

[< Previous](#)[Next >](#)

Unit 1 / Lesson 1 / Assignment 5

Controlling the Code Flow

Estimated Time: 2-3 hours

Now that you understand the basic data types and their operations, it's time to learn about the flow of execution in a program. When a Python script is run, it evaluates code from top to bottom. If you have a list of `print()` statements, for example, Python will always execute them in the same order. However, quite often you want to change the order in which the program executes. This is achieved by using the control flow operations.



If you understand this sentence, then you already understand control flow. Congratulations! Control flow is just a combination of conditional statements. "If this, then that, or else this other thing." And so on and so on. We use conditionals all the time in our everyday dialogue, you just need to realize that conditionals in programming are basically the same thing.

There are four fundamental control flow statements: `if` , `for` , `while` and `try` . Let's take a look at each of these.

The `if` Statement

The Python `if` is the same as the English `if` in semantics, but not syntax. It means the same thing, but you'll need to understand the structure in order to use it correctly in your program. Take a look at the following code:

```
if 5 > 3:
    print("Yep, math works today.")
```

This reads almost the same as English. If 5 is greater than 3 (which it is) then Python will print the line "Yep, math works today." Note that the colon at the end of the `if` statement tells Python that a block of statements follows. Try typing this code into your Python interpreter to make sure math works today. Be sure to indent the print call with one or more spaces/tabs (generally either four spaces or one tab), and when you're done, press Enter one more time to tell the interactive interpreter to proceed.

But what if the conditional is not satisfied? In the above code, if 5 is actually less than 3, the program wouldn't do anything. However, you can give it an `else` statement to control for this:

```
if 5 < 3:
    print("Things might be a little off...")
else:
    print("Yep, math works today.")
```

This code will still print `Yep, math works today.` You can also combine the `else` and `if` statements to make an `elif` conditional:

```
if 5 < 3:
    print("Things might be a little off...")
elif 5 == 3:
    print("Maybe we should stay inside.")
else:
    print("Yep, math works today.")
```

Of course, this code will still return `Yep, math works today.` because math always works.

Try it!

Write if statements which tell you which record label the popular beat combo [Ted Leo and the Pharmacists](#) were signed to at a certain year. Note: 2001-2006 was Lookout Records, 2007-2009 was Touch and Go Records, and 2010- is Matador Records.

The for Statement

A `for` loop allows you to repeatedly run some block of code over a list, a range, or any other kind of sequence. It *iterates* over a series of things. Take a look:

```
>>> beatles = ("John", "Paul", "George", "Ringo")
>>> for beatle in beatles:
...     print(beatle)
...
John
Paul
George
Ringo
```

Let's make a (slightly) more realistic example. Let's say you want to know all of the numbers less than 100 that are divisible by 3. For this you can nest an `if` statement inside of a `for` loop. This is called a *nested if statement*, naturally.

```
for n in range(1,100):
    if n % 3 == 0:
        print(n)
```

Voila! Math still works!

Try it!

Loop over the following dictionary, printing out the name of the actor, and the character which they play:

```
actors = {  
    "Kyle MacLachlan": "Dale Cooper",  
    "Sheryl Lee": "Laura Palmer",  
    "Lara Flynn Boyle": "Donna Hayward",  
    "Sherilyn Fenn" : "Audrey Horne"  
}
```

The while Statement

If you want to repeatedly do something as long as a condition is true, then you should use a `while` statement. It also creates a loop, but instead of looping over a sequence, it loops *while* something is true.

```
miles_run = 0  
running = True  
  
while running:  
    if miles_run <= 10:  
        print("Still running! On mile {}".format(miles_run))  
        miles_run += 1  
    else:  
        running = False  
  
print("Whew! I'm tired")
```

This program will print `Still running!` eleven times, one for each mile, and then print `Whew! I'm tired`. The `+=` operator simply increases `miles_run` by one. So, in plain English, the code above:

1. Sets up two variables, `miles_run` and `running`
2. Starts a while loop, which checks to see if you're still running (if `running == True`)
3. Then it checks if `miles_run` is less than or equal to 10. If so, it prints `Still running!` with the current mile and increases `miles_run` by one.

4. If you have run 10 miles, then the `while` loop exits, and the last `print` statement is executed, which prints `Whew! I'm tired.`

That's only 4 things. See? Very simple!

Try it!

Use a `while` loop to solve the following problem: A slow, but determined, walker sets off from Leicester to cover the 102 miles to London at 2 miles per hour. Another walker sets off from London heading to Leicester going at 1 mile per hour. Where do they meet?

The `try` Statement

Unless you are an excellent typist you've probably already made a mistake in your code somewhere and seen Python throw an error message. The `try` statement allow you to catch error conditions and handle them cleanly.

For example, imagine you are writing a calculator application, and your user enters `1 / 0`. When we try to run this in Python an exception is thrown:

```
>>> 1 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

Rather than have our calculator app fail in this way we'd rather print a nice friendly error message. So we use a `try` block to catch the `ZeroDivisionError`:

```
a = 1
b = 0
try:
    a / b
except ZeroDivisionError:
    print("Cannot divide by zero.")
```

Here we *try* to divide two numbers. In the *exceptional* case that we try to divide by zero we print an error message.

Try it! (Pun alert...)

Try looking up Jamie Theakston in the following phone book. When it fails, catch the exception and print an appropriate error message.

```
phone_book = {  
    "Sarah Hughes": "01234 567890",  
    "Tim Taylor": "02345 678901",  
    "Sam Smith": "03456 789012"  
}
```

Final Thoughts

Control flow can get complicated, but since you are already familiar with conditionals in English, it's rather easy to relate this knowledge to Python. The hardest part is the remembering syntax, but thankfully Python has a relatively simple syntax, and there are plenty of reference material (including this curriculum!) online, in addition to any notes you take yourself.



· [Report a typo or other issue](#)

Completed



Previous

Next

