Introduction to Programming/ Week 3: Programming in Python, variables, conditionals, strings / Writing and manipulating text

Strings

Assigning

A string is a bit of text. We saw how to assign a string to a variable earlier.

```
joseph = "It was love at first sight. The first time Yossarian saw the chaplai
```

Note the difference between `elephant` and `"elephant"`. The first is a variable, and the second is a string. Do you see how the string is surrounded by quote marks?

In the following code, on the first line the variable `hello` is created and assigned as its value the string `"goodbye"`. Then on the second line, the variable `goodbye` is created and assigned the value held by the variable `hello` (which, as it happens, is the string `"goodbye"`). Then both variables `hello` and `goodbye` are printed (i.e. output to the screen). What do you expect to see as the output? Try running this program and see what output you get. Please make sure you understand this thoroughly, as it will help you avoid mixups in future.

```
hello = "goodbye"
goodbye = hello
print(hello)
print(goodbye)
```

You can use either double quote marks `"..."` (which I prefer) or single quote marks `'...'` for a string. Python treats these interchangeably (though this is not true of all programming languages).

Printing

Previously we typed commands one at a time into the IDLE Shell window. The IDLE Shell automatically prints the output from the commands it runs, but a program does not. If we want to get output from our program, we must use the `print` function.

For example, here we define a string and then print it out.

```
herbert = "No one would have believed in the last years of the nineteenth cent
print(herbert)
```

Here we print a string that isn't defined as a variable first.

```
print("Being wrong isn't a bad thing like they teach you in school. It is an o
```

Dynamic string formation

Things get interesting when we want to build strings dynamically. For example, to concatenate two strings together or to insert a variable into a string.

String formatting

You can put together strings involving variables like this. Note the `f` before the first quote mark and the `{...}` around the variable name.

```
theanswer = 42
print(f"The Answer to the Great Question of Life, the Universe and Everything 
```

The `f` tells Python we want the string to be formatted to include the values of the variables names, and the `{theanswer}` tells Python to put the value of `theanswer` at that point (the `{` and `}` will not appear).

You can do more sophisticated things with the variables than just reproduce them, as the following example shows.

```
A=6
B=7
print(f"{A} times {B} is {A*B}")
```

A simple way to join strings together

In simple cases, you can concatenate strings using simple `+` notation, as in the example below.

```
part2 = "It's the first helpful or intelligible thing anybody's said to me all
part1 = "Don't Panic. "
print(part1 + part2)
```

However, this can get you into trouble, for example when the variables you are using are not strings. For example, the following code will give an error because `theanswer` is an integer and Python doesn't know how to add a string to an integer.

```
theanswer = 42
print("The Answer to the Great Question of Life, the Universe and Everything i
```

You could get around this by telling Python to convert the integer to a string using `str(theanswer)`, but this relies on you knowing which variables are strings and which are not. This is why I recommended the `f""` method above, because it thinks about all this so you don't have to.

Tabs, new lines, etc.

There are various special strings used in Python and elsewhere to generate special characters.

For example, you can write a tab using `\t` and a new line using `\n`. You may have noticed that the `print` function automatically puts a `\n` at the end of your string. Run the following code and check you understand how each of the tabs and new lines are generated.

```
print("Happy\tBirthday\tTo\nYou")
print("Happy Birthday")
```

One particular issue is what to do, when using quote marks to declare a string, if you want to include a quote mark within the string. For example, in the following code, Python will think that the second `'` is ending the string `'Don'` and then get confused by the `t` and the rest of the line.

```
reassurance = 'Don't panic'
```

You can use a quote mark, but you tell Python it is just a part of the string by *escaping* it. To escape special characters in Python, use `\`. You can see this in action if you run the following example.

```
print("\"Luck is my middle name,\" said Rincewind, indistinctly. \"Mind you, m
```

Similarly, you can escape a single quote using `\'` and a backslash itself using `\\`.

```
print('Don\'t panic.')
```

You could also be careful to choose either `"..."` or `'...'` to avoid this problem, as in the following example. Of course, this won't work if you have both `"` and `'` in your string!

```
print('"Luck is my middle name," said Rincewind, indistinctly. "Mind you, my f
print("Don't panic.")
```

Maths in SymPy

When we defined a maths function variable in SymPy we did something like this.

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

f = 2*x**2 - 9*x - 35
```

In the first two lines we import SymPy and then use its pre-defined variable `x`. Then the expression `2*x**2 - 9*x - 35` doesn't need quote marks around it because it isn't a string.

We can `print` SymPy expressions and the output from SymPy functions. For example:

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

f = 2*x**2 - 9*x - 35
print(f)
```

The output from Python can be hard to read. SymPy has other options for output which can help with this. For example, `pprint` ('pretty print') attempts to format the output in a way that is more recognisably mathematical. Try this:

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

f = Integral(2*x**2 - 9*x - 35, x)
pprint(f)
```