Python语言基础回顾之函数

一、 函数的作用

需求：用户到ATM机取钱：

1. 输入密码后显示"选择功能"界面
2. 查询余额后显示"选择功能"界面
3. 取2000钱后显示"选择功能"界面

> 特点：显示“选择功能”界面需要重复输出给用户，怎么实现？

函数就是将一段具有独立功能的代码块 整合到一个整体并命名，在需要的位置调用这个名称即可完成对应的需求。

> 函数在开发过程中，可以更高效的实现代码重用。

二、 函数的使用步骤

2.1 定义函数

```
def 函数名(参数):
    代码1
    代码2
    ......
```

2.2 调用函数

```
函数名(参数)
```

> 注意：

1. 不同的需求，参数可有可无。
2. 在Python中，函数必须先定义后使用。

2.3 快速体验

需求：复现ATM取钱功能。

1. 搭建整体框架(复现需求)

```
print('密码正确登录成功')

# 显示"选择功能"界面

print('查询余额完毕')

# 显示"选择功能"界面

print('取了2000元钱')

# 显示"选择功能"界面
```

2. 确定“选择功能”界面内容

```
print('查询余额')
print('存款')
print('取款')
```

3. 封装"选择功能"

> 注意：一定是先定义函数，后调用函数。

```
# 封装ATM机功能选项 -- 定义函数
def select_func():
    print('-----请选择功能-----')
    print('查询余额')
    print('存款')
    print('取款')
    print('-----请选择功能-----')
```

4. 调用函数

在需要显示“选择功能”函数的位置调用函数。

```
print('密码正确登录成功')
# 显示"选择功能"界面 -- 调用函数
select_func()

print('查询余额完毕')
# 显示"选择功能"界面 -- 调用函数
select_func()

print('取了2000元钱')
# 显示"选择功能"界面 -- 调用函数
select_func()
```

三、函数的参数作用

思考：完成需求如下：一个函数完成两个数1和2的加法运算，如何书写程序？

```
# 定义函数
def add_num1():
    result = 1 + 2
    print(result)


# 调用函数
add_num1()
```

思考：上述add_num1函数只能完成数字1和2的加法运算，如果想要这个函数变得更灵活，可以计算任何用户指定的两个数字的和，如何书写程序？

分析：用户要在调用函数的时候指定具体数字，那么在定义函数的时候就需要接收用户指定的数字。函数调用时候指定的数字和定义函数时候接收的数字即是函数的参数。

```
# 定义函数时同时定义了接收用户数据的参数a和b，a和b是形参
def add_num2(a, b):
    result = a + b
    print(result)


# 调用函数时传入了真实的数据10 和 20，真实数据为实参
add_num2(10, 20)
```

四、函数的返回值作用

例如：我们去超市购物，比如买烟，给钱之后，是不是售货员会返回给我们烟这个商品，在函数中，如果需要返回结果给用户需要使用函数返回值。

```
def buy():
    return '烟'

# 使用变量保存函数返回值
goods = buy()
print(goods)
```

4.1 应用

需求：制作一个计算器，计算任意两数字之和，并保存结果。

```
def sum_num(a, b):
    return a + b


# 用result变量保存函数返回值
result = sum_num(1, 2)
print(result)
```

五、函数的说明文档

思考：定义一个函数后，程序员如何书写程序能够快速提示这个函数的作用？

答：注释

思考：如果代码多，我们是不是需要在很多代码中找到这个函数定义的位置才能看到注释？如果想更方便的查看函数的作用怎么办？

答：函数的说明文档

> 函数的说明文档也叫函数的文档说明。

5.1 语法

- 定义函数的说明文档

```
def 函数名(参数):
    """ 说明文档的位置 """
    代码
    ......
```

- 查看函数的说明文档

```
help(函数名)
```

5.2 快速体验

```
def sum_num(a, b):
    """ 求和函数 """
    return a + b


help(sum_num)
```

```
C:\Users\黑马程序员\AppData\Local\Programs\Python\Python37\python3.exe
Help on function sum_num in module __main__:

sum_num(a, b)
    求和函数


Process finished with exit code 0
```

六、函数嵌套调用

所谓函数嵌套调用指的是一个函数里面又调用了另外一个函数。

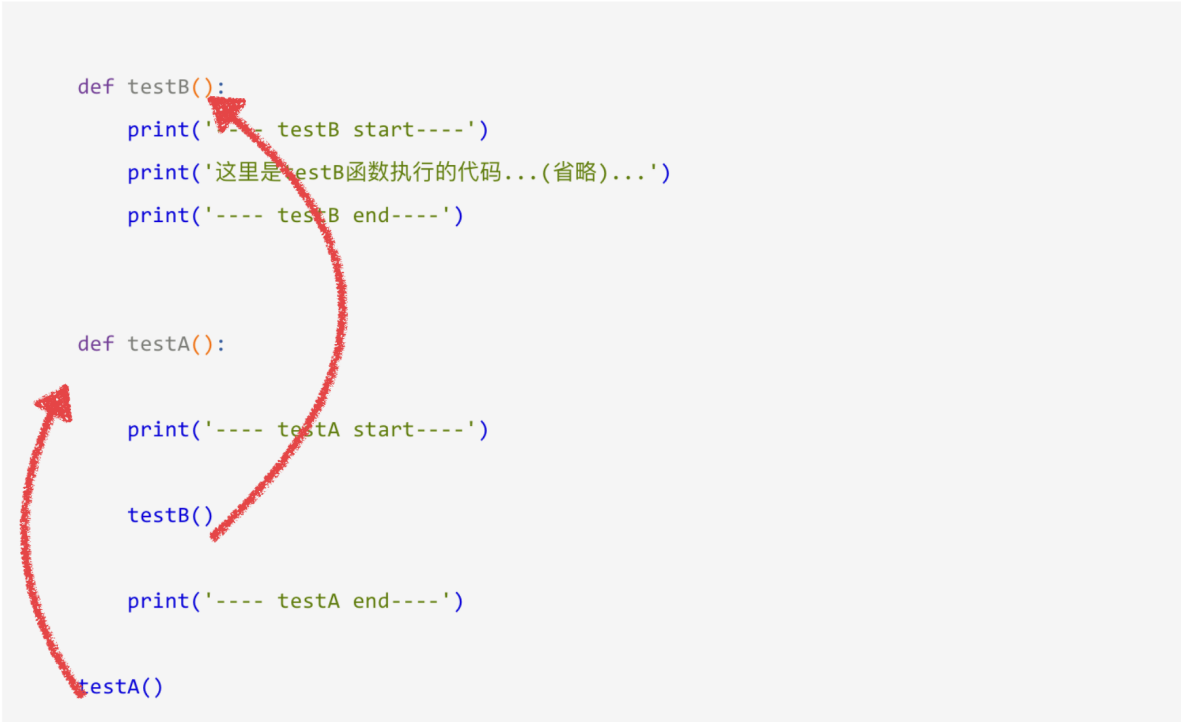
- 示例

```python
def testB():
    print('---- testB start----')
    print('这里是testB函数执行的代码...(省略)...')
    print('---- testB end----')

def testA():
    print('---- testA start----')
    testB()
    print('---- testA end----')

testA()
```

- 效果

```
C:\Users\黑马程序员\AppData\Local\Programs\Python\Python37\python3.exe
---- testA start----
---- testB start----
这里是testB函数执行的代码...(省略)...
---- testB end----
---- testA end----

Process finished with exit code 0
```

- 执行流程



> - 如果函数A中，调用了另外一个函数B，那么先把函数B中的任务都执行完毕之后才会回到上次 函数A执行的位置。

七、 函数应用

7.1 打印图形

1. 打印一条横线

```python
def print_line():
    print('-' * 20)


print_line()
```

```
C:\Users\黑马程序员\AppData\Local\Programs\Python\Python37\python3.exe
--------------------

Process finished with exit code 0
```

2. 打印多条横线

```python
def print_line():
    print('-' * 20)


def print_lines(num):
    i = 0
    while i < num:
        print_line()
        i += 1


print_lines(5)
```

```
C:\Users\黑马程序员\AppData\Local\Programs\Python\Python37\python3.exe
--------------------
--------------------
--------------------
--------------------
--------------------

Process finished with exit code 0
```

7.2 函数计算

1. 求三个数之和

```python
def sum_num(a, b, c):
    return a + b + c


result = sum_num(1, 2, 3)
print(result)  # 6
```

2. 求三个数平均值

```python
def average_num(a, b, c):
    sumResult = sum_num(a, b, c)
    return sumResult / 3

result = average_num(1, 2, 3)
print(result)  # 2.0
```

八、变量作用域

变量作用域指的是变量生效的范围，主要分为两类：局部变量和全局变量。

- 局部变量

所谓局部变量是定义在函数体内部的变量，即只在函数体内部生效。

```
def testA():
    a = 100

    print(a)


testA()  # 100
print(a)  # 报错：name 'a' is not defined
```

> 变量a是定义在`testA`函数内部的变量，在函数外部访问则立即报错。

局部变量的作用：在函数体内部，临时保存数据，即当函数调用完成后，则销毁局部变量。

- 全局变量

所谓全局变量，指的是在函数体内、外都能生效的变量。

思考：如果有一个数据，在函数A和函数B中都要使用，该怎么办？

答：将这个数据存储在一个全局变量里面。

```
# 定义全局变量a
a = 100


def testA():
    print(a)  # 访问全局变量a，并打印变量a存储的数据


def testB():
    print(a)  # 访问全局变量a，并打印变量a存储的数据


```

```
testA()  # 100
testB()  # 100
```

思考：`testB`函数需求修改变量a的值为200，如何修改程序？

```
a = 100


def testA():
    print(a)


def testB():
    a = 200
    print(a)


testA()  # 100
testB()  # 200
print(f'全局变量a = {a}')  # 全局变量a = 100
```

思考：在`testB`函数内部的`a = 200`中的变量a是在修改全局变量`a`吗？

答：不是。观察上述代码发现，15行得到a的数据是100，仍然是定义全局变量a时候的值，而没有返回

`testB`函数内部的200。综上：`testB`函数内部的`a = 200`是定义了一个局部变量。

思考：如何在函数体内部修改全局变量？

```
a = 100


def testA():
    print(a)

```

```python

def testB():
    # global 关键字声明a是全局变量
    global a
    a = 200
    print(a)


testA()  # 100
testB()  # 200
print(f'全局变量a = {a}')  # 全局变量a = 200
```

九、 多函数程序执行流程

一般在实际开发过程中，一个程序往往由多个函数（后面知识中会讲解类）组成，并且多个函数共享某些数据，如下所示：

- 共用全局变量

```python
# 1. 定义全局变量
glo_num = 0


def test1():
    global glo_num
    # 修改全局变量
    glo_num = 100


def test2():
    # 调用test1函数中修改后的全局变量
    print(glo_num)


```

```
# 2. 调用test1函数，执行函数内部代码：声明和修改全局变量
test1()
# 3. 调用test2函数，执行函数内部代码：打印
test2()  # 100
```

- 返回值作为参数传递

```
def test1():
    return 50


def test2(num):
    print(num)


# 1. 保存函数test1的返回值
result = test1()


# 2.将函数返回值所在变量作为参数传递到test2函数
test2(result)  # 50
```

十、 函数的返回值

思考：如果一个函数如些两个return (如下所示)，程序如何执行？

```
def return_num():
    return 1
    return 2


result = return_num()
print(result)  # 1
```

答：只执行了第一个return，原因是因为return可以退出当前函数，导致return下方的代码不执行。

思考：如果一个函数要有多个返回值，该如何书写代码？

```python
def return_num():
    return 1, 2


result = return_num()
print(result)  # (1, 2)
```

> 注意：
>
> 1. `return a, b`写法，返回多个数据的时候，默认是元组类型。
> 2. return后面可以连接列表、元组或字典，以返回多个值。

十一、 函数的参数

11.1 位置参数

位置参数：调用函数时根据函数定义的参数位置来传递参数。

```python
def user_info(name, age, gender):
    print(f'您的名字是{name}, 年龄是{age}, 性别是{gender}')


user_info('TOM', 20, '男')
```

> 注意：传递和定义参数的顺序及个数必须一致。

11.2 关键字参数

函数调用，通过“键=值”形式加以指定。可以让函数更加清晰、容易使用，同时也清除了参数的顺序需求。

```
def user_info(name, age, gender):
    print(f'您的名字是{name}, 年龄是{age}, 性别是{gender}')


user_info('Rose', age=20, gender='女')
user_info('小明', gender='男', age=16)
```

注意：**函数调用时，如果有位置参数时，位置参数必须在关键字参数的前面，但关键字参数之间不存在先后顺序。**

11.3 缺省参数

缺省参数也叫默认参数，用于定义函数，为参数提供默认值，调用函数时可不传该默认参数的值（注意：所有位置参数必须出现在默认参数前，包括函数定义和调用）。

```
def user_info(name, age, gender='男'):
    print(f'您的名字是{name}, 年龄是{age}, 性别是{gender}')


user_info('TOM', 20)
user_info('Rose', 18, '女')
```

> 注意：函数调用时，如果为缺省参数传值则修改默认参数值；否则使用这个默认值。

11.4 不定长参数

不定长参数也叫可变参数。用于不确定调用的时候会传递多少个参数(不传参也可以)的场景。此时，可用包裹(packing)位置参数，或者包裹关键字参数，来进行参数传递，会显得非常方便。

- 包裹位置传递

```python
def user_info(*args):
    print(args)


# ('TOM',)
user_info('TOM')
# ('TOM', 18)
user_info('TOM', 18)
```

> 注意：传进的所有参数都会被args变量收集，它会根据传进参数的位置合并为一个元组(tuple)，args是元组类型，这就是包裹位置传递。

- 包裹关键字传递

```python
def user_info(**kwargs):
    print(kwargs)


# {'name': 'TOM', 'age': 18, 'id': 110}
user_info(name='TOM', age=18, id=110)
```

> 综上：无论是包裹位置传递还是包裹关键字传递，都是一个组包的过程。

十二、 拆包和交换变量值

12.1 拆包

- 拆包：元组

```
def return_num():
    return 100, 200


num1, num2 = return_num()
print(num1)  # 100
print(num2)  # 200
```

- 拆包：字典

```
dict1 = {'name': 'TOM', 'age': 18}
a, b = dict1

# 对字典进行拆包，取出来的是字典的key
print(a)  # name
print(b)  # age

print(dict1[a])  # TOM
print(dict1[b])  # 18
```

12.2 交换变量值

需求：有变量`a = 10`和`b = 20`，交换两个变量的值。

- 方法一

借助第三变量存储数据。

```
# 1. 定义中间变量
c = 0

```

```
# 2. 将a的数据存储到c
c = a

# 3. 将b的数据20赋值到a，此时a = 20
a = b

# 4. 将之前c的数据10赋值到b，此时b = 10
b = c

print(a)  # 20
print(b)  # 10
```

- 方法二

```
a, b = 1, 2
a, b = b, a
print(a)  # 2
print(b)  # 1
```

十三、 引用

13.1 了解引用

在python中，值是靠引用来传递来的。

我们可以用`id()`来判断两个变量是否为同一个值的引用。 我们可以将id值理解为那块内存的地址标识。

```
# 1. int类型
a = 1
b = a

print(b)  # 1

print(id(a))  # 140708464157520
```

```
print(id(b))  # 140708464157520

a = 2
print(b)  # 1,说明int类型为不可变类型

print(id(a))  # 140708464157552，此时得到是的数据2的内存地址
print(id(b))  # 140708464157520


# 2. 列表
aa = [10, 20]
bb = aa

print(id(aa))  # 2325297783432
print(id(bb))  # 2325297783432


aa.append(30)
print(bb)  # [10, 20, 30], 列表为可变类型

print(id(aa))  # 2325297783432
print(id(bb))  # 2325297783432
```

13.2 引用当做实参

代码如下：

```
def test1(a):
    print(a)
    print(id(a))

    a += a

    print(a)
    print(id(a))
```

```python

# int：计算前后id值不同
b = 100
test1(b)

# 列表：计算前后id值相同
c = [11, 22]
test1(c)
```

效果图如下：

```
C:\Users\黑马程序员\AppData\Local\Programs\Python\Python37\python3.exe
100
140708464160688
200
140708464163888
[11, 22]
2008286519944
[11, 22, 11, 22]
2008286519944

Process finished with exit code 0
```

十四、 可变和不可变类型

所谓可变类型与不可变类型是指：数据能够直接进行修改，如果能直接修改那么就是可变，否则是不可变.

- 可变类型
  - 列表
  - 字典
  - 集合
- 不可变类型
  - 整型
  - 浮点型
  - 字符串
  - 元组

十五、函数应用：学员管理系统

15.1 系统简介

需求：进入系统显示系统功能界面，功能如下：

- 1、添加学员
- 2、删除学员
- 3、修改学员信息
- 4、查询学员信息
- 5、显示所有学员信息
- 6、退出系统

系统共6个功能，用户根据自己需求选取。

15.2 步骤分析

1. 显示功能界面
2. 用户输入功能序号
3. 根据用户输入的功能序号，执行不同的功能(函数)

    3.1 定义函数

    3.2 调用函数

15.3 需求实现

15.3.1 显示功能界面

定义函数`print_info`，负责显示系统功能。

```
def print_info():
    print('-' * 20)
    print('欢迎登录学员管理系统')
    print('1: 添加学员')
    print('2: 删除学员')
    print('3: 修改学员信息')
    print('4: 查询学员信息')
    print('5: 显示所有学员信息')
    print('6: 退出系统')
    print('-' * 20)


print_info()
```

15.3.2 用户输入序号，选择功能

```
user_num = input('请选择您需要的功能序号: ')
```

15.3.3 根据用户选择，执行不同的功能

```python
if user_num == '1':
    print('添加学员')
elif user_num == '2':
    print('删除学员')
elif user_num == '3':
    print('修改学员信息')
elif user_num == '4':
    print('查询学员信息')
elif user_num == '5':
    print('显示所有学员信息')
elif user_num == '6':
    print('退出系统')
```

> 工作中，需要根据实际需求调优代码。
>
> 1. 用户选择系统功能的代码需要循环使用，直到用户主动退出系统。
> 2. 如果用户输入1-6以外的数字，需要提示用户。

```python
while True:
    # 1. 显示功能界面
    print_info()

    # 2. 用户选择功能
    user_num = input('请选择您需要的功能序号：')

    # 3. 根据用户选择，执行不同的功能
    if user_num == '1':
        print('添加学员')
    elif user_num == '2':
        print('删除学员')
    elif user_num == '3':
        print('修改学员信息')
    elif user_num == '4':
        print('查询学员信息')
    elif user_num == '5':
        print('显示所有学员信息')
```

```
    elif user_num == '6':
        print('退出系统')
    else:
        print('输入错误，请重新输入!!!')
```

15.3.4 定义不同功能的函数

所有功能函数都是操作学员信息，所有存储所有学员信息应该是一个全局变量，数据类型为列表。

```
info = []
```

15.3.4.1 添加学员

- 需求分析

1. 接收用户输入学员信息，并保存
2. 判断是否添加学员信息

   2.1 如果学员姓名已经存在，则报错提示

   2.2 如果学员姓名不存在，则准备空字典，将用户输入的数据追加到字典，再列表追加字典数据
3. 对应的if条件成立的位置调用该函数

- 代码实现

```
def add_info():
    """ 添加学员 """
    # 接收用户输入学员信息
    new_id = input('请输入学号：')
    new_name = input('请输入姓名：')
    new_tel = input('请输入手机号：')


```

```python
    # 声明info是全局变量
    global info

    # 检测用户输入的姓名是否存在，存在则报错提示
    for i in info:
        if new_name == i['name']:
            print('该用户已经存在！')
            return

    # 如果用户输入的姓名不存在，则添加该学员信息
    info_dict = {}

    # 将用户输入的数据追加到字典
    info_dict['id'] = new_id
    info_dict['name'] = new_name
    info_dict['tel'] = new_tel

    # 将这个学员的字典数据追加到列表
    info.append(info_dict)

    print(info)
```

15.3.4.2 删除学员

- 需求分析

按用户输入的学员姓名进行删除

1. 用户输入目标学员姓名
2. 检查这个学员是否存在

   2.1 如果存在，则列表删除这个数据

   2.2 如果不存在，则提示“该用户不存在”
3. 对应的if条件成立的位置调用该函数

- 代码实现

```python
# 删除学员
def del_info():
    """删除学员"""
    # 1. 用户输入要删除的学员的姓名
    del_name = input('请输入要删除的学员的姓名：')

    global info
    # 2. 判断学员是否存在:如果输入的姓名存在则删除，否则报错提示
    for i in info:
        if del_name == i['name']:
            info.remove(i)
            break
    else:
        print('该学员不存在')

    print(info)
```

15.3.4.3 修改学员信息

- 需求分析

1. 用户输入目标学员姓名
2. 检查这个学员是否存在

   2.1 如果存在，则修改这位学员的信息，例如手机号

   2.2 如果不存在，则报错

3. 对应的if条件成立的位置调用该函数

- 代码实现

```python
# 修改函数
def modify_info():
    """修改函数"""
    # 1. 用户输入要修改的学员的姓名
    modify_name = input('请输入要修改的学员的姓名：')
```

```
    global info
    # 2. 判断学员是否存在：如果输入的姓名存在则修改手机号，否则报错提示
    for i in info:
        if modify_name == i ['name']:
            i['tel'] = input('请输入新的手机号：')
            break
    else:
        print('该学员不存在')

    print(info)
```

15.3.4.4 查询学员信息

- 需求分析

1. 用户输入目标学员姓名
2. 检查学员是否存在

   2.1 如果存在，则显示这个学员的信息

   2.2 如果不存在，则报错提示
3. 对应的if条件成立的位置调用该函数

- 代码实现

```
# 查询学员
def search_info():
    """查询学员"""
    # 1. 输入要查找的学员姓名：
    search_name = input('请输入要查找的学员姓名：')

    global info
    # 2. 判断学员是否存在：如果输入的姓名存在则显示这位学员信息，否则报错提示
```

```
    for i in info:
        if search_name == i['name']:
            print('查找到的学员信息如下：----------')
            print(f"该学员的学号是{i['id']}, 姓名是{i['name']}, 手机号是{i['tel']}")
            break
    else:
        print('该学员不存在')
```

15.3.4.5 显示所有学员信息

- 需求分析

打印所有学员信息

- 代码实现

```
# 显示所有学员信息
def print_all():
    """ 显示所有学员信息 """
    print('学号\t姓名\t手机号')
    for i in info:

 print(f'{i["id"]}\t{i["name"]}\t{i["tel"]}')
```

15.3.4.6 退出系统

在用户输入功能序号`6`的时候要退出系统，代码如下：

```
    ......
    elif user_num == '6':
        exit_flag = input('确定要退出吗？yes or no')
        if exit_flag == 'yes':
            break
```

十六、 递归

16.1 递归的应用场景

递归是一种编程思想，应用场景：

1. 在我们日常开发中，如果要遍历一个文件夹下面所有的文件，通常会使用递归来实现；
2. 在后续的算法课程中，很多算法都离不开递归，例如：快速排序。

16.1.1 递归的特点

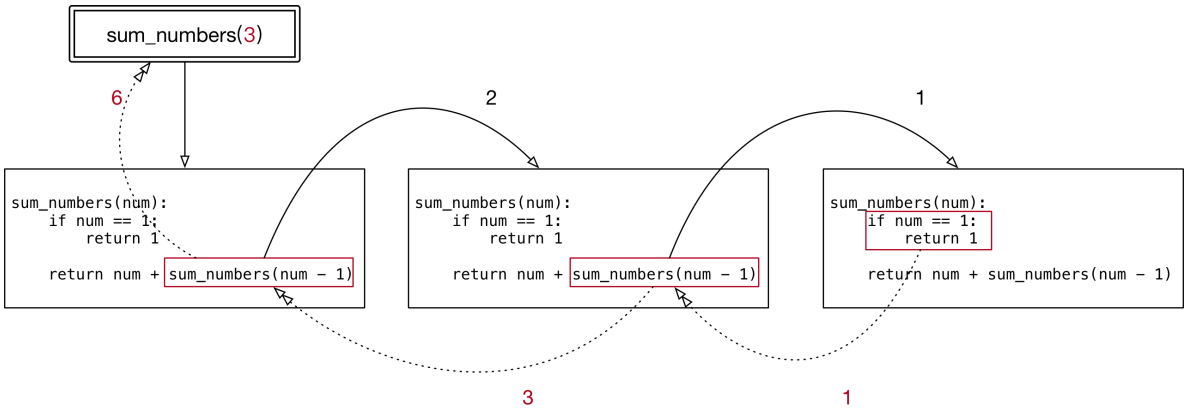
- 函数内部自己调用自己
- 必须有出口

16.2 应用：3以内数字累加和

- 代码

```python
# 3 + 2 + 1
def sum_numbers(num):
    # 1.如果是1，直接返回1 -- 出口
    if num == 1:
        return 1
    # 2.如果不是1，重复执行累加并返回结果
    return num + sum_numbers(num-1)


sum_result = sum_numbers(3)
# 输出结果为6
print(sum_result)
```

- 执行结果



十七、 lambda 表达式

17.1 lambda的应用场景

如果一个函数有一个返回值，并且只有一句代码，可以使用lambda简化。

17.2 lambda语法

```
lambda 参数列表 ： 表达式
```

> 注意

- lambda表达式的参数可有可无，函数的参数在lambda表达式中完全适用。
- lambda表达式能接收任何数量的参数但只能返回一个表达式的值。

快速入门

```
# 函数
def fn1():
    return 200


print(fn1)
print(fn1())


# lambda表达式
fn2 = lambda: 100
print(fn2)
print(fn2())
```

> 注意：直接打印lambda表达式，输出的是此lambda的内存地址

17.3 示例：计算a + b

17.3.1 函数实现

```python
def add(a, b):
    return a + b


result = add(1, 2)
print(result)
```

> 思考：需求简单，是否代码多？

17.3.2 lambda实现

```python
fn1 = lambda a, b: a + b
print(fn1(1, 2))
```

17.4 lambda的参数形式

17.4.1.无参数

```python
fn1 = lambda: 100
print(fn1())
```

17.4.2.一个参数

```python
fn1 = lambda a: a
print(fn1('hello world'))
```

17.4.3.默认参数

```python
fn1 = lambda a, b, c=100: a + b + c
print(fn1(10, 20))
```

17.4.4.可变参数：*args

```python
fn1 = lambda *args: args
print(fn1(10, 20, 30))
```

> 注意：这里的可变参数传入到lambda之后，返回值为元组。

17.4.5.可变参数：**kwargs

```python
fn1 = lambda **kwargs: kwargs
print(fn1(name='python', age=20))
```

17.5 lambda的应用

17.5.1. 带判断的lambda

```python
fn1 = lambda a, b: a if a > b else b
print(fn1(1000, 500))
```

17.5.2. 列表数据按字典key的值排序

```python
students = [
    {'name': 'TOM', 'age': 20},
    {'name': 'ROSE', 'age': 19},
    {'name': 'Jack', 'age': 22}
]

# 按name值升序排列
students.sort(key=lambda x: x['name'])
print(students)

# 按name值降序排列
students.sort(key=lambda x: x['name'], reverse=True)
print(students)
```

```
14
15 # 按age值升序排列
16 students.sort(key=lambda x: x['age'])
17 print(students)
```

十八、高阶函数

把函数作为参数传入，这样的函数称为高阶函数，高阶函数是函数式编程的体现。函数式编程就是指这种高度抽象的编程范式。

18.1 体验高阶函数

在Python中，`abs()`函数可以完成对数字求绝对值计算。

```
1 abs(-10)  # 10
```

`round()`函数可以完成对数字的四舍五入计算。

```
1 round(1.2)  # 1
2 round(1.9)  # 2
```

需求：任意两个数字，按照指定要求整理数字后再进行求和计算。

- 方法1

```
1 def add_num(a, b):
2     return abs(a) + abs(b)
3
4
5 result = add_num(-1, 2)
6 print(result)  # 3
```

- 方法2

```python
def sum_num(a, b, f):
    return f(a) + f(b)


result = sum_num(-1, 2, abs)
print(result)  # 3
```

> 注意：两种方法对比之后，发现，方法2的代码会更加简洁，函数灵活性更高。

函数式编程大量使用函数，减少了代码的重复，因此程序比较短，开发速度较快。

18.2 内置高阶函数

18.2.1 map()

map(func, lst)，将传入的函数变量func作用到lst变量的每个元素中，并将结果组成新的列表(Python2)/迭代器(Python3)返回。

需求：计算`list1`序列中各个数字的2次方。

```python
list1 = [1, 2, 3, 4, 5]


def func(x):
    return x ** 2


result = map(func, list1)

print(result)  # <map object at 0x0000013769653198>
print(list(result))  # [1, 4, 9, 16, 25]
```

18.2.2 reduce()

reduce(func，lst)，其中func必须有两个参数。每次func计算的结果继续和序列的下一个元素做累积计算。

> 注意：reduce()传入的参数func必须接收2个参数。

需求：计算`list1`序列中各个数字的累加和。

```
import functools

list1 = [1, 2, 3, 4, 5]


def func(a, b):
    return a + b


result = functools.reduce(func, list1)

print(result)  # 15
```

18.2.3 filter()

filter(func, lst)函数用于过滤序列, 过滤掉不符合条件的元素, 返回一个 filter 对象。如果要转换为列表, 可以使用 list() 来转换。

```python
list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]


def func(x):
    return x % 2 == 0


result = filter(func, list1)

print(result)  # <filter object at 0x0000017AF9DC3198>
print(list(result))  # [2, 4, 6, 8, 10]
```