Lecture 4 第 4 讲

Libraries 图书馆

- Generally, libraries are bits of code written by you or others that you can use in your program.
  通常，库是您或其他人编写的可在程序中使用的代码。
- Python allows you to share functions or features with others as "modules".
  Python 允许您将函数或特性作为 "模块" 与他人共享。
- If you copy and paste code from an old project, chances are you can create such a module or library that you could bring into your new project.
  如果您从旧项目中复制并粘贴代码，则很有可能可以创建这样一个模块或库，并将其引入到新项目中。

Random 随机

- `random` is a library that comes with Python that you could import into your own project.
  `random` 是 Python 附带的库，您可以将其导入到自己的项目中。
- It's easier as a coder to stand on the shoulders of prior coders.
  作为编码员，站在前任编码员的肩膀上更容易。
- So, how do you load a module into your own program? You can use the word `import` in your program.
  那么，如何将模块加载到自己的程序中呢？您可以在程序中使用单词 `import`。
- Inside the `random` module, there is a built-in function called `random.choice(seq)`. `random` is the module you are importing. Inside that module, there is the `choice` function. That function takes into it a `seq` or sequence that is a list.
  在 `random` 模块中，有一个名为 `random.choice（seq）` 的内置函数。`random` 是您要导入的模块。在该模块中，有 `choice` 函数。该函数将 `seq` 或 sequence 放入其中，它是一个列表。
- In your terminal window type `code generate.py`. In your text editor, code as follows:
  在终端窗口中键入 `code generate.py`。在文本编辑器中，按如下方式编写代码：

```
import random

coin = random.choice(["heads", "tails"])
print(coin)
```

  Notice that the list within `choice` has square braces, quotes, and a comma. Since you have passed in two items, Python does the math and gives a 50% chance for heads and tails. Running your code, you will notice that this code, indeed, does function well!
  请注意，`choice` 中的列表包含方括号、引号和逗号。由于您传入了两项，因此 Python 会进行数学运算并给出 50% 的正面和反面机会。运行您的代码，您会注意到这段代码确实运行良好！
- We can improve our code. `from` allows us to be very specific about what we'd like to import. Prior, our `import` line of code is bringing the entire contents of the functions of `random`. However, what if we want to only load a small part of a module? Modify your code as follows:
  我们可以改进我们的代码。`from` 允许我们非常具体地定义我们想要导入的内容。之前，我们的 `import` 代码行带来了 `random` 函数的全部内容。但是，如果我们想只加载模块的一小部分怎么办？按如下方式修改代码：

```
from random import choice

coin = choice(["heads", "tails"])
print(coin)
```

  Notice that we now can import just the `choice` function of `random`. From that point forward, we no longer need to code `random.choice`. We can now only code `choice` alone. `choice` is loaded explicitly into our program. This saves system resources and potentially can make our code run faster!
  请注意，我们现在可以只导入 `random` 的 `choice` 函数。从那时起，我们不再需要编写 `random.choice`。我们现在只能单独编写 `code choice`。`choice` 被显式加载到我们的程序中。这样可以节省系统资源，并可能使我们的代码运行得更快！
- Moving on, consider the function `random.randint(a, b)`. This function will generate a random number between `a` and `b`. Modify your code as follows:
  继续，考虑函数 `random.randint（a, b）`。此函数将在 `a` 和 `b` 之间生成一个随机数。按如下方式修改代码：

```
import random

number = random.randint(1, 10)
print(number)
```

  Notice that our code will randomly generate a number between `1` and `10`.
  请注意，我们的代码将随机生成一个介于 `1` 和 `10` 之间的数字。
- We can introduce into our card `random.shuffle(x)` where it will shuffle a list into a random order.

我们可以在我们的卡片中引入 `random.shuffle（x），` 它会将列表随机排列成随机顺序。

```
import random

cards = ["jack", "queen", "king"]
random.shuffle(cards)
for card in cards:
    print(card)
```

Notice that `random.shuffle` will shuffle the cards in place. Unlike other functions, it will not return a value. Instead, it will take the `cards` list and shuffle them inside that list. Run your code a few times to see the code functioning.

请注意， `random.shuffle` 会将卡片洗牌到位。与其他函数不同，它不会返回值。相反，它将获取 卡片 列表并在该列表中随机排列它们。运行代码几次以查看代码正常运行。

- We now have these three ways above to generate random information.

  我们现在有上述三种方法来生成随机信息。

- You can learn more in Python's documentation of `random`.

  您可以在 Python 的 `random` 文档中了解更多信息。

Statistics 统计学

- Python comes with a built-in `statistics` library. How might we use this module?

  Python 带有一个内置的 统计 库。我们如何使用这个模块？

- `mean` is a function of this library that is quite useful. In your terminal window, type `code average.py`. In the text editor window, modify your code as follows:

  `mean` 是这个库的一个非常有用的函数。在终端窗口中，键入 `code average.py` 。在文本编辑器窗口中，按如下方式修改代码：

```
import statistics

print(statistics.mean([100, 90]))
```

Notice that we imported a different library called `statistics`. The `mean` function takes a list of values. This will print the average of these values. In your terminal window, type `python average.py`.

请注意，我们导入了一个名为 `statistics` 的不同库。 `mean` 函数采用一个值列表。这将打印这些值的平均值。在终端窗口中，键入 `python average.py` 。

- Consider the possibilities of using the `statistics` module in your own programs.

  考虑在您自己的程序中使用 `statistics` 模块的可能性。

- You can learn more in Python's documentation of `statistics`.

  您可以在 Python 的 统计 文档中了解更多信息。

Command-Line Arguments 命令行参数

- So far, we have been providing all values within the program that we have created. What if we wanted to be able to take input from the command-line? For example, rather than typing `python average.py` in the terminal, what if we wanted to be able to type `python average.py 100 90` and be able to get the average between `100` and `90`?

  到目前为止，我们一直在提供我们创建的程序中的所有价值。如果我们希望能够从命令行获取输入，该怎么办？例如，如果我们希望能够键入 `python average.py 100 90` 并能够获得 `100` 到 `90` 之间的平均值，而不是在终端中键入 `python average.py` 怎么办？

- `sys` is a module that allows us to take arguments at the command line.

  `sys` 是一个模块，它允许我们在命令行中获取参数。

- `argv` is a function within the `sys` module that allows us to learn about what the user typed in at the command line. Notice how you will see `sys.argv` utilized in the code below. In the terminal window, type `code name.py`. In the text editor, code as follows:

  `argv` 是 `sys` 模块中的一个函数，它允许我们了解用户在命令行中输入的内容。请注意你将如何看到 `sys.argv` 在下面的代码中使用。在终端窗口中，键入 `code name.py` 。在文本编辑器中，编写如下代码：

```
import sys

print("hello, my name is", sys.argv[1])
```

Notice that the program is going to look at what the user typed in the command line. Currently, if you type `python name.py David` into the terminal window, you will see `hello, my name is David`. Notice that `sys.argv[1]` is where `David` is being stored. Why is that? Well, in prior lessons, you might remember that lists start at the `0`th element. What do you think is held currently in `sys.argv[0]`? If you guessed `name.py`, you would be correct!

请注意，该程序将查看用户在命令行中键入的内容。目前，如果您在终端窗口中键入 `python name.py David` ，您将看到 `hello, 我叫 David` 。请注意， `sys.argv[1]` 是存储 `David` 的位置。为什么？好吧，在前面的课程中，你可能还记得列表从第 `0` 个元素开始。您认为 `sys.argv[0]` 中当前保存了什么？如果你猜对 了 `name.py` ，那你就对了！

- There is a small problem with our program as it stands. What if the user does not type in the name at the command line? Try it yourself. Type `python name.py` into the terminal window. An error `list index out of range` will be presented by the compiler. The reason for this is that there is nothing at `sys.argv[1]` because nothing was typed! Here's how we can protect our program from this type of error:

  我们的计划目前存在一个小问题。如果用户没有在命令行中键入名称，该怎么办？自己试试。在终端窗口中键入 `python name.py` 。编译器将提供 超出范围的错误列表索引 。这样做的原因是 `sys.argv[1]` 中没有任何内容，因为没有输入任何内容！以下是我们如何保护我们的程序免受此类错误的影响：

```
import sys

try:
    print("hello, my name is", sys.argv[1])
except IndexError:
    print("Too few arguments")
```

Notice that the user will now be prompted with a useful hint about how to make the program work if they forget to type in a name. However, could we be more defensive to ensure the user inputs the right values?

请注意，如果用户忘记键入名称，现在将提示用户如何使程序正常工作。但是，我们是否可以更加防御以确保用户输入正确的值？

- Our program can be improved as follows:

  我们的计划可以改进如下：

```
import sys

if len(sys.argv) < 2:
    print("Too few arguments")
elif len(sys.argv) > 2:
    print("Too many arguments")
else:
    print("hello, my name is", sys.argv[1])
```

  Notice how if you test your code, you will see how these exceptions are handled, providing the user with more refined advice. Even if the user types in too many or too few arguments, the user is provided clear instructions about how to fix the issue.

  请注意，如果您测试代码，您将看到这些异常是如何处理的，从而为用户提供更精确的建议。即使用户键入的参数过多或过少，也会向用户提供有关如何解决问题的明确说明。

- Right now, our code is logically correct. However, there is something very nice about keeping our error checking separate from the remainder of our code. How could we separate out our error handling? Modify your code as follows:

  现在，我们的代码在逻辑上是正确的。但是，将错误检查与代码的其余部分分开是非常好的。我们如何分离我们的错误处理呢？按如下方式修改代码：

```
import sys

if len(sys.argv) < 2:
    sys.exit("Too few arguments")
elif len(sys.argv) > 2:
    sys.exit("Too many arguments")

print("hello, my name is", sys.argv[1])
```

  Notice how we are using a built-in function of `sys` called `exit` that allows us to exit the program if an error was introduced by the user. We can rest assured now that the program will never execute the final line of code and trigger an error. Therefore, `sys.argv` provides a way by which users can introduce information from the command line. `sys.exit` provides a means by which the program can exit if an error arises.

  请注意我们如何使用 `sys` 的内置函数 `exit`，该函数允许我们在用户引入错误时退出程序。我们现在可以放心，该程序永远不会执行最后一行代码并触发错误。因此，`sys.argv` 提供了一种用户可从命令行引入信息的方法。`sys.exit` 提供了一种方法，程序可以通过该方法在出现错误时退出。

- You can learn more in Python's documentation of `sys`.

  您可以在 Python 的 `sys` 文档中了解更多信息。

`slice` 片

- `slice` is a command that allows us to take a `list` and tell the compiler where we want the compiler to consider the start of the `list` and the end of the `list`. For example, modify your code as follows:

  `slice` 是一个命令，它允许我们获取一个 `列表` 并告诉编译器我们希望编译器考虑 `列表` 的开头和 `结尾` 的位置。例如，按如下方式修改您的代码：

```
import sys

if len(sys.argv) < 2:
    sys.exit("Too few arguments")

for arg in sys.argv:
    print("hello, my name is", arg)
```

  Notice that if you type `python name.py David Carter Rongxin` into the terminal window, the compiler will output not just the intended output of the names, but also `hello, my name is name.py`. How then could we ensure that the compiler ignores the first element of the list where `name.py` is currently being stored?

  请注意，如果在终端窗口中键入 `python name.py David Carter Rongxin`，编译器不仅会输出名称的预期输出，还会输出 `hello, my name is name.py`。那么，我们如何确保编译器忽略列表中当前存储 `name.py` 的第一个元素呢？

- `slice` can be employed in our code to start the list somewhere different! Modify your code as follows:

  `slice` 可以在我们的代码中使用，以在不同的地方开始列表！按如下方式修改代码：

```
import sys

if len(sys.argv) < 2:
    sys.exit("Too few arguments")

for arg in sys.argv[1:]:
    print("hello, my name is", arg)
```

  Notice that rather than starting the list at `0`, we use square brackets to tell the compiler to start at `1` and go to the end using the `1:` argument. Running this code, you'll notice that we can improve our code using relatively simple syntax.

  请注意，我们不是从 `0` 开始列表，而是使用方括号告诉编译器从 `1` 开始，并使用 `1:` 参数到结尾。运行此代码，您会注意到我们可以使用相对简单的语法来改进我们的代码。

Packages 包

- One of the reasons Python is so popular is that there are numerous powerful third-party libraries that add functionality. We call these third-party libraries, implemented as a folder, "packages".

  Python 如此受欢迎的原因之一是有许多强大的第三方库可以添加功能。我们将这些以文件夹形式实现的第三方库称为“包”。

- PyPI is a repository or directory of all available third-party packages currently available.

  PyPI 是当前可用的所有可用第三方软件包的存储库或目录。

- `cowsay` is a well-known package that allows a cow to talk to the user.

  `Cowsay` 是一个众所周知的包，它允许 CoL 与用户交谈。

- Python has a package manager called `pip` that allows you to install packages quickly onto your system.

  Python 有一个名为 `pip` 的包管理器，可让您将包快速安装到系统上。

- In the terminal window, you can install the `cowsay` package by typing `pip install cowsay`. After a bit of output, you can now go about using this package in your code.

  在终端窗口中，您可以通过键入 `pip install cowsay` 来安装 `cowsay` 包。在一些输出之后，您现在可以在代码中使用此包。

- In your terminal window type `code say.py`. In the text editor, code as follows:

  在终端窗口中键入 `code say.py`。在文本编辑器中，编写如下代码：

```
import cowsay
import sys

if len(sys.argv) == 2:
    cowsay.cow("hello, " + sys.argv[1])
```

  Notice that the program first checks that the user inputted at least two arguments at the command line. Then, the cow should speak to the user. Type `python say.py David` and you'll see a cow saying "hello" to David.

  请注意，程序首先检查用户是否在命令行中输入了至少两个参数。然后，奶牛应该与用户交谈。键入 `python say.py David`，您将看到一头奶牛向 David 说“你好”。

- Further modify your code:

  进一步修改您的代码：

```
import cowsay
import sys

if len(sys.argv) == 2:
    cowsay.trex("hello, " + sys.argv[1])
```

  Notice that a t-rex is now saying "hello".

  请注意，霸王龙现在正在说“hello”。

- You now can see how you could install third-party packages.

  您现在可以了解如何安装第三方软件包。

- You can learn more on PyPI's entry for `cowsay`

  您可以在 PyPI 的 `cowsay` 条目上了解更多信息

- You can find other third-party packages at PyPI

  您可以在 PyPI 上找到其他第三方软件包

APIs 蜜蜂属

- APIs or "application program interfaces" allow you to connect to the code of others.

  API 或“应用程序接口”允许您连接到其他人的代码。

- `requests` is a package that allows your program to behave as a web browser would.

  `requests` 是一个包，它允许您的程序像 Web 浏览器一样运行。

- In your terminal, type `pip install requests`. Then, type `code itunes.py`.

  在您的终端中，键入 `pip install requests`。然后，键入 `code itunes.py`。

- It turns out that Apple iTunes has its own API that you can access in your programs. In your internet browser, you can visit https://itunes.apple.com/search?entity=song&limit=1&term=weezer and a text file will be downloaded. David constructed this URL by reading Apple's API documentation. Notice how this query is looking for a `song`, with a `limit` of one result, that relates to the `term` called `weezer`. Looking at this text file that is downloaded, you might find the format to be similar to that we've programmed previously in Python.

  事实证明，Apple iTunes 有自己的 API，您可以在您的程序中访问它。在您的互联网浏览器中，您可以访问 https://itunes.apple.com/search?entity=song&limit=1&term=weezer 并将下载一个文本文件。David 通过阅读 Apple 的 API 文档构建了这个 URL。请注意此查询如何查找与名为 `weezer` 的术语相关的歌曲，其结果限制为一个。查看下载的这个文本文件，您可能会发现其格式与我们之前在 Python 中编程的格式相似。

- The format in the downloaded text file is called JSON, a text-based format that is used to exchange text-based data between applications. Literally, Apple is providing a JSON file that we could interpret in our own Python program.

  下载的文本文件中的格式称为 JSON，这是一种基于文本的格式，用于在应用程序之间交换基于文本的数据。从字面上看，Apple 提供了一个 JSON 文件，我们可以在自己的 Python 程序中解释该文件。

- In the terminal window, type `code itunes.py`. Code as follows:

  在终端窗口中，键入 `code itunes.py`。代码如下：

```
import requests
import sys

if len(sys.argv) != 2:
    sys.exit()

response = requests.get("https://itunes.apple.com/search?entity=song&limit=1&term=" + sys.argv[1])
print(response.json())
```

  Notice how the returned value of `requests.get` will be stored in `response`. David, having read the Apple documentation about this API, knows that what is returned is a JSON file. Running `python itunes.py weezer`, you will see the JSON file returned by Apple. However, the JSON response is converted by Python into a dictionary. Looking at the output, it can be quite dizzying!

  请注意 `requests.get` 的返回值将如何存储在 `response` 中。David 阅读了有关此 API 的 Apple 文档，知道返回的是一个 JSON 文件。在 `weezer itunes.py 运行 python`，您将看到 Apple 返回的 JSON 文件。但是，JSON 响应由 Python 转换为字典。看看输出，可能会让人眼花缭乱！

- It turns out that Python has a built-in JSON library that can help us interpret the data received. Modify your code as follows:

  事实证明，Python 有一个内置的 JSON 库，可以帮助我们解释收到的数据。按如下方式修改代码：

```
import json
import requests
import sys

if len(sys.argv) != 2:
```

```
    sys.exit()

response = requests.get("https://itunes.apple.com/search?entity=song&limit=1&term=" + sys.argv[1])
print(json.dumps(response.json(), indent=2))
```

Notice that `json.dumps` is implemented such that it utilizes `indent` to make the output more readable. Running `python itunes.py weezer`, you will see the same JSON file. However, this time, it is much more readable. Notice now that you will see a dictionary called `results` inside the output. Inside that dictionary called `results` there are numerous keys present. Look at the `trackName` value in the output. What track name do you see in your results?

请注意，`json.dumps` 的实现是这样利用 `indent` 使输出更具可读性。在 `weezer itunes.py 运行 python`，您将看到相同的 JSON 文件。然而，这一次，它的可读性要强得多。请注意，现在您将在输出中看到一个名为 `results` 的字典。在名为 `results` 的字典中，存在许多键。查看输出中的 `trackName` 值。您在结果中看到什么轨道名称？

- How could we simply output the name of just that track name? Modify your code as follows:

  我们如何简单地输出该轨道名称的名称？按如下方式修改代码：

```
import json
import requests
import sys

if len(sys.argv) != 2:
    sys.exit()

response = requests.get("https://itunes.apple.com/search?entity=song&limit=50&term=" + sys.argv[1])

o = response.json()
for result in o["results"]:
    print(result["trackName"])
```

Notice how we are taking the result of `response.json()` and storing it in `o` (as in the lowercase letter). Then, we are iterating through the `results` in `o` and printing each `trackName`. Also notice how we have increased the limit number of results to `50`. Run your program. See the results.

请注意我们如何获取 `response.json()` 的结果并将其存储在 `o` 中（如小写字母）。然后，我们在 `o` 中迭代 `结果` 并打印每个 `trackName`。另请注意我们如何将结果的限制数量增加到 `50`。运行您的程序。查看结果。

- You can learn more about `requests` through the library's documentation.

  您可以通过库的文档了解有关 `请求` 的更多信息。

- You can learn more about JSON in Python's documentation of JSON.

  您可以在 Python 的 JSON 文档中了解有关 JSON 的更多信息。

Making Your Own Libraries
制作您自己的库

- You have the ability as a Python programmer to create your own library!

  作为 Python 程序员，您有能力创建自己的库！

- Imagine situations where you may want to re-use bits of code time and time again or even share them with others!

  想象一下，您可能希望一次又一次地重复使用代码，甚至与他人共享它们！

- We have been writing lots of code to say "hello" so far in this course. Let's create a package to allow us to say "hello" and "goodbye". In your terminal window, type `code sayings.py`. In the text editor, code as follows:

  到目前为止，在本课程中，我们已经编写了大量代码来打招呼。让我们创建一个包来允许我们说 "hello" 和 "goodbye"。在终端窗口中，键入 `code sayings.py`。在文本编辑器中，编写如下代码：

```
def hello(name):
    print(f"hello, {name}")


def goodbye(name):
    print(f"goodbye, {name}")
```

Notice that this code in and of itself does not do anything for the user. However, if a programmer were to import this package into their own program, the abilities created by the functions above could be implemented in their code.

请注意，此代码本身不会为用户执行任何操作。但是，如果程序员将此包导入到他们自己的程序中，则上述函数创建的功能可以在他们的代码中实现。

- Let's see how we could implement code utilizing this package that we created. In the terminal window, type `code say.py`. In this new file in your text editor, type the following:

  让我们看看如何利用我们创建的这个包实现代码。在终端窗口中，键入 `code say.py`。在文本编辑器的 this new file 中，键入以下内容：

```
import sys

from sayings import goodbye

if len(sys.argv) == 2:
    goodbye(sys.argv[1])
```

Notice that this code imports the abilities of `goodbye` from the `sayings` package. If the user inputed at least two arguments at the command line, it will say "goodbye" along with the string inputed at the command line.

请注意，此代码从 `sayings` 包中导入 `goodbye` 的 abilities。如果用户在命令行中输入了至少两个参数，它将说"goodbye"以及在命令行中输入的字符串。

Summing Up 总结

Libraries extend the abilities of Python. Some libraries are included by default with Python and simply need to be imported. Others are third-party packages that need to be installed using `pip`. You can make your own packages for use by yourself or others! In this lecture, you learned about...

库扩展了 Python 的功能。默认情况下，某些库包含在 Python 中，只需导入即可。其他是需要使用 `pip` 安装的第三方软件包。您可以制作自己的软件包供自己或他人使用！在本次讲座中，您了解了...

- Libraries 图书馆

- Random 随机
- Statistics 统计学
- Command-Line Arguments 命令行参数
- Slice 片
- Packages 包
- APIs 蜜蜂属
- Making Your Own Libraries
  制作您自己的库