

🔗 楼+之Python实战第10期 (/courses/1190)

Python 基础语法

简介

楼+ 课程的同学技术基础不同，本周实验内容只包含 Python 最常用到的知识点，仍然是以动手实践为主，一些高级用法会在后续的项目实战中用到的时候再详细讲解。

如果先前没有 Python 基础，请务必仔细学习本周的所有实验，每个知识点都要弄懂。如果遇到无法理解的问题，欢迎随时在讨论区或 QQ 讨论组中提问及讨论。

本周脑图

Python3 的基础知识点比较多，为了帮助大家梳理，我们整理了一份脑图包含了尽可能多的 Python3 的基础知识。这份脑图大家可以在本周每个实验中都进行对照学习，让动手实践过程中学习到的知识点建立更加清晰的体系。

请点击以下链接进行访问：

- Python3 基础知识点脑图
(<http://naotu.baidu.com/file/f3effc51a94420eb5686647e9f973326?token=00e911d0ed10e629>)

请注意，由于学习时间的限制，我们在本周的实验中只会实践学习最常用的知识点，部分知识点会在后面的项目中使用到。如果在学习的过程中，遇到任何不清楚的内容，请发到 QQ 讨论组里与同学和助教进行讨论，技术交流也是学习成长的必经之路。

知识点

- Python 开发环境
- 数据类型与变量
- 字符串
- 运算符
- 条件判断与循环
- 终端运行 Python 程序
- 模块
- 异常处理
- Python 包管理工具

Python 是什么

Python 是一种编程语言，能够快速上手并且功能强大。Python 提供了非常丰富的基础模块和第三方模块，可以直接在程序中使用，这些模块包含了数据库、Web 开发、文件操作等一系列的功能，避免了重复造轮子，提高编程的效率。

目前的 Python 版本包括 Python 2 和 Python 3（通常使用比较广泛并且经典的两个版本分别是 Python 2.7 和 Python 3.5，两个版本之间代码不兼容），3 比 2 有很大的改进，大部分的库都已经支持了 Python 3，同时 Python 的核心团队计划在 2020 年停止支持 Python 2，所以本系列课程使用的是 Python 3 版本（具体是 Python 3.5 版本）。如果已经有过 Python 2 的编程基础，可以非常容易的转向 Python 3。

优点与缺点与应用场景

人生苦短，我用 Python。

这不是一句戏言，Python 最大的优势就是简单，少量的代码实现复杂的功能。对比其他的编程语言，C 语言实现一个简单的命令行聊天室可能至少需要上千行代码，同样功能的，在 Python 中一百多行就能够解决了。并且，Python 的基础库和第三方库足够多，我们开发的时候可以利用的现成的模块非常多。这一点，能够极大提高工作效率。

缺点方面，Python 是一个解释型的编程语言，每次执行的时候会一行一行的解释执行，因此执行的性能比不上编译型的语言。性能的损失之外，Python 程序的源代码是完全开放的。因为 Python 程序是脚本，不是编译成二进制分发的，通常很难进行代码保护。

实验楼 (<https://www.shiyanlou.com>) 网站就是 Python 开发的，我们的大部分组件，例如实验环境管理模块、课程管理、Web 服务等都是 Python 开发的。我们选择使用 Python 的一大因素是能够快速开发，团队学习成本低，虽然性能上有损失，但结合我们自己的业务完全可以接受。此外，国内的 豆瓣 (<https://www.shiyanlou.com>) 也是 Python 开发的。

目前，Python 最主要的应用场景包括网络爬虫，Web 开发，自动化运维，数据分析等领域，这些主要领域，楼+课程中分别提供了实战项目进行学习。并在每个实战项目之后，提供了一系列功能扩展的挑战等待你自己独立思考完成。

楼+ Python 实战课程虽然不能保证你学完后就是个 Python 大牛（这仍然需要很长时间的多次代码实战练习，请不要幻想有任何不通过努力就能够技能快速成长的课程），但课程体系的设置是让你在最短的时间内能够掌握 Python 最主要的应用开发方式，遇到类似的问题能够自己使用 Python 快速解决。

Python 的语言特点

- 跨平台：Python 程序几乎可以在任何主流平台下运行，包括：Windows、Mac OS X、Linux 系统。但是注意跨平台不是指程序不进行任何修改直接就在不同环境下运行，很多地方还是需要考虑不同操作系统相关的环境因素，但是由于 Python 的编译器本身在大多数环境下可以运作，再加上 Python 拥有丰富的函数式库，所以这样大大降低了跨平台程序将会遇到的问题。

- 解释型：在程序运行的时候，通过解释器对 Python 程序逐行进行解释，然后直接运行。编译的结果会被保存在 .pyc 文件中，当程序下一次运行的时候会先在硬盘中寻找 .pyc 文件，如果找到就直接载入，否则就重复上面的过程。因此生成 .pyc 文件的目的在于可复用。同时在载入的时候会检查 .py 文件和 .pyc 文件保存的最后日期，如果不一致会重新生成一份 .pyc 文件。
- 脚本语言：这主要强调 Python 采用的是解释执行的方式运行，这一点区别于编译型语言。脚本语言一般都有更高层次的抽象和封装、更少的代码量、高可读性、学习曲线更平缓等特点。
- 动态语言：也就是在运行的过程中可以修改代码；这一点区别于静态语言，静态语言是在编译的时候已经确定好代码，运行过程中不能修改。动态语言在运行的过程中可以添加新的函数、对象或是代码，同时也可以删除已有的对象属性或是方法。
- 默认 UTF-8 编码：Python 3 默认使用的是 UTF-8 编码可以正常解析中文；在 Python 2 中就需要在代码首行进行声明。

练习题：查看当前系统的 Python 3 版本号

本节将考察使用命令查看当前系统的 Python 3 版本号，请按照题目要求完成，点击 下一步 系统将自动检测完成结果。

在实验环境中双击打开 Xfce 终端，在命令行中输入如下命令：

```
python3 -V
```

然后就会输出当前系统使用的 Python 3 版本号。

程序完成后，点击 下一步 ，系统将自动检测完成结果。

Python 开发环境

实验楼提供的 Ubuntu 14.04 操作系统环境中已经安装了 Python 3.x。如果需要自己在 Linux 环境下安装 Python 也非常简单，默认的包管理都可以支持直接安装，例如 Ubuntu 下使用 apt-get 安装：

```
sudo apt-get update
sudo apt-get install python3
```

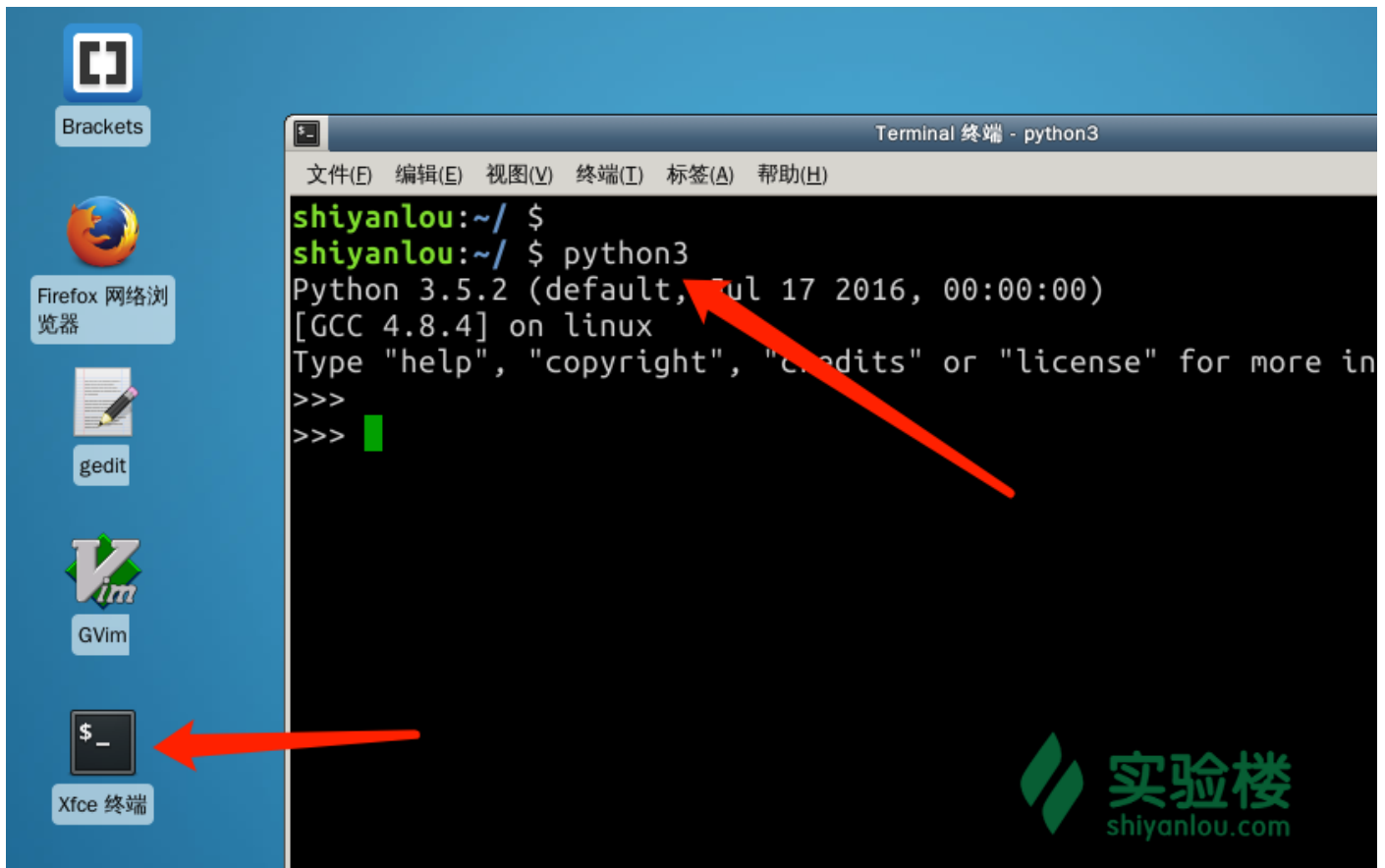
在 Mac 下可以使用 `brew install python3` 命令进行安装，而 Windows 下则可以在 Python 官网下载后通过图形界面一步步点击下一步进行安装。

python 程序的执行方式一般分为两种，一种是通过交互式命令行执行，另一种是以程序文件的方式运行。

交互式

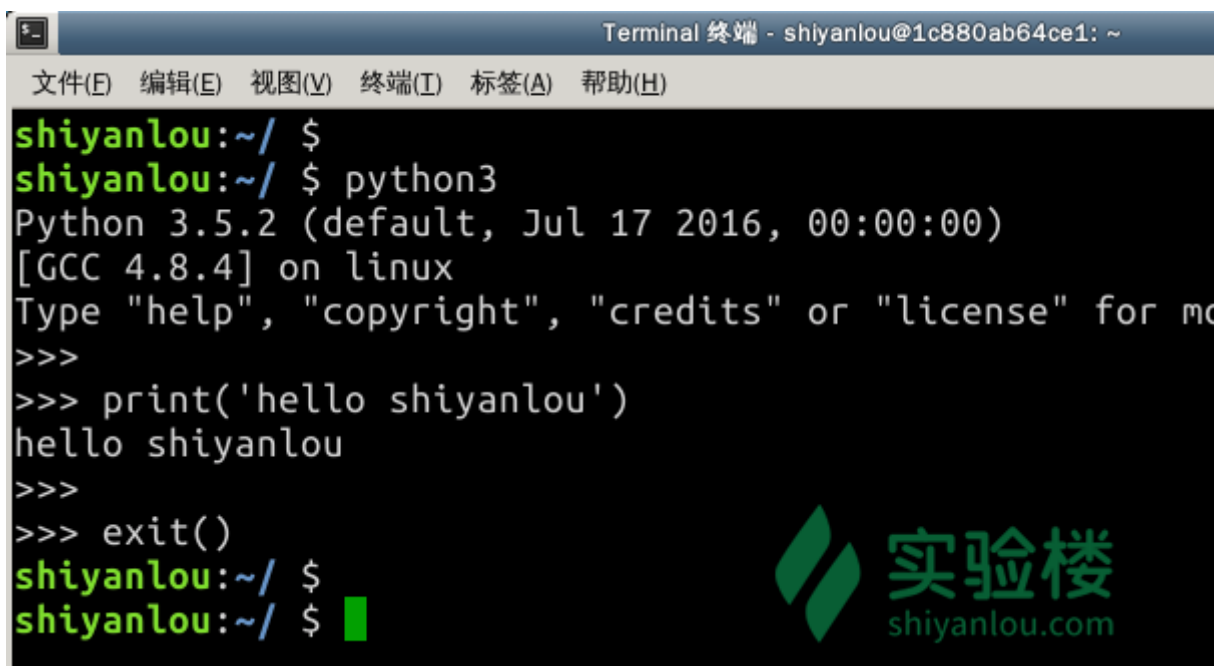
在实验环境中，我们使用 `python3` 命令启动进入到 Python 3 交互式的环境中，在这个环境中输入的代码都可以立即执行并得到输出结果。交互式的环境，在开发中经常用来调试和测试代码。

点击右边桌面上的 `xfce 终端`，启动 Linux 命令行终端，在终端中输入 `python3` 进入到 Python 交互式环境：



在这个环境中的 `>>>` 提示符后面输入代码，先输入 `print('hello shiyanlou')`，将一串字符串打印到屏幕上。

退出交互式环境的方式比较简单，可以使用 `Ctrl-D` 快捷键，也可以输入 `exit()` 代码退出。



执行脚本

楼+之Python实战第10期 (/courses/1190)

在开发过程中经常被使用到的 Python 程序是一个或多个存有源代码的脚本文件，通常会以 .py 作为扩展名存储。

当完成脚本的编写后，可以使用 Python 解释器来运行 Python 脚本。

创建一个简单的脚本并尝试执行，脚本中只包含上一节交互式环境中的一行代码，注意 echo 命令需要在 Xfce 终端执行，下面执行的这一条命令的目的是创建 shiyanlou.py 文件并写入内容：

```
echo "print('hello shiyanlou')" > /home/shiyanlou/shiyanlou.py
```

Linux命令讲解：

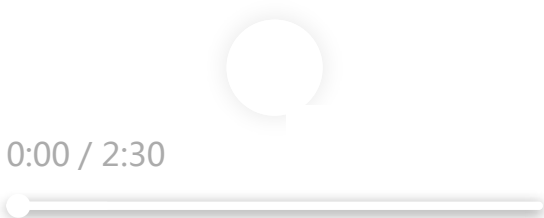
1. echo 命令是用于字符串的输出；比如在命令行中执行 echo "xxx"，就会在终端中打印出 xxx。
2. > 符号用于把前面输出的内容写入到 shiyanlou.py 文件中，并覆盖掉该文件中原来的内容。

直接使用 Python 3 解释器执行脚本：

```
python3 /home/shiyanlou/shiyanlou.py
```

```
shiyanlou:~/ $ [16:48:38]
shiyanlou:~/ $ echo "print('hello shiyanlou')" > /home/shiyanlou/shiyanlou.py
shiyanlou:~/ $ python3 /home/shiyanlou/shiyanlou.py [16:50:30]
hello shiyanlou
shiyanlou:~/ $ [16:50:41]
shiyanlou:~/ $ [16:50:44]
```

基本操作视频：



练习题：运行程序，打印 hello world

本节将考察使用命令行运行一个 Python 文件，请按照题目要求完成，点击 下一步 系统将自动检测完成结果。

使用命令行的方式创建 /home/shiyanlou/hello.py 文件，并向其中写入 print('hello world')。

参考命令如下：

🔗 楼+之Python实战第10期 (/courses/1190)

```
echo "xxx" > /home/shiyanlou/hello.py
```

最后使用命令运行 `hello.py` 文件，查看输出结果：

```
$ python3 /home/shiyanlou/hello.py  
hello world
```

初次尝试

现在我们开始尝试编写第一个 Python 程序，这个程序的作用是计算一个半径为 5 的圆的面积，并输出。

在这个程序中，我们将初次接触 Python 的脚本编写与执行、模块引入和使用的概念。

选择编辑器

实验楼环境中内置了很多种代码编辑工具，可以根据自己的需求选择：

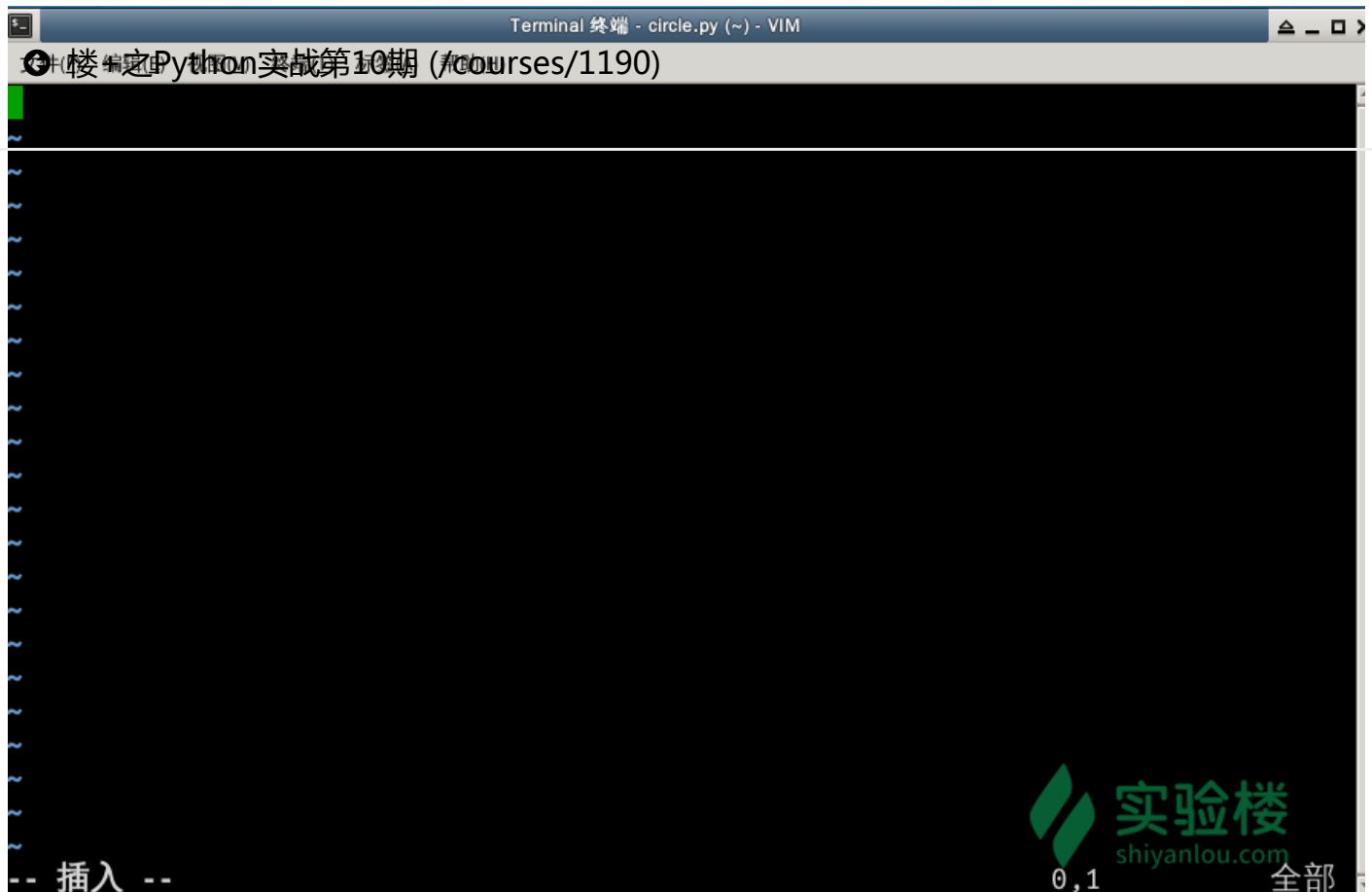
1. vim：程序员最喜欢的编辑工具，命令行中执行，上手比较复杂，感兴趣可以先学习下实验楼的 Vim 基础课程 (<https://www.shiyanlou.com/courses/2>)
2. gedit：类似记事本，编写的时候有代码高亮，注意保存的文件路径
3. sublime：又一个 Python 程序编写常用的编辑器，有很多插件可以选择，有代码高亮和提示等功能

在本课程的学习过程中，如果对 Linux 比较熟悉，推荐使用 Vim，如果不熟悉，推荐使用 sublime。下面的步骤将使用 Vim 编辑器。

打开文件

在 Xfce 终端中输入 `vim circle.py` 来启动 Vim 并编辑 `circle.py`，启动后不要乱按键。

然后按 `i` 键进入插入模式，此时你可以看到左下角有“插入”两个字，现在你可以键入下面的代码了。



第一行

第一行代码输入下面的内容：

```
#!/usr/bin/env python3
```

其中第一行的前两个字符 `#!` 称为 Shebang，目的是告诉 shell 使用 Python 3 解释器执行其下面的代码。

如果有这行代码，并且给脚本通过 Linux 的 `chmod a+x XXX.py` 命令增加了执行权限，则可以使用 `./XXX.py` 这种方式直接执行脚本，否则需要用 `python3 XXX.py` 这种方式执行。

模块与引入

由于要计算圆的面积，所以我们需要知道 π 值，就是 3.1415... 这个小数。我们会使用到一个基础库 `math` 这个基础库中包含大量常用公式和数学相关的处理函数，当然也包含我们需要的 π 值。

输入代码：

```
from math import pi
```

这句代码的意思是从 `math` 库中引入 `pi`，让当前的文件中可以使用 `pi`。



实验楼+之Python实战第10期 (/courses/1190)

使用引入的 π 值，计算半径 5 的圆的面积，相信你还没有忘记圆面积的计算公式，在文本中输入代码：

```
# calculate
result = 5 * 5 * pi
```

此处代码中以 # 开始的一行是程序的注释，用来帮助我们阅读和理解代码，良好的注释习惯是能够提高程序的可维护性，哪怕是写的很好的代码，如果没有注释，过一段时间连作者都很难去改动了。

最后一行代码中，我们定义了一个变量 result，用来保存 $5 * 5 * \pi$ 公式计算的结果。

输出

输出的代码很简单，就是上一节中使用的 print 函数：

```
print(result)
```

所有代码

代码合并到一起：

```
#!/usr/bin/env python3
from math import pi
# calculate
result = 5 * 5 * pi
print(result)
```

保存代码

然后按 Esc 键退出插入模式，再键入 :wq 回车，Vim 就会保存文件并退出。

执行代码

执行代码有两种方式，一种是先前写的用 python3 命令指定脚本 python3 执行，一种是本节我们将要使用到的直接执行脚本。

要运行脚本文件 circle.py，还要为文件添加可执行权限：

```
$ chmod +x circle.py
```


然后执行脚本：

🔗 楼+之Python实战第10期 (/courses/1190)

```
$ ./circle.py
```

执行脚本的时候前面需要输入 `./` 表示在当前目录下的脚本。

脚本执行的结果如下图所示：

```
shiyanolou:~/ $ ls -l circle.py [16:55:17]
-rw-rw-r-- 1 shiyanolou shiyanolou 91  9月  5 16:55 circle.py
shiyanolou:~/ $ cat circle.py [16:55:24]
#!/usr/bin/env python3

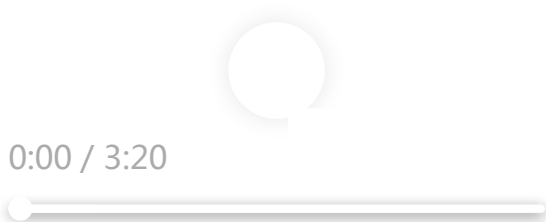
from math import pi

# calculate
result = 5 * 5 * pi
print(result)
shiyanolou:~/ $ [16:55:29]
shiyanolou:~/ $ chmod +x circle.py [16:55:34]
shiyanolou:~/ $ ./circle.py [16:55:39]
78.53981633974483
shiyanolou:~/ $ [16:55:44]
```

注意的细节

1. Vim 上手比较困难，如果有难度，请直接使用 sublime 或 gedit 编写代码
2. Python 代码对缩进的要求很严格，这个例子中间所有的代码都要靠最左侧放置，否则就会报错

完整的一个 Python 程序编写操作视频：



- 拓展学习 实验楼课程-《Vim 编辑器使用》(<https://www.shiyanlou.com/courses/2>)
- 拓展阅读 《Sublime 代码编辑器介绍》(<https://baike.baidu.com/item/Sublime%20Text?fromtitle=Sublime&fromid=8130415>)

数据类型与变量

计算机不仅能处理数字之间的运算，同时也能对文本、图片、音频、视频等数据进行处理。在各个编程语言中，不同的数据对应着不同的数据类型，Python 中的主要数据类型有 int(整数)/float(浮点数)、字符串、布尔值、None、列表、元组、字典、集合等，后四种数据类型比较复杂，我们将在下一节实验中详细讲解。

None 和 Bool

前面已经接触了数值和字符串，下面简单介绍一下 None 和布尔值。字符串将在本节后面的文档中详细讲解。

None

每个语言都有一个专门的词用来表示空，例如 JavaScript 中的 null，MySQL 中也是用 null 表示空，Python 中使用 None 表示空对象，注意它与空字符串、数值 0 是不同的：

```
>>> type(None)
<class 'NoneType'>
>>> type('')
<class 'str'>
>>> type(0)
<class 'int'>
```

当我们在计算机系统中安装了 Python，这个 None 对象就自动生成了，它在内存中的地址（就是一串十进制的数字）就不再变化了，所以我们判断一个对象是否为 None，通常用 is，而不是 == 来判断：

```
>>> '' is None
False
>>> test = None
>>> test is None
True
```

Bool 布尔值

Python 中使用 True 和 False 来表示布尔值，注意首字母大写，即判断 Python 对象、返回值、表达式真假的一组特殊数据类型。通常数值 1 和 0 也有同样的作用。任何非零数值的布尔值都是 True，0 的布尔值是 False。非空字符串的布尔值为 True，空字符串的布尔值为 False。以此类推，空列表、空元组、空字典、空集合的布尔值为 False，以上非空数据类型的布尔值为 True，None 的布尔值也是 False。

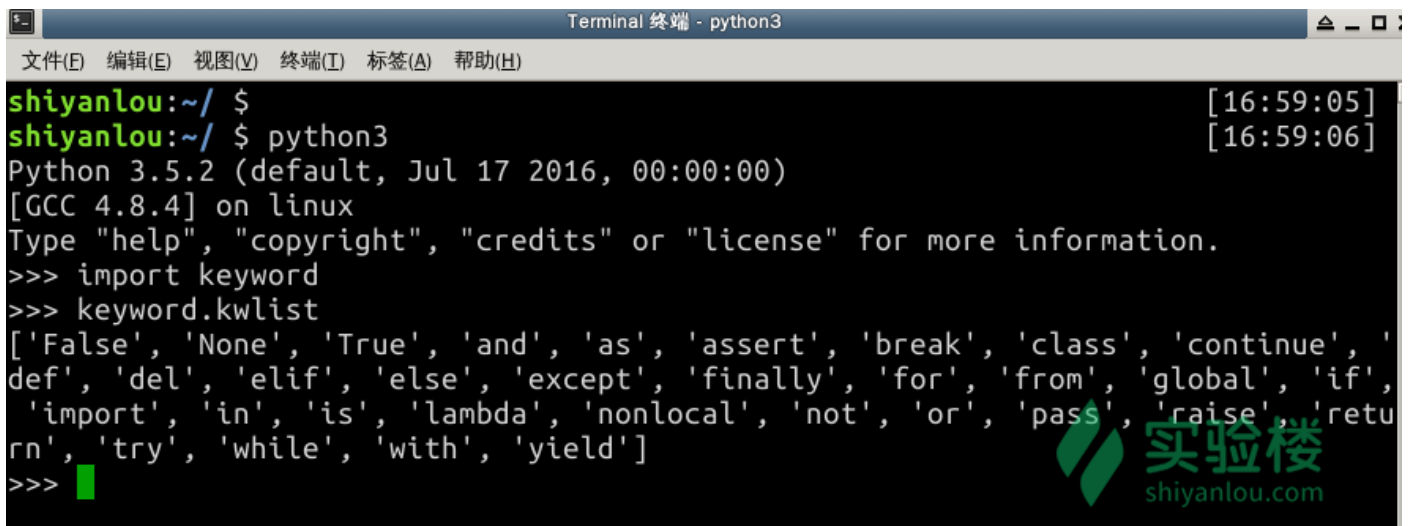
Python 内置方法 bool 可以用来判断对象的布尔值：

```
>>> bool(1)
True
>>> bool(0)
False
>>> bool(False)
False
>>> bool(True)
True
>>> bool('False')
True
>>> bool([])
False
>>> bool('')
```

保留字与标识符

保留字，又称为关键字，每种语言都有自己的一套预先保留的特殊标识符，Python 也不例外，它自带的 keyword 模块可以查看全部关键字。在 Python3 交互式命令行中执行如下命令，引入 keyword 模块就可以查看到 Python 中的关键字：

```
import keyword
keyword.kwlist
```



```
Terminal 终端 - python3
文件(E) 编辑(E) 视图(V) 终端(T) 标签(A) 帮助(H)
shianlou:~/ $ [16:59:05]
shianlou:~/ $ python3 [16:59:06]
Python 3.5.2 (default, Jul 17 2016, 00:00:00)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
>>>
```

这些保留字是需要配合其它程序语句共同使用，然后在这个过程中发挥它们各自的作用。例如 def 关键字用来创建函数；for 定义循环语句（本节后面会讲到）；and 表示“与”；lambda 生成匿名函数；return 定义函数的返回值，等等。

在 Python 中一切皆对象。标识符就是这些对象的代号，例如 s='hello world'，s 就是一个标识符，它是自定义变量，变量值是一个字符串。标识符通常定义是计算机语言中允许用作名字的有效字符串集合。它往往被用作变量、函数、类的名称。在 Python 中设置标识符（定义变量名）需要遵循一定的规则。

有两类不能用作标识符（变量名）：

🔗 楼+之Python实战第10期 (/courses/1190)

1. 保留字，又称为关键字（其实就是特殊标识符），例如：False、True、with、if 等，它们都是 Python 保留的标识符，也就是指其它命名都不能使用这些保留的标识符。如果使用了将会出现语法错误（SyntaxError 异常）。
2. 内置方法名称。它们是 Python 提前定义好的一些模块，这些模块可以随时被程序所调用。所以为了避免冲突，尽量也不要使用这些内建模块名作为自定义的标识符。比如：int、len、max 等等，注意这些名字如果被当做自定义变量名，并不会报错，但会导致对应的内置方法失效。

内置函数(方法)

函数在一些情况下又被称作方法，例如在类中。Python 内置了大量很好用的函数，这些函数分别支持一些基本的功能。可以通过在 Python 交互式解释器中执行 help() 来获取这些内置函数的帮助，例如我们希望查看 len() 函数的作用：

```
>>> help(len)
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

返回的信息中告诉我们，len() 可以获取一个容器中元素的数量。

Python 中常用的内置函数列表如下，后续实验和项目会用到很多，大家无需记忆，只需要有个基本印象：

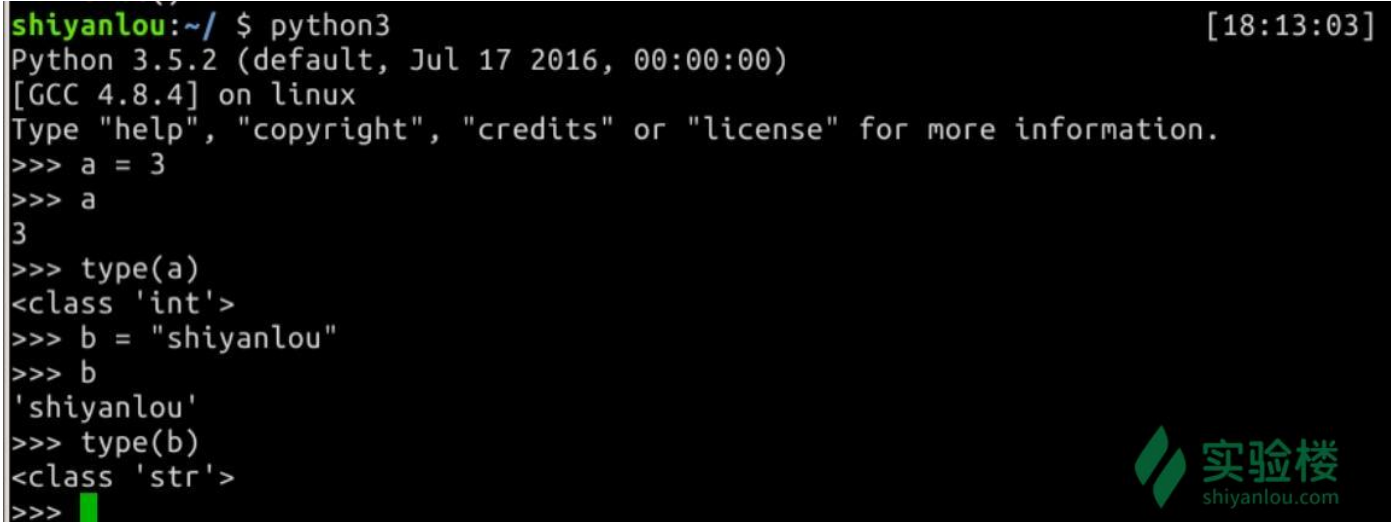
Built-in Functions				
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	

变量

编程语言中为了能够更好的处理数据，都需要使用一些变量。变量基本上就是代表(或是引用某值的名字)。Python 语言的变量可以是各种不同的数据类型，使用变量的时候不需要声明，Python 解释器会自动判断数据类型。使用 `type(变量)` 可以查看该变量的类型。

在 Python 3 交互式命令行中执行如下命令：

```
a = 3
a
type(a)
b = "shianlou"
b
type(b)
```



```
shianlou:~/ $ python3
Python 3.5.2 (default, Jul 17 2016, 00:00:00)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 3
>>> a
3
>>> type(a)
<class 'int'>
>>> b = "shianlou"
>>> b
'shianlou'
>>> type(b)
<class 'str'>
>>>
```

在上面的操作中，用 `a` 代表了整数 3，这个过程被称为“赋值”，也就是整数 3 被赋值给了变量 `a`，而且 `a` 的类型也被定义为了整型。同样的，把字符串 `shianlou` 赋值给了变量 `b`，那么以后就可以直接在表达式中使用变量 `b` 了。

变量命名规则

变量的命名属于标识符命名的一种。因此变量命名也不能使用保留字和内置函数名，除此之外还遵循一些其它规则。

变量命名一般有两种方式：驼峰命名法和下划线命名法。

驼峰命名法具体分为两种：

- 小驼峰：第一个单词首字母小写，后面其它单词首字母大写。比如：`firstName`、`myLastName` 等等。
- 大驼峰：每个单词的第一个字母都大写。比如：`FirstName`、`MyLastName` 等等。

下划线命名法需要注意以下几点：

- 第一个字符必须是字母表中的字母(大写或小写)或者一个下划线('_')。

- 命名的其它部分可以由字母(大写或小写)、下划线('_')或数字(0-9)组成。一般而言, 单词与单词之间采用下划线分割开。
👉 楼之Python实战第10期 (/courses/1190)
- 命名区分大小写。比如: `my_name` 与 `My_Name` 是两个不同的变量名。

因此, 符合下划线命名法的例子有: `i`、`A`、`my_name`、`a1_b2_c3`、`My_Name`、`__first_name` 等等。

不符合下划线命名法的例子有: `2`、`my name`、`first-name` 等等。

不管是哪一种命名法都需要注意, 命名的时候要做到词达其义, 变量的命名就能够很方便的表明它的用途或是含义, 增加代码的可读性。

一般而言, 变量的命名风格可以根据个人习惯或是喜好选择驼峰命名法或是下划线命名法。但是 Python 官方推荐使用使用的是下划线命名法, 下划线命名法相对而言会更加方便理解。另外, 团队协作开发时也许也会规定使用哪一类变量命名方法, 以使代码风格更加统一。

变量声明与赋值

Python 中的变量赋值不需要类型声明。每个变量在内存中创建, 都包括变量的标识, 名称和数据这些信息。每个变量在使用前都必须赋值, 变量赋值以后该变量才会被创建。等号 `=` 用来给变量赋值。等号运算符左边是一个变量名, 右边是存储在变量中的值。

在 Python 3 交互式命令行中执行如下命令:

```
>>> number = 12 # 整型变量
>>> float_number = 1.68 # 浮点型变量
>>> school = 'No.1 School' # 字符串型变量
>>> print(number)
12
>>> print(float_number)
1.68
>>> print(school)
No.1 School
```

两种特殊的赋值格式:

```
>>> a = b = 1 # 连等号赋值给两个变量相同的值
>>> print(a)
1
>>> print(b)
1
>>> c = 1; d = 1 # 分号为语句分隔符, 相当于换行符
>>> print(c)
1
>>> print(d)
1
```

使用变量及打印

实验楼+之Python实战第10期 (/courses/1190)

Python 3 中，包括以下几种基本数据类型，本节打印变量中会用到其中的一些数据类型：

1. 整数：例如 100，-200，0 等
2. 布尔数：True 或 False
3. 浮点数：小数，例如 1.5，2.5
4. None：空值，注意与0是不同的，可以理解为未定义的值。

Python 可以处理整数，包括负整数，在程序中的表示方法和数学上的写法一模一样，例如：1、1024、-96、0 等等。

浮点数也就是小数，之所以称为浮点数，是因为按照科学记数法表示时，一个浮点数的小数点位置是可变的，例如 123.4 和 1.234×100 是完全相等的。浮点数可以用数学写法，如 1.23，3.1415926，-9.18，等等。对于很大或很小的浮点数，可以使用科学计数法表示，把 10 用 e 替代，1.2 乘以 10 的 9 次方就是 $1.23e9$ ，或者 $12.3e8$ ，0.000012 可以写成 $1.2e-5$ ，等等。

除了这四种之外，还有一些其他不常用的类型，例如复数。

在 XFce 终端中输入 `python3`，进入交互环境，尝试输入如下的代码，并理解输出的含义，注意执行后不要退出，需要继续下一节的实验内容：

```
>>> a = 10
>>> b = 10.6
>>> c = True
>>> d = None
>>> print(a,b,c,d)
>>> print(type(a),a)
>>> print(type(b),b)
>>> print(type(c),c)
>>> print(type(d),d)
```

在上述的代码中，`type` 是 Python 3 内置的一个函数，用来显示变量的数据类型。

```
>>> a = 10
>>> b = 10.6
>>> c = True
>>> d = None
>>> print(a,b,c,d)
10 10.6 True None
>>> print(type(a),a)
<class 'int'> 10
>>> print(type(b),b)
<class 'float'> 10.6
>>> print(type(c),c)
<class 'bool'> True
>>> print(type(d),d)
<class 'NoneType'> None
>>>
```



练习题：判断是否符合下划线命名法

本节将考察下划线命名法的命名规则，请按照题目要求完成，点击 [下一步](#) 系统将自动检测完成结果。

下面将列出一些变量的命名，请将符合下划线命名法进行命名的变量名写入 `/home/shiyanlou/name.txt` 文件中并保存，每一个变量名占一行。

```
analysisProvince、get_key_info、False、PersonOne、b3、if、5apple、__name__
```

程序完成后，点击 [下一步](#)，系统将自动检测完成结果。

空行、缩进与多行代码

对于空行、缩进与多行代码的使用，Python 官方推荐遵循 PEP8 (<https://www.python.org/dev/peps/pep-0008/>) 的代码规范。因此在写代码的时候也应该尽量符合规范进行书写，这样能够避免一部分不必要的 bug，以及使代码更加具有可读性。

空行

一般空行用于不同函数之间、不同类之间、以及类和函数之间进行分隔。空行不是 Python 的语法，即使不插入空行程序运行也不会出错。插入空行的主要目的是方便代码阅读以及日后的维护。

比如下面的例子，使用空行就可以提高代码的可读性：

楼+之Python实战第10期 (/courses/1190)

```
def a():  
    pass  
  
def b():  
    pass  
  
a()  
b()
```

缩进

Python 对于缩进的要求是非常严格的，它依靠缩进距离区分代码块，所以在 Python 中缩进也是一种语法，缩进的距离是可变的，具有相同缩进距离的属于同一个代码块，比如 if 或 else 下方的代码块必须保持相同的缩进。Python 中依靠缩进和冒号区分代码层次。

如果采用不合理的代码缩进，就会抛出异常。比如下面这个例子：

```
>>> if True:  
... print("haha")  
    File "<stdin>", line 2  
        print("haha")  
        ^  
IndentationError: expected an indented block  
>>>
```



上面这个异常表示的是需要一个缩进。

另外需要格外强调的一点是，**缩进的时候一定不要混用空格和 TAB，强烈建议只使用空格**，为了保持良好的代码风格，建议使用四个空格作为缩进。混用空格和 TAB，用人眼是区分不出来的，但是运行程序就会报语法错误。

比如下面这个例子：

```
shiyanolou:~/ $ vim test.py [14:51:49]  
shiyanolou:~/ $ python3 test.py [14:53:18]  
File "test.py", line 6  
    print(b)  
    ^  
TabError: inconsistent use of tabs and spaces in indentation  
shiyanolou:~/ $ cat test.py [14:53:22]  
#!/usr/bin/env python3  
  
a = 3  
if a == 3:  
    b = 4  
    print(b)  
shiyanolou:~/ $ [14:53:30]
```



在这个例子中，就是混用了空格和 TAB，导致抛出 TabError 异常。而用人眼是无法区分的，所以大家书写代码的时候一定要注意这个问题。

多行代码

楼+之Python实战第10期 (/courses/1190)

Python 代码中一般一行只写一条语句。如果想要在一行中写多条语句，那么语句之间就需要使用 `;` 进行分隔。

比如下面的例子：

```
>>> num1=1
>>> num2=2
>>> num3=3
>>> num1=1;num2=2;num3=3
>>>
```



如果一行语句太长，可以在语句的末尾使用斜杠 `\` 将一行语句进行分隔。

```
>>> num1=1;num2=2;num3=3
>>> total = num1 + \
... num2 + \
... num3
>>> total
6
>>>
```



不过如果语句在列表 `[]`、字典 `{}`、元组 `()` 中太长就可以直接换行，它们不需要使用斜杠。比如：

```
nums = [1,2,
        3,4]
print(nums)
```

代码块

Python 中依靠缩进的距离和冒号来区分代码块。

在复合语句(它们不是一行就结束，而是有多行，并且代码之间有明显的层次之分)中经常出现代码块，它们的首行都是以关键字开始(常见的有 `class`、`def`、`if`、`while` 等)，以冒号 `:` 结束，在冒号的下一行或是多行代码构成的就是代码块。

比如下面的例子：

🔗 [楼+之Python实战第10期 \(/courses/1190\)](#)

```
a = 3
```

```
if a < 2:
    print("-----")
    print("a<2")
elif a == 2:
    print("-----")
    print("a=2")
elif a == 3:
    print("-----")
    print("a=3")
else:
    print("a>3")
```

在这个例子中，`if a < 2:`、`elif a == 2:`、`elif a == 3:`、`else:` 各自下面的两句具有相同缩进距离的语句就被称为一个代码块。

字符串

Python 3 中的字符串可以使用双引号或单引号标示，如果字符串中出现引号，则可以使用 `\` 来去除引号标示字符串的特殊作用。

几种字符串的表示方法：

```
str1 = "hello"
str2 = 'shianlou'
str3 = 'hello, \'shianlou\''
str4 = "hello, 'shianlou'"
str5 = 'hello, "shianlou"'
```

注意 `str4` 和 `str5` 都没有使用 `\`，但仍然可以在字符串中使用引号，相信你已经发现了原因。


如果需要输入多行字符串，又该如何处理呢？可以尝试使用 `"""` 三个双引号：

```
str6 = """ hello,
shianlou """
```

支持使用 `+` 连接字符串：

```
str1 + ' ' + str2
```

索引

字符串是字符的有序集合，可以通过其位置来获得具体的元素。在 python 中，字符串中的字符是  通过索引来提取的，索引从 0 开始，第一个字符的索引为 0，第二个字符的索引为 1，以此类推。

python 字符串的索引可以取负值，表示从末尾提取，最后一个字符的索引为 -1，倒数第二个字符的索引为 -2，即程序认为可以从结束处反向计数。

简单的示例：

```
>>> string = 'hello_shiyanlou'
>>> string[0] # 获取第一个字符
'h'
>>> string[1] # 获取第二个字符
'e'
>>> string[2]
'l'
>>> string[-1] # 获取倒数第一个字符
'u'
>>> string[-2] # 获取倒数第二个字符
'o'
```

切片

切片即获取字符串的片段，格式为 [头索引:尾索引:步长]。索引又称作下标。

举例说明：

```
>>> string[0:5] # 取前 5 个字符，即下标为 0 1 2 3 4 的五个字符
'hello'
>>> string[:5] # 头索引为 0 可以省略
'hello'
>>> string[6:15] # 取第 7 至 15 个字符，即下标为 6 至 14 的九个字符
'shiyanlou'
>>> string[6:] # 取下标为 6 及其后面的全部字符
'shiyanlou'
>>>
>>> string[3:7] # 取下标为 3 4 5 6 的四个字符
'lo_s'
>>> string[3:11:2] # 取下标为 3 至 10 的字符，步长为 2 即每两个取第一个
'l_hy'
>>> string[::3] # 取全部字符，每三个取第一个
'hlsyl'
>>> string[::-1] # 这是一种特殊写法，步长为 -1，即反向取全部字符，也就是倒序排列
'uolnayihs_olleh'
>>> string[:] # 这也是一种特殊写法，取全部字符
'hello_shiyanlou'
```

字符串的常用属性和方法

count

获取字符串中某个字符的数量：

🔗 楼+之Python实战第10期 (/courses/1190)

```
>>> string
'hello_shiyanlou'
>>> string.count('s')
1
>>> string.count('l')
3
```

split 和 strip

字符串中有很多常用的方法可以使用，在 Python Shell 中可以使用 `help(str)` 查看所有的字符串中的方法，这里介绍两个常用的，并且后面的挑战作业中会用到的。

- `strip()`：默认情况下会删除字符串首尾的空格及换行等空白符。如果 `strip()` 函数中使用参数则会删除这些参数中的字符（仅限于出现在字符串首尾的情况），例如 `str1.strip('ab')` 则只会删除 `str1` 字符串中头尾部的 `a` 和 `b` 字符。
- `split()`：默认情况下会用空格将字符串中进行切分得到一个列表，传入参数的时候会用传入的参数对字符串进行切分。

上述两个函数的举例：

```
>>> str1 = ' shiyanlou '
>>>
>>> str2 = str1.strip()
>>> str2
'shiyanlou'
>>> str1
' shiyanlou '
>>>
>>>
>>> str3 = 'hello shiyanlou'
>>> list1 = str3.split()
>>> list1
['hello', 'shiyanlou']
>>> str3
'hello shiyanlou'
>>>
>>> str4 = 'hello:shiyanlou'
>>> list2 = str4.split(':')
>>> list2
['hello', 'shiyanlou']
>>> str4
'hello:shiyanlou'
>>>
```

upper 和 lower

前者将字符串中每个英文字母变成大写，后者将每个英文字母变成小写：

```
>>> string = 'LOUPLUS_PYTHON'
>>> string.upper()
'LOUPLUS_PYTHON'
>>> string.lower()
'louplus_python'
```

__len__

该方法等同于 Python 3 中的内置函数 `len()`，可以获得字符串包括的字符数量：

```
>>> 'louplus_python'.__len__()
14
>>> len('louplus_python')
14
```

单引号、双引号与三个引号

Python 3 中的字符串可以使用双引号 `""` 或者单引号 `' '` 标示，如果只出现一对引号那么无论使用单引号还是双引号都是没有差别的。

```
>>> "hello"
'hello'
>>> 'hello'
'hello'
```



如果字符串中既有单引号又有双引号，那么在最外面使用双引号，里面使用单引号。

```
>>> "Let's go!"
"Let's go!"
```

如果字符串中多次只出现单引号或是双引号，则内部的引号需要使用转义字符反斜杠 `\` 来对字符串的引号进行转义。

```
Let's go!
>>> 'Let's go!'
File "<stdin>", line 1
  'Let's go!'
    ^
SyntaxError: invalid syntax
>>> 'Let\'s go!'
"Let's go!"
>>> "she said "yes""
File "<stdin>", line 1
  "she said "yes""
    ^
SyntaxError: invalid syntax
>>> "she said \"yes\""
'she said "yes"'
>>>
```



在 Python 中也会出现注释多行的情况，这时就会使用三个单引号 `'''` 或者三个双引号 `"""` 将多行注释括起来。注释多行的情况一般出现在代码文件开头用于讲解整个文件的功能、使用方法、注意事项，或是在定义类的下面加上这个类的说明以及用法，也可以用于在一个函数说明它的使用方法或是注意点。在程序执行的过程中，注释多行中的内容也会被程序给忽略掉。

比如在 `/home/shiyanlou/dog.py` 中添加如下代码，然后执行：

```
#!/usr/bin/env python3

...
这是使用三个单引号的情况
这个文件主要用于打印 dog
它的使用方法是在命令行中执行：python3 dog.py
...
print("dog")
"""
这是使用三个双引号的情况
这个注释也是会被程序忽略掉的
"""
```

```

shiyanolou:~/ $ vim dog.py [13:29:24]
shiyanolou:~/ $ python3 dog.py [13:33:16]
dog
shiyanolou:~/ $ cat dog.py [13:33:21]
#!/usr/bin/env python3

'''
这是使用三个单引号的情况
这个文件主要是用于打印 dog
它的使用方法是在命令行中执行：python3 dog.py
'''

print("dog")

'''
这是使用三个双引号的情况
这个注释也是会被程序忽略掉的
'''
shiyanolou:~/ $ █ [13:33:28]

```



注释

注释是一段文字性的描述，在写代码的过程中随时都可以被添加，它主要用来帮助我们阅读和理解代码，良好的注释习惯是能够提高程序的可维护性，哪怕是写的很好的代码，如果没有注释，过一段时间连作者都很难去改动了。

井号(#)常被用作单行注释的符号，当程序被处理的时候，# 号后的注释内容会被忽略，不会被当作代码处理。

比如在 Python 3 交互式命令行中输入如下代码，# 后面的中文注释主要是方便自己以及别人能够快速理解该代码的作用：

```

# 这里可以写一段注释：下面是打印"hello shiyanlou"
print("hello shiyanlou")
print("hello world") # 这里也可以写一段注释：这是用于打印"hello world"

```

```

shiyanolou:~/ $ python3 [10:22:58]
Python 3.5.2 (default, Jul 17 2016, 00:00:00)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> # 打印"hello shiyanlou"
... print("hello shiyanlou")
hello shiyanlou
>>> print("hello world") #打印"hello world"
hello world
>>> █

```



另外，如果有一些代码暂时不需要被执行，这时好的解决方法不是删掉它(因为后面可能还会使用)，而是使用 # 将其注释掉，这样代码也会成为注释，它在被注释的时候不会执行，删掉 # 就被程序理解为代码可以正常运行。

比如在 Python 3 交互式命令行中输入如下代码，被注释掉的代码也是不会被执行的：


```
>>> print("hello world") #打印"hello world"
hello world
>>> # print("hello world")
...
>>>
```



除了上面的两种用法(也就是 # 之后的内容不会被程序所执行外), 还有一类特殊的注释, 这类特殊的注释应该存在于每一个你编写的 py 文件中, 而且在正常情况下应该是第一行和第二行的通用格式。程序示例如下:

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-

print("hello shiyanlou") #打印"hello shiyanlou"
```

#!/usr/bin/env python3 必须放在文件的第一行, 必须以 #! 开头, 它的作用是告诉系统可以到 /usr/bin/env 目录下寻找 python3 的解释器。凡是在文件中添加了这一句, 然后使用 chmod a+x xxx.py 给文件赋予可执行权限, 那么就可以使用命令 ./xxx.py 运行该文件。

-*- coding:utf-8 -*- 一般都放在文件的第二行, 当文件中出现中文的时候就必须使用它, 它是告诉 Python 解释器文件的编码格式应该使用 utf-8。

比如在终端中执行如下命令:

```
# 创建代码文件
touch /home/shiyanlou/comment.py
# 可以使用 vim 或 gedit 编辑代码文件输入以上的代码
vim /home/shiyanlou/comment.py
# 为代码文件增加执行权限
chmod a+x /home/shiyanlou/comment.py
# 执行代码文件
cd /home/shiyanlou
./comment.py
```

```
shiyanlou:~/ $ touch /home/shiyanlou/comment.py [11:12:41]
shiyanlou:~/ $ vim comment.py [11:12:58]
shiyanlou:~/ $ chmod a+x comment.py [11:14:47]
shiyanlou:~/ $ ./comment.py [11:15:01]
hello shiyanlou
shiyanlou:~/ $ cat comment.py [11:15:05]
#!/usr/bin/env python3
# -*- coding:utf-8 -*-

print("hello shiyanlou") #打印"hello shiyanlou"
shiyanlou:~/ $
```



练习题：以脚本的方式运行程序，打印你的名字

本节将考察 # 作为注释和特殊注释的用法，请按照题目要求完成，点击 下一步 系统将自动检测完成结果。
实验楼+之Python实战第10期 (/courses/1190)

在 /home/shiyanlou 目录下创建代码文件 commentTest.py 文件：

```
cd /home/shiyanlou
touch commentTest.py
```

在 /home/shiyanlou/commentTest.py 中完成以下需求：

- 1. 打印出你的名字(中文或是英文)
- 2. 给代码的作用进行注释
- 3. 在命令行中以 ./commentTest.py 的方式可以正常运行程序

程序完成后，点击 下一步 ，系统将自动检测完成结果。

运算符

运算符用于执行程序代码运算。例如：2+3, 其操作数是 2 和 3，而运算符则是 “+”。在计算器语言中运算符大致可以分为 6 种类型：算术运算符、连接运算符、比较(关系)运算符、位运算符、赋值运算符和逻辑运算符。运算符也是计算机里比较好理解的人类语言之一，只要稍微懂点数学的，都能看懂运算符。

除了数学运算之外，还有 and 和 or 的逻辑运算，and 表示与运算，只有两个运算值都是 True 才返回 True，而 or 表示或运算，有一个为 True 则返回 True。

现在我们开始对 Python 中的运算符进行详细的介绍。

算数运算符

运算符	名称	描述
+	加	两个对象相加
-	减	得到负数或是一个数减去另一个数
*	乘	两个数相乘或是返回一个被重复若干次的字符串
/	除	x 除以 y
%	取模	返回除法的余数
**	幂	返回 x 的 y 次幂
//	取整除	返回商的整数部分（向下取整）

在 Python 3 交互式命令行中执行如下命令：

🔗 楼+之Python实战第10期 (/courses/1190)

```
>>> a = 12
>>> b = 3
>>> c = 5
>>> a + b
15
>>> a - b
9
>>> a * b
36
>>> a / b
4.0
>>> a / c
2.4
>>> b ** c
243
>>> c ** b
125
>>> a % c
2
>>> a // c
2
>>> a // b
4
```

比较运算符

运算符	描述
==	等于：比较对象是否相等
!=	不等于：比较两个对象是否不相等
>	大于：返回 x 是否大于 y
<	小于：返回 x 是否小于 y
>=	大于等于：返回 x 是否大于等于 y
<=	小于等于：返回 x 是否小于等于 y

所有比较运算符返回 1 表示真，返回 0 表示假。这分别与特殊的变量 True 和 False 等价。在 Python 3 交互式命令行中执行如下命令：

🔍 楼之Python实战第10期 (/courses/1190)

```
>>> b = 2
>>> c = 3
>>> d = 2
>>> a == b
False
>>> a != b
True
>>> a > b
False
>>> a < b
True
>>> c >= d
True
>>> b <= c
True
```

赋值运算符

运算符	描述	实例
=	简单的赋值运算符	c = a + b 将 a + b 的运算结果赋值为 c
+=	加法赋值运算符	c += a 等效于 c = c + a
-=	减法赋值运算符	c -= a 等效于 c = c - a
*=	乘法赋值运算符	c *= a 等效于 c = c * a
/=	除法赋值运算符	c /= a 等效于 c = c / a
%=	取模赋值运算符	c %= a 等效于 c = c % a
**=	幂赋值运算符	c **= a 等效于 c = c ** a
//=	取整除赋值运算符	c //= a 等效于 c = c // a

在 Python 3 交互式命令行中执行如下命令：


实验楼之Python实战第10期 (/courses/1190)

```
>>> b = 12
>>> c = 10
>>> c += a
>>> print(c)
13
>>> b -= a
>>> b
9
>>> c *= 5
>>> c
65
>>> c %= a
>>> c
2
>>> c **= 8
>>> c
256
>>> c //= 10
>>> c
25
```

位运算符（选学）

运算符	描述
&	按位与运算符：参与运算的两个值, 如果两个相应位都为 1, 则该位的结果为 1, 否则为 0
\	按位或运算符：只要对应的二个二进位有一个为 1 时，结果位就为 1
^	按位异或运算符：当两对应的二进位相异时，结果为 1
~	按位取反运算符：取原数值的二进制值的负数，再减 1
<<	左移动运算符：运算数的各二进位全部左移若干位，由 "<<" 右边的数指定移动的位数，高位丢弃，低位补 0
>>	右移动运算符：把 ">>" 左边的运算数的各二进位全部右移若干位，">>" 右边的数指定移动的位数

在 Python 3 交互式命令行中执行如下命令：



楼之Python实战第10期 (/courses/1190)

```
>>> a = 10 # 0000 1010
>>> b = 25 # 0001 1001
>>> a & b # 0000 1000
>>> 8
>>> a | b # 0001 1011
>>> 27
>>> a ^ b # 0001 0011
>>> 19
>>> ~a # -0000 1011
>>> -11
>>> a>>1 # 0000 0101
>>> 5
>>> a<<2 # 0010 1000
>>> 40
```

逻辑运算符

运算符	逻辑表达式	描述
and	x and y	布尔 "与" - 如果 x 为 False , x and y 返回 False , 否则它返回 y 的计算值
or	x or y	布尔 "或" - 如果 x 是非 0 , 它返回 x 的值 , 否则它返回 y 的计算值
not	not x	布尔 "非" - 如果 x 为 True , 返回 False 。 如果 x 为 False , 它返回 True

在 Python 3 交互式命令行中执行如下命令：

```
>>> a = 1
>>> b = 2
>>> c = 0
>>> a and c
0
>>> a and b
2
>>> a or c
1
>>> a or b
1
>>> b or a
2
>>> not c
True
>>> not a
False
```

成员运算符

除了以上的一些运算符之外，Python 还支持成员运算符：

🔗 楼+之Python实战第10期 (/courses/1190)

运算符	描述
in	如果在指定的序列中找到值返回 True，否则返回 False
not in	如果在指定的序列中没有找到值返回 True，否则返回 False

在 Python 3 交互式命令行中执行如下命令：

```
>>> l = ['a', 'b', 'c']
>>> if 'a' in l:
...     print('字符 "a" 在列表中')
... else:
...     print('字符 "a" 不在列表中')
...
字符 "a" 在列表中
>>> if 'd' in l:
...     print('字符 "d" 在列表中')
... else:
...     print('字符 "d" 不在列表中')
...
字符 "d" 不在列表中
```

身份运算符

先介绍一下 Python 内置的 id 方法：

在 Python 语言中，一切皆对象。当我们创建一个 Python 对象，就会在内存中分配一段内存用来存储这个对象，这段内存会有一个十进制的编号，id 方法就用来获得这个内存地址的编号，举例如下：

```
>>> id('hello shiyanlou')
4463109104
>>> l = ['a', 'b', 'c']
>>> id(l)
4462852040
```

身份运算符就是用来判断两个变量所指向的对象是否存储在同一个内存单元中：

运算符	描述	实例
is	is 是判断两个标识符是不是引用自一个对象	x is y, 类似 id(x) == id(y)，如果引用的是同一个对象则返回 True，否则返回 False

运算符	描述	实例
is not	is not 是判断两个标识符是不是引用自不同对象	x is not y ，类似 id(a) != id(b)。 如果引用的不是同一个对象则返回结果 True，否则返回 False

在 Python 3 交互式命令行中执行如下命令：

```
>>> a = 'hello'
>>> a is not 'hellllo'
True
>>> a is 'hello'
True
>>> c = 33
>>> d = 33
>>> c is d
True
>>> c is not d
False
>>> c is not a
True
```

运算符优先级

以下表格列出了从最高到最低优先级的所有运算符：

运算符	描述
**	指数 (最高优先级)
~ + -	按位翻转, 一元加号和减号 (最后两个的方法名为 +@ 和 -@)
* / % //	乘，除，取模和取整除
+ -	加法减法
>> <<	右移，左移运算符
&	位'AND'
^	位运算符
<= <> >=	比较运算符
<> == !=	等于运算符
= %= /= //= -= += *= **=	赋值运算符
is is not	身份运算符

运算符	描述
in not in	成员运算符
not and or	逻辑运算符

举两个常用的例子，在 Python 3 交互式命令行中执行如下命令：

```
>>> (2 + 3) * 5 # 先计算 2 + 3，将其结果乘以 5
25
>>> a = 3.14
>>> b = 10
>>> c = 2
>>> a**2 * (b / c) # 先计算 a**2 和 b / c，再计算两者的乘积
49.298
```

条件判断与循环

我们本节进入到 Python 3 程序的控制结构，包括两部分：条件判断和循环控制。

条件判断

非常多的编程语言都会使用 `if` 关键字作为流程控制，除此之外，Python 3 的流程控制还包括 `elif` 和 `else` 两个关键字，这两个在选择控制中都是可选的。`elif` 的意思是 `else if`，增加进一步的判断是否选择该路径。

条件控制的基本语法如下：

```
if 判断条件1:    # 当判断条件1为真时，执行语句1，否则为假就继续下一个判断条件
    执行语句1
elif 判断条件2:
    执行语句2
elif 判断条件3:
    执行语句3
else:
    执行语句4
```

注意：

- 条件语句后面需要使用冒号 `:`，表示下面的代码块是满足条件后需要执行的
- 使用缩进的距离来区分不同的语句块，具有相同缩进距离的是同一组语句块，同一组语句块将会按顺序依次被执行
- 在布尔表达式中，会被 Python 解释器判断为“假”的有：`False`、`None`、`0`、`""`、`()`、`[]`、`{}`
- `else` 后面不能加判断语句

举例说明，下面的代码：

🔗 楼+之Python实战第10期 (/courses/1190)

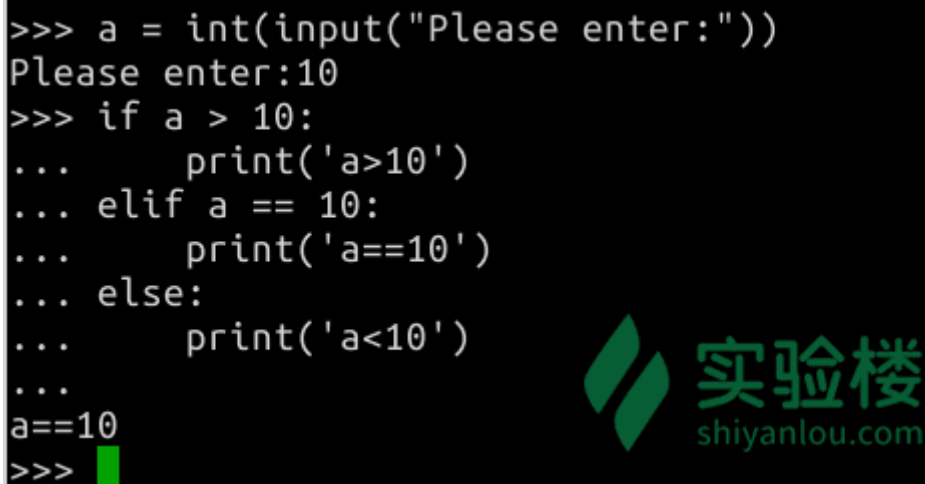
```
>>> a = int(input("Please enter: "))
Please enter: 10
>>> if a > 10:
...     print('a > 10')
... elif a == 10:
...     print('a == 10')
... else:
...     print('a < 10')
```

`input("Please enter: ")` 这句代码是使用 `input` 函数获取用户输入，`input` 中的参数字符串将输出到屏幕上，用户输入的内容会被函数返回，返回的值为字符串。如果不输入，程序将始终阻塞等待。

`int(input("Please enter: "))` 将用户的输入的字符串转成整数，并把数字赋值给变量 `a`。

这个例子中会根据输入的 `a` 的值不同选择不同的路径，可以将代码写入一个脚本文件中并重复执行尝试不同的输入。

程序执行的截图：



```
>>> a = int(input("Please enter:"))
Please enter:10
>>> if a > 10:
...     print('a>10')
... elif a == 10:
...     print('a==10')
... else:
...     print('a<10')
...
a==10
>>>
```

可以在一个 `if` 判断中嵌套另外一个 `if` 判断，基本语法如下：

```
if 表达式1:
    if 表达式a:
        执行语句a
    elif 表达式b:
        执行语句b
    else:
        执行语句c
else:
    执行语句2
```

`if` 判断中常用的操作运算符有：

- 小于：<

- 小于或等于：`<=`
- 大于：`>`
- 大于或等于：`>=`
- 等于：`==`
- 不等于：`!=`

判断语句中也可以使用 `and` 和 `or` 来对符合的条件进行控制。如下所示：

```
if A and B:    # A和B必须同时满足才能执行语句1
    执行语句1
elif C or D:   # C和D只要满足其中一个就可以执行语句2
    执行语句2
```

在程序开发过程中，当遇到一些代码暂时不写(等到后面写)又不想程序在执行的时候报错就可以使用 `pass` 关键字，程序执行遇到 `pass` 就会跳过这里的代码块继续执行后面的代码：

```
>>> a = 3
>>> if a<1:
...     print("a<1")
... else:
...     pass
...
>>> #程序没有报错
```

练习题：年龄区间判断

本节将考察之前学习的条件控制知识点，请按照题目要求完成，点击 [下一步](#) 系统将自动检测完成结果。

在 `/home/shiyanlou` 目录下创建代码文件 `ageRange.py`：

```
cd /home/shiyanlou
touch ageRange.py
```

需要在 `/home/shiyanlou/ageRange.py` 中完成以下需求：

1. 使用 `python3 ageRange.py 年龄` 的方式执行程序
2. 使用 `sys.argv[1]` 获取用户的年龄
3. 使用 `and` 进行两个条件语句的连接
4. 具体的年龄判断如下：

提示：需要在开头使用 `import sys` 引入模块

```
# 楼之Python实战第10期 (/courses/1190)
[0,10): 打印"you belong to kids"
[10,18): 打印"you belong to teenager"
[18,30): 打印"you belong to adult"
[30,60): 打印"you belong to older"
[60,120): 打印"you belong to oldest"
```

程序运行示例结果如下：

```
shiyancelou:~/ $ python3 ageRange.py 8
你属于儿童
shiyancelou:~/ $ python3 ageRange.py 46
你属于中年
shiyancelou:~/ $ python3 ageRange.py 75
你属于老年
shiyancelou:~/ $
```



程序完成后，点击 下一步 ，系统将自动检测完成结果。

循环控制

Python 中包含两种循环方式，一种是 for，一种是 while。

for 循环

for 循环主要用在依次取出一个列表中的项目，对列表进行遍历处理。

代码示例如下：

```
strlist = ['hello','shiyancelou','.com']
for s in strlist:
    print(s)
```

```
>>> strlist = ['hello', 'shiyancelou', '.com']
>>> for s in strlist:
...     print(s)
...
hello
shiyancelou
.com
>>>
```



如果需要迭代一组数字列表，并且数字列表满足一定的规律，可以使用内置函数 range()：

楼之Python实战第10期 (/courses/1190)
for a in range(10):
 print(a)

```
>>> for a in range(10):  
...     print(a)  
...  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
>>>
```



range() 函数还有很多不同的使用方法，感兴趣可以查看 help 帮助文档。

while 循环

另外一种循环是 while，while 不同于 for 是使用一个表达式作为判断的条件，如果条件不能够达成则停止循环。

```
w = 100  
while w > 10:  
    print(w)  
    w -= 10
```

这里要注意 `w -= 10`，等同于 `w = w - 10`。当 `w` 的值小于等于 10 的时候，循环退出。

```
>>> w = 100
>>> while w > 10:
...     print(w)
...     w -= 10
...
100
90
80
70
60
50
40
30
20
>>>
>>>
```



我们在循环控制中，可以使用 `break` 和 `continue` 两个关键字，`break` 表示停止当前循环，`continue` 表示跳过当前循环轮次中后续的代码，去执行下一循环轮次。

代码示例：

```
for a in range(10):
    if a == 5:
        break
    print(a)
```

执行如下图，当 `a` 为 5 的时候循环退出：

```
>>> for a in range(10):
...     if a == 5:
...         break
...     print(a)
...
0
1
2
3
4
>>>
```



```
w = 100
while w > 10:
    w -= 10
    if w == 50:
        continue
    print(w)
```

执行如下图，当 w 为 50 的时候不执行后续的 print 代码：

🔔 楼+之Python实战第10期 (/courses/1190)

```
>>> w = 100
>>> while w > 10:
...     w -= 10
...     if w == 50:
...         continue
...     print(w)
...
90
80
70
60
40
30
20
10
>>>
```



练习题：使用 while 循环计算 1 到 100 的总和

本节将考察循环中 while 的使用，请按照题目要求完成，点击 [下一步](#) 系统将自动检测完成结果。

在 /home/shiyanlou 目录下新建 calculation.py 文件：

```
cd /home/shiyanlou
touch calculation.py
```

在 /home/shiyanlou/calculation.py 中完成以下需求：

1. 使用 while 循环计算 1 到 100 的总和
2. 最后打印 " 1 到 100 之和为:(这里是计算出的值) "

程序运行示例效果如下：

```
shiyanlou:~/ $ python3 calculation.py
1 到 100 之和为: 5050
shiyanlou:~/ $
```

程序完成后，点击 [下一步](#)，系统将自动检测完成结果。

终端运行 Python 程序

终端运行 Python 文件时，可以添加参数，Python 内置的 `sys` 模块可以获取终端命令行参数，而 `sys.argv` 属性中的 `__name__` 属性的变化需要特别注意，下面我们来对这两点进行说明。

命令行参数

命令行参数获取方法是使用 `sys` 模块的 `sys.argv`，其中 `sys.argv[0]` 为脚本名称，`sys.argv[1]` 为第一个参数，在文件 `/home/shiyanlou/argtest.py` 中写入：

```
#!/usr/bin/env python3

import sys
print(len(sys.argv))

for arg in sys.argv:
    print(arg)
```

执行这段代码的过程：

```
$ python3 argtest.py hello shiyanlou
3
argtest.py
hello
shiyanlou
```

第一个数字表示命令行参数的数量，后面循环打印每一个参数，第一个为 `sys.argv[0]` 就是运行的程序文件，从 `sys.argv[1]` 开始才是程序的参数。

练习题：打印符合条件的命令行参数

本节将考察之前学习的命令行参数、逻辑判断、循环、字符串相关知识点，请按照题目要求完成，点击 [下一步](#) 系统将自动检测结果。

在 `/home/shiyanlou` 目录下创建代码文件 `newargtest.py`：

```
$ cd /home/shiyanlou/
$ touch newargtest.py
```

在这个文件中，我们需要实现以下需求：

1. 执行程序可以输入多个命令行参数，程序将打印输出长度超过3个字符的参数

例如：


```
$ 楼+Python实战第10期 (/courses/1190)
$ python3 newargtest.py shiyan hi louplus py 123
shiyan
louplus
```

注意输出的内容不包含程序的名称，即应该从第一个参数开始处理，所以循环中使用的是 `sys.argv[1:]`，如果你忘记这个冒号的意思，可以回顾下字符串中的内容。

程序完成后，点击 下一步，系统将自动检测完成结果。

__name__ 与 __main__

在 Python 中一个 .py 文件就是一个模块，每一个模块都有一个内置变量就是 `__name__`。

如果想要查看 Python 更多的内置系统变量，可以在交互式命令行中输入 `dir(__builtins__)`：

```
Python 3.5.2 (default, Jul 17 2016, 00:00:00)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'ZeroDivisionError', '_build_class_', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
>>>
```

比如在 `/home/shiyanlou` 目录下有两个文件，它们之间的目录结构和名称如下所示：

```
|_ louplus.py
|_ courses.py
```

`courses.py` 文件中的代码如下：

🔗 楼+之Python实战第10期 (/courses/1190)

```
#!/usr/bin/env python3

print('此时 __name__ 的值是: {}'.format(__name__))
```

运行代码，结果如下：

```
shiyancelou:~/ $ python3 courses.py
此时 __name__ 的值是: __main__
shiyancelou:~/ $
```



由此可以看出如果单独运行一个 `.py` 文件，它的 `__name__` 的值就是 `__main__`。

下面尝试一下如果是把一个 `.py` 文件，当作一个模块导入，它的 `__name__` 值会发生变化吗？在交互式命令行中输入如下代码：

```
import courses
```

```
shiyancelou:~/ $ python3
Python 3.5.2 (default, Jul 17 2016, 00:00:00)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import courses
此时 __name__ 的值是: courses
>>>
```



可以看到如果把一个 `.py` 文件当作一个模块导入，它的 `__name__` 的值就是这个文件去掉 `.py` 后的文件名(也就是这个模块的名字)，在这里就是 `courses`。

通常情况下，一个 `xxx.py` 文件只有两种使用情况，要么是使用 `python3 xxx.py` 的方式被单独执行，要么是使用 `import xxx` 的方式被其它文件当作一个模块所引用。而且以模块的方式导入存在一个问题，被导入的模块中的代码会自上而下依次运行（参考上面的那张图），有的时候我们不想文件中的某些代码在导入的时候被执行，怎么办呢？

这个时候 `__name__` 的这个重要特性就发挥用场了，可以在文件中单独加一个判断进行控制，当 `__name__ == '__main__'` 时才执行特定的代码。

现在修改 `courses.py` 文件中的代码如下：

```
#!/usr/bin/env python3

print('此时 __name__ 的值是: {}'.format(__name__))

if __name__ == '__main__':
    print("shiyancelou has many courses.") # 当以单独的文件运行时才打印
```

运行结果如下：

楼+之Python实战第10期 (/courses/1190)

```
shiyancelou:~/ $ python3 courses.py
此时 __name__ 的值是: __main__
shiyancelou has many courses.
shiyancelou:~/ $
```



在 loupplus.py 文件中导入 courses 模块：

```
#!/usr/bin/env python3

import courses
print("this is loupplus.py file")
```

```
shiyancelou:~/ $ python3 loupplus.py
此时 __name__ 的值是: courses
this is loupplus.py file
shiyancelou:~/ $
```



像这样就实现了对程序执行的选择性控制，当以单独的文件运行时 if 判断中的代码才会被执行。这也是 `__name__` 在程序开发过程中的主要应用场景。大部分程序都会使用到这个特性，大家需要多多注意。

练习题：应用 `__name__`

本节将考察 `__name__` 的用法，请按照题目要求完成，点击 [下一步](#) 系统将自动检测结果。

在 `/home/shiyanlou` 目录下新建 `choose.py` 文件：


```
cd /home/shiyanlou
touch choose.py
```

在 `choose.py` 文件中写入代码，要求完成以下的需求：

1. 当 `choose.py` 以文件的形式单独运行时打印，分两行分别打印 "hello,world" 和 "hello,shiyanlou"
2. 当 `choose.py` 文件以模块的形式被导入时，只打印 "hello,world"

运行示例如下：

```
shiyancelou:~/ $ python3 choose.py
hello,world
hello,shiyancelou
shiyancelou:~/ $ python3
Python 3.5.2 (default, Jul 17 2016, 00:00:00)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import choose
hello,world
>>>
```



程序完成后，点击 下一步 ，系统将自动检测完成结果。

模块

在程序的开发过程中，随着时间的推移，实现的功能越多，相对应代码量也会越来越多，如果把所有的代码写入一个文件中，那么肯定是没有办法进行维护和修改的，而且也不方便再次复用。因此就出现了把代码根据实现的功能不同进行拆分放在不同的文件夹中，采用松耦合的方式写代码，这样一方面方便对代码进行修改和维护，另外一方面也方便对代码进行复用，提高开发效率。这些运行良好且实现了某些功能的代码就可以被打包放在互联网上，供大家下载使用。

Python 的一个优势就是拥有丰富的库，库是具有相关功能模块的集合，也就是常提到的标准库、第三方库以及自定义模块，在写代码的过程中会经常被引用到。

引入模块

在 Python 中，一个 .py 文件就被称为一个模块(Module)。

比如：一个 loupplus.py 的文件就是一个名为 loupplus 的模块，一个 courses.py 的文件就是一个名为 courses 的模块。通过在 loupplus.py 文件中导入 courses 模块就可以复用其中的类、函数和变量等。

现在有两个文件，它们之间的目录结构和名称如下所示：

```
|_ loupplus.py
|_ courses.py
```

courses.py 中的代码如下：

楼+之Python实战第10期 (/courses/1190)

```
# 定义了一个 Python 函数
def python():
    print("i love python")

# 定义了一个 java 函数
def java():
    print("i love java")
```

如果想要在 `louplus.py` 文件中使用 `courses.py` 文件中的所有函数，就可以使用 `import courses` 引入模块，这个等价于 `from courses import *`，这时如果想要使用 `courses` 模块中的 `python` 函数，通过 `courses.python` 就可以使用 `python` 函数。

对于有些文件拥有大量的类或是函数，而我们只需要使用到其中一两个函数，就不需要使用上面的导入方式，推荐使用局部导入。比如只想要使用 `courses.py` 文件中的 `java` 函数，可以通过 `from courses import java` 的方式导入模块，使用的时候就不再需要前缀模块名而是直接通过函数名 `java` 就可以使用 `java` 函数。

另外，当模块名比较长的时候或是避免模块名冲突的情况下，也可以对导入的模块指定别名，比如：`import courses as cs`，后面使用的时候通过 `cs.java` 就可以使用 `java` 函数。

推荐引入顺序

在当前代码中引入模块，推荐的引入顺序为：标准库 > 第三方 > 自定义。

也就是说：优先引用 Python 内置的模块，如果内置的模块没有需要的功能，再去查看第三方是否有现成的模块可以引用，如果依然没有才自定义模块进行使用。这样做主要是为了节省开发时间以及提高性能。

模块搜索路径

在 Python 中可以使用 `import xxx` 或 `from xxx import yyy` 这样的形式来引入某个模块或者模块中的某个函数、类等内容到当前的代码文件中。那么 Python 怎么知道去哪里找这些模块呢？

Python 中存在一个默认的模块搜索路径。在当前代码文件中导入一个模块时，Python 解释器先在当前包中查找模块，如果找不到就会在内置模块中查找，如果依然找不到就会按 `sys.path` 给定的路径查找对应的模块文件。`sys.path` 包括当前目录及系统中的一些 Python 模块的主要安装目录，可以通过下面的方法查看搜索路径：

```
>>> import sys
>>> sys.path
```



```
shiyancelou:~/ $ python3 [9:56:54]
Python 3.5.2 (default, Jul 17 2016, 00:00:00)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.path
['', '/usr/lib/python3.5', '/usr/lib/python3.5/plat-x86_64-linux-gnu', '/usr/lib
/python3.5/lib-dynload', '/usr/local/lib/python3.5/dist-packages', '/usr/lib/pyt
hon3/dist-packages']
>>>
```

模块与包

包是一个文件夹，在其中可以定义多个模块或是多个子包。通常 Python 的第三方工具或是应用都是以包的形式发布的。

在 Python 中文件夹可以被识别成一个包，前提是这个文件夹中有一个 `__init__.py` 文件（注意从 Python 3.3 之后不再需要这个文件了），文件中可以不用写任何内容。


比如有下面这样的文件结构，它的存放路径是 `/home/shiyancelou`，注意，是该目录下还有一个 `shiyancelou` 目录，以下目录结构需要自己手动创建：

```
shiyancelou
├── __init__.py
├── loupplus.py
├── courses.py
└── users
    ├── __init__.py
    └── totalnums.py
```

因为 `shiyancelou` 目录下有 `__init__.py` 文件，所以它可以被 Python 识别为一个包（从 Python 3.3 开始，就不再需要目录下必须有 `__init__.py` 文件了）。

如果想要在 `/home/shiyancelou/Code` 目录下引入 `courses` 模块就可以用 `import shiyancelou.courses` 这种代码来引入，前提是 `shiyancelou` 目录已经放到了 Python 模块搜索的默认路径下了，可以通过 `sys.path.append(yourModulePath)`。

```
shiyanolou:Code/ $ python3 [10:47:11]
Python 3.5.2 (default, Jul 17 2016, 00:00:00)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from shiyanolou import courses
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named 'shiyanolou'
>>> import sys
>>> sys.path.append('/home/shiyanolou')
>>> sys.path
['', '/usr/lib/python3.5', '/usr/lib/python3.5/plat-x86_64-linux-gnu', '/usr/lib/python3.5/lib-dynload', '/usr/local/lib/python3.5/dist-packages', '/usr/lib/python3/dist-packages', '/home/shiyanolou']
>>> from shiyanolou import courses
>>> import shiyanolou.courses
>>>
```




在 Python 包内部可以使用相对路径的方式来简化相对层级中包内模块的相互引用。比如在 shiyanlou 包的内部，如果想要在 loupplus.py 文件中引入 courses 模块，可以使用 `import courses` 或是 `from . import courses`。如果想要单独运行 loupplus.py 文件查看效果就需要在 /home/shiyanlou 目录下执行 `python3 -m shiyanolou.loupplus`。

Python 的 `-m` 参数用于将一个模块或包当作一个脚本运行。

而如果想要在 totalnums.py 文件中引入 courses.py 文件中的 java 函数，可以使用 `from ..courses import java`。

```
shiyanolou:~/ $ python3 -m shiyanolou.loupplus
i love java
shiyanolou:~/ $ python3 -m shiyanolou.users.totalnums
i love java
```



- 拓展阅读 《Python 模块和包的区别》 (<https://liam0205.me/2017/07/23/modules-and-packages-of-python/>)
- 拓展阅读 《Python 导入模块的几种姿势》 (<http://codingpy.com/article/python-import-101/>)

练习题：根据需求创建包和模块

本节将考察之前学习的模块和包的知识点，请按照题目要求完成，点击 [下一步](#) 系统将自动检测完成结果。

在 /home/shiyanlou 目录下创建代码文件 moduletest.py：

```
cd /home/shiyanlou/
touch moduletest.py
```

Linux 基础命令的提示：

1. `cd` 命令是 Linux 系统中切换目录的操作命令，`cd /home/shiyanlou/` 表示切换到 `/home/shiyanlou` 目录。
楼+之Python实战第10期 (/courses/1190)
2. `touch` 可以用来创建一个空文件，`touch moduletest.py` 表示创建一个空文件，文件名为 `moduletest.py`
3. 解决该题目还需要用到的命令 `mkdir`，该命令为 Linux 上用来创建目录的命令，例如在 `/home/shiyanlou` 下创建一个新的 `loupplus` 目录，可以使用下面的操作：

```
cd /home/shiyanlou
mkdir loupplus
```

在这个文件中，我们希望引入模块 `hello` 中的变量 `message`，然后打印出来：

```
#!/usr/bin/env python3

from loupplus.test.hello import message

print(message)
```

需要在 `/home/shiyanlou/` 目录下创建相应的包以及模块，保证用户可以正常运行并输出以下内容：

```
$ cd /home/shiyanlou
$ python3 moduletest.py
hello shiyanlou
```

程序完成后，点击 下一步，系统将自动检测完成结果。

异常处理

异常处理是工作中编写代码必须要完成的内容，对于不符合预期的用户操作或数据输入，程序总会出现异常情况，而对异常情况能够妥善处理，是保证程序稳定性的关键工作之一。

异常出现的原因非常多，逻辑错误，用户输入错误都会造成异常。

举个例子，告诉我们什么是异常：

```
filename = input("Enter file path:")
f = open(filename)
print(f.read())
```

这个简单的程序中我们会用到后续章节中将详细介绍的文件操作，`open()` 函数打开文件，`read()` 函数读取文件内容。

首先 `input()` 函数会读取用户的输入作为文件的路径，如果用户输入的文件不存在会怎么样呢？


```
shiyancelou:~/ $ python3 [17:31:49]
Python 3.5.2 (default, Jul 17 2016, 00:00:00)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
>>> filename = input("Enter file path:")
Enter file path:/home/shiyancelou/nothing.txt
>>> f = open(filename)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: '/home/shiyancelou/nothing.txt'
>>>
```

会出现文件不存在的异常，并且会发现 Traceback，这就是系统抛出的异常，异常的类型是 FileNotFoundError。

Python 常用的异常类有很多，我们不需要去记住，只需要在收到异常的时候能通过查询文档了解含义。这里介绍几个最常见的异常类：

- NameError 访问一个未定义的变量
- SyntaxError 语法错误，这个严格讲算是程序的错误
- IndexError 对于一个序列，访问的索引超过了序列的范围（序列的概念会在后续实验中讲到），可以理解为我的序列里只有三个元素，但要访问第4个
- KeyError 访问一个不存在的字典 Key，字典也会在下一节实验中详细讲到，Key 如果不存在字典就会抛出这个异常
- ValueError 传入无效的参数
- AttributeError 访问类对象中不存在的属性

异常处理方式

如果出现了异常，我们不可以直接将异常抛给用户，应该使用 Python 提供的异常处理方法来捕获并处理异常，处理方法为使用 try，except 和 finally 三个关键字。

其中我们把可能出现异常的代码放到 try 代码块，然后在 except 代码块中添加处理异常的方法，回到刚才的文件读取，我们将 open 和 read 放到 try 代码块中，在 except 中处理异常。

代码格式如下：

```
try:
    有可能抛出异常的代码
except 异常类型名称:
    处理代码
except 异常类型名称:
    处理代码
```

这里需要注意的是 except 可以有多个，每个处理不同类型的异常，也可以不写任何异常类型名称，则会处理所有捕获的异常。

有的时候如果我们需要获得异常的具体信息的话，也会写成 `except FileNotFoundError as err:` 这种形式，`err` 为异常对象，我们可以通过这个对象获取更详细的异常信息。

改进的文件读取程序为：

```
filename = input("Enter file path:")
try:
    f = open(filename)
    print(f.read())
    f.close()
except FileNotFoundError as err:
    print("Error: {}".format(err))
```

当 `try` 代码块中一旦出现异常，这个代码块后续的代码不会继续执行，会直接进入 `except` 异常处理代码块中。

我们把这个程序写到 `/home/shiyanlou/fileexc.py` 中，然后执行，并输入上面例子中的不存在的文件，这个时候会被 `except` 捕获并处理。

```
shiyanlou:~/ $ vim fileexc.py
shiyanlou:~/ $
shiyanlou:~/ $ python3 fileexc.py
Enter file path:/home/shiyanlou/nothing.txt
File not found
shiyanlou:~/ $
shiyanlou:~/ $ cat fileexc.py
#!/usr/bin/env python3

filename = input("Enter file path:")
try:
    f = open(filename)
    print(f.read())
    f.close()
except FileNotFoundError:
    print("File not found")
shiyanlou:~/ $
```



`finally` 关键字是用来进行清理工作，经常和 `except` 一起使用，即无论是正常还是异常，这段代码都会执行。

如果一个文件处理的程序中异常出现在 `f.write()` 向文件中写入数据的时候，就无法执行 `close` 操作，使用 `finally` 可以保证无论 `try` 代码块中的代码是否抛出异常，都能够执行 `finally` 代码块里的内容，保证文件被正常关闭。

修改上述的程序如下，改为写入操作，引入 `finally` 保证文件可以被正常关闭：

```
filename = '/etc/passwd'
f = open(filename)
try:
    f.write('shianlou')
except:
    print("File write error")
finally:
    print("finally")
    f.close()
```

程序运行的结果：

```
File write error
finally
```

表示虽然异常，但仍然执行到了 finally 代码块。

这里需要说明下抛出异常的原因是以只读的模式打开了一个文件，但尝试向文件中写入内容，所以会抛出异常。另外 except: 这个语句后不写任何参数，表示将处理所有 try 代码块中抛出的异常。

抛出异常

如果我們希望在程序执行过程中抛出一些异常，该如何操作呢？可以使用 raise 语句。

```
raise 异常名称
```

例如，我们在代码里希望抛出一个 ValueError，直接使用：

```
raise ValueError()
```

外部的代码就可以使用 except ValueError 进行捕获和处理了。

练习题：为程序补充异常处理代码

本节将考察之前学习的异常处理知识点，请按照题目要求完成，点击 下一步 系统将自动检测完成结果。

在 /home/shianlou 目录下创建代码文件 extest.py：

```
$ cd /home/shianlou/
$ touch extest.py
```

不包含异常处理的程序如下，需要你补充异常处理代码：

🔗 楼+之Python实战第10期 (/courses/1190)

```
num = input("Enter number:")
new_num = int(num)
print('INT:{}'.format(new_num))
```

在 `/home/shiyanlou/extest.py` 中完成以下需求：

1. 使用 `input` 获取用户输入一个字符串
2. 使用 `int()` 函数将字符串转为整数
3. 例如用户输入字符串为 "123" 就可以转为整数，然后打印 `INT:123`
4. 例如用户输入字符串为 "abc" 就会出现异常，需要处理异常并打印错误信息 "ERROR:abc"

程序完成后，点击 下一步 ，系统将自动检测完成结果。

Python 包管理工具

Python 拥有大量的第三方包，如果下载它们都需要通过源码下载岂不是很不方便？所以 Python 提供了包管理工具 `pip`，通过这个工具可以实现对 Python 主流的第三方模块的下载、安装、卸载等功能，它就像手机上的“应用市场”一样。

`pip` 的官方网站是 PyPi (<https://pypi.org/project/pip/>)，一般安装 Python 的时候就会自动安装 `pip` 工具。由于 Python 分为两个版本，因此 `pip` 也分为两个版本，一般 `pip` 对应的是 Python 2.x，`pip3` 对应的是 Python 3.x。

`pip` 的常见命令如下：(也可以在 `pip` 前加上 `sudo`，获取 root 权限)

- 显示版本和路径：`pip3 --version`
- 升级 `pip`：`sudo pip3 install --upgrade pip`
- 安装包：`(sudo) pip3 install package`，如果需要指定版本就是：`pip3 install package==1.0.3` (写具体的版本号)
- 卸载包：`pip3 uninstall package`
- 升级包：`pip3 install --upgrade package`，可以使用 `==, >=, <=, <, >` 来指定版本号
- 查看安装已安装的包：`pip3 list`
- 把需要安装的一系列包写入 `requirements.txt` 文件中，然后执行：`pip3 install -r requirements.txt`

练习题：安装指定包

本节将考察之前学习的模块和包的知识点，请按照题目要求完成，点击 下一步 系统将自动检测完成结果。

请在环境中使用 `pip3` 安装 `xlrd`，如果权限不够可以在命令前加上 `sudo`。

程序完成后，点击 下一步，系统将自动检测完成结果。

🔒 楼+之Python实战第10期 (/courses/1190)

总结

本节实验没有需要提交到 Github 代码仓库中的代码，如果你觉得有哪些代码需要保存，可以自行提交。后续较大的示例代码、项目实验及挑战的代码我们都会要求你提交到自己的 Github 中保存。

这是楼+实验的第一节，内容比较多，也非常基础。虽然无法把 Python 语法讲解的面面俱到，但已经包含了最常用的内容：

1. Python 开发环境
2. 变量与数据类型
3. 字符串
4. 运算符
5. 条件判断与循环
6. 终端运行 Python 程序
7. 模块
8. 异常处理
9. Python 包管理工具

请把文档中所有的示例代码都输入一遍，尽可能不要使用复制粘贴，只有这样才能够更加熟悉代码，遇到问题的话仔细对照文档，也可以到QQ群或讨论区寻求帮助和交流。Python 3 语言基础语法并不难，难的是坚持写程序和遇到问题不含糊求根问底的学习态度。

**本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播。*

上一节：楼+ 课程介绍及Python在实验楼中的应用 (/courses/1190/labs/8517/document)

下一节：挑战：实现个税计算器 (/courses/1190/labs/8519/document)