

Python程序设计基础

吴振鹏

wuzhp26@mail2.sysu.edu.cn

赖衍亦

laiyy28@mail2.sysu.edu.cn



参考资料与建议阅读

- 《Python编程：从入门到实践》
- 《人工智能（第3版）》附录A
- 超算习堂-在线实训
- <https://www.python.org>
- <https://www.runoob.com/python3/python3-tutorial.html>



目录

- 1 初识Python
- 2 简单数据类型
- 3 控制结构
- 4 复杂数据结构与操作
- 5 函数与类
- 6 文件与异常
- 7 模块与库
- 8 总结



目录

- 1 初识Python
- 2 简单数据类型
- 3 控制结构
- 4 复杂数据结构与操作
- **5 函数与类**
- 6 文件与异常
- 7 模块与库
- 8 总结



传递实参

- 对某些数据类型来说，在函数内部对传入变量所做的修改，会导致函数外的值同时发生修改，产生副作用。
 - 在目前学过的类型中，**列表**和**字典**符合这种情况
- 对于数值、字符串等，可通过返回值将函数内的值传至函数外。



传递列表

- 将列表传递给函数

```
def greet_users(names):  
    for name in names:  
        msg = "Hello, " + name.title() + "!"  
        print(msg)  
  
username = ['hannah', 'ty', 'margot']  
greet_users(username)
```

```
Hello, Hannah!  
Hello, Ty!  
Hello, Margot!
```



传递列表：在函数中修改列表

- 在函数中对传入列表所做的任何修改都是永久性的

```
def modify_list(source, index, target):  
    source[index] = target  
source = [1, 2, 3]  
index = 2  
target = 5  
modify_list(source, index, target)  
print("The modified source is:", source)  
index = 1  
target = [1, 3]  
modify_list(source, index, target)  
print("The modified source is:", source)
```

The modified source is: [1, 2, 5]
The modified source is: [1, [1, 3], 5]



传递列表：防止函数修改列表

- 有时候，需要防止函数修改列表。
- 向函数传递列表副本，可保留函数外原始列表的内容：

```
modify_list(source[:], index, target)
```

- `function_name(list_name[:])`
- 利用切片创建副本
- 除非有充分的理由需要传递副本，否则还是应该将原始列表传递给函数。
 - 避免花时间和内存创建副本，从而提高效率
 - 在处理大型列表时尤其如此



传递任意数量的实参

- 形参前加*号，可传递任意数量的实参

```
def make_pizza(*toppings):  
    print("\nMaking a pizza ...")  
    print("Toppings:")  
    for topping in toppings:  
        print("- " + topping)  
    print(toppings)  
  
make_pizza('pepperoni')  
make_pizza('mushrooms', 'peppers', 'cheese')
```

```
Making a pizza ...  
Toppings:  
- pepperoni  
( 'pepperoni', )  
  
Making a pizza ...  
Toppings:  
- mushrooms  
- peppers  
- cheese  
( 'mushrooms', 'peppers', 'cheese' )
```

- 结合使用位置实参和任意数量实参

```
def make_pizza(size, *toppings):
```

Python先匹配位置实参和关键字实参，再将余下的实参收集到最后一个形参中



传递任意数量的关键字实参

- 形参前加**号，可传递任意数量的关键字实参

user_info是一个字典

```
def build_profile(first, last, **user_info):  
    profile = {}  
    profile['first_name'] = first  
    profile['last_name'] = last  
    for key, value in user_info.items():  
        profile[key] = value  
    return profile
```

```
user_profile = build_profile('albert', 'einstein', location='princeton', field='physics')  
print(user_profile)
```

```
{'first_name': 'albert', 'last_name': 'einstein', 'location': 'princeton', 'field': 'physics'}
```



类

- 面向对象编程
- 将现实世界中的事物和情景编写成类，并定义通用行为
- 实例化：基于**类**创建实例（对象）



定义类

- 创建Dog类

```
class Dog():  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    def sit(self):  
        print(self.name.title() + " is now sitting.")  
  
    def roll_over(self):  
        print(self.name.title() + " rolled over!")
```

- 方法__init__()

- 构造函数，创建新对象时自动调用
- 开头、末尾各有两个下划线
- 类中的成员函数成为方法

- self

- 要写 在所有方法参数列表的第一位
- 指代这个对象自身
- 以self为前缀的成员变量可供类中所有方法使用，称为属性
- 通过self. 变量名，可访问、创建与修改属性



类的实例化：对象

- 使用Dog类

```
class Dog():  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    def sit(self):  
        print(self.name.title() + " is now sitting.")  
  
    def roll_over(self):  
        print(self.name.title() + " rolled over!")
```

- 创建对象

```
my_dog = Dog('willie', 6)
```

- 访问属性 Python默认是公有属性

```
print(my_dog.name.title())  
print(my_dog.age)
```

```
Willie  
6
```

- 调用方法 不需要传实参给self

```
my_dog.sit()  
my_dog.roll_over()
```

```
Willie is now  
sitting.  
Willie rolled over!
```

属性的修改



```
class Car():
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        long_name = str(self.year) + " " + self.make + " " + self.model
        return long_name.title()

    def read_odometer(self):
        print("This car has " + str(self.odometer_reading) + " miles on it.")

    def updata_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        self.odometer_reading += miles
```

```
my_new_car = Car("audi", "a4", 2016)
print(my_new_car.get_descriptive_name())
my_new_car.read_odometer()
```

2016 Audi A4

This car has 0 miles on it.

- 直接修改属性的值 破坏了封装

```
my_new_car.odometer_reading = 500
my_new_car.read_odometer()
```

This car has 500 miles on it.

- 通过方法修改属性的值

```
my_new_car.updata_odometer(23500)
my_new_car.read_odometer()
```

This car has 23500 miles on it.

- 通过方法递增属性的值

```
my_new_car.increment_odometer(100)
my_new_car.read_odometer()
```

This car has 23600 miles on it.



继承

- 子类是父类的特殊版本
- 子类继承父类的所有属性和方法

```
class ElectricCar(Car):  
    def __init__(self, make, model, year):  
        super().__init__(make, model, year)  
  
my_tesla = ElectricCar('tesla', 'model s', 2016)  
print(my_tesla.get_descriptive_name())
```

class 子类名(父类名)

2016 Tesla Model S

- `super()`是一个特殊函数，在子类中通过`super()`指向父类

- 可以给子类定义自己的属性和方法
- 子类可以重写父类的方法

```
class ElectricCar(Car):  
    ...  
    def get_descriptive_name(self):  
        return "Electric" + super().get_descriptive_name()  
  
my_tesla = ElectricCar('tesla', 'model s', 2016)  
print(my_tesla.get_descriptive_name())
```

Electric 2016 Tesla Model S

- 实例可作为属性（类的成员）



函数与类：小结

- 函数
 - 定义函数：语法
 - 返回值：允许多个返回值，允许任意类型
 - 传递实参：关键字实参，默认值，副作用，传递任意数量的实参
- 类
 - 定义类：构造函数、self
 - 类实例化为对象：创建对象、访问属性、调用方法、修改属性
 - 继承



练习：函数与类

9. 魔术师，了不起的魔术师与不变的魔术师：

- a) 创建一个包含魔术师名字的列表，并将其传递给show_magicians()函数，这个函数打印列表中每个魔术师的名字。
- b) 编写一个名为make_great()的函数，对魔术师列表进行修改，在每个魔术师的名字中都加入字样“the Great”。调用show_magicians()，确认魔术师列表确实变了。
- c) 在调用make_great()时，向它传递魔术师列表的副本。由于不想修改原始列表，请返回修改后的列表，并将其存储到另一个变量中。分别对这两列表调用show_magicians()，确认一个列表包含的是原来的魔术师名字，而另一个列表包含的是添加了字样“the Great”的魔术师名字。



练习：函数与类

10. 餐馆、就餐人数与冰淇淋小店：

- a) 创建一个名为Restaurant的类，其方法__init__()设置两个属性：restaurant_name和cuisine_type。创建一个名为describe_restaurant()的方法和一个名为open_restaurant()的方法，其中前者打印前述两项信息，而后者打印一条消息，指出餐馆正在营业。
- b) 添加一个名为number_served的属性，并将其默认值设置为0。根据这个类创建一个名为restaurant的实例；打印有多少人在这家餐馆就餐过，然后修改这个值并再次打印它。添加一个名为set_number_served()的方法，它让你能够设置就餐人数；调用这个方法并向它传递一个值，然后再次打印这个值。添加一个名为increment_number_served()的方法，它让你能够将就餐人数递增；调用这个方法并向它传递一个这样的值：你认为这家餐馆每天可能接待的就餐人数。
- c) 冰淇淋小店是一种特殊的餐馆。编写一个名为IceCreamStand的类，让它继承Restaurant类。添加一个名为flavors的属性，用于存储一个由各种口味的冰淇淋组成的列表。编写一个显示这些冰淇淋的方法。创建一个IceCreamStand实例，并调用这个方法。



目录

- 1 初识Python
- 2 简单数据类型
- 3 控制结构
- 4 复杂数据结构与操作
- 5 函数与类
- **6 文件与异常**
- 7 模块与库
- 8 总结



读取文件

```
1 3.1415926535
2 8979323846
3 2643383279
```

- 读取整个文件：read()

```
with open('pi_digits.txt') as file_object:
    contents = file_object.read()
    print(contents)
```

```
3.1415926535
8979323846
2643383279
```

- 函数open的参数为**文件路径**
 - 相对路径与绝对路径都可以
- 关键字with在不再需要访问文件后将文件关闭
 - 此时无需调用close()

- 逐行读取：readlines()

```
with open('pi_digits.txt') as file_object:
    for line in file_object.readlines():
        print(line)
```

```
3.1415926535
8979323846
2643383279
```

- 得到一个列表
- 空白行：
- 文件中每行末尾有一个换行符，而print语句也会加上一个换行符。可以使用rstrip()去掉。



写入文件

open()的模式参数（第二个）：

'r': 读取模式（默认）

'w': 写入模式

'a': 追加模式

...

只能将字符串写入文件。

如需换行，记得写换行符。

- 写入空文件

```
filename = 'programming.txt'
```

```
with open(filename, 'w') as file_object:
```

```
    file_object.write("I love programming.\n")
```

```
    file_object.write("I love creating new games.\n")
```

```
1 I love programming.  
2 I love creating new games.  
3
```

- 追加到文件

```
filename = 'programming.txt'
```

```
with open(filename, 'a') as file_object:
```

```
    file_object.write("I also love finding meaning in large datasets.\n")
```

```
    file_object.write("I love creating apps that can run in a browser.\n")
```

```
1 I love programming.  
2 I love creating new games.  
3 I also love finding meaning in large datasets.  
4 I love creating apps that can run in a browser.  
5
```



使用模块json存储数据

- 支持将简单的Python数据结构转储到文件中，并在程序再次运行时加载该文件中的数据。
 - `json.dump()`: 保存
 - `json.load()`: 读取
- 序列化 (Serialization)
 - 将对象的状态信息转换为可以存储或传输的形式过程。
 - 反序列化
- 除了json模块，还有pickle等序列化模块，支持将Python对象存储在文件中，以供未来读取。

```
import json
```

```
data1 = {  
    'date': 224,  
    'room': 'D402',  
    'time': '14:20-16:00'  
}
```

```
with open('data.json', 'w') as f:  
    json.dump(data1, f)
```

```
with open('data.json', 'r') as f:  
    data2 = json.load(f)
```

```
print(data2)
```

```
{ 'date': 224, 'room': 'D402', 'time': '14:20-16:00' }
```



异常

- Python使用被称为**异常**的特殊对象来管理程序**执行期间**发生的错误。每当Python运行发生错误时，它都会创建一个异常对象。
- 如果你未对异常进行处理，程序块将**在错误处停止**，并显示一个traceback，其中包含有关异常的报告，指出发生了**哪种异常**。

```
>>> 5/0
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
ZeroDivisionError: division by zero
```

- 使用try-except代码块处理异常或显示你编写的友好的错误信息，此时，即使出现异常，程序也将继续运行。



使用try-except代码块

- 处理FileNotFoundError异常

```
filename = 'alice.txt'

try:
    with open(filename) as f_obj:
        contents = f_obj.read()
except FileNotFoundError:
    print("Sorry, the file " + filename + " does not exist.")
```

```
Sorry, the file alice.txt does not  
exist
```

- 如果try代码块中的代码正常运行，将跳过except代码块；如果代码出错，Python查找并运行对应类型的except代码块中的代码。



使用try-except-else代码块

- 处理ZeroDivisionError异常

```
first_number = input("\nFirst number: ")
second_number = input("Second number: ")
try:
    answer = int(first_number) / int(second_number)
except ZeroDivisionError:
    pass
else:
    print(answer)
print("Finished!")
```

```
First number: 5
Second number: 0
Finished!
```

```
First number: 5
Second number: 2
2.5
Finished!
```

- 仅在try代码块成功执行时才运行的代码，应放在else代码块中。
- pass语句：在代码块中使用，指示Python什么都不做。



自定义异常类

- 只要继承了Exception, 就可以定义自己的异常类

```
class FileEmptyException(Exception):  
    def __init__(self, file_name):  
        self.file_name = file_name  
  
    def __str__(self):  
        return "file: " + self.file_name + " is empty!"
```

```
try:  
    with open("data.json") as f:  
        data = f.read()  
    if len(data) == 0:  
        raise FileEmptyException("data.json")  
    else:  
        print(data)  
  
except FileEmptyException as e:  
    print(e)
```



文件与异常：小结

- 文件处理
 - 读取文件：读取整个文件、逐行读取
 - 写入文件：写入空文件、追加到文件
 - 使用模块json存储数据
- 异常处理
 - try-except
 - try-except-else
 - pass语句
 - 自定义异常类



练习：文件与异常

11. 访问Project Gutenberg (<https://gutenberg.org/>), 并找一些你想分析的图书。下载这些作品的文本文件或将浏览器中的原始文本复制到文本文件中。编写一个程序读入这些文本文件, 分别对每个文本文件进行初步的统计分析:
 - a) 统计文本中包含多少个单词 (提示: 使用字符串方法split()与函数len())
 - b) 统计单词“the”出现了多少次 (提示: 使用字符串方法count()与lower())
12. 请你建立一个字典, 并以一个字典中不存在的键访问该字典。
 - a) 观察报错信息, 识别程序发生了哪种异常。
 - b) 改写代码, 使程序可以处理这种异常。



目录

- 1 初识Python
- 2 简单数据类型
- 3 控制结构
- 4 复杂数据结构与操作
- 5 函数与类
- 6 文件与异常
- **7 模块与库**
- 8 总结



将函数与类存储在模块中

- 将函数与类存储在被称为**模块**的独立文件中，与主程序分离。
- 模块是扩展名为.py的文件，包含要导入到程序中的代码。

- **模块的导入**

模块名，即py模块文件的文件名

- import 模块名
 - 调用方式：模块名.函数名或类名
- from 模块名 import 函数名或类名
 - 调用方式：函数名或类名
 - 可以同时导入多个函数或类，中间用逗号分隔
- import 模块名 as 模块别名
 - 调用方式：模块别名.函数名或类名
- from 模块名 import 函数名或类名 as 函数或类的别名
 - 调用方式：函数或类的别名
- from 模块名 import *
 - 调用方式：函数名或类名
 - 如遇相同名称容易造成覆盖，不推荐



Python中的“main函数”

- swap.py

name和main的前与后，均有两个下划线。

```
def swap(a, b):  
    return b, a  
  
if __name__ == '__main__':  
    a = 224  
    b = 'Good day!'  
    print(a, b)  
    a, b = swap(a, b)  
    print(a, b)  
    a, b = b, a  
    print(a, b)
```

```
224 Good day!  
Good day! 224  
224 Good day!
```

- 导入一个模块时，该模块文件的无缩进代码将自动执行。
 - 例如，若当前“main”下的代码没有缩进在if中，则import swap时，这些代码全部都会执行一次。
- 因此，在编写自己的模块时，模块测试代码等要记得缩进于if __name__ == '__main__':



Python标准库

- Python标准库是一组Python自带的模块，例如：
 - math
 - random
 - copy
 - csv
 - heapq
 - time
 - os
 - multiprocessing
 - collections
 - unittest
 - ...
- 了解Python标准库：Python Module of the Week
 - <https://pymotw.com/>



外部模块： 安装

- 使用pip安装Python包
 - pip是一个负责为你下载并安装Python包的程序，大部分较新的Python都自带pip
- 安装命令：
 - `pip install 包的名称`
- 类似的，如果计算机同时安装了Python2和Python3，则需使用：
 - `pip3 install 包的名称`



一些常用的包

- 交互实时编程: jupyter notebook
- 数据分析、计算与可视化: numpy、scipy、pandas、matplotlib
- 机器学习: scikit-learn
- 深度学习: pytorch、tensorflow、keras
- 文本挖掘: genism
- 推荐学习: 超算习堂-在线实训



模块与库：小结

- 模块导入
- 编写自己的模块
- Python标准库
- 外部模块



练习：模块与库

- 在超算习堂中，完成“NumPy入门”在线实训





目录

- 1 初识Python
- 2 简单数据类型
- 3 控制结构
- 4 复杂数据结构与操作
- 5 函数与类
- 6 文件与异常
- 7 模块与库
- **8 总结**



总结

- 至此，你已经对Python 3的基本语法有了足够的了解。
 - 学无止境，还有一些高级语法和丰富的第三方包等待你探索…
 - 在实践中，你还可能会遇到很多无法预料的报错和坑…
- Python官网：Python is a programming language that lets you **work quickly** and **integrate systems more effectively**.
 - “胶水语言”，完成不同模块间的简单处理
 - 例如：将A库的输出经过一定形式的转换送进B库做处理
 - 用numpy处理得到数据矩阵，送进sklearn中进行机器学习模型训练与预测

为什么只有python成了胶水语言？ <https://www.zhihu.com/question/402762138>

实验任务



思考题：

- 1. 如果用列表作为字典的键，会发生什么现象？用元组呢？
- 2. 在本课件第2章和第4章提到的数据类型中，哪些是可变数据类型，哪些是不可变数据类型？试结合代码分析。
 - 可变/不可变数据类型：变量值发生改变时，变量的内存地址不变/改变。
 - 提示：① 你可能会用到id()函数。② Python的赋值运算符（=）是引用传递。

第2周作业(第1页/共4页)



给定 `student_data.txt` 文本文件, 每一行是一名学生的信息, 从左到右分别是该学生的姓名, 学号, 性别和年龄, 每个属性以空格间隔. 数据类型如下:

student_data.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
name: str # 姓名
stu_num: str # 学号
gender: str # 性别, "M"为男性, "F"为女性
age: int # 年龄
```

Aaron 243 M 18

Eric 249 M 19

Alex 812 M 19

Leo 092 M 17

Sam 230 F 18

Ruth 942 M 19

Beryl 091 F 20

Cynthia 920 F 19

编写 `StuData` 类, 须有以下方法:

- 构造函数(即 `__init__`), 以文件名(`str`类型, 带 `.txt`后缀)为输入, 读取文件中的学生信息, 存储到类成员 `data` 中. `data` 的数据类型为 `list`, 其中每一个学生的信息以列表方式存储. 例如, 读入一行学生信息 "Aaron 243 M 18", 则 `data` 变为

```
[["Aaron", "243", "M", 18]]
```

第2周作业(第2页/共4页)



再读入学生信息"Eric 249 M 19", 则 `data` 变为

```
[["Aaron", "243", "M", 18], ["Eric", "249", "M", 19]]
```

- `AddData`方法, 以单个学生的信息作为输入, 存储到 `data` 中. 调用该方法的参数形式为学生属性的4个关键字实参. 例如, 执行 `self.AddData(name="Bob", stu_num="003", gender="M", age=20)`后, `data`变为

```
[["Aaron", "243", "M", 18], ["Eric", "249", "M", 19], ["Bob", "003", "M", 20]]
```

第2周作业(第3页/共4页)



- **SortData**方法，以学生某个属性(**str**类型，是'**name**', '**stu_num**', '**gender**', '**age**'的其中之一)作为输入，将**data**按该属性从小到大排序.可以假定不会输入非学生属性的字符串.例如，执行**self.Sort('stu_num')**后，**data**的学生信息按学号从小到大排序，变为

```
[["Bob", "003", "M", 20], ["Aaron", "243", "M", 18], ["Eric", "249", "M", 19]]
```

- **ExportFile**方法，以导出的文件名(**str**类型，带**.txt**后缀)为输入，新建一个**txt**文件，将**data**中的数据按当前列表顺序导出到该文件内，格式同原**student_data.txt**文本文件，即"姓名学号性别年龄",并存储到目标文件夹.例如，调用**self.ExportFile('new_stu_data.txt')**,则将**data**中数据导出到**new_stu_data.txt**文件.

第2周作业(第4页/共4页)



提示

- 1.学号信息数据类型为str而不是int.可以假定学号都由3个0-9数字组成.
- 2.可以假设每个学生有且只有这4个属性，且不会缺省.
- 3.可以在类中编写其他辅助方法，也可以在同一个代码文件中编写其他函数或类供自己调用.
- 4.本次作业中，类方法的输入参数名可自定义，但参数数据类型需保证测试程序正常运行.



实验提交

- 提交到课程网站（超算习堂）中对应的课程作业，并**注意网站上公布的截止日期**
- 提交格式：提交一个命名为“学号_姓名.zip”的压缩包，压缩文件下包含三部分：code文件夹、result文件夹和实验报告pdf文件
 - 实验报告是pdf格式，命名为：**学号_姓名.pdf**
 - **code**文件夹：存放实验代码，一般有**多个代码文件**的话需要有readme
 - **result**文件夹：存放上述提到的结果文件（不是每次实验都需要交result，**如果没有要求提交结果，则不需要result文件夹**）
 - “学号_姓名”样例：20*****_wangxiaoming
- 如果需要更新提交的版本，则在后面加_v1, _v2。如第一版是“学号_姓名.zip”，第二版是“学号_姓名_v1.zip”，依此类推
- **截止日期：2024年3月11日24点**

Thanks!