Lecture 6: Python代码调试

Revision: *2023/05/10*

调试是理解代码的好方法，在写代码的过程中，往往需要在调试状态下写代码。

关键字：边调试边写代码

1. 基于调试理解代码

调试（Debug），从字面意思上来看主要是用来查找/定位程序中出现的错误/缺陷的。但“bug”是个很大的概念，既可以是代码中的缺陷，也可以是我们不理解的地方；我们的代码中出现问题，很多时候，并不是具体出错的地方出错了，可能是一开始就理解错了。

比如，我们要统计一次比赛的各个选手的成绩。我们可能会有很多疑问：

1. `player_scores` 字符串中的 “\” 是何意?
2. 由字符串给出的 `player_scores` 与从文件中读取的选手得分有何不同?

```
player_scores = '\
    李: 17 6 3 3   2 5 3 6 8\
    楼: 6  9 5 10 14 3 8 4 6\
    翁: 3  4 8 9   3 5 4 6 6'
```

```
# scores.txt
李: 17 6 3 3   2 5 3 6 8
楼: 6  9 5 10 14 3 8 4 6
翁: 3  4 8 9   3 5 4 6 6
```

这些问题，我们可以去问别人，也可以百度，但这些都不是最高效的做法，高效的做法是直接调试，观察变量在实际运行过程中的真实数值。

2. 基于调试编写有挑战的代码

在高尔夫足球赛的得分统计问题中，最有挑战的是对输入的得分字符串的解析。为了正确解析字符串，最好的做法就是在调试状态下写代码。这样做可以在写代码的同时，看到代码的执行情况，从而在很大程度上避免了写出错误的代码。因此，好的集成开发环境（IDE）就显得尤为重要。

2.1 分割 `self.all_scores`

```
all_scores = self.all_scores.replace(': ', ':')
all_scores

all_scores = all_scores.split(':')

```

2.2 分析分割后的结果

1. `all_scores` 的第 `0` 个和最后一个（第 `-1` 个）元素比较特殊；
2. 如何获取 `all_scores` 的除了第 `0` 个和第 `-1` 个之外的其它 item?

```
mid_items = all_scores[1:-1]
```

3. 如何对 `all_scores` 的其它 item 进行分析?

```
# 取 all_scores 的第一项 item_1,
#  可以发现它是由上一个选手的 scores 和 当前选手的姓名构成的
item_1 = mid_items[0]
scores_in_item_1 = item_1[:-1]

# 通过调试可以发现，scores_in_item_1 的末尾可能是 `\r\n`,
# 也可能是多个空格，如何去掉这些冗余的空白符?
scores_in_item_1.strip()

# 但考虑到我们的目标是分离分数的各子项，也可以不进行 .strip() 操作
```

4. Python中的枚举：
   a. 通常不是通过索引获取迭代器中的元素，而是直接枚举；
   b. 如果需要获取迭代器中元素的索引，可以使用 `enumerate()` .

```
[item for item in all_scores[1:-1]]

x = ['a', 'b', 'c']
for i in range(len(x)):
    print(x[i])

```

```
for item in x:
    print(item)

for i, item in enumerate(x):
    print(i, item)
```

2.3 解析出 `mid_scores`：

```
mid_scores = [(item[:-1].strip(), item[-1]) for item in all_scores[1:-1]]
```

a. 理解tuple：

```
for score, name in x:
    print(score)
    print(name)

a = ('hello', 'world')
x, y = a
```

2.4 方法1：不使用拼接，直接构建字典。

```
players = {}
name = all_scores[0]
for score, next_name in mid_scores:
    players[name] = score
    name = next_name
players[name] = all_scores[-1]
```

2.5 方法2：基于拼接构建字典。

1. 嵌套列表展开（参考：https://blog.csdn.net/XX_123_1_RJ/article/details/80591107）：

```
# 1) 普通方法
mid_scores_2 = []
for _ in mid_scores:  # 这里比较神奇的地方有2: 1) _ 表示的是啥?
    mid_scores_2 += _  # 2) 怎么可以直接追加 _? 什么原理? 怎么验证?
print(mid_scores_2)

```

```
# 2）列表推导
mid_scores_2 = [i for item in mid_scores for i in item]

# 3）使用sum
mid_scores_2 = sum(mid_scores, [])
```

a. 如何理解 sum? 如何查看帮助?

```
>>> help(sum)
Help on built-in function sum in module builtins:
sum(iterable, /, start=0)
    Return the sum of a 'start' value (default: 0) plus an iterable of num

    When the iterable is empty, return the start value.
    This function is intended specifically for use with numeric values and
    reject non-numeric types.
```

2. 拼接:

```
x = [all_scores[0], mid_scores_2, all_scores[-1]]

x = [all_scores[0]] + mid_scores_2 + [all_scores[-1]]

x = all_scores[:1] + mid_scores_2 + all_scores[-1:]
```

3. list2dict:

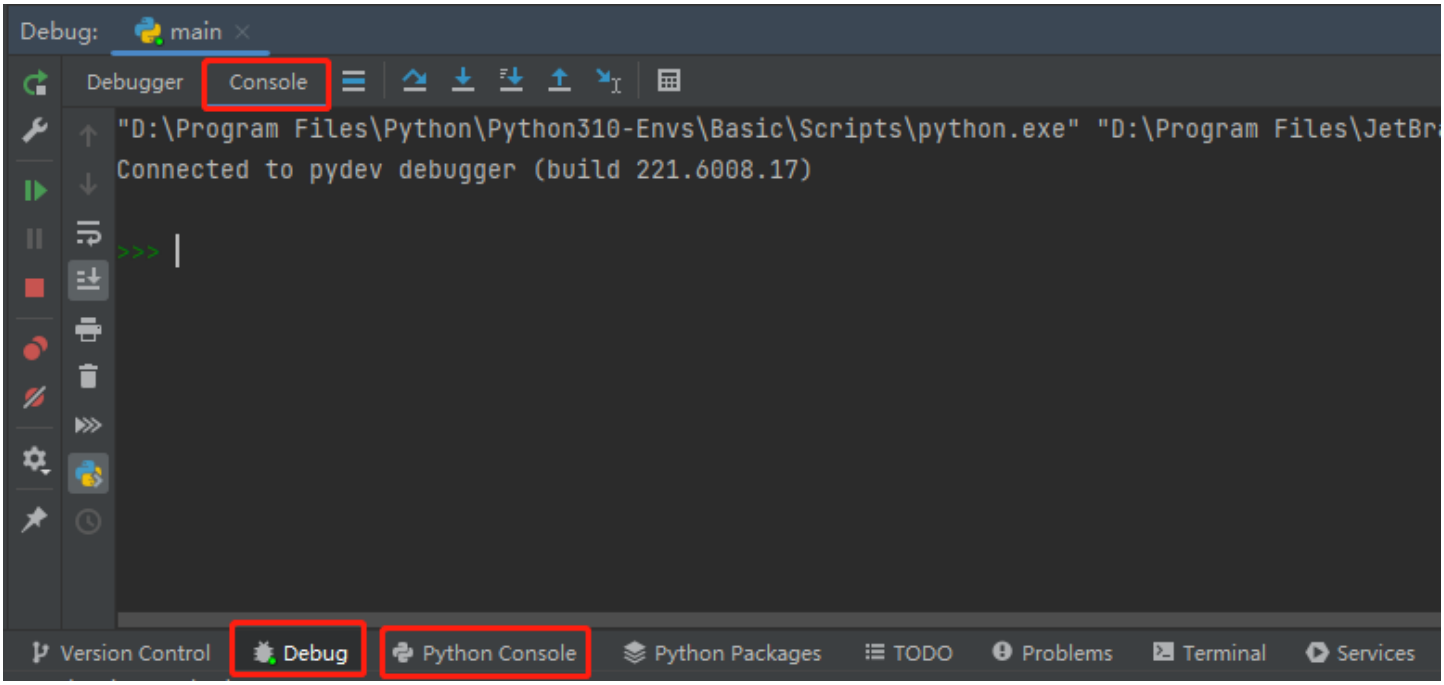
```
players = dict(zip(x[::2], x[1::2]))
```

4. 合并:

```
mid_scores = sum(mid_scores, [])
x = all_scores[:1] + mid_scores + all_scores[-1:]
players = dict(zip(x[::2], x[1::2]))
```

3. 问题

1. Python Console 与 Debug 中的 Console 的区别?



2. 如何计算由数字组成的 `list` 的各项之和?
3. 如何把一个字符串转换成由字符串的各字符组成的 `list` ?
4. 重载（overload）、重写/覆盖（overwite/override）。Python中没有重载的概念，详见：https://www.jianshu.com/p/48f6b391ecee。