

Artificial Intelligence

人工智能实验

中山大学计算机学院
2024年春季

目录

1. 实验课课程要求
2. 实验课课程安排
3. Python程序设计基础
4. 作业
5. 作业提交说明

目录

1. 实验课课程要求
2. 实验课课程安排
3. Python程序设计基础
4. 作业
5. 作业提交说明

1. 实验课课程要求

- **实验课课程内容**

- 由助教讲解实验内容;
- 会进行**考勤**;
- PPT课件和作业放在Github, 自行下载:
<https://github.com/ZYC9894/2024AI>

- **实验课代码要求**

- 编程语言建议使用**Python**;
- 不能使用现有算法高级库 (除非作业中明确说明可以使用), 若有疑问可以询问助教;
- 禁止抄袭 (代码和实验报告都禁止抄袭, 包括修改变量名、代码结构等, 若被发现后果严重);

1. 实验课课程要求

• 实验报告内容要求

- 算法原理：用**自己的语言**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）。要求尽可能简洁、扼要，减少无关紧要的内容；
- 关键代码展示：可截图或复制代码并对每个模块进行解释，包括注释；
- 创新点&优化：如果有的话，分点列出自己的**创新点**（加分项）；
- 实验结果展示：基础算法的结果以及优化的算法结果+分析；
- 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。

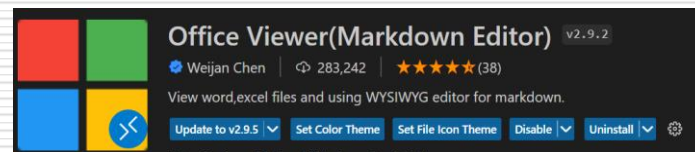
1. 实验课课程要求

• 作业提交要求

- 提交源代码文件(py格式);
- 源代码会执行检查, 请确保代码可以跑通并得出正确结果。若代码无法运行或者结果错误, 将会酌情扣分。
- 源代码在必要的地方给注释 (说明每部分代码的实现功能, 变量的含义等) 。
- 实验报告可使用Word/Markdown/Latex撰写, 务必以pdf格式提交。
- 若报告中含有公式推导, 可以手写拍照并复制到文件中, 或用公式编辑器直接编辑, 但请确保最后导出的pdf文件中公式不会乱码。
- 若报告中有代码展示, 可以直接复制代码也可以截图展示。若复制代码务必保留正确的缩进。

1. 实验课课程要求

• 推荐的免费Markdown编辑器

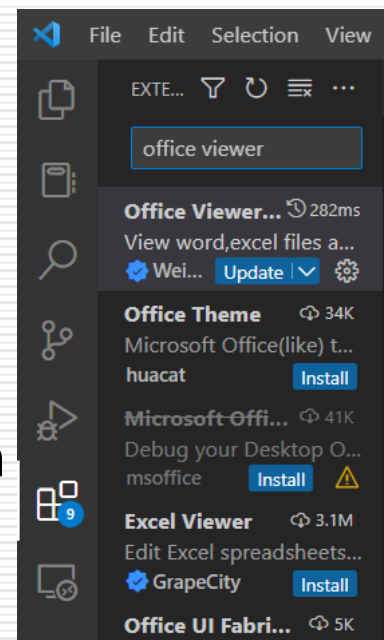


安装步骤:

- 浏览器进入网址[Download Visual Studio Code - Mac, Linux, Windows](#), 下载vscode
- 打开vscode, 左侧工具栏选择Extensions, 搜索插件" Office Viewer(Markdown Editor)"
- 安装插件后即可使用

使用方式:

- 打开vscode, 左侧工具栏选择Explorer, 打开文件夹Open Folder
- 在文件夹中新建.md后缀文件, 例如" homework.md"
- 双击该文件即可编辑
- 编辑完成后, 点击上方Export to pdf 导出pdf文件



目录

1. 实验课课程要求
2. 实验课课程安排
3. Python程序设计基础
4. 作业
5. 作业提交说明

2. 实验课课程安排

• 课程安排（暂定）

周次	课上	课下	重要时间节点
1	Python程序设计基础 I	实验1-1	
2	Python程序设计基础 II	实验1-2	
3	归结推理	实验2-1	实验1提交
4	知识图谱	实验2-2	
5	盲目搜索	实验3-1	实验2提交
6	启发式搜索	实验3-2	
7	博弈树搜索	实验4-1	实验3提交
8	高级搜索算法	实验4-2	

2. 实验课课程安排

• 课程安排（暂定）

周次	课上	课下	重要时间节点
9	贝叶斯网络	实验5-1	实验4提交
10	机器学习 I	实验5-2	
11	机器学习 II	实验5-3	
12	机器学习 III	实验6-1	实验5提交
13	机器学习 IV	实验6-2	
14	Pytorch	实验7-1	实验6提交
15	强化学习I	实验7-2	
16	强化学习II	实验7-3	
17	强化学习III	实验7-4	
18	答疑		

目录

1. 实验课课程要求
2. 实验课课程安排
3. Python程序设计基础
4. 作业
5. 作业提交说明

3. Python程序设计基础

- 参考资料与建议阅读

- 《Python编程：从入门到实践》
- 《Python中文指南》 (<https://python.iswbm.com/>)
- 《人工智能（第3版）》附录A
- <https://www.python.org>
- <https://www.runoob.com/python3/python3-tutorial.html>

3. Python程序设计基础

- 目录

- 0 Python环境配置
- 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

3. Python程序设计基础

- 目录

- 0 Python环境配置

- 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

3. Python程序设计基础

• 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](https://docs.conda.io/en/latest/miniconda.html)

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

Anaconda — <https://zhuanlan.zhihu.com/p/75717350>

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。

```
C:\Users\Administrator>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.
  compare              Compare packages between conda environments.
  config               Modify configuration values in .condarc. This is modeled
                        after the git config command. Writes to the user .condarc
                        file (C:\Users\Administrator\condarc) by default.
  create               Create a new conda environment from a list of specified
                        packages
```

3. Python程序设计基础

- 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

3. Python程序设计基础

- 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

3. Python程序设计基础

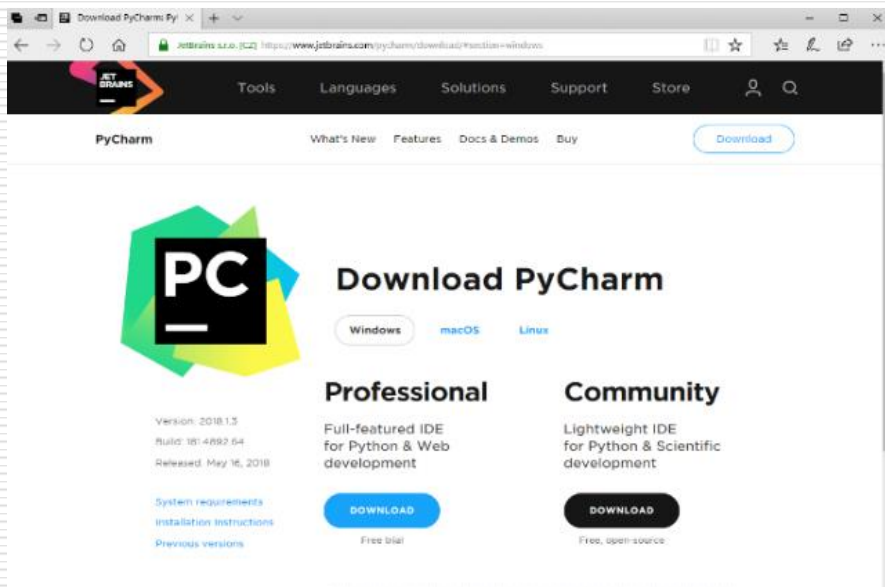
- Pycharm

- PyCharm有两个重要版本：社区版和专业版
- 官方下载地址：

<http://www.jetbrains.com/pycharm/download/>

- 安装Pycharm并配置Python 环境

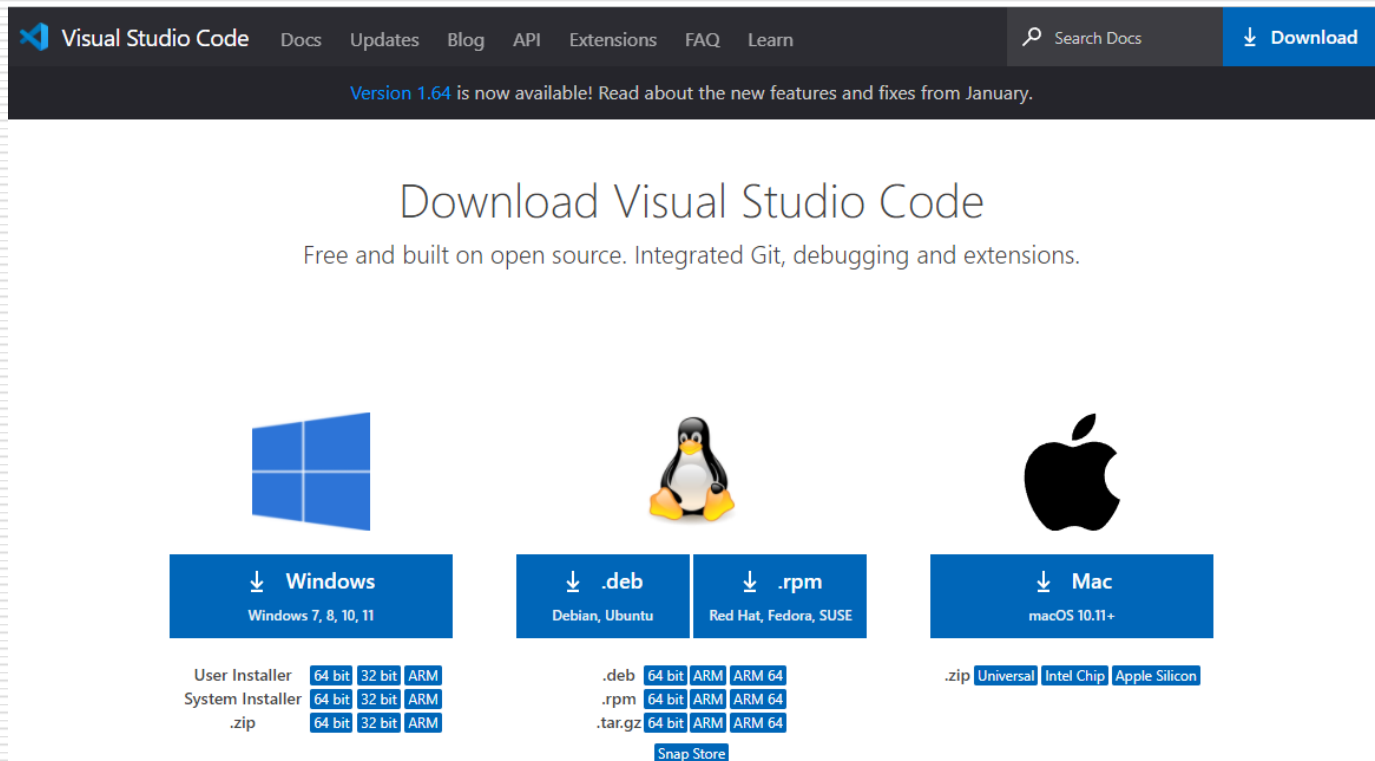
<https://zhuanlan.zhihu.com/p/140485845>



下载页面提供了适用于Windows、macOS和Linux等操作系统的各版本PyCharm下载，其中Professional（专业版）可供试用，Community（社区版）是轻量级（Lightweight）的免费版。

3. Python程序设计基础

- Vscode
 - VS Code 安装:
<https://zhuanlan.zhihu.com/p/264785441>
 - VS Code python配置:
<https://zhuanlan.zhihu.com/p/31417084>



3. Python程序设计基础

- 目录

0 Python环境配置

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

3. Python程序设计基础

• 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>

3. Python程序设计基础

• 什么是Python

□ 应用场景



3. Python程序设计基础

- Hello World程序与运行

- Hello World程序

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

Hello World!

- 注意：一行表示一条语句，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

3. Python程序设计基础

- **Hello World程序：注释**

- ❑ 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!")
```


3. Python程序设计基础

- **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。

3. Python程序设计基础

• Hello World程序：输出

□ 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 = 2024  
print(message_1, message_2)  
print(message_1, message_2, sep="AI", end="SYSU")
```

```
Hello World! 2024  
Hello World!AI2024SYSU
```

3. Python程序设计基础

- **Hello World程序：用户输入**

- 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

3. Python程序设计基础

• 目录

- 0 Python环境配置
- 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

3. Python程序设计基础

- 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`

3. Python程序设计基础

• 数字

- 整数：加 (+)、减 (-)、乘 (*)、除 (/)、整除 (//)、幂 (**)、模 (%)
- 浮点数：加 (+)、减 (-)、乘 (*)、除 (/)、整除 (//)、幂 (**)、模 (%)

```
>>> 2 + 3 * 4
14
>>> 3 / 2
1.5
>>> 3 // 2
1
>>> 4 / 2
2.0
```

```
>>> 4 // 2
2
>>> 3 ** 2
9
>>> 10 ** 6
1000000
>>> 5 % 3
2
```

```
>>> 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
```

3. Python程序设计基础

• 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
```

3. Python程序设计基础

• 运算符

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

注意: Python中没有类似于“++”的运算符 !

3. Python程序设计基础

• 字符串

- 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

3. Python程序设计基础

• 字符串

- 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)
```

hahahahahahahahaha

- 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

3. Python程序设计基础

• 字符串

□ 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
```

□ 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++.
```

3. Python程序设计基础

- **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!

3. Python程序设计基础

• 目录

- 0 Python环境配置
- 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

3. Python程序设计基础

• 控制结构

□ 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

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

3. Python程序设计基础

- 控制结构

- 逻辑运算符

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

3. Python程序设计基础

• 控制结构

□ 分支结构：if

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

```
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) + ".")
```

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。

3. Python程序设计基础

• 控制结构

□ 循环结构: while

- 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 condition:  
    do something  
    if condition:  
        break  
    do something
```

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

```
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

41

3. Python程序设计基础

• 控制结构

□ 循环结构: for

■ 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)
```

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

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

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

5050

5050

5050

2500

3. Python程序设计基础

• 目录

- 0 Python环境配置
- 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

3. Python程序设计基础

• 数据容器

- 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]
```

3. Python程序设计基础

- 数据容器

- 列表 (list) : 访问元素

```
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.
```

3. Python程序设计基础

• 数据容器

□ 列表 (list) : 修改、添加、删除

□ 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。

□ 添加

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

□ 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

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

3. Python程序设计基础

• 数据容器

□ 列表 (list) : 修改、添加、删除

□ 删除

□ 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']
```

□ 使用pop()方法弹出 (任何位置的) 元素

```
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 second motorcycle I owned was a Yamaha.  
['honda', 'suzuki']
```

3. Python程序设计基础

• 数据容器

□ 列表 (list) : 长度、翻转、排序、切片

□ 长度

□ 内置函数len()

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

4

□ 翻转

□ reverse()方法

□ 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。

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

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


3. Python程序设计基础

• 数据容器

□ 列表 (list) : 长度、翻转、排序、切片

□ 排序

- 使用方法sort()对列表进行永久性排序、使用函数sorted()对列表进行临时排序
- 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。sorted()函数返回一个新的列表对象, 且不影响输入列表的原始排列顺序

```
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']
```

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

3. Python程序设计基础

- 数据容器

- 列表 (list) : 长度、翻转、排序、切片

- 切片

- 切片: 从列表中“切”出一段子列表

- 格式为: `list_name[start: stop(: step)]`

- 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

```
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']
```

3. Python程序设计基础

- 数据容器

- 列表 (list) : 赋值、复制

- 赋值

```
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)
```

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

- ▲ **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
```

3. Python程序设计基础

• 数据容器

□ 列表 (list) : 赋值、复制

□ 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)
```

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

▲ **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)
```

```
1986166285064  
1986166284744  
False  
False
```

3. Python程序设计基础

- 数据容器

- 列表 (list) : 赋值、复制

- 浅复制和深复制

```
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 )
```

```
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']]
```

3. Python程序设计基础

• 数据容器

□ 列表 (list) : 赋值、复制

□ 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	<code>x is y</code> , 类似 <code>id(x) == id(y)</code> , 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	<code>x is not y</code> , 类似 <code>id(a) != id(b)</code> 。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
```

python的大整数对象和小整数对象

<https://www.cnblogs.com/Victor-ZH/p/13044135.html>

3. Python程序设计基础

- 数据容器

- 列表 (list) : for循环遍历

```
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!
```

3. Python程序设计基础

- 数据容器

- 列表 (list) : 数值列表

- 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

3. Python程序设计基础

• 数据容器

□ 列表 (list) : 使用if处理列表

□ 确定列表不是空的

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

□ 类似的有：0、空列表、空字符串、None等

Empty

□ 判断元素在列表中，成员运算符：in

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

Hi, Alice!
False

3. Python程序设计基础

- 数据容器

- 元组 (tuple) :

- 不可变的列表

- 圆括号标识

```
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
```

3. Python程序设计基础

- 数据容器

- 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>

3. Python程序设计基础

• 数据容器

□ 字典 (dict) :

- 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
```

- 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'}
```

3. Python程序设计基础

- 数据容器

- 字典 (dict) :

- 遍历所有的键-值对

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

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

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

3. Python程序设计基础

• 目录

- 0 Python环境配置
- 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

3. Python程序设计基础

• 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。

❑ 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**

3. Python程序设计基础

• 函数

- ❑ 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
        return full_name.title()
    else:
        full_name = first_name + ' ' + last_name
        return first_name.title(), last_name.title()
```

- ❑ 用一个或多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')
Jimi Hendrix Hooker

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

Jimi Hendrix

3. Python程序设计基础

• 函数

- 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}
```

3. Python程序设计基础

• 函数

□ 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() + ".")
```

□ 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')
```

3. Python程序设计基础

• 函数

- 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。
- 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']
```

3. Python程序设计基础

• 函数

- 有时候，需要防止函数修改列表，则需向函数传递列表副本，可保留函数外原始列表的内容：
 - 利用切片创建副本： `function_name(list_name[:])`

- 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')
```

toppings是一个元组

```
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):
```

3. Python程序设计基础

- 函数

- 形参前加**号，可传递任意数量的关键字实参

```
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_info是一个字典

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

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

3. Python程序设计基础

• 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）

```
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

3. Python程序设计基础

• 类

□ 创建对象

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

□ 访问属性

Python默认是公有属性，在变量名或函数名前加上两个下划线（“__”），这个函数或变量就变为私有了

```
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!
```

3. Python程序设计基础

• 类

□ 属性修改

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

```
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
```

□ 直接修改属性的值

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

破坏了封装

□ 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.

3. Python程序设计基础

• 类

□ 继承

Python允许父类调用子类方法!!!

□ **子类是父类的特殊版本**

□ 子类继承父类的所有属性和方法

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

```
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())
```

class 子类名(父类名) 2016 Tesla Model S

Electric 2016 Tesla Model S

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

□ **实例可作为属性（类的成员）**

3. Python程序设计基础

• 目录

- 0 Python环境配置
- 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

3. Python程序设计基础

• 文件与异常

□ 读取文件

□ 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()

pi_digits.txt

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

□ 逐行读取：readlines()

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

□ 得到一个列表

```
3. 1415926535  
  
8979323846  
  
2643383279
```

□ 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。

3. Python程序设计基础

• 文件与异常

- `open(“文件路径”, “模式”)`, 对于模式参数:
 - `'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**。

3. Python程序设计基础

• 文件与异常

□ 写入文件

□ 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
```

3. Python程序设计基础

• 文件与异常

- 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。

3. Python程序设计基础

• 文件与异常

□ 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。

3. Python程序设计基础

• 文件与异常

□ 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。

3. Python程序设计基础

• 目录

- 0 Python环境配置
- 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

3. Python程序设计基础

• 模块

- 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
 - `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

3. Python程序设计基础

• 模块

□ main函数

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__':

if __name__ == '__main__' 如何正确理解？

<https://www.zhihu.com/question/49136398>

3. Python程序设计基础

• 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：<https://pymotw.com/>

3. Python程序设计基础

• Python标准库

□ Python包安装：

- 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`，如 `pip install numpy`
- 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`

□ 常用包

- 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：gensim

目录

1. 实验课课程要求
2. 实验课课程安排
3. Python程序设计基础
4. 作业
5. 作业提交说明

4. 作业

• 第一周作业（第1页/共2页）

1.二分查找

给定一个 n 个元素有序的（升序）整型数组 `nums` 和一个目标值 `target`，写一个函数 `BinarySearch` 搜索 `nums` 中的 `target`，如果目标值存在返回下标，否则返回 `-1`。

2.矩阵加法,乘法

给定两个 $n \times n$ 的整型矩阵 `A` 和 `B`，写两个函数 `MatrixAdd` 和 `MatrixMul`，分别得出这两个矩阵加法和乘法的结果。

两个矩阵的数据类型为嵌套列表，即 `list[list]`，且满足 `len(list)==n`，`len(list[0])==n`。

注意不要打乱原矩阵 `A` 和 `B` 中的数据。

4. 作业

• 第一周作业 (第2页/共2页)

3.字典遍历

给定非空字典 `dict1`, 其键为姓名, 值是学号. 写一个函数 `ReverseKeyValue` 返回另一个字典, 其键是学号, 值是姓名.


例如, `dict1={'Alice':'001', 'Bob':'002'}`, 则 `ReverseKeyValue(dict1)` 返回的结果是 `{'001':'Alice', '002':'Bob'}`.

4. 作业

• 第二周作业 (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:

```
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
```

 student_data.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

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]]
```

4. 作业

• 第二周作业 (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]]
```

4. 作业

• 第二周作业 (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.

4. 作业

• 第二周作业 (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.

目录

1. 实验课课程要求
2. 实验课课程安排
3. Python程序设计基础
4. 作业
5. 作业提交说明

5 作业提交说明

• 作业提交说明

- 提交一个压缩包。压缩包命名为：“学号_姓名_作业编号”，例如：20240227_张三_实验1。
- 压缩包包含两部分：Code文件夹和实验报告PDF文件
 - Code文件夹：存放实验代码
 - PDF文件格式参考发的模板
- 如果需要提交新版本，则在压缩包后面加_v1等。如“学号_姓名_作业编号_v1.zip”，以此类推。
- **截止日期：**
 - **DDL：2024年3月12日24点**
- **提交邮箱：**zhangyc8@mail2.sysu.edu.cn