



5-1.py

定义函数

- 最简单的函数结构

```
def greet_user():  
    print("Hello!")
```

```
greet_user()
```

Hello!

- 函数定义以关键字def开头
- 函数名、括号
- 定义以冒号结尾
- 紧跟的所有缩进行构成函数体
- 函数调用

- 向函数传递参数

```
def greet_user(username):  
    print("Hello, " + username.title() + "!")
```

```
greet_user('zachary')
```

Hello, Zachary!

- 变量username是一个形式参数
- 值'zachary'是一个实际参数



5-1.py

返回值

- 函数可以处理一组数据，并返回一个或一组值

```
def get_formatted_name(first_name, last_name, middle_name=""):
    if middle_name:
        full_name = first_name + ' ' + middle_name + ' ' + last_name
    else:
        full_name = first_name + ' ' + last_name
    return full_name.title()
```

通过默认值让实参变成可选的
Python将非空字符串解读为True
注意：这里是两个单引号

- 用一个变量存储返回的值，或直接使用返回的值

```
musician = get_formatted_name('jimi', 'hendrix')
print(musician)
print(get_formatted_name('john', 'hooker', 'lee'))
```

Jimi Hendrix
John Lee Hooker



5-1.py

返回值： 返回多个值

- 函数可以处理一组数据， 并返回一个或一组值

```
def get_formatted_name(first_name, last_name):  
    return first_name.title(), last_name.title()
```

- 用多个变量存储返回的值

```
first_name, last_name = get_formatted_name('jimi', 'hendrix')  
print(first_name, last_name)
```

Jimi Hendrix

- 返回的是其实是元组

```
name = get_formatted_name('jimi', 'hendrix')  
print(name)
```

('Jimi', 'Hendrix')



5-1.py

返回值： 返回字典

- 函数可以返回任何类型的值， 包括列表和字典等复杂的数据结构

```
def build_person(first_name, last_name, age=""):  
    person = {'first': first_name, 'last_name': last_name}  
    if age:  
        person['age'] = age  
    return person
```

```
musician = build_person('jimi', 'hendrix', age=27)  
print(musician)
```

```
{'first': 'jimi', 'last_name': 'hendrix', 'age': 27}
```



5-1.py

传递实参

```
def describe_pet(pet_name, animal_type='dog'):  
    """show descriptive information of a pet"""  
    print("\nI have a " + animal_type + ".")  
    print("The " + animal_type + "'s name is " + pet_name.title() + ".")
```

这里的注释称作函数的**文档字符串**，描述了函数的功能

- 位置实参：基于实参的顺序，将实参关联到函数定义中的形参

```
describe_pet('harry', 'cat')
```

I have a cat.
My cat's name is Harry.

- 默认值：具有默认值的形参需排列在参数列表的后面

```
describe_pet('willie')
```

I have a dog.
My dog's name is Willie.

- 关键字实参：无需考虑实参顺序

```
describe_pet(animal_type='dog', pet_name='willie')  
describe_pet('willie', animal_type='dog')
```



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。



5-1.py

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!
```



5-1.py

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 print_models(unprinted_designs, completed_models):
```

```
    while unprinted_designs:
```

```
        current_design = unprinted_designs.pop()
```

```
        print("Printing model: " + current_design)
```

```
        completed_models.append(current_design)
```

```
unprinted_designs = ['iphone case', 'robot pendant', 'dodecahedron']
```

```
completed_models = []
```

```
print_models(unprinted_designs, completed_models)
```

```
print("The following models have been printed:\n", completed_models)
```

```
Printing model: dodecahedron
```

```
Printing model: robot pendant
```

```
Printing model: iphone case
```

```
The following models have been printed:
```

```
['dodecahedron', 'robot pendant', 'iphone case']
```




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：

```
print_models(unprinted_designs[:], completed_models)
```

- `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



5-1.py

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

5-1.py

- 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）



5-2.py

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



5-2.py

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.



5-2.py



5-2.py

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



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。



6-1.py

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, 如果该文件已存在则打开文件, 并从开头开始编辑, 即原有内容会被删除。如果该文件不存在, 创建新文件。

'a': 追加模式. 如果该文件已存在, 文件指针将会放在文件的结尾。也就是说, 新的内容将会被写入到已有内容之后。如果该文件不存在, 创建新文件进行写入。

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
```



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。



6-2.py

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。



6-2.py

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。



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
- 异常处理
 - try-except
 - try-except-else
 - pass语句



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

```
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)
```

name和
main的
前与后,
均有两个
下划线。

```
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



实验报告提交

报告提交要求

- ❑ 大家的作业通过连接ftp服务器上传
- ❑ 主机IP: 222.200.177.152
- ❑ 端口: 1021
- ❑ 用户: ftpstu
- ❑ 密码: 123456



- 第1周的【实验报告】提交到【hw1_report】文件夹, 命名格式为【hw1_学号_姓名拼音.pdf】, 例如"hw1_21000000_zhangsan.pdf";
【源代码文件】提交到【hw1_code】文件夹, 命名格式为【hw1_学号_姓名拼音.py】, 例如"hw1_21000000_zhangsan.py". 作业提交后不可修改, 若要提交新版本请在文件名加后缀"_v1", "_v2"等, 例如"hw1_21000000_zhangsan_v1.py".
- 第2周的【实验报告】提交到【hw2_report】文件夹, 命名格式为【hw2_学号_姓名拼音.pdf】, 例如"hw2_21000000_zhangsan.pdf";
【源代码文件】提交到【hw2_code】文件夹, 命名格式为【hw2_学号_姓名拼音.py】, 例如"hw2_21000000_zhangsan.py". 作业提交后不可修改, 若要提交新版本请在文件名加后缀"_v1", "_v2"等, 例如"hw2_21000000_zhangsan_v1.py".



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", 并存储在当前文件夹. 例如, 调用 `self.ExportFile('new_stu_data.txt')`, 则将 `data`中数据导出 `new_stu_data.txt`文件到当前文件夹.



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
5. 调试代码时请将 `student_data.txt` 文件与代码文件放到同一文件夹中, 以避免不必要的bug. 提交代码时只提交一个 `.py` 代码文件, 请不要提交其他文件.

□hw1截止日期: 2023年3月12日23:59

□hw2截止日期: 2023年3月19日23:59