Matlab概論

Python 判斷

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

- 在沒有特殊情形下，Python 會從程式碼的第一行開始執行，依序向下
- 特殊情況有以下四種：
  - 判斷式：當程式執行到 if 判斷式時，Python 會判斷現在的情形是否符合程式所設定的條件，可能會因條件不符合而發生跳過不執行某些區塊程式

- 迴圈：迴圈內的程式是可能要重複執行的，有 for 迴圈跟 while 迴圈兩種類型，當程式執行到迴圈時會根據程式所特定的條件做判斷而重複執行
- 函數：函數是為了完成特定功能而撰寫，有可以被重複呼叫的特性，當程式執行時遇到呼叫函數，會去尋找此函數的完整程式，利用程式呼叫函數時傳入的參數，進而執行該函數，在這種情形產生跳躍式執行程式

- 例外：這裡所說的例外就是錯誤，出現這種情形如果沒有妥善處理，程式的執行就會強制中斷

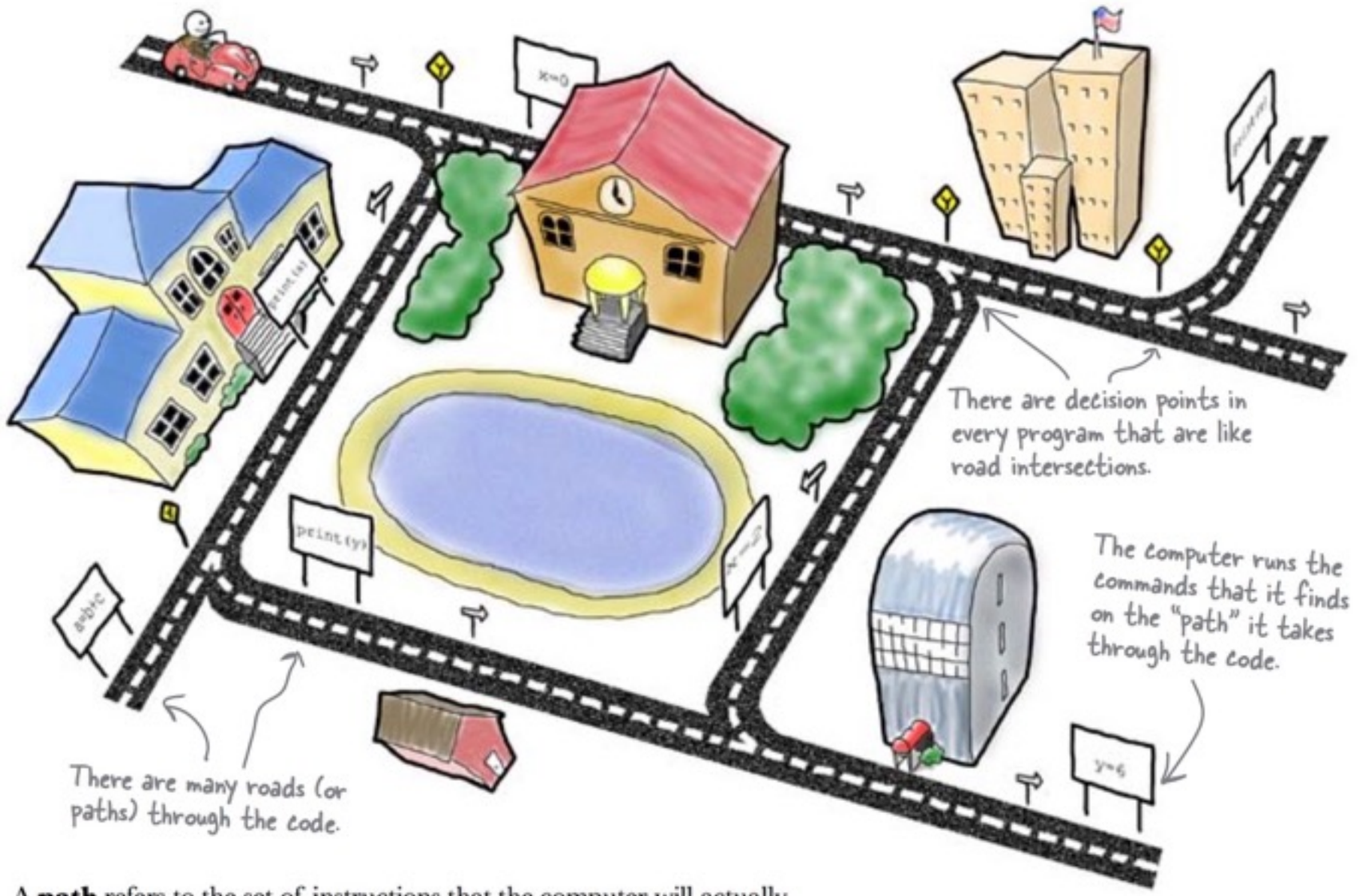
A program is more than a list of commands

You *could* create a program that was simply a **list** of commands. But you almost never will. This is because a simple list of commands can only be run in one direction. It's just like driving down a straight piece of road: there's really only one way of doing it.

But programs need to be much smarter than that.

Codeville: Your program is like a network of roads

Programs need to do different things under different circumstances. In the game, the code displays "You win!" if the user guesses the number correctly, and "You lose!" if not. This means that all programs, even really simple programs, typically have multiple **paths** through them.



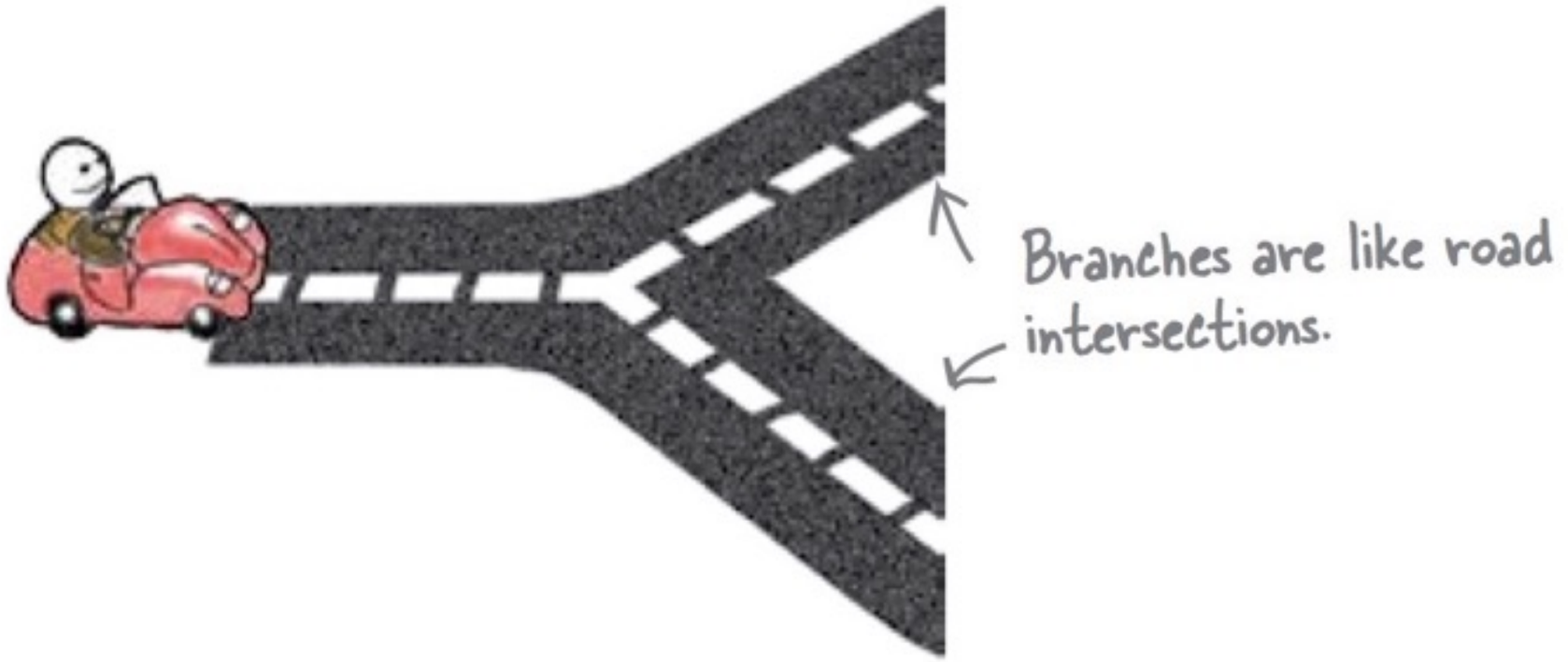
A **path** refers to the set of instructions that the computer will actually follow (or execute). Your code is like a street network, with lots of sections of code connected together just like the streets in a city. When you drive through a city, you make decisions as to which streets you drive down by turning left or right at different intersections. It's the same for a program. It also needs to make decisions from time to time as to which path to take, but for your code, it is not like driving along a road, *it's executing a particular path.*

Let's look in more detail at how a program decides which path to take.

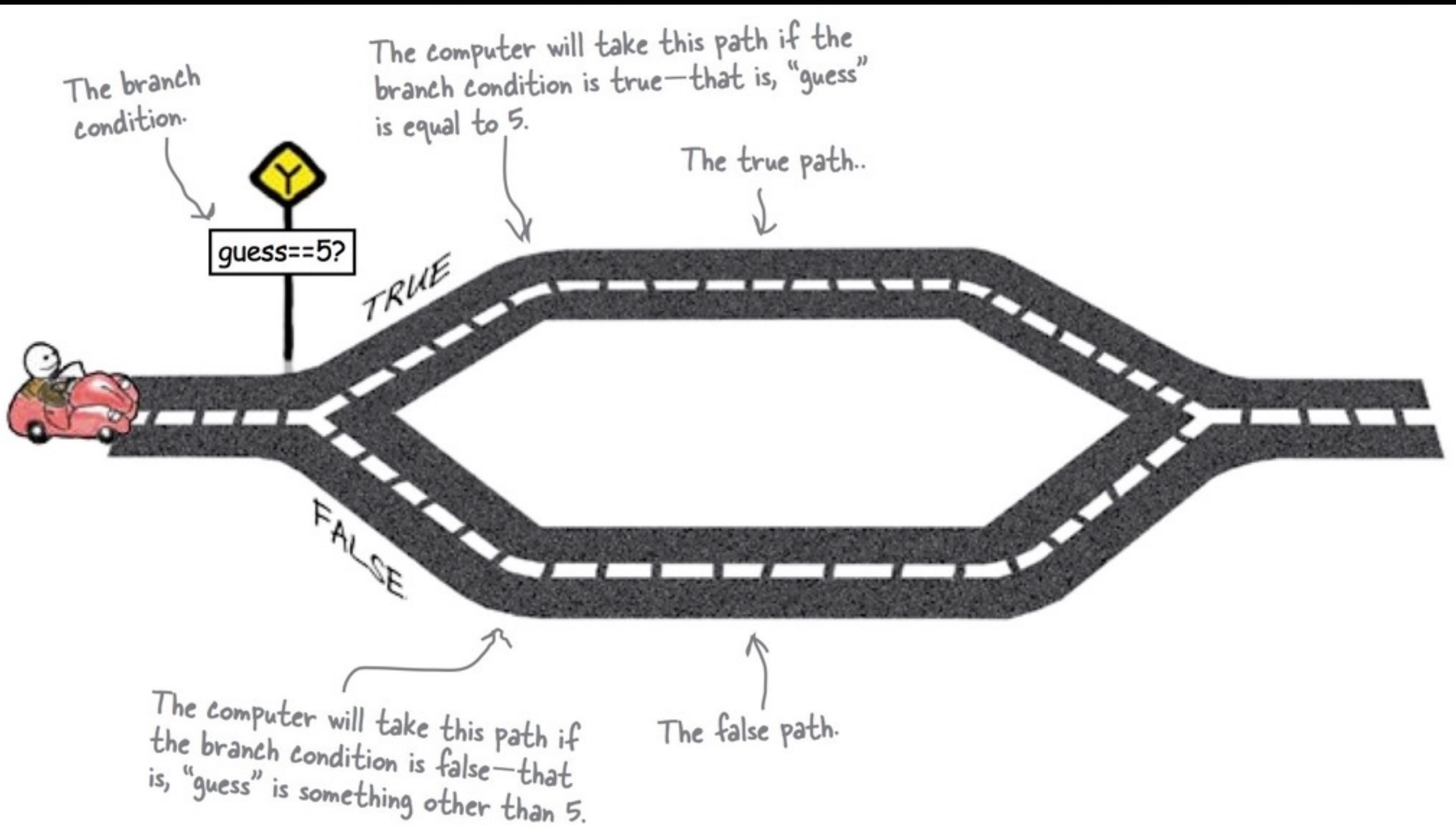
Branches are code intersections

Driving down a street is easy. You need to make a decision only when you get to an intersection. It's the same for your program. When a program has a list of commands, it can blindly execute them one after another. But sometimes, your program needs to make a decision. Does it run ***this*** piece of code or ***that*** piece of code?

These decision points are called **branches**, and they are the road intersections in your code.



Your program makes a decision using a **branch condition**. A branch condition has the value **true** or **false**. If the branch condition is true, it runs the code on the true branch. And if the branch condition is false, it runs the code on the false branch.

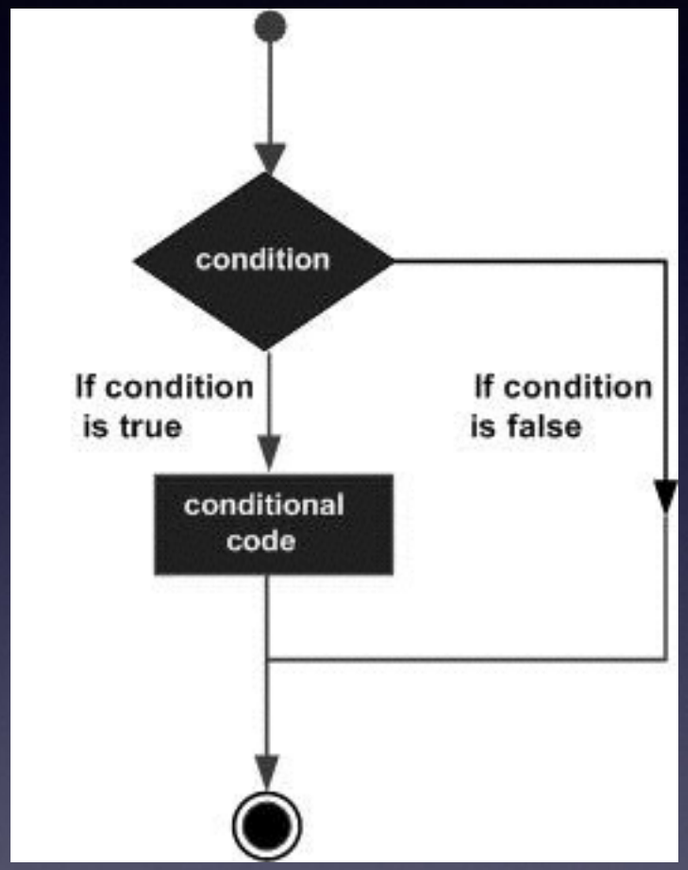

The branch condition.
The computer will take this path if the branch condition is true—that is, "guess" is equal to 5.
The true path..
guess==5?
TRUE
FALSE
The computer will take this path if the branch condition is false—that is, "guess" is something other than 5.
The false path.

單向判斷（if...）

- Python 中最簡單的選擇結構 (selection structure) 為單一個 if 陳述 (if statement)
- 在關鍵字 (keyword) if 後到冒號間的運算式 (expression) 即為條件 (condition)
- 形式如下

```
if condition:
```

- 若運算式的結果為 True ， 也就是條件為真，其後縮排 (indentation) 的程式區塊 (block) 便會執行
- 而若運算式的結果為 False ，這樣一來條件就為假，程式會跳過縮排的程式區塊，沒有縮排的第一個陳述 (statement) 執行


condition
If condition
is true
If condition
is false
conditional
code

- 以下程式示範使用單一的 if 陳述

```
a = "h"
b = "h"

if a == b:
    print("Welcome to my world!")
```

- 程式的第 4 行由於 a 及 b 兩個變數 (variable) 連到相同的物件 (object) ，所以條件為真， if 陳述後頭縮排的陳述便會執行

雙向判斷（if...else）

- if 可以跟關鍵字 else 連用，這樣一來， if 的條件為假時，程式跳過 if 陳述後頭縮排的程式碼，而去找 else 後頭縮排的程式碼來執行

- 以下程式示範 if-else 陳述的使用

```
a = "h"
b = "k"

if a == b:
    print("Welcome to my world!")
else:
    print("How do you do?")
```

- 此例中，由於第 4 行變數 a 不等於變數 b ，所以程式會跳到執行第 6 行去執行 else 的部份

巣状判断

- if-else 也可以是巢狀 (nested) 的使用，例如

```
1   a = "h"
2   b = "k"
3
4   if a == "a":
5       print("Yes!")
6   else:
7       if a == b:
8           print("No!")
9       else:
10          print("What?")
```

- 所謂巢狀的 if-else 陳述就是形如第 4 到 10 行的地方

多向判斷
(if...elif...else)

- Python 利用另一個關鍵字 elif 使多重選擇的寫法較為簡單，如下例

```
a = "h"
b = "k"

if a == "a":
    print("Yes!")
elif a == b:
    print("No!")
else:
    print("What?")
```

- if-elif-else 陳述中， else 可有可無，若加入 else 陳述， else 就會是以上皆非的預設選項

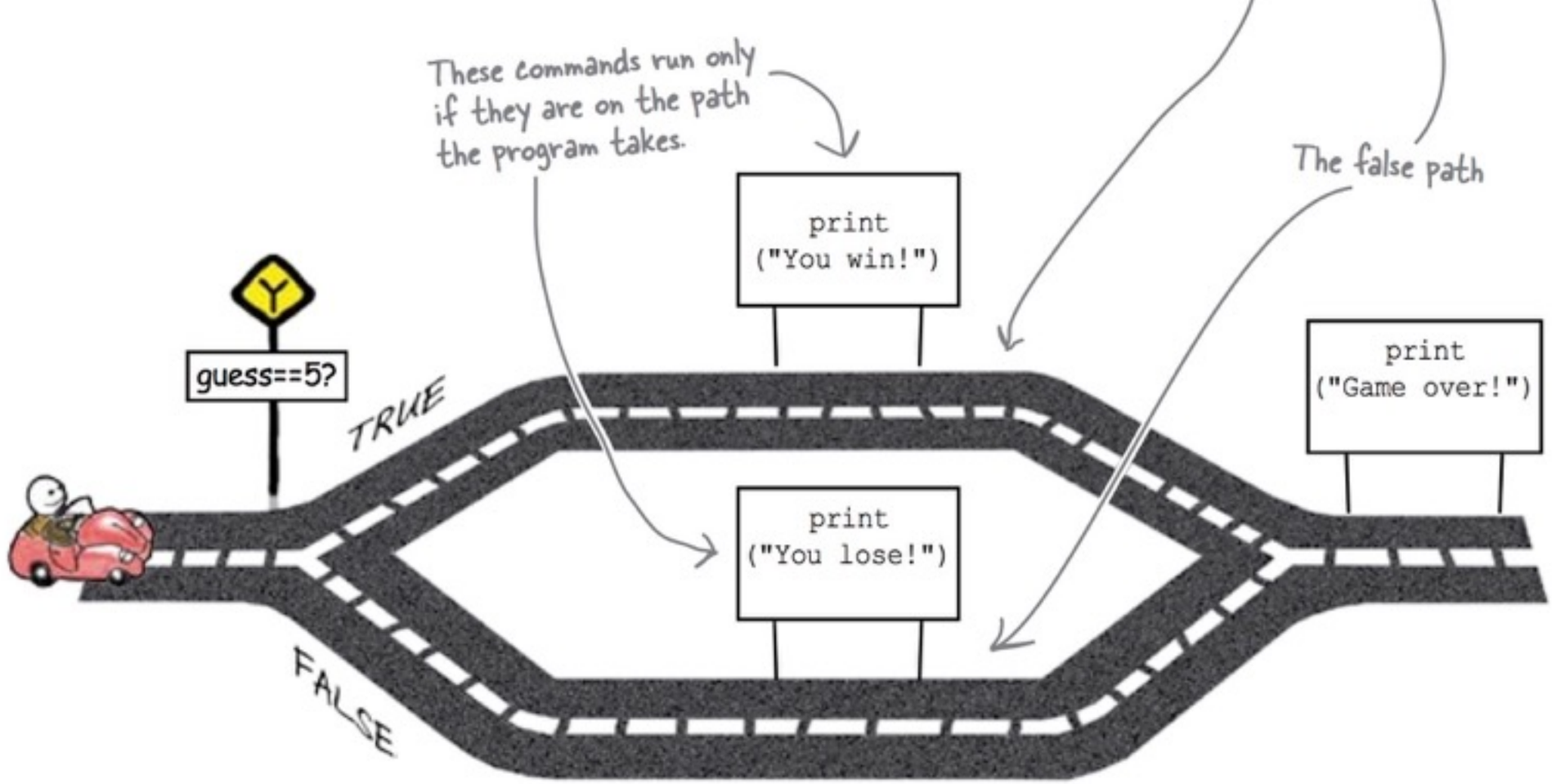
if/else branches

We've already seen a branch in the Python game program:

```
if guess == 5:
    print("You win!")
else:
    print("You lose!")
print("Game over!")
```

Python, like many languages, has `if/else` branches. In our example, the branch condition is the `if guess == 5` piece of code. This is a **test for equality** and it will result in the value **true** or **false**.

The code on the true path is indented, and given after the **if** line. The code on the false path is indented, and given after the **else** line:

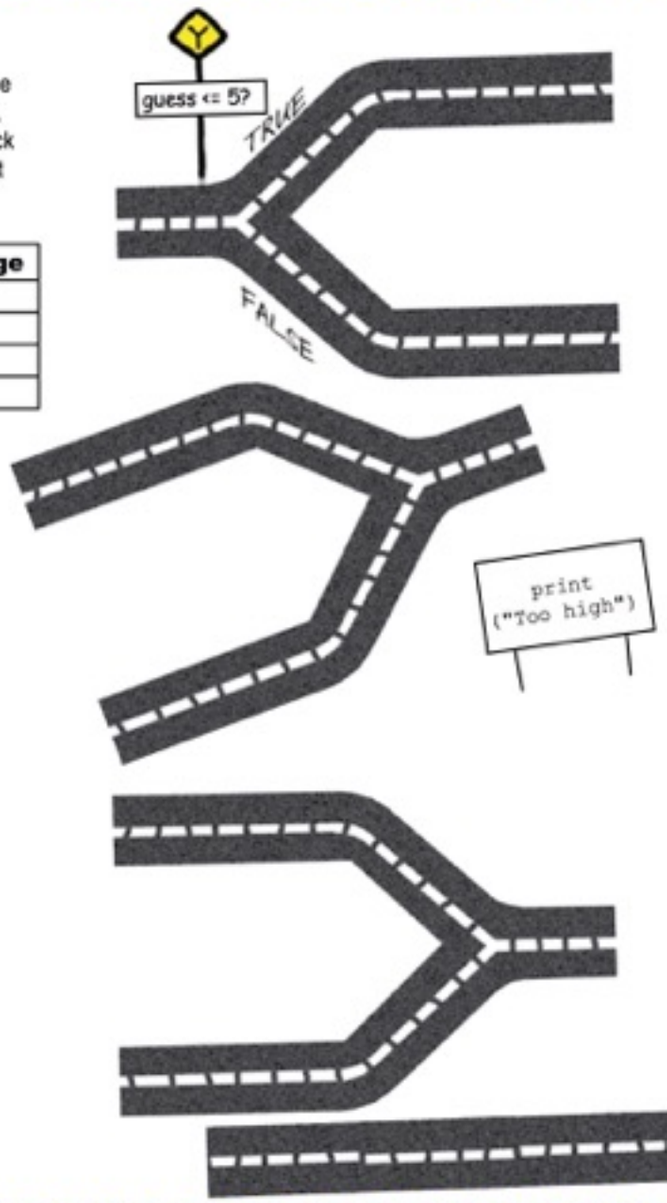


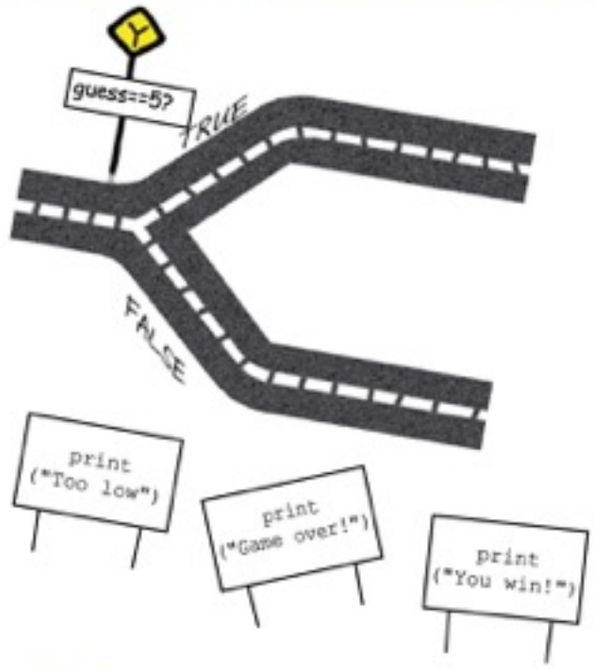
Race Track Construction Kit

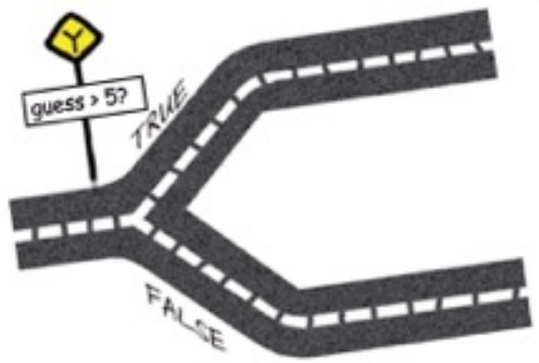
The countdown's started on the Codeville Grand Prix. The cars have arrived, they're warming their tires on the grid, and the race is about to start. Can you assemble the track so that it displays the right feedback message? Note that you might not need all the pieces of track.

Number entered	Feedback message
3	Too low
5	You win!
7	Too high
8	Too high

The race start line is fixed here. START





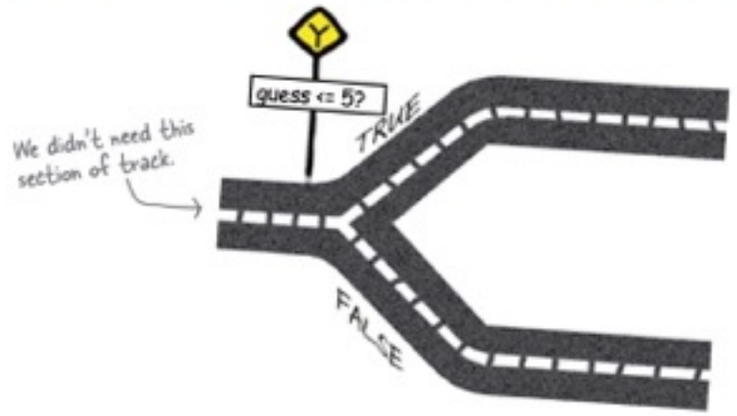


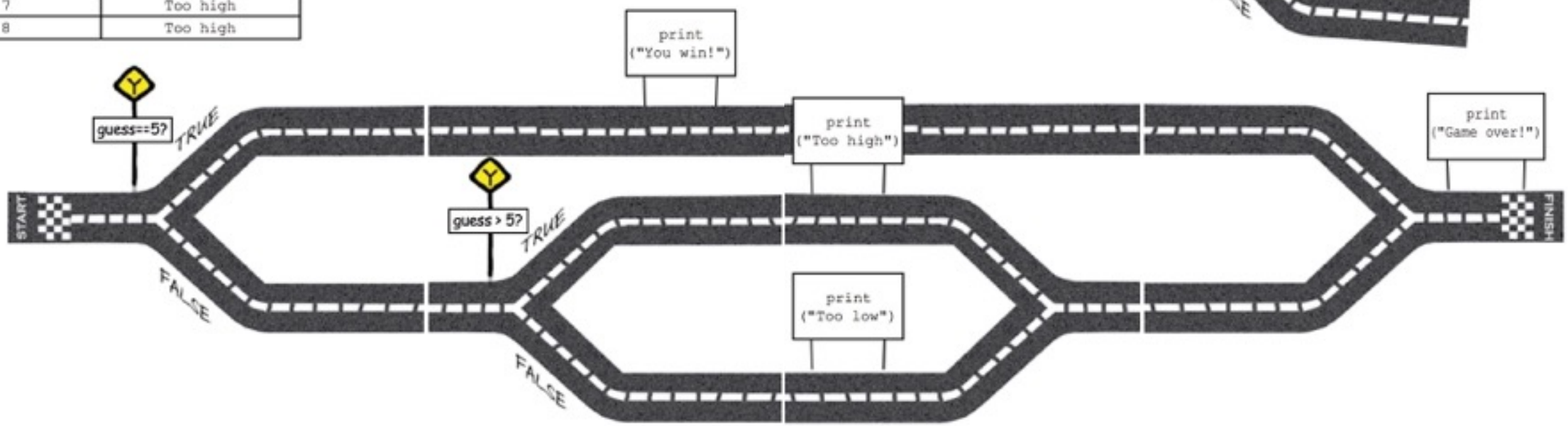
FINISH The race finish line is fixed here.

Race Track Construction Kit Solution

The countdown's started on the Codeville Grand Prix. The cars have arrived, they're warming their tires on the grid, and the race is about to start. Were you able to assemble the track so that it displays the right feedback message?

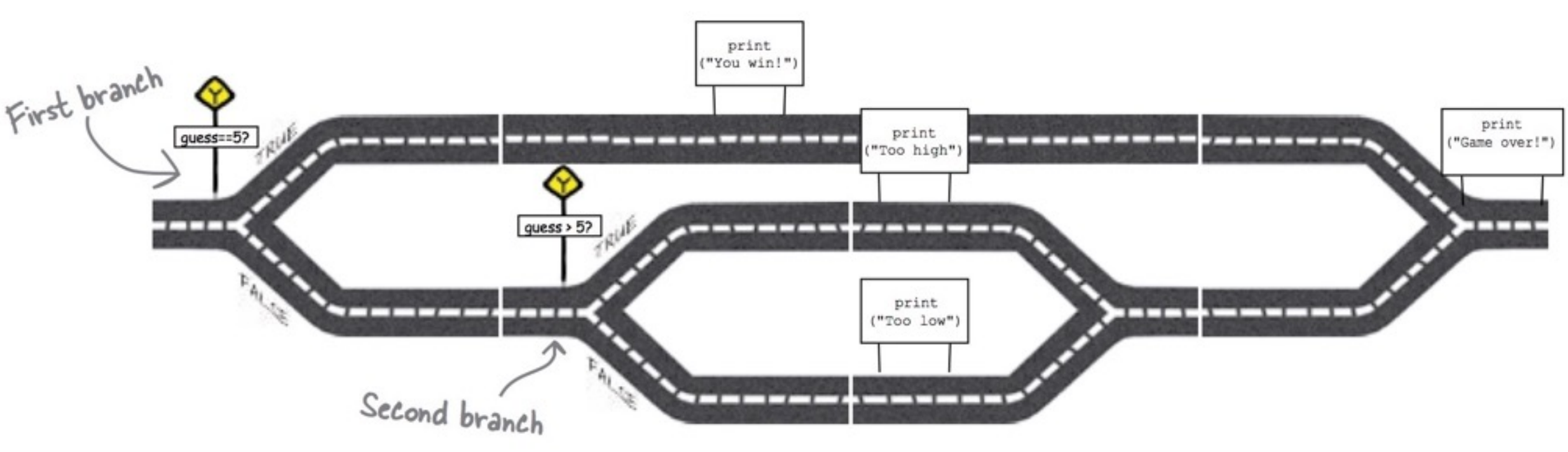
Number entered	Feedback message
3	Too low
5	You win!
7	Too high
8	Too high





The Python code needs interconnecting paths

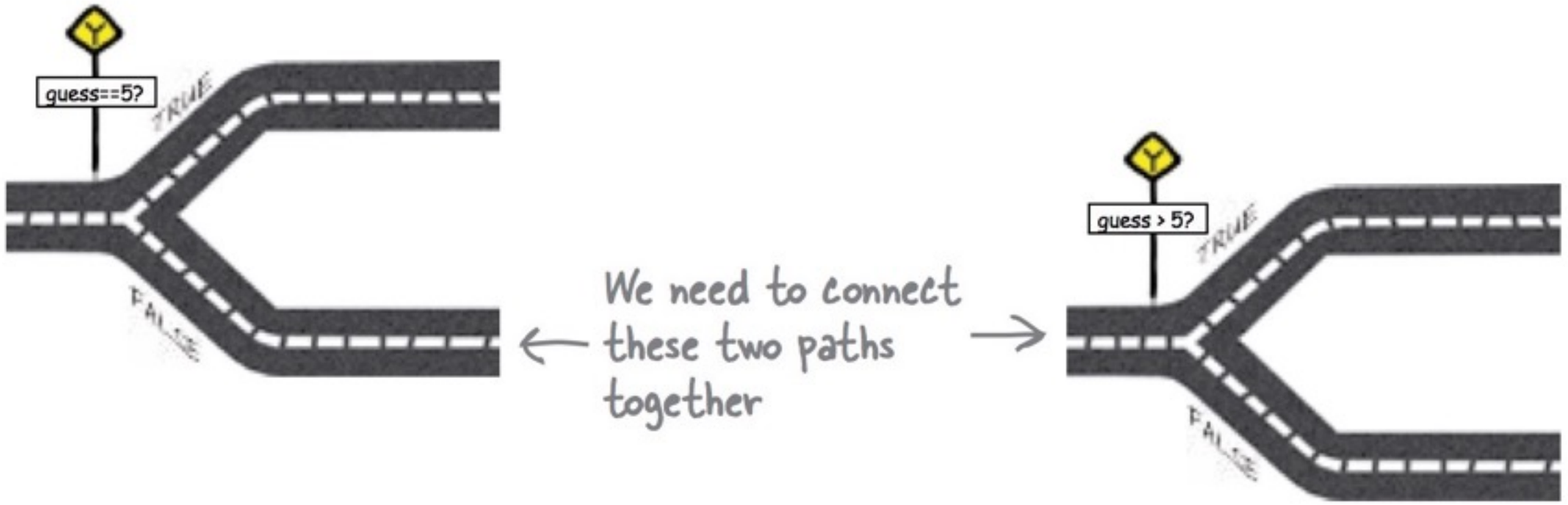
The solution's mapped out, and now we know that the program code will need to have paths that match this:



But isn't there a problem here? In the design there are many *interconnecting* paths, but so far, we have only written code that contains just **one** branch:

```
if guess == 5:
    print("You win!")
else:
    print("You lose!")
```

In the new code, we will need to *connect two branches together*. We need the second branch to appear on the **false path** of the first.



So how do you connect branches together in Python?

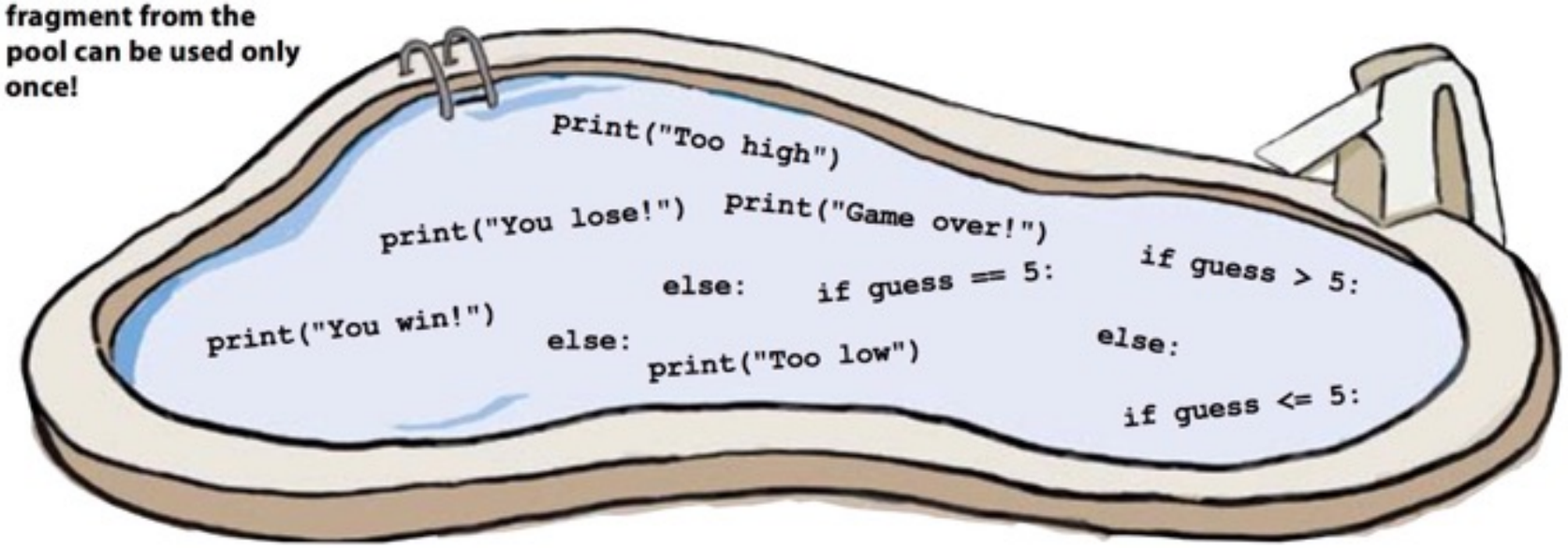
Pool Puzzle

Your **task** is to take the Python code fragments from the pool and place them into the blank lines in the game. You may **not** use the same code fragment more than once, and you won't need to use all the code fragments. Your **goal** is to complete the guessing game program.

Hint: Don't forget to indent.

```
print("Welcome!")
g = input("Guess the number: ")
guess = int(g)

.....

.....

.....

.....

.....

.....

.....

.....
```

Note: each code fragment from the pool can be used only once!



Pool Puzzle Solution

Your **task** was to take the Python code fragments from the pool and place them into the blank lines in the game. You could **not** use the same code fragment more than once, and you didn't need to use all the code fragments. Your **goal** was to complete the guessing game program.

Hint: Don't forget to indent.

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

Did you remember to indent your code deep enough?

All of this code is indented under the else part of the original if part.

This piece of code from the first version of this program is no longer needed.

Test Drive

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

Let's try a few tests. Remember, you will need to switch back to the program window for each run and choose **`Run module`** from the menu.

Python Shell

File Edit Shell Debug Options Windows Help

```
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
Game over!
>>> ================================ RESTART ================================
>>>
Welcome!
Guess the number: 7
Too high
Game over!
>>> ================================ RESTART ================================
>>>
Welcome!
Guess the number: 5
You win!
Game over!
>>>
```

Looks like the first guess was too low, so we need to try again.

Now the guess is too high. Let's have another go, and...

...success! We've guessed the correct answer.

Ln: 22 Col: 4