Lecture 6 第 6 讲

File I/O 文件 I/O

- Up until now, everything we've programmed has stored information in memory. That is, once the program is ended, all information gathered from the user or generated by the program is lost.

  到目前为止，我们编程的所有内容都已将信息存储在内存中。也就是说，一旦程序结束，从用户那里收集的或由程序生成的所有信息都将丢失。

- File I/O is the ability of a program to take a file as input or create a file as output.

  文件 I/O 是程序将文件作为输入或创建文件作为输出的能力。

- To begin, in the terminal window type `code names.py` and code as follows:

  首先，在终端窗口中键入 `code names.py` 并按如下方式编写代码：

```
name = input("What's your name?" )
print(f"hello, {name}")
```

  Notice that running this code has the desired output. The user can input a name. The output is as expected.

  请注意，运行此代码具有所需的输出。用户可以输入名称。输出符合预期。

- However, what if we wanted to allow multiple names to be inputted? How might we achieve this? Recall that a `list` is a data structure that allows us to store multiple values into a single variable. Code as follows:

  但是，如果我们想允许输入多个名称怎么办？我们如何实现这一目标？回想一下，`列表` 是一种数据结构，它允许我们将多个值存储到一个变量中。代码如下：

```
names = []

for _ in range(3):
    name = input("What's your name?" )
    names.append(name)
```

  Notice that the user will be prompted three times for input. The `append` method is used to add the `name` to our `names` list.

  请注意，系统将提示用户输入三次。`append` 方法用于将 `name` 添加到我们的 `names` 列表中。

- This code could be simplified to the following:

  此代码可以简化为以下内容：

```
names = []

for _ in range(3):
    names.append(input("What's your name?" ))
```

  Notice that this has the same result as the prior block of code.

  请注意，这与前面的代码块具有相同的结果。

  - Now, let's enable the ability to print the list of names as a sorted list. Code as follows:

    现在，让我们启用将姓名列表打印为排序列表的功能。代码如下：

```
names = []

for _ in range(3):
    names.append(input("What's your name?" ))

for name in sorted(names):
    print(f"hello, {name}")
```

  Notice that once this program is executed, all information is lost. File I/O allows your program to store this information such that it can be used later.

  请注意，一旦执行此程序，所有信息都将丢失。文件 I/O 允许您的程序存储此信息，以便以后使用。

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

  您可以在 Python 的 sorted 文档中了解更多信息。

`open` 打开

- `open` is a functionality built into Python that allows you to open a file and utilize it in your program. The `open` function allows you to open a file such that you can read from it or write to it.

  `open` 是 Python 中内置的一项功能，允许您打开文件并在程序中使用它。`open` 函数允许您打开文件，以便您可以从中读取或写入文件。

- To show you how to enable file I/O in your program, let's rewind a bit and code as follows:

  为了向您展示如何在程序中启用文件 I/O，让我们快退一点代码，如下所示：

```
name = input("What's your name? ")

file = open("names.txt", "w")
```

```
file.write(name)
file.close()
```

Notice that the `open` function opens a file called `names.txt` with writing enabled, as signified by the `w`. The code above assigns that opened file to a variable called `file`. The line `file.write(name)` writes the name to the text file. The line after that closes the file.

请注意，`open` 函数打开一个名为 `names.txt` 的文件，并启用了写入功能，如 `w` 所示。上面的代码将打开的文件分配给名为 `file` 的变量。`行 file.write（name）` 将名称写入文本文件。之后的行将关闭文件。

- Testing out your code by typing `python names.py`, you can input a name and it saves to the text file. However, if you run your program multiple times using different names, you will notice that this program will entirely rewrite the `names.txt` file each time.

  通过键入 `python names.py` 来测试您的代码，您可以输入一个名称并将其保存到文本文件中。但是，如果您使用不同的名称多次运行程序，您会注意到该程序每次都会完全重写 `names.txt` 文件。

- Ideally, we want to be able to append each of our names to the file. Remove the existing text file by typing `rm names.txt` in the terminal window. Then, modify your code as follows:

  理想情况下，我们希望能够将每个名称附加到文件中。通过在终端窗口中键入 `rm names.txt` 删除现有文本文件。然后，按如下方式修改您的代码：

```
name = input("What's your name? ")

file = open("names.txt", "a")
file.write(name)
file.close()
```

Notice that the only change to our code is that the `w` has been changed to `a` for "append". Rerunning this program multiple times, you will notice that names will be added to the file. However, you will notice a new problem!

请注意，对代码的唯一更改是 `w` 已更改为 `a` 表示 "append"。多次重新运行此程序，您将注意到名称将被添加到文件中。但是，您会注意到一个新问题！

- Examining your text file after running your program multiple times, you'll notice that the names are running together. The names are being appended without any gaps between each of the names. You can fix this issue. Again, remove the existing text file by typing `rm names.txt` in the terminal window. Then, modify your code as follows:

  在多次运行程序后检查文本文件，您会注意到这些名称一起运行。附加名称时，每个名称之间没有任何间隙。您可以修复此问题。同样，通过在终端窗口中键入 `rm names.txt` 来删除现有的文本文件。然后，按如下方式修改您的代码：

```
name = input("What's your name? ")

file = open("names.txt", "a")
file.write(f"{name}\n")
file.close()
```

Notice that the line with `file.write` has been modified to add a line break at the end of each name.

请注意，`file.write` 的行已被修改，在每个名称的末尾添加一个换行符。

- This code is working quite well. However, there are ways to improve this program. It so happens that it's quite easy to forget to close the file.

  此代码运行良好。但是，有一些方法可以改进这个程序。碰巧很容易忘记关闭文件。

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

  您可以在 Python 的 open 文档中了解更多信息。

`with` 跟

- The keyword `with` allows you to automate the closing of a file.

  关键字 `with` 允许您自动关闭文件。

- Modify your code as follows:

  按如下方式修改代码：

```
name = input("What's your name? ")

with open("names.txt", "a") as file:
    file.write(f"{name}\n")
```

Notice that the line below `with` is indented.

请注意，下面的行 `with` 是缩进的。

- Up until this point, we have been exclusively writing to a file. What if we want to read from a file? To enable this functionality, modify your code as follows:

  到目前为止，我们一直只写入文件。如果我们想从文件中读取数据怎么办？要启用此功能，请按如下方式修改您的代码：

```
with open("names.txt", "r") as file:
    lines = file.readlines()

for line in lines:
    print("hello,", line)
```

Notice that `readlines` has a special ability to read all the lines of a file and store them in a list called `lines`. Running your program, you will notice that the output is quite ugly. There seem to be multiple line breaks where there should be only one.

请注意，`readlines` 具有读取文件的所有行并将它们存储在名为 `lines` 的列表中的特殊功能。运行您的程序，您会注意到输出非常难看。似乎有多个换行符，而这里应该只有一个。

- There are many approaches to fix this issue. However, here is a simple way to fix this error in our code:

  有多种方法可以解决此问题。但是，这里有一个简单的方法来修复我们代码中的这个错误：

```
with open("names.txt", "r") as file:
    lines = file.readlines()

for line in lines:
    print("hello,", line.rstrip())
```

Notice that `rstrip` has the effect of removing the extraneous line break at the end of each line.

请注意，`rstrip` 具有删除每行末尾无关的换行符的效果。

- Still, this code could be simplified even further:

  不过，这段代码可以进一步简化：

```
with open("names.txt", "r") as file:
    for line in file:
        print("hello,", line.rstrip())
```

  Notice that running this code, it is correct. However, notice that we are not sorting the names.

  请注意，运行此代码是正确的。但是，请注意，我们没有对名称进行排序。

- This code could be further improved to allow for the sorting of the names:

  此代码可以进一步改进，以允许对名称进行排序：

```
names = []

with open("names.txt") as file:
    for line in file:
        names.append(line.rstrip())

for name in sorted(names):
    print(f"hello, {name}")
```

  Notice that `names` is a blank list where we can collect the names. Each name is appended to the `names` list in memory. Then, each name in the sorted list in memory is printed. Running your code, you will see that the names are now properly sorted.

  请注意，`names` 是一个空白列表，我们可以在其中收集 names。每个名称都附加到内存中 `的名称` 列表中。然后，打印内存中排序列表中的每个名称。运行您的代码，您将看到名称现在已正确排序。

- What if we wanted the ability to store more than just the names of students? What if we wanted to store both the student's name and their house as well?

  如果我们想要存储的不仅仅是学生姓名，该怎么办？如果我们同时想存储学生的姓名和他们的房子怎么办？

CSV CSV 格式

- CSV stands for "comma separated values".

  CSV 代表"逗号分隔值"。

- In your terminal window, type `code students.csv`. Ensure your new CSV file looks like the following:

  在终端窗口中，键入 `code students.csv`。确保您的新 CSV 文件如下所示：

```
Hermione,Gryffindor
Harry,Gryffindor
Ron,Gryffindor
Draco,Slytherin
```

- Let's create a new program by typing `code students.py` and code as follows:

  让我们通过键入 `code students.py` 和 code 来创建一个新程序，如下所示：

```
with open("students.csv") as file:
    for line in file:
        row = line.rstrip().split(",")
        print(f"{row[0]} is in {row[1]}")
```

  Notice that `rstrip` removes the end of each line in our CSV file. `split` tells the compiler where to find the end of each of our values in our CSV file. `row[0]` is the first element in each line of our CSV file. `row[1]` is the second element in each line in our CSV file.

  请注意，`rstrip` 删除了 CSV 文件中每行的结尾。`split` 告诉编译器在 CSV 文件中每个值的末尾的位置。`row[0]` 是 CSV 文件每行中的第一个元素。`row[1]` 是 CSV 文件中每行的第二个元素。

- The above code is effective at dividing each line or "record" of our CSV file. However, it's a bit cryptic to look at if you are unfamiliar with this type of syntax. Python has built-in ability that could further simplify this code. Modify your code as follows:

  上面的代码可以有效地划分 CSV 文件的每一行或"记录"。但是，如果您不熟悉这种类型的语法，那么查看它有点神秘。Python 具有可以进一步简化此代码的内置功能。按如下方式修改代码：

```
with open("students.csv") as file:
    for line in file:
        name, house = line.rstrip().split(",")
        print(f"{name} is in {house}")
```

  Notice that the `split` function actually returns two values: The one before the comma and the one after the comma. Accordingly, we can rely upon that functionality to assign two variables at once instead of one!

  请注意，`split` 函数实际上返回两个值：一个在逗号之前，一个在逗号之后。因此，我们可以依靠该功能一次分配两个变量，而不是一个！

- Imagine that we would again like to provide this list as sorted output? You can modify your code as follows:

  想象一下，我们想再次将此列表作为排序输出提供？您可以按如下方式修改代码：

```
students = []

with open("students.csv") as file:
    for line in file:
        name, house = line.rstrip().split(",")
        students.append(f"{name} is in {house}")

for student in sorted(students):
    print(student)
```

  Notice that we create a `list` called `students`. We `append` each string to this list. Then, we output a sorted version of our list.

  请注意，我们创建了一个名为 `students` `的列表`。我们将每个字符串 `附加到` 此列表中。然后，我们输出列表的排序版本。

- Recall that Python allows for `dictionaries` where a key can be associated with a value. This code could be further improved

  回想一下，Python 允许将键与值相关联的 `字典`。此代码可以进一步改进

```
students = []
```

```
students = []

with open("students.csv") as file:
    for line in file:
        name, house = line.rstrip().split(",")
        student = {}
        student["name"] = name
        student["house"] = house
        students.append(student)

for student in students:
    print(f"{student['name']} is in {student['house']}")
```

Notice that we create an empty dictionary called `student`. We add the values for each student, including their name and house into the `student` dictionary. Then, we append that student to the `list` called `students`.

请注意，我们创建了一个名为 `student` 的空字典。我们将每个学生的值（包括他们的姓名和学院）添加到 `学生` 字典中。然后，我们将该学生附加到名为 `students` 的列表中 。

- We can improve our code to illustrate this as follows:

  我们可以改进我们的代码来说明这一点，如下所示：

```
students = []

with open("students.csv") as file:
    for line in file:
        name, house = line.rstrip().split(",")
        student = {"name": name, "house": house}
        students.append(student)

for student in students:
    print(f"{student['name']} is in {student['house']}")
```

Notice that this produces the desired outcome, minus the sorting of students.

请注意，这会产生所需的结果，减去 students 的排序。

- Unfortunately, we cannot sort the students as we had prior because each student is now a dictionary inside of a list. It would be helpful if Python could sort the `students` list of `student` dictionaries that sorts this list of dictionaries by the student's name.

  遗憾的是，我们无法像以前那样对学生进行排序，因为每个学生现在都是列表内的字典。如果 Python 可以对 `学生` 词典列表进行排序，该列表按 学生姓名对词典列表进行排序，那将会很有帮助。

- To implement this in our code, make the following changes:

  要在我们的代码中实现这一点，请进行以下更改：

```
students = []

with open("students.csv") as file:
    for line in file:
        name, house = line.rstrip().split(",")
        students.append({"name": name, "house": house})


def get_name(student):
    return student["name"]


for student in sorted(students, key=get_name):
    print(f"{student['name']} is in {student['house']}")
```

Notice that `sorted` needs to know how to get the key of each student. Python allows for a parameter called `key` where we can define on what "key" the list of students will be sorted. Therefore, the `get_name` function simply returns the key of `student["name"]`. Running this program, you will now see that the list is now sorted by name.

请注意， `sorted` 需要知道如何获取每个学生的 key。Python 允许使用一个名为 `key` 的参数，我们可以在其中定义学生列表将按哪个 "key" 进行排序。因此， `get_name` 函数只返回 `student["name"]` 的 key。运行此程序，您现在将看到列表现在按名称排序。

- Still, our code can be further improved upon. It just so happens that if you are only going to use a function like `get_name` once, you can simplify your code in the manner presented below. Modify your code as follows:

  尽管如此，我们的代码还可以进一步改进。碰巧的是，如果你只打算使用一次像 `get_name` 这样的函数，你可以按照下面显示的方式简化你的代码。按如下方式修改代码：

```
students = []

with open("students.csv") as file:
    for line in file:
        name, house = line.rstrip().split(",")
        students.append({"name": name, "house": house})

for student in sorted(students, key=lambda student: student["name"]):
    print(f"{student['name']} is in {student['house']}")
```

Notice how we use a `lambda` function, an anonymous function, that says "Hey Python, here is a function that has no name: Given a `student`, access their `name` and return that to the `key`.

请注意我们如何使用 `lambda` 函数，一个匿名函数，它说"嘿 Python，这是一个没有名称的函数：给定一个 `学生` ，访问他们 `的名字` 并将其返回给 `密钥` 。

- Unfortunately, our code is a bit fragile. Suppose that we changed our CSV file such that we indicated where each student grew up. What would be the impact of this upon our program? First, modify your `students.csv` file as follows:

  不幸的是，我们的代码有点脆弱。假设我们更改了 CSV 文件，以便我们指示每个学生的成长地点。这对我们的计划有什么影响？首先，按如下方式修改 `students.csv` 文件：

```
Harry,"Number Four, Privet Drive"
Ron,The Burrow
Draco,Malfoy Manor
```

Notice how running our program how will produce a number of errors.

请注意，运行我们的程序 how 将产生许多错误。

- Now that we're dealing with homes instead of houses, modify your code as follows:

现在，我们处理的是 homes 而不是 houses，请按如下方式修改您的代码：

```
students = []

with open("students.csv") as file:
    for line in file:
        name, home = line.rstrip().split(",")
        students.append({"name": name, "home": home})

for student in sorted(students, key=lambda student: student["name"]):
    print(f"{student['name']} is in {student['home']}")
```

Notice that running our program still does not work properly. Can you guess why?

请注意，运行我们的程序仍然无法正常工作。你能猜到为什么吗？

- The `ValueError: too many values to unpack` error produced by the compiler is a result of the fact that we previously created this program expecting the CSV file is `split` using a `,` (comma). We could spend more time addressing this, but indeed someone else has already developed a way to "parse" (that is, to read) CSV files!

  编译器产生的 `ValueError: too many values to unpack` 错误是由于我们之前创建此程序时期望使用 `,` （逗号）`拆分` CSV 文件。我们可以花更多时间来解决这个问题，但实际上，其他人已经开发了一种“解析”（即读取） CSV 文件的方法！

- Python's built-in `csv` library comes with an object called a `reader`. As the name suggests, we can use a `reader` to read our CSV file despite the extra comma in "Number Four, Privet Drive". A `reader` works in a `for` loop, where each iteration the `reader` gives us another row from our CSV file. This row itself is a list, where each value in the list corresponds to an element in that row. `row[0]`, for example, is the first element of the given row, while `row[1]` is the second element.

  Python 的内置 `csv` 库附带一个称为 `reader` 的对象。顾名思义，我们可以使用 `阅读器` 来读取我们的 CSV 文件，尽管“Number Four， Privet Drive”中多了逗号。`读取器` 在 `for` 循环中工作，每次迭代读取 `器` 都会从 CSV 文件中为我们提供另一行。此行本身是一个列表，其中列表中的每个值对应于该行中的一个元素。例如，`row[0]` 是给定行的第一个元素，而 `row[1]` 是第二个元素。

```
import csv

students = []

with open("students.csv") as file:
    reader = csv.reader(file)
    for row in reader:
        students.append({"name": row[0], "home": row[1]})

for student in sorted(students, key=lambda student: student["name"]):
    print(f"{student['name']} is from {student['home']}")
```

Notice that our program now works as expected.

请注意，我们的程序现在按预期工作。

- Up until this point, we have been relying upon our program to specifically decide what parts of our CSV file are the names and what parts are the homes. It's better design, though, to bake this directly into our CSV file by editing it as follows:

  到目前为止，我们一直依赖我们的程序来专门决定 CSV 文件的哪些部分是名称，哪些部分是房屋。不过，更好的设计是将其直接烘焙到我们的 CSV 文件中，方法是按如下方式对其进行编辑：

```
name,home
Harry,"Number Four, Privet Drive"
Ron,The Burrow
Draco,Malfoy Manor
```

Notice how we are explicitly saying in our CSV file that anything reading it should expect there to be a name value and a home value in each line.

请注意，我们在 CSV 文件中明确表示，读取它的任何内容都应该期望每行中都有一个 name 值和一个 home 值。

- We can modify our code to use a part of the `csv` library called a `DictReader` to treat our CSV file with even more flexibilty:

  我们可以修改我们的代码，以使用 `csv` 库中名为 `DictReader` 的部分来更灵活地处理我们的 CSV 文件：

```
import csv

students = []

with open("students.csv") as file:
    reader = csv.DictReader(file)
    for row in reader:
        students.append({"name": row["name"], "home": row["home"]})

for student in sorted(students, key=lambda student: student["name"]):
    print(f"{student['name']} is in {student['home']}")
```

Notice that we have replaced `reader` with `DictReader`, which returns one dictionary at a time. Also, notice that the compiler will directly access the `row` dictionary, getting the `name` and `home` of each student. This is an example of coding defensively. As long as the person designing the CSV file has inputted the correct header information on the first line, we can access that information using our program.

请注意，我们已将 `reader` 替换为 `DictReader`，它一次返回一个字典。另外，请注意，编译器将直接访问 `行` 字典，获取每个学生 `的姓名` 和 `家庭`。这是防御性编码的一个示例。只要设计 CSV 文件的人在第一行输入了正确的标题信息，我们就可以使用我们的程序访问该信息。

- Up until this point, we have been reading CSV files. What if we want to write to a CSV file?

  到目前为止，我们一直在读取 CSV 文件。如果我们想写入 CSV 文件怎么办？

- To begin, let's clean up our files a bit. First, delete the `students.csv` file by typing `rm students.csv` in the terminal window. This command will only work if you're in the same folder as your `students.csv` file.

  首先，让我们稍微清理一下文件。首先，通过在终端窗口中键入 `rm students.csv` 删除 `students.csv` 文件。此命令仅在您与 `students.csv` 文件位于同一文件夹中时有效。

- Then, in `students.py`, modify your code as follows:

  然后，在 `students.py` 中，按如下方式修改您的代码：

```
import csv

name = input("What's your name? ")
home = input("Where's your home? ")

with open("students.csv", "a") as file:
    writer = csv.DictWriter(file, fieldnames=["name", "home"])
```

```
    writer.writerow({"name": name, "home": home})
```

Notice how we are leveraging the built-in functionality of `DictWriter`, which takes two parameters: the `file` being written to and the `fieldnames` to write. Further, notice how the `writerow` function takes a dictionary as its parameter. Quite literally, we are telling the compiler to write a row with two fields called `name` and `home`.

请注意我们如何利用 `DictWriter` 的内置功能，它采用两个参数：要写入 `的文件` 和要写入的 `字段名称`。此外，请注意 `writerow` 函数如何将字典作为其参数。从字面上看，我们告诉编译器编写一行，其中包含两个字段，分别是 `name` 和 `home`。

- Note that there are many types of files that you can read from and write to.

  请注意，您可以读取和写入多种类型的文件。

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

  您可以在 Python 的 CSV 文档中了解更多信息。

Binary Files and `PIL`

二进制文件和 `PIL`

- One more type of file that we will discuss today is a binary file. A binary file is simply a collection of ones and zeros. This type of file can store anything including, music and image data.

  我们今天要讨论的另一种类型的文件是二进制文件。二进制文件只是 1 和 0 的集合。这种类型的文件可以存储任何内容，包括音乐和图像数据。

- There is a popular Python library called `PIL` that works well with image files.

  有一个名为 `PIL` 的流行 Python 库，它适用于图像文件。

- Animated GIFs are a popular type of image file that has many image files within it that are played in sequence over and over again, creating a simplistic animation or video effect.

  动画 GIF 是一种流行的图像文件类型，其中包含许多图像文件，这些文件一遍又一遍地按顺序播放，从而创建简单的动画或视频效果。

- Imagine that we have a series of costumes, as illustrated below.

  想象一下我们有一系列服装，如下图所示。

- Here is `costume1.gif`.

  这是 `costume1.gif`。

- Here is another one called `costume2.gif`. Notice how the leg positions are slightly different.

  这是另一个名为 `costume2.gif` 的。请注意，腿的位置略有不同。

- Before proceeding, please make sure that you have downloaded the source code files from the course website. It will not be possible for you to code the following without having the two images above in your possession and stored in your IDE.

  在继续之前，请确保您已从课程网站下载了源代码文件。如果您拥有上述两个图像并将其存储在 IDE 中，则无法编写以下内容。

- In the terminal window type `code costumes.py` and code as follows:

  在终端窗口中，键入 `code costumes.py` 和 code，如下所示：

```
import sys

from PIL import Image

images = []

for arg in sys.argv[1:]:
    image = Image.open(arg)
    images.append(image)

images[0].save(
    "costumes.gif", save_all=True, append_images=[images[1]], duration=200, loop=0
)
```

Notice that we import the `Image` functionality from `PIL`. Notice that the first `for` loop simply loops through the images provided as command-line arguments and stores theme into the `list` called `images`. The `1:` starts slicing `argv` at its second element. The last lines of code saves the first image and also appends a second image to it as well, creating an animated gif. Typing `python costumes.py costume1.gif costume2.gif` into the terminal. Now, type `code costumes.gif` into the terminal window, and you can now see an animated GIF.

请注意，我们从 `PIL` 导入 `Image` 功能。请注意，第一个 `for` 循环只是遍历作为命令行参数提供的图像，并将 theme 存储到名为 `images` 的列表中。`1:` 开始在 argv 的第二个元素处切片 `argv`。最后几行代码保存第一张图片，并将第二张图片附加到其中，从而创建一个动画 gif。在终端中键入 `python costumes.py costume1.gif costume2.gif`。现在，在终端窗口中键入 `code costumes.gif`，您现在可以看到一个动画 GIF。

- You can learn more in Pillow's documentation of PIL.
  您可以在 Pillow 的 PIL 文档中了解更多信息。

Summing Up 总结

Now, we have not only seen that we can write and read files textually—we can also read and write files using ones and zeros. We can't wait to see what you achieve with these new abilities next.

现在，我们不仅看到我们可以通过文本方式写入和读取文件，还可以使用 1 和 0 读取和写入文件。我们迫不及待地想看看您接下来通过这些新功能实现什么。

- File I/O 文件 I/O
- `open` 打开
- `with` 跟
- CSV CSV 格式
- `PIL` 必