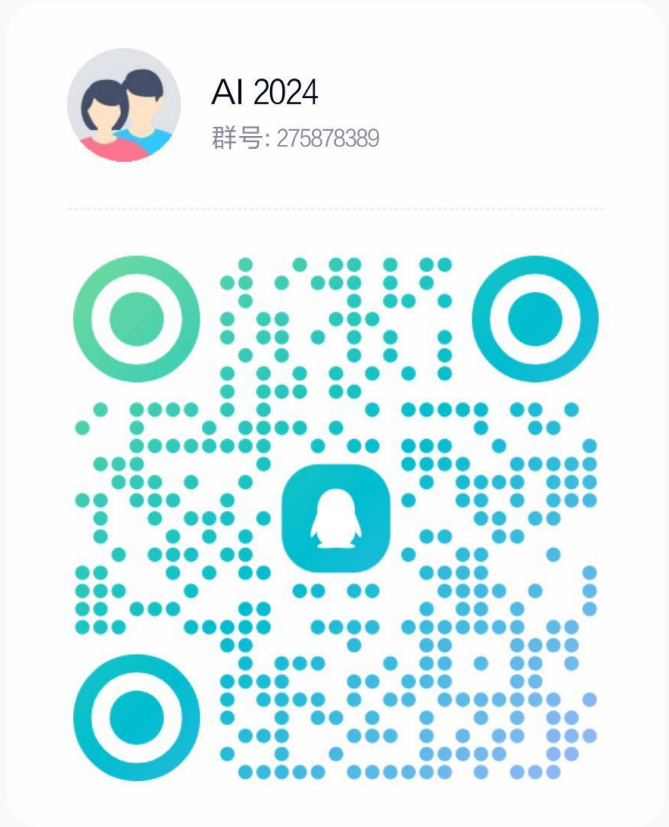
人工智能实验

2024年春季

课程信息

- 课程邮箱 ai_course_2024@163.com
- 课程Q群 275878389



实验课程要求

实验课程内容:

- 由助教讲解实验内容
- 验收前一次的实验内容（包括公式推导、代码解释、现场运行代码产生结果等）
- 会进行考勤

实验课程要求:

- 实验需要一定的数学基础以及编程基础（公式的推导以及代码的实现）
- 编程语言使用Python/C++/Java
  - 不能使用现有算法高级库（除非助教特别说明），否则扣分。
- 禁止抄袭（代码和实验报告都禁止抄袭，若被发现后果严重）

实验课程安排（暂定）

实验成绩评定方法（暂定）：

本学期预计共8个平时作业+4个项目。

实验成绩（100%） = 平时作业（8*5%=40%）
+ 考勤与课堂表现（10%）
+ 项目（2*10% + 2*15% =50%）

实验课程安排（暂定）

周次	内容	内容要点
1 2	Python程序设计基础	python
3 4	知识表示和推理：基于归结原理的推理系统	归结
5	知识表示和推理：常用工具介绍	Prolog
6 7	A*搜索算法	A*，IDA*
8	博弈树搜索	alpha-beta剪枝
9	智能规划	STRIPS planner

实验课程安排（暂定）

周次	内容	内容要点
10	机器学习1	Knn及朴素贝叶斯
11	机器学习2	Kmeans
12	机器学习3	神经网络
13	机器学习4	卷积神经网络
14 15	机器学习5	强化学习
16 17 18	人工智能专题应用项目	NLP的前沿应用

实验报告要求

- 实验报告可使用Word/Markdown/Latex等撰写，以**pdf格式**提交，可参考q群的模板与实验报告编写建议，应包含如下内容：
  - (1) 算法原理：用自己的话解释一下自己对算法/模型的理解（不可复制PPT和网上文档内容）
  - (2) 伪代码：伪代码或者流程图（注意简洁规范清晰，包含关键步骤）
  - (3) 关键代码展示：可截图或贴文本并对每个模块进行解释，包括代码+注释
  - (4) 创新点&优化：如果有的话，分点列出自己的创新点（加分项）
  - (5) 实验结果展示：基础算法的结果&(4)中对应分点优化后的算法结果+**分析**
  - (6) 思考题：PPT上写的思考题（如有）一般需要在报告最后写出解答
  - (7) 参考资料：参考的文献、博客、网上资源等需规范引用，否则涉嫌抄袭

实验提交

- 统一提交到邮箱 ai_course_2024@163.com
- 提交格式：提交一个命名为”E实验编号_学号.zip”的压缩包，含：
  - 实验报告：pdf格式，命名为：**E实验编号_学号.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）
- 如果需要更新提交的版本，则在后面加_v1，_v2。
- 示例：第1次实验“E1_12345678.zip”，第二版是“E1_12345678_v1.zip”，以此类推。
- 提交deadline：小作业如无额外说明，则在下一周课前一天提交（即周一23点59分）；项目一般2-3周时间完成。

第一次实验任务

实验1 最短路径算法

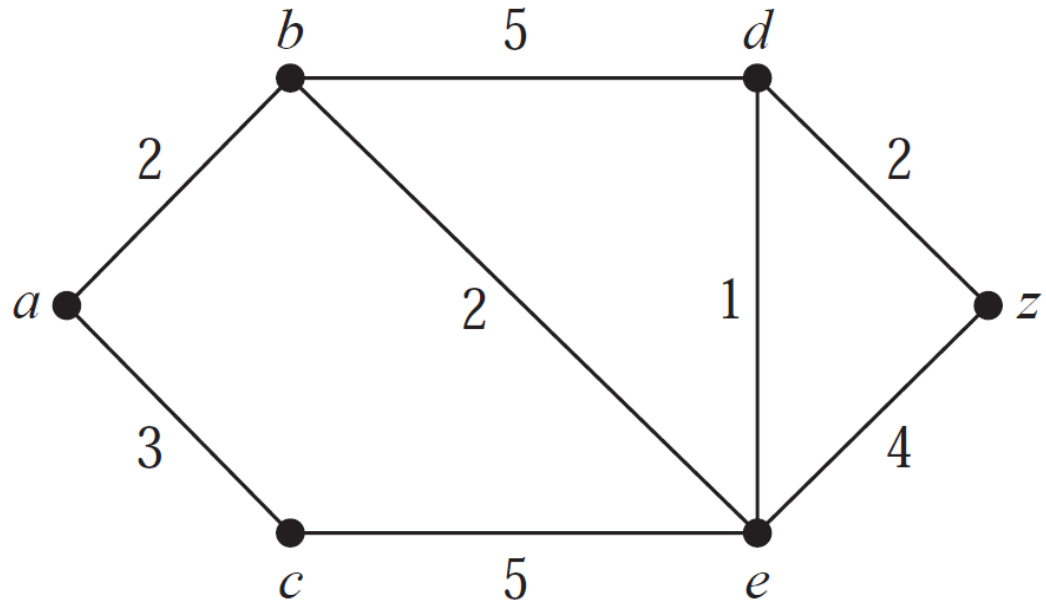
- 给定无向图，及图上两个节点，求其最短路径及长度
- 要求：使用Python实现，至少实现Dijkstra算法
- DDL：**3月11号23:59**，即给两周时间完成

- 输入（统一格式，便于之后的验收）
  - 第1行：节点数m 边数n（中间用空格隔开，下同）；
  - 第2行到第n+1行是边的信息，每行是：节点1名称 节点2名称 边权；
  - 第n+2行开始可接受循环输入，每行是：起始节点名称 目标节点名称。
- 输出（格式不限）
  - 最短路径及其长度。

实验1 最短路径算法

- 样例

```
6 8
a b 2
a c 3
b d 5
b e 2
c e 5
d e 1
d z 2
e z 4
a z
```



实验1 提示

- 使用s=input()这种格式输入时，每次读取一行。
- 回忆：s.split()返回将字符串s按空格切分后得到的字符串数组。
- 怎么存储数据？
  - 是否要存储某个点到其他点的距离表？怎么存？
  - 是否要存储所有点的距离表？怎么存？
  - （*进阶：numpy数组？）
- 文件输入：使用open()函数打开一个文件；txt文件可以作为一整个字符串读入（是否可以用split处理换行？）

Python程序设计基础

张宇聪

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
- https://www.python.org
- https://www.runoob.com/python3/python3-tutorial.html

目录

目录

环境配置

- conda是一种编程环境管理工具，适用于多种语言，如: Python, R, Scala, Java, Javascript, C/ C++, FORTRAN
- 安装地址：Miniconda — conda documentation

Platform	Name
Windows	Miniconda3 Windows 64-bit
	Miniconda3 Windows 32-bit
macOS	Miniconda3 macOS Intel x86 64-bit bash
	Miniconda3 macOS Intel x86 64-bit pkg
	Miniconda3 macOS Apple M1 64-bit bash
	Miniconda3 macOS Apple M1 64-bit pkg
Linux	Miniconda3 Linux 64-bit
	Miniconda3 Linux-aarch64 64-bit
	Miniconda3 Linux-ppc64le 64-bit
	Miniconda3 Linux-s390x 64-bit

选择适合自己系统的软件版本，下载并默认安装即可。

安装完成后，重启终端，键入'conda'，若出现以下界面即安装成功，命令行开头括号内为当前环境名。

```
(base) PS C:\Users\yanghl> conda
usage: conda-script.py [-h] [-V] command ...

conda is a tool for managing and deploying applications, environments and packages.

Options:

positional arguments:
  command
    clean        Remove unused packages and caches.
```

常用conda命令

- conda env list　查看conda环境列表
- conda create –n env_id python=3.9　创建新py3.9虚拟环境
- conda activate env_id　激活对应python环境
- conda deactivate　关闭对应python环境

目录

什么是Python

Life is short, you need Python.
by Bruce Eckel

- 一种编程语言
  - 开发效率高：清晰简洁的语法结构，更贴近自然语言；开发生态好…
  - 运行效率慢：语句需实时解释；变量的数据类型是动态的…[1]

- 优越的AI生态：有很多可以在AI中使用的库
  - 数据分析与计算：numpy、scipy、pandas
  - 机器学习：scikit-learn
  - 深度学习：pytorch、tensorflow、keras
  - 特定应用领域（如文本挖掘）：gensim
  - …

[1] 为什么 Python 这么慢？https://zhuanlan.zhihu.com/p/47795989

Hello World程序与运行

- Hello World程序

```
print("Hello World!")
```

```
Hello World!
```

  - 注意：一行为一条语句，而不是分号分隔

- 由Python解释器运行
  - 命令行运行：直接运行语句
  - 命令行运行：运行.py文件
  - 文本编辑器或IDE运行.py文件

Hello World程序：注释

1-1.py

- 井号（#）注释单行
- 三个单/双引号注释多行

```
# My first Python program

'''
a Hello World program
'''

"""
print a Hello World message
"""

print("Hello World!")
```

Hello World程序：变量

- 用一个变量存储字符串”Hello World!”

```
message = "Hello World!"
print(message)
```

- Python是动态类型语言，变量不需要声明类型
- 变量名
  - 变量名只能包括字母、数字和下划线;
  - 变量名不能以数字开头，不能包含空格;
  - Python关键字和函数名最好不要用作变量名。

Hello World程序：输出

1-2.py

- print函数
  - 输入参数为要打印的对象；
  - 可接收一个或多个参数；
  - sep参数，默认值为” ”（即多个输出内容之间，默认由空格分开）；
  - end参数，默认值为”\n”（即print后默认换行）。

- 多条消息的输出

```
message_1 = "Hello World!"
message_2 = 2022
print(message_1, message_2)
print(message_1, message_2, sep= "AI", end="SYSU")
```

```
Hello World! 2022
Hello World!AI2022SYSU
```

Hello World程序：用户输入

1-3.py

- 接收来自用户的输入

```
message_1 = “Hello!”
message_2 = input(“Please enter a message:\n”)
print(“Greeting:”, message_1, message_2)
```

```
Please enter a message:
SYSU
Greeting: Hello! SYSU
```

- input函数
  - 输入参数（可选）：提示字符串
  - 返回值：字符串

初识Python：小结

- Python基本概念、安装与配置
- Hello World程序与运行
- 注释
- 变量
- 输出函数print()
- 输入函数input()

目录

简单/基本数据类型

- 数字
  - 整数
  - 浮点数
  - 布尔值：True / False （注意大写）
    - int(True)返回1，int(False)返回0
- 字符串
- ※空值：

```
a = None
```

数字：整数

- 加（+）、减（-）、乘（*）、除（/）、整除（//）、幂（**）、模（%）

```
>>> 2 + 3
5
>>> 3 - 2
1
>>> 2 * 3
6
```

```
>>> 2 + 3 * 4
14
>>> (2 + 3) * 4
20
```

```
>>> 3 / 2
1.5
>>> 3 // 2
1
>>> 4 / 2
2.0
>>> 4 // 2
2
```

向下取整

```
>>> 3 ** 2
9
>>> 3 ** 3
27
>>> 10 ** 6
1000000
>>> 5 % 3
2
```

数字：浮点数

- 加（+）、减（-）、乘（*）、除（/）、整除（//）、幂（**）
- 但要注意的是，结果包含的小数位数可能是不确定的：

```
>>> 0.2 + 0.1
0.30000000000000004
>>> 3 * 0.1
0.30000000000000004
```

- 保留k位小数四舍五入：round(x, k)

```
>>> 5 / 3
1.6666666666666667
>>> 5 // 3
1
>>> round(5 / 3)
2
>>> round(5 / 3, 2)
1.67
```

数字：函数

- 绝对值
  - abs(a)
- 最大值、最小值
  - max(a, b)
  - min(a, b)
- math模块：用“import math”导入
  - math.floor(a)
  - math.ceil(a)
  - …

```
>>> a = -1.5
>>> b = 3.25
>>>
>>> abs(a)
1.5
>>> max(a, b)
3.25
>>> min(a, b)
-1.5
>>>
>>> import math
>>> math.floor(a)
-2
>>> math.ceil(a)
-1
```

i += 1

运算符	描述	实例
=	简单的赋值运算符	c = a + b 将 a + b 的运算结果赋值为 c
+=	加法赋值运算符	c += a 等效于 c = c + a
-=	减法赋值运算符	c -= a 等效于 c = c - a
*=	乘法赋值运算符	c *= a 等效于 c = c * a
/=	除法赋值运算符	c /= a 等效于 c = c / a
%=	取模赋值运算符	c %= a 等效于 c = c % a
**=	幂赋值运算符	c **= a 等效于 c = c ** a
//=	取整除赋值运算符	c //= a 等效于 c = c // a

注意：Python中没有类似于“++”的运算符!

https://www.runoob.com/python3/python3-basic-operators.html

字符串

2-1.py

- 字符串就是一系列字符。
- 在Python中，用引号括起的都是字符串，其中的引号可以是单引号也可以是双引号;
  - 这种灵活性能让你在字符串中包含引号或撇号，而无需使用转义字符。

```
s1 = "I told my friend, \"Python is my favorite language!\""
s2 = 'I told my friend, "Python is my favorite language!"'
print(s1 == s2)
```

```
True
```

字符串：拼接

2-2.py

- 用加号（+）实现两个字符串的拼接

```
first_name = "Zhiqi"
last_name = "Lei"
name = first_name + " " + last_name
print(name)
print("Hello, " + name + "!")
```

```
Zhiqi Lei
Hello, Zhiqi Lei!
```

- 用乘号（*）实现重复自拼接

```
s = "haha"
print(s * 5)
```

```
hahahahahahahahahaha
```

字符串：方法

2-2.py

- 大小写

```
name = "zhiQi lei"
print(name.title()) # 每个单词的首字母转化为大写
print(name.lower()) # 所有字母转化为小写
print(name.upper()) # 所有字母转化为大写
```

```
Zhiqi Lei
zhiqi lei
ZHIQI LEI
```

- 删除空白（空格、换行、制表符）

```
name = " \tzhiQi lei\n"
print(name.strip()) # 删除字符串前后的空白字符
print(name.rstrip()) # 删除字符串后面的空白字符
print(name.lstrip()) # 删除字符串前面的空白字符
```

```
zhiQi lei
        zhiQi lei
zhiQi lei
```

字符串：方法

2-2.py

- 分割

```
sentence = "Life is short, you need Python."
print(sentence.split())
print(sentence.split(","))
```

```
['Life', 'is', 'short,', 'you', 'need', 'Python.']
['Life is short', ' you need Python.']
```

  - 以输入的符号为界，分割字符串，得到“列表”

- 替换

```
sentence = "Life is short, you need Python."
print(sentence.replace("h", "XD"))
print(sentence.replace("short", "long").replace("Python", "C++"))
```

```
Life is sXDort, you need PytXDon.
Life is long, you need C++.
```

类型转换

- 格式：datatype()
  - int()、float()、str()…
- 例：

```
print(5 // 3)  # 1
print(int(5 / 3))  # 1
```

```
import random

num = random.random()
message = "random number: " + str(num)
print(message)
```

▲通过import导入其它模块/库
▲如果直接将数值和字符串相加会导致出错！

简单数据类型：小结

- 数字
  - 包括：整数、浮点数
  - 运算：加（+）、减（-）、乘（*）、除（/）、整除（//）、幂（**）…
- 字符串
  - 拼接（+）
  - 方法：大小写、删除空白、分割、替换…
- 类型转换
- 模块的导入（import）
- 布尔值、空值

目录

控制结构

- 分支结构：if
- 循环结构：while
- 循环结构：for

控制结构：分支结构

```
if condition_A:
    do something
elif condition_B:
    do something
elif condition_C:
    do something
else:
    do something
```

注意：

- 是elif，而不是else if；
- if和elif后的条件不用括号包裹，if、elif和else最后加冒号；
- 每个分支内部的代码缩进，不加花括号包裹。

- if

```
age = 19
if age >= 18:
    print("You are old enough to vote!")
    print("Have you registered to vote yet?")
```

- if-else

```
age = 17
if age >= 18:
    print("You are old enough to vote!")
    print("Have you registered to vote yet?")
else:
    print("Sorry, you are too young to vote.")
    print("Please register to vote when 18!")
```

- if-elif-else

```
age = 12
if age < 4:
    price = 0
elif age < 18:
    price = 5
elif age < 65:
    price = 10
else:  # elif age >= 65:
    price = 5
print("Your admission cost is $" + str(price) + ".")
```

3-1.py

- 为了清晰和明确，最后的 else 也可改为 elif age >= 65
- 变量price虽然在缩进块内定义，但走出分支后依然可用

代码块与缩进

- 在C++中，代码块由花括号（{…}）包裹；
- 在Python中，代码块由缩进控制，Python根据缩进来判断代码行与前一个代码行的关系；

- 编写Python代码时要小心严谨地进行缩进，避免发生缩进错误：
  - 缩进量要统一（例如统一用4个空格）；
  - 分支/循环结束后的一行，记得删除一次缩进；
  - …

比较运算符

a=10, b=20

运算符	描述	实例
==	等于 - 比较对象是否相等	(a == b) 返回 False。
!=	不等于 - 比较两个对象是否不相等	(a != b) 返回 True。
>	大于 - 返回x是否大于y	(a > b) 返回 False。
<	小于 - 返回x是否小于y。	(a < b) 返回 True。
>=	大于等于 - 返回x是否大于等于y。	(a >= b) 返回 False。
<=	小于等于 - 返回x是否小于等于y。	(a <= b) 返回 True。

https://www.runoob.com/python3/python3-basic-operators.html

逻辑运算符

Python语言支持逻辑运算符，以下假设变量 a 为 10, b为 20:

运算符	逻辑表达式	描述	实例
and	x and y	布尔"与" - 如果 x 为 False，x and y 返回 x 的值，否则返回 y 的计算值。	(a and b) 返回 20。
or	x or y	布尔"或" - 如果 x 是 True，它返回 x 的值，否则它返回 y 的计算值。	(a or b) 返回 10。
not	not x	布尔"非" - 如果 x 为 True，返回 False 。如果 x 为 False，它返回 True。	not (a and b) 返回 False

https://www.runoob.com/python3/python3-basic-operators.html

while循环

- while循环不断地运行，直到指定的条件不满足为止。

```
while condition:
    do something
```

- while循环中的特殊语句：
  - break
  - continue

while循环： break和continue

```
while condition:
    do something
```

```
while condition:
    do something
    if condition:
        break
    do something
```

```
while condition:
    do something
    if condition:
        continue
    do something
```

while循环：例子

```
s = 0
i = 1
while i <= 100:
    s += i
    i += 1
print(s)
```

```
5050
```

```
s = 0
i = 1
while True:
    s += i
    i += 1
    if i > 100:
        break
print(s)
```

```
5050
```

```
i = 0
while i < 10:
    i += 1
    if i % 2 == 0:
        continue
    print(i)
```

```
1
3
5
7
9
```

```
s = 0
i = 1
flag = True
while flag:
    s += i
    i += 1
    flag = True if i <= 100 else False
print(s)
```

```
5050
```

3-2.py

for循环：初识

3-2.py

- range类型：可遍历的数字序列
  - range(stop)：从0到stop-1，步长为1
  - range(start, stop[, step])：从start到stop-step，步长为step，step默认值1

```
s = 0
for i in range(100):
    s += i + 1
print(s)
```

```
5050
```

```
s = 0
for i in range(1, 101):
    s += i
print(s)
```

```
5050
```

```
s = 0
for i in range(100, 0, -1):
    s += i
print(s)
```

```
5050
```

```
s = 0
for odd in range(1, 101, 2):
    s += odd
print(s)
```

```
2500
```

https://docs.python.org/3/library/stdtypes.html#ranges

控制结构：小结

- 分支
  - if、if-else、if-elif-else、if-elif
- 循环
  - while循环
  - 初识for循环与range
- 条件判断运算符与布尔表达式
  - 比较运算符
  - 逻辑运算符

目录

复杂数据类型（数据容器）

- 列表（list）
- 元组（tuple）
- 集合（set）
- 字典（dict）

列表list

- 列表由一系列按特定顺序排列的元素构成
- 回忆：字符串的split()方法

```
sentence = "Life is short, you need Python."
print(sentence.split())
```

```
['Life', 'is', 'short,', 'you', 'need', 'Python.']
```

- 在Python中，用方括号（[ ]）来表示列表，并用逗号来分隔其中的元素。

```
bicycles = ["trek", "cannondale", "redline", "specialized"]
```

- 注意：一个列表中的元素可以是不同类型的

```
l = ["abc", 123, 4.5, True, None]
```

列表：访问元素

4-1.py

```
bicycles = ["trek", "cannondale", "redline", "specialized"]

print(bicycles[0])
print(bicycles[0].title())
print(bicycles[1])
print(bicycles[3])
print(bicycles[-1])  # access the last element in the list
print(bicycles[-3])
message = "My first bicycle was a " + bicycles[0].title() + "."
print(message)
```

```
trek
Trek
cannondale
specialized
specialized
cannondale
My first bicycle was a Trek.
```

列表：修改、添加和删除元素

4-1.py

- 修改

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
motorcycles[0] = 'ducati'
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki']
['ducati', 'yamaha', 'suzuki']
```

- 列表创建后，元素可在程序运行过程中动态增删。

列表：修改、添加和删除元素

4-1.py

- 添加

```
motorcycles = []
motorcycles.append('honda')
motorcycles.append('yamaha')
motorcycles.append('suzuki')
print(motorcycles)

motorcycles.insert(0, 'ducati')
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki']
['ducati', 'honda', 'yamaha', 'suzuki']
```

- append()方法
  - 在列表末尾添加元素
- insert()方法
  - 在列表中插入元素
  - 第一个参数是插入位置
  - 第二个参数是插入的值

列表：修改、添加和删除元素

4-1.py

- 使用del语句删除元素

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
del motorcycles[1]
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki']
['honda', 'suzuki']
```

- 根据值删除元素（remove方法）

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
motorcycles.remove('yamaha')
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki']
['honda', 'suzuki']
```

列表：修改、添加和删除元素

- 使用pop()方法弹出（任何位置的）元素

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
last_owned = motorcycles.pop()
print("The last motorcycle I owned was a " + last_owned.title() + ".")
print(motorcycles)

motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
second_owned = motorcycles.pop(1)
print("The second motorcycle I owned was a " + second_owned.title() + ".")
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki']
The last motorcycle I owned was a Suzuki.
['honda', 'yamaha']
['honda', 'yamaha', 'suzuki']
The second motorcycle I owned was a Yamaha.
['honda', 'suzuki']
```

列表：长度

4-1.py

- 内置函数len()

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print(len(cars))
```

```
4
```

列表：翻转

4-1.py

- reverse()方法

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print(cars)
cars.reverse()
print(cars)
```

```
['bmw', 'audi', 'toyota', 'subaru']
['subaru', 'toyota', 'audi', 'bmw']
```

- 注意：reverse()方法做的是原地（in place）操作，即直接对cars永久地修改。
  - 可再次调用reverse()恢复原来的排列顺序。

列表：排序

4-1.py

- 使用方法sort()对列表进行永久性排序

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print(cars)
cars.sort()  # ascending
print(cars)
cars.sort(reverse=True)  # descending
print(cars)
```

```
['bmw', 'audi', 'toyota', 'subaru']
['audi', 'bmw', 'subaru', 'toyota']
['toyota', 'subaru', 'bmw', 'audi']
```

- sort()方法做的是原地（in place）操作，即直接对cars永久地修改。
  - 且无法恢复原来的排列顺序。

列表：排序

4-1.py

- 使用函数sorted()对列表进行临时排序

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print(cars)
cars_ascending = sorted(cars)
cars_descending = sorted(cars, reverse=True)
print(cars_ascending)
print(cars_descending)
print(cars)
```

```
['bmw', 'audi', 'toyota', 'subaru']
['audi', 'bmw', 'subaru', 'toyota']
['toyota', 'subaru', 'bmw', 'audi']
['bmw', 'audi', 'toyota', 'subaru']
```

- sorted()函数返回一个新的列表对象，且不会对输入的列表产生副作用（side effect），即不影响输入列表的原始排列顺序

列表：切片

4-1.py

- 切片：从列表中“切”出一段子列表。格式为：list_name[start: stop(: step)]

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print(players[0:3])
print(players[:3])
print(players[2:4])
print(players[2:])
print(players[-3:])
print(players[::2])
top_players = players[0:3]
```

```
['charles', 'martina', 'michael']
['charles', 'martina', 'michael']
['michael', 'florence']
['michael', 'florence', 'eli']
['michael', 'florence', 'eli']
['charles', 'michael', 'eli']
```

- 子列表中包含下标从start到stop-step，步长为step的所有元素
  - step默认为1可省略，start默认为0可留空，stop默认为列表长度可留空

列表：赋值与复制

- 赋值：这是你预期的结果吗？

4-1.py

```
my_foods = ['pizza', 'falafel', 'carrot cake']
print('my_foods:', my_foods)
your_foods = my_foods
your_foods[-1] = 'apple'
print('yr_foods:', your_foods)
print('my_foods:', my_foods)
```

python中的赋值操作
https://www.cnblogs.com/zf-blog/p/10613981.html

```
my_foods: ['pizza', 'falafel', 'carrot cake']
yr_foods: ['pizza', 'falafel', 'apple']
my_foods: ['pizza', 'falafel', 'apple']
```

- 分析
  - ▲ id()函数用于获取对象内存地址。
  - ▲ 身份运算符 is：**x is y**, 类似 **id(x) == id(y)**

```
print(id(your_foods))
print(id(my_foods))
print(id(your_foods) == id(my_foods))
print(your_foods is my_foods)
```

```
1986166284936
1986166284936
True
True
```

列表：赋值与复制

- 利用切片复制

```
my_foods = ['pizza', 'falafel', 'carrot cake']
print('my_foods:', my_foods)
your_foods = my_foods[:]
your_foods[-1] = 'apple'
print('yr_foods:', your_foods)
print('my_foods:', my_foods)
```

切片会创建一个新的对象，
分配新的内存空间

4-1.py

```
my_foods: ['pizza', 'falafel', 'carrot cake']
yr_foods: ['pizza', 'falafel', 'apple']
my_foods: ['pizza', 'falafel', 'carrot cake']
```

- 分析
  - ▲ id()函数用于获取对象内存地址。
  - ▲ 身份运算符：**x is y**, 类似 **id(x) == id(y)**

```
print(id(your_foods))
print(id(my_foods))
print(id(your_foods) == id(my_foods))
print(your_foods is my_foods)
```

```
1986166285064
1986166284744
False
False
```

浅复制与深复制

4-1.py

```
import copy
a = [1, 2, 3, 4, ['a', 'b']]

b = a  # assign
c = a[:]  # slice (shallow copy)
d = copy.copy(a)  # shallow copy
e = copy.deepcopy(a)  # deep copy

a.append(5)
a[4].append('c')
```

```
print( 'a = ', a )
print( 'b = ', b )
print( 'c = ', c )
print( 'd = ', d )
print( ‘e = ', e )
```

- 利用Python标准库的copy库

- 结果

```
a =  [1, 2, 3, 4, ['a', 'b', 'c'], 5]
b =  [1, 2, 3, 4, ['a', 'b', 'c'], 5]
c =  [1, 2, 3, 4, ['a', 'b', 'c']]
d =  [1, 2, 3, 4, ['a', 'b', 'c']]
e =  [1, 2, 3, 4, ['a', 'b']]
```

https://www.runoob.com/w3cnote/python-understanding-dict-copy-shallow-or-deep.html

身份运算符

身份运算符用于比较两个对象的存储单元

运算符	描述	实例
is	is 是判断两个标识符是不是引用自一个对象	**x is y**, 类似 **id(x) == id(y)** , 如果引用的是同一个对象则返回 True，否则返回 False。
is not	is not 是判断两个标识符是不是引用自不同对象	**x is not y**，类似 **id(a) != id(b)**。如果引用的不是同一个对象则返回结果 True，否则返回 False。

▲ id()函数用于获取对象内存地址

▲ is 与 == 区别:
is 用于判断两个变量引用对象是否为同一个(同一块内存空间),
== 用于判断引用变量的值是否相等。

```
>>> a = 2
>>> b = 2
>>> a == b
True
>>> a is b
True
```

```
>>> a = 300
>>> b = 300
>>> a == b
True
>>> a is b
False
```

https://www.cnblogs.com/Victor-ZH/p/13044135.html
python的大整数对象和小整数对象

https://www.runoob.com/python3/python3-basic-operators.html

列表：for循环遍历

4-2.py

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(magician.title() + ", that was a great trick!")
    print("I can't wait to see your next trick, " + magician.title() + ".\n")
print("Thank you, everyone. That was a great magic show!")
```

```
Alice, that was a great trick!
I can't wait to see your next trick, Alice.

David, that was a great trick!
I can't wait to see your next trick, David.

Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.

Thank you, everyone. That was a great magic show!
```

列表：数值列表

4-2.py

- 使用range的for循环

```
squares = []
for value in range(1, 11):
    squares.append(value**2)
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

- 复习：range类型
  - range(stop)：
    - 从0到stop-1，步长为1
  - range(start, stop[, step])：
    - 从0到stop-step，步长为step，step默认值1

https://docs.python.org/3/library/stdtypes.html#ranges

列表：列表解析

4-2.py

- 使用range的for循环

```
squares = []
for value in range(1, 11):
    squares.append(value**2)
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

- 列表解析

```
squares = [value**2 for value in range(1, 11)]


print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

- 例子：4*3矩阵初始化

```
>>> [[0 for j in range(3)] for i in range(4)]
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

列表：使用if语句处理列表

4-2.py

- 确定列表不是空的

```
list = []
if list:
    print("Not empty")
else:
    print("Empty")
```

```
Empty
```

- 判断元素在列表中

```
magicians = ['alice', 'david', 'carolina']
if 'alice' in magicians:
    print("Hi, Alice!")
print('zachary' in magicians)
```

```
Hi, Alice!
False
```

- 类似的有：0、空列表、空字符串、None等
- 成员运算符：in

元组tuple

- 不可变的列表
- 圆括号标识

4-2.py

```
dimensions = (200, 50)  #原始元组
for dimension in dimensions:
    print(dimension)
dimensions = (150, 50)  #整体修改
for dimension in dimensions:
    print(dimension)
dimensions = list(dimensions)
dimensions[0] = 100  #转为列表修改
dimensions = tuple(dimensions)
for dimension in dimensions:
    print(dimension)
dimensions[0] = 200  #试图直接修改
```

```
200
50
150
50
100
50
Traceback (most recent call last):
  File "4-2.py", line 29, in <module>
    dimensions[0] = 200
TypeError: 'tuple' object does not support item assignment
```

集合set

- 集合（set）是一个无序的不重复元素序列。
- 其中一种常用场景：元素去重

```
>>> a = [1, 4, 2, 1, 2]
>>> list(set(a))
[1, 2, 4]
```

- 扩展阅读：https://www.runoob.com/python3/python3-set.html

字典dict

4-3.py

- 字典是一系列键-值对（key-value pair），每个键都与一个值相关联。

```
alien_0 = {'color': 'green', 'points': 5}
```

- 访问字典中的值

```
print(alien_0['color'])
print(alien_0['points'])
```

```
green
5
```

- 修改字典中的值

```
alien_0['color'] = 'yellow'
print(alien_0['color'])
```

```
yellow
```

字典

4-3.py

- 添加键-值对

```
alien_0 = {}
alien_0['color'] = 'green'
alien_0['points'] = 5
print(alien_0)
```

```
{'color': 'green', 'points': 5}
```

- 删除键-值对

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)
del alien_0['points']
print(alien_0)
```

```
{'color': 'green', 'points': 5}
{'color': 'green'}
```

字典：遍历

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'ruby',
    'phil': 'python',
    }
```

- 遍历所有的键-值对

```
for name, language in favorite_languages.items():
    print(name.title() + "'s favorite language is " + language.title() + ".")
```

```
Jen's favorite language is Python.
Sarah's favorite language is C.
Edward's favorite language is Ruby.
Phil's favorite language is Python.
```

- 遍历字典中的所有键

```
for name in favorite_languages.keys():
    print(name.title())
```

```
Jen
Sarah
Edward
Phil
```

- 遍历字典中的所有值

```
for language in favorite_languages.values():
    print(language.title())
```

```
Python
C
Ruby
Python
```

字典：遍历

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'ruby',
    'phil': 'python',
    }
```

- 按顺序遍历字典中的所有键

```
for name in sorted(favorite_languages.keys()):
    language = favorite_languages[name]
    print(name.title() + "'s favorite language is " + language.title() + ".")
```

```
Edward's favorite language is Ruby.
Jen's favorite language is Python.
Phil's favorite language is Python.
Sarah's favorite language is C.
```

  - 虽然在一些py3版本中遍历顺序与存储/添加顺序相同，但这不被保证。[1]
  - 因此，我们可以利用sorted规定遍历顺序

- 遍历字典中的所有不重复值

```
for language in set(favorite_languages.values()):
    print(language.title())
```

```
Ruby
C
Python
```

[1] https://exp.newsmth.net/topic/article/e8a1d1ca4cffec5c61b42f4debb665e7

复杂数据结构与操作：小结

- 列表（list）
  - 元素的访问、修改、添加和删除
  - 列表的操作：长度、翻转、排序、切片与复制
  - 列表的遍历（for），数字列表，列表解析
  - 使用if语句处理列表
- 元组（tuple）、集合（set）
- 字典（dict）
  - 访问、修改、添加和删除
  - 遍历

目录

函数

- “带名字的代码块”
- 要执行函数定义的特定任务，可调用该函数。
- 需要在程序中多次执行同一项任务时，你无需反复编写完成该任务的代码，而只需调用执行该任务的函数。
- 通过使用函数，程序的编写、阅读、测试和修复都将更容易。

定义函数

5-1.py

- 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**

返回值

5-1.py

- 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
```

返回值：返回多个值

5-1.py

- 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')
```

返回值：返回字典

5-1.py

- 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}
```

传递实参

5-1.py

```
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。

传递列表

5-1.py

- 将列表传递给函数

```
def greet_users(names):
    for name in names:
        msg = "Hello, " + name.title() + "!"
        print(msg)

usernames = ['hannah', 'ty', 'margot']
greet_users(usernames)
```

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

传递列表：在函数中修改列表

5-1.py

- 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

传递任意数量的实参

5-1.py

- 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）

定义类

5-2.py

- 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

5-2.py

- 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.
```

5-2.py

- 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

5-2.py

- **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

读取文件

pi_digits.txt

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

6-1.py

- 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，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。

6-1.py

- 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。

使用try-except代码块

6-2.py

- 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

6-2.py

- 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

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)
```

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

name和main的前与后，均有两个下划线。

- 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__':

if __name__ == '__main__' 如何正确理解？ https://www.zhihu.com/question/49136398

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

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