Lecture 2 第 2 讲

Loops 循环

- Essentially, loops are a way to do something over and over again.

  从本质上讲，Loop 是一种一遍又一遍地做某事的方法。
- Begin by typing `code cat.py` in the terminal window.

  首先在终端窗口中键入 `代码 cat.py`。
- In the text editor, begin with the following code:

  在文本编辑器中，从以下代码开始：

```
print("meow")
print("meow")
print("meow")
```

  Running this code by typing `python cat.py`, you'll notice that the program meows three times.

  通过键入 `python cat.py` 运行此代码，您会注意到程序发出 3 次喵喵叫。
- In developing as a programmer, you want to consider how one could improve areas of one's code where one types the same thing over and over again. Imagine where one might want to "meow" 500 times. Would it be logical to type that same expression of `print("meow")` over and over again?

  在以程序员身份进行开发时，您需要考虑如何改进代码中一遍又一遍地键入相同内容的区域。想象一下，人们可能想在哪里"喵喵"500 次。一遍又一遍地输入相同的 `print（"meow"）` 表达式是否合乎逻辑？
- Loops enable you to create a block of code that executes over and over again.

  循环使您能够创建一遍又一遍地执行的代码块。

While Loops While 循环

- The `while` loop is nearly universal throughout all coding languages.

  `while` 循环在所有编码语言中几乎是通用的。
- Such a loop will repeat a block of code over and over again.

  这样的循环将一遍又一遍地重复一段代码。
- In the text editor window, edit your code as follows:

  在文本编辑器窗口中，按如下方式编辑您的代码：

```
i = 3
while i != 0:
    print("meow")
```

  Notice how even though this code will execute `print("meow")` multiple times, it will never stop! It will loop forever. `while` loops work by repeatedly asking if the condition of the loop has been fulfilled. In this case, the compiler is asking, "does `i` not equal zero?" When you get stuck in a loop that executes forever, you can press `control-c` on your keyboard to break out of the loop.

  请注意，即使此代码将多次执行 `print（"meow"）`，它也永远不会停止！它将永远循环。`while` 循环的工作原理是反复询问循环的条件是否已满足。在这种情况下，编译器会问"`i` 不等于 0 吗？当您陷入永远执行的循环中时，您可以按键盘上的 `control-c` 来跳出循环。
- To fix this loop that lasts forever, we can edit our code as follows

  为了修复这个永远持续的循环，我们可以按如下方式编辑我们的代码

```
i = 3
while i != 0:
  print("meow")
  i = i - 1
```

  Notice that now our code executes properly, reducing `i` by `1` for each "iteration" through the loop. The term iteration has special significance within coding. By iteration, we mean one cycle through the loop. The first iteration is the "0th" iteration through the loop. The second is the "1st" iteration. In programming, we count starting with 0, then 1, then 2.

  请注意，现在我们的代码可以正确执行，循环中每个"迭代"都会将 `i` 减少 `1`。术语 iteration 在编码中具有特殊意义。迭代是指循环中的一个循环。第一次迭代是循环中的"第 0 次"迭代。第二个是 "1st" 迭代。在编程中，我们从 0 开始计数，然后是 1，然后是 2。
- We can further improve our code as follows:

  我们可以进一步改进我们的代码，如下所示：

```
i = 1
while i <= 3:
    print("meow")
```

```
    i = i + 1
```

Notice that when we code `i = i + 1` we assign the value of `i` from the right to the left. Above, we are starting `i` at one, like most humans count (1, 2, 3). If you execute the code above, you'll see it meows three times. It's best practice in programming to begin counting with zero.

请注意，当我们编码 `i = i + 1` 时，我们从右到左分配 `i` 的值。在上面，我们从 `i` 开始，就像大多数人一样（1，2，3）。如果你执行上面的代码，你会看到它喵喵叫 3 次。编程中的最佳实践是从 0 开始计数。

- We can improve our code to start counting with zero:

  我们可以改进我们的代码，从 0 开始计数：

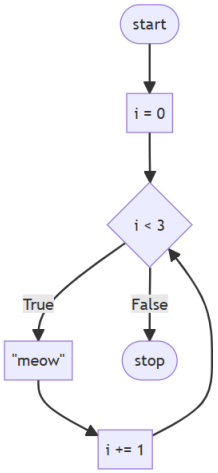
```
i = 0
while i < 3:
    print("meow")
    i += 1
```

Notice how changing the operator to `i < 3` allows our code to function as intended. We begin by counting with 0 and it iterates through our loop three times, producing three meows. Also, notice how `i += 1` is the same as saying `i = i + 1`.

请注意，将运算符更改为 `i < 3` 如何使我们的代码按预期运行。我们从 0 开始计数，它遍历我们的循环 3 次，产生 3 次喵喵声。另外，请注意 `i += 1` 与说 `i = i + 1` 是一样的。

- Our code at this point is illustrated as follows:

  此时我们的代码如下所示：



Notice how our loop counts i up to, but not through `3`.

请注意我们的循环是如何计数 i 的，而不是到 `3`。

For Loops For 循环

- A `for` loop is a different type of loop.

  `for` 循环是一种不同类型的循环。

- To best understand a `for` loop, it's best to begin by talking about a new variable type called a `list` in Python. As in other areas of our lives, we can have a grocery list, a to-do list, etc.

  为了更好地理解 `for` 循环，最好先讨论一种在 Python 中称为 `list` 的新变量类型。就像我们生活的其他领域一样，我们可以有购物清单、待办事项清单等。

- A `for` loop iterates through a `list` of items. For example, in the text editor window, modify your `cat.py` code as follows:

  `for` 循环遍历项目 `列表`。例如，在文本编辑器窗口中，按如下方式修改 `cat.py` 代码：

```
for i in [0, 1, 2]:
    print("meow")
```

Notice how clean this code is compared to your previous `while` loop code. In this code, `i` begins with `0`, meows, `i` is assigned `1`, meows, and, finally, `i` is assigned `2`, meows, and then ends.

请注意，与之前的 `while` 循环代码相比，此代码是多么干净。在此代码中，`i` 以 `0` 开头，喵喵，`i` 被分配为 `1`，喵喵，最后，`i` 被分配为 `2`，喵喵，然后结束。

- While this code accomplishes what we want, there are some possibilities for improving our code for extreme cases. At first glance, our code looks great. However, what if you wanted to iterate up to a million? It's best to create code that can work with such extreme cases. Accordingly, we can improve our code as follows:

  虽然这段代码完成了我们想要的，但对于极端情况，我们有一些改进代码的可能性。乍一看，我们的代码看起来很棒。但是，如果您想迭代到 100 万怎么办？最好创建可以处理此类极端情况的代码。因此，我们可以按如下方式改进我们的代码：

```
for i in range(3):
    print("meow")
```

Notice how `range(3)` provides back three values (`0`, `1`, and `2`) automatically. This code will execute and produce the intended effect, meowing three times.

请注意 `range（3）` 如何自动提供三个值（`0`、`1` 和 `2`）。此代码将执行并产生预期的效果，喵喵叫 3 次。

- Our code can be further improved. Notice how we never use `i` explicitly in our code. That is, while Python needs the `i` as a place to store the number of the iteration of the loop, we never use it for any other purpose. In Python, if such a variable does not have any other significance in our code, we can simply represent this variable as a single underscore `_`. Therefore, you can modify your code as follows:

  我们的代码可以进一步改进。请注意，我们从来没有在代码中显式使用 `i`。也就是说，虽然 Python 需要 `i` 作为存储循环迭代编号的地方，但我们从不将其用于任何其他目的。在 Python 中，如果这样的变量在我们的代码中没有任何其他意义，我们可以简单地将此变量表示为单个下划线 `_`。因此，您可以按如下方式修改代码：

```
for _ in range(3):
    print("meow")
```

Notice how changing the `i` to `_` has zero impact on the functioning of our program.

请注意，将 `i` 更改为 `_` 对程序运行的影响为零。

- Our code can be further improved. To stretch your mind to the possibilities within Python, consider the following code:

  我们的代码可以进一步改进。要了解 Python 中的可能性，请考虑以下代码：

```
print("meow" * 3)
```

Notice how it will meow three times, but the program will produce `meowmeowmeow` as the result. Consider: How could you create a line break at the end of each meow?

请注意它会如何喵喵叫 3 次，但程序会生成 `喵喵喵`。考虑一下：你怎么能在每个喵喵的末尾创建一个换行符？

- Indeed, you can edit your code as follows:

  实际上，您可以按如下方式编辑代码：

```
print("meow\n" * 3, end="")
```

Notice how this code produces three meows, each on a separate line. By adding `end=""` and the `\n` we tell the compiler to add a line break at the end of each meow.

请注意此代码如何生成三个 meow，每个 meow 在单独的行上。通过添加 `end=""` 和 `\n`，我们告诉编译器在每个 meow 的末尾添加一个换行符。

Improving with User Input

根据用户意见进行改进

- Perhaps we want to get input from our user. We can use loops as a way of validating the input of the user.

  也许我们想从用户那里获得输入。我们可以使用 Loop 作为验证用户输入的一种方式。

- A common paradigm within Python is to use a `while` loop to validate the input of the user.

  Python 中的一种常见范例是使用 `while` 循环来验证用户的输入。

- For example, let's try prompting the user for a number greater than or equal 0:

  例如，让我们尝试提示用户输入一个大于或等于 0 的数字：

```
while True:
    n = int(input("What's n? "))
    if n < 0:
        continue
    else:
        break
```

- Notice that we've introduced two new keywords in Python, `continue` and `break`. `continue` explicitly tells Python to go to the next iteration of a loop. `break`, on the other hand, tells Python to "break out" of a loop early before it has finished all of its iterations. In this case, we'll `continue` to the next iteration of the loop when `n` is less than 0—ultimately reprompting the user with "What's n?". If, though, `n` is greater than or equal to 0, we'll `break` out of the loop and allow the rest of our program to run.

  请注意，我们在 Python 中引入了两个新关键字 `continue` 和 `break`。`continue` 显式告诉 Python 转到循环的下一次迭代。另一方面，`break` 告诉 Python 在完成所有迭代之前尽早“突破”循环。在这种情况下，当 `n` 小于 0 时，我们将继续循环 的下一次迭代，最终用 “What's n? ” 重新提示用户。但是，如果 `n` 大于或等于 0，我们将 跳 出循环并允许程序的其余部分运行。

- It turns out that the `continue` keyword is redundant in this case. We can improve our code as follows:

  事实证明，在这种情况下，`continue` 关键字是多余的。我们可以按如下方式改进我们的代码：

```
while True:
    n = int(input("What's n? "))
    if n > 0:
        break

for _ in range(n):
    print("meow")
```

Notice how this while loop will always run (forever) until `n` is greater than `0`. When `n` is greater than `0`, the loop breaks.

请注意，这个 while 循环将始终（永远）运行，直到 `n` 大于 `0`。当 `n` 大于 `0` 时，循环中断。

- Bringing in our prior learning, we can use functions to further improve our code:

  结合我们之前的学习，我们可以使用 functions 来进一步改进我们的代码：

```
def main():
    meow(get_number())


def get_number():
    while True:
        n = int(input("What's n? "))
        if n > 0:
            return n


def meow(n):
    for _ in range(n):
        print("meow")


main()
```

Notice how not only did we change your code to operate in multiple functions, but we also used a `return` statement to `return` the value of `n` back to the `main` function.

请注意，我们不仅更改了您的代码以在多个函数中运行，而且还使用 `return` 语句将 `n` 的值 返回 给 `main` 函数。

More About Lists 更多关于 Lists

- Consider the world of Hogwarts from the famed Harry Potter universe.

  想想著名的哈利波特宇宙中的霍格沃茨世界。

- In the terminal, type `code hogwarts.py`.

  在终端中，键入 `code hogwarts.py`。

- In the text editor, code as follows:

  在文本编辑器中，编写如下代码：

```
students = ["Hermione", "Harry", "Ron"]

print(students[0])
print(students[1])
print(students[2])
```

  Notice how we have a `list` of students with their names as above. We then print the student who is at the 0th location, "Hermione". Each of the other students is printed as well.

  请注意，我们有一个学生 `列表`，上面有他们的名字。然后，我们打印位于第 0 个位置的学生 "Hermione"。其他每个学生也被打印出来。

- Just as we illustrated previously, we can use a loop to iterate over the list. You can improve your code as follows:

  正如我们之前演示的那样，我们可以使用循环来迭代列表。您可以按如下方式改进代码：

```
students = ["Hermione", "Harry", "Ron"]

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

  Notice that for each `student` in the `students` list, it will print the student as intended. You might wonder why we did not use the `_` designation as discussed prior. We choose not to do this because `student` is explicitly used in our code.

  请注意，对于 `students` 列表中的每个 `学生`，它将按预期打印学生。您可能想知道为什么我们没有像前面讨论的那样使用 `_` 名称。我们选择不这样做，因为在我们的代码中明确使用了 `student`。

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

  您可以在 Python 的列表文档中了解更多信息。

Length 长度

- We can utilize `len` as a way of checking the length of the `list` called `students`.

  我们可以使用 `len` 作为检查名为 `students` 的`列表`长度的一种方式。

- Imagine that you don't simply want to print the name of the student but also their position in the list. To accomplish this, you can edit your code as follows:

  想象一下，您不仅想打印学生的姓名，还想打印他们在列表中的位置。为此，您可以按如下方式编辑代码：

```
students = ["Hermione", "Harry", "Ron"]

for i in range(len(students)):
    print(i + 1, students[i])
```

  Notice how executing this code results in not only getting the position of each student plus one using `i + 1`, but also prints the name of each student. `len` allows you to dynamically see how long the list of the students is regardless of how much it grows.

  请注意，执行此代码不仅会导致使用 `i + 1` 获取每个学生加 1 的位置，而且还会打印每个学生的姓名。`len` 允许您动态查看 Students 列表的长度，而不管它增长了多少。

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

  您可以在 Python 的 len 文档中了解更多信息。

Dictionaries 字典

- `dict`s or dictionaries is a data structure that allows you to associate keys with values.

  `dict`s 或 dictionaries 是一种数据结构，允许您将键与值相关联。

- Where a `list` is a list of multiple values, a `dict` associates a key with a value.

  其中 `list` 是多个值的列表，`而 dict` 将键与值相关联。

- Considering the houses of Hogwarts, we might assign specific students to specific houses.

  考虑到霍格沃茨的学院，我们可能会将特定的学生分配到特定的学院。

Hermione	Harry	Ron	Draco
Gryffindor	Gryffindor	Gryffindor	Slytherin

- We could use `list`s alone to accomplish this:

  我们可以单独使用 `list`s 来完成这个：

```
students = ["Hermione", "Harry", "Ron", "Draco"]
houses = ["Gryffindor", "Gryffindor", "Griffindor", "Slytherin"]
```

  Notice that we can promise that we will always keep these lists in order. The individual at the first position of `students` is associated with the house at the first position of the `houses` list, and so on. However, this can become quite cumbersome as our lists grow!

请注意，我们可以承诺我们将始终保持这些列表的顺序。位于 `students` 第一个位置的 individual 与 `houses` 列表第一个位置的 house 相关联，依此类推。但是，随着我们列表的增长，这可能会变得非常麻烦！

- We can better our code using a `dict` as follows:

  我们可以使用 `dict` 改进我们的代码，如下所示：

```
students = {
    "Hermione": "Gryffindor",
    "Harry": "Gryffindor",
    "Ron": "Gryffindor",
    "Draco": "Slytherin",
}
print(students["Hermione"])
print(students["Harry"])
print(students["Ron"])
print(students["Draco"])
```

  Notice how we use `{}` curly braces to create a dictionary. Where `list`s use numbers to iterate through the list, `dict`s allow us to use words.

  请注意我们如何使用 `{}` 大括号来创建字典。`list`s 使用数字遍历列表，而 `dict`s 允许我们使用单词。

- Run your code and make sure your output is as follows:

  运行代码并确保输出如下所示：

```
$ python hogwarts.py
Gryffindor
Gryffindor
Gryffindor
Slytherin
```

- We can improve our code as follows:

  我们可以按如下方式改进我们的代码：

```
students = {
    "Hermione": "Gryffindor",
    "Harry": "Gryffindor",
    "Ron": "Gryffindor",
    "Draco": "Slytherin",
}
for student in students:
    print(student)
```

  Notice how, executing this code, the for loop will only iterate through all the keys, resulting in a list of the names of the students. How could we print out both values and keys?

  请注意，在执行此代码时，for 循环将仅遍历所有键，从而生成学生姓名列表。我们如何同时打印出值和键？

- Modify your code as follows:

  按如下方式修改代码：

```
students = {
    "Hermione": "Gryffindor",
    "Harry": "Gryffindor",
    "Ron": "Gryffindor",
    "Draco": "Slytherin",
}
for student in students:
    print(student, students[student])
```

  Notice how `students[student]` will go to each student's key and find the value of their house. Execute your code, and you'll notice how the output is a bit messy.

  注意 `students[student]` 将如何转到每个学生的密钥并找到他们房子的价值。执行您的代码，您会注意到输出有点混乱。

- We can clean up the print function by improving our code as follows:

  我们可以通过改进我们的代码来清理 print 函数，如下所示：

```
students = {
    "Hermione": "Gryffindor",
    "Harry": "Gryffindor",
    "Ron": "Gryffindor",
    "Draco": "Slytherin",
}
for student in students:
    print(student, students[student], sep=", ")
```

  Notice how this creates a clean separation of a `,` between each item printed.

  请注意，这如何在打印的每个项目之间创建 `,` 的清晰分隔。

- If you execute `python hogwarts.py`, you should see the following:

  如果您执行 `python hogwarts.py`，您应该会看到以下内容：

```
$ python hogwarts.py
Hermione, Gryffindor
Harry, Gryffindor
Ron, Gryffindor
Draco, Slytherin
```

- What if we have more information about our students? How could we associate more data with each of the students?

  如果我们有更多关于学生的信息怎么办？我们如何将更多数据与每个学生相关联？

	name	house	patronus
0	Hermione	Gryffindor	Otter
1	Harry	Gryffindor	Stag
2	Ron	Gryffindor	Jack Russell terrier

3	Draco	Slytherin	

- You can imagine wanting to have lots of data associated with multiple keys. Enhance your code as follows:

  你可以想象希望将大量数据与多个键相关联。按如下方式增强您的代码：

```
students = [
    {"name": "Hermione", "house": "Gryffindor", "patronus": "Otter"},
    {"name": "Harry", "house": "Gryffindor", "patronus": "Stag"},
    {"name": "Ron", "house": "Gryffindor", "patronus": "Jack Russell terrier"},
    {"name": "Draco", "house": "Slytherin", "patronus": None},
]
```

  Notice how this code creates a `list` of `dict`s. The `list` called `students` has four `dicts` within it: One for each student. Also, notice that Python has a special `None` designation where there is no value associated with a key.

  请注意此代码如何创建 `dict` 列表。名为 `students` 的列表包含四个字典：每个学生一个。另请注意，Python 有一个特殊的 `None` 指定，其中没有与键关联的值。

- Now, you have access to a whole host of interesting data about these students. Now, further modify your code as follows:

  现在，您可以访问有关这些学生的大量有趣数据。现在，进一步修改您的代码，如下所示：

```
students = [
    {"name": "Hermione", "house": "Gryffindor", "patronus": "Otter"},
    {"name": "Harry", "house": "Gryffindor", "patronus": "Stag"},
    {"name": "Ron", "house": "Gryffindor", "patronus": "Jack Russell terrier"},
    {"name": "Draco", "house": "Slytherin", "patronus": None},
]

for student in students:
    print(student["name"], student["house"], student["patronus"], sep=", ")
```

  Notice how the `for` loop will iterate through each of the `dict`s inside the `list` called `students`.

  请注意 `for` 循环将如何遍历名为 `students` 的列表中的每个 `dict`s。

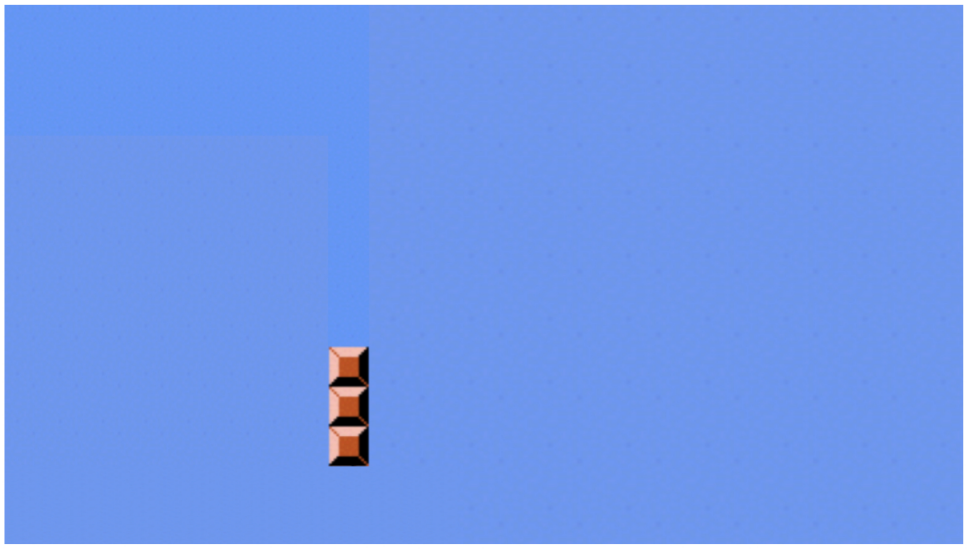
- You can learn more in Python's Documentation of `dict`s.

  您可以在 Python 的 `dict`s 文档中了解更多信息。

Mario 马里奥

- Remember that the classic game Mario has a hero jumping over bricks. Let's create a textual representation of this game.

  请记住，经典游戏马里奥有一个英雄跳过砖块。让我们创建此游戏的文本表示。



- Begin coding as follows: 按如下方式开始编码：

```
print("#")
print("#")
print("#")
```

  Notice how we are copying and pasting the same code over and over again.

  请注意我们如何一遍又一遍地复制和粘贴相同的代码。

- Consider how we could better the code as follows:

  考虑如何改进代码，如下所示：

```
for _ in range(3):
    print("#")
```

  Notice how this accomplishes essentially what we want to create.

  请注意这基本上是如何实现我们想要创建的。

- Consider: Could we further abstract for solving more sophisticated problems later with this code? Modify your code as follows:

  考虑一下：我们是否可以进一步抽象，以便稍后使用这段代码解决更复杂的问题？按如下方式修改代码：

```
def main():
```

```
    print_column(3)


def print_column(height):
    for _ in range(height):
        print("#")


main()
```

Notice how our column can grow as much as we want without any hard coding.

请注意，我们的列可以在没有任何硬编码的情况下随心所欲地增长。

- Now, let's try to print a row horizontally. Modify your code as follows:

  现在，让我们尝试水平打印一行。按如下方式修改代码：

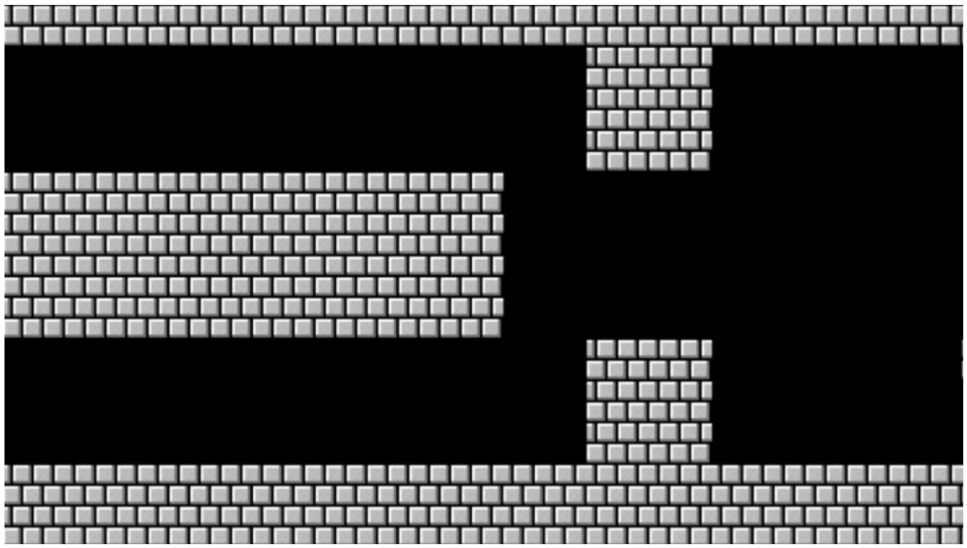
```
def main():
    print_row(4)


def print_row(width):
    print("?" * width)


main()
```

Notice how we now have code that can create left-to-right blocks.

请注意我们现在如何拥有可以创建从左到右块的代码。

- Examining the slide below, notice how Mario has both rows and columns of blocks.

  检查下面的幻灯片，请注意 Mario 如何同时具有行和列块。



- Consider: How could we implement both rows and columns within our code? Modify your code as follows:

  考虑一下：我们如何在代码中同时实现行和列？按如下方式修改代码：

```
def main():
    print_square(3)


def print_square(size):

    # For each row in square
    for i in range(size):

        # For each brick in row
        for j in range(size):

            #  Print brick
            print("#", end="")

        # Print blank line
        print()


main()
```

Notice that we have an outer loop that addresses each row in the square. Then, we have an inner loop that prints a brick in each row. Finally, we have a `print` statement that prints a blank line.

请注意，我们有一个外部循环，用于寻址正方形中的每一行。然后，我们有一个内循环，它在每行中打印一个 brick。最后，我们有一个打印空行的 `print` 语句。

- We can further abstract away our code:

  我们可以进一步抽象出我们的代码：

```
def main():
    print_square(3)


def print_square(size):
    for i in range(size):
        print_row(size)
```

```
def print_row(width):
    print("#" * width)


main()
```

Summing Up 总结

You now have another power in your growing list of your Python abilities. In this lecture, we addressed...

现在，在不断增长的 Python 能力列表中，您又多了一件大事。在本次讲座中，我们讨论了...

- Loops 循环
- `while` 而
- `for` 为
- `len` 莱恩
- `list` 列表
- `dict` dict （字典）