Lab00 实验讲义

同学们是否听说过“编程”这个看起来很厉害的东西，但又不知道它到底是什么？是否因为看到过电影里的黑客对着黑色的屏幕敲了一串字母最后酷眩地按下回车键，从而对计算机的世界感到好奇？

在本次实验中，我们将会用Python3这个编程语言为主来教会大家如何编写代码、如何运行代码，以及教大家如何像黑客一样对着你的电脑键盘一阵“乱敲”，然后按下回车键耍帅！当然，我们还需要教会你一些不那么酷炫的事情——如何完成作业以及提交作业。

废话不多说，让我们开始吧。

1. 环境配置

> 工欲善其事，必先利其器

1.1 Python3下载与安装

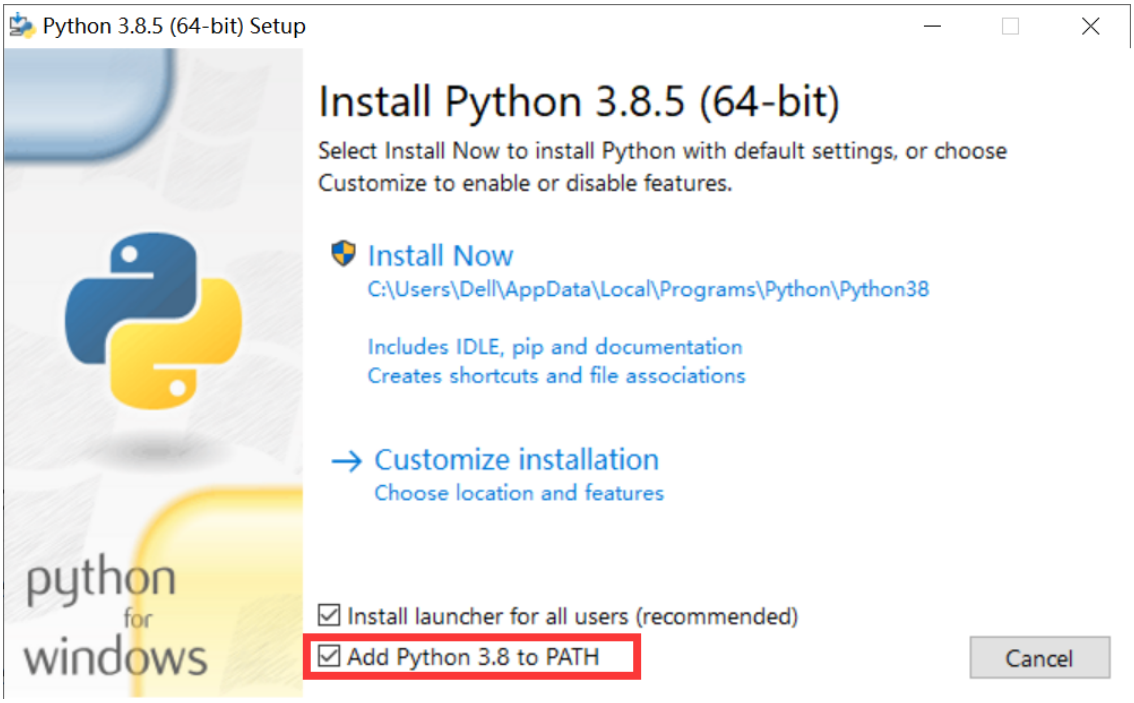
在本课程的实验中，我们主要以Python3这个编程语言为主来教大家编程。我们想要用它进行编程，首先需要在自己的电脑上安装运行Python3的环境。

> 大家平时可能听得比较多的说法是Python，而不是Python3。实际上，Python3表示Python这个语言的“3号版本”。聪明的同学可能想到，是不是Python还有其它版本？没错，Python还有一个“2号版本”，叫做Python2。那还有Python1吗？有是有，不过这是1991年发布的东西，现在可能已经见不到了。Python3和Python2有很多不同的地方。但Python3比之Python2更加的新，也是被主流所使用的（Python2正逐渐被淘汰），因此我们的教学也采用了Python3。在实验讲义和以后的课程中，如果我们没有明确说明，那么Python指的都是Python3。

目前最主流操作系统有Windows系统、macOS系统和Linux系统。考虑到同学们可能使用各种不同的操作系统，本实验讲义的演示会兼顾这三种主流操作系统。但在这三种系统操作流程相似的地方，我们主要以Windows 10为例来做演示。若在按照实验讲义操作的过程中有难以解决的问题，欢迎大家在实验课上或课程群中随时提问。

接下来，让我们来看看如何在这三种系统上安装运行Python3的环境。

- **Windows系统**：主要有两种途径（二选一即可）。
    - 途径一：在Python官方网站下载Python3安装包，你可以选择你喜欢的版本，但出于减少奇怪问题的考虑，建议你选择3.6之后的版本。这里以3.8.5为例，绝大多数同学的Windows系统为64位系统，可直接点击下载链接。拥有其他架构的Windows系统的同学可在Files目录自行选择下载链接。下载完成后请按照下图的示意，勾选“Add Python 3.8 to PATH”，随后点击“Install Now”，等待安装完成后关闭安装程序。



    - 途径二：打开Windows系统内置的Microsoft Store，搜索“Python 3.8”，点击安装即可。
- **macOS系统**：主要有两种途径（二选一即可）。
    - 途径一：与Windows系统下的途径一相似，在Python官方网站下载Python3安装包，可直接点击下载链接，在安装界面一直点击”继续“或”安装”。
    - 途径二：已安装homebrew的同学可在终端输入`brew install python3`完成安装。
- **Linux系统**（以Ubuntu 20.04 LTS为例，下同）：在终端输入`sudo apt install python3`即可完成Python3的全部安装流程。

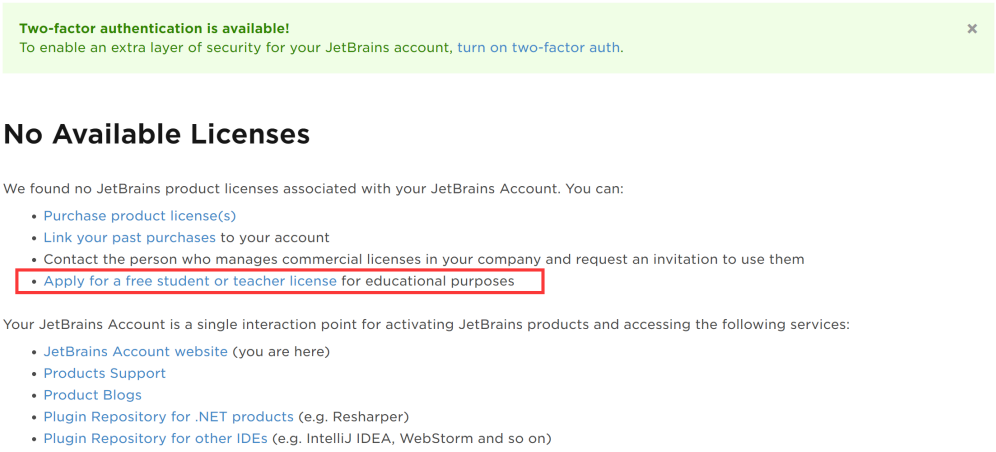
注: 不同的Linux发行版会选用不同的包管理工具, 例如CentOS为yum, Fedora为dnf。这里我们默认大部分使用Linux的同学使用Ubuntu。 如果你使用其它Linux发行版且遇到了安装问题，请联系助教。

1.2 PyCharm安装

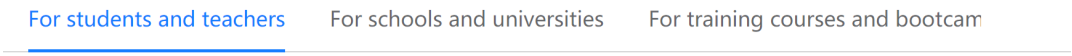
既然我们已经安装好了Python的运行环境，那么大家很自然地会问：在哪里编写和运行Python代码呢？在这里我们为大家介绍PyCharm这个软件。PyCharm是Python的一种集成开发环境（Integrated Development Environment, 简称IDE）。顾名思义，集成开发环境就是一个把各种功能集成到一起给大家使用的软件。这些功能包括但不限于编写代码和运行代码。它不仅可以帮助熟练的程序猿提高工作效率，也可以帮助新手来适应开发的流程。

接下来，让我们看看安装PyCharm的方法：

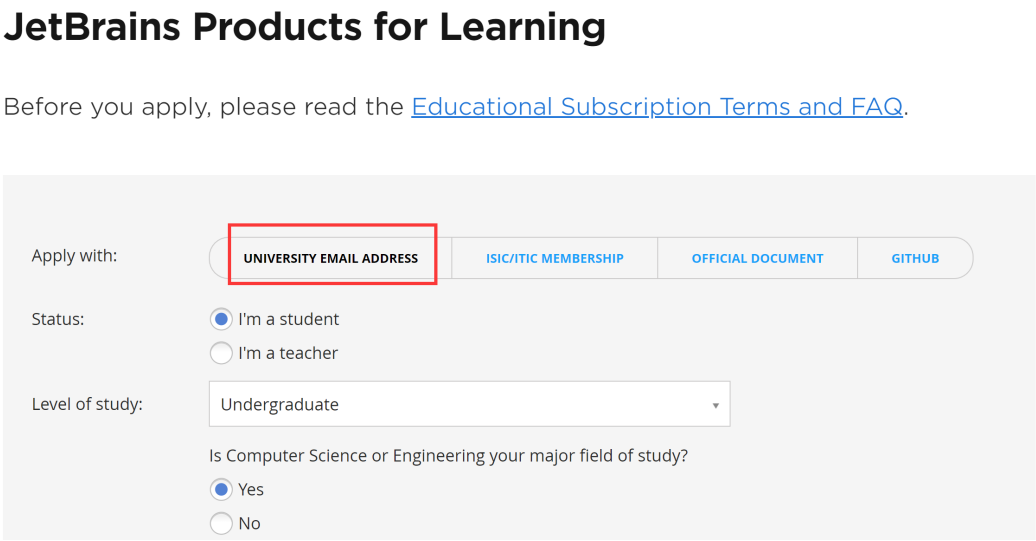
1. 访问官网下载界面，根据自己电脑的操作系统选择对应的下载链接，其中有Community版本和Professional版本这两种选择（二选一即可）：
    - Community版本：完全免费且开源，面向纯Python程序的开发，对于本课程实验已足够适用，同学们可选择这一版本进行下载。
    - Professional版本：有30天免费试用期，过期后需付费使用，但是近年来JetBrains官方（即开发PyCharm的公司）为广大师生提供了免费的专业版证书，大学生可以免费使用。流程如下：首先，在JetBrains官方网站注册账号，注册完成后在当前界面点击“Apply for a free student or teacher license”。



接着点击"For students and teachers"这一栏，并通过点击"Apply Now"进入申请界面。



在申请界面，填写申请表，申请依据选择”UNIVERSITY EMAIL ADDRESS"，并在下面的邮箱中填写学校邮箱，在完成邮箱确认后，账号即可收到证书，在PyCharm Professional软件登录界面用此账号登录，即可在证书的使用时限里免费使用PyCharm Professional。



2. 完成下载后，点击安装，以Windows系统下的PyCharm Community版本为例（其他操作系统和软件版本的安装流程基本类似）。安装时仅需注意安装路径对应的磁盘是否足够容纳PyCharm Community（大小约为800MB），其余按照默认配置点击“下一步”即可，等待安装完成。

1.3 没有PyCharm的时候大家用什么？（可选）

刚才我们说过，PyCharm是一种集成的Python开发环境——它为编辑代码、运行代码等功能提供了一揽子的解决方案。但好奇的同学可能会想，如果没有这样提供一揽子解决方案的软件，“原始”的程序猿是怎么写代码、运行代码的呢？

本质上，代码就是一堆文本。那我们平时怎么编辑文本？当然是Windows上的记事本！（别回答word！）所以，理论上我们使用Windows自带的记事本就可以写代码了。你可以用记事本打开`code`目录下的`hello.py`试试。只要看到以下文本就说明没问题：

```
2020
print(2020)
```

我们把像Windows里这样的记事本软件称作编辑器（Editor）。“原始”的程序猿们就是使用编辑器来编写代码的。而“现代”的IDE比如PyCharm就内置了一个足够好用的编辑器来给大家编辑代码。

> 千万别用word打开代码！word不是编辑器！word是一个所见即所得的文字处理和排版工具！

> “原始”不代表真的原始。适合自己需求的才是最好的。

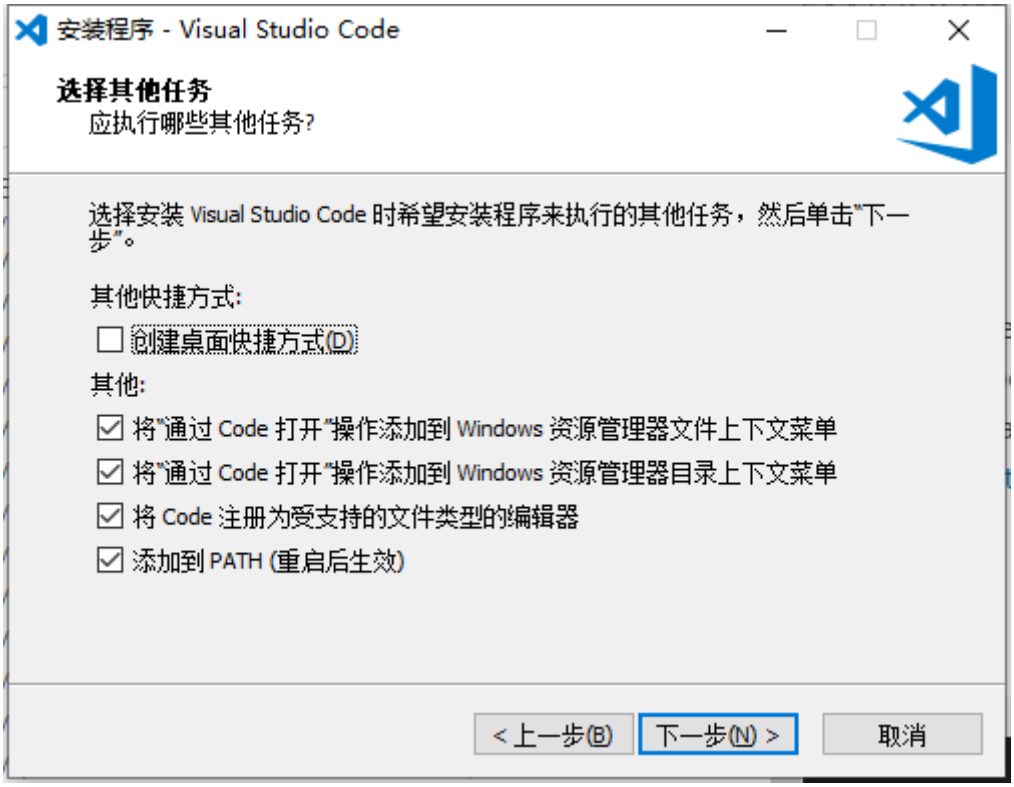
虽说现在我们知道了使用Windows自带的编辑器也可以写代码，但现实中真没有什么人这样干。因为这个编辑器不是为我们写代码而设计的。程序猿们开发了很多专门用来写代码的编辑器，VSCode就是目前最为流行的编辑器之一！接下来我们以Windows平台为例，教大家下载安装VSCode。对于macOS和Linux平台上VSCode的安装，请有需要的同学自行使用搜索引擎查找安装方法。

1.3.1 下载安装VScode

大家首先打开VSCode官网，然后点击该页面上的`Download For Windows`按钮下载VSCode的安装包。点击下载后，网页可能会跳转到一个写着“Getting Started”的页面。这是VSCode的**手册**。

> 一般来说，软件开发者会维护一个手册来教大家如何使用这个软件。因此，未来大家在没有助教引导的时候，遇上新的软件，就要学会自己阅读官方手册。手册多半是英文的，因此学好英语很重要。

安装包下载好后，即可双击开始安装。基本上大家只要保留默认选项，然后不断选择下一步即可。但是，大家需要注意，当出现下图中的界面时，要按照图中所示把“其它”一栏下的复选框都勾选上（自己按需求勾选“创建桌面快捷方式”）。这一步如果不按图中的方式把选项都勾选上，问题倒也不大，只是会让大家的VSCode用得不那么方便。接下来，大家应该能自己操作安装好VSCode了。



> 大家会在程序设计课会用到一个C/C++的IDE，叫做Visual Studio（简称VS）。它和这里的VSCode有什么关系呢？——它们都是微软的。但VS是IDE，VSCode是编辑器。

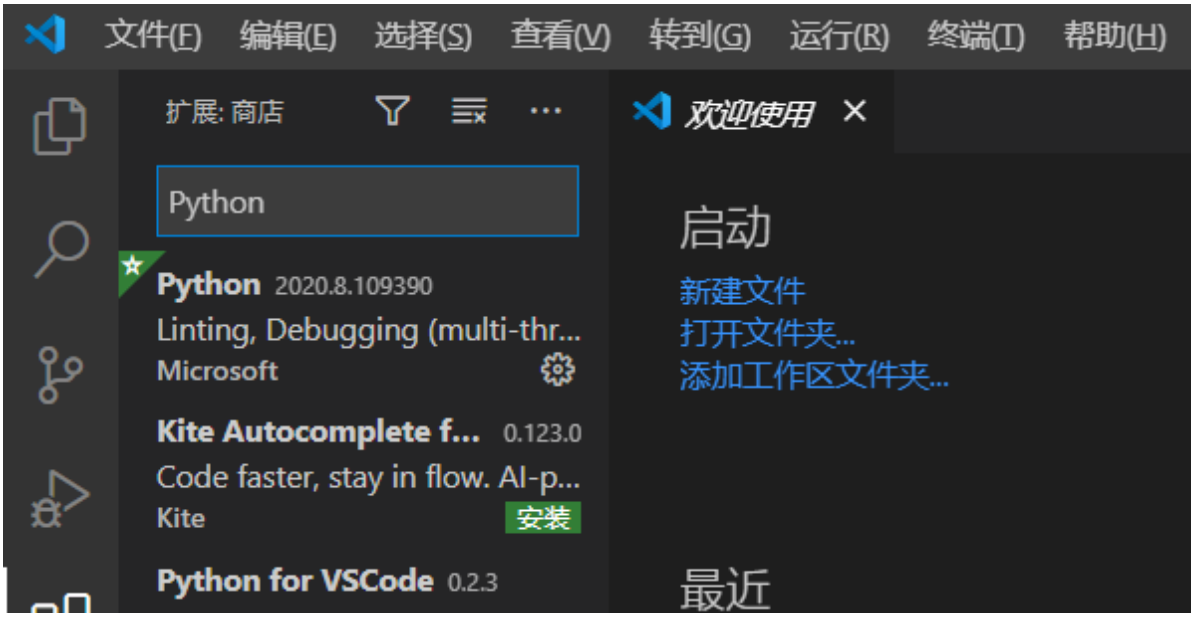
1.3.2 插件安装

目前，我们刚安装好的VSCode只是一个还算不错的编辑器而已。它能流行的真正原因，在于官方以及各路程序猿们合伙给它开发的插件上。接下来，我们来给我们的VSCode安装一下插件，使得它能接近PyCharm的编辑体验。

首先，我们打开VSCode，然后点击下图中红框标出的按钮，进入插件商店。

大家刚打开的VSCode应该是英文界面，而不是中文的。所以我们先来安装一下中文环境。在下图所示的插件商店搜索栏中搜索“Chinese”，然后能看到由微软开发的“中文(简体)"插件。接着，请点击右下角的“安装”按钮进行安装。安装完成后VSCode可能会提醒你重新打开它。

接下来，用同样的方法，我们在插件商店的搜索栏中搜索“Python”，然后安装下图中第一个还是由微软开发的Python插件。图中是已经安装好的样子。



以上就是我们实验中所需要的最好用的两个插件。大家可以在未来根据自己的需求不断添加新的插件打造你的“最强编辑器”。

1.3.3 怎么运行代码?

对了，善于思考的同学可能会问，“现在我们知道没有IDE的时候程序猿们是怎么编辑代码了，但他们又是怎么运行代码的呢？”。先给个答案：程序猿们在“终端（Terminal）”上使用“命令（Command）”来运行代码。什么是终端？什么又是命令？想象一下我们在开头提过的影视作品里黑客们的例子。有没有一些感觉呢？马上我们就会教给大家这些东西。

1.4 小结

到现在，同学们应该已经完成了以下内容：

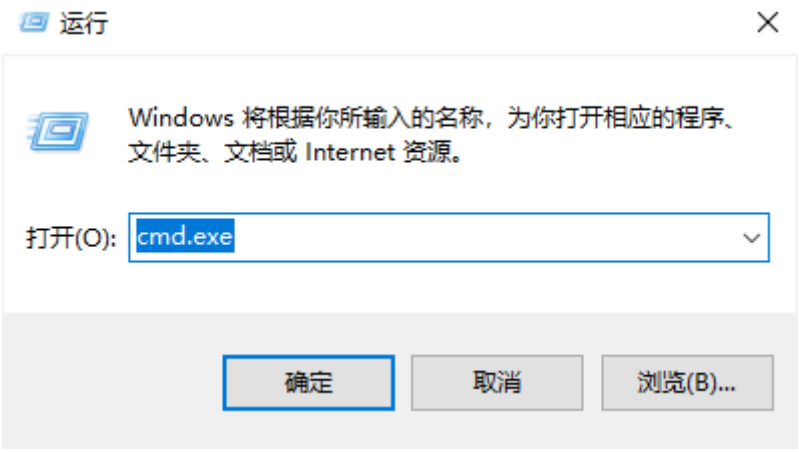
- 安装好了Python环境
- 安装好了PyCharm或者VSCode
- （可选）知道了“原始”的程序猿使用类似记事本、VSCode的编辑器来编写程序，并使用终端和命令来运行程序。从而知道了IDE（比如PyCharm，以及程序设计课会用到的VS）无非是把这些功能整合在一起的软件。它并不神秘。

2. Python基础

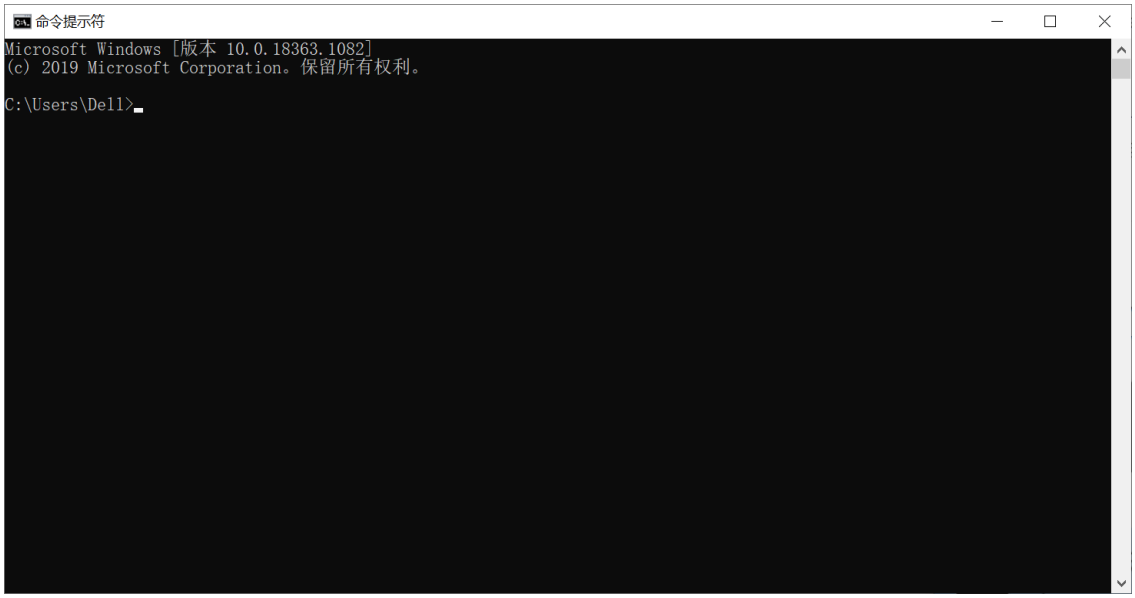
现在环境都已经搭建好，我们可以开始编写和运行Python程序了！首先，我们就来说说前面所提到过的“终端”这一工具，同学们将在日后的课程实验中经常使用它。终端就类似于大家在电影、游戏里经常看见的黑底白字、敲一会代码就一行行文字疯狂往下刷的黑客玩具。我们可以在终端上输入“命令”来让计算机执行程序（比如Python程序）。让我们来看看如何在自己的电脑里打开终端吧。

2.1 打开终端

- **Windows系统**：考虑到本课程的实验仅需最简单的终端命令，我们以Windows系统内置的“命令提示符”为例。首先敲击快捷键`Win + R`，然后会弹出下图所示的窗口。



接着，在输入栏中输入`cmd.exe`，按回车键执行。若见到如下的终端界面，便说明成功打开了Windows的终端。

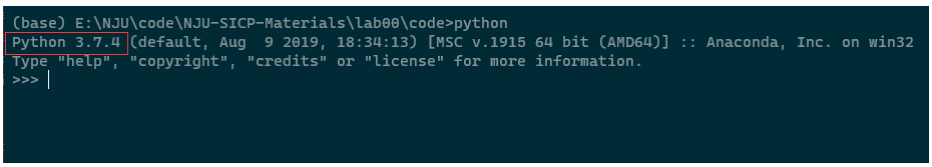


- **macOS系统**：在Apple官网能找到《终端使用手册》在访达（Finder）的"应用程序/实用工具"目录中寻找"终端"（或"Terminal"），打开即可。也可以点击程序坞的"启动台"图标，在搜索栏中键入"终端"，然后点击"终端"。
- **Linux系统**：在桌面或文件夹界面右击，在下拉框中选择"Open in terminal"，就能打开终端。也可以利用快捷键`Ctrl + Alt + T`打开。

2.2 与Python进行交互

打开了终端后，我们就可以开始输入指令做一些事情了。但现在就开始编写Python程序可能为时尚早。不如我们先通过Python的"交互模式"来学习Python程序的基本要素吧。

在打开的终端中，我们可以直接用键盘输入`python`(或者`python3`)这个指令，然后潇洒地按一下回车键进入Python的交互模式界面。效果如下图所示：



> 在Windows上，请大家优先使用`python`指令。如果没有类似上图所显示的效果，再尝试用`python3`指令试试。如果这样还不行，那么请询问助教。在macOS和Linux上，大家可以优先使用`python3`这个指令。在以后的演示中，助教可能会因为操作系统不同，分别采用不同的指令做演示。大家需要根据自己的操作系统作调整。

大家很可能看不懂上面图片中显示的内容。不过不用慌，我们带大家分析一下图中的每串字符。首先是第一行

```
(base) E:\NJU\code\NJU-SICP-Materials\lab00\code>python
```

这串字符可以以“>”为界分成两部分。左侧的“`(base) E:\.....\code>`”被叫做“提示符”（Prompt），它表示你可以在后面输入指令了。所以紧跟在提示符后面的就是我们刚才输入的`python`指令。

提示符的格式对于每个人来说都是可以自己定制的。这里，助教的提示符分为三部分

- 第一部分是“`(base)`”，大家不用了解它的含义。只要知道这是助教自己凭喜好添加的东西。
- 第二部分“`E:\....\code`”是终端的“当前工作目录”。这个概念我们会在后面讲解。
- 第三部分是“`>`”，一个比较形象的符号来提醒你在它后面可以输入指令了。Linux上默认会用“`$`”这个符号。

正常来说，大家打开终端看到的提示符是由第二和第三部分组成的，即类似于第2.1节图中助教打开的终端里的提示符：“`C:\User\Dell>`”。

当我们输入完`python`指令，并按下回车键后，Python程序的交互模式正式开始运行，并在图中的第二和第三显示印了一些信息。比如，图中被红框圈出的信息代表了你安装的Python的具体版本。大家终端上显示的信息不必和图中的完全一致，但红框位置上的版本号必须是`3.x.y`的形式。

最后，Python的交互模式在第四行显示了“`>>>`”这个符号。这个符号也是提示符，只不过没先前终端显示的提示符那么长。它代表了我们可以输入一些“东西”跟Python交互了。

在以后，我们会避免使用图片来显示指令的运行结果，而更喜欢使用文本的方式来显示。比如，在刚才的输入`python`指令进入Python交互模式的例子中，我们会像下面这样来显示大家应该看到的结果：

```
$ python
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)] ::
Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

可以看到，为了减少视觉负担，我们还使用了“`$`”这一简短的符号来代替原本图片中一长串的提示符。未来大家会在网络上看到更多这样的表达。

2.3 表达式与语句

接下来，我们在Python交互模式中来学习表达式和语句的概念。大家在高中阶段一定见过`8 * 8`，`a + b`，`7 - (x + y)`之类的数学表达式，而程序表达式也是类似的，它们由数字（亦称为常数）、变量(如`a`，`x`等，用字母表示)、运算符（如加法、乘法符号等）等组成，一些表达式还包含了函数，这在课上会讲到（不妨试着用数学里的“函数”概念去理解它）。我们知道，如果表达式里面每个部分的值都是已知的（例如，`a + b`中`a`的值为4、`b`的值为5），那么整个表达式就可以被**求值**。让我们来看看Python里面的表达式和求值是什么样的。

2.3.1 基本表达式

基本表达式只需要一步便可以求得它的值，例如数和布尔值：

```
>>> 3
3
>>> 12.5
12.5
>>> True
True
```

同学们可以像上面这样在交互模式下输入表达式，并按回车键查看结果。

变量名同样也是基本表达式，求值的结果为在当前程序的状态下绑定到的值（更多内容见赋值语句）。

2.3.2 布尔值与布尔表达式

布尔值只有两种值，`True`和`False`，表示逻辑上的“真”与“假”。例如，我们知道数字`1`是不等于`2`的，所以若在Python交互模式中输入`1 == 2`（这个表达式意思是“1等于2”，在Python中一般用`==`表示“相等”），则会得到它的值为`False`；相反，如果输入`1 != 2`（表示”1不等于2“，在Python中一般用`!=`表示”不相等“），则会得到值`True`。这种能计算出布尔值的表达式我们称之为布尔表达式。

```
>>> 1 == 2
False
>>> 1 != 2
True
```

2.3.3 算术表达式

可以通过对数字和数学运算符的组合产生复杂的算术表达式。Python中常见的数学运算有`+`(加)、`-`(减)、`*`(乘)、`**`(乘方)和以下三种：

- 浮点除法 (/)：计算第一个数除以第二个数的结果，得到一个浮点数（即使能够整除）。浮点数是一种带小数点的数。之所以叫它浮点数，是因为它和计算机底层的表示有关。等大家上《计算机系统基础》这门课的时候就会理解了。
- 下取整除法 (//)：计算第一个数除以第二个数的下取整后的结果，得到一个整数。
- 取模 (%)：计算第一个数除以第二个数的余数(>0), 得到一个整数。

像大家平时学过的数学运算一样，算数操作也有优先级，比如乘方运算的优先级大于乘除运算，乘除运算的优先级大于加减运算。同样地，你也可以通过“加括号”来改变求值优先级。注意，优先级不需要记忆。当你对运算优先级拿不准的时候，就多加括号！

```
>>> 7 / 4
1.75
>>> (2 + 6) / 4
2.0
>>> 2 + 6 / 4
3.5
>>> 7 // 4
1
>>> 7 % 4
3
```

2.3.4 赋值语句

一个赋值语句由一个变量名和一个表达式组成。它会在当前帧（帧的概念后续课程会讲）下把变量名与该表达式的值绑定。程序的状态也因此而改变。

```
>>> a = (100 + 50) // 2
```

注意不能对语句进行求值。

你可以输入一个之前定义过的变量名，例如`a`，Python解释器会输出其绑定到的值，如果你还记得之前的内容，a也是一个基本表达式：

```
>>> a
75
```

注意这里`a`绑定到`75`，而非`(100 + 50) // 2` —— 变量名绑定到值而非表达式上。

2.3.5 print

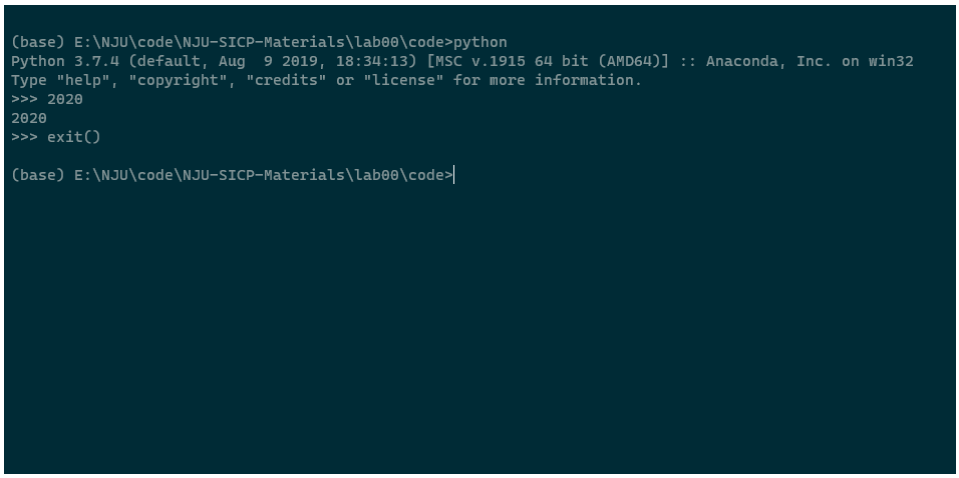
`print()`是Python3的内置函数，它可以将一个表达式的值输出到终端上（如果还不清楚函数的概念，就单纯记忆一下它的用法吧）。`print(2020)`也是一条语句：

```
>>> 2020
2020
>>> print(2020)
2020
```

注意，上面在交互模式下输入`2020`和`print(2020)`显示的结果看上去相同，但前者是交互模式下自动输出的每一行表达式的值，后者是`print()`在终端中的输出。在非交互模式下（即执行Python代码时，后面会讲），如果你想知道结果，请用`print()`。

2.4 退出交互

当你完成与Python的交互后，可以输入`exit()`并回车或按下`Ctrl+Z` (Windows/Linux) / `Command+Z` (macOS)退出交互窗口。



可以看到，你的终端不再显示Python的提示符，而是显示了自己的提示符。这说明你又可以向终端（而不是向Python交互界面）输入新的指令了。

2.5 执行Python文件

除了交互模式，大家也可以把之前学过的表达式和语句都写入到文件中，然后让Python一行一行地从文件中读取并执行（就像交互模式一般）。其实，这个过程就是我们提到很多次的编写代码与运行代码：

- 用编辑器把原先输入给Python解释器的表达式和语句写到文件中就是“编写代码”的过程；
- 让Python直接读取文件像交互模式一样不断执行我们事先写入文件的东西就是“运行代码”过程。

关于编写代码，我们已经事先准备了一个编写好的文件`code/hello.py`（约定俗成，大家以`.py`的后缀来表示这是一个Python代码文件）。这个文件中有下面内容：

```
2020
print(2020)
```

你也可以利用任何任何文本编辑器修改`hello.py`，尝试前面提到的其它表达式和语句，或者试一试其它你感兴趣的内容。后面我们会教大家如何用PyCharm修改它。

接下来，你需要使用`python 文件名`（或`python3 文件名`）指令来执行这个Python代码文件。在这里，你需要输入：

```
$ python hello.py
```

然后可以得到结果

```
2020
```

但多半你会发现出了错误！请往下看。但另外，你可能还会疑惑为什么只显示了一个2020。现在你需要回顾一下前面2.3.5小节的内容。你会发现，原来`2020`这种表达式仅会在交互模式下显示求值结果。这里不是交互模式，所以表达式`2020`没有显示任何结果。屏幕上的2020是由print函数“打印”出来的（难怪它叫print）！

2.6 文件查找与当前工作目录的切换

在执行`python hello.py`时，大家有可能会看到终端上显示出错误：

```
$ python hello.py
python: can't open file 'hello.py': [Errno 2] No such file or directory。
```

这和python的文件查找机制有关。

大家还记得先前提到的“当前工作目录”的概念吗。不记得了可以跳回2.2节去看一眼：终端会在提示符里面显示当前的工作目录。而当你键入`python hello.py`时，Python会在当前的工作目录里去找`hello.py`并执行。而如果你当前工作目录下没有`hello.py`文件时，Python就会报错了。

举个错误的例子：假设你当前的工作目录是`C:\SICP\lab00`，而`hello.py`在`C:\SICP\lab00\code`目录下，那么你执行

```
C:\SICP\lab00>python hello.py
```

就会出现问题。但如果你当前的工作目录是`C:\SICP\lab00\code`，那么你执行

```
C:\SICP\lab00\code>python hello.py
```

就不会有任何问题。

那么接下来你就会问，该如何改变当前的工作目录呢？你可以通过`cd`这个命令切换终端的当前工作目录！比如，如果你当前的工作目录为`C:\SICP\lab00`，那你可以像这改变当前工作目录：

```
C:\SICP\lab00>cd code
C:\SICP\lab00\code>
```

你会发现，提示符中的当前工作目录改变了。这个过程就很像你在资源管理器的lab00目录下双击了code目录，然后就进入了lab00\code目录一样。

> `cd`的全称是change directory。

接下来你可能会想知道如何回到上一级目录或者切换磁盘。下面是一些演示：

- “`..`”的含义为上一级目录。“`.`”的含义为当前目录。可以使用`cd a\b\c`这样的形式一口气进入多层目录。“`#`”及后面的文字表示注释说明。

```
C:\SICP\lab00\code>cd ..      # 进入上层目录
C:\SICP\lab00>cd ..           # 再次进入上层目录
C:\SICP>cd lab00\code         # 进入lab00目录下的code目录
C:\SICP\lab00\code>cd ..\..   # 进入上层目录的上层目录
C:\SICP>cd .\lab00            # 进入当前目录的lab00目录
C:\SICP\lab00>cd ..\lab00     # 进入上层目录的lab00目录 (即还是当前目录)
C:\SICP\lab00>
```

- 在Windows如果当前目录的盘符与目标盘符不同，你首先需要通过"{盘符}:"命令切换盘符。

```
C:\SICP\lab00>cd E:\NJU    # 无效的切换
C:\SICP\lab00>E:           # 有效的切换
E:\NJU>
```

- Windows的路径不区分大小写，但是Linux和MacOS均区分。
- Windows上路径的目录之间使用“`\`”分割，而Linux和MacOS则使用“`/`”分割。

2.6.1 当前工作目录下的有哪些目录与文件？（可选）

你可以通过`dir`指令（在Windows上）或`ls`指令（在Linux/MacOS上）显示当前目录下的文件。例如，在Windows下：

```
C:\SICP\lab00\code>dir
 驱动器 C 中的卷是 Windows
 卷的序列号是 4C0F-CDB0

 C:\SICP\lab00\code 的目录

2020/09/21  10:37    <DIR>          .
2020/09/21  10:37    <DIR>          ..
2020/09/21  15:03                19 hello.py
2020/09/19  17:48               480 lab00.py
               2 个文件            499 字节
               3 个目录 100,005,732,352 可用字节

C:\SICP\lab00\code>
```

在Linux下：

```
~\SICP\lab00\code$ ls
hello.py lab00.py
~\SICP\lab00\code$
```

2.7 如何用PyCharm做以上这些事情?

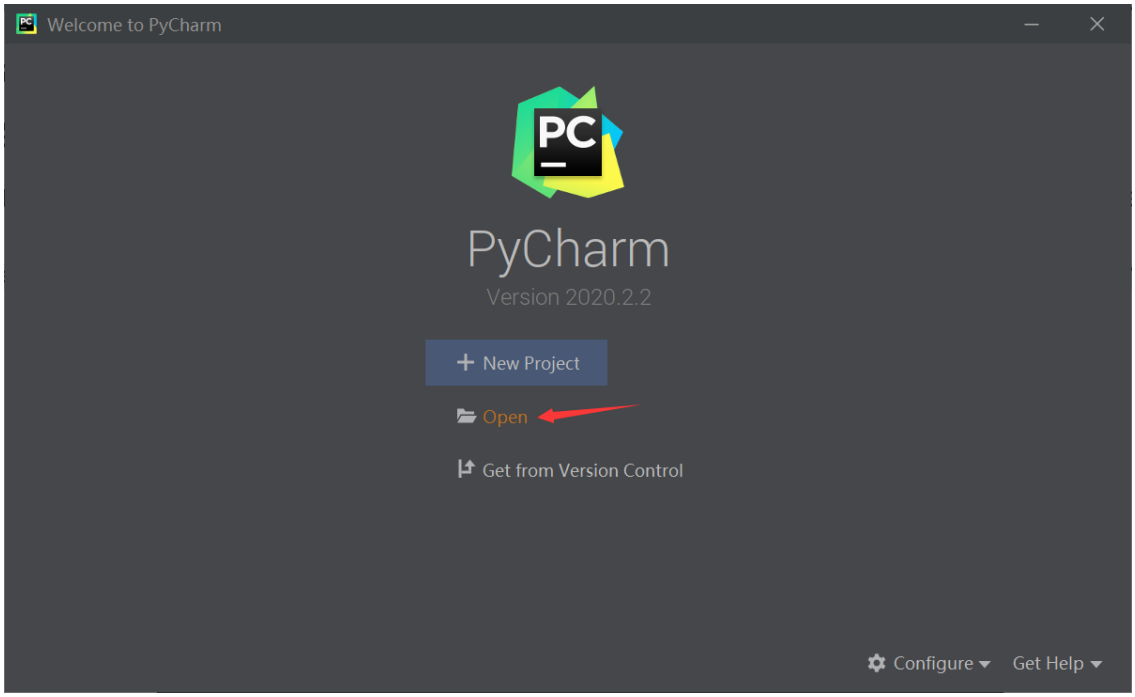
在理解了基本的Python交互与文件执行方法之后，我们把视线移回辛辛苦苦安装好的PyCharm。前面大家输入了那么多指令，不知道是觉得很酷炫，还是觉得很麻烦？但至少，PyCharm会让大家更轻松点。

在编写程序方面，相比于最普通的记事本，PyCharm最明显的好处是——时常可以提醒你代码哪儿可能写错了，而且可以用不同的高亮颜色区分程序的各个不同部分，减轻你的视觉负担，再加上便捷的代码调试、代码跳转等功能，让你的实验时间骤减！当你通过逐步探索而更加熟悉PyCharm时，它会更显著地提高你的编程效率，成为你的~~好伴侣~~。

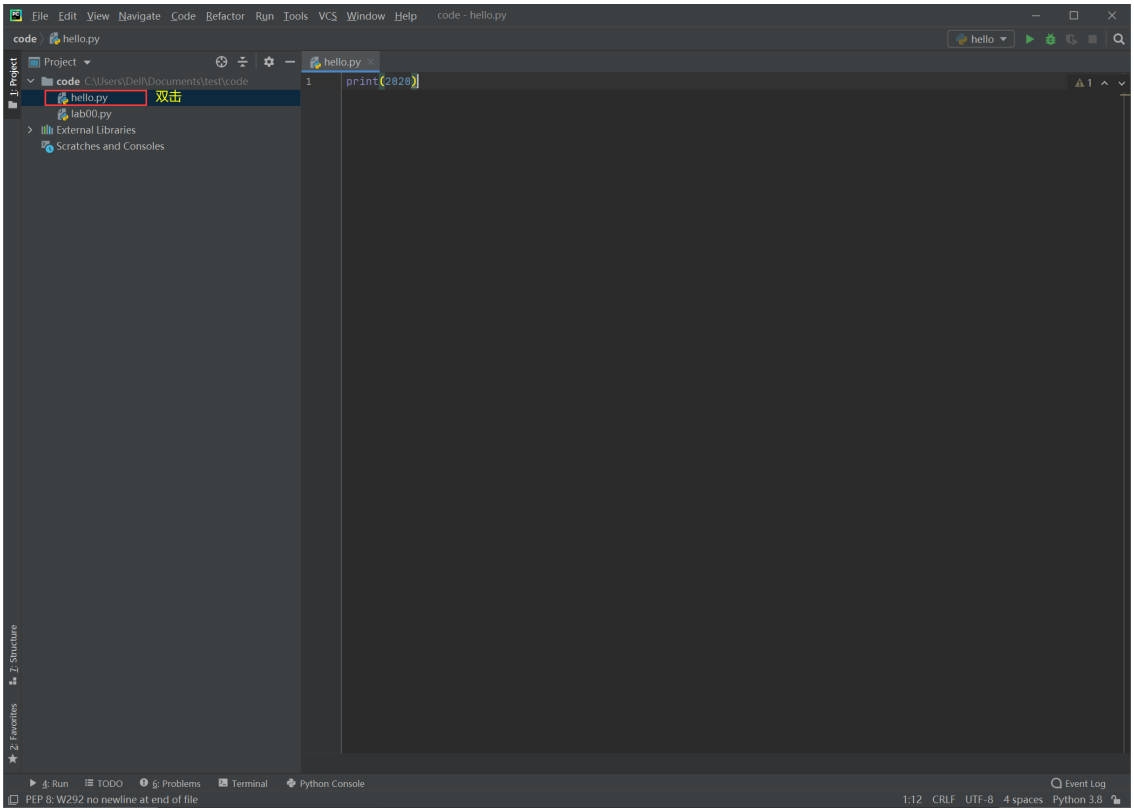
每次实验发布时，都会附带一个包含实验框架代码的文件夹。我们怎样在PyCharm里打开它，以便于编写和运行里面的程序呢?

2.7.1 PyCharm导入和运行已有的Python文件

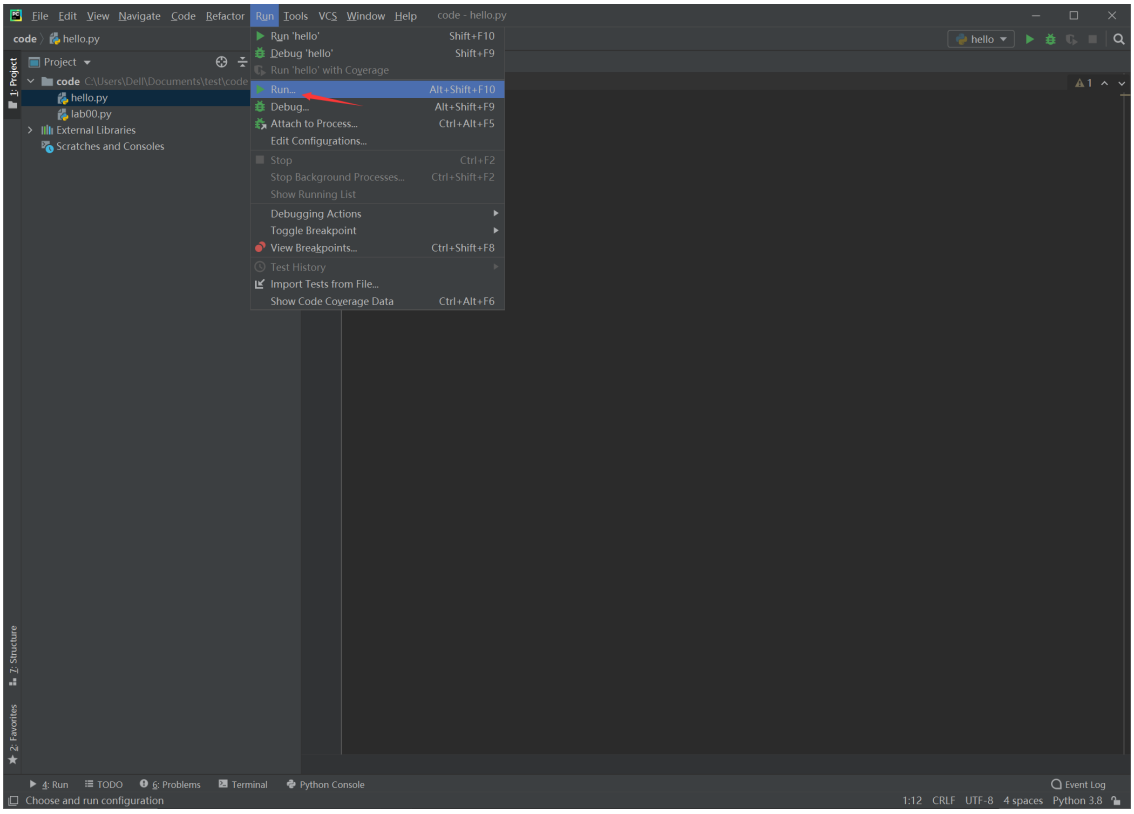
1. 首先我们双击已安装的PyCharm Community软件，启动后有界面主题配置，选择自己喜欢的主题即可，无需安装其他插件。随后大家会看到如下所示的欢迎界面，点击“Open”，寻找实验代码所在的文件夹（例如，本次实验我们要完成的函数都在文件`lab00.py`中，它所在的文件夹为`code`，那么我们就点击`code`文件夹，然后点击右下角的“OK”），PyCharm会为这个文件夹自动创建一个项目，它包含着所有实验文件。

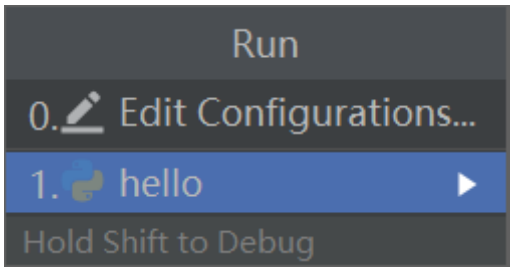


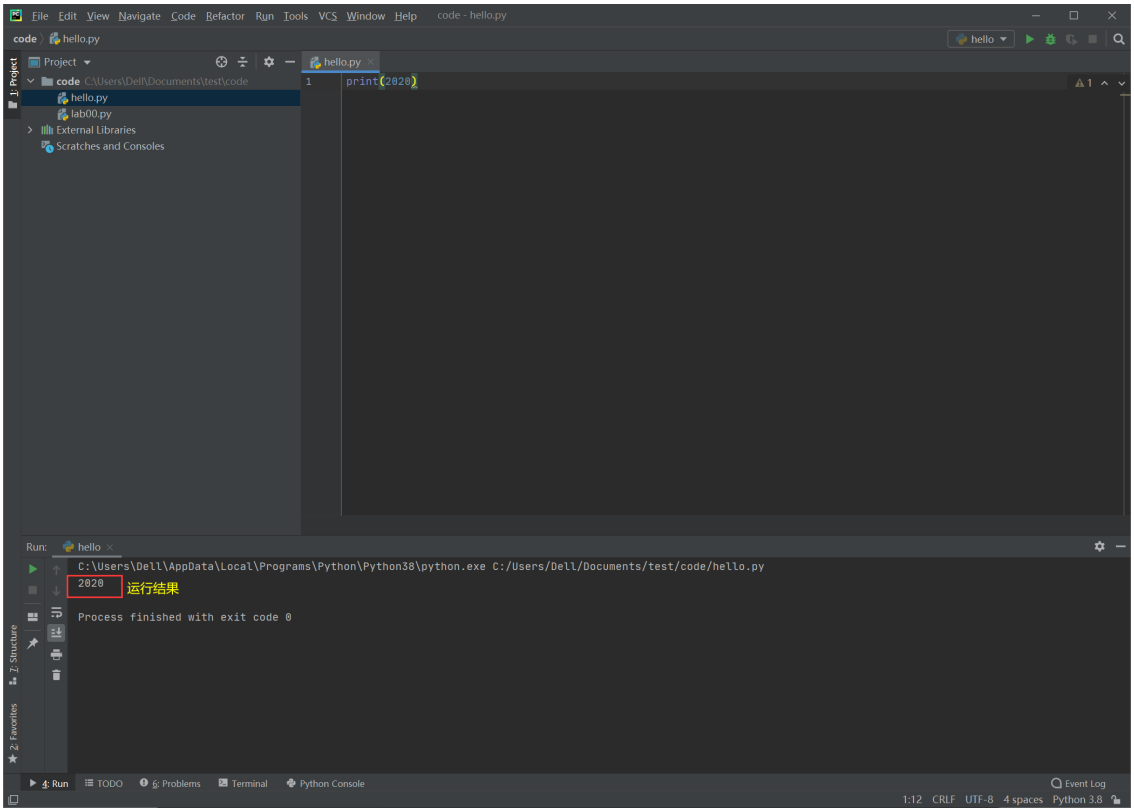
2. 在打开项目后，双击左边的`hello.py`文件，此时大家就可以看到`hello.py`这个文件的内容已成功地显示在PyCharm主界面中（图中`hello.py`的内容仅为演示使用，不代表最终版发给大家的`hello.py`的内容）。



3. 下面我们要运行`hello.py`这个文件，看看它的结果能否和终端运行一样。点击上方菜单栏的”Run“，选择下拉框中的”Run“，可以看到一个提示框。单击`hello`（表示运行`hello.py`这个程序），就可以看到输出结果`2020`显示在了下面的运行栏中。

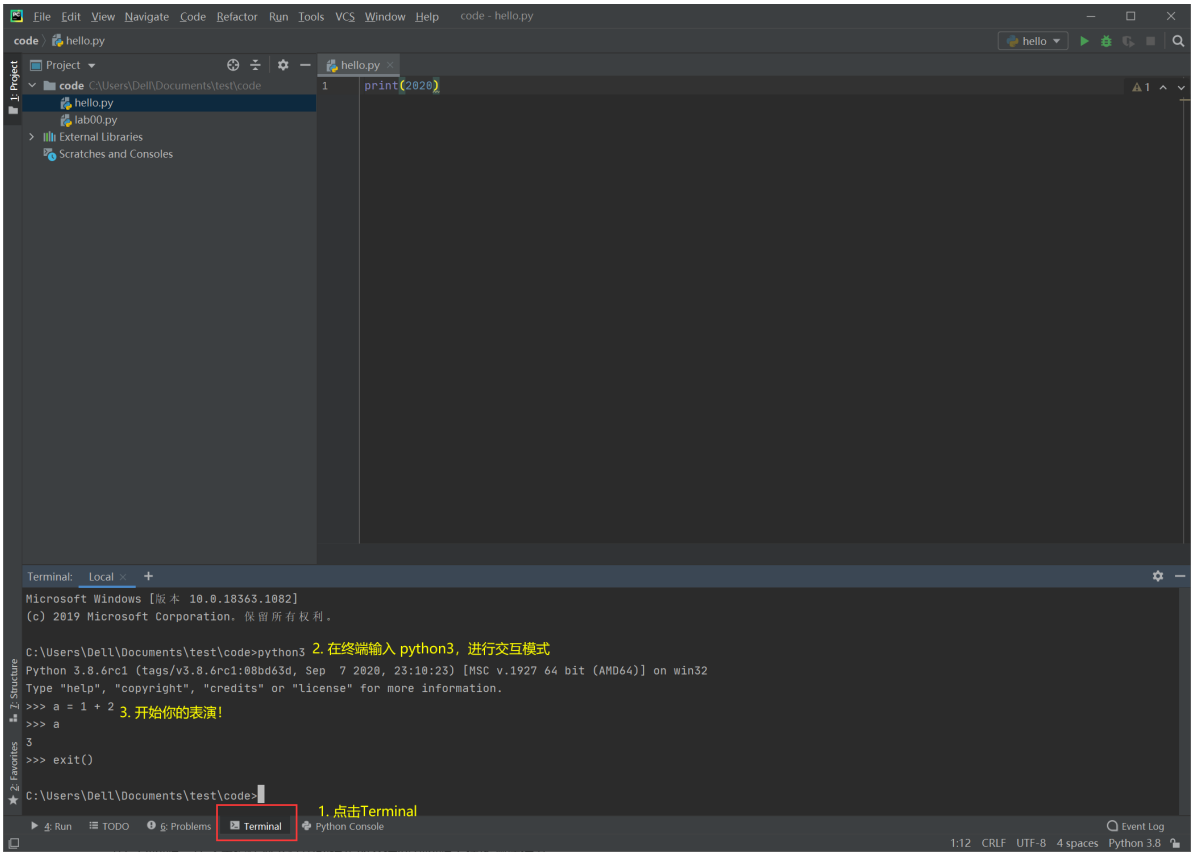






2.7.2 在PyCharm中打开终端

如果想在PyCharm里使用先前介绍的终端，我们可以看见界面下方有个"Terminal"按钮，点击即可在PyCharm界面里打开我们先前讲解过的终端，输入`python`便可进入Python交互模式，像先前讲解过的一样玩转Python！当然，还可以在终端输入命令`python hello.py`来执行`hello.py`程序，这也是刚刚提到的PyCharm运行方式的一种替代方法。



2.8 如何用VSCode做以上这些事情？（可选）

对于之前选择VSCode的同学，我们现在来看看如何用VSCode打开文件、打开目录和打开终端。因为VSCode是个轻量的编辑器，所以你会发现，相比于PyCharm的流程，VSCode会更简单。

2.8.1 打开文件

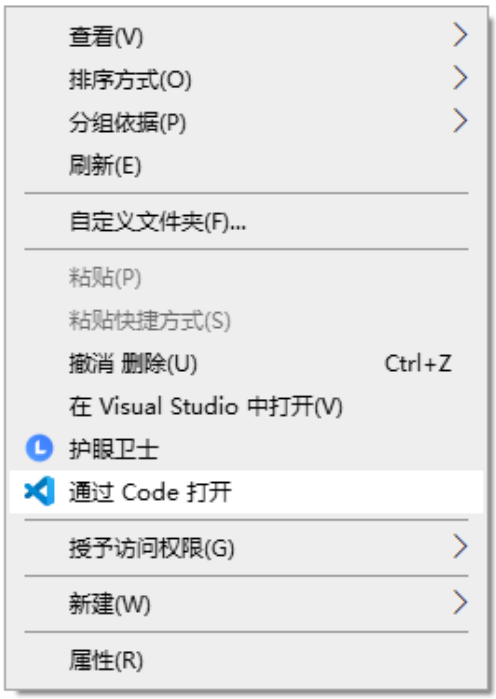
我们可以像标准的编辑器一样（比如记事本）直接右击文件、选择“通过Code打开”来打开文件。如下图所示，我们演示了在实验代码的code目录下用VSCode打开`hello.py`文件。

如果你在安装VSCode的时候没有勾选`将"通过Code打开"操作添加到Windows资源管理器文件上下文菜单`这一选项，就比较麻烦了。请自己探索如何打开文件吧。

2.8.2 打开目录

VSCode有一个“打开目录”的概念。比如，一个项目可能由多个代码文件组成放在一个目录下。我们希望“打开这个目录”，然后自由选择打开目录下的文件。解释起来可能有点难理解，但操作几下就很容易明白了。

如下图所示，依旧在code目录下，我们右击空白处，选择“通过Code打开”来打开code目录。打开以后，你会发现VSCode左侧多了个文件列表。你可以点击文件来打开你想要的文件。可以说，“打开目录”这个概念很像是PyCharm中的打开项目的概念。



如果你在安装VSCode的时候没有勾选`将"通过code打开"操作添加到windows资源管理器目录上下文菜单`这一选项，就比较麻烦了。请自己探索如何打开目录吧。

2.8.3 打开终端

你可以通过快捷键`Ctrl+`` `来打开终端。如果你先前通过"打开目录"来打开了VSCode，那么终端的工作目录即是你刚才所打开的目录。现在，你可以在终端上进行之前我们教过的任何操作了，比如与Python交互、运行Python代码。

2.9 小结

在本小节，同学们应该已经完成了以下内容：

- 掌握了终端的基本概念：
  - 如何打开终端；
  - 使用`cd`指令来切换终端的当前工作目录。
- 掌握了在终端下使用`python`指令来：
  - 进入Python的交互模式，执行简单的表达式和语句，以及退出交互；
  - 执行`.py`代码文件。
- 掌握了使用PyCharm来：
  - 打Python项目和文件
  - 点击鼠标来执行`.py`代码文件
  - 打开终端
- （可选）掌握了使用VSCode来：
  - 打开目录和文件
  - 打开终端

3. 如何完成实验?

现在，大家应该熟悉了如何用Python编写代码以及运行代码的过程。那接下来，我们就要讲解大家可能最不愿意看到的东西了——如何完成实验？

在我们的课程实验中，大家需要补充作业文件中的函数定义来完成一些题目（暂时不知道函数是什么也没关系）。我们实验的作业文件一般会放在code目录下，随实验讲义一起发布给大家。不要担心。我们首先带大家一起完成一道题目来熟悉这个流程。

3.1 作业文件

本次实验的作业文件是`code/lab00.py`。大家可以使用编辑器或者PyCharm打开它。打开`lab00.py`后，我们首先能看到下面这段代码：

```
def twenty_twenty():
    """Come up with the most creative expression that evaluates to 2020,
    using only numbers and the +, *, and - operators.

    >>> twenty_twenty()
    2020
    """
    return _______
```

上面代码里包含了我们将要完成的第一个题目——2020（这个题目就叫做2020）。看到上面的代码时，你可能是一脸懵逼的。不过不要慌，我们来一步一步理解。

首先，这段代码使用`def twenty_twenty()`定义了一个函数`twenty_twenty`。如果在实验课前老师还没讲如何定义函数，大家也不用着急。不知道这个知识点也能完成本次实验。不过，在下一次课上你应该就会学到如何定义函数了。

接下来，我们会看到一段被三引号`"""`包围的文本。我们把这段文本叫做**docstring**（全称document string，即文档字符串）。它一般被用来描述一个函数应该做什么。在我们`twenty_twenty`的例子中，这段docstring告诉我们：“写出一个富有创造力的表达式，使得计算它的值能得到2020”（当然，在本次实验中没那么有创造力也无妨）。此外，它还要求我们只可以使用数字和加减乘除。

紧接着，我们能看到有一行docstring是以“`>>>`”开头的。从这行开始往下的docstring被叫做**doctest**(举一反三一下，它的全称为什么？——document test)。回想一下大家在前面使用的Python交互模式，“`>>>`”被我们称为提示符——它告诉我们可以在它后面输入表达式或者语句。是不是当你在“`>>>`”后面输入一个表达式后，Python解释器就会在下一行打印这个表达式的值？doctest就是通过这样的示例来告诉你一个函数会做什么：“当我们在交互模式下这样输入代码，就会得到这样的结果”。在我们`twenty_twenty`的例子中，doctest告诉我们：“当我们在交互模式下输入`twenty_twenty()`，就会得到`2020`”。

总的来说，docstring和doctest分别使用“描述”和“举例”的手段来告诉我们一个函数的行为。未来在大家完成实验题目的时候，除了阅读实验讲义，也最好读一读它们加深自己的理解。

> 在实验中，你不需要修改任何docstring，除非你想添加自己的doctest。你可能会问“我为什么会想添加自己的doctest？”。继续读下去就知道了。

3.2 写点代码

现在，我们应该理解了作业文件中的第一个题目，即写一个(有创造力的)表达式，使得它的值为2020。

那么我们该把这个表达式写在哪里呢？我们的实验讲义会把这种要求都描述清楚。本次实验其它题目的描述可以在下面的“本次实验内容”一节中找到。作为一个带大家一起做的例子，我们现在先告诉大家第一个题目的要求：**请用你想到的那个表达式替换掉`twenty_twenty`中下划线的部分。**

当然，我们之前说过第一个题目我们会带大家一起做。所以虽然不知道富有创造力的大家想到了什么表达式，但没有创造力的助教们就想了一个这样的答案：

```
def twenty_twenty():
    """Come up with the most creative expression that evaluates to 2020,
    using only numbers and the +, *, and - operators.

    >>> twenty_twenty()
    2020
    """
    return 2019 + 1
```

大家可以修改各自得到的`lab00.py`文件，把下划线替换成自己想到的表达式。

3.3 我实现的对不对?

现在，大家已经完成了第一个题目。这个题目当然是很简单的，但之后随着知识的增多，我们要完成的题目也越发复杂。那么有没有什么方法能检验我们写的代码对不对呢?

回想一下，doctest描述了"当我们在交互模式下这样输入代码，就会得到这样的结果"，那么我们是不是可以打开交互模式，按照doctest的描述输入代码，然后观察结果是不是一致的呢?

答案是肯定的。我们首先打开终端（不管是从PyCharm里打开，还是直接运行cmd.exe），然后使用`cd`指令把当前目录切换到`lab00.py`所在的目录下（回忆一下实验讲义前面教你如何切换当前目录）。然后输入`python -i lab00.py`。这个指令长得和我们在前面学过的`python lab00.py`很像，但多了个`-i`。`-i`是告诉Python，在使用`python lab00.py`执行完`lab00.py`后，进入交互模式。而开发Python的程序员之所以采用`i`这个字母，是因为它是interactive(交互)的首字母，方便记忆。现在让我们执行`python -i lab00.py`。你应该能在终端上看到下面的效果：

```
$ python -i lab00.py
>>>
```

接下来的过程就和交互模式一模一样了。你可以对照着doctest，输入`twenty_twenty()`，然后就会得到

```
$ python -i lab00.py
>>> twenty_twenty()
2020
>>>
```

这和doctest描述的一模一样，所以你现在可以松口气，相信自己的代码是对的了。如果发现不一样，你就需要看看自己哪里出问题了。

等一等！刚才其实有一个概念的陷阱。现在让我们思考一个问题：为什么说，经过刚才这样一个过程，我们就有理由相信我们的代码是对的呢?

实际上，"相信自己的代码是对的"是一个错误的说法！准确的说，大家只能有理由相信"自己写的代码和doctest所预期的相符合"，进而推论，自己的代码**可能**是对的。

其实，像这样一个比较代码的实际执行结果和预期结果的过程就叫做**测试**。这也是"doctest"中test的中文含义。测试不能保证"代码是对的"，只能保证代码在多大程度上与预期相符合。与预期符合得程度越高，我们就更加有信心来相信，我们的代码是对的。

> 实际上，还有一种检查程序正确的方式叫做形式化验证（Formal Verification），通过规则和逻辑（而非测试样本）来判断程序的正确性。这正是冯新宇老师研究的领域之一。在你之后的学习过程中可能还会进一步接触相关内容。

对于助教来说，很难一一审查每个人的代码来验证大家写得正不正确，所以我们会采用测试的方法来给大家评分。像我们发布给同学的doctest一般都写得比较简单，所以那些写得不正确的代码也有可能通过doctest，比如像下面这段代码，本来助教希望大家写`return a + b`，但有个小机灵鬼直接返回3试图蒙混过关：

```
def add(a, b):
    """return the sum of a and b

    >>> add(1, 2)
    3
    """
    return 3
```

但如果我们的测试写得越"详细"，写得不正确的代码就越难蒙混过关。比如像现在：

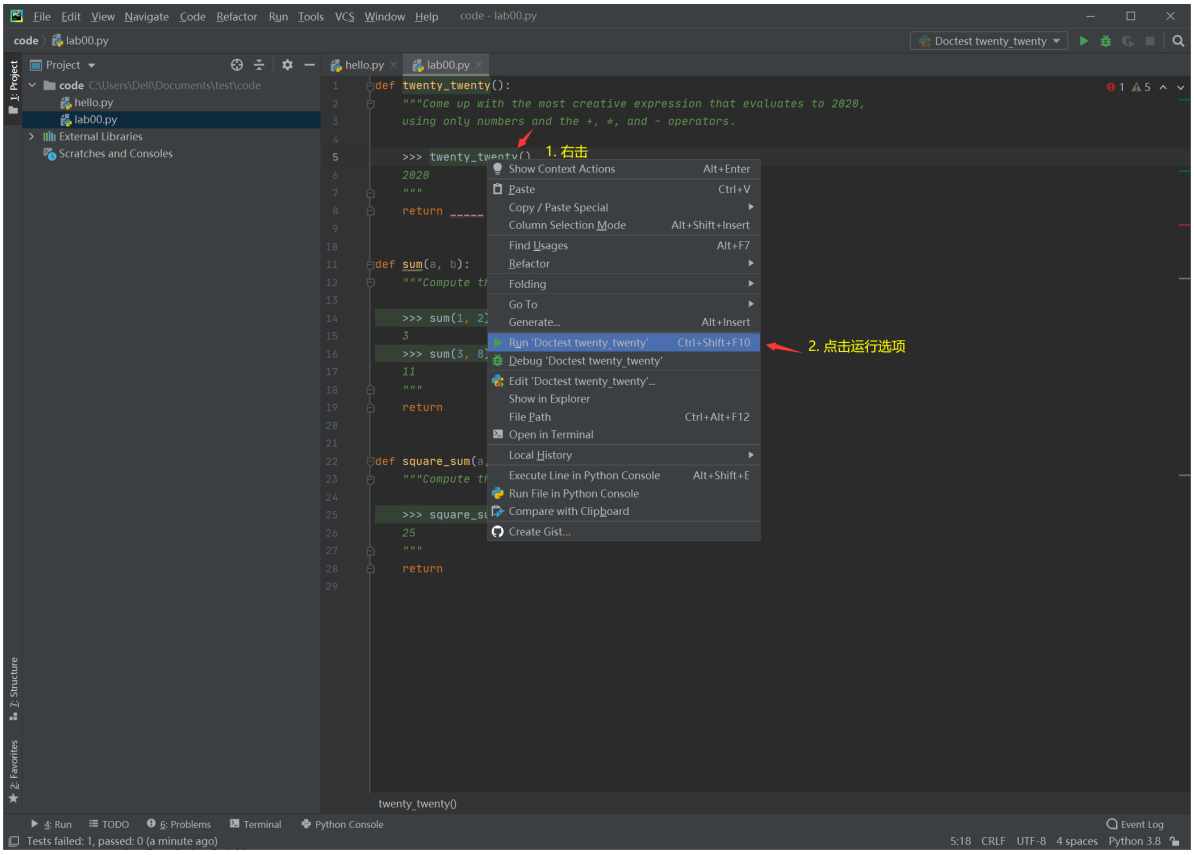
```
def add(a, b):
    """return the sum of a and b

    >>> add(1, 2)
    3
    >>> add(3, 4)
    7
    """
    return 3
```

因此，助教们会设置一个在线测试网站，用更加"详细"的测试来给大家评分，尽可能防止不正确的代码蒙混过关。我们会在下一节中教大家如何在网站上提交作业，让助教测试你的代码。

> 本节关于测试的介绍，如果大家理解得不清楚也没关系。甚至这一节提到的方法不用也没关系。重点是记住下一节要讲的提交作业的方法，并知道如何查看自己写的代码拿了多少分。

随着大家做的题目越来越多、越来越复杂，同学们可能会发现，每当自己代码写错了，就要重新走上面一套doctest的流程，然后嫌手动输入表达式很麻烦。这时候，我们也准备了解决方案。

对于已安装PyCharm且喜欢图形界面交互的同学：在PyCharm界面里右击以`>>>`开头的doctest表达式，如图所示运行doctest，可以直接在下方运行界面查看测试结果。



如果没有通过所有的doctest，则会显示红色惊叹号和错误信息；如果通过了所有的doctest，你会看到代表测试通过的勾形标记以及`Tests passed: xxx`的提示框，像下面一样。祝你在每一次实验都能看到这一场景！

对于喜欢终端操作的同学：首先，和之前一样，你需要使用`cd`指令把当前目录切换到`lab00.py`所在的目录下，接下来输入`python -m doctest lab00.py`。这个指令的内部原理基本上和你之前手动执行的过程一模一样。只不过，它会把`lab00.py`中其它你还没做的题目的doctest也一起执行了。所以实验刚开始你可能会看到很多其它的错误信息。比如，当你写完第一个题目后，你会看到下面的错误信息：

```
$ python -m doctest lab00.py
**********************************************************************
File "C:\...\lab00\code\lab00.py", line 23, in lab00.square_sum
Failed example:
    square_sum(3, 4)
Expected:
    25
Got nothing
**********************************************************************
File "C:\...\lab00\code\lab00.py", line 13, in lab00.sum
Failed example:
    sum(1, 2)
Expected:
    3
Got nothing
```

```
**********************************************************************
File "C:\...\lab00\code\lab00.py", line 15, in lab00.sum
Failed example:
    sum(3, 8)
Expected:
    11
Got nothing
**********************************************************************
2 items had failures:
1 of   1 in lab00.square_sum
2 of   2 in lab00.sum
***Test Failed*** 3 failures.
```

但仔细看下来，这都和大家接下来要做的两个题目（`sum`和`square_sum`）相关。所以暂时不用在意这样的错误信息。当你把所有题目做完后，如果使用`python -m doctest lab00.py`没有显示任何信息，那么就代表你通过了所有的doctest的测试。

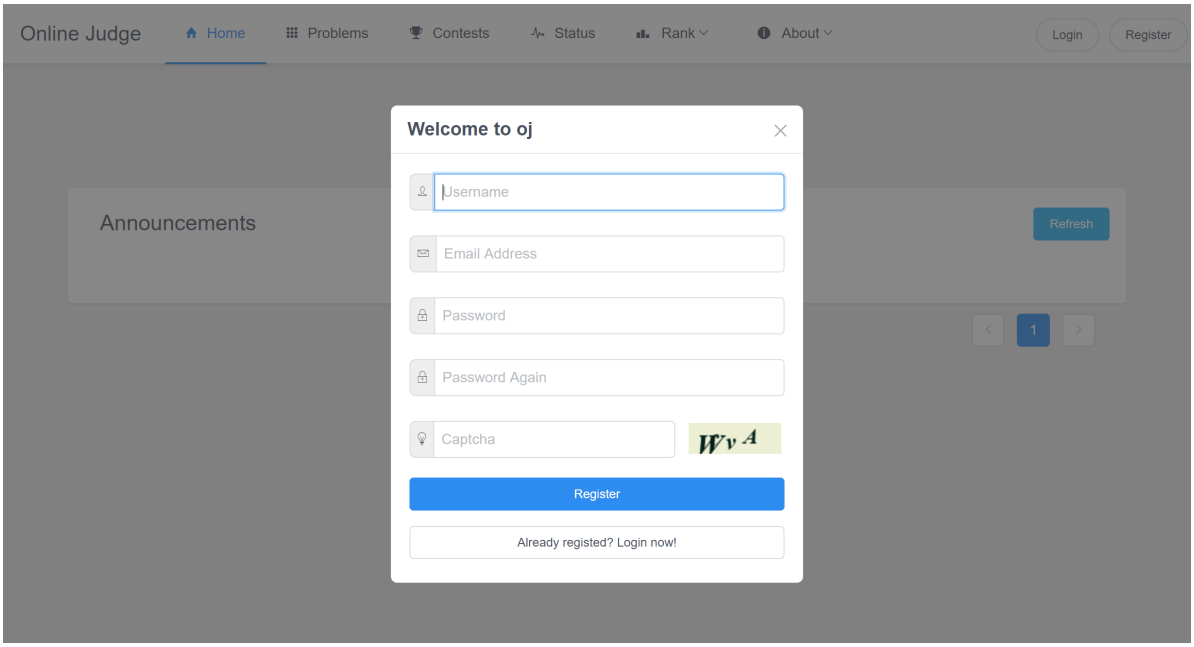
3.4 提交作业与评分

现在我们不仅把代码写好了，还对它做了点测试相信它大概是对的。所以我们现在就来把它提交到助教准备的在线测试（OnlineJudge, 简称OJ）网站，取得应得的分数！

首先，打开OJ网站，网址为114.212.84.18。因为我们的网站架设在内网，所以如果在没有校园网的外网环境下，大家需要使用南大的vpn才能访问到OJ网站。

3.4.1 注册账号

点击右上角 register 按钮，然后就能看到以下内容：

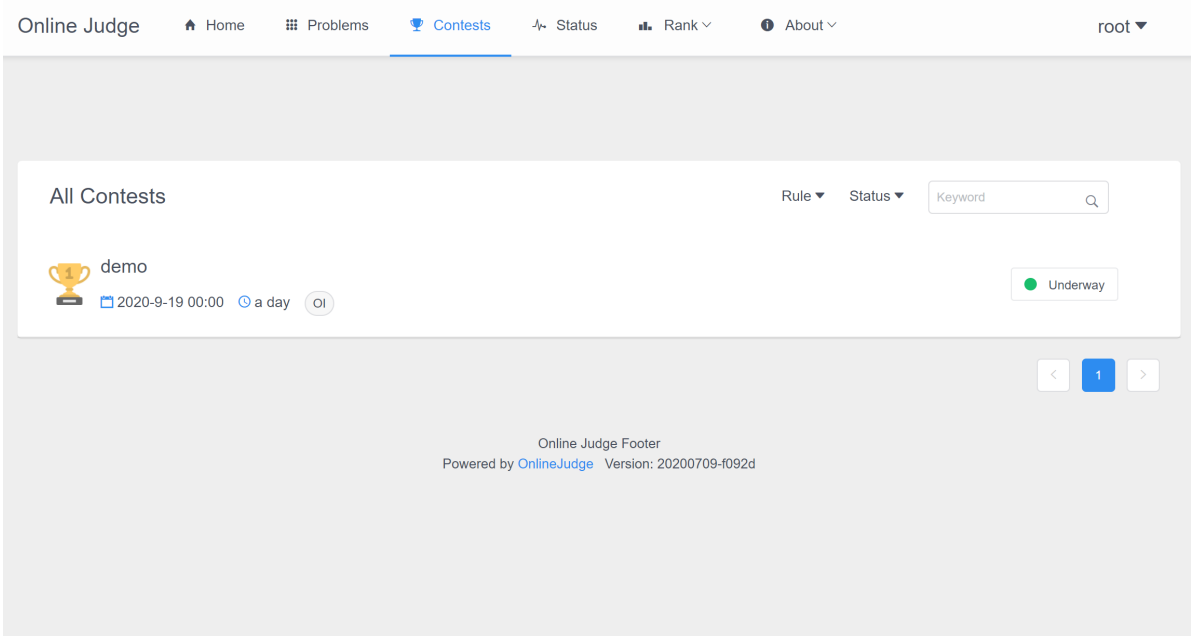


Username请务必正确填写自己的学号。我们会按照Username的学号来把这个账号获得的分给到你的成绩单上。Email Address和Password按自己心意填写即可。

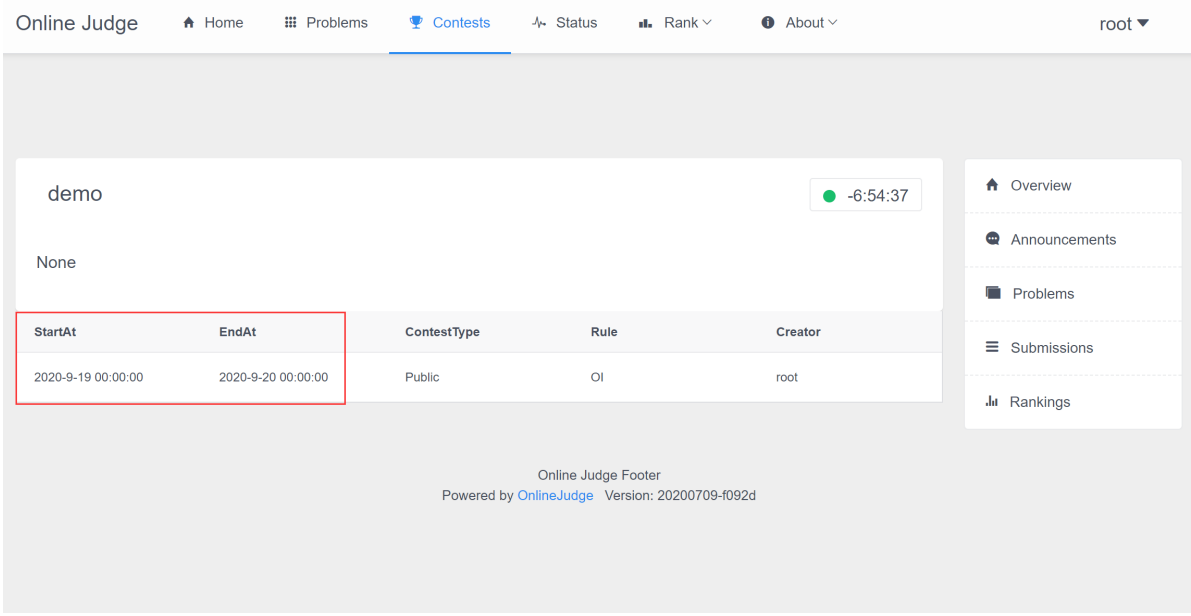
注册完成之后网页会自动跳转到登录界面。大家可以输入自己的学号和刚刚填写的密码进行登录。

3.4.2 提交代码

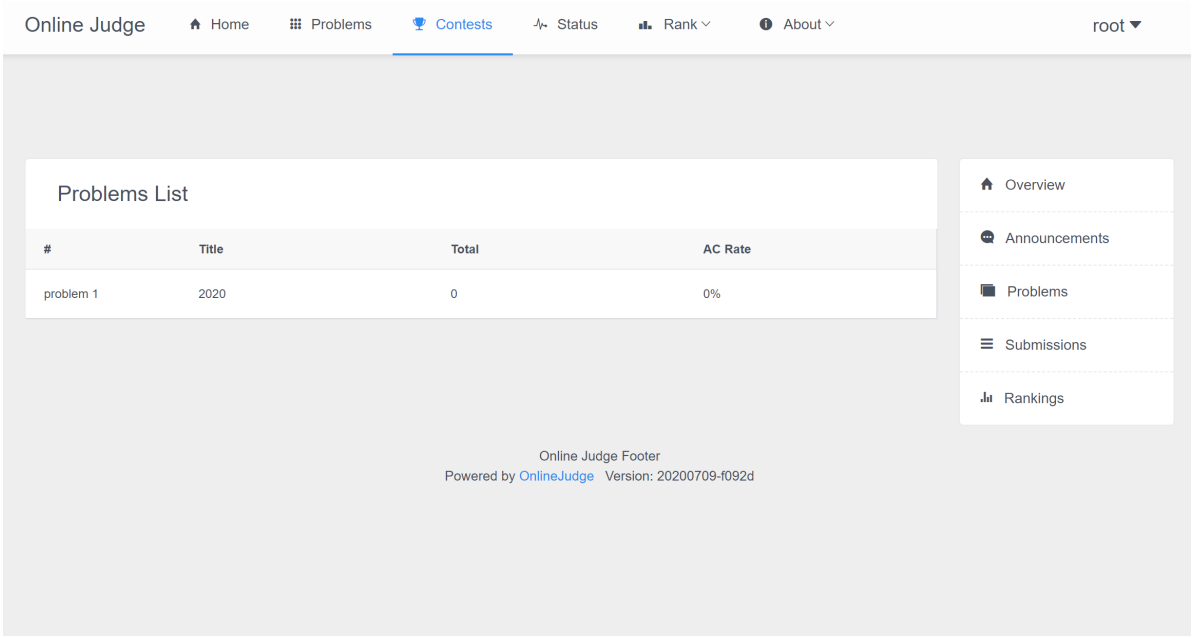
接下来点击页面顶部Contests菜单，进入Contests页面。我们以后的lab和homework都会以contest的形式来发布。下图中显示了一个叫做demo的contest。在大家进行实验的时候，会有一个叫做lab00的contest。请大家到时候进入lab00完成接下来演示的操作。



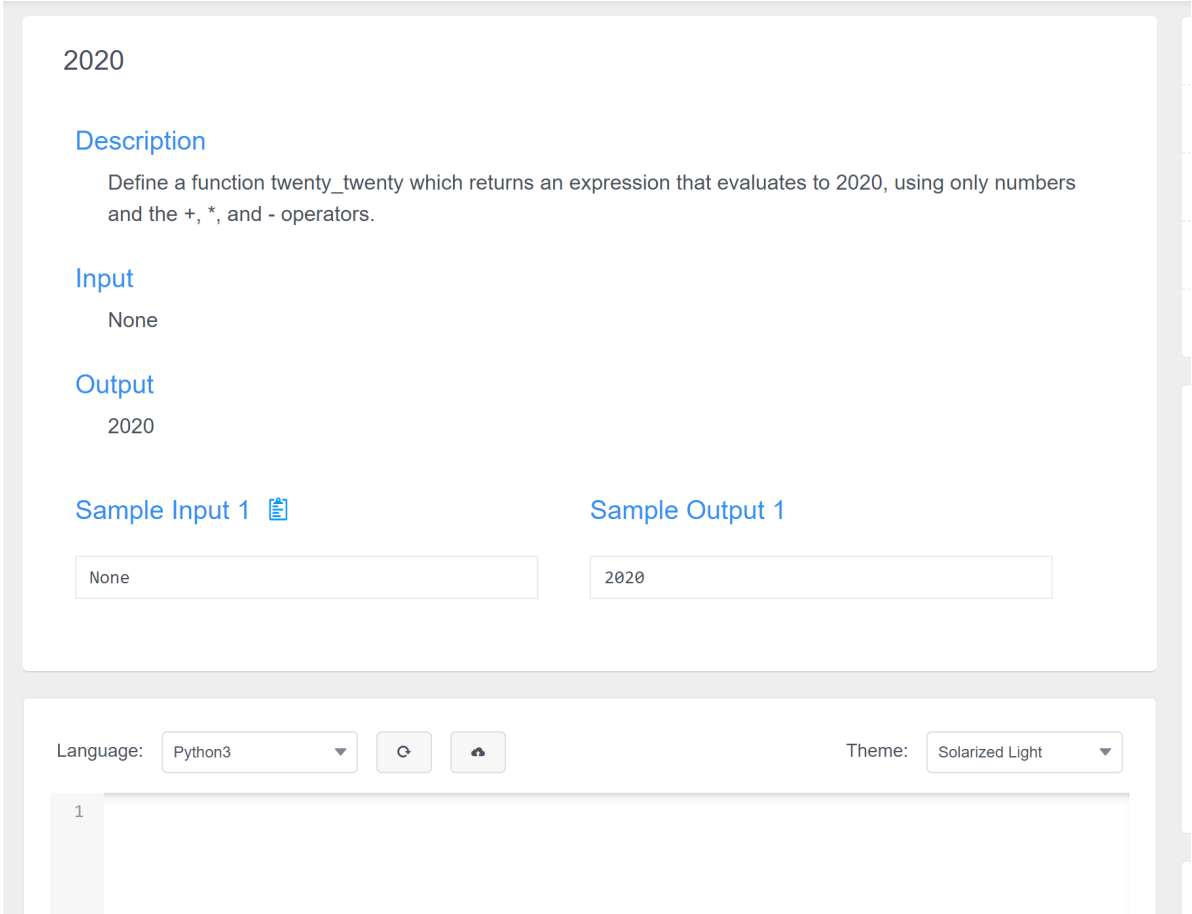
接下来，点击进入我们的demo contest（如下图所示）。首先可以看到此次作业的发布时间和截止时间。大家一定要留意ddl，否则过了时间，就无法提交作业了。



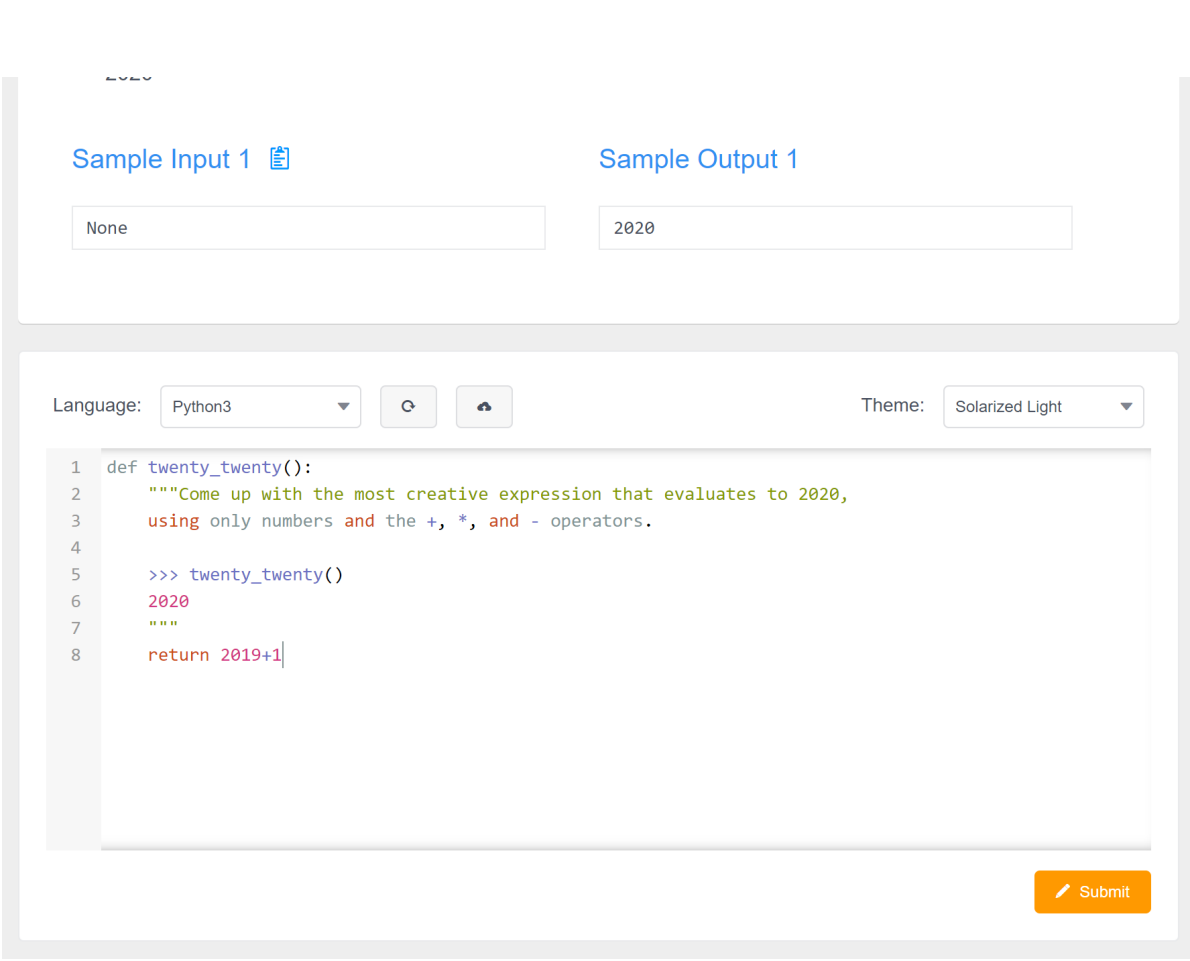
然后，大家点击右侧的Problems，查看本次contest有哪些题目。网页如下图所示。



作为演示，我们这边只显示了之前我们带大家做的题目“2020”。接下来，点击进去，首先能看到题目描述 (如下图所示)。实验讲义和OJ网站上题目的描述都应该是一致的。



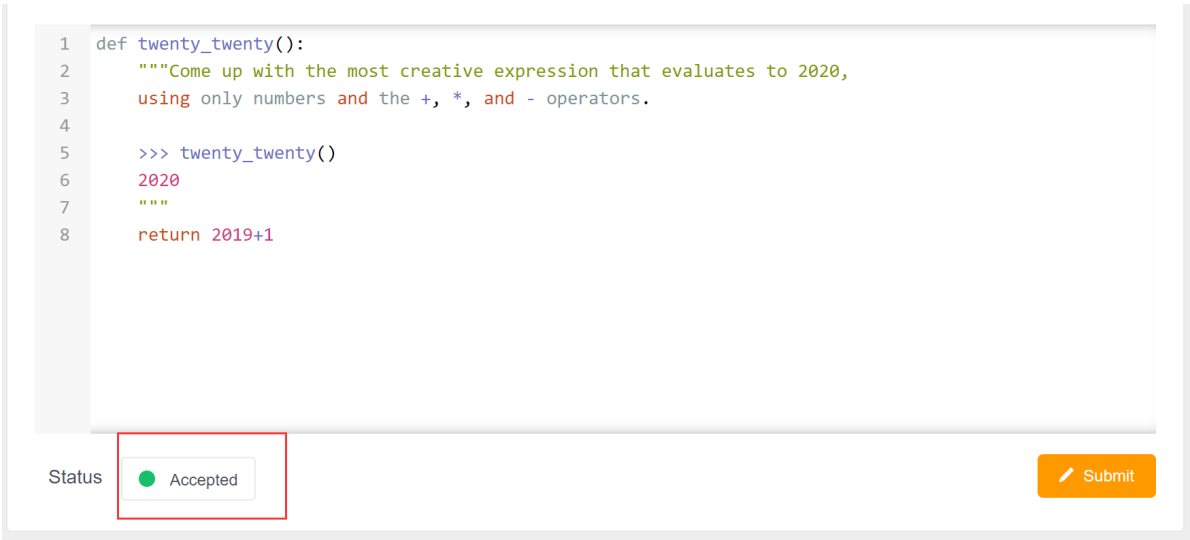
然后我们把`lab00.py`中关于twenty_twenty中的代码复制过去，即得到如下图的结果：



最后，点击右下角的submit按钮，你的代码就完成提交了。

3.4.3 查看结果

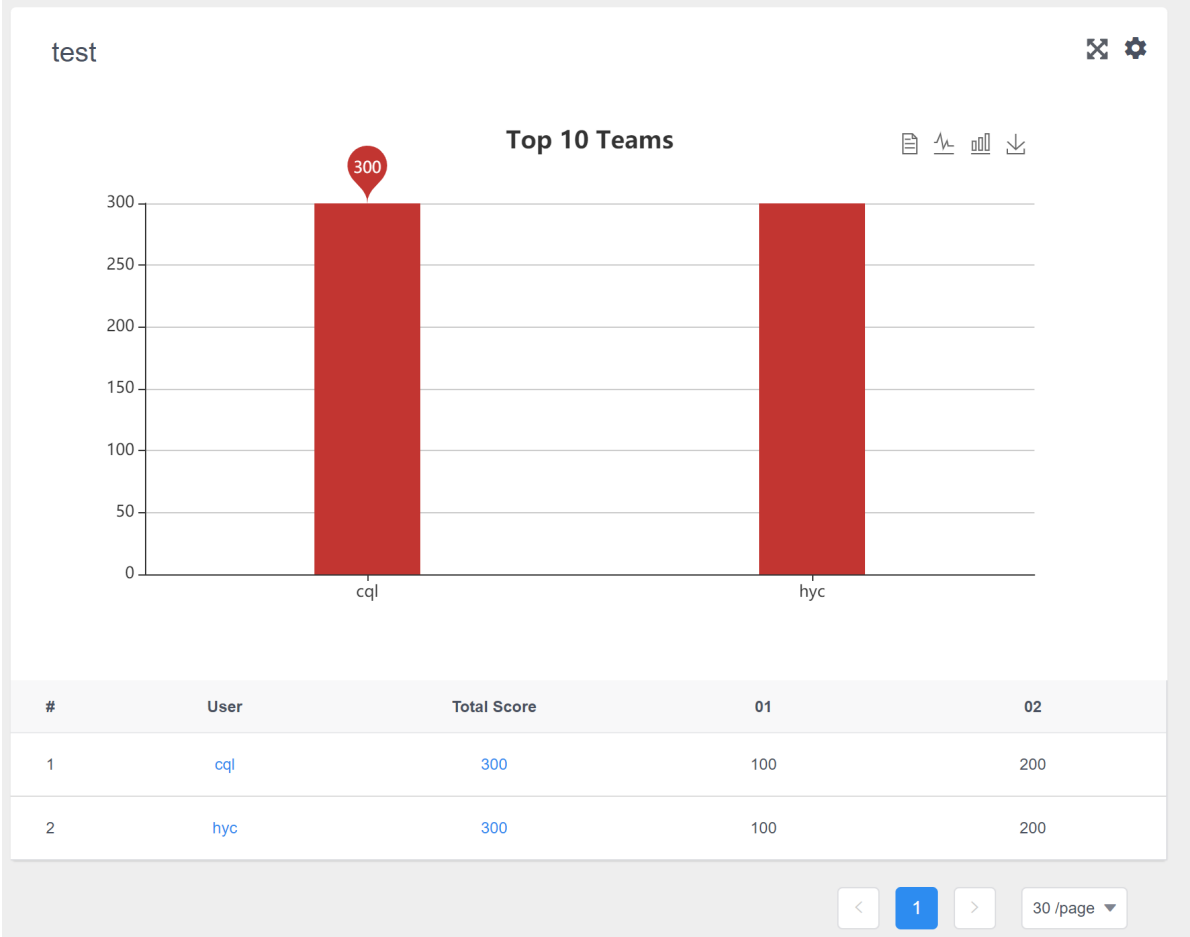
我们使用的OJ网站能即时地给大家反馈测试结果，以方便同学们重新修改代码再次提交。下图显示了点击submit按钮后的结果——Accepted。大家也可以点击右侧的submissions查看自己历次的提交结果。



OJ网站在我们的实验设置下可能会显示以下的结果：

- Accepted(AC): 恭喜！你拿到了这道题的所有分数！
- Partial Accepted: 你的代码只通过了部分测试，但你依旧能获得相应的部分分数。大家的每一份努力都不应该被辜负。
- Wrong Answer(WA): 你的代码没有通过任何测试。很遗憾，你无法获得任何分数。
- Runtime Error(RE): 你的代码触发了运行时的错误。建议按照3.3节的指导在本地测试一下。
- Time Limit Exceeded(TLE): 你的代码运行时间超出了限制。比如出现了死循环。
- Memory Limit Exceeded(MLE): 你的代码使用了过多内存。
- System Error: OJ服务器出了问题。需要联系我们修复。

此外，大家可以通过点击contest右侧的rank来查看自己目前获得的总分，如下图所示。下图中的contest有两道题。第一题满分100分，第二题满分200分。我们会在实验讲义上给出每道题的总分。



#	User	Total Score	01	02
1	cql	300	100	200
2	hyc	300	100	200

1
30 /page

3.4.4 其它要点

> 注意，OJ网站上，同一个题目的最新的提交结果会覆盖之前的提交结果。所以请大家务必确保每道题最后一次提交是你分数最高的代码。为了确保这一点，大家可以通过查看rank来确定自己的分数。助教们会从这里统计分数并录入成绩。

4. 本次实验内容

到此，大家应该顺利地做完并提交了第一道题目。希望这段旅程对大家足够顺利。接下来，我们还有两道简单的题目让大家再熟悉一下整个实验流程。如果你足够熟练的话，可能几分钟就做完了。此次实验的ddl设置为一周，大家不用着急。

Problem 1: 2020 (100pts)

定义一个函数`twenty_twenty`，用数字和`+`,`*`和`-`运算符写一个最酷炫的表达式，让它的值为2020.

```
def twenty_twenty():
    """Come up with the most creative expression that evaluates to 2020,
    using only numbers and the +, *, and - operators.

    >>> twenty_twenty()
    2020
    """
    return ______
```

Problem 2: A + B (100pts)

定义一个函数`sum`，参数是两个整数，返回值为它们两个的和。

```
def sum(a, b):
    """Compute the sum of a and b

    >>> sum(1, 2)
    3
    >>> sum(3, 8)
    11
    """
    return
```

Problem 3: Square Sum (100pts)

定义一个函数`square_sum`，参数是两个整数`a`、`b`，返回值为`a`的平方与`b`的平方之和。

```
def square_sum(a, b):
    """Compute the sum of square a and square b

    >>> square_sum(3, 4)
    25
    """
    return
```