SI100+ 2024 Python Lecture 5

Python 进阶

Part 0: Overview

- Python 中最为常用的数据结构：列表（`list`），字典（`dict`），以及元组（`tuple`）和集合（`set`）
- Python 常用的第三方库：`os`，`math`……
- 如何安装第三方库？`pip install` 与 `conda install`
- 一些巧妙的 Python 函数：`map`，`enumerate`……
- 类型标记：让你的代码更清晰
- 优美的代码：什么是好的代码？什么是差的代码？如何写出

Part 01: 常见的数据结构

什么是数据结构?

- 所谓“数据结构”，是计算机存储、组织数据的方式（什么破官方定义）
- 我们可以理解为，数据结构就像一个盒子，里面存着若干数据，我们可以按照一些特定的方式去读、写这些数据！
- CS 的同学以后会学习到“数据结构与算法”这门课，将会对其理论知识进行更深入的学习
- 在这里，我们**仅简单学习 Python 中常用的内置数据结构的特性和使用方法。**

列表，元组，字典和集合

这四个数据结构都是 Python 中很常见的数据结构。在详细介绍之前，我们对这四类数据结构进行一个简单的介绍：

- 列表和元组
  - 列表和元组**按顺序存储数据。**
  - 列表通过 `[]` 生成，而元组通过 `()` 生成。
  - 首先明确一点：列表和元组是**几乎完全相同**的数据结构，唯一的差别是：**元组是只读的。**
  - **很多操作都会产生元组**，例如：`x = 1,` 这个逗号就会导致元组的产生
- 字典
  - 字典通过 `{}` 生成，是存储**一个“键”（Key）和一个“值”（Value）之间一对一关系**的数据结构
- 集合
  - 相对而言用的很少的数据结构，特点是无序、不重复
  - 因此，经常**用集合来去重复**x

List（列表）

List 的生成、访问与修改

- Construct（构造，即“生成”一个列表）：`li = [1, 4, 3]`
  - 生成列表时，列表的元素之间用逗号分隔
  - `1, 4, 3` 的 Index 分别为 `0, 1, 2`
- Visit（访问，即根据索引“获取”列表的某个元素）：`li[index]`
  - 与字符串的索引一样，列表索引**从 0 开始**（这个命名法有其内在原因）
  - 例如：`li[0]` 会获得 `li` 的第一个元素：`1`
  -
- Change（修改，即修改列表的特定元素）：`li[index] = ...`
  - 例如：`li[0] = 5` 会让列表变成 `[5,4,3]`

List 的常用操作

- Slide（切片）：`li[from:to:step]`
  - 获得列表中的某一部分子列表
  - e.g.: `li[0:2]` 会获得 `[1, 4]`
- length（长度）
  - 通过`len(li)` 获取列表中元素个数
- operator运算符
  - '+' 可以连接两个list
  - 比如：`[1,2] + [3,4]` 会获得 `[1,2,3,4]`

List 的一些 Methods（方法）

- append `li.append(item)` 结尾追加一个元素
- insert `li.insert(idx, item)` 在指定位置插入一个元素
- remove `li.remove(item)` 删除指定值的元素
- pop `item = li.pop(index)` 删除指定位置的元素，并返回删除的元素的值
- index `obj = li.index(item)` 从列表中找出某个值第一个匹配项的索引位置
- sort `li.sort()` 排序，默认从小到大
- reverse `li.reverse()` 将列表反向

List

Nested list 嵌套列表

- 我们可以在 list 中嵌套其他 list
- 事实上，列表的元素可以是任意内容，但是我们并不推荐大家把什么乱七八糟的东西都装进列表里
    - **最好保持列表里的元素都是相同类型的**！

```
matrix = [[1, 2, 3], [4, 5, 6]] # 二维列表
```

Tuple（元组）

- Python 的元组与列表类似，**不同之处在于元组的元素不能修改。**
- 元组使用小括号，列表使用方括号。
- 元组创建很简单，只需要在括号中添加元素，并使用逗号隔开即可。

```
tup1 = ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5 )
tup3 = (50,) #元组中只包含一个元素时，需要在元素后面添加逗号
```

- 访问方法和list相同，只是元组中的元素值是不允许修改的。

Set（集合）

- 集合（set）是一个无序的不重复元素序列。
- 集合中的元素不会重复，并且可以进行交集、并集、差集等常见的集合操作
- 可以使用大括号 { } 创建集合，元素之间用逗号 , 分隔， 或者也可以使用 set() 函数创建集合

```
set1 = {1, 2, 3, 4}                 # 直接使用大括号创建集合
set2 = set([4, 5, 6, 7])        # 使用 set() 函数从列表创建集合
```

Dict（字典）

- 字典的每个键值 **key:value** 对用冒号 : 分割，每个Key-Value pairs之间用逗号 , 分割，整个字典包括在花括号 {} 中 ,格式如下所示：

```
d = {key1 : value1, key2 : value2 }
```

- 其中，`key1: value1` 就称之为 Key-Value pairs（键值对）
- Key 一般是唯一的、不可变的，如果重复，后加入字典的 Key-Value pairs 会替换前面加入的Key-Value pairs。值不需要唯一。
- 以下为一个生成字典的事例。

```
tinydict = {'Alice': '2341',
        'Beth': '9102',
        'Cecil': '3258'}
```

Dict（字典）

- Construct(构造): `di = {"1": 1, "2": 2, "3": 3}`
- Visit(访问): `print(di["1"])`
- Change(修改): `di["1"] = 100`
- Delete(删除): `del di["1"]`

Example

```
>>> tinydict = {'a': 1, 'b': 2, 'b': '3'}
>>> tinydict['b']
'3'
>>> tinydict
{'a': 1, 'b': '3'}
```

小结：Python 中的四种常见数据类型（列表，元组，集合，字典）

1. 列表（List）：有序，可更改，可以有重复的成员
2. 元组（tuple）：有序，不可更改，可以有重复的成员
3. 集合（set）：无序，无索引，没有重复的成员
4. 字典 （Dictionary）：无序，可更改，有索引，没有重复的成员

Part 02：Python 常用的第三方库：`os`，`math`……

- OS 库提供基本的与操作系统交互的功能，包括：
  - 文件路径管理：`os.path` 子库，可以遍历寻找文件、创建文件夹等
  - 进程管理：通过启动其他程序
  - 环境参数管理：获得系统中的一些环境参数

OS 库之文件路径管理

> 指定路径可以是绝对路径，也可以是相对路径！

- `os.path.exists(path)`：返回指定路径（可能是文件，也可能是文件夹）是否存在
- `os.file.isdir / os.file.isfile`：判断指定的路径是否为文件 / 文件夹
- `os.file.getsize`：获得指定路径所对应的文件大小
- `os.makedirs(path, exist_ok = True)`：根据指定路径创建文件夹。如果文件夹已经存在，则不进行操作（非常非常常用！）
- …… 还有很多！

OS 库之文件进程管理

- OS 库的进程管理最基本的执行方式是 `os.system`
- `os.system(command)` 相当于在命令行（Windows：Powershell；Linux：Bash 或其他终端）中执行特定指令。
  - 例如：`os.system("Genshin.exe")`：原神，启动！
    - 这是用的相对路径！
  - 再比如：`os.system("C:\\Windows\\System32\\calc.exe")`
    - 这是用的绝对路径！
    - 你的 Python 应该可以运行这个代码，会打开你的计算器~

OS 库之文件环境参数管理

- `os.chdir(path)`：修改当前程序操作的路径
- `os.getcwd()`：返回程序的当前路径
- `os.getlogin()`：获得当前系统登录用户名称
- `os.cpu count()`：获得当前系统的 CPU 数量
- `os.urandom(n)`：获得n个字节长度的随机字符串，通常用于加解密运算

Part 03：如何安装第三方库？`pip install` 与 `conda install`

pip 和 conda 都是**包下载工具**，可以帮我们下载第三方库。

`pip` 和 `conda`

- `pip` 是 Python 官方推荐的下载工具，但只能下载 Python 包
  - pip 从PyPI（Python Package Index）上拉取数据。上面的**数据更新更及时，涵盖的内容也更加全面**
  - pip 不能提供虚拟环境
- `conda` 支持安装一些 Python 之外的包，如 C++，R 等
  - conda 从 Anaconda.org 上拉取数据。虽然Anaconda上有一些主流Python包，但在数量级上明显少于PyPI，缺少一些小众的包
  - 不过 conda 安装会强制保证包的依赖要求都会被满足
  - conda 可以创建虚拟环境
- 选择 pip 还是 conda?
  - 见仁见智！并没有一个绝对正确的答案
  - 我选择用 conda 创建空环境，然后用 pip 安装库（成年人不做选择.jpg）

`pip` 的一些小技巧：换源

- `pip` 需要从 PyPI 上拉取数据，很多时候肥肠的慢，可能还会卡死
- 国内有一些镜像源，内容定期与 PyPI 同步，我们可以通过镜像源进行下载。

```
pip install numpy # 直接使用 pip 下载 numpy 库
pip install numpy -i https://pypi.tuna.tsinghua.edu.cn/simple # 使用清华源下载
```

- 如果你配置了环境变量，则你应该可以直接使用 `pip --version` 查看你的 pip 版本。
    - 如果遇到问题（例如：输入上述指令报错），请及时联系助教！

Part 04： 一些巧妙的 Python 函数

map

> `map()` 函数接收两个参数，一个是函数，一个是一个可以迭代的对象（如：列表、字典），`map` 将传入的函数依次作用到序列的每个元素，并把结果作为新的可迭代对象返回。

我们通过一个简单的例子来看 `map` 的作用

```
def f(x):
    return x * x

map(f, [1, 2, 3, 4, 5, 6, 7, 8, 9])
```

你会得到:

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

`map` 的一些用法

`map()` 作为高阶函数，事实上，它把运算规则进行了抽象。

因此，我们不但可以计算简单的 $f(x) = x^2$ ，还可以进行其他更加复杂的操作，比如，把一个 list 所有数字转为字符串：

```
list(map(str, [1, 2, 3, 4, 5, 6, 7, 8, 9]))
```

```
['1', '2', '3', '4', '5', '6', '7', '8', '9']
```

我们也可以通过 `map` 获取输入的数字：

```
list(map(int, input().split()))
```

- 这里 `split` 是字符串的一个方法，把字符串按空格分割
- 想一想，这个操作是如何实现读取并分割的?

`enumerate`

- `enumerate()` 函数用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。
    - 可以让我们的代码更简单

假设有一个列表 `lst = [1,1,4,5,1,4]` ，我们想获取其中每个下标对应的值。

不用 `enumerate` 的写法：

```
cnt = 0
for val in lst:
    print(f"{cnt}: {val}")
    cnt += 1
```

而使用 `enumerate` ：

```
for i, val in enumerate(lst):
    print(f"{i}: {val}")
```

Part 05： 类型标记：让你的代码更清晰

Type hints 即类型提示，是 Python 在 3.5 版本中加入的语法，并在 Python 3.6 基本可用。在此后的版本中，Type hints 的功能不断扩充，至今已经能够实现一个比较完善的静态类型系统。

正如其名称暗示的那样，Type hints 是“类型提示”而不是“类型检查”，Python 并不会在程序运行时检查你所标注的类型与变量的真实类型是否一致——如果不一致，也并不会产生错误。

Python 的 Type hints 例子

基本使用方法：标注 `x` 的类型应该是整型，而 `y` 是浮点型， `z` 是字符串

```
x: int = 114514
y: float = 1.114514
z: str = "24 years old, is student"
```

Python 的 Type hints 例子

进阶方法：需要 `import typing`（一个专门用来类型标注的官方库）

```python
from typing import List, Dict

x: List[int] = [1,2,4,5]
y: Dict[str, int] = {"SZHGG": 666, "SI100P": 6}
```

- 你能自己看懂这两个类型注释的含义吗?
- 事实上，Python 还支持更加复杂的类型注解，如联合类型、类型别名、Optional 等，我们不在这里赘述。

Part 06：优美的代码

6.1. 使用合理的命名规则

变量的命名

- 应该简洁且具有描述性。
- 常见命名规则：
  - 驼峰命名法（camelCase）：首字母小写，后面每个单词都大写
  - 下划线命名法（sname_case）：用下划线连接单词
- 一个项目中，代码命名风格应保持一致。
- 在某些特殊的部分可以用一些特殊标记：
  - 某个类的成员可以用 `m_` 开头：比如一个图书管理的类可以用 `m_books` 来命名。（你不知道类是什么？没关系，记住这一点，以后学会了这么写就行！）
  - 有些项目用后置下划线表示私有变量或者临时变量：`value_` 表示某些临时的值
  - 循环变量可以用 `ijk`，但如果有特殊含义建议单独命名

6.1. 使用合理的命名规则

函数的命名

函数的命名最好能够直接说明函数的作用。

```
# 不好的命名

def foo(l):
    return sum(l)

# 好的命名
def calculate_sum(input_list: List[int]) -> int:
    return sum(input_list)
```

6.2. 保持代码简洁

避免重复代码

我们需要尽量把相似的逻辑整理到一起：

```
# 不好的代码
def calculate_area_of_square(side):
    return side * side

def calculate_area_of_rectangle(length, width):
    return length * width


# 好的代码
def calculate_area(shape, *dimensions):
    if shape == 'square':
        return dimensions[0] * dimensions[0]
    elif shape == 'rectangle':
        return dimensions[0] * dimensions[1]
```

6.2. 保持代码简洁

善于使用内置库

```
# 不好的代码
def calculate_sum(numbers):
    total = 0
    for number in numbers:
        total += number
    return total

# 好的代码
def calculate_sum(numbers):
    return sum(numbers)
```

6.3. 良好的注释和文档

行内注释不要说一些没用的废话！要突出两个点：

- 这个代码的主要目的是什么，起到什么样的作用
- 如果代码的实现方式比较抽象，则需要做一定的解释

```
x = 19 # 对 x 赋值
max_retries = 19 # 设定最大的重试次数
```

6.3. 良好的注释和文档

每个类和函数用字符串单独描述其功能、参数和返回值

```
# 不好的文档字符串
def calculate_area(shape, *dimensions):
    pass

# 好的文档字符串
def calculate_area(shape, *dimensions):
    """
    计算不同形状的面积。
    Parameters:
        shape (str): 形状类型，可以是'square'或'rectangle'。
        *dimensions (float): 形状的尺寸。
    Return:
        float: 形状的面积。
    """
    pass
```

6.4. 良好的书写习惯

- 使用统一的长度作为缩进符号
  - 我选择四格、Tab 缩进（而非空格缩进）
- 行内代码适当使用空格作为分割

```
# 不好的示范
for i in range(5):
  v=(i*i+i**2+1)/4
  lst = {i:v,i*2:v,i*3,v}
# 相对比较好的写法
for i in range(5):
  v = (i * i + i**2 + 1) / 4
  lst = {
      i       : v,
      i * 2   : v,
      i * 3   : v
  }
```

Part 06：优美的代码

总而言之：在度过新手阶段之后，代码的语法将不再成为大家写代码时最大的困扰。在这时候，大家一方面需要开始关注如何梳理代码的逻辑，快速写出准确的代码，另一方面则需要让自己的代码更加简洁优美、具有可维护性。

上述提到的一些对代码优美性的要求并不是全部。大家在平时写、读代码的时候也要勤加思考，写出更好的代码哟！