函数的定义：如何优雅地反复引用同一段代码?

目录

函数的用途

- 包装需要多次执行的代码片段，提高代码的重复利用率

定义函数

- def 函数名称([函数参数]):

  函数体

  [return 返回值]

- def foo():

  print("定义一个叫做foo的函数")

调用函数

```
def foo():
    ...
```

函数名后面使用圆括号，表示调用该函数：foo()

匿名函数

- 匿名函数是借用 lambda 表达式进行函数的定义和调用

```
add_1 = lambda x: x+1
print( add_1(100) )
```

相当于：

```
def add_1(x):
    return x+1
print( add_1(100) )
```

总结

1. 函数可以简化反复调用代码的复杂度
2. lambda 表达式可以精简简单函数的定义和调用
3. 函数使 Python 代码的结构化进一步增强

课后作业

请找到“飞机大战”的课程代码，将输出飞机位置的代码块改为以函数方式实现。

函数的参数：怎样实现函数与外部数据进行通信?

目录

实参与形参

让函数接收任意值：

- 函数定义：

```
def func( args )
    对 args 进行处理
```

- 函数调用：

```
func( variables )
```

实参与形参

- def func(args)

  定义变量的参数 args 是形参

- func(variables)

  调用函数的参数 variables 是实参

类型提示

- PEP484 Python 3.5 以上版本能够支持类型提示（Type Hint）功能

```
def func(变量名: 类型) -> 返回类型：
  return 变量名
```

- 类型提示仅能对开发人员起到提示作用，不能用于类型检查

https://peps.python.org/pep-0484/

位置参数

- 需要传递多个参数时，最简单的方法是按顺序传递，这种基于顺序关联参数的方式，称作位置参数

位置参数

```
def foo(argv1, argv2, argv3):
    pass

foo(one, two, three)
```

关键字参数

以内置函数 open() 为例，当你需要文件名和编码时，按照如下方式调用：

```
open(file, mode='r', buffering=-1, encoding=None,
        errors=None, newline=None, closefd=True, opener=None)

open("/path/to/file", encoding="UTF-8")
```

关键字参数

关键字参数用于函数调用时：

- 采用了不同的参数传递顺序
- 根据需要传递特定的参数

默认值

- 使用关键字参数的函数，往往参数较多
- 当用户调用函数时，希望有大量参数采用默认值
- 定义函数时，可以直接为参数指定默认值

如：open(file, mode='r', …)

总结

1. 函数的参数可以按照位置传参，也可以使用关键字方式来传参
2. 当需要多人协作开发时，可以指定参数的类型提示
3. 当参数较多时，可以使用默认值简化函数的调用

课后作业

编写一个函数，计算 PI 的近似值。函数的参数越大，PI 的值越精确。

函数的参数：当函数操作对象不固定时怎么处理?

目录

不定长参数

- 固定长度的参数可以让函数功能更明确
- 不定长度的参数可以让函数更灵活

不定长参数

电话本函数：

```
def address_book(name, *telphone, alias_name=None, **custom):


    print(f"name: {name}, tel: {telphone}, aname: {alias_name}, custom:{custom}")
```

* 接收位置参数

** 接收关键字参数

不定长参数

不定长参数的变种：

```
def address_book(name, *, alias_name):
    pass
```

* 号之后没有变量名称，只处理第一个和最后一个参数，第二个参数忽略

注意：由于最后一个是位置参数，所以参数长度被固定为 3 个

函数文档

```
def foo():
    """ 这个函数的用途、用法、
        注意事项等"""
    pass
```

- 查看文档的方法：foo.__doc__

函数内省

```
>>> dir(foo)
['__annotations__', '__call__', '__class__', '__closure__', '__code__', '__defaults__',
'__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__get__',
'__getattribute__', '__globals__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__kwdefaults__', '__le__', '__lt__', '__module__', '__name__', '__ne__', '__new__',
'__qualname__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__']
```

也可以使用：foo.__dir__

函数内省

- 函数内部有很多内置的属性，这些属性是在将一个函数定义为函数对象后，自动产生的
- 你学过的默认数字类型也可以使用 dir() 函数查看其内部定义的属性和方法，这些属性和方法有很多眼熟的名字，都是我们学过的数据类型默认方法

总结

1. 函数可以使用不定长参数增加灵活性
2. 可以增加文档，帮助使用函数的人了解你定义的复杂参数
3. 文档是通过内省方式实现的，借用字符串的形式来进行编写

课后作业

编写一个函数，使其能得到长度不等的参数。在函数执行时，应当显示该函数的参数数量和参数全部内容。

函数的返回值：如何得到函数的执行结果?

目录

return 语句

- 函数可以使用参数对输入进行处理，也可以使用输出将结果展示给最终用户
- return 语句可以将函数执行的结果赋值给变量，让输出可以继续处理，如：

```
var = func1()

var2 = func1(var)
```

返回值类型

- return 表达式定义如下：

  return［表达式］

- 如果函数内不定义 return 语句，或返回表达式为空，那么默认返回 None

  https://docs.python.org/zh-cn/3.10/reference/simple_stmts.html?highlight=return#the-return-statement

返回值类型

return 变量

return 变量[, 变量, 变量]

return 字面值

return 函数调用

return …

返回函数调用

```
def func1():
    return 1

def func2():
    return func1()

result = func2()
```

```
def func1():
    return 1

def func2():
    return func2()

result = func2()
```

返回函数调用

函数返回值调用自身：

报错: 递归错误

[Previous line repeated 996 more times]

RecursionError: maximum recursion depth exceeded

递归：程序自身调用自身的算法

三大基本算法：判断、循环、递归

返回函数调用

递归：

- 由于函数自己调用自己会导致无限循环，在编写递归函数时，要定义好停止条件
- 以求阶乘为例：

```
def  计算阶乘（初始数值）
    if 数值为 1 或者 0 :
        返回 1
    else:
        返回 初始数值*计算阶乘(数值 –1 )
```

总结

1. return 语句可以将函数执行结果赋值给变量
2. 返回值可以是基本数据类型，也可以是函数，甚至函数本身
3. 函数返回本身构成了递归，递归和判断、循环构成了基本的算法

课后作业

编写一个程序，计算圆的周长和面积。

通过用户输入的整数作为圆的半径，在程序中，需要编写两个函数：

circumference() 函数用于计算周长，area_of_circle() 函数用于计算圆的面积。

将周长和面积存入一个列表中，并将该列表在终端进行输出。

小试牛刀：如何利用函数实现电商购物车功能?

目录

购物车功能分析

- 方便客户确认购买商品名称、数量、金额、总额
- 方便客户调整商品数量
- 方便用户合并付款

购物车功能分析

简单的功能划分：

- 展示商品
- 添加商品、删除商品
- 调整商品数量
- 费用总计

- 忽略功能：库存数量、商品类目

实现商品展示

- 定义商品为列表
- 定义购物车为字典

实现商品展示

- 展示商品功能 = 展示字典的所有内容

实现费用统计

合计费用 = 单价 * 数量 + 单价 * 数量 + 单价 * 数量

- 为了避免每个商品的总价和合计费用在数量修改后出现不一致，

  应当只允许修改数量，而每个商品的总价和合计费用应在此基础上计算得到

实现购物车内商品数量修改

- 修改商品数量时，应避免选择的数量为 0 或超过库存（暂不考虑）
- 购物车中数量为 0 时，应将商品移除
- 商品数量增加或减少时，应考虑同时修改价格
- 采用自定义函数比修改字典的默认值更适合当前需求

总结

1. 购物车程序中，首要解决的是问题分解，也就是将完整功能拆分成几个小功能，再由每个函数分别实现这些小功能
2. 当多个变量相互作用时，为了将多个变量组合为一个整体处理逻辑，往往使用函数将它们组合在一起（封装）
3. 基于你的理解，使用学习过的函数、变量等功能，继续完善购物车功能

课后作业

请你继续完善购物车程序，新建另一个列表，按要求实现库存功能：

- 当某个商品被添加进购物车时，该商品库存减少
- 当某个商品在购物车中减少数量时，该商品库存增加
- 当商品库存为 0 时，不再允许增加购物车中该商品数量

避坑指南：列表作为参数传递出错了怎么办？

目录

列表类型的特殊性

- 列表是可变数据类型
- 元组的元素可以是列表
- 可变类型，作为函数的参数，导致数据处理时发生变化

列表作为函数参数

```
list1 = [ 1, 2, 3 ]
def func(list1):
    list1.append(4)

func(list1)
print(list1) # [ 1, 2, 3, 4]
func(list1)
print(list1) # [ 1, 2, 3, 4, 4]
```

总结

1. 可变数据类型不应作为函数参数
2. 列表作为函数参数，多次调用后，列表值会发生变化

课后作业

请编写函数，将一个列表中的元素按从大到小的顺序进行排序。

高阶函数：函数对象与函数调用的用法区别

目录

函数对象和函数调用

```
def foo():
    print("foo函数被执行")
```

- 函数调用：

```
foo()
```

- 函数对象：

```
var = foo
var()
```

高阶函数

- 高阶函数通常指的是 map、filter、reduce
- map、filter 是内置函数，可以直接使用
- reduce 需要通过 fuctools 库导入

高阶函数

map() 函数

map(函数，可迭代对象)

- 函数：函数对象、lambda 表达式
- 可迭代对象：列表、元组、range 等
- 将可迭代对象的每个元素作为函数参数执行

高阶函数

map() 函数

```
def add(number):
    return ( number + number )

for i in map( add, range(5) ):
    print( i )
```

也可以简化成 list(map(lambda x: x+x, range(5)))

高阶函数

filter() 函数

filter(函数，可迭代对象)

- 过滤掉可迭代对象中返回值不为 True 的元素
- filter 函数返回一个可迭代对象

高阶函数

filter() 函数

当 function 不是 None 的时候为

(item for item in iterable if function(item))；

function 是 None 的时候为

(item for item in iterable if item)

高阶函数

reduce() 函数

```
from functools import reduce
```

reduce(函数，可迭代对象)

- 对可迭代对象进行累积

偏函数

- 偏函数：固定某个参数，形成新的函数

```
from functools import partial

new_func  = partial( 旧的函数对象, 被固定的参数 )

new_func( 参数 )
```

偏函数

```
int(数值, base=进制)

from functools import partial

int_16 = partial(int, base=16)

int_16("ff")

# int_16("ff") 等同于 int("ff", base=16)
```

总结

1. 函数对象表示函数本身，可以赋值，也可以作为其他函数的参数和返回值

2. 函数参数为函数对象时，可以形成更复杂的函数。Python 定义了三个高阶函数：map、filter、reduce

3. 固定函数参数可以使用偏函数，偏函数的参数也是函数对象

课后作业

使用高阶函数合并两个列表，使每个列表对应位置的元素相加，得到新的列表，

如：[1, 3, 5, 7, 9] 和 [2, 4, 6, 8, 10] 相加后得到 [3, 7, 11, 15, 19]。

装饰器：函数嵌套的定义与调用的区别

目录

变量作用域

- LEGB 规则：

  Local 本地变量

  Enclosed 闭包变量

  Global 全局变量

  Builtin 内置变量

变量作用域

- 全局变量作用域包含函数，因此函数内可以使用全局变量
- 函数内的变量属于局部变量，在函数外调用会提示变量未被定义错误

闭包

- 函数内的再次定义的内部函数形成闭包
- 闭包作用域之内，内部函数可以访问外部函数的变量

闭包

```
def func_out():
    var_out = 1
    def func_in():
        return var_out
    return func_in

print( func_out()() )
```

装饰器语法

- 外部函数和内部函数同时定义，往往出现在两个不同的工作环节，即：

  A 开发人员定义外部函数，B 开发人员定义内部函数

- 定义和调用都不优雅

  （定义内部函数的人要修改外部函数定义，调用要用两个圆括号）

- 引入 @ 语法实现函数嵌套定义

装饰器语法

- 以计算函数运行时间为例

```
import time

def func():
    print("func 函数开始执行")
    time.sleep(1)
    print("func 函数执行结束")

start = time.time()
func()
stop = time.time()
print(f"func 函数一共执行了{int(stop-start)}秒")
```

装饰器语法

- 以计算函数运行时间为例

```
import time

def time_it(func):
    def wrapper():
        start = time.time()
        func()
        stop = time.time()
        print(f"func 函数一共执行了{int(stop-start)}秒")
    return wrapper
```

装饰器语法

```
@time_it
def work():
    print("func函数开始执行")
    time.sleep(1)
    print("func函数执行结束")


work()
```

自带装饰器

- Python 内置的装饰器使用了 functools 包
- 常用的有：

  @lrucache # 缓存

  @wraps # 被装饰函数保持原对象不变

  https://docs.python.org/zh-cn/3.10/library/functools.html

总结

1. 变量作用域保证同名，但不同作用范围不会出现引用错误
2. 装饰器可以使函数功能更明确，更容易实现函数抽象
3. 系统内置装饰器可以辅助用户编写更复杂的函数功能

课后作业

定义一个装饰器，可以让函数运行 5 遍，并输出平均运行时间。

THANKS