# Python函数、参数及参数解构


## 讲师：Wayne

从业十余载，漫漫求知路

# 函数

- 函数
  - 数学定义：y=f(x) ，y是x的函数，x是自变量。y=f(x0, x1, ..., xn)
  - Python函数
    - 由若干语句组成的语句块、函数名称、参数列表构成，它是组织代码的最小单元
    - 完成一定的功能
- 函数的作用
  - 结构化编程对代码的最基本的封装，一般按照功能组织一段代码
  - 封装的目的为了复用，减少冗余代码
  - 代码更加简洁美观、可读易懂
- 函数的分类
  - 内建函数，如max()、reversed()等
  - 库函数，如math.ceil()等

# 函数定义、调用

- def语句定义函数

def 函数名(参数列表):

    函数体（代码块）

    [return 返回值]

  - 函数名就是标识符，命名要求一样
  - 语句块必须缩进，约定4个空格
  - Python的函数没有return语句，隐式会返回一个None值
  - 定义中的参数列表称为形式参数，只是一种符号表达（标识符），简称**形参**
- 调用
  - 函数**定义**，只是**声明**了一个函数，它不会被执行，需要**调用**
  - 调用的方式，就是函数名后加上小括号，如有必要在括号内写上参数
  - 调用时写的参数是实际参数，是实实在在传入的值，简称**实参**

# 函数定义、调用

- 函数举例

```
def add(x, y):
    result = x+y
    return result
out = add(4,5)
print(out)
```

  - 上面只是一个函数的定义，有一个函数叫做add，接受2个参数
  - 计算的结果，通过返回值返回，需要return语句
  - 调用通过函数名add加2个参数，返回值可使用变量接收。函数名也是标识符，返回值也是值
  - 定义需要在调用前，也就是说调用时，已经被定义过了，否则抛NameError异常
  - 函数是**可调用的对象**，callable()
  - 看看这个函数是不是通用的？体会一下函数的好处

# 函数参数 传参

- 参数调用时传入的参数要和定义的个数相匹配（可变参数例外）
- 位置传参
  - def f(x, y, z) 调用使用 f(1, 3, 5)
  - 按照参数定义顺序传入实参
- 关键字传参
  - def f(x, y, z) 调用使用 f(x=1, y=3, z=5)
  - 使用形参的名字来传入实参的方式，如果使用了形参名字，那么传参顺序就可和定义顺序不同
- 传参
  - f(z=None, y=10, x=[1])
  - f((1,), z=6, y=4.1)
  - f(y=5, z=6, 2) #
  - **要求位置参数必须在关键字参数之前传入**，位置参数是按位置对应的

# 函数参数默认值

- 参数默认值（缺省值）
  - 定义时，在形参后跟上一个值

  def add(x=4, y=5):

  return x+y

  测试调用 add(6, 10) 、add(6, y=7) 、add(x=5) 、add()、add(y=7)、 add(x=5, 6) 、add(y=8, 4)、add(x=5, y=6)、add(y=5, x=6)

  测试定义后面这样的函数 def add(x=4,y)
- 作用
  - 参数的默认值可以在未传入足够的实参的时候，对没有给定的参数赋值为默认值
  - 参数非常多的时候，并不需要用户每次都输入所有的参数，简化函数调用
- 举例
  - 定义一个函数login，参数名称为host、port、username、password

# 函数参数默认值

- 举例
  - 定义一个函数login，参数名称为host、port、username、password

```
def login(host='127.0.0.1',port='8080',username='wayne',password='magedu'):
    print('{}:{}@{}/{}'.format(host, port, username, password))
login()
login('127.0.0.1', 80, 'tom', 'tom')
login('127.0.0.1', username='root')
login('localhost', port=80,password='com')
login(port=80, password='magedu', host='www')
```

# 可变参数

- 问题
  - 有多个数，需要累加求和

```
def add(nums):
    sum = 0
    for x in nums:
        sum += x
    return sum
```

  add([1,3,5])、add((2,4,6))

  传入一个可迭代对象，迭代元素求和
- 可变参数
  - 一个形参可以匹配任意个参数

# 可变参数

- 位置参数的可变参数
  - 有多个数，需要累加求和

```
def add(*nums):
    sum = 0
    print(type(nums))
    for x in nums:
        sum += x
    return sum
add(3, 6, 9) # 调用
```

- 在形参前使用*表示该形参是可变参数，可以接收多个实参
- 收集多个实参为一个tuple
- 思考：关键字参数能否也能传递任意多个吗？

# 可变参数

- 关键字参数的可变参数
  - 配置信息打印

```
def showconfig(**kwargs):
    for k,v in kwargs.items():
        print('{}={}'.format(k,v), end=', ')


showconfig(host='127.0.0.1', port=8080, username='wayne', password='magedu')
```

- 形参前使用**符号，表示可以接收多个关键字参数
- 收集的实参名称和值组成一个字典

# 可变参数

- 可变参数混合使用
  - 配置信息打印

  def showconfig(username, password, **kwargs)

  def showconfig(username, *args, **kwargs)

  def showconfig(username, password, **kwargs, *args) # ?

# 可变参数

- 总结
  - 有位置可变参数和关键字可变参数
  - 位置可变参数在形参前使用一个星号*
  - 关键字可变参数在形参前使用两个星号**
  - 位置可变参数和关键字可变参数都可以收集若干个实参，位置可变参数收集形成一个tuple，关键字可变参数收集形成一个dict
  - 混合使用参数的时候，普通参数需要放到参数列表前面，可变参数要放到参数列表的后面，位置可变参数需要在关键字可变参数之前

# 可变参数

- 举例

```
def fn(x, y, *args, **kwargs):
    print(x)
    print(y)
    print(args)
    print(kwargs)
fn(3,5,7,9,10,a=1,b='python')
fn(3,5)
fn(3,5,7)
fn(3,5,a=1,b='python')
fn(x=3, y=8, 7, 9, a=1, b='python') #
fn(7,9,y=5,x=3,a=1,b='python') # 错误，7和9分别赋给了x，y，又y=5、x=3，重复了
```

# 可变参数

- 举例

```
def fn(*args, x, y, **kwargs):
    print(x)
    print(y)
    print(args)
    print(kwargs)


fn(3,5) #
fn(3,5,7) #
fn(3,5,a=1,b='python') #
fn(3, 4, y=6, x=5, a=1, b='python')
```

# keyword-only参数

- keyword-only参数（Python3加入）
  - 如果在一个星号参数后，或者一个位置可变参数后，出现的普通参数，实际上已经不是普通的参数了，而是keyword-only参数

```
def fn(*args, x):
    print(x)
    print(args)


fn(3,5) #
fn(3,5,7) #
fn(3,5,x=7)
```

args可以看做已经截获了所有的位置参数，x不使用关键字参数就不可能拿到实参

思考：def fn(**kwargs, x) 可以吗？

# keyword-only参数

- 举例

  def(**kwargs, x):

  print(x)

  print(kwargs)

  直接报语法错误

  可以理解为kwargs会截获所有的关键字参数，就算你写了x=5，x也永远得不到这个值，所以语法错误

# keyword-only参数

- keyword-only 参数另一种形式

```
def fn(*, x,y):
    print(x,y)
fn(x=5,y=6)
```

*号之后，普通形参都变成了必须给出的keyword-only 参数

# 可变参数和参数默认值

- 举例

```
def fn(*args, x=5):
    print(x)
    print(args)
fn() # 等价于fn(x=5)
fn(5)
fn(x=6)
fn(1,2,3,x=10)
```

# 可变参数和参数默认值

- 举例

```
def fn(y, *args, x=5):
    print('x={}, y={}'.format(x, y))
    print(args)
fn() #
fn(5)
fn(x=6)  #
fn(1,2,3,x=10)
fn(y=17,2,3,x=10)  #
fn(1,2,y=3,x=10)  #
x 是 keyword-only参数
```

# 可变参数和参数默认值

- 举例

```
def fn(x=5, **kwargs):
    print('x={}'.format(x))
    print(kwargs)
fn()
fn(5)
fn(x=6)
fn(y=3,x=10)
fn(3,y=10)
```

# 函数参数

- 参数规则
  - 参数列表参数一般顺序是，普通参数、缺省参数、可变位置参数、keyword-only参数（可带缺省值）、可变关键字参数

```
def fn(x, y, z=3, *arg, m=4, n, **kwargs):
    print(x,y,z,m,n)
    print(args)
    print(kwargs)
```

- 注意
  - 代码应该易读易懂，而不是为难别人
  - 请按照书写习惯定义函数参数

# 函数参数

- 参数规则举例
  - 参数列表参数一般顺序是，普通参数、缺省参数、可变位置参数、keyword-only参数（可带缺省值）、可变关键字参数

```
def connect(host='localhost', port='3306', user='admin', password='admin', **kwargs):
    print(host, port)
    print(user, password)
    print(kwargs)
connect(db='cmdb')
connect(host='192.168.1.123', db='cmdb')
connect(host='192.168.1.123', db='cmdb', password='mysql')
```

# 参数解构

- 举例
  - 加法函数

```
def add(x, y):
    return x+y
add(4, 5)
add((4,5))
t = (4, 5)
add(t[0], t[1])
add(*t) 或 add(*(4,5))          add(*[4,5])          add(*{4,6})
add(*range(1,3))
```

# 参数解构

- 参数解构
  - 给函数提供实参的时候，可以在集合类型前使用*或者**，把集合类型的结构解开，提取出所有元素作为函数的实参
  - 非字典类型使用*解构成位置参数
  - 字典类型使用**解构成关键字参数
  - 提取出来的元素数目要和参数的要求匹配，也要和参数的类型匹配

```
def add(x, y):
    return x+y
add(*(4,5))                add(*[4,5])          add(*{4,6})
d = {'x': 5, 'y': 6}
add(**d)
add(**{'a': 5, 'b': 6})                add(*{'a': 5, 'b': 6})
```

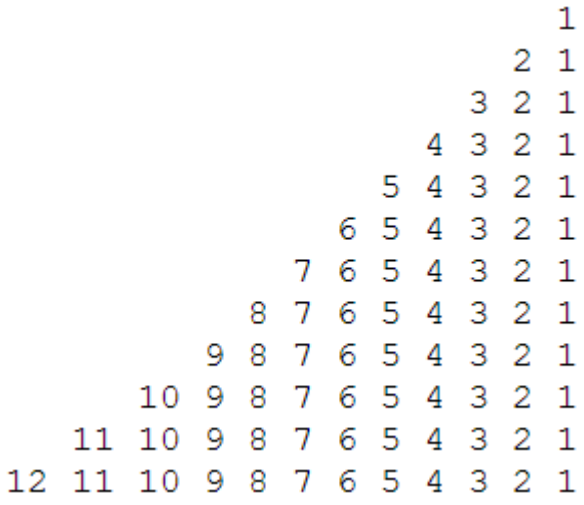
# 参数解构

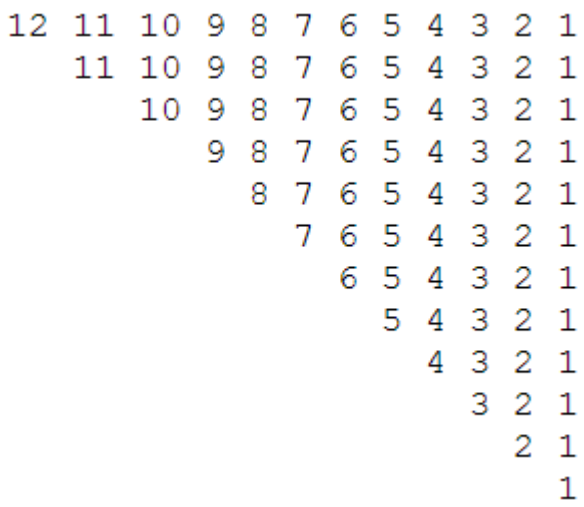
- 参数解构和可变参数
  - 给函数提供实参的时候，可以在集合类型前使用*或者**，把集合类型的结构解开，提取出所有元素作为函数的实参

```
def add(*iterable):
    result = 0
    for x in iterable:
        result += x
    return result
add(1,2,3)
add(*[1,2,3])
add(*range(10))
```

# 练习

- 编写一个函数，能够接受至少2个参数，返回最小值和最大值
- 编写一个函数，接受一个参数n，n为正整数，左右两种打印方式。要求数字必须对齐



# 谢谢

咨询热线 400-080-6560