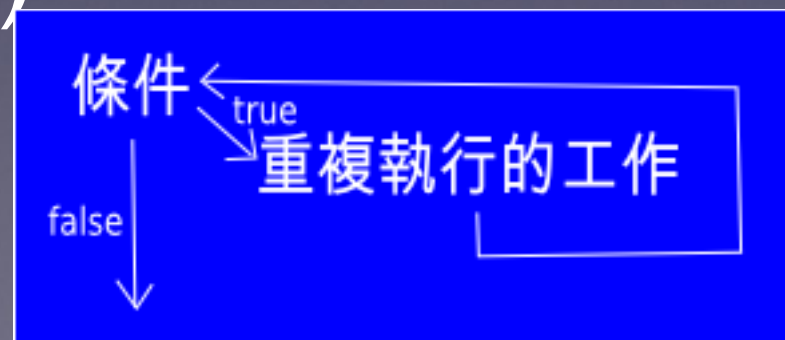


Matlab 概論

Python 迴圈

Brian Wang 王昱景
brian.wang.frontline@gmail.com

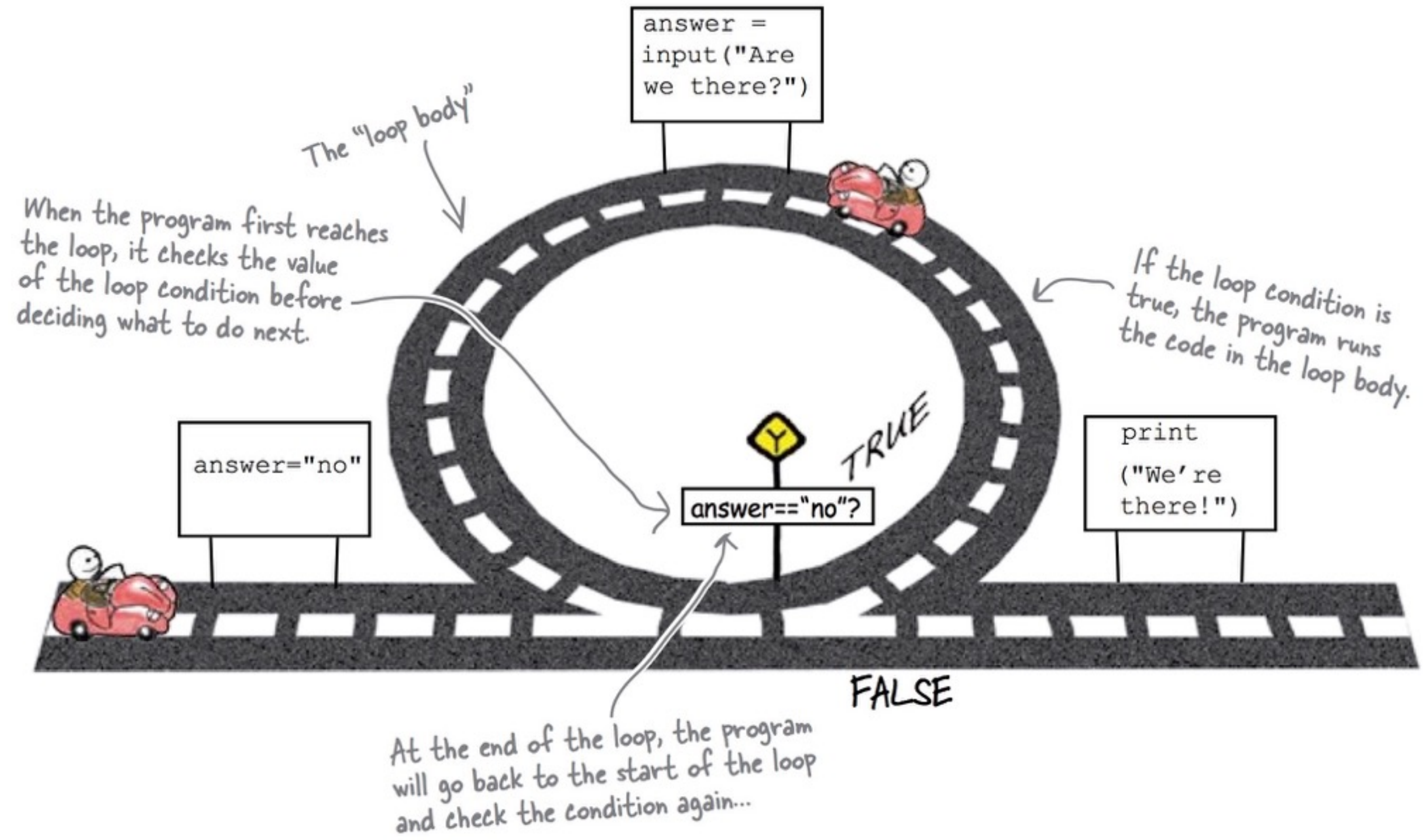
- 當有工作需要重複執行時，重複的條件為真，此工作就重複執行一次，執行完畢再行測試條件，條件為真，繼續重複執行此工作，直到條件為假時，結束工作的執行
- 依此執行方式為重複結構，也被稱為迴圈 (loop)



Loops let you run the same piece of code over and over again

Programs often need to keep running the same piece of code many times. In addition to branches, programming languages also provide **loops**.

Loops are a little like branches. Just like branches, loops have a condition (the **loop condition**) that is either true or false. Also, like the `if` part of branches, if the loop condition is true, then a loop will run a given piece of code. For a branch, this code is called the **body**. For a loop, it's called the **loop body**.

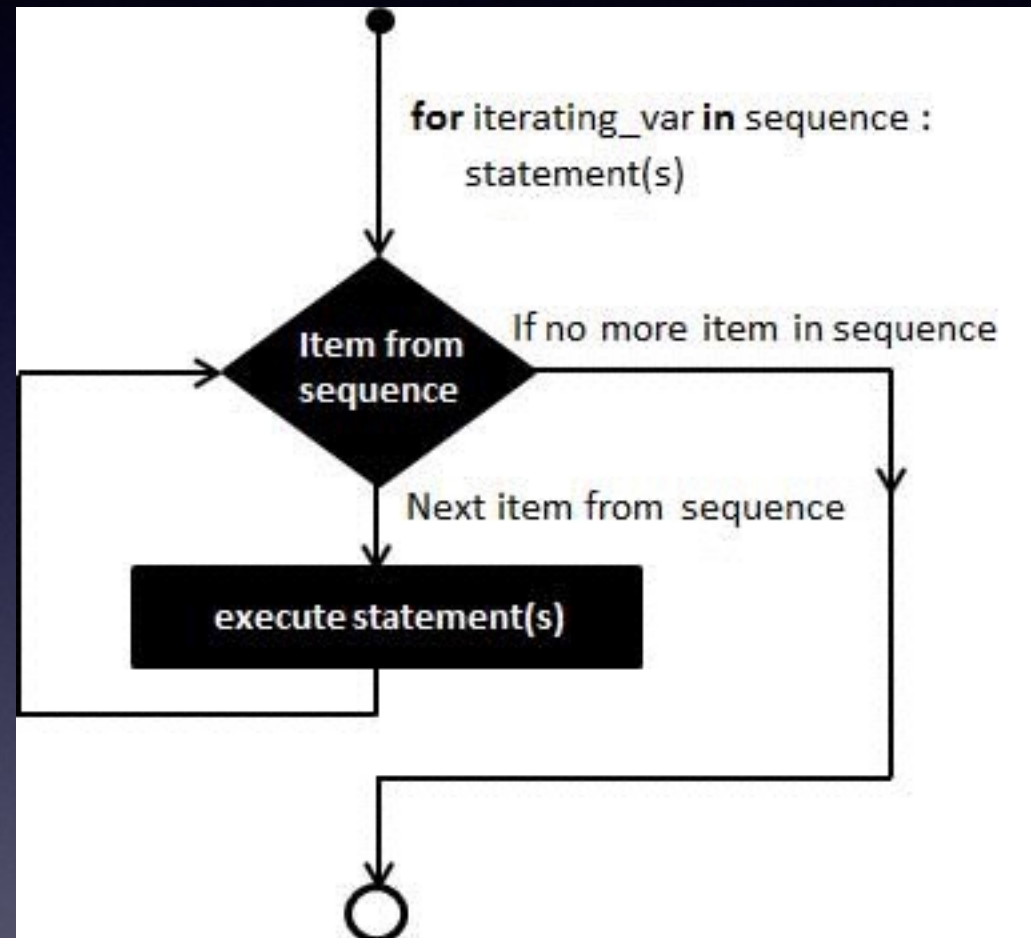


The big difference between a loop and a branch is **how many times** it runs the code associated with it. A branch will run its code only once. But a loop will run the loop body, then check the loop condition again and, if it's still true, it will run the loop body again. And again. And again. In fact, it will keep running the loop body until the loop condition becomes **false**.

for...in 迴圈

- for...in 迴圈，一般也會簡稱為 for 迴圈
- 通常來處理有次序性且已知應該執行次數的問題

- 在執行 for 迴圈時需要下列三個條件：
 - 設定控制變數的初始值
 - 設定控制變數的終止值，也就是迴圈的結束條件
 - 設定控制變數值的變動方向與量，也就是迴圈的迭代
- for 迴圈只需要取得元素的控制變數 (variable)，其餘條件 (condition) 測試與調整控制變數的部份，迴圈中會自動完成



- 典型的 for 迴圈如下應用

```
1  a = "abcdefghijklmnopqrstuvwxyz"
2  i = 0
3
4  for x in a:
5      print(x, end=" ")
6      i += 1
7
8      if i % 10 == 0:
9          print()
10
11 print()
```

- 此例印出 26 個英文小寫字母，資料儲存在第 1 行建立的變數 a 之中
- 第 2 行的變數 i，用來計數，每印 10 個字母便印出新行符號，因此出值設為 0
- for 迴圈從第 4 行到第 9 行

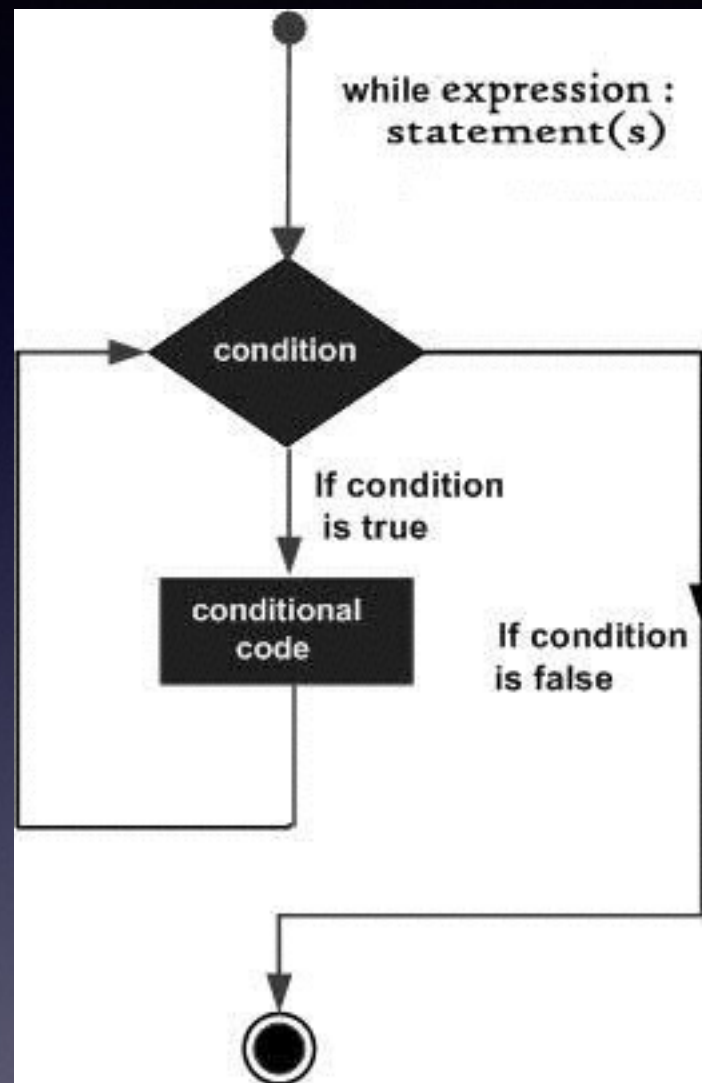
- 第 5 行的 `print()` 函數 (function) ，我們多增加一個預設的參數 (parameter) `end` ，這是指定印出 `x` 之後的結尾符號，如沒有指定，就會印出新行
- 第 8 行的 `if` 陳述 (if statement) ，我們取 `i` 除以 10 的餘數是否等於 0 ，也就是說 `i` 從 0 一路遞增到 9 印出 10 個英文字母後，當 `i` 遞增到 10 或 10 的倍數，此時程式就會印出新行，因此一行會印 10 個英文字母

while 迴圈

- while 迴圈比較適合拿來處理沒有次序性，或者不知道需要執行幾次的問題
- while 後到冒號間的運算式 (expression) 為迴圈結束的條件 (condition) 測試，即為迴圈開始前進行迴圈結束條件的測試，形式如下
- while condition:

- 由於 while 陳述僅需迴圈的結束條件測試，所以有關控制變數 (variable) 的初始設定及調整，這都需要放在其它地方
- while 迴圈在使用的彈性較大，for 迴圈可以做到的，while 迴圈也都能，不過在能夠使用 for 迴圈的行情下，還是建議使用 for 迴圈就好

- 因為 for 迴圈是特別設計於處理較制式化的問題，故語法通常較為便捷
- 在 while 迴圈因為需要手動加上迭代的條件，因此程式的行數比使用 for 迴圈會來得多



- 下例計算 1 到 100 所有整數的和，示範 while 迴圈的簡單使用

```
1 sum = 0
2 i = 1
3 while i <= 100:
4     sum += i
5     i += 1
6
7 print("1 + 2 + 3 + ... + 98 + 99 + 100 =", sum)
```

巢狀迴圈

- 迴圈 (loop) 也可以是巢狀的 (nested) ，所謂巢狀的迴圈是指迴圈中包含其他的迴圈
- 在巢狀迴圈中，內部迴圈的執行是在外部迴圈之內，如果內部迴圈有使用到外部迴圈中有更動的變數，則內部迴圈會隨著外部迴圈的更動而改變

- 下例程式印出九九乘法表，就是利用兩個 for 迴圈，一個 for 迴圈之中包含另一個 for 迴圈

```
1  for x in range(10):  
2      for y in range(10):  
3          print(x * y, end=" ")  
4      print()
```

- `range()` 為內建函數，會自動產生一個從 0 開始到參數減 1 的串列 (list)，因此 `for x in range(10):` 就表示 `x` 會從 0 一路遞增到 `10 - 1`，也就是 9

```
1 i = 1
2 j = 1
3 while i <= 9:
4     while j <= 9:
5         print(i * j, end = " ")
6         j += 1
7     i += 1
8     j = 1
9     print()
```


break & continue

- 若利用關鍵字 break 則可以跳出迴圈，而利用關鍵字 continue 則會跳過迴圈這一輪的執行
- 下例在印出九九乘法表的程式中加入了 break 與 continue 陳述

```
1  for x in range(10):
2      if x == 0:
3          continue
4      elif x == 8:
5          break
6
7      for y in range(10):
8          if y == 0:
9              continue
10             print(x * y, end=" ")
11         print()
```

- 由於 x 等於 0 與 y 等於 0 的時候都使用 `continue` 陳述，因此 0 列與 0 行不會被印出
- 此外， x 等於 8 時使用 `break` 陳述跳出迴圈，結束印出乘法表的迴圈

```
1 i = 1
2 j = 1
3 while i <= 9:
4     if i == 8:
5         break
6
7     while j <= 9:
8         if j == 4:
9             j += 1
10            continue
11
12            print(i * j, end = " ")
13            j += 1
14
15        i += 1
16        j = 1
17        print()
```


- 第 5 行使用 `break` 陳述，因此只印到 7 為止
- 此外，第 10 行使用 `continue` 陳述，因此乘法表的 4 不會被印出



Why do I have to keep rerunning the program? You mean I only get **one guess**????

The users still don't like it.

The program works, and now generates extra feedback, but there's a problem. If the users want to have another guess, they have to run the program again. They *really* want the program to keep asking them for another guess until they finally get the correct answer.

Can you see what the problem is?

How do we get the computer to do something repeatedly? Should we just make a copy of the code and paste it at the end of the file? That would make sure the user is asked twice. But what if they need to make **3 guesses**? Or **4 guesses**? Or **10,000 guesses**? What about the case where the guess is correct?

The guessing game program needs to be able to run *some* code repeatedly.



Ready Bake Code

If you add these two lines of code to the top of your program:

```
from random import randint  
secret = randint(1, 10)
```

The `secret` variable will be set to a random number between 1 and 10. Modify your program from the facing page so that instead of the answer always being 5, it will instead use a random number from 1 to 10 as the answer.

Write the next version of your program here. This version of your program uses the value of the "secret" variable as the correct answer.



.....

.....

.....

.....

Long Exercise Solution

You needed to rewrite your game program so it keeps running until the user guesses the correct answer. You needed to use all of the things you've learned in this chapter. You needed to work out the conditions for each of the branches and loops that are required.

Hint: If you need to test that two things have different values, use the `!=` operator.

Did you remember to set the guess to some sensible default value to make sure the loop ran the first time?

```
print("Welcome!")
```

```
guess = 0
```

We need to keep running while the guess is wrong.

```
while guess != 5:
```

```
    g = input("Guess the number: ")
```

```
    guess = int(g)
```

```
    if guess == 5:
```

```
        print("You win!")
```

```
    else:
```

```
        if guess > 5:
```

```
            print("Too high")
```

```
        else:
```

```
            print("Too low")
```

```
print("Game over!")
```

All of this code is indented, which means it's all inside the loop body.

Don't worry if your code doesn't look exactly like this. The important thing is that it works in the same way when you run it.

This part of the program is very similar to what you had before.



Ready Bake Code

If you add these two lines of code to the top of your program:

```
from random import randint  
secret = randint(1, 10)
```

The `secret` variable will be set to a random number between 1 and 10. You were to modify your program from the facing page so that instead of the answer always being 5, it will instead use a random number from 1 to 10 as the answer.

Here are the two
lines that create the
random number.

→ `from random import randint`

→ `secret = randint(1, 10)`

`print("Welcome!")`

`guess = 0`

→ `while guess != secret:`

`g = input("Guess the number: ")`

`guess = int(g)`

→ `if guess == secret:`

`print("You win!")`

`else:`

→ `if guess > secret:`

`print("Too high")`

`else:`

`print("Too low")`

`print("Game over!")`

Now, instead of checking if the
answer's 5, we check it against
the random number, which is held
in the "secret" variable.



Test Drive

So, what happens when you run the new version of your program?

```
Python 3.0.1 (r301:69556, Feb 17 2009, 15:15:57)
[GCC 4.3.2] on linux2
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
--
>>>
Welcome!
Guess the number: 3
Too low
Guess the number: 9
Too high
Guess the number: 4
Too low
Guess the number: 7
Too low
Guess the number: 8
You win!
Game over!
>>> |
```

The program keeps asking for another guess for as long as you keep getting the answer wrong.

The correct answer is now a random number and it should be different each time you play the game.

The program stops when you get the answer right.

Ln: 18 Col: 4