python基础4：函数

python基础4：函数

一、函数

1. 函数的基本使用

1.1 【理解】函数的作用

1.2 【重点】函数的定义和调用

1.3 【理解】函数的执行过程

1.4 【记忆】函数的文档注释

2. 函数的参数

2.1 【理解】函数参数的作用

2.2 【重点】函数参数的使用

2.3 【记忆】形参的作用域

3. 函数的返回值

3.1 【理解】函数返回值的作用

3.3 【重点】通过return给函数设置返回值

3.4 【记忆】函数默认返回值

3.5 【重点】return中断函数

4. 【重点】四种函数的类型

5. 【理解】函数的嵌套调用

6. 局部变量和全局变量

6.1 【记忆】局部变量和全局变量的区别

6.2 【重点】通过global声明修改全局变量

7. 函数参数详解

7.1【知道】位置参数和关键字参数区别
7.2【重点】默认形参(缺省形参)
7.3【重点】元组型不定长参数
7.4【重点】字典型不定长参数

一、函数

1. 函数的基本使用

1.1【理解】函数的作用

- 提高代码**编写效率**, 代码**重用**

1.2【重点】函数的定义和调用

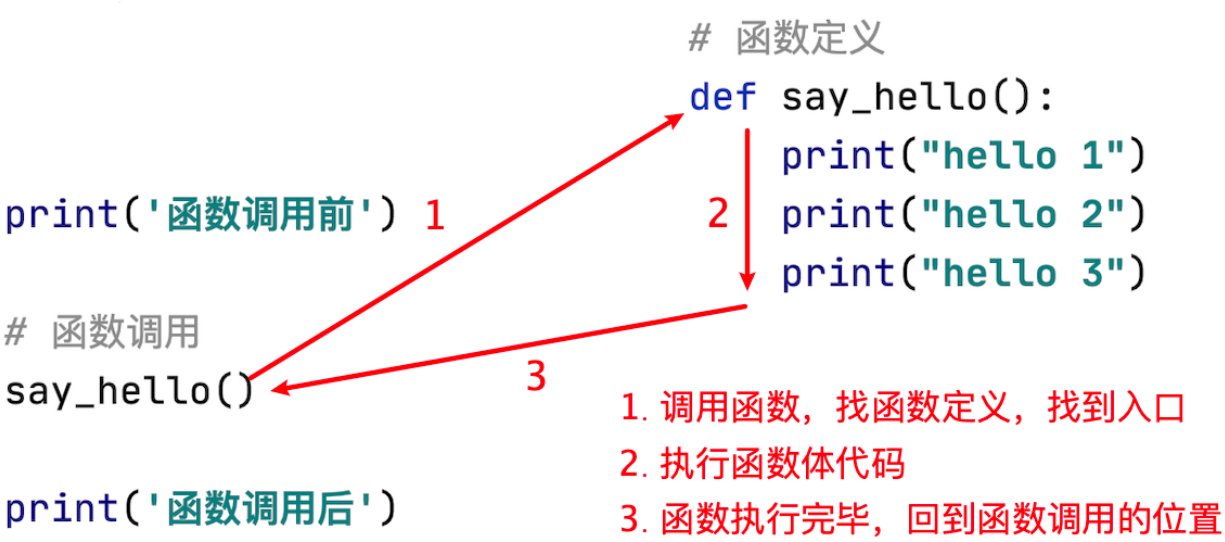
- 函数定义：

```
'''
定义函数：
def 函数名():
    函数的代码块
'''

# 注意：函数只是定义，不调用用户看不到没有效果的
def say_hello():
    print("hello 1")
    print("hello 2")
    print("hello 3")
```

- 函数调用：

```
'''
函数名()
'''

say_hello()
```

1.3【理解】函数的执行过程



1.4【记忆】函数的文档注释

- 函数体内第一行多行注释即为函数的文档注释，主要写函数的描述信息

```
# 函数的文档注释：  函数体内第一行多行注释即为函数的文档注释，主要写函数的描述信息

# 函数定义


def foo():
    """
    作用：计算1+2的结果,并打印输出
    """
    result = 1 + 2
    print(result)
```

```
# 光标移动到函数名上,使用ctrl + q 查看函数说明
foo()
```

2. 函数的参数

2.1【理解】函数参数的作用

- 函数参数，**可以传递数据给函数内部**，增加函数的 **通用性**

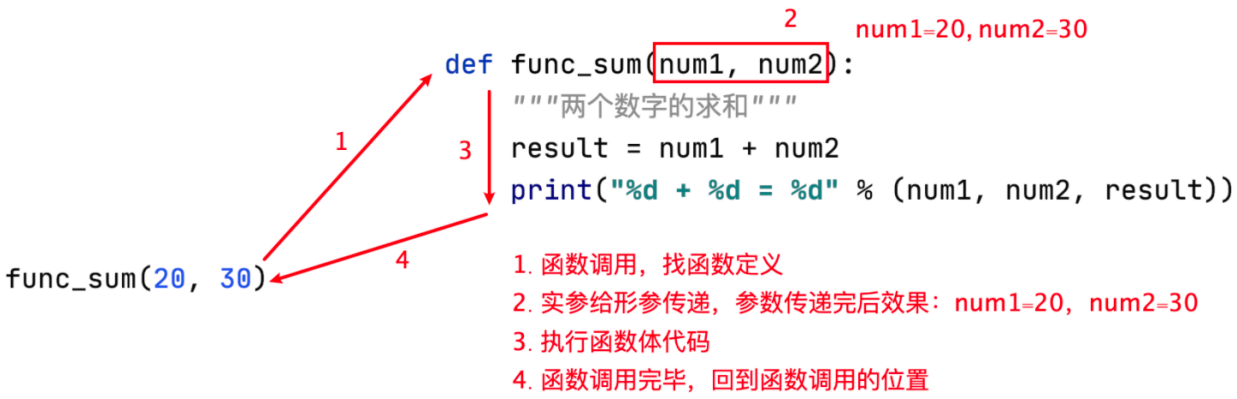
2.2【重点】函数参数的使用

- 带参数函数的定义格式：

```
def 函数名(形参1, 形参2, ...):
    函数体代码块
```

- 带参数函数的调用格式：

```
函数名(实参1, 实参2, ...)
```




```
"""
函数定义时的参数叫 形参，函数调用时的参数叫 实参

带参数函数的定义格式：
def 函数名字(形参1，形参2，……):
    函数体代码块

带参数函数的调用格式：
函数名字(实参1，实参2，……)
"""


# 带参数函数的函数定义
# def 函数名字(形参1，形参2，……):
#     函数体代码块

def foo(num1, num2):
    result = num1 + num2
    print(f"{num1} + {num2} = {result}")
```

```
# 带参数函数的调用
# 函数名字(实参1，实参2，……)
foo(10, 20)
foo(20, 30)
```

2.3【记忆】形参的作用域

- 作用域：变量起作用的范围
- 形参的作用域：**只在定义函数的代码块** 中

```
# 形参：函数定义是()里面的参数
# 作用域：变量起作用的范围
# 形参的作用域只在当前函数的内部有效，和外面没有任何关系


def func_sum(num1, num2):
    result = num1 + num2
    print(f"{num1} + {num2} = {result}")


def func_sub(num1, num2):
```

```
    result = num1 - num2
    print(f"{num1} - {num2} = {result}")


# 形参作用域仅在函数内部，所以不同函数的同名参数，互不影响
func_sum(10, 20)
# print(num1, num2)   # NameError: name 'num1' is not defined
func_sub(100, 50)
```

3. 函数的返回值

3.1【理解】函数返回值的作用

- 开发中，有时会希望 **一个函数执行结束后，告诉调用者一个结果**，以便调用者针对具体的结果做后续的处理

```
"""
返回值变量 = input()
返回值变量 = len(容器变量)
"""
```

```
# 从键盘输入一个内容，内容返回给name赋值
name = input("请输入用户名：")
print(name)

# 获取my_str内容的元素个数，返回给n赋值
my_str = "hello"
n = len(my_str)
print(n)
```

3.3【重点】通过return给函数设置返回值

```
"""
函数返回值作用：函数处理完，返回处理结果给调用者
return关键字：中断函数，同时也返回一个结果

函数定义格式：
def 函数名():
    return 结果

函数调用格式：
返回值变量 = 函数名()
"""
```

```
# 函数定义格式:
# def 函数名():
def func_add(num1, num2):
    result = num1 + num2
    #     return 结果
    return result


# 函数调用格式:
# 返回值变量 = 函数名()
ret = func_add(10, 20)
print(ret)
```

3.4【记忆】函数默认返回值

- 函数内部没有任何return语句，默认返回None，表示没有任何数据
- return不设置返回值，默认也返回None

```
"""
1. 函数内部没有任何return语句，默认返回None，表示没有任何数据
2. return不设置返回值，默认有返回None
```

```
"""


# 函数定义，函数内部没有任何return语句
def foo1():
    print("^_^")


# 函数定义，return不设置返回值
def foo2():
    return


# 函数调用
ret = foo1()
print(ret)

ret = foo2()
print(ret)
```

3.5【重点】return中断函数

- 函数一旦执行return，return下一句往后的代码不会执行

```
# 函数一旦执行return，函数内return下一句往后的代码不会执行


# 函数定义
def foo():
    print("不努力,你来黑马程序员干嘛?")
    return
    print("我要玩游戏")


# 函数调用
foo()
```

4. 【重点】四种函数的类型

- 无参数，无返回值

```
"""无参数，无返回值的格式

函数定义:
def 函数名():
    函数体
```

```
函数调用:
函数名()
"""


# 函数定义
def menu():
    """
    函数功能:实现打印菜单
    :return: 没有返回值
    """
    print("功能菜单")
    print("="*40)
    print("= 1. 添加学生")
    print("= 2. 查询学生")
    print("= 3. 删除学生")
    print("="*40)


# 函数调用
menu()
```

- 无参数，有返回值

```
"""无参数，有返回值的函数
```

```
函数定义:
def 函数名字():
    return 返回结果

函数调用:
返回值变量 = 函数名字()
"""


# 函数定义
def get_pi():
    """
    返回圆周率
    :return: 3.14
    """
    return 3.14


# 函数调用
pi = get_pi()
print(pi)
```

- 有参数，无返回值

```
"""有参数，无返回值的函数
函数定义:
```

```
def 函数名字(形参1，形参2，……):
    函数体

函数调用:
函数名(实参1，实参2，……)
"""


# 函数定义
def print_char(num, char):
    """
    输出指定数量的字符
    :param num: 字符数量
    :param char: 输出的字符
    :return: None
    """
    print(char * num)


# 函数调用
print_char(10, '^-^')
```

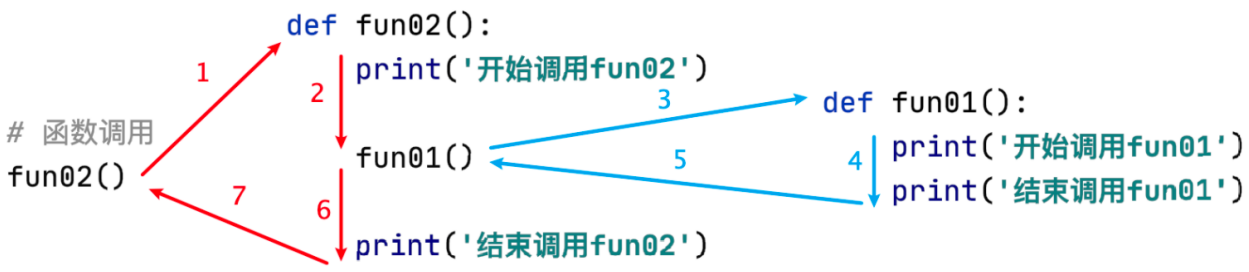
- 有参数，有返回值

```
"""有参数，有返回值的函数
函数定义:
```

```
def 函数名字(形参1，形参2，……):
    函数体
    return 返回结果

函数调用:
返回变量 = 函数名字(实参1，实参2，……)

循环累加流程:
# 1. 设置条件变量
# 2. while 条件(i <= n):
    # 3. 累加
    # 4. 条件变量改变
# 5. 循环外面，返回累加的最终结果
"""


# 函数定义
def func_sum(n):
    """
    求1~n的累加
    :param n: 累加的数字个数
    :return: 累加的结果
    """
    # 循环累加流程:
    # 1. 设置条件变量
    i = 1
```

```
    _sum = 0
    # 2. while 条件(i <= n):
    while i <= n:
        # 3. 累加
        _sum += i
        # 4. 条件变量改变
        i += 1
    # 5. 循环外面，返回累加的最终结果
    return _sum


# 函数调用
ret = func_sum(100)
print(f"1~100累加: {ret}")
```

5. 【理解】函数的嵌套调用




```
"""
函数的嵌套调用：函数里面调用别的函数
"""
```

```
# 定义fun01函数
def fun01():
    print("调用fun01开始")
    print("调用fun01结束")


# 定义fun02函数，在代码块中间调用fun01
def fun02():
    print("调用fun02开始")
    fun01()
    print("调用fun02结束")


# 函数调用
fun02()
```

- 案例1-打印多行分割线:

```
"""
1. 设计一个函数，打印一行分隔线：可指定数量，可指定分隔线字符的样式
    如:  一行分隔线字符的数量为5，字符样式为'^_^'
    ^_^ ^_^ ^_^ ^_^ ^_^
```

```
2. 设计一个函数，打印n行分隔线，可指定一行分隔线字符的数量，可指定分隔线字符的样式
    如：3行分隔线，一行分隔线字符的数量为5，字符样式为'^_^ '
    ^_^ ^_^ ^_^ ^_^ ^_^
    ^_^ ^_^ ^_^ ^_^ ^_^
    ^_^ ^_^ ^_^ ^_^ ^_^

步骤流程:
# 1. 设置条件变量 i = 0
# 2. while 条件:
    # 3. 打印一行的分隔线
    # 4. 条件变量的改变
"""

"""
1. 设计一个函数，打印一行分隔线：可指定数量，可指定分隔线字符的样式
    如：  一行分隔线字符的数量为5，字符样式为'^_^ '
    ^_^ ^_^ ^_^ ^_^ ^_^
"""


def print_char(num, char):
    """
```

```
    打印一行分隔线
    :param num: 指定数量
    :param char: 指定分隔线字符的样式
    :return: 没有返回值
    """
    print(num * char)


"""
2. 设计一个函数，打印n行分隔线，可指定一行分隔线字符的数量，可指定分隔线字符的样式
    如：3行分隔线，一行分隔线字符的数量为5，字符样式为'^_^ '
    ^_^ ^_^ ^_^ ^_^ ^_^
    ^_^ ^_^ ^_^ ^_^ ^_^
    ^_^ ^_^ ^_^ ^_^ ^_^
"""


def print_lines(n, num, char):
    """
    打印n行分隔线
    :param n: n行
    :param num: 指定一行分隔线字符的数量
    :param char: 指定分隔线字符的样式
    :return: 没有返回值
```

```
    """
    # 1. 设置条件变量 i = 0
    i = 0
    # 2. while 条件:
    while i < n:
        # 3. 打印一行的分隔线
        print_char(num, char)
        # 4. 条件变量的改变
        i += 1


# print_char(5, "^_^")

print_lines(10, 10, "^_^ ")
```

- 案例2-求三个数的平均值

```
"""
1. 设计一个函数求三个数的和
2. 设计一个函数求三个数的平均值
    # 2.1 先对3数求和，返回值为求和后的结果
    # 2.2 接着，再求平均值
    # 2.3 最终平均值结果作为函数返回值
"""
```

```python
# 1. 设计一个函数求三个数的和
def func_sum(n1, n2, n3):
    """
    求出三个数的和
    :param n1: 第一个数
    :param n2: 第二个数
    :param n3: 第三个数
    :return: 返回三个数之和
    """
    return n1 + n2 + n3


# 2. 设计一个函数求三个数的平均值
def func_avg(n1, n2, n3):
    """
    求出三个数的平均值
    :param n1: 第一个数
    :param n2: 第二个数
    :param n3: 第三个数
    :return: 返回三个数平均值
    """
    #     # 2.1 先对3数求和，返回值为求和后的结果
    _sum = func_sum(n1, n2, n3)
    #     # 2.2 接着，再求平均值
```

```
    _avg = _sum / 3
    #     # 2.3 最终平均值结果作为函数返回值
    return _avg


ret = func_avg(10, 20, 30)
print(ret)
```

6. 局部变量和全局变量

6.1【记忆】局部变量和全局变量的区别

- 定义方式不同
  - 局部变量是定义在函数内部的变量
  - 全局变量是定义在函数外部的变量
- 作用域不同
  - 局部变量只在定义所在的函数内部有效
  - 全局变量在所有函数内都有效，函数内和函数外都可以访问
- 局部变量:

```
"""
局部变量：函数定义的形参、函数内部定义的变量是局部变量，局部变量只能在函数内部使用
局部变量的作用域只在函数内部
"""


# 函数定义
def foo(temp):
    a = 10
    print(a, temp)


# 函数调用
foo(22)

# print(a)    # NameError: name 'a' is not defined
# print(temp) # NameError: name 'temp' is not defined
```

- 全局变量

```
"""
```

```
1. 在函数外部定义的变量叫做 全局变量
2. 全局变量能够在所有的函数中进行访问(不修改)
"""

# 定义全局变量
g_num = 10


def foo():
    print("函数中访问全局变量: ", g_num)


foo()
print("函数外访问全局变量: ", g_num)
```

6.2【重点】通过global声明修改全局变量

- 函数内修改全局变量：先global声明全局变量，再修改

```
# 函数内修改全局变量，先global声明全局变量，再修改

# 定义全局变量
```

```python
g_num = 10


def foo():
    # 使用global在函数内声明 num 是全局变量
    global g_num
    g_num = 250  # g_num 是全局变量，修改全局变量
    print("函数中访问全局变量：", g_num)


foo()
print("函数外访问全局变量：", g_num)
```

7. 函数参数详解

7.1【知道】位置参数和关键字参数区别

- 位置参数：按形参的位置，从左往右，一一匹配传递参数
- 关键字参数：通过 **形参=值** 方式为函数形参传值，无需和形参位置一一对应

位置参数:

```
"""
1. 函数调用时，按形参的位置，从左往右，一一匹配传递参数
2. 位置参数必须一一对应，缺一不可
"""


# 函数定义
def foo(num1, num2):
    print(num1, num2)


def foo2(name, age, sex):
    print("姓名: %s 年龄: %d 性别: %s" %(name, age, sex))


# 函数调用
foo(10, 20)

foo2("小明", 20, '男')

# foo2("小芳", '女', 30) # TypeError: %d format: a number is required, not str
```

- 关键字参数：

```
# 函数调用时，通过形参=值方式为函数形参传值
# 不用按照位置为函数形参传值，这种方式叫关键字参数


# 函数定义
def foo(name, age, sex):
    print("姓名：%s 年龄：%d 性别：%s" %
(name, age, sex))


# 函数调用
# 位置参数传递：按照形参的顺序,从左到右依次传递
# foo("小明", 20, '男')

# 关键字参数传递：通过形参=值方式为函数形参传值，
不用按照位置为函数形参传值
foo(sex='男', name='小刘', age=22)

# 第一个参数是位置参数：按照形参的顺序传递. 第二和
第三个参数是关键字参数：按照形参=值
#    位置参数在左边，关键字参数在右边
foo("小李", sex='男', age=20)
```

```
# 注意1：关键字参数必须在位置参数的右边
# SyntaxError: positional argument follows keyword argument
# foo(sex='男', "小郭",  age=20)

# 注意2：关键字参数不能重复赋值
# SyntaxError: keyword argument repeated
# foo(name="小郭", name="小李", sex='男', age=22)
```

7.2【重点】默认形参(缺省形参)

- 形参设置默认值则为缺省参数，也叫默认参数
- 调用函数时，如果没有传入默认参数对应的实参，则使用默认值
- 默认参数必须在普通参数的后边

```
# 形参设定默认值，称为缺省参数，也叫默认参数
# 调用函数时，如果没有传入默认参数对应的实参，则使用默认值


# 默认参数必须在普通参数的后边
def func(a, b=20, c=30):
```

```
    print(a, b, c)

# 注意：默认参数必须在普通参数的后边
# SyntaxError: non-default argument follows
default argument
# def func2(a=10, b, c):
#     print(a, b, c)

# SyntaxError: non-default argument follows
default argument
# def func3(a=10,b,c=30):
#     print(a, b, c)


# 函数调用
func(1)  # func(1,20,30)
func(1, 2)  # func(1, 2, 30)
func(1, 2, 3)  # func(1, 2, 3)
```

7.3【重点】元组型不定长参数

- **形参变量名前面加上一个*，**这个参数则为**元组型不定长参数**

```
# 函数形参变量的前面加一个*，  这个参数就是不定长参数


# 把实参的1,2,3, 包装成元组(1, 2, 3)再传递, 等价于args = (1, 2, 3)
# def func(a, b, c):
#     print(a, b, c)

# *args 不定长参数, 可以接收0~多个位置实参
def func(*args):
    print(args, type(args))


# 函数调用
func()
func(1)
func(1, 2)
func(1, 2, 3)   # 发现函数定义了不定长参数, 1. 先把1,2,3打包成元组 (1,2,3) 2. 再传递参数给args
```

7.4【重点】字典型不定长参数

- 定义参数时需要在形参名前添加`**`，则为字典型不定长参数

- 字典型可变形参必须在形参列表的最后边

```
# 函数形参变量，前面有2个*，字典型不定长参数、也叫关键字型不定长参数
# 函数内部使用，无需加*
# **kwargs，这个参数一定是放在最右边


# 把实参包装成 {'city': 'sz', 'age': 18}给kwargs传递
# kwargs = {'city': 'sz', 'age': 18}
# def foo(name, age, sex):
#     print(name, age, sex)


def foo(name, **kwargs):
    print(name, kwargs, type(kwargs))


# 实参的写法：  变量=数据，变量=数据
foo(name="rose", age=18, sex="male")
foo(name="tom")
foo("lily")
foo("mike", age=19, sex='male', city='sz', like=['sing', 'dance'])
```

```
# 1. 关键字参数必须在位置参数右边
# 2. args: 不定长的位置参数
# 3. kwargs: 不定长的关键字参数


# 扩展: 多种参数同时出现时的函数定义的模板:
def foo(name, *args, city="sz", **kwargs):
    print(name, args, city, kwargs)


foo("rose")      # rose () sz {}
foo("tom", city='bj')   # tom () bj {}
foo("tom", 18, 'male')  # tom (18, 'male')
sz {}

# tom (18, 'male') shenzhen {'cls':
'python42期', 'like': 'python'}
foo("tom", 18, 'male', city='shenzhen',
cls="python42期", like='python')
```