夯实 python 基础

为进阶做准备

python 实战圈出品

20180812

第一次码代码

从今天开始，咱们正式进入敲代码的阶段。首先，不要害怕这件事情。敲代码听上去挺牛 x，其实和写作文一样。只要你有一个题目，然后进行拆解就可以了，至于文采，也就是代码漂不漂亮是另一回事情了，以后慢慢给大家介绍。但是，我想很多朋友已经看过很多 python 基础入门书籍，或者购买了很多 python 视频课，但是还是不知道怎么写代码。其中的原因只有一个，就是编码的思路没有转变。希望咱们实战圈能给大家带来的，不单单是编码的技能，更是编码思想的提升，也就是编程思维转变。

转变 python 编码思路的唯一一个方法就是实战。因为实战中你才能发现：原来我的代码因为少或者多了一个字母，没有出现结果，又或者我的代码中用的是中午字符而不是英文，更令人麻烦的是代码可以运行，没有语法错误，但是结果不是自己想要等等。但是当你慢慢通过模仿其他人的代码得到运行结果，是不是也很开心。然后慢慢的自己能学会设计代码，还有可能去教会别人写代码。这条路的捷径只有一个“动手写代码”，以后我们实战圈会给大家准备各种各样的实战项目，希望大家都能参与进来，一起把敲代码这件事情搞定。真正的从敲代码这种”体力活“，变成设计代码的“脑力劳动”，最后再变成敲代码的“体力活”。

总之，咱们的目标就是，给定任何一个编码项目，脑中离开有编码思路，剩下的就是动手写一下而已。

数据--程序的原材料

真正开始编码之前，咱们需要搞明白两件事情，第一个是编码规则；第二个是什么是数据。在咱们实战圈，编码用的是一种叫做 python 的高级编程语言。既然是语言，肯定有语法，并且也需要素材，你可以把它想成中文或者英文。语言的语法也就是编码的规则，这也是 python 编程基础，《如何七天入门 python》项目都是在说这个事情。等基础语法完成的时候，我们就可以进入项目实战阶段。

接着，咱们说一下另外一个事情：数据。

数据简单说就是在电脑中的任何东西，比如听的音乐、看的电影、读的文章等等。python 编程就是要处理这些数据，利用自己的语法规则对其加工处理，然后呈现出你想要的结果数据，所以你可以把程序或者代码看成一个“服装加工厂”：布料就是程序中使用的数据，服装机器就是根据语法处理数据，衣服就是代码输出的结果。

有时数据太多，不好理解。对其进行分类，是一个方法，非常便于理解与处理。在 python 语言中，常见的数据分类有，也就是 python 数据类型，如下所示。

数据类型	定义
数字	和数学中的一样。分为整数与浮点数（小数）
字符串	用引号引起来的一系列文本字符，比如‘I am a boy.’
布尔类型	真和假（True 和 False)， 主要用来判断
空	代表无的概念，也就是 None
结构体	特殊数据类型，比如列表、字典等
自定义类型	自己定义的类型 ，主要是面向对象中使用

注意：第一次接触数据类型的圈友可能有点迷糊，以后的文章会详细给大家介绍。

学会写注释，方便你我他

注释就是在你的代码里面添加上解释说明。代码是告诉阅读的人做什么事情，而注释是告诉阅读的人为啥这么做。这点在学习编程的时候，特别重要，尤其刚开始的时候一定要养成写注释的习惯，不要谦虚麻烦，因为写注释真的很方便阅读代码的人理解。

在实际工作中，项目一般都很大，需要协作完成。如果有人没写注释就给下一个阅读，这是特别痛苦的一件事情。代码漂不漂亮有时候也要看注释是不是全面。不过，据我所知，国内的很多人，即使工作很多年的老程序员也不喜欢写注释。我有时候看国内的 python/java 视频教程，很多老师也不写注释，因为麻烦，认为是多干活。这个观点是错误的，因为即使自己写的代码，过上 2 年，我相信你自己都不记得写的是什么了。另外，据我了解，一些大公司代码注释写的都非常全面，比如 Google、Oracle 等。

在 Python 中，单行注释用井号（#）标识，也就是#后面的内容；多行注释用一对三引号（''' '''）表示。单行注释一般用在某一行的解说说明，而多行注释一般用在真个文本或者某一个代码区域的解释说明。

python 三剑客工具会忽略，不会在运行。如下所示。Python 解释器就会忽略该注释，只是输出 Hello World. 请在你的第一次作业上添加上注释吧。

例子：多行注释与单行注释

```
'''
```

```
本文为第二天：夯实基础的内容
主要包括：
    数据类型
    注释

'''

#定义变量，并使用 print 函数打印出来
my_name = "刘德华"
print(my_name)
```

注意：并不是每一行都代码都需要注释，只有关键的地方需要，比如。

- **新的语法点**
- **代码要解决的问题**
- **重要的细节或者结论**
- **等等**

什么是常量与变量？

常量，顾名思义就是值不能被改变的量，比如 5、10 等数字或者一个字符串的文本。它字面的意思就是本身的含义，不需要多的解释。

与常量相对应的就是变量，顾名思义就是值一直在改变的量。正式因为值在改变，我们需要给它取一个名字，也就是标识符。在 python 编程中，我们叫标识符为变量名，并且使用等号（=）把变量名和值关联起来，具体的语法是：

变量名 = 值

例子：

```
`python`
#定义变量，并使用 print 函数打印出来
```

```
#my_name 是变量名，刘德华为值。
#变量名不变，值可以变，比如换成周杰伦
my_name = "刘德华"
print(my_name)
```

注意：1. 变量是存在内存中的

2. python 是大小写敏感的，也就是 my_name 与 My_name 是两个不同的变量。

变量命名规则

变量的命名是有一定的规则的。如果违背则会出错，具体如下：

- **只能是字母、数字和下划线，但是不能数字开头，但是可以结尾。例子：3_log = 'This is a log file'（错误);log_3 = 'This is a log file'(正确)**
- **不能包含空格，否则认为是语法错误。比如 my name(错误的)，解决方法：使用（_）链接起来，变成 my_name**
- **千万不能使用 python 中的关键字作为变量名，比如 if = 2 等（x）**

变量命名方法

在符合变量命名规则的前提下，变量名最好简短易懂，也就是从变量名就能看出代表的意思。比如 my_name 肯定比 a 好懂（千万不要使用 a b c 表示变量名字）。当变量需要两个以上单词表示时候，常用的命名方法有两个。有兴趣的可以搜索（匈牙利命名法、驼峰命名法和帕斯卡命名法）

第一个命名方法： 驼峰式大小写：第一个单字以小写字母开始；第二个单字的首字母大写，例如：firstName、lastName。或者 每一个单字的首字母都采用大写字母，例如：FirstName、LastName、CamelCase，也被称为 Pascal 命名法

第二个命名方法：两个单词直接不能使用连接号（-）或者空格链接，但是有时候使用下划线，比如 first_name、last_name

数字类型

整数

整数也就是 int 类型，在 Python 中，可以直接对整数进行算数运算，具体如下：

操作	操作符
加	+
减	-
乘	*
除	/
取模	%
幂	**
取整除	//

例子：

```
'''
  整数运算
'''
#加法
add = 3 + 4
#Python 中,format 方法是格式化输出，也就是在{}的地方替换为变量的值。后面项目实战中经常用到
print('3+4 的值是 {}'.format(add))

#减法
sub = 10 - 8
print('10 - 8 的值是{}'.format(sub))

#乘法
```

```
multi = 23 * 3
print(' 23 * 3 的值是{}'.format(multi))

#除法
div = 10 / 2
print('10 /2 的值是{}'.format(div))

#取模，返回除法的余数
delivery = 7 % 3
print('7%3 的值是{}'.format(delivery))

#取整除，返回商的整数
round_number = 7 // 3
print(' 7 //3 的值是{}'.format(round_number))

#幂运算 -- X 的几次方
power = 7 ** 3
print('7**3 的值是{}'.format(power))
```

运行结果为：

```
3+4 的值是 7
10 - 8 的值是 2
 23 * 3 的值是 69
10 /2 的值是 5.0
7%3 的值是 1
 7 //3 的值是 2
7**3 的值是 343
```

浮点数

带小数点的数字都是浮点数，也可以进行类型整数的运算，比如加减乘除

等。

例子：

```
'''
   浮点数运算
'''
print('以下为浮点数运算例子')
#加法
add = 0.2 + 0.1
#Python 中,format 方法是格式化输出，也就是在{}的地方替换为变量的值。后面项目实战中经常用到
print('0.2+0.1 的值是 {}'.format(add))

`补充内容`
# 格式化输出 format,
# 在 python3.6 以上版本中，为了减少{}，可以使用 f'的方法
com = 'Complex'
comp = 'complicated'

#3.6 以下的用法
print('\n 3.6 以下的 format 用法：')
print('{} is better than {}'.format(com,comp))
#3.6 以上版本的用法
print('\n 3.6 以上的 format 用法：')
print(f'{com} is better than {comp}')

#减法
sub = 10.9 - 8.1
print('10.9 - 8.1 的值是{}'.format(sub))

#乘法
multi = 0.1 * 3
```

```
print(' 0.1 * 3 的值是{}'.format(multi))

#除法
div = 10.0 / 2.0
print('10.0 /2.0 的值是{}'.format(div))

#取模，返回除法的余数
delivery = 7 % 4.3
print('7%4.3 的值是{}'.format(delivery))

#取整除，返回商的整数
round_number = 7 // 4.3
print(' 7 //4.3 的值是{}'.format(round_number))

#幂运算 -- X 的几次方
power = 7 ** 2.0
print('7**2.0 的值是{}'.format(power))
```

以下为浮点数运算例子运行结果

```
0.2+0.1 的值是 0.30000000000000004
10.9 - 8.1 的值是 2.8000000000000007
 0.1 * 3 的值是 0.30000000000000004
10.0 /2.0 的值是 5.0
7%4.3 的值是 2.7
 7 //4.3 的值是 1.0
7**2.0 的值是 49.0
```

注意：结果包含的小数位数可能是不确定的，这个是可以忽略的。

布尔类型

Python 支持布尔类型的数据，布尔类型只有 True 和 False 两种值，但是布尔类型有以下几种运算：

与运算：只有两个布尔值都为 True 时，计算结果才为 True。

例子：

```
True and True   # ==> True
True and False  # ==> False
False and True  # ==> False
False and False  # ==> False
```

或运算：只要有一个布尔值为 True，计算结果就是 True。

例子：

```
True or True   # ==> True
True or False  # ==> True
False or True  # ==> True
False or False  # ==> False
```

非运算：把 True 变为 False，或者把 False 变为 True：

例子：

```
not True   # ==> False
not False  # ==> True
```

布尔运算在计算机中用来做条件判断，根据计算结果为 True 或者 False，计算机可以自动执行不同的后续代码。

在 Python 中，布尔类型还可以与其他数据类型做 and、or 和 not 运算，请看下面的代码：

例子：

```
#布尔类型
a = True
print(a and 'a=T' or 'a=F')
```

结果为：

```
a=T
```

计算结果不是布尔类型，而是字符串 'a=T'，这是为什么呢？因为 Python 把 0、空字符串''和 None 看成 False，其他数值和非空字符串都看成 True，所以：True and 'a=T' 计算结果是 'a=T'。继续计算 'a=T' or 'a=F' 计算结果还是 'a=T'

要解释上述结果，又涉及到 and 和 or 运算的一条重要法则：短路计算。

1. 在计算 a and b 时，如果 **a** 是 **False**，则根据与运算法则，整个结果必定为 **False**，因此返回 **a**；如果 **a** 是 **True**，则整个计算结果必定取决与 **b**，因此返回 **b**。

2. 在计算 a or b 时，如果 **a** 是 **True**，则根据或运算法则，整个计算结果必定为 **True**，因此返回 **a**；如果 **a** 是 **False**，则整个计算结果必定取决于 **b**，因此返回 **b**。

所以 **Python** 解释器在做布尔运算时，只要能提前确定计算结果，它就不会往后算了，直接返回结果。

字符串类型

什么是字符串

字符串就是一系列字符。在 Python 中，单引号、双引号或者三引号里面的内容就是字符串。如何字符串中包括单引号或者双引号，python 使用反斜线（\）对字符串中的字符进行转义

例子：

```
#单引号里面的文本就是字符串
'I am a boy'

#双引号其实和单引号一样，一般推荐使用单引号
"欢迎您加入 python 实战圈"

#三引号表示的字符串，一般表示很长的文字，只要引号没有结束就可以一直写。
#一般用来写文本注释
'''
我们实战圈的第一个项目就是<如何七天入门 python>，
每一天都有安排学习内容， 只需要 40 分钟就可以搞定，
学完以后，记得写作业并且提交到知识星球。
刚开始，咱们节奏放缓慢一些。计划三天更新一次内容。
希望都您能参与进来。
'''
```

```
#转意字符串(\n)
command = 'Let\'s go!'
print('\n 使用转移字符输出 ： ',command)
```

结果为：

使用转移字符输出 ： Let's go!

字符串的基本用法

- 添加空白

在编程中，一定的空白输出是为了方便阅读。Python 常用的添加空白的方法有制表符(\t)、空格或者换行符(\n)。

例子：

```
#添加空白 \n 表示换行 \t 表示制表符，把文字空两格输出
#制表符可以组合使用
print("欢迎来到 python 实战圈,\n")
print('\t 你想要学习 PYTHON 的哪方面内容，请留言 ')
```

结果为：

欢迎来到 python 实战圈,

你想要学习 PYTHON 的哪方面内容，请留言

- 链接字符串

拼接字符串就是把两个或两个以上的字符串合并在一起。该操作在项目中经常用到，比如爬虫的时候，网页的正则表达式（以后会介绍）太长，可以用拼接的方法链接起来；也可以把两个变量的字符串拼接为一个等等用法。Python 使用加号（+）来链接字符串。

例子：

```
#链接字符串,使用加号 +
log_1_str = 'The error is a bug.'
log_2_str = ' We should fix it.'
log_str = log_1_str + log_2_str
print('\n 拼接后的字符串就是：',log_str)
```

结果：

拼接后的字符串就是： The error is a bug. We should fix it.

字符串的常见运算

- 使用方法修改字符串的大小写

在 python 中，你会经常听到的两个名词是 函数和方法。函数就是一个能独自完成特定任务的独立代码块 ，可以被调用；方法是面向对象编程语言中使用到的名词.Python 是面向对象的编程语言，面向对象就是一切都对象，比如你我他，统一为人（people），人就是一个对象。人可以走路(run)，走路就是一个方法。合起来就是 people.run()

例子：

```
#字符串大小写转换
welcome = 'Hello, welcome to python  practical circle'
# 每个单词的首字母大写， title()
print('\n 每个单词的首字母大写： ', welcome.title())
# 段落的首字母大写， capitalize()
print('\n 段落的首字母大写： ',welcome.capitalize())
# lower(), 所有字母小写
print('\n 所有字母小写： ',welcome.lower())
# upper(), 所有字母大写
print('\n 所有字母大写： ',welcome.upper())
# 大写转小写，小写转大写
print('\n 大写转小写，小写转大写： ',welcome.swapcase())
#String.isalnum() 判断字符串中是否全部为数字或者英文,符合就返回 True，
```

不符合就返回 False，如果里面包含有符号或者空格之类的特殊字符也会返回 False。

```
print('\n 判断字符串是否全部为数字或者英文',welcome.isalnum())
#String.isdigit() 判断字符串中是否全部为整数
print('\n 判断字符串中是否全部为整数', welcome.isdigit())
```

结果为：

```
每个单词的首字母大写： Hello, Welcome To Python Practical Circle
段落的首字母大写： Hello, welcome to python practical circle
所有字母小写： hello, welcome to python practical circle
所有字母大
写： HELLO, WELCOME TO PYTHON PRACTICAL CIRCLE
大写转小写，小写转大
写： hELLO, WELCOME TO PYTHON PRACTICAL CIRCLE
判断字符串是否全部为数字或者英文 False
判断字符串中是否全部为整数 False
```

- 字符串分割

按照某种特定的方法，把字符串分割开。这个在数据分析中经常用到。

```
#字符串分割
string_example = 'Now is better than never .'
# 分割
print('分割字符串： ',string_example.split())
#安装某一个字母分割
print('按照指定的字母分割字符串: ',string_example.split('n'))
# 去掉换行符，以换行符分割成列表
#splitlines() 以换行为分割
print('以换行符为分割','1+2\n+3+4'.splitlines())
```

结果为：

分割字符串： ['Now', 'is', 'better', 'than', 'never', '.']

按照指定的字母分割字符串：['Now is better tha', ' ', 'ever .']

以换行符为分割 ['1+2', '+3+4']

- 字符串删除两边空白

去除字符串两端的空白，在数据清理的时候经常被用到。常见的操作是去除两端或者一段的空格。

例子：

```
#删除两边的空白
love_python = '  Hello, Python Practical Circle  '
# 去除字符串两端的空白
print('去除字符串两端的空白',love_python.strip())
# 去除字符串右侧的空白
print('去除字符串右侧的空白',love_python.rstrip())
# 去除字符串左侧的空白
print('去除字符串左侧的空白',love_python.lstrip())
```

结果为：

去除字符串两端的空白 Hello, Python Practical Circle

去除字符串右侧的空白 Hello, Python Practical Circle

去除字符串左侧的空白 Hello, Python Practical Circle

- **其他注意事项**

python 中的字符串操作非常的多，以上只是列出了部分常用的操作。以后我们学习过程中慢慢补充。有一点需要注意的是 python 中的字符串并不允许修改值，只允许覆盖值。也就是字符串只能重新赋值，不能修改其中的一个字母。

字符串的切片

切片操作（slice）是 pytho 中经常用到的操作。字符串的切片就是可以从一个字符串中获取子字符串（字符串的一部分）。我们使用一对方括号、起始偏移量 start、

终止偏移量 **end** 以及可选的步长 **step** 来定义一个分片。

格式： **[start:end:step]**

- **[:]** 提取从开头（默认位置 **0**）到结尾（默认位置**-1**）的整个字符串
- **[start:]** 从 **start** 提取到结尾
- **[:end]** 从开头提取到 **end - 1**
- **[start:end]** 从 **start** 提取到 **end - 1**
- **[start:end:step]** 从 **start** 提取到 **end - 1**，每 **step** 个字符提取一个
- 左侧第一个字符的位置/偏移量为 **0**，右侧最后一个字符的位置/偏移量为**-1**

几个特别的 **examples** 如下：

提取最后 **N** 个字符：

```
>>> letter = 'abcdefghijklmnopqrstuvwxyz'>>> letter[-3:]'xyz'
```

从开头到结尾，**step** 为 **N**：

```
>>> letter[::5]'afkpuz'
```

将字符串倒转**(reverse)**， 通过设置步长为负数：

```
>>> letter[::-1]'zyxwvutsrqponmlkjihgfedcba'
```

字符串的编码问题

Python 3 版本中，字符串是以 Unicode 编码的，也就是说，Python 的字符串支持多语言，**比如中文。**当你的源代码中包含中文的时候，在保存源代码时，就需要务必指定保存为 UTF-8 编码。当 Python 解释器读取源代码时，为了让它按 UTF-8 编码读取，我们通常在文件开头写上这行：

```
# -*- coding: utf-8 -*-
```

注释是为了告诉 Python 解释器，按照 UTF-8 编码读取源代码，否则，你在源代码中写的中文输出可能会有乱码。但是申明了 UTF-8 编码并不意味着你的.py 文件就是 UTF-8 编码的，必须并且要确保文本编辑器正在使用 UTF-8 without BOM 编码。

无论是爬虫还是数据分析的时候，我们经常看到有一个转换编码的代码：

```
encode('utf-8')
```

由于 Python 的字符串在内存中以 Unicode 表示，一个字符对应若干个字节。如果要

在网络上传输，或者保存到磁盘上，就需要把 str 变为以字节为单位的 bytes。大家看到这行代码表示为了编码格式的转换即可。

各种类型转换

python 中，各个数据类型是可以互相转化的，并且可以使用 type()函数产看某一个变量的类型。

语法：type(变量名) 用来查看变量的数据类型

type()在实际项目中经常用到，因为只有知道了变量是什么类型才可以进行相应的运算，比如字典类型还是列表类型，会有不同的运算（后面会有介绍）。类型转换在项目实战中也经常用到，比如一个超市的月销售额是一个字符类型，则需要转化为数字类型才可以进行统计，比如计算平均数等。具体转化等语法是

float(a) 将变量 a 转化为浮点数

int(b) 将变量 b 转化为整数

str(c)将变量 c 转化为字符串

其中 a,b,c 为任意变量类型

```
'''
   各个数据类型转换
'''

print('\n 各个数值类型的转换')
number = 100

#number 的数据类型是 整型，用 int 表示
print('number 的数据类型是：')
print(type(number))

#讲整数转化为浮点数
float_number = float(number)
print('\nfloat_number 的数据类型是:')
```

```
print(type(float_number))

#讲整型转化为字符串数
print('\nnumber 转化为字符串类型')
str_number = str(number)
print('str_number 的数据类型是:')
print(type(str_number))

#将字符串转换为整型 int()或者浮点数 float()
print('\nstr_number 转化为数字类型')
int_str_number = int(str_number)
float_str_number = float(str_number)
print('int_str_number 的数据类型是：')
print(type(int_str_number))
print('float_str_number 的数据类型是:')
print(type(float_str_number))
```

各个数值类型的转换例子的运行结果是

```
number 的数据类型是：
<class 'int'>

float_number 的数据类型是:
<class 'float'>

number 转化为字符串类型
str_number 的数据类型是:
<class 'str'>

str_number 转化为数字类型
int_str_number 的数据类型是：
<class 'int'>
float_str_number 的数据类型是:
```

```
<class 'float'>
```

欢迎扫码加入 Python 实战圈