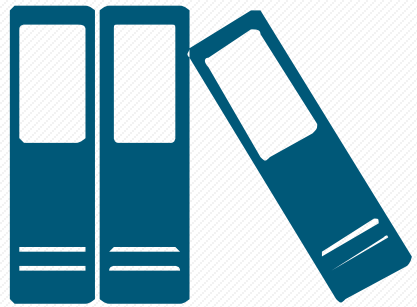
第4章 函数



什么叫函数?

为什么要使用函数?

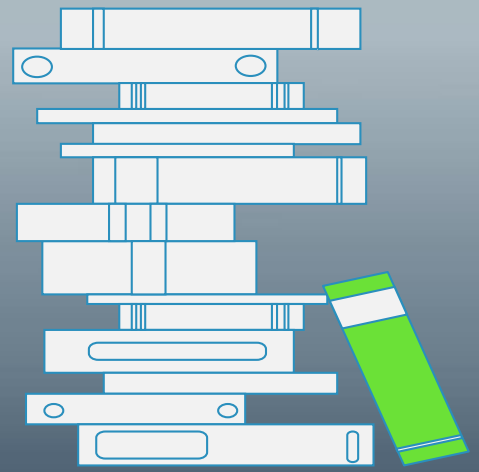
如何定义函数?

如何使用函数?

参数传递机制?

01

函数的基本概念



初见函数

函数及使用	描述
input([msg])	输入，msg为提示信息，返回一个字符串
len(s)	求字符串的长度
abs(x)	绝对值，x的绝对值 abs(-10.01) 结果为 10.01
divmod(x,y)	商余，(x//y, x%y)，同时输出商和余数 divmod(10, 3) 结果为 (3, 1)
pow(x, y[, z])	幂余，(x**y)%z，[..]表示参数z可省略 pow(3, pow(3, 99), 10000) 结果为 4587

已经见识了很多函数!

函数分为以下4类：

- 内置函数：系统提供的函数，直接使用
- 标准库函数。
- 第三方库函数。
- 用户自定义的函数：为满足特定要求，可以自己定义函数

如何写自己的函数?

求平方和

10~20的平方和
20~30的平方和
30~50的平方和

```
sum = 0
for i in range(10, 20+1):
    sum += i*i
print(sum)

sum = 0
for i in range(20, 30+1):
    sum += i*i
print(sum)

sum = 0
for i in range(30, 50+1):
    sum += i*i
print(sum)
```

求平方和

三段几乎一模一样的代码

代码复制是程序代码质量不良的表现

```
sum = 0
for i in range(10, 20+1):
    sum += i*i
print(sum)
```

```
sum = 0
for i in range(20, 30+1):
    sum += i*i
print(sum)
```

```
sum = 0
for i in range(30, 50+1):
    sum += i*i
print(sum)
```

```
def sqsum(start, end):
    s = 0
    for i in range(start, end+1):
        s += i * i
    print(s)
```

```
sqsum(10, 20)
sqsum(20, 30)
sqsum(30, 50)
```

可以将这段重复的代码抽出来，定义为一个函数，起名为sqsum，这样，我们就可以在任何需要的地方使用这个名字来运行这段语句块。

实现代码重用

为什么要用函数?

函数用来将复杂的问题分解为若干子问题，并逐个解决。

两个作用:

1) 代码复用

2) 降低编程难度：分而治之（Divide and Conquer）

- 把大而复杂的问题分解成小问题。
- 有助于提升代码的整洁度，使代码更易于理解，有利于程序的维护。
- 可继承前人的劳动成果，简化编程。

什么叫函数?

函数是一块代码，

接收0个或多个参数，

做一件事情，

并返回一个或多个值

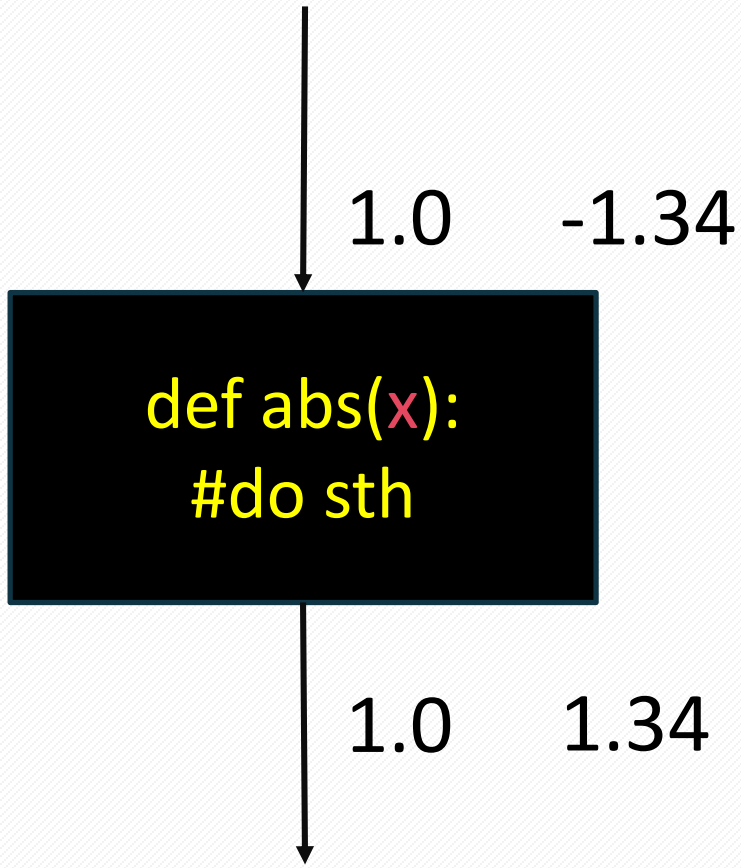
允许你给它命名一个名字。比如想象成数学函数：y = isprime(x)
这样，你可以在你的程序的任何地方使用这个名字来运行这个语句块。

了解：

函数名

函数的参数——输入

函数的返回值——计算结果，输出



例如：abs函数

功能：对一个数据进行取绝对值操作。

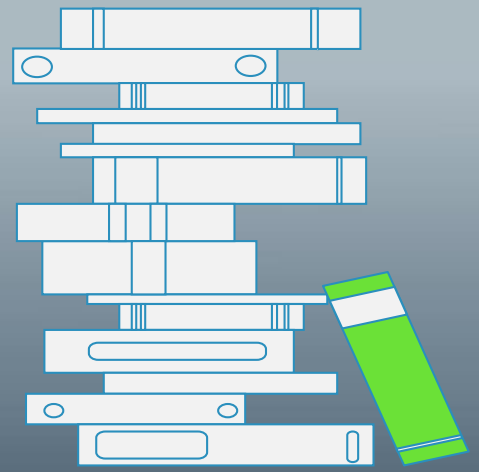
定义：`abs(x)`，我们可知：

- 函数名：abs。
- 形式参数（形参）：1个，x
- 函数返回值：x的绝对值。

```
>>> abs(1.0)    # 1.0为实际给出的参数
1.0
>>> abs(-1.34)   #-1.34为实参
1.34
```

02

函数的定义和使用



函数的定义

——函数在使用前必须先定义

```
sqsum(10, 20)
sqsum(20, 30)
sqsum(30, 50)
```

NameError: name ' sqsum' is not defined

```
def sqsum(start, end):
    s = 0
    for i in range(start, end+1):
        s += i * i
    print(s)
```

```
def sqsum(start, end):
    s = 0
    for i in range(start, end+1):
        s += i * i
    print(s)
```

定义函数

正确，定义放在调用之前

```
sqsum(10, 20)
sqsum(20, 30)
sqsum(30, 50)
```

调用函数

定义函数

在Python中，**定义函数**的语法如下：

```
def 函数名（[参数列表]）：
    函数体
    return [返回值]
```

1) 函数使用 *def* 进行定义，后接函数名和圆括号 ()

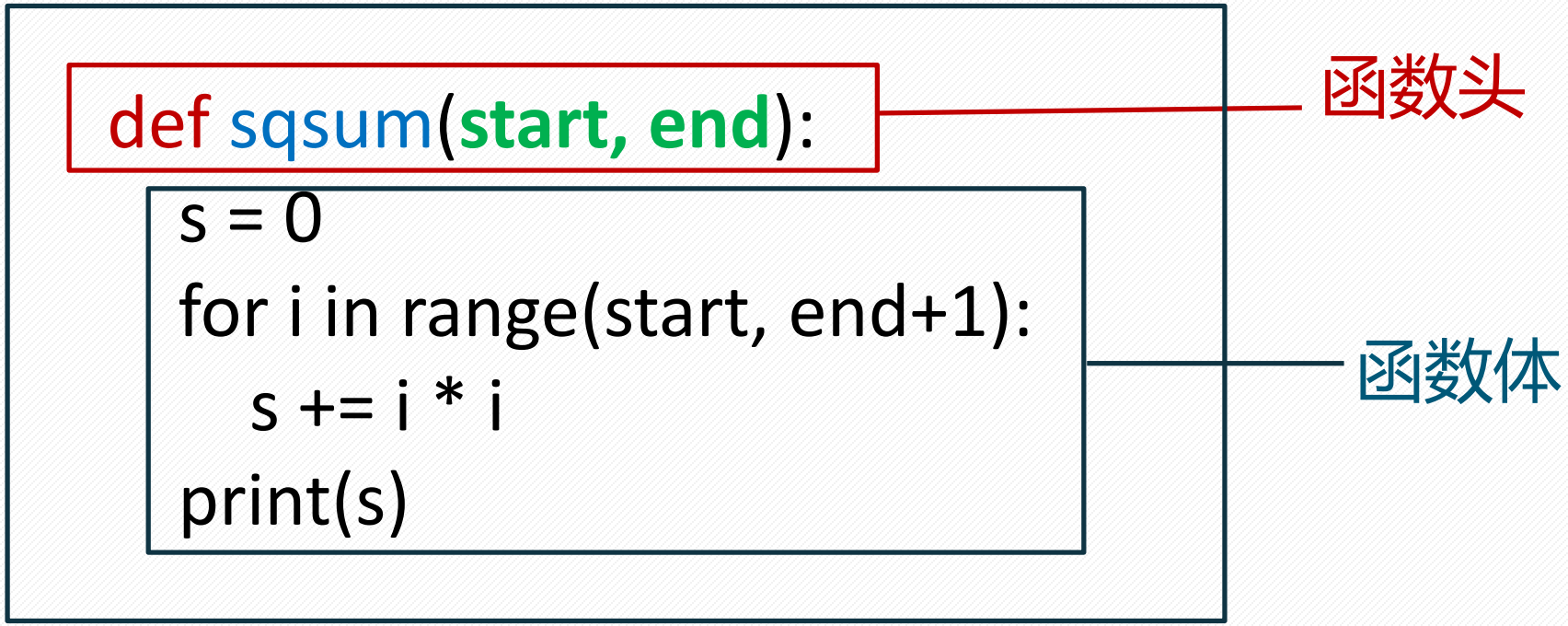
2) 圆括号内是形参列表，如果有多个参数则使用逗号分隔开，即使该函数不需要接收任何参数，也必须保留一对空的圆括号。

3) 圆括号后的“:”必不可少。

4) 函数体相对于def关键字必须保持一定的空格缩进。

5) 函数体中可以使用return语句返回值。return语句可以多条，在这种情况下，一旦第一条return得到执行，函数立即终止。return可以出现在函数体的任何位置。

函数名　　形参列表


def sqsum(start, end):
s = 0
for i in range(start, end+1):
s += i * i
print(s)
函数头
函数体

```
def greeting():       #无参函数
    print(“hello world”)
    return
```

```
def triangleArea(a, b, c): #有参函数
    area  = 0
    if a+b>c and a+c>b and b+c>a:
        s = (a+b+c)/2
        area = pow((s*(s-a)*(s-b)*(s-c)), 0.5)
    return area
```

单选题 1分

设置

此题未设置答案，请点击右侧设置按钮

定义一个函数时，形参可以是一个，也可以是多个，但是不能没有

A 正确

B 错误

提交

函数的返回值

函数的返回值

函数可以返回0个或多个结果

- *return*保留字用来传递返回值
- *return*可以传递0个返回值，也可以传递任意多个返回值

使用形式：**return 表达式**

作用：用于函数体中，将表达式的值返回到调用处。

- 执行return语句，意味着函数执行终止
- 如果return后面没有表达式，返回值为None。如果函数没有return语句，函数返回值也为None；

（None是Python中一个特殊的值，虽然它不表示任何数据，但仍然具有重要的作用。Greeting()函数的返回值为None.）

- 一个函数中可以有多个return语句，只会执行其中的一个。
- 可以返回多个值，以元组、列表形式返回。

```
def triangleArea(a, b, c): #有参函数
    area  = 0
    if a+b>c and a+c>b and b+c>a:
        s = (a+b+c)/2
        area = pow((s*(s-a)*(s-b)*(s-c)), 0.5)
    return area
```

#返回一个值area

```
def fun(x, y):
    if x < y:
        x, y = y, x
    return x, y
```

#返回多个值将以元组形式返回,即(x,y)
a, b = fun(x,y)

```
def isprime(x) :
    for i in range(2, x):
        if x % i == 0:
            return False
    return True
```

#有多个return语句

```
def greeting():       #无参函数
    print(“hello world”)
```

#无return语句，返回None

单选题 1分

设置

此题未设置答案，请点击右侧设置按钮

下面说法正确的是

A 函数中没有return语句，表明函数没有返回值

B 函数中只能有一个return语句

C 函数中可以有多个return语句，每个return语句会顺序依次执行

D 函数中可以有多个return语句，但是只能执行其中的一个return语句

提交

单选题 1分

设置

此题未设置答案，请点击右侧设置按钮

函数定义后，只能被调用一次。

A 正确

B 错误

提交

函数的调用

调用形式
调用过程
参数调用机制

```
def isprime(x) :
    for i in range(2, x):
        if x % i == 0:
            return False
    return True
```

#有多个return语句

函数必须“先”定义，再使用（调用）！

函数定义后，不经过调用，不会被执行!

在主程序中的调用形式可以为：

作为print函数的参数：

```
print(isprime(x+2))
```

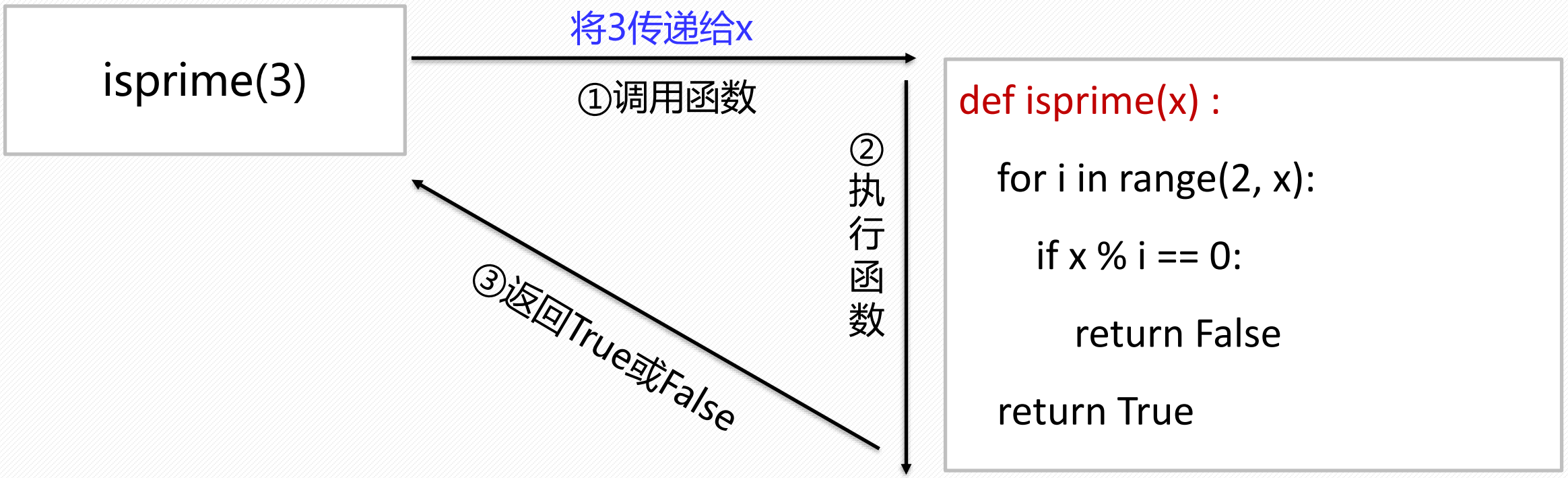
直接赋给变量：

```
s = isprime(5)
```

构成表达式：

```
if ( isprime(x)):
    #do sth.
```

函数调用过程


isprime(3)
将3传递给x
①调用函数
②执行函数
③返回True或False
def isprime(x) :
for i in range(2, x):
if x % i == 0:
return False
return True

函数参数传递机制

1. 实参：
   - 是在程序运行时，实际传递给函数的数据。实参列表必须与函数定义时的形参列表一一对应（个数、类型）。参数默认按位置传递，即按参数顺序，第一个实参的值传递给第一个形参，第二个实参的值传递给第二个形参，以此类推。
2. 形参：
   - 调用前，形参不存在。
   - 调用时，形参得到实参传递过来的值。
   - 调用结束后，形参被回收。

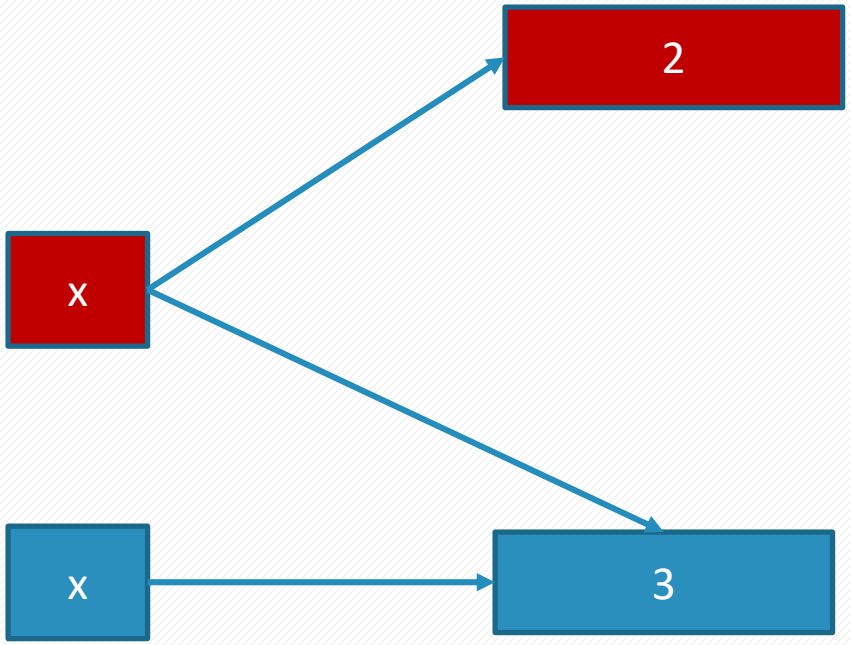
函数参数传递机制

- 由实参 向 形参的单向、值传递
- 形参：在函数调用时获得实参传递过来的值
- 当传递的是不可变类型时，形参的修改不会影响到实参
- 当传递的是可变类型数据（字典，列表）时，由于传递的是引用，因此可能会修改实参的值！！

参数传递：单向值传递

```
def  fun(x):
    x = 2
```

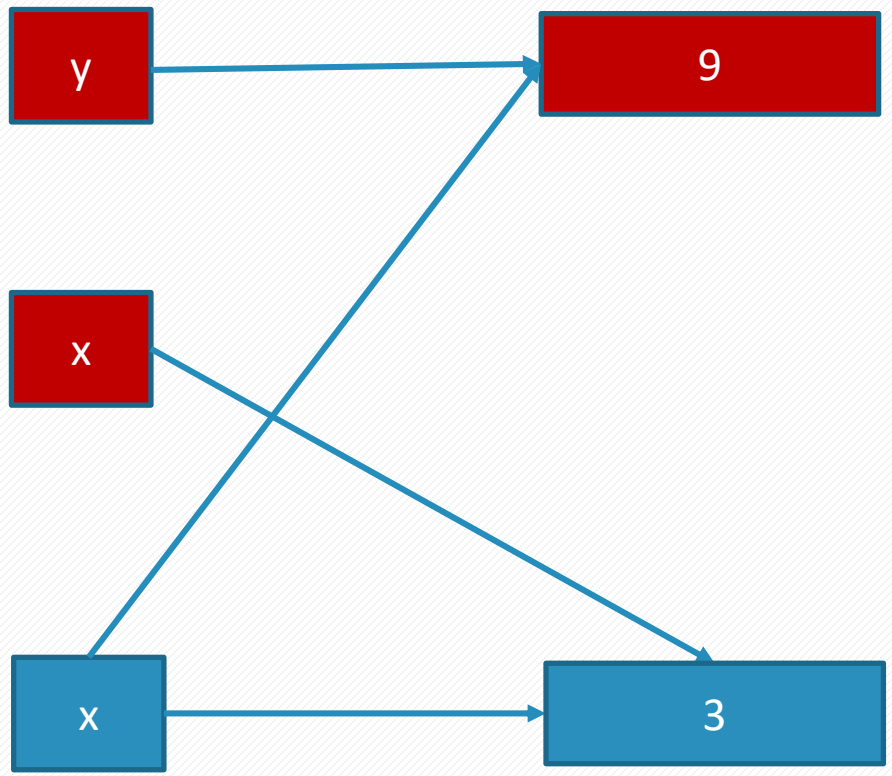
```
#主程序
x = 3
fun(x)
print(x) #?
```



理解参数传递机制

```
def  fun(x):
    y = x ** 2
    return y
```

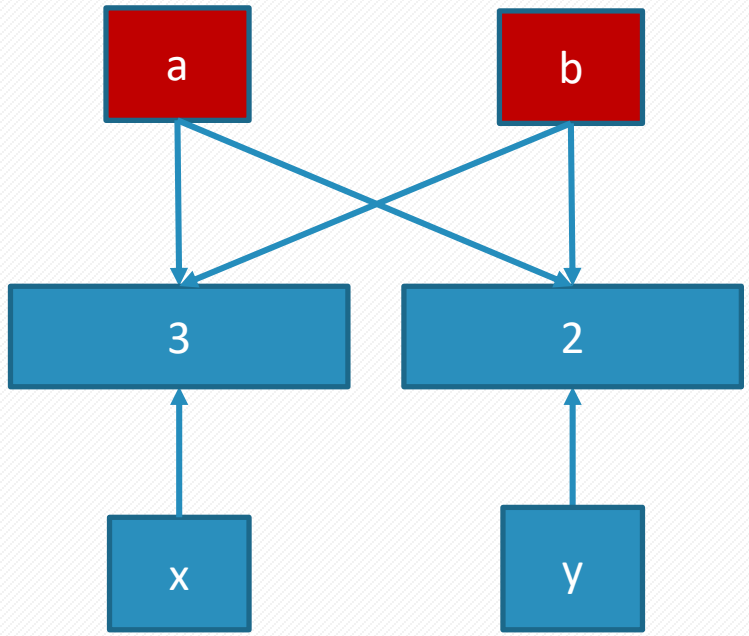
```
x = 3
x = fun(x)
print(x)
```



交换两个变量1

```
def swap(a, b):
    a, b = b, a

x, y = 3, 2
swap(x, y)
print(x, y)
```



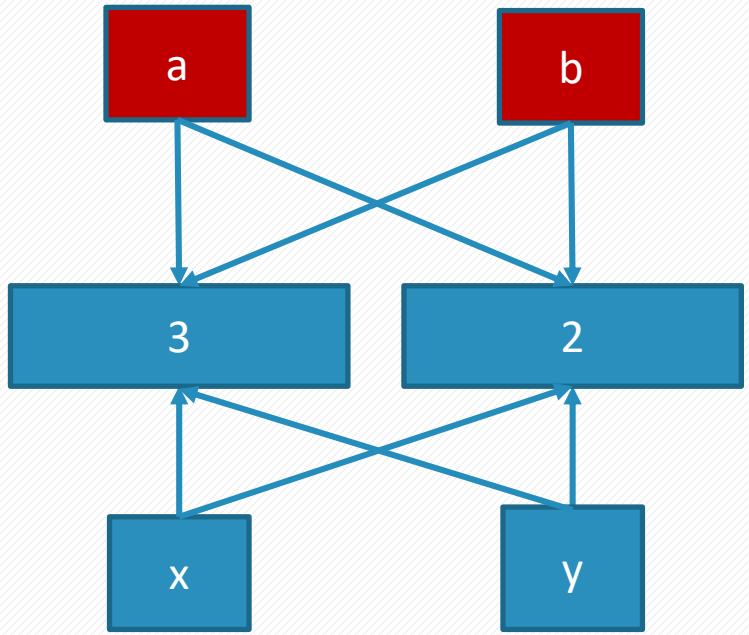
当传递的是不可变类型数据时，形参的修改不会影响到实参

整形、浮点型、字符串类型和布尔类型

交换两个变量2

```
def swap(a, b):
    a, b = b, a
    return a, b
```

元组：(a, b)




```
x, y = 3, 2
x, y = swap(x, y)
print(x, y)
```

单选题 1分

设置

此题未设置答案，请点击右侧设置按钮

依次执行语句

```
def modify(x):
    x=5
```

```
x = 3
modify(x)
```

之后，x的值为

A 3

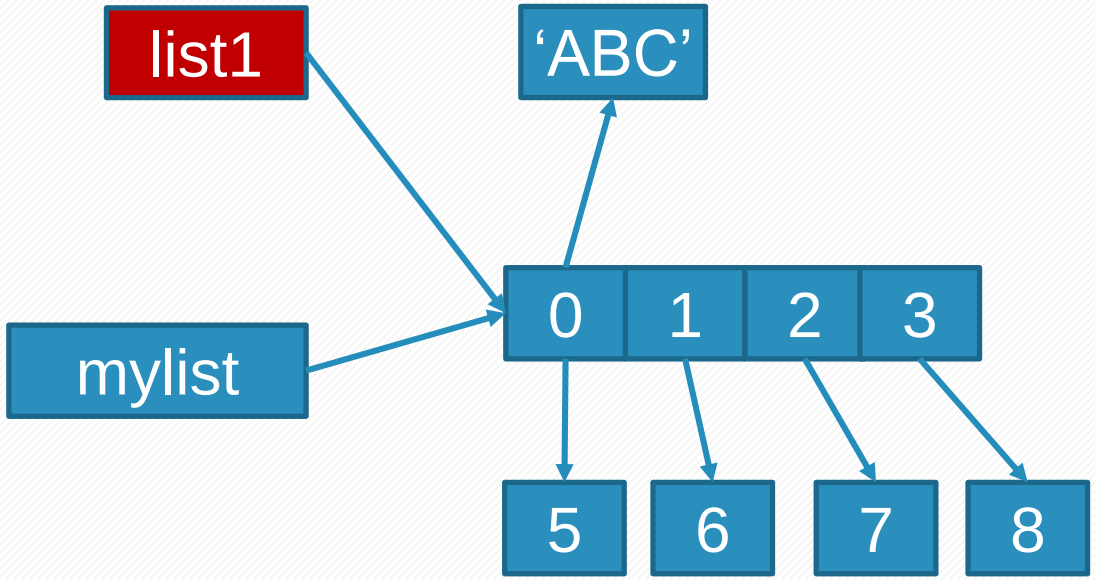
B 5

提交

列表作为函数参数

```
def fun(list1):
    list1[0] = "ABC"

mylist = [5,6,7,8]
fun(mylist)
print(mylist)
```



list1 = mylist

当传递的是可变类型数据时，形参的修改会影响到实参

列表、字典、集合

单选题 1分

设置

此题未设置答案，请点击右侧设置按钮

```
def increment(lst):
    for i in range(len(lst)):
        lst[i] += 1

mylist = [1,2,3,4]
increment(mylist)
print(mylist)
```

A [1，2，3，4]

B [2，3，4，5] √

提交

单选题 1分

设置

此题未设置答案，请点击右侧设置按钮

```
def increment(lst):
    for item in lst:
        item += 1

mylist = [1,2,3,4]
increment(mylist)
print(mylist)
```

A √ [1，2，3，4]

B [2，3，4，5]

提交

函数的编写和使用

问题：编写函数，求任意个连续整数的和

```
def calSum():
    sum = 0
    for i in range(1, 10+1):
        sum += i
    print("sum=", sum)

calSum()
```

Version1:

函数没有参数，只能求出1至100的整数和。

调用函数calSum()，没return，使用单条语句调用

函数名（）

函数的编写和使用

问题：编写函数，求任意个连续整数的和

```
def calSum(n1,n2):
    sum = 0
    for i in range(n1, n2+1):
        sum += i
    print("sum=", sum)

m1 = int(input("初值: "))
m2 = int(input("终值: "))
calSum(m1, m2)
```

```
初值: 10
终值: 20
sum= 165
```

Version2:

根据用户的输入来确定求和的范围

函数名（实参列表）

- **调用时要给出实参**
- **实参的值传递给形参**

函数的编写和使用

问题：求(2+3+……+19+20)+(11+12+……+99+100)的和

```
def calSum(n1,n2):
    sum = 0
    for i in range(n1, n2+1):
        sum += i
    return sum                    #函数调用后得到返回值


print("sum=", calSum(2, 20) + calSum(11, 100))
```

Version3:

一个函数只做一件事

在表达式中调用函数calSum()，或者直接输出

小结：

函数先定义再调用

函数不调用，不执行

函数的形参是函数的“输入”(I)

函数体是函数的“处理”(P)

函数的返回值是函数的“输出”(O)

函数的参数类型

默认值参数、
按名称传递参数
可变数量参数

默认值参数

在声明函数时，如果希望函数的一些参数是可选的，可以在声明函数时为这些参数指定默认值。调用该函数时，如果没有传入对应的实参值，则函数使用声明时指定的默认参数值。

```
>>> def babble(words,times=1):
        print((words+"   ")*times)


>>> babble('hello',3)
hello   hello   hello
>>> babble('Tiger')
Tiger
```

默认值参数必须写在形参列表的右边

【默认值参数示例】：基于期中成绩和期末成绩，按指定的权重计算总评成绩。

```
def mySum1(mid_score,end_score, rate = 0.4):  # 默认权重为0.4
    score = mid_score * rate + end_score * (1 - rate)
    return score

print("总评成绩: {:.2f}".format(mySum1(88, 93)))        # 权重0.4
print("总评成绩: {:.2f}".format(mySum1(88, 93, 0.5)))  # 权重0.5
print("总评成绩: {:.2f}".format(mySum1(62, 78, 0.6)))  # 权重0.6
```

名称传递参数（关键字参数）

1. 函数调用时，通常实参默认按照位置顺序传递参数，按照位置传递的参数称为位置参数。函数调用时，也可以通过名称（关键字）指定传入的参数，按照名称指定传入的参数称为名称参数，也称为关键字参数。

2. 使用名称参数具有三个优点：参数按名称意义明确；传递的参数与顺序无关；如果有多个可选参数，则可以选择指定某个参数值。

【名称传递参数示例】基于期中成绩和期末成绩，按指定的权重计算总评成绩。

```
def mySum1(mid_score,end_score, rate = 0.4):
    score = mid_score * rate + end_score * (1 - rate)
    return score
                基于期中成绩和期末成绩，按指定的权重计算总评成绩。
print(mySum1(88, 93))
print(mySum1(mid_score = 88, end_score = 93, rate = 0.5))
print(mySum1(rate = 0.5, end_score = 93, mid_score = 88))
```

【名称传递参数示例】

再看一个在print中使用名称传递参数的例子。假设我们希望不同输出项之间用“-”分隔，可以采用调用形式print(……, sep= ‘-’)；假设不希望在一次print之后换行，可以写print(……, end= ‘’)。

```
>>> print(1,2,3,sep='-')
1-2-3
>>> print(23,5,34,sep='/')
23/5/34
>>> for i in range(1,4):
        print(i,end='')


123
```

```
>>> for i in range(1, 10):
                print(i,end="," if i%3!=0 else "\n")



1,2,3
4,5,6
7,8,9
```

名称传递参数的特点：

1. 函数调用时，按照名称（关键字）指定传入的参数，可以采用与函数定义时不同的顺序。
2. 使用关键字参数具有三个优点：
   - 参数意义明确；
   - 传递的参数与顺序无关；
   - 如果有多个可选参数，则可以选择指定某个参数值

单选题 1分

设置

```
def  fun(x, y):
    return x ** y

z = fun(y = 3, x = 2)
print(z)
```

A 9

B 8

√

提交

函数的参数类型3

可变数量参数

- 当函数参数数目不确定时，可使用可变数量参数。
- 可数量变参数主要有两种形式：在形式参数名前加1个*或2个**
  - 1个星号*：在定义函数的时候，通过一个带星号的参数，允许向函数传递可变数量的参数。调用函数时，从该参数之后所有的参数都被收集为一个元组。
  - 2个星号**：在定义函数的时候，也可以通过带两个星号的参数，允许向函数传递可变数量的参数。调用函数时，从该参数之后所有的参数都被收集为一个字典。

【可变数量参数示例】

```
def max(*s):  #s为可变数量参数
    maxx=s[0]
    for x in s[1:]:
        if maxx<x:
            maxx=x
    return maxx

z=max(1,2,3,8,4,5,-3,6)
print('最大值：',z)
```

调用时，s接收下列所有数据

(1,2,3,8,4,5,-3,6)形成元组

【可变数量参数示例】

```
def commonMultiple(*c):     # c为可变参数
    for i in c:
        print("{:^4}".format(i), end='')
    return len(c)

count = commonMultiple("李白", "杜甫")
print("共{}人".format(count))
count = commonMultiple("李白", "杜甫", "王维", "袁枚")
print("共{}人".format(count))
```

```
李白   杜甫  共2人
李白   杜甫   王维   袁枚  共4人
```

填空题 1分

设置

已知有函数定义

```
def demo(*p):
    return sum(p)
```

10

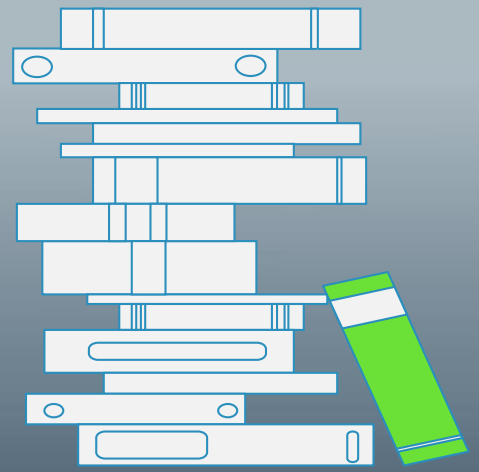
那么表达式 demo(1, 2, 3, 4) 的值为 [填空1]

正常使用填空题需3.0以上版本雨课堂

提交

03

lambda函数



lambda函数

lambda函数是一种简便的，在同一行定义函数的方法。lambda实际上是生成一个函数对象，即匿名函数，也就是没有函数名字的临时使用的小函数，它广泛用于需要函数对象作为参数或函数比较简单并且只使用一次的场合。

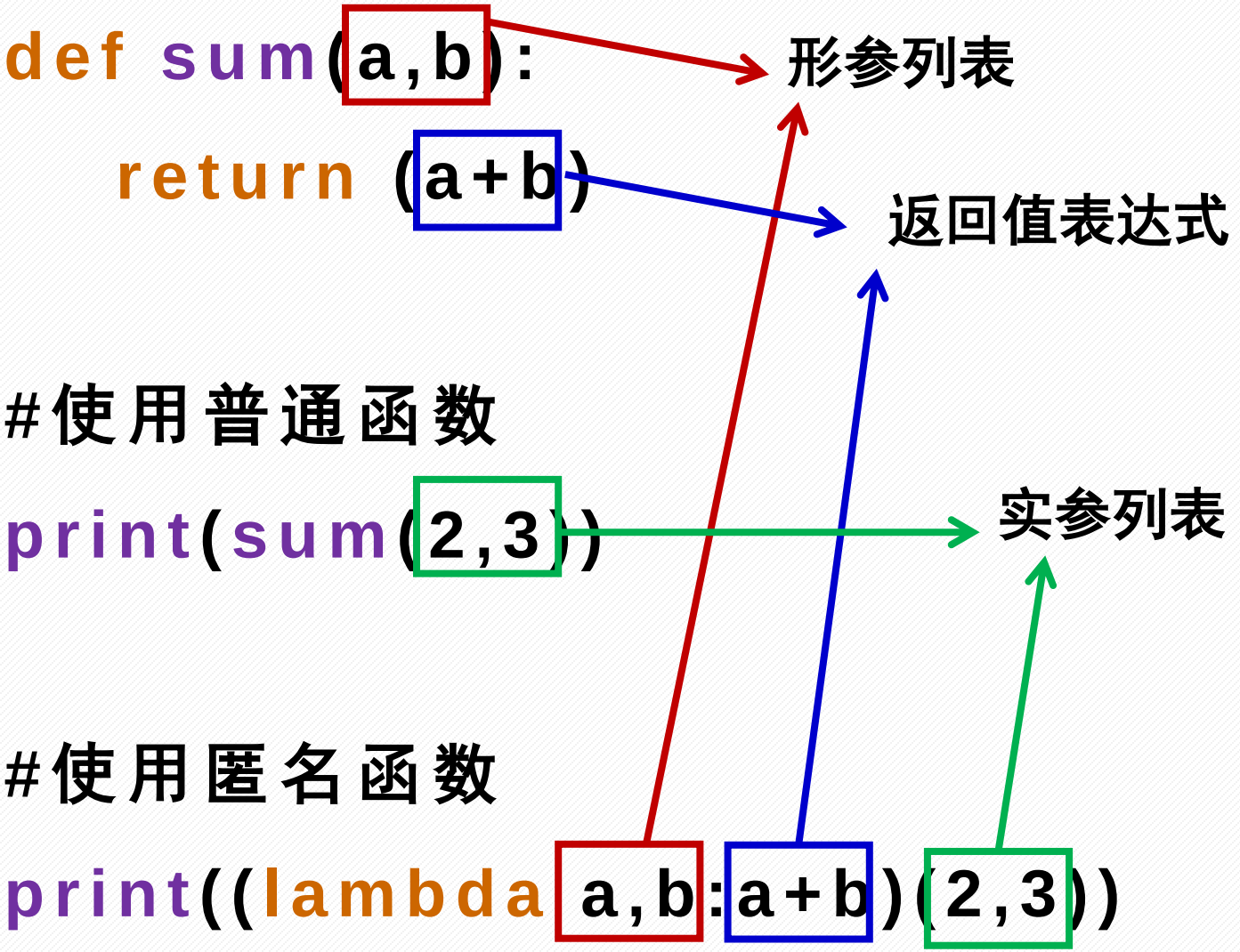
lambda函数的定义格式：

```
lambda 参数1，参数2，…… ： <函数语句>
```

输入是传入到参数列表里的值

函数语句只能是单行，其值是函数的返回值（输出），语句中出现的参数需要在参数列表中有定义。

lambda与普通函数对比



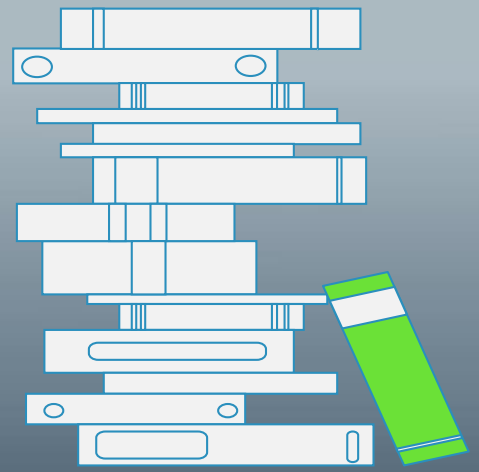
lambda 常见用法

- 将lambda函数赋值给一个变量，通过这个变量间接调用该lambda函数

```
>>>f = lambda x, y,z: x+y+z
>>>f(1,2,3)
6
>>> g = lambda x, y=2, z=3: x+y+z    #默认值参数
>>> g(1)
6
>>> g(2, z=4, y=5)                  #关键字参数
11
```

04

变量的作用域



变量的作用域

- **作用域**：就是变量可以在程序中被引用的范围。
- 一个程序的所有变量并不是在任何位置都可以访问的。访问权限决定于**这个变量是在哪里赋值的。**
- 每个变量都有自己的**作用域**（命名空间），变量的作用域决定了在某些代码段内使用该变量**是否合法。不同作用域内变量名可以相同**，互不影响。
- 两种最基本的变量作用域是“全局变量”和“局部变量”。

变量的作用域

局部变量

```
>>> def f():
        x = 10              # 局部变量
        return x*x

>>> f()
100
>>> print(x)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
```

变量x是局部变量，它的作用域在函数f()的内部。在函数f()外部访问x变量的时候，超出x的作用域范围，因此代码报错。

变量的作用域

全局变量

```
def f():
    return x*x                    # 返回100

x = 10                            # 全局变量
print(f())
print(x)
```

代码中只有一个x变量，它在函数之外赋值，是全局变量，全局可见。在函数f()内当然也可以访问这个全局的x，所以代码运行后会得到结果：

100

10

变量的作用域

全局变量

```
def f():
    x = 10                    # A 局部变量
    return x*x

x = 1000                      # B 全局变量
print(x)                      # C
```

A行定义的x作用域只在函数f()的内部，它是局部变量；

而B行定义的x作用域在函数f()的外部，它是全局变量，全局可见；

所以c行的print语句只能访问到等于1000的那个x，代码的输出结果应该是**1000**。

变量的作用域

全局变量和局部变量同名

```
def f():
    x = 5           # 局部变量
    print("f内部: x=", x)   # A
    return x*x

x = 10              # 全局变量
print("f()=", f())
print("f外部: x=", x)   # B
```

这里的“x=5”和“x=10”变量名称虽然一样，但作用域不同。前者是局部变量，后者是全局变量。

当执行A行print语句的时候，既能访问局部的x，也能访问全局的x，在这种情况下，采用局部优先的原则：在局部变量（包括形参）和全局变量同名的时候，局部变量屏蔽全局变量。因此A行

运行代码，结果如下：

f内部：x= 5

f()= 25

f外部：x= 10

变量的作用域

global声明将使用全局变量

```
def f():
    global x
    x = 5      # A    访问全局变量x
    print("f内部: x=",x)
    return x*x

x = 10
print("f()=",f())
print("f外部: x=",x)
```

在代码中，由于有“**global x**”这句话，A行访问的x为全局变量x，A行将全局变量x重新赋值为5，运行代码，结果为：

f内部：x= 5

f()= 25

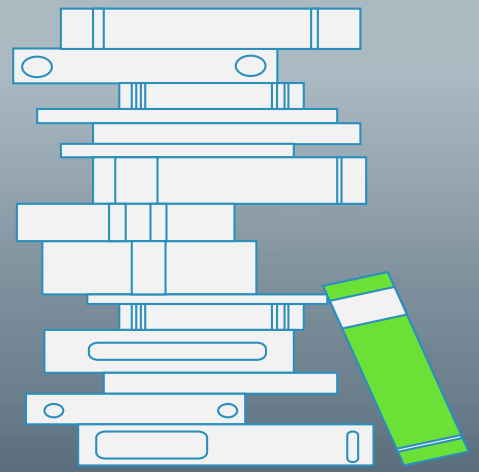
f外部：x= 5

局部变量和全局变量小结：

- 定义（首次赋值即定义！）
- 在函数内部定义的变量为局部变量，在函数外定义的变量为全局变量。
- 局部变量只能在其被声明的函数内部访问，而全局变量可以在整个程序范围内访问。
- 局部变量和全局变量重名，采用局部优先的原则：在局部变量（包括形参）和全局变量同名的时候，局部变量屏蔽全局变量
- 可以使用 global 保留字在函数内部声明全局变量。

05

递归函数（自学）



递归函数

Python中，一个函数既可以调用另一个函数，也可以调用它自己。如果一个函数调用了它们自己，就称为递归。

例如，非负整数的阶乘定义为：

$$n! = n \times (n-1) \times (n-2) \times \cdots\cdots \times 2 \times 1$$，当n=1时，n!=1

即n!是所有小于或等于n的正整数的乘积。显然，我们可以用循环结构来求n!，而另一种简便方法就是递归。我们将阶乘的定义转换为如下形式：

$$n! = 1 \quad n = 1$$
$$n! = n \times (n-1)! \quad n > 1$$

递归函数

递归方法求阶乘

```
def fact(n):
    if n == 1: return 1
    else: return n * fact(n-1)

for i in range(1, 9+1):
    print("{}! =".format(i),
```

递归终止条件

递归步骤

```
1! = 1
2! = 2
3! = 6
```

每个递归函数必须包括：

- 终止条件：表示递归的结束条件，用于返回函数值，不再递归调用。
- 递归步骤（递归链条）：递归步骤把第n步的函数与第n-1步的函数关联。

递归函数

递归方法求斐波那契数列前20项

```
def fibo(n):
    if n == 1 or n == 2: return 1
    else: return fibo(n-1) + fibo(n-2)

for i in range(1, 20+1):
    print("{:>8}".format(fibo(i)), end="   " if i%5!=0 else "\n")
```

```
       1        1        2        3        5
       8       13       21       34       55
      89      144      233      377      610
     987     1597     2584     4181     6765
```

递归方法求斐波那契数列前20项

```
def fibo(n):
    if n == 1 or n == 2:
        f=1
    else:
        f=fibo(n-1) + fibo(n-2)
    return f
```

递归链条

$$fibo(n)=\begin{cases}1 & n=1\\ 1 & n=2\\ fibo(n-1)+fibo(n-2) & n>2\end{cases}$$

```
for i in range(1, 20+1):
    print("{:>8}".format(fibo(i)), end="  " if i%5!=0 else "\n")
```

```
       1       1       2       3       5
       8      13      21      34      55
      89     144     233     377     610
     987    1597    2584    4181    6765
```

递归与循环的特点:

1. 递归和循环的本质都是**代码复用**

2. 递归的复用单位是**函数**，循环的复用单位是**语句**

3. **递归往往是自顶向下**（1←n），将问题的规模逐步缩小，直到缩小至递归结束条件成立（n == 1）

4. **循环既**可以是自顶向下（1 ← n），**也可以**是自底向上（1 → n），但是一般还是自底向上的比较多。

5. **一切递归问题都可以转化为循环求解**，反之不一定。

编写函数的几个原则

1. 函数设计要**尽量短小**，嵌套层次不宜过深
2. 函数声明应该做到合理、简单、易用
3. 函数参数设计应该考虑向下兼容
4. 一个函数尽量**只做一件事**
5. 一般不要在函数里使用input/print
6. **少用全局变量**传递参数
7. 每个函数中尽量使用**一个return**返回

形参列表形式：

- fun(a, b)
- fun(a=1, b=2)
- fun(*args)
- fun(**kwargs)
- fun(a, b=1, *args)
- fun()
- ……

返回值的形式：

- return 1
- return None
- return a + b
- return abs(a)
- return 1 if a >10 else 0
- return a,b,c 或 return (a,b,c)
- return [a,b,c]
- ……

本章总结

- 函数的定义和调用方法 *
- 参数及返回值的概念*
- 参数的类型*
- Lambda函数*
- 作用域
- 递归函数

带*的部分为重点内容

本周课程作业

① 阅读 第3章循环和第4章函数

掌握多重循环、break和continue语句，函数的定义，参数传递机制，

② 复习 课件和上课讲的实例

可在解释器中输入代码，运行输出结果

③ 上机课任务

实验4 范例5-6，p75编程题4.(4)-(6)
实验5 范例1-3
练习系统 lab.hebut.edu.cn函数模块（涉及到列表和元组的题目暂时不做）

④ 提交 python123中的作业（再练循环结构）

The End