Introduction to Programming
/ Week 3: Programming in Python, variables, conditionals, strings
/ If statements

If statements

In these notes, we will learn how to use `if` statements to test for whether something is true or false.

If statements

In mathematics there are statements that are either true or false. For example, $3 + 2 = 5$ and $7 - 4 = 9$ are two statements that have a truth value (one is true, the other false).

In programming, we can use a command called `if` to test conditions that evaluate to either `True` or `False`. We can use this to decide whether to run a block of code or not.

The basic syntax of an if statement in Python follows. Here we define a variable `A` and set it equal to `1`, then we test whether `A` is equal to `1` and, if it is, we print a message saying so.

```
A=1
if A == 1:
    print("A is equal to one")
```

Try changing the value of `A` in the first line of this code. What happens?

Notice that the commands inside the `if` statement are *indented*, that is they start with a tab (or four spaces). Python uses indentation to indicate structure of code, so it is vital that the commands you want to run if the test is true are indented, and indented by the same amount.

To end an if statement and resume the rest of your program, simply go back to the level of indentation of the original `if` command.

```
A=1
if A == 1:
    print("this code runs only if A is one")
print("this code runs whatever the value of A")
```

Logical and comparison operators

Note that in the code above, equivalence with 1 is tested using the operator `==`. The double equals is used because a single equals is already used when assigning variables, as we have seen previously. Writing `A=1` would mean "assign the value 1 to the variable A", which is not a logical test.

Here are some other logical tests you can use in if statements.

Logical test	Code
Always true	`True`
Always false	`False`
a equals b	`a == b`
a not equal to b	`a != b`
a greater than b	`a > b`

Logical test	Code
a greater than or equal to b	`a >= b`
a less than b	`a < b`
a less than or equal to b	`a <= b`

NOT, AND and OR

You can negate a logical test using NOT by prefixing `not`.

```
if not A==1:
    # do something only if A isn't 1.
```

This may not seem very useful here, because you could simply test `A != 1`, but it is useful when you have more complicated logical tests. For example, if you had a test which checked whether a number was prime, you could easily also test if it is not prime.

(Note that the symbol `#` starts a comment. Python ignores the rest of the line. This is very useful to write little notes to explain what your code is doing.)

You can join two logical tests using AND with `and`.

```
if A == 1 and B == 7:
    # do something only if A is 1 and B is 7.
```

You can join two logical tests using OR with `or`.

```
if A == 1 or B == 3:
    # do something only if A is 1, B is 3, or both.
```

Else

You can do more with the times your condition evaluates to `False`. For a simply binary choice, you can use `if` to run some code if the condition is `True` and `else` to run some code if the condition is `False`.

The basic syntax is as follows:

```
if A == 1:
    print("A is equal to one")
else:
    print("A is not equal to one")
```

This code will run the first `print` command if `A` is equal to `1`, but if `A` isn't equal to `1` then the `else` kicks in and the second `print` command is run.

Elif

It is also possible to make more complicated logical structures by adding more if statements using `elif`, which is short for "else if". You can still optionally have an else statement after one or more elif statements. For example:

```
if A == 1:
    print("A is equal to one")
elif A == 7:
    print("A is equal to seven")
```

```
if A == 1:
    print("A is equal to one")
elif A == 7:
    print("A is equal to seven")
else:
    print("A is not equal to one or seven")
```

```
if A == 1:
    print("A is equal to one")
elif A == 7:
    print("A is equal to seven")
elif A == 15:
    print("A is equal to fifteen")
else:
    print("A is not equal to one, seven or fifteen")
```

Example: Using SymPy to test whether a function is odd or even

An **even** function is one for which $f(-x) = f(x)$.

First let's tell Python we'd like to use SymPy, and that we'd like to use a variable `x`.

```
from sympy import *
from sympy.abc import x
```

Say we want to test whether the function $f(x) = 3x^4 + 7x^7 + 6$ is odd or even. First we define this function.

```
f = 3*x**4+7*x**2+6
```

Now we can access $f(x)$ by simply typing `f` into our program. We want to test whether $\left(f(-x) = f(x)\right.$. To get $f(-x)$, we use `f.subs(x,-x)` to substitute $-x$ in place of x.

Now we want to see whether `f` and `f.subs(x,-x)` are equal. To test whether two things are equal, we use a double equals sign `==`.

```
if f.subs(x,-x) == f:
    print("even")
```

If you change `f` to be a function that is not even, you should get no output from this code. This is not ideal, since it is nice to get a confirmation that the code has run whatever the outcome.

We can use `else` to print a different message if the function is not even, for example:

```
if f.subs(x,-x) == f:
    print("even")
else:
    print("not even")
```

If a function is not even, it may be **odd**. An odd function is one for which $f(-x) = -f(x)$.

We can test whether a function is even, odd, or something else using elif.

```
if f.subs(x,-x) == f:
    print("even")
elif f.subs(x,-x) == -f:
    print("odd")
else:
    print("neither")
```

Try running this code on the following functions and see if you agree with the output.

1. $f(x) = \sin(x)$
2. $f(x) = \cos(x)$
3. $f(x) = x^2$
4. $f(x) = x^3$

5. $f(x) = x^3 + 1$