('a', 24), ('k', 33), ('g', 52)]
```

`x[0]` 代表用 `key` 进行排序；`x[1]` 代表用 `value` 进行排序。

请按alist中元素的age由大到小排序

```
1 >>> alist = [{'name': 'a', 'age': 20}, {'name': 'b', 'age': 30}, {'na
2 >>> sorted(alist, key=lambda x:x['age'], reverse=True)
3
```

```
[{'name': 'b', 'age': 30}, {'name': 'c', 'age': 25}, {'name': 'a', 'a
```

列表内，字典按照 value 大小排序

```
1 >>> list1 = [{'name': 'guanyu', 'age':29},
2 ...         {'name': 'zhangfei', 'age': 28},
3 ...         {'name': 'liubei', 'age':31}]
4 >>> sorted(list1, key=lambda x:x['age'])
5 [{'name': 'zhangfei', 'age': 28}, {'name': 'guanyu', 'age': 29}, {'na
6 >>> sorted(list1, key=lambda x:x['name'])
7 [{'name': 'guanyu', 'age': 29}, {'name': 'liubei', 'age': 31}, {'name
```

99. 将字符串处理成字典

```
1 >>> k = "k:1|k1:2|k2:3|k3:4"
2 >>> dict1 = {}
3 >>> for items in k.split("|"):
4 ...     key, value = items.split(":")
5 ...     dict1[key] = value
```

字典推导式：

```
1 >>> {k:v for items in k.split("|") for k, v in (items.split(":"), )}
2 {'k': '1', 'k1': '2', 'k2': '3', 'k3': '4'}
```

100. 下面代码的输出结果将是什么？

```
1 list = ['a','b','c','d','e']
2 print(list[10:])
```

代码将输出 `[]`，不会产生 `IndexError` 错误，就像所期望的那样，尝试用超出成员的个数的index来获取某个列表的成员。例如，尝试获取 `list[10]` 和之后的成员，会导致 `IndexError`。然而，尝试获取列表的切片，开始的 `index` 超过了成员个数不会产生 `IndexError`，而是仅仅返回一个空列表。这成为特别让人恶心的疑难杂症，因为运行的时候没有错误产生，导致Bug很难被追踪到。

📖 文章知识点与官方知识档案匹配，可进一步学习相关知识

Python入门技能树 > 首页 > 概览 353591 人正在系统学习中




显示推荐内容

 AlphaGuaGua

关注

18 条评论 >

 Discontinuouser

热评 53中的sort()有参数, reverse, 默...

写评论