`if` statements

A regular `if` statement allows the programmer to include a group of statements that Python will only execute if a certain condition is true. If the condition is false, the extra group of statements is ignored.

```
if condition:
   statement
   statement
   [ more statements if you want ]
```

When Python gets to an **`if`** statement in your code, Python will examine the "condition," which is a piece of code you put after the word **`if`**. If the condition is true, all of the statements indented after the **`if`** line are executed. If the condition is false, all of the statements indented after the **`if`** line are ignored, as if they hadn't been written at all. In this situation, your program will find the next line of code after the **`if`** that is not indented, and your program will pick up from that point.

What does a condition look like?

A condition is any Python expression that can be interpreted as "true" or "false." Conditions often involve comparisons between two things, such as:

```
x > 2          (x + y) < z          name == "Sam"        revenue >= costs
```

Imagine someone approaching you and asking you one of these conditions as a question. If you would answer "yes," then the condition is true, otherwise, it is false.

Another name for a condition is a *Boolean expression*, named after the logician George Boole.

Building conditions

You can use the following *relational operators* to build tests:

`>`	*(greater than)*	`<`	*(less than)*	`==`	*(equal to)*
`>=`	*(greater than or equal to)*	`<=`	*(less than or equal to)*	`!=`	*(not equal to)*

Building more elaborate conditions

Sometimes you need to combine two tests into one, making a single large test comprised of two smaller tests. You can do this with these *logical operators*:

`and`	*(true if the tests on both sides of the "and" are true, false otherwise)*
`or`	*(true if either test on either side of the "or" is true, false otherwise)*

And there's one more operator that only works on a single test:

`not`	*(true if the test on the right side is false, false otherwise)*

Examples of more elaborate conditions

```
(x > y) and (y > z)                (letter == "A") or (letter == "B")

not ((pages_read > 100) and (hours_studied > 3))
```

`if-else` statements

Use an if-else statement if you want Python to do one thing if a condition is true, but do a different thing if the condition is false.

```
if condition:
   statement
   statement
   [ more statements ]
else:
   statement
   statement
   [ more statements ]
```

An `if-else` statement contains a condition and a group of indented statements following the condition, just like a regular `if`-statement. However, after the first group of statements, there is line marked "`else:`" followed by a second group of statements.

If the condition is true, the first group of statements is executed, and the second group is ignored. If the condition is false, the first group of statements is ignored, and the second group is executed.

Use a regular `if` statement when you want your program take an action only when a condition is true. Use an `if-else` statement when you want your program to take one action when the condition is false.

Notice that there is no condition that goes on the `else` line! There's no need for a condition there because the statements that go with the else clause will be executed if the original condition (on the `if` line) was false.

`if-elif-else` statements

Use an `if-elif-else` statement if you have a situation involving three or more possible outcomes, and your program needs to handle each outcome differently.

```
if condition1:
   statement
   statement
   [ more statements ]
elif condition2:
   statement
   statement
   [ more statements ]
elif condition3:
   statement
   statement
   [ more statements ]
Can add as many elif
 clauses as desired…
else:
   statement
   statement
   [ more statements ]
```

This kind of structure is an extension of `if-else` to add additional conditions. This is often used when you need to make a multi-way decision, such as a numeric score that can fall into many different ranges of numbers, or a string variable that could be in one of many different categories.

In `if-elif-else` statements, Python evaluates `condition1` first. If `condition1` is true, then the group of statements immediately underneath is executed, and all the other groups of statements are ignored. If `condition1` is false, Python skips over the first group of statements and evaluates `condition2`. If `condition2` is true, then its associated group of statements is executed, and all the other groups are ignored. This continues for all the remaining conditions. If none of the conditions were true, then when Python gets to the `else` clause, the last group of statements is executed.

The end result is that only *one* of the groups of statements will ever be executed every time Python sees the code.