

函数

本章目标

- 掌握函数的基本概念
- 掌握函数的编写、使用方法
- 掌握变量名解析方式
- 掌握函数参数传递规范
- 掌握匿名函数
- 了解函数的高级用法

1. 函数基础

1.1 pycharm快捷键

Ctrl + P 参数信息（在方法中调用参数）

Ctrl + Q 快速查看文档

1.2 函数作用

- 最大化代码复用
- 最小化代码冗余
- 对逻辑流程进行分解
- 面向过程程序设计的基本单元

函数在开发过程中，可以更高效的实现**代码重用**(代码层次结构更清晰)。

1.3 函数定义

(1) 用于封装一个特定的功能，表示一个功能或者行为。

(2) 函数是可以重复执行的语句块，可以重复调用。

语法：

```
1 def <name> ([arg1][, arg2][, ...][, argn]):  
2     <statements>  
3     .....  
4     [return [val]]
```

说明：

def 关键字：全称是define，意为“定义”。

函数名：对函数体中语句的描述，规则与变量名相同。

形式参数：函数定义时的参数，实际就是变量名。

函数体：完成该功能的语句。

注意：

- 1、代码中的缩进很重要，区分不同的代码块

- 2、不同的需求，参数可有可无。
- 3、在Python中，函数必须**先定义后使用**。

def语句：

- def是可执行的Python语句
- def执行之后函数对象才被创建
- def可以出现在程序的任何位置
- 本质是将函数对象赋值给函数名
- 函数也可以是空函数
- def后面跟函数名和参数

return语句：

- 返回一个对象给调用者
- return之后的语句被忽略
- 返回值形式

返回值的数目	实际返回的对象
=0	None
=1	object
>1	tuple

函数的三要素：

- 函数名
- 参数列表
- 返回值

1.4 函数调用

(1) 语法：函数名(实际参数)

(2) 说明：

实际参数：函数调用时的参数，实际就是变量值。

```
1 # 实际参数：真实的具体的数据
2 attack(5)
3 attack(2)
```

函数调用的基本使用

```

1  #创建函数对象，赋值给add
2  def add(a, b):
3      # 必须要缩进
4      return a+b
5  #alias与add引用同一函数
6  alias = add #函数可以有多个引用
7  add(1, 2) #调用函数
8  alias(1, 2) #调用函数

```

练习1：定义函数，在终端中打印一维列表。

```

1  list01 = [5, 546, 6, 56, 76, ]
2  for item in list01:
3      print(item)
4
5  list02 = [7,6,879,9,909,]
6  for item in list02:
7      print(item)

```

练习2：创建函数，在终端中打印矩形。

```

1  number = int(input("请输入整数: ")) # 5
2  for row in range(number):
3      if row == 0 or row == number - 1:
4          print("'" * number)
5      else:
6          print("%s*" % (" " * (number - 2)))

```

1.5 返回值

(1) 定义：

函数定义者告诉调用者的结果。

(2) 语法：

return 数据

(3) 说明：

return后没有语句，相当于返回 None。

函数体没有return，相当于返回None。

```

1  def func01():
2      print("func01执行了")
3      return 100
4
5  # 1. 调用者,可以接收也可以不接收返回值
6  func01()
7  res = func01()
8  print(res)

```

```

9
10 # 2.在Python语言中,
11 # 函数没有return或return后面没有数据,
12 # 都相当于return None
13 def func02():
14     print("func02执行了")
15     return
16
17 res = func02()
18 print(res) # None
19
20 # 3.return可以退出函数
21 def func03():
22     print("func03执行了")
23     return
24     print("func03又执行了")
25
26 func03()
27
28 # 4. return 可以退出多层循环嵌套
29 def func04():
30     while True:
31         while True:
32             while True:
33                 # break 只能退出一层循环
34                 print("循环体")
35                 return
36
37 func04()

```

1.6 课堂案例

创建一个累加函数，计算湖北省52天总病人数

```

1 # 新冠疫情发生之后湖北省52日增长的数值
2 Hubei = [
3     549, 181, 328, 365, 1291, 840, 1032, 1220, 1347, 1921, 2103, 2345, 3156,
4     2987, 2447, 2841, 2147, 2531, 2097, 1638, 14840, 3780, 2420, 1843, 1933,
5     1807, 1693, 349, 631, 792, 630, 203, 499, 401, 409, 318, 42, 570, 196,
6     114, 114, 135, 126, 74, 41, 36, 17, 13, 8, 5, 4, 4
7 ]
8 # 创建一个函数
9 def add_data(data):
10     # 定义一个总和
11     sum_num = 0
12     # 循环遍历
13     for i in data:
14         # 累加
15         sum_num += i
16     # 返回想要的结果
17     return sum_num
18 # 调用函数
19 req = add_data(Hubei)

```

```
20 # 打印结果
21 print(req)
```

练习1: 创建计算治愈比例的函数。

```
1 confirmed = int(input("请输入确诊人数: "))
2 cure = int(input("请输入治愈人数: "))
3 cure_rate = cure / confirmed * 100
4 print("治愈比例为" + str(cure_rate) + "%")
```

练习2: 定义函数, 根据总两数, 计算几斤零几两。

提示: 使用容器包装需要返回的多个数据

```
1 total_liang = int(input("请输入两: "))
2 jin = total_liang // 16
3 liang = total_liang % 16
4 print(str(jin) + "斤零" + str(liang) + "两")
```

练习3: 创建函数, 根据课程阶段计算课程名称。

```
1 number = input("请输入课程阶段数: ")
2 if number == "1":
3     print("Python语言核心编程")
4 elif number == "2":
5     print("Python高级软件技术")
6 elif number == "3":
7     print("web全栈")
8 elif number == "4":
9     print("项目实战")
10 elif number == "5":
11     print("数据分析、人工智能")
```

练习4: 创建函数, 计算IQ等级。

```
1 ma = int(input("请输入你的心里年龄: "))
2 ca = int(input("请输入你的实际年龄: "))
3 iq = ma / ca * 100
4 if 140 <= iq:
5     print("天才")
6 elif 120 <= iq:
7     print("超常")
8 elif 110 <= iq:
9     print("聪慧")
10 elif 90 <= iq:
11     print("正常")
12 elif 80 <= iq:
```

```
13     print("迟钝")
14 else:
15     print("低能")
```

练习5: 创建函数, 根据年龄计算人生阶段。

```
1 age = int(input("请输入年龄: "))
2 if age <= 6:
3     print("童年")
4 elif age <= 17: # 程序能执行到本行,说明age一定大于6
5     print("少年")
6 elif age <= 40:
7     print("青年")
8 elif age <= 65:
9     print("中年")
10 else:
11     print("老年")
```

练习6: 定义函数, 根据年月日计算是这一年的第几天。

如果2月是闰年, 则29天平年28

```
1 year = int(input("请输入年份: "))
2 if year % 4 == 0 and year % 100 != 0 or year % 400 == 0:
3     day = 29
4 else:
5     day = 28
```

```
1 month = int(input("请输入月: "))
2 day = int(input("请输入日: "))
3 days_of_month = (31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
4 total_days = sum(days_of_month[:month - 1])
5 total_days += day
6 print(f"{month}月{day}日是第{total_days}天。")
```

1.7 可变 / 不可变类型在传参时的区别

(1) 不可变类型参数有:

数值型(整数、浮点数)

布尔值bool

None 空值

字符串str

元组tuple

(2) 可变类型参数有:

列表 list

字典 dict

集合 set

(3) 传参说明：

不可变类型的数据传参时，函数内部不会改变原数据的值。

可变类型的数据传参时，函数内部可以改变原数据。

练习1：根据下列代码，创建降序排序函数。

```
1 list01 = [5, 15, 25, 35, 1, 2]
2
3 for r in range(len(list01) - 1):
4     for c in range(r + 1, len(list01)):
5         if list01[r] < list01[c]:
6             list01[r], list01[c] = list01[c], list01[r]
7
8 print(list01)
```

练习2：定义函数，将列表中大于某个值的元素设置为None

参数

结果

[34, 545, 56, 7, 78, 8] -10-> [None, None, None, 7, None, 8]

[34, 545, 56, 7, 78, 8] -100-> [34, None, 56, 7, 78, 8]

2. 函数参数

定义：函数名后面的括号中定义参数

本质：参数传递本质是将对象赋值给本地变量名

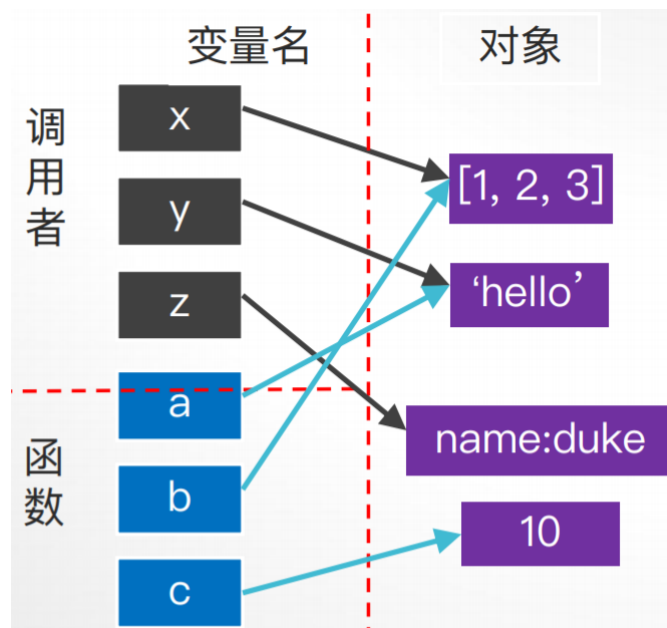
2.1 实参传递方式

2.1.1 默认参数

```

1  def foo(a, b, c=10):
2      print(a, b, c)
3  x = [1, 2, 3]
4  y = 'hello'
5  z = {'name': 'duke'}
6  d={
7      'a':x,
8      'b':y,
9      'c':z
10 }
11 # d 的abc不影响传参
12 foo(y, x)

```

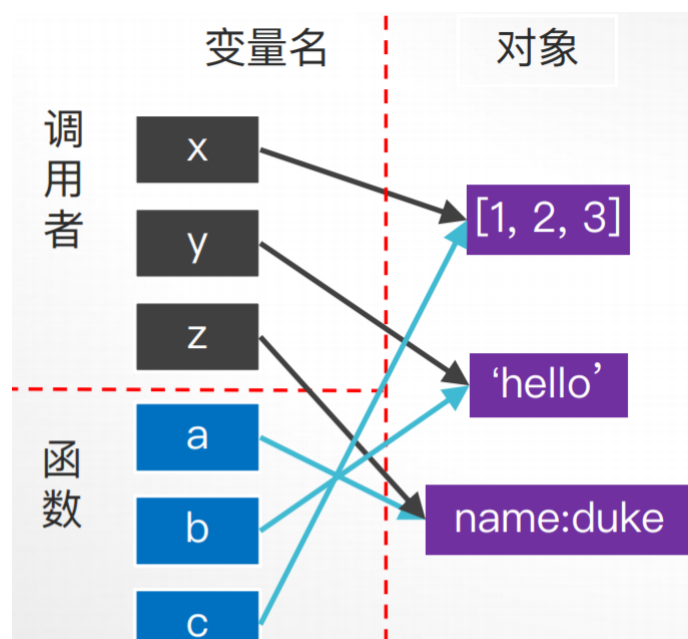
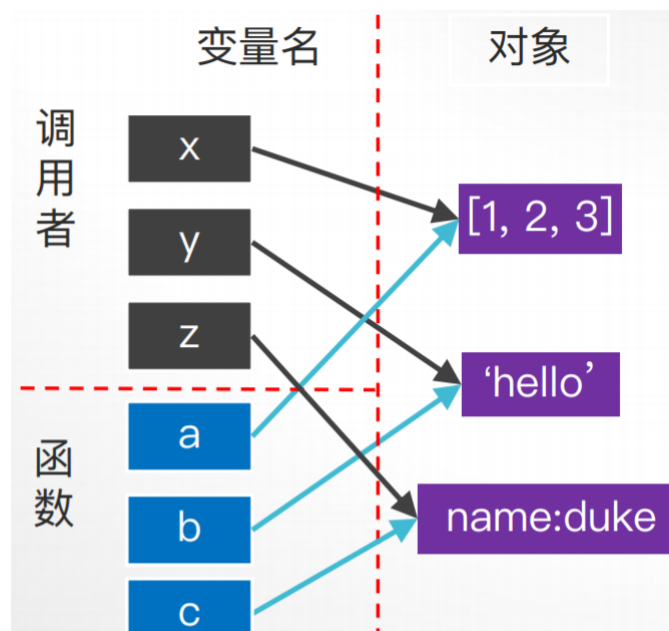


2.1.2 位置传参：按位置进行传参

```

1  def foo(a, b, c):
2      print(a, b, c)
3  x = [1, 2, 3]
4  y = 'hello'
5  z = {'name': 'duke'}
6  foo(x, y, z)
7  foo(z, y, x)

```

2.1.3 序列传参

定义：实参用*将序列拆解后与形参的位置依次对应。

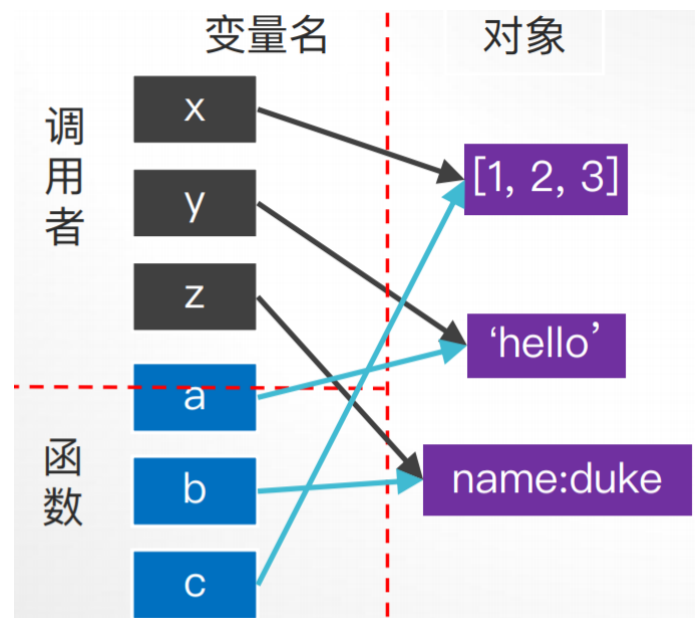
```

1  def func01(p1, p2, p3):
2      print(p1, p2, p3)
3
4  # 序列实参：拆,按照顺序与形参对应
5  list01 = [1, 2, 3]
6  name = "孙悟空"
7  tuple01 = (4, 5, 6)
8  # func01(list01)
9  func01(*list01) # 拆 1, 2, 3
10 func01(*name) # 拆 孙 悟 空
11 func01(*tuple01) # 拆 4, 5, 6

```

2.1.4 关键字传参：按关键字进行传参

```
1 def foo(a, b, c):
2     print(a, b, c)
3 x = [1, 2, 3]
4 y = 'hello'
5 z = {'name': 'duke'}
6 foo(y, c=x, b=z)
```



2.1.5 可变参数

1、可变参数-列表/元组

```
1 # 给定一组数字a, b, c....., 请计算a*a + b*b + c*c + .....。
2 # 由于参数个数不确定, 我们首先想到可以把a, b, c..... 作为一个list或tuple传进来
3 def calc(numbers):
4     sum = 0
5     for n in numbers:
6         sum = sum + n * n
7     return sum
8 # 使用函数
9 data1 = [1,1,2,3]
10 data2 = (2,2,3,4)
11 res = calc(data1)
12 print(res)
```

2、可变参数-字典

```

1  # 可变参数
2  def foo(**arg): #函数定义时采用 '*' 或者 '**'
3      for x in arg:
4          print(x, arg[x])
5  x = [1, 2, 3]
6  y = 'www.briup.com'
7  z = list('Python')
8  d = {'a':x, 'b':y, 'c':z}
9  #调用时需用 'key-value' 形式或传入字典
10 foo(a=x, b=y, c=z)
11 foo(**d)

```

4、keyword-only参数

- 函数定义时参数列表中用'*'分割
- '*'之后的参数必须采用**关键字**方式传递

```

1  def foo(a, *b, c):
2      print(a, b, c)
3
4  foo(1, c=3)          #ok
5  foo(1, 2, c=3)       #ok
6  foo(1, 2, 3, 4, c=5) #ok
7  foo(1, 2, 3)         #err
8
9
10 def foo(a, b, *, c):
11     print(a, b, c)
12
13 foo(1, 2, c=3)        #ok
14 foo(1, 2, 3)         #err
15 foo(1, 2, 3, 4, c=5) #err

```

注意：

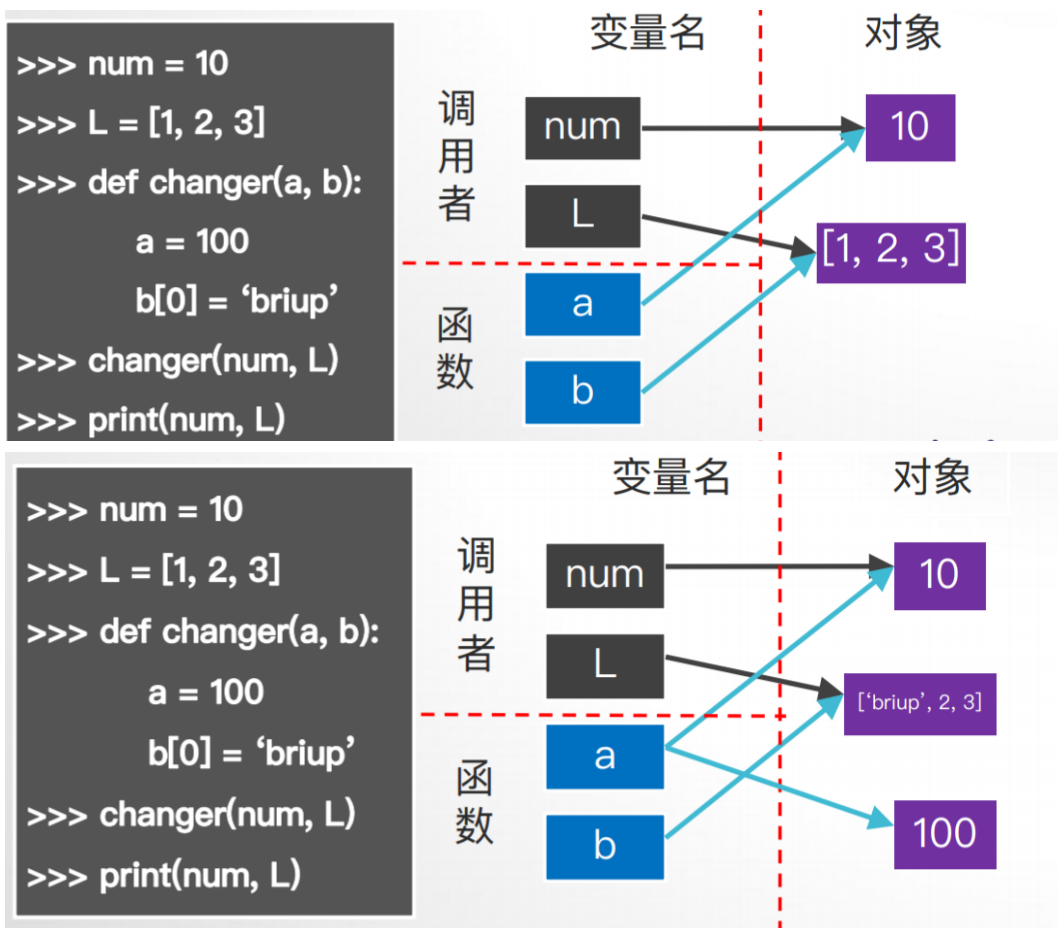
- 混合方式传参时：关键字传参写在位置传参之后
- 函数内部对已有变量名赋值不会影响原有的变量（例子）

```

1  a = 1
2  def changes(a):
3      a = 2
4      return a
5  print(a)
6  print(changes(a))

```

- 有时候则会修改原有的变量



- 可变参数就是传入的参数个数是可变的，可以是1个、2个到任意个，还可以是0个。
 - 函数定义时使用 `*` 或者 `**`

```

1  # 给定一组数字a, b, c....., 请计算a2 + b2 + c2 + .....。
2  # 由于参数个数不确定，我们首先想到可以把a, b, c..... 作为一个list或tuple传进来
3  def calc(numbers):
4      sum = 0
5      for n in numbers:
6          sum = sum + n * n
7      return sum
8  # 使用函数
9  data1 = [1,1,2,3]
10 data2 = (2,2,3,4)
11 res = calc(data1)
12 print(res)

```

```

1  # 使用可变参数
2  def calc(*numbers):
3      sum = 0
4      for n in numbers:
5          sum = sum + n * n
6      return sum
7  # 使用函数
8  res = calc(1,1,2,1,2)
9  print(res)

```

```

1  # 函数定义
2  def foo(位置参数, 关键字参数, *解包参数, **解包参数):
3      pass
4  # 函数调用
5      # 和函数定义相同
6

```

函数设计原则

- 高内聚、低耦合
- 函数功能单一，目标明确
- 非必要情况下尽量减少使用全局变量
- 尽量不修改可变类型参数，除非函数功能如此
- 函数规模尽量小
- 避免直接修改在另一个模块中的变量

2.2 形参定义方式

2.2.1 缺省形参

(1) 语法：

def 函数名(形参名1=默认实参1, 形参名2=默认实参2, ...):

函数体

(2) 说明：

缺省参数必须自右至左依次存在，如果一个参数有缺省参数，则其右侧的所有参数都必须有缺省参数。

```

1  def func01(p1 =0, p2="", p3=100):
2      print(p1)
3      print(p2)
4      print(p3)
5
6  func01(p2=2)
7  func01(p2=2, p3=3)
8  # 支持同时使用位置实参与关键字实参
9  func01(1, p3=3)
10 # 注意：先位置实参,后关键字实参
11 # func01(p1 =1,2,3) # 错误

```

练习：

定义函数，根据小时、分钟、秒，计算总秒数。

调用：提供小时、分钟、秒

调用：提供分钟、秒

调用：提供小时、秒

调用：提供分钟

2.2.2 位置形参

语法:

```
def 函数名(形参名1, 形参名2, ...):
```

```
    函数体
```

2.2.3 命名关键字形参

(1) 语法:

```
def 函数名(*args, 命名关键字形参1, 命名关键字形参2, ...):
```

```
    函数体
```

```
def 函数名(*, 命名关键字形参1, 命名关键字形参2, ...):
```

```
    函数体
```

(2) 作用:

强制实参使用关键字传参

```
1  # 命名关键字形参:
2  # 星号元组形参后面的位置形参
3  # 限制实参必须是关键字实参
4  def func01(*args, p1, p2):
5      print(args)
6      print(p1)
7      print(p2)
8
9
10 func01(p1=1, p2=2)
11 func01(1, 2, 3, p1=1, p2=2)
12
13
14 def func02(p1, *, p2=0):
15     print(p1)
16     print(p2)
17
18
19 # 通常星号后面的命名关键字形参属于辅助参数,可选。
20 func02(1)
21 func02(1, p2=2)
```

2.2.4 星号元组形参

(1) 语法:

```
def 函数名(*元组形参名):
```

```
    函数体
```

(2) 作用:

可以将多个位置实参合并为一个元组

(3) 说明:

一般命名为'args'

形参列表中最多只能有一个

```
1 # 位置实参数量可以无限
2 def func01(*args):
3     print(args)
4
5 func01() # 空元组
6 func01(1, 2, 34) # (1, 2, 34)
7 # 不支持关键字实参
8 # func01(args = 1,a=1)
```

练习：定义数值累乘的函数

2.2.5 双星号字典形参

(1) 语法：

def 函数名(**字典形参名):

函数体

(2) 作用：

可以将多个关键字实参合并为一个字典

(3) 说明：

一般命名为'kwargs'

形参列表中最多只能有一个

```
1 # 关键字实参数量无限
2 def func01(**kwargs):
3     print(kwargs) # {'a': 1, 'b': 2}
4
5 func01(a=1,b=2)
6 # func01(1,2,3) # 报错
```

2.2.6 参数自左至右的顺序

位置形参 --> 星号元组形参 --> 命名关键字形参 --> 双星号字典形参

练习：说出程序执行结果。

```
1 def func01(list_target):
2     print(list_target)# ?
3
4 def func02(*args):
5     print(args)# ?
6
7 def func03(*args,**kwargs):
8     print(args)# ?
9     print(kwargs)# ?
10
```

```

11 def func04(p1,p2,*,p4,**kwargs):
12     print(p1)# ?
13     print(p2)# ?
14     print(p4)# ?
15     print(kwargs)# ?
16
17 func01([1,2,3])
18 func02(*[1,2,3])
19 func03(1,2,3,a=4,b=5,c=6)
20 func04(10,20,p4 = 30,p5 = 40)

```

3. 作用域（命名空间）

3.1 定义

一段程序代码中所用到的名字不总是有效和可用的，而限定这个名字的可用性的代码范围就是这个名字的作用域。简单来说，作用域决定了一个标识的可用范围。

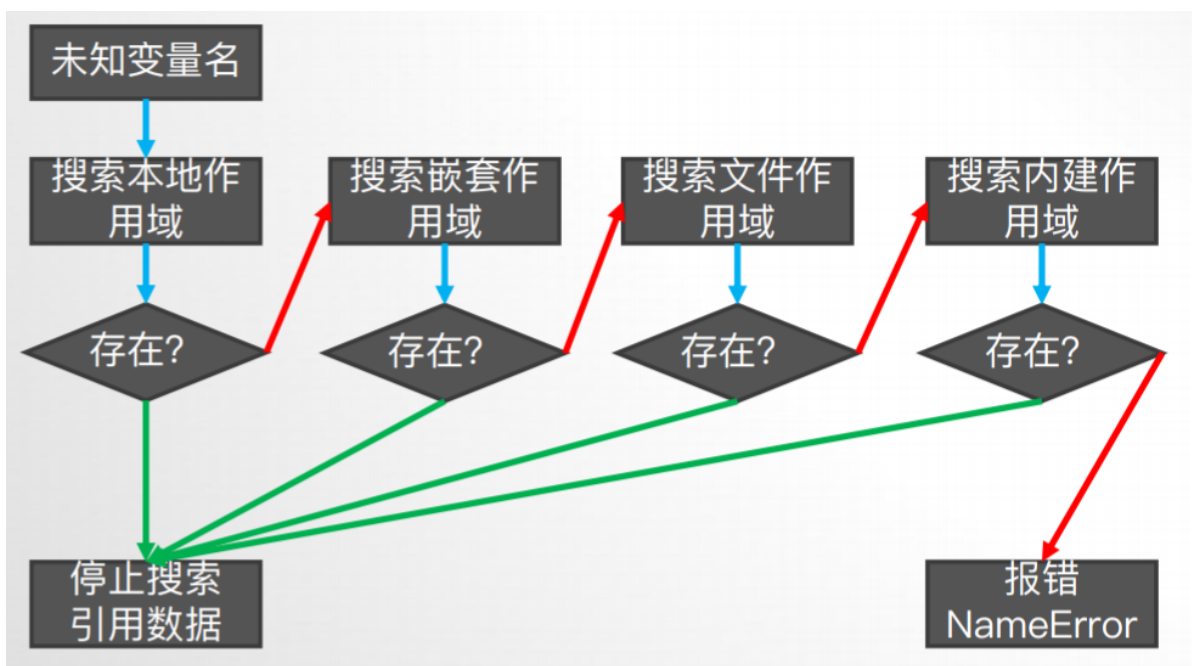
3.2 作用域分类

- 本地作用域【Local】：函数内部
- 嵌套作用域【Enclosing function local】：函数嵌套
- 全局作用域【Global】：模块(.py文件)内部
- 内建作用域【Built-in】builtins.py文件

3.3 变量名的查找规则

(1) 由内到外：L -> E -> G -> B

(2) 在访问变量时，先查找本地变量，然后是包裹此函数外部的函数内部的变量，之后是全局变量，最后是内置变量。



3.4 局部变量

- (1) 定义在函数内部的变量(形参也是局部变量)
- (2) 只能在函数内部使用
- (3) 调用函数时才被创建，函数结束后自动销毁

3.5 全局变量

- (1) 定义在函数外部，模块内部的变量。
- (2) 在整个模块(py文件)范围内访问（但函数内不能对其直接赋值）。

3.3 本地作用域

- 所有在**函数内部**使用的（首次赋值）标识符
- 且没有global关键字修饰
- 本地作用域用于存放函数的实参、本地变量等

```
1 def getName(url):
2     names = url.split('.')
3     name = names[1]
4     return name
5 getName("www.briup.com")
```

本地变量：

- 在函数内部定义的，只允许函数内访问的变量
- 不同的函数，可以定义相同名字的本地变量，同名变量之间 互不影响
- 在函数中定义本地变量来存储临时数据
- 当函数处于活动状态时存在
- 本地变量具备本地作用域

嵌套作用域

- 函数内定义新的函数时会出现【Enclosing local】
- 定义在当前函数之外，上层函数之内的变量

```
def foo():
    var = 10
    def foo1():
        var1 = var
        print(var1)
    foo1()
```

全局作用域

全局变量的默认作用域是整个程序，即全局变量既可以在各个函数的外部使用，也可以在各函数内部使用。

- 全局作用域也称文件作用域
- 定义在模块文件最外层的变量

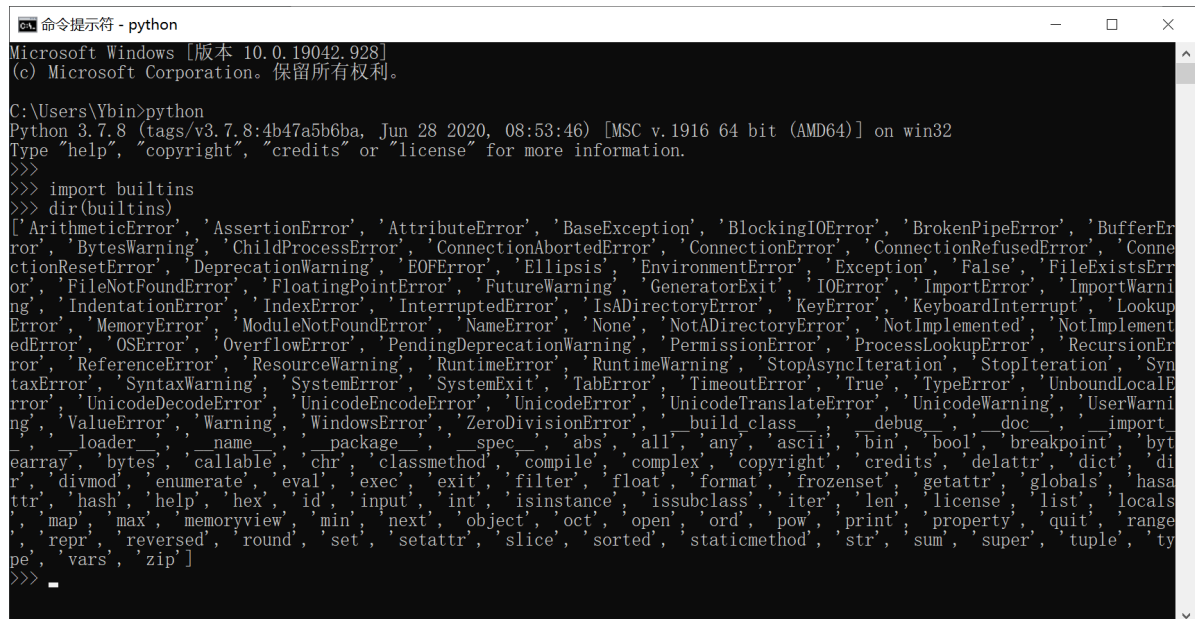
- 被global关键字修饰的变量

```
1 # 全局变量 作用域是整个程序
2 url = "www.briup.com"
3
4 def getName(url1):
5     # 局部变量
6     names = url1.split('.')
7     # print(names)
8     # 拼接
9     url_name = url + names[1]
10    # name = names[1]
11    return url_name
12
13 url_name = getName("123.ab")
14 print(url_name)
15
```

内建作用域

所有的内建作用域：

```
1 import builtins
2 dir(builtins)
```



注意：

- 内建作用域属于整个python系统
- 所有python程序共用其中的标识符
- 常用的内建函数：
 - type()
 - dir()
 - len()
 - help()
 - print()
 -

- 本地作用域可能覆盖全局作用域或内置作用域
- 若本地标识符与内置标识符重名，无警告或报错信息，会引起其他编程BUG(尽量不要重名)

全局变量 作用域是整个程序

```
url = "www.briup.com"
```

```
def getName(weburl):
```

```
    # 局部变量
```

```
    names = weburl.split('.')
```

```
    name = names[1]
```

内建作用域

```
    print("名字是: %s"%name)
```

本地作用域

```
def printVersion(vision):
```

嵌套作用域

```
    print(name + "vision: %s"%vision)
```

```
printVersion("hello,briup")
```

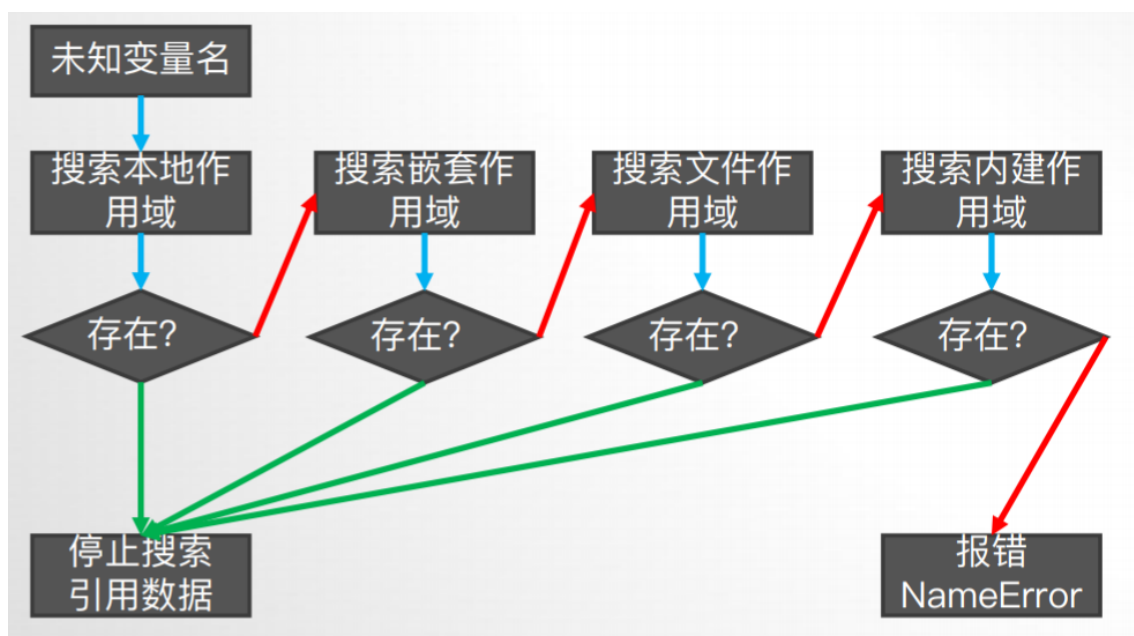
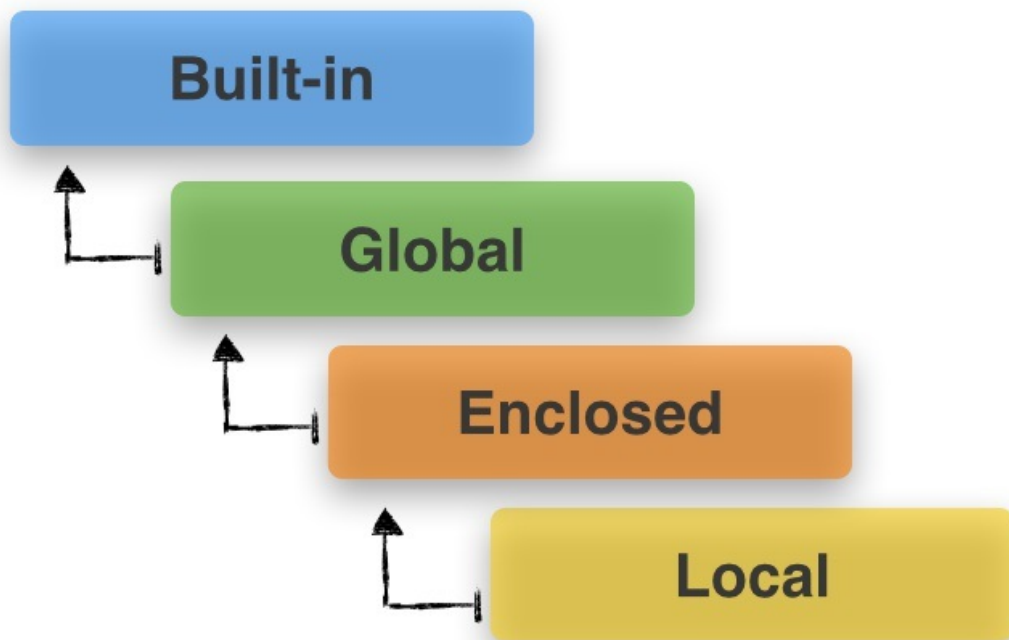
```
if __name__ == '__main__':
```

```
    getName(url)
```

全局作用域

作用域/变量名解析

- Python解释器检测到未知变量名时启动【变量解析】
- 变量解析用于检测变量名是否有效并找到正确的引用对象
- 变量名解析遵循LEGB原则



全局变量和Global

- 1、位于模块内部顶层的变量默认为全局变量
- 2、函数内部对全局变量赋值需使用global关键字
- 3、全局变量名在函数内部可以直接使用

```
1 x = 100
2 def demo():
3     x = 30
4     return x
5 print(demo())
6 print(x)
7
8 # -----分界线-----
9
10 x = 100
```

```
11 def demo():
12     # 声明全局属性
13     global x
14     x = 30
15     return x
16 print(demo())
```

全局变量：

- 全局变量是位于模块文件内部的顶层的变量名
- 全局变量如果是在函数内部被**赋值**的话，**必须经过声明**
- 全局变量名在函数的内部不经过声明也可以被**引用**
- 全局变量具备全局作用域

Global语句

- 在**函数内部**通过global修饰的变量为全局变量
- Global语句的作用：
 - 改变变量属性，将本地变量改为全局变量
 - 在函数内部，如果想要对全局变量进行重新赋值，需使用Global 进行声明