#### 1.模块和包

目前为止，我们可以编写大量的代码，但是代码可维护性是我们应该考虑的一个点，我们不可能将所有的代码写道一个文件里面，为了提高代码的维护性，方便代码管理，就引入了模块和包的概念。

**重点**：掌握模块和包的编写，熟悉一些python中的内置模块

#### 2.模块

模块是一个包含 Python 定义和语句的文件。它可以包含函数、类、变量和其他可执行代码。模块提供了一种将相关代码组织起来的方式，使得代码可以被其他程序引用和复用。在Python中，每个py文件都可以作为一个模块。

# 创建一个名为"my\_module.py"的模块  
# my\_module.py  
def greeting(name):  
 print("Hello, " + name)  
  
def add(a, b):  
 return a + b

在另一个Python文件中使用模块：

# main.py  
import myModule  
  
myModule.greeting("Alice") # 输出：Hello, Alice  
  
result = myModule.add(2, 3)  
print(result) # 输出：5

在上面的示例中，我们创建了一个名为 "myModule.py" 的模块，其中包含了 greeting() 和 add() 两个函数。在另一个Python文件 "main.py" 中，我们通过 import 语句引入了 "myModule" 模块，并使用其中的函数。

或者使用下面的代码导入模块。

from myModule import add,greeting  
  
greeting("Alice") # 输出：Hello, Alice  
  
result = add(2, 3)  
print(result) # 输出：5

#### 3.包

包是一种用于组织模块的层次结构。它是一个包含了多个模块的目录，并且包含一个特殊的 \_\_init\_\_.py 文件，用于标识该目录为一个包。包提供了更高级别的模块组织方式，可以将相关的模块组织在一起形成一个功能集合。

使用包的好处包括：

* 更好的模块组织和管理。
* 更清晰的命名空间，避免模块命名冲突。

my\_package/  
|-- \_\_init\_\_.py  
|-- module1.py  
|-- module2.py  
`-- sub\_package/  
 |-- \_\_init\_\_.py  
 |-- module3.py

在包中，可以有多个模块文件，它们可以相互引用。在 \_\_init\_\_.py 文件中，可以提供一些初始化代码或者导入其他模块。\_\_init\_\_代码示例：

# \_\_init\_\_.py  
from .module1 import function1  
from .module2 import function2

在其他Python文件中使用包：

# main.py  
import my\_package.module1  
import my\_package.module2  
  
my\_package.module1.function1()  
my\_package.module2.function2()

在上面的示例中，我们创建了一个名为 "my\_package" 的包，其中包含了两个模块文件 "module1.py" 和 "module2.py"。在另一个Python文件 "main.py" 中，我们通过 import 语句引入了包中的模块，并使用其中的函数。

#### 4.python中常见的内置模块

python中有很多常见的内置模块，提供各种功能，常见的有math，random，datatime，json，os，sys，re，gzip，csv，urllib等

##### 4.1math模块

math 是 Python 的一个内置数学包，提供了许多数学运算相关的函数和常量。可以使用 import math 来导入该包。下面是一些 math 包中常见的用法和示例：

1. 数学常量：
   * math.pi：表示圆周率 π 的近似值。
   * math.e：表示自然对数的底数 e 的近似值。

* 示例：
* import math  
  print(math.pi) # 输出圆周率的近似值 3.141592653589793  
  print(math.e) # 输出自然对数的底数 e 的近似值 2.718281828459045

1. 数学函数：
   * math.sqrt(x)：返回 x 的平方根。
   * math.pow(x, y)：返回 x 的 y 次幂。
   * math.sin(x)、math.cos(x)、math.tan(x)：返回 x 的正弦、余弦和正切值。
   * math.log(x)：返回 x 的自然对数。
   * math.floor(x)：返回不大于 x 的最大整数。
   * math.ceil(x)：返回不小于 x 的最小整数。

* import math  
    
  print(math.sqrt(16)) # 输出 4.0  
  print(math.pow(2, 3)) # 输出 8.0  
  print(math.sin(math.pi / 2)) # 输出 1.0  
  print(math.log(10)) # 输出 2.302585092994046  
  print(math.floor(3.7)) # 输出 3  
  print(math.ceil(3.2)) # 输出 4

1. 角度与弧度转换：
   * math.radians(x)：将角度 x 转换为弧度。
   * math.degrees(x)：将弧度 x 转换为角度。

* import math  
    
  print(math.radians(180)) # 输出 3.141592653589793  
  print(math.degrees(math.pi)) # 输出 180.0

1. 其他函数：
   * math.abs(x)：返回 x 的绝对值。
   * math.factorial(x)：返回 x 的阶乘。
   * math.gcd(x, y)：返回 x 和 y 的最大公约数。

* 示例：
* import math  
    
  print(math.fabs(-5)) # 输出 5.0  
  print(math.factorial(5)) # 输出 120  
  print(math.gcd(24, 36)) # 输出 12

##### 4.2random模块

random 是 Python 的一个内置模块，提供了生成伪随机数的功能。可以使用 import random 来导入该模块。下面是一些 random 模块的常见用法和示例：

1. 生成随机数：
   * random.random()：返回一个0到1之间的随机浮点数。
   * random.randint(a, b)：返回一个在范围 [a, b] 内的随机整数。
   * random.uniform(a, b)：返回一个在范围 [a, b] 内的随机浮点数。

* import random  
    
  print(random.random()) # 输出一个0到1之间的随机浮点数  
  print(random.randint(1, 10)) # 输出一个1到10之间的随机整数  
  print(random.uniform(2.5, 5.5)) # 输出一个2.5到5.5之间的随机浮点数

1. 从序列中随机选择元素：
   * random.choice(sequence)：从一个非空序列中随机选择一个元素。
   * random.sample(sequence, k)：从一个非空序列中随机选择 k 个不重复的元素。

* import random  
    
  fruits = ['apple', 'banana', 'orange', 'grape']  
  print(random.choice(fruits)) # 从 fruits 中随机选择一个水果  
    
  cards = ['A', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K']  
  print(random.sample(cards, 5)) # 从 cards 中随机选择5张不重复的牌

1. 打乱序列的顺序：
   * random.shuffle(sequence)：随机打乱一个序列的顺序。

* import random  
    
  numbers = [1, 2, 3, 4, 5]  
  random.shuffle(numbers)  
  print(numbers) # 输出打乱顺序后的 numbers

1. 设置随机数种子：
   * random.seed(seed)：设置随机数的种子，使得随机数生成具有可预测性。

* import random  
    
  random.seed(42)  
  print(random.random()) # 输出固定的随机浮点数

##### 4.3datatime模块

datetime 模块是 Python 的一个内置模块，提供了处理日期和时间的功能。它包含了多个类和函数，用于创建、操作和格式化日期时间对象。

###### 4.3.1创建日期时间对象

datetime.datetime(year, month, day, hour, minute, second)：创建一个表示特定日期和时间的 datetime 对象。

datetime.date(year, month, day)：创建一个表示特定日期的 date 对象。

datetime.time(hour, minute, second)：创建一个表示特定时间的 time 对象。

示例：

import datetime  
  
current\_datetime = datetime.datetime.now() # 获取当前日期和时间  
print(current\_datetime)  
  
some\_date = datetime.date(2022, 5, 13) # 创建一个特定的日期  
print(some\_date)  
  
some\_time = datetime.time(15, 30, 45) # 创建一个特定的时间  
print(some\_time)

###### 4.3.2格式化日期时间对象为字符串

datetime\_obj.strftime(format)：将 datetime 对象按照指定的格式转换为字符串。

import datetime  
  
current\_datetime = datetime.datetime.now()  
formatted\_datetime = current\_datetime.strftime("%Y-%m-%d %H:%M:%S")  
print(formatted\_datetime)

###### 4.3.3解析字符串为日期时间对象

* datetime.datetime.strptime(date\_string, format)：将字符串按照指定的格式解析为 datetime 对象。

示例：

import datetime  
  
date\_string = "2022-05-13 15:30:45"  
parsed\_datetime = datetime.datetime.strptime(date\_string, "%Y-%m-%d %H:%M:%S") #表示合理的日期格式  
print(parsed\_datetime)

###### 4.3.4进行日期时间计算和操作：

* datetime.timedelta(days, seconds, microseconds, milliseconds, minutes, hours, weeks)：表示一个时间间隔，可以进行日期时间的加减操作。

示例：

import datetime  
  
current\_datetime = datetime.datetime.now()  
one\_day = datetime.timedelta(days=1)  
next\_day = current\_datetime + one\_day # 当前日期加上一天  
print(next\_day)

##### 4.4json模块

json 模块是 Python 的一个内置模块，提供了处理 JSON（JavaScript Object Notation）数据的功能。它包含了多个函数，**用于将 Python 对象转换为 JSON 格式的字符串，以及将 JSON 字符串解析为 Python 对象**

###### 4.4.1将 Python 对象转换为 JSON 字符串

* json.dumps(obj, indent=None, separators=None, ensure\_ascii=True)：将 Python 对象转换为 JSON 格式的字符串。
* json.dump(obj, fp, indent=None, separators=None, ensure\_ascii=True)：将 Python 对象写入文件对象（file-like object）中，并以 JSON 格式进行序列化。

示例：

import json  
  
data = {  
 "name": "John Doe",  
 "age": 30,  
 "city": "New York"  
}  
  
json\_string = json.dumps(data)  
print(json\_string)  
  
with open("data.json", "w") as fp:  
 json.dump(data, fp)

###### 4.4.2将 JSON 字符串解析为 Python 对象

* json.loads(json\_string)：将 JSON 格式的字符串解析为 Python 对象。
* json.load(fp)：从文件对象（file-like object）中读取 JSON 数据并解析为 Python 对象。

示例：

import json  
  
json\_string = '{"name": "John Doe", "age": 30, "city": "New York"}'  
  
data = json.loads(json\_string)  
print(data["name"])  
  
with open("data.json", "r") as fp:  
 data = json.load(fp)  
 print(data["age"])

##### 4.5os模块

os 模块是 Python 的一个内置模块，提供了访问操作系统功能的函数。

###### 4.5.1文件和目录操作：

* os.getcwd()：获取当前工作目录的路径。
* os.chdir(path)：改变当前工作目录到指定路径。
* os.listdir(path)：返回指定目录下的所有文件和目录的列表。
* os.mkdir(path)：创建一个目录。
* os.remove(path)：删除指定的文件。
* os.rmdir(path)：删除指定的目录。

import os  
  
current\_dir = os.getcwd()  
print(current\_dir)  
  
os.chdir('D:\\project\\tutorial\\python基础')  
print(os.getcwd())  
  
files = os.listdir('D:\\project\\tutorial\\')  
print(files)  
  
os.mkdir('newDir')  
  
os.rmdir('newDir')

###### 4.5.2路径操作：

* os.path.join(path1, path2, ...)：将多个路径组合成一个路径。
* os.path.exists(path)：检查路径是否存在。
* os.path.isfile(path)：检查路径是否为文件。
* os.path.isdir(path)：检查路径是否为目录。
* os.path.abspath(path)：返回路径的绝对路径。

示例：

import os  
  
path = os.path.join('path', 'to', 'file.txt')  
print(path)  
  
exists = os.path.exists('path\\to\\file.txt')  
print(exists)  
  
is\_file = os.path.isfile('path\\to\\file.txt')  
print(is\_file)  
  
is\_dir = os.path.isdir('D:\\project\\tutorial\\python基础')  
print(is\_dir)  
  
absolute\_path = os.path.abspath('file.txt')  
print(absolute\_path)

###### 4.5.3系统信息：

* os.name：返回操作系统的名称。
* os.uname()：返回包含操作系统相关信息的元组（仅在类 Unix 系统上可用）。
* os.environ：包含环境变量的字典。
* os.getpid()：返回当前进程的进程 ID。
* os.system(command)：在子shell中执行系统命令。

示例：

import os  
  
print(os.name)  
  
if os.name == 'posix':  
 print(os.uname())  
  
print(os.environ)  
  
pid = os.getpid()  
print(pid)  
  
os.system('ls -l')

这些只是 os 模块提供的一些常见函数的示例。os 模块还支持更多功能，如进程管理、文件权限操作、路径处理等。你可以查阅 Python 官方文档以了解更多关于 os 模块的详细信息和其他可用的函数。

在 Python 的 `os.name` 属性中，`nt` 表示 Windows 操作系统。 Windows 系统的内核名称是 "Windows NT".  
- `'posix'`：表示类 Unix 操作系统，如 Linux、macOS 等。  
- `'nt'`：表示 Windows 操作系统。  
- `'java'`：表示 Java 虚拟机。  
- `'os2'`：表示 IBM OS/2 操作系统。

##### 4.6sys模块

sys模块提供了与 Python 解释器和操作系统交互的功能。sys 模块提供了许多与系统相关的函数和变量，可以用于访问命令行参数、处理标准输入输出、操作解释器和操作系统

1. 访问命令行参数：

import sys  
  
# 获取脚本名称  
script\_name = sys.argv[0]  
print("脚本名称:", script\_name)  
  
# 获取传递给脚本的参数  
args = sys.argv[1:]  
print("参数:", args)

1. 使用标准输入输出：

import sys  
  
# 从标准输入读取用户输入  
input\_data = sys.stdin.readline()  
print("输入:", input\_data)  
  
# 向标准输出打印输出  
sys.stdout.write("这是一条输出\n")  
  
# 输出错误消息  
sys.stderr.write("发生错误\n")

1. 系统配置和操作：

import sys  
  
# 获取当前操作系统平台标识符  
platform = sys.platform  
print("平台:", platform)  
  
# 获取 Python 解释器版本信息  
version = sys.version  
print("版本:", version)  
  
# 退出程序  
sys.exit(0)

1. 模块和导入：

import sys  
  
# 查看已导入的模块  
import math  
print("已导入的模块:", sys.modules.keys())  
  
# 查看模块搜索路径  
print("模块搜索路径:", sys.path)

##### 4.7re

re 模块用于在字符串中进行模式匹配、搜索和替换。

1. 匹配数字：\d+
   * \d 表示匹配任意一个数字字符。
   * + 表示前面的模式匹配多次，可以匹配一个或多个数字字符。
2. 匹配字母：\w+
   * \w 表示匹配任意一个字母或数字字符。
   * + 表示前面的模式匹配多次，可以匹配一个或多个字母或数字字符。
3. 匹配空白字符：\s+
   * \s 表示匹配任意一个空白字符，包括空格、制表符、换行符等。
   * + 表示前面的模式匹配多次，可以匹配一个或多个空白字符。
4. 匹配特定字符：[abc]
   * [abc] 表示匹配字符集中的任意一个字符，这个例子匹配字符 'a'、'b' 或 'c'。
5. 匹配重复字符：(abc)+
   * (abc) 表示将括号内的模式作为一个整体进行匹配，匹配 "abc"。
   * + 表示前面的模式匹配多次，可以匹配一个或多个 "abc"。
6. 匹配任意字符：.
   * . 表示匹配除换行符之外的任意一个字符。

匹配数字：

import re  
  
text = "abc 123 def 456"  
pattern = r"\d+"  
result = re.findall(pattern, text)  
print(result) # 输出: ['123', '456']

匹配邮箱地址：

import re  
  
text = "Contact us at info@example.com or support@example.com"  
pattern = r"\w+@\w+\.\w+"  
result = re.findall(pattern, text)  
print(result) # 输出: ['info@example.com', 'support@example.com']

替换文本：

import re  
  
text = "Hello, my name is John."  
pattern = r"John"  
replacement = "Alice"  
result = re.sub(pattern, replacement, text)  
print(result) # 输出: Hello, my name is Alice.

拆分字符串：

import re  
  
text = "apple,banana,cherry"  
pattern = r","  
result = re.split(pattern, text)  
print(result) # 输出: ['apple', 'banana', 'cherry']

更多用法查找python进阶。

##### 4.8gzip

在Python中，gzip 用于对数据进行压缩和解压缩。它提供了一种使用 GZIP 文件格式进行压缩和解压缩的简单方式。GZIP 是一种常用的压缩算法，通常用于压缩文件和网络传输的数据。

1. 压缩数据：

import gzip  
  
data = b"This is some data that needs to be compressed."  
compressed\_data = gzip.compress(data)

1. 解压缩数据：

import gzip  
  
compressed\_data = b"\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\x0b\xcd\xcc\xcb\x07\x00\x82Q\x0c\x05\x00\x00\x00"  
data = gzip.decompress(compressed\_data)

1. 压缩文件：

import gzip  
  
with open("file.txt", "rb") as f\_in:  
 with gzip.open("file.txt.gz", "wb") as f\_out:  
 f\_out.write(f\_in.read())

1. 解压缩文件：

import gzip  
  
with gzip.open("file.txt.gz", "rb") as f\_in:  
 with open("file.txt", "wb") as f\_out:  
 f\_out.write(f\_in.read())

使用 gzip 模块可以有效地压缩和解压缩数据，减少存储空间和网络传输的数据量。在压缩文件时，压缩后的文件通常会带有 .gz 扩展名，以示区分。

##### 4.9csv

CSV（Comma-Separated Values）是一种常见的文件格式，用于存储和传输结构化的表格数据。CSV文件由纯文本组成，使用逗号或其他字符作为字段之间的分隔符，每行表示一条记录，每个字段表示一列数据。

下面是一个示例，展示了一个简单的CSV文件的内容：

Name,Age,Email  
John,25,john@example.com  
Alice,30,alice@example.com  
Bob,35,bob@example.com

在这个示例中，CSV文件的第一行是标题行，描述了每一列的含义。接下来的每一行表示一个数据记录，每个字段使用逗号进行分隔。在这个例子中，每个记录包含三个字段：姓名、年龄和电子邮件。

在Python中，可以使用内置的 csv 模块来处理CSV文件。

1. 读取CSV文件：

import csv  
  
with open('data.csv', 'r') as file:  
 reader = csv.reader(file)  
 for row in reader:  
 print(row)

1. 写入CSV文件：

import csv  
  
data = [  
 ['Name', 'Age', 'Email'],  
 ['John', '25', 'john@example.com'],  
 ['Alice', '30', 'alice@example.com'],  
 ['Bob', '35', 'bob@example.com']  
]  
  
with open('data.csv', 'w', newline='') as file:  
 writer = csv.writer(file)  
 writer.writerows(data)

在读取CSV文件时，我们使用 csv.reader() 函数创建一个读取器对象，然后使用迭代器遍历每一行数据。在写入CSV文件时，我们使用 csv.writer() 函数创建一个写入器对象，并使用 writerows() 方法将数据写入文件。