第6章：函数

6.1 什么是函数

定义：为实现一个操作或特定功能而集合在一起的语句集。

好处：避免代码复制带来的错误或漏洞，不仅可以实现代码的复用，还可以保证代码的一致性。

6.2 定义函数

定义函数

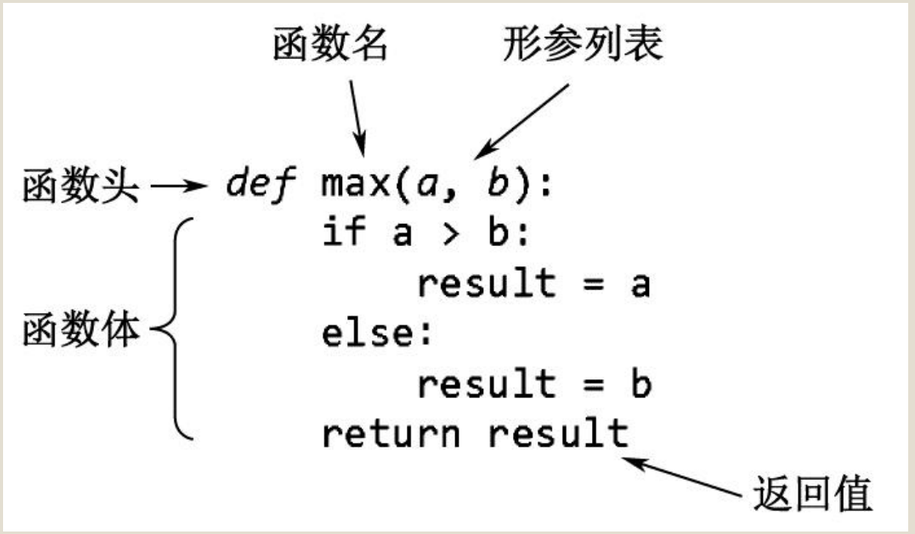
- 函数头
- 函数体

函数头

- 函数名
- 形参列表

函数体

- 缩进语句块
- 返回值（return）



即使函数没有参数，也要保留括号

没有return，则默认返回空值None

注意冒号，缩进和对齐三原则

6.3 调用函数

- 对于一个函数，可以通过函数名(参数列表)这样的语法来调用函数；
- 如果函数有返回值，则可以在函数调用的同时将返回值传递出来，此时这个函数调用可以当作一个值来处理；
- 函数也可以通过一条语句调用而不接收任何值，这种情况一般应用于无返回值的函数，实际上，如果函数有返回值，也可以当作语句被调用，此时函数返回值会被忽略；
- main函数：一般作为程序的入口函数。

```
def max(a, b):                  #定义max函数
    if a > b:
        result = a
    else:
        result = b
    return result

def main():                     #定义main函数
    x = 1
    y = 2
    z = max(x, y)               #main函数中调用max函数
    print("The larger number of", x, "and", y, "is", z )

main()                          #全局调用main函数
```

6.4 变量的作用域

局部变量

- 在函数内部定义的变量称为局部变量。局部变量只能在函数内部被访问，其作用域从创建变量的地方开始，到包含该局部变量的函数结束为止。

全局变量

- 在所有函数之外创建的变量被称为全局变量，可以被所有函数访问。全局变量可以在程序的任意位置访问，而如果试图在作用域外访问局部变量就会造成错误。

想一想：

1：局域变量和全局变量重名是否可以?

2：如何在函数内部定义一个全局变量?

6.5 函数的参数

形参：

- 函数定义时函数头中所包含的参数，形参类似于占位符的作用，并不是拥有值的变量。

实参：

- 函数调用时所使用的参数。在函数被调用时，调用者将实参的值传递给形参，形参才具有值。

6.5 函数的参数

基本的参数传递机制：

- 值传递
- 引用传递

Python的参数传递机制：传对象引用

- 需要对Python的对象按照内容是否可变划分为可变对象和不可变对象。所谓可变对象指的是对象的内容是可变的，而不可变对象指的是对象内容不可变。
- (1)不可变对象包括数字(整型、浮点型、布尔型等)、字符串、元组。
- (2)可变对象包括列表、字典。

```
def incrementInt(x):
    x += 1                          #整型为不可变对象，原实参内容不变

def incrementList(x):
    x += [1]                        #列表为可变对象，原实参内容会发生变化

def main():
    a = 3
    b = [2, 3, 4]
    incrementInt(a)
    print(a)
    incrementList(b)
    print(b)

main()
```

默认参数

Python中的函数允许定义默认参数。如果函数调用时没有传入某些参数的值，那么参数的默认值就会被传递给实参。

函数如果混用默认值参数或非默认值参数，则非默认值参数必须定义在默认值参数之前。

Python不支持函数重载。

```
def power(base, index=2):                    #index参数给出了默认值
    return base ** index

def calc(a=1, b=2, c=3):                     #a，b，c参数均给出了默认值
    return a + b * c

print(power(2, 5))                            #两参数均传入非默认实参
print(power(10))                              #index参数传入默认值
print(calc())                                 #a，b，c参数均传入默认值
print(calc(4))                                #b，c参数传入默认值
print(calc(4, 5))                             #c参数传入默认值
print(calc(4, 5, 6))                          #a，b，c参数均传入非默认实参
```

位置参数和关键字参数

使用位置参数要求参数按照函数定义时的顺序进行传递。

使用关键字参数调用函数通过类似“name=value”的格式传递每个参数。

当函数的参数有默认值时，使用关键字参数能够选择其中某些参数传入，其他参数传递其默认值。

另外，位置参数和关键字参数也可以混合使用，但要注意位置参数不能出现在任何关键字参数之后，并且位置参数和关键字参数不能传给一个形参。

可变长度

可变长度参数：指的是在函数定义时可以使用个数不确定的参数，同一函数可以使用不同个数的参数调用。

Python使用类似“*parameter”的语法来表示可变长度参数，在函数体内可以使用for循环来访问可变长度参数中的内容。

另外，Python还支持另一种形式的可变长度参数的用法，即使用类似“**parameter”的语法来表示参数，在调用时使用类似关键字参数的格式进行传递。

6.6 功能的封装

```
#检测年份是否为闰年
def isLeapYear(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False

#返回该月的最大天数，参数leapYear表示是否为闰年
def getMaxDayInMonth(month, leapYear):
    if month in {1, 3, 5, 7, 8, 10, 12}:
        return 31
    elif month in {4, 6, 9, 10}:
        return 30
    elif leapYear:
        return 29
    else:
        return 28

#检测年信息是否合法
def isYearValidate(year):
    if year > 0:
        return True
    else:
        return False
```

```
#检测月信息是否合法
def isMonthValidate(month):
    if 0 < month < 13:
        return True
    else:
        return False

#检测日信息是否合法，调用了getMaxDayInMonth函数
def isDayValidate(month, day, leapYear):
    if 1 <= day <= getMaxDayInMonth(month, leapYear):
        return True
    else:
        return False

#检测日期是否合法，调用了上面三个函数
def isDateValidate(year, month, day):
    leapYear = isLeapYear(year)
    if isYearValidate(year) and isMonthValidate(month) \
                    and isDayValidate(month, day, leapYear):
        return True
    else:
        return False
```

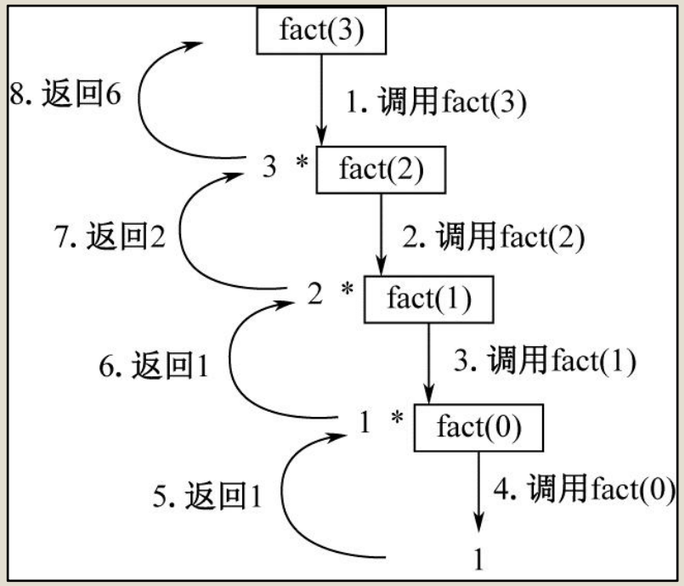
```
#定义main函数
def main():
    year = int(input("Please input the year:"))
    month = int(input("Please input the month:"))
    day = int(input("Please input the day:"))
    if isDateValidate(year, month, day):
        print("Valid date.")
    else:
        print("Invalid date.")

main()                  #执行main函数
```

6.7 递归

一个函数在内部也可以调用自身。我们将直接或间接调用自身的函数称为递归函数，将使用递归函数来解决问题的编程技巧称为递归。

递归能够将一个大型的、复杂的问题层层转换为一个与原问题相似的小规模问题来求解，给出一个自然、直观、简单的解决方案。



递归

递归函数的特点：

- (1)函数会使用选择结构将问题分成不同的情况。
- (2)函数中会有一个或多个基础情况用来结束递归。
- (3)非基础情况的分支会递归调用自身，递归会将原始问题简化为一个或多个子问题，这些子问题与原问题性质一样但规模更小。
- (4)每次递归调用会不断接近基础情况，直到变成基础情况，终止递归。

递归函数的优点是定义简单、逻辑清晰，但是一般的递归函数会占用大量的程序栈，尤其是当递归深度特别大的时候，有可能导致栈溢出。

在编写递归函数时，需要仔细考虑边界情况。当递归不能使全部的问题简化收敛到边界情况时，程序就会无限运行下去并且在程序栈溢出时导致运行时的错误。

```
def fact(n):
    if n == 0:
        return 1                                    #递归边界条件
    else:
        return n * fact(n - 1)                      #递归调用fact(n-1)

def main():
    n = int(input("Please input a nonnegative integer:"))
    print("Factorial of", n, "is", fact(n))

main()
```

一般而言，兔子在出生两个月后，就有繁殖能力，一对兔子每个月能生出一对小兔子来。
如果初始有1对0月龄的兔子，所有的兔子都不死，那么一年以后可以繁殖多少对兔子？

依次类推可以列出下表：

经过月数	0	1	2	3	4	5	6	7	8	9	10	11	12
总体对数	0	1	1	2	3	5	8	13	21	34	55	89	144

```
def fibonacci(x):
    if x == 0:
        return 0                                          #递归边界条件
    elif x == 1:
        return 1                                          #递归边界条件
    else:
        return fibonacci(x - 1) + fibonacci(x - 2)        #递归调用

def main():
    index = int(input("Please input an index:"))
    print("Fibonacci(", index, ") =", fibonacci(index))

main()
```

```
def fibonacci(x):
    a = 0
    b = 1
    c = 0
    for i in range(0, x):          #进行x次迭代
        a = b
        b = c
        c = a + b                  #每次迭代产生下一个斐波那契数
    return c

def main():
    index = int(input("Please input an index:"))
    print("Fibonacci(", index, ") =", fibonacci(index))

main()
```

6.8 lambda表达式

lambda表达式用于创建一个匿名函数，即没有与标识符绑定的函数。

- lambda 参数列表：表达式
- lambda表达式以lambda关键字开始，参数列表与一般函数的参数列表的语法格式相同，参数间用逗号隔开，允许参数有默认值。表达式为匿名函数的返回值，但只能由一个表达式组成，不能有其他的复杂结构。
- 通常而言，可将lambda表达式赋值给一个变量，这个变量就可以作为函数使用，此时就把lambda表达式和变量绑定在一起了。
- 调用lambda表达式的语法与调用函数完全相同。

```
sum = lambda x, y: x + y
sub = lambda x, y: x - y
max = lambda x, y: x if x > y else y          #带条件表达式的lambda表达式
min = lambda x, y: x if x < y else y          #带条件表达式的lambda表达式

print(sum(2, 3))
print(sub(5, 4))
print(max(2, 5))
print(min(4, 1))
print((lambda x, y: y * x)(2, 3))             #输出匿名lambda表达式结果
```

6.9 生成器

生成器是创建迭代器对象的一种简单而强大的工具。生成器的语法就像正常的函数，只是返回数据时需要使用yield语句而非return语句。

与一般函数不同的是，一般函数在执行到return语句时，会结束函数的执行；而生成器在执行到yield语句时，并不会终止执行，而是继续向后执行，直至函数结束。如果生成器中执行了多个yield语句，那么生成器将会把这些yield语句中所有要返回的值组成一个生成器对象并返回。

在生成器外部，可以通过next函数依次获得每一个值，也可以将其转换为某一类型的可迭代对象（如列表、元组等）。

```
def fibonacci(x):
    a = 0
    b = 1
    c = 0
    yield 0                                  #生成第一项斐波那契数
    for i in range(0, x):
        a = b
        b = c
        c = a + b
        yield c                              #每次迭代生成下一项斐波那契数

def main():
    index = int(input("Please input an index:"))
    fibs = fibonacci(index)
    for i in range(0,index+1):
        print("Fibonacci(",i,")=",next(fibs) )     #循环访问生成器对象的每个值

main()
```