

# 函数初步

黄书剑





# 函数示例

- 函数是一种程序实体
- 用于描述一个给定的功能（功能抽象）

```
>>> def add_by_3(x):  
...     return x + 3  
...  
>>> type(add_by_3)  
<class 'function'>
```

```
>>> add_by_3(100)  
103
```

C++:

```
int add_by_3(int x){  
    return x + 3;  
}  
add_by_3(100);
```

# 定义函数



- 复合语句

- clauses
- header/suite

```
def <name> (<formal parameters>) :  
    <suite>
```

- 将name与当前定义函数进行绑定
- 当函数被调用时，执行子句中的语句
  - 即函数功能

- 子句中 can 包含return语句

```
return <expression>
```

- 计算表达式的值，并完成函数子句的执行

- 函数名与函数的绑定关系

- 函数可以被绑定到新的变量
- 也可以通过赋值解除绑定

```
>>> def add_by_3(x):  
...     return x + 3  
...  
>>> type(add_by_3)  
<class 'function'>
```

```
>>> func = add_by_3  
>>> type(func)  
<class 'function'>  
>>> func = 5  
>>> type(func)  
<class 'int'>  
>>> func(1)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: 'int' object is not callable
```

# 函数调用

- 函数调用表达式

- 函数名：操作符
- 参数：操作数

\*也可以使用位置、名称、默认参数等方式进行参数绑定

- 函数执行：

- 将参数argument与函数的形参formal parameter进行绑定
- 执行函数定义子句中的操作语句
- 执行return语句时结束子句执行，并将return表达式的计算结果返回

```
def <name> (<formal parameters>) :  
    <suite>  
    return <expression>
```

```
<name>(<arguments>)
```

```
>>> def add_by_3(x):  
...     return x + 3  
...  
>>> add_by_3(100)  
103  
>>> func = add_by_3  
>>> func(1)  
4
```



## 参数传递的其他形式

- 默认的参数传递按位置逐一匹配实参和形参（位置参数）

```
>>> def dividing_op(num, denom):  
...     return num/denom  
...  
>>> print(dividing_op(1, 3))  
0.3333333333333333  
>>> print(dividing_op(3, 1))  
3.0
```

— 注意：参数传递时不进行类型匹配



## 参数传递的其他形式

- 允许按名进行参数传递 (关键字参数)
  - 指定与某个形参绑定

```
>>> def dividing_op(num, denom):  
...     return num/denom  
...
```

```
>>> print(dividing_op( num=3, denom=1 ))  
3.0  
>>> print(dividing_op( denom=3, num=1 ))  
0.3333333333333333  
>>>
```



## 参数传递的其他形式

- 允许定义函数时指定参数的默认值
  - 存在默认绑定对象

```
>>> def dividing_op(num, denom = 1):  
...     return num/denom  
...  
>>> print(dividing_op(3))  
3.0
```



## 参数传递的其他形式

- 可变长位置参数
  - 元组参数
  - 事先不确定个数的参数
- 可变长关键字参数
  - 字典参数 (名、值)
  - 事先不确定个数的参数对

```
>>> def myprint(first, *mylist):  
...     print(first)  
...     print(mylist)  
...  
>>> myprint(1, 2)  
1  
(2,)  
>>> myprint(1, 2, 3, 4)  
1  
(2, 3, 4)
```

```
>>> def myprint(**kwargs):  
...     for (k,v) in kwargs.items():  
...         print(type(k), ":", k, type(v), ":", v)  
...  
>>> myprint(a=1, b=2)  
<class 'str'> : a <class 'int'> : 1  
<class 'str'> : b <class 'int'> : 2
```

packing



# 函数的嵌套调用

- 函数中调用函数

```
>>> def square(x):  
...     return x * x  
...  
>>> def add(x, y):  
...     return x + y  
...  
>>> def sum_square(x, y):  
...     return add(square(x), square(y))  
...  
...
```

```
>>> x = 3  
>>> y = 5  
>>> z = sum_square(x, y)  
>>> z  
34
```



## 函数的返回值

- **return**表达式可以返回一个或多个对象 (元组, packing)

```
>>> def compute(x, y):  
...     return x + y, x - y  
...  
>>> z = compute(3, 5)  
>>> z  
(8, -2)  
>>> z1, z2 = compute(3, 5)  
>>> z1  
8  
>>> z2  
-2
```



## 使用文档字符串说明函数功能

- 文档字符串能够自动被识别为函数的帮助信息

```
>>> def compute(x, y):  
...     """ compute the sum and difference of x and y """  
...     return x + y, x - y  
...  
>>> help(compute)
```

```
Help on function compute in module __main__:
```

```
compute(x, y)  
    compute the sum and difference of x and y  
(END)
```



# 文档字符串

- 文档字符串可以用于包, 模块, 类或函数
  - 是这些单元中的第一个语句
  - 可以通过对象的\_\_doc\_\_成员获取
  - 可以被pydoc使用, 用于自动生成文档
  - \*可以用于doctest自动测试

<https://www.python.org/dev/peps/pep-0257/>

[https://zh-google-styleguide.readthedocs.io/en/latest/google-python-styleguide/python\\_style\\_rules/#comments](https://zh-google-styleguide.readthedocs.io/en/latest/google-python-styleguide/python_style_rules/#comments)

```
def fetch_bigtable_rows(big_table, keys, other_silly_variable=None):
    """Fetches rows from a Bigtable.

    Retrieves rows pertaining to the given keys from the Table instance
    represented by big_table. Silly things may happen if
    other_silly_variable is not None.

    Args:
        big_table: An open Bigtable Table instance.
        keys: A sequence of strings representing the key of each table row
            to fetch.
        other_silly_variable: Another optional variable, that has a much
            longer name than the other args, and which does nothing.

    Returns:
        A dict mapping keys to the corresponding table row data
        fetched. Each row is represented as a tuple of strings. For
        example:

        {'Serak': ('Rigel VII', 'Preparer'),
         'Zim': ('Irk', 'Invader'),
         'Lrrr': ('Omicron Persei 8', 'Emperor')}

        If a key from the keys argument is missing from the dictionary,
        then that row was not found in the table.

    Raises:
        IOError: An error occurred accessing the bigtable.Table object.
    """
    pass
```



## 函数作为抽象手段（初步）

- 程序设计层级的提升

```
for(int i = 0; i < length; i++)  
    //list[i]
```

```
for item in list:  
    # item
```

```
reversedstr = ''  
for item in inputstr:  
    reversedstr = item \  
        + reversedstr
```

```
"".join(reversed(inputstr))
```



## 函数作为抽象手段（初步）

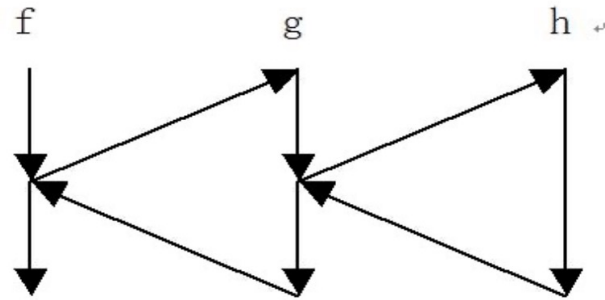
- 功能的一般化
  - 处理丰富、可变的数据

```
def money_split():  
    """ 将100拆为1、2、5分的组合 """  
  
def money_split(x):  
    """ 将x拆为1、2、5分的组合 """  
  
def money_split(x, pieces):  
    """ 将x拆为给定pieces的组合（pieces为整数的列表） """
```

# 递归函数

- **递归函数**

- 函数执行过程中直接或者间接调用其本身



- **问题求解：将复杂问题分解为一个或多个相对简单的同构问题**

- 分而治之 (divide and conquer)
- 其正确性一般可由归纳法证明





# 递归程序示例

- 求一元人民币换成一分、两分和五分的所有兑换方案数

```
>>> count = 0
>>> for i in range(21):
...     for j in range(51):
...         if ( 100 - 5 * i - 2 * j ) >= 0:
...             count += 1
...
>>> count
541
```

功能抽象：如何对于任意给定的金额x求解？

```
def split(x):
    count = 0
    for i in range(x//5 + 1):
        for j in range(x//2 + 1):
            k = x - 5 * i - 2 * j
            if k >= 0 :
                count += 1
                print("5"*i, "2"*j, "1"*k)
    return count
```

- 求一元人民币换成一分、两分和五分的所有兑换方案数
  - 发生一次选择，问题的规模就会变小
  - 直到无法选择为止

```
def split_recursive(x, current):  
    if x < 0:  
        return 0  
    if x == 0 :  
        print("".join(current))  
        return 1  
    else :  
        return split_recursive(x - 1, current + ['1'])  
            + split_recursive(x - 2, current + ['2'])  
            + split_recursive(x - 5, current + ['5'])
```

- 求一元人民币换成一分、两分和五分的所有兑换方案数
  - 对选择进行排序

```
def split_recursive_ordered(x, current, piece):  
    if x < 0:  
        return 0  
    if x == 0 :  
        print("".join(current))  
        return 1  
    elif piece == 5:  
        return split_recursive_ordered(x - 1, current + ['1'], 1) \  
            + split_recursive_ordered(x - 2, current + ['2'], 2) \  
            + split_recursive_ordered(x - 5, current + ['5'], 5) \  
    elif piece == 2:  
        return split_recursive_ordered(x - 1, current + ['1'], 1) \  
            + split_recursive_ordered(x - 2, current + ['2'], 2)  
    elif piece == 1:  
        return split_recursive_ordered(x - 1, current + ['1'], 1)
```

功能抽象：如何对于任意给定的硬币单元求解？

- try : 细胞分裂

- 有一个细胞每一个小时分裂一次，一次分裂一个子细胞，第三个小时后会死亡。那么 $n$ 个小时后有多少细胞？

- 函数的使用
  - 定义、调用、参数传递、返回值、文档注释
- 利用函数进行程序抽象
- 使用递归函数求解问题