Python Lesson 1

Getting Started

Installing Python at home

If you want to try this at home you can install Python 3 using the instructions at https://www.python.org/downloads/. Ask your parents' permission first before installing any software, or get them to help!

If you have a Raspberry Pi, Python 3 is almost certainly already installed.

Start IDLE (Python 3)

- From your computer's Start Menu (or equivalent).
- Typing `idle3` at a command prompt.

Using Python Option 1: Type commands directly into IDLE's "Python Shell" window. This is good for learning and testing, but not so good for writing proper programs. Try typing some things to test the shell:

```
>>> print("Hello World")

>>> 5

>>> 5+5

>>> my_name="Mr Happy"
>>> print(my_name)
```

We will explain what these commands do later, but you might be able to work it out for yourself.

Using Python Option 2: Create a new Python program file and run it.

1. File → New File (or File → Open File to edit an existing program)
2. Type your Python program
3. File → Save (or Ctrl+S). Python files are normally saved with a .py extension at the end.
4. Run → Run Module (or F5)
5. Make fixes/corrections and repeat from step 3!

Let's create our first Python program file!

My First Python Program

Create a new Python file called **hello.py.**

This is a simple program that only contains a single instruction (**statement**).

```
print("Hello Stamford Green!")
```

Run the program by pressing F5 or selecting Run → Run Module from the IDLE menu.

`print()` is a built-in Python **function** that displays a message.

`"Hello Stamford Green!"` is the **string value** that we want to give to the print function. The speech marks indicate the beginning and end of the string.

Brackets (braces) and speech marks (double quotes) always come in matching pairs! See what happens if you leave out a speech mark or bracket. Can you make sense of the error message?

What happens if you put a spelling mistake in "print"?

Making mistakes when writing computer programs is normal, and a lot of the fun is working out what went wrong. It can be satisfying like solving a tricky maths problem, but you can ask for help when you get stuck!

My Second Python Program

Create a new file called **school_year.py** containing the lines below and run it to see what happens.

```
my_name="Sophie"  # Change this to your name. This is a comment.
my_school_year=5  # Change this to your school year

print("My name is", my_name, "and I am in year", my_school_year)

if my_school_year <= 6:
    print("I am in primary school")
    print("I will be leaving in", 6-my_school_year, "years")
else:
    print("I am already in secondary school")
```

Change `my_school_year` to 7 and try again.

The program looks simple, but there's a lot new here that we need to explain!

Comments

Python ignores everything after a **#** (hash) symbol until the end of the line, so you can use it to write friendly **comments** to explain what the program does. It's a good idea to write comments to explain complicated parts of your program, but it's even better to make your program simple to read so it doesn't need explanation!

Variables

The tokens `my_name` and `my_school_year` are **variables**. You can think of variables as being a bit like named boxes that you can store values in.[1]

The = (assignment) operator stores a value into a variable. The syntax is *`variable=value`*.

- `my_name="Sophie"` stores the **string value** "Sophie" in a variable called `my_name`.
- `my_school_year=5` stores the **whole number (integer) value** 5 in a variable called `my_school_year`.

To read the value stored in a variable, just type the variable name. Anywhere that you want to use the value "Sophie" you can now use the variable `my_name` instead, unless you change the value stored in the variable later in your program!

We already saw the `print()` function in our first program. It can take more than one value to print, separated by commas. The values in the printed message will be separated by spaces.

---

1 You can choose your own variable names. They can contain letters, numbers or the underscore '_' character, but cannot start with a number. It's normal to use only small letters (lower case) with words separated by an underscore. A small number of special words are reserved by Python (if, else, elif, while, for, import, def, …) so you can't use these as variable names!

If / Else Statement and Conditions

The line "**`if`** `my_school_year <= 6:`" means "if the value stored in `my_school_year` is less than or equal to 6 then…".

Python has several other operators for comparing values: `==` (equal to), `!=` (not equal to), `<` (less than), `>` (greater than) and `>=` (greater than or equal to). You can try these out in the Python shell:

- `5 > 6`
- `"Sausages" == "Sausages"`
- `5 != "Sausages"`
- `5 < "Sausages"` ← this doesn't make sense and causes an error!

If the **condition** `my_school_year <= 6` evaluates to `True`[2], the indented block of statements after the if statement will be performed, otherwise they will not.

In Python, blocks are grouped together by indenting the lines with the same number spaces, normally 4. It's very important to use correct indentation! Python can sometimes warn you if you get it wrong, but sometimes your program will do the wrong thing – a bug!

If you add an **`else:`** statement immediately following an `if` statement, it means "otherwise do this…". It provides another block of statements that will only be performed if the condition evaluated to `False`.

We aren't using it in this program, but Python also has a keyword **`elif`** short for "else if". You can add any number of `elif` blocks after an `if` statement. The `else` block at the end will only be performed if the `if` condition *and* all the `elif` conditions are false. Here is an example of `if`, `elif` and `else` from a "Stamford Green Dungeons" adventure game.

```
print("You are in the secret dungeons under the school.")
direction = input("Which way do you want to turn, left or right? ")

if direction == "left":
    print("You are in a dark cave surrounded by teachers. It is terrifying.")
elif direction == "right":
    print("Congratulations! You have escaped from Stamford Green dungeons.")
elif direction == "back":
    print("There is no going back.")
else:
    print("I don't understand that direction.")
```

2The Python values `True` and `False` are called **Boolean** values. Python has keywords for working with Boolean values and combining them including **not**, **and** and **or**. For example:

```
if my_name == "Sophie" and my_school_year > 6:
    print("Sophie, you are in secondary school!")
```

My Third Python Program

Create a new file called **insult_generator.py**:

```
import random

people=["Harry", "Ron", "Hermione", "Ginny", "Fred", "George"]
adjectives=["hot", "cold", "smelly", "green", "wet", "damp", "cool", "nice"]
things=["toad", "rat", "cat", "monkey", "owl", "banana", "muggle"]

print("Here are your insults for today:")

count=1
while count < 21:
    person=random.choice(people)
    adjective=random.choice(adjectives)
    thing=random.choice(things)
    insult=person + " is a " + adjective + " " + thing  + "."

    print(count, insult)

    count=count+1
```

Run it to see what happens!

Importing Modules

The `import` statement on the first line tells Python that this program needs to use a **module** called "`random`". Modules can be thought of as a collection or library of additional **functions** which another programmer has written for you to use.

Python comes with lots of standard modules that you can import and use, and you can download additional modules (if you have permission) or even create your own.

In this particular program we use a function called `random.choice()` from the `random` module. This function selects a random item from a list.

Lists

The variables `people`, `adjectives` and `things` contain **list** values. Lists are containers for other values. They are written as comma-separated values between [and] square brackets, for example ["red", "green", "blue"] or [1, 1, 2, 3, 5, 8].

There are lots of interesting things you can so with lists, like add and remove items, sort the items, count the items, process the items once at a time in a loop, etc.

Feel free to change the people, adjectives and things in this program to use your own values, but nothing too rude please!

While Loop

The `while` statement is a bit like the `if` statement that we saw in the previous program except it keeps *repeatedly* running the indented block multiple times (it "loops" back to the beginning) until the condition is no longer true. The condition is checked again every time the loop restarts.

In this program the `while` condition is `count < 21`, which means that the loop will continue running when the value stored in the `count` variable is less than 21. When the count value reaches 21, the loop will complete and any next instruction after the loop will be run.

`count` starts off with the value 1, and each time through the loop we add 1 to the stored value using `count=count+1` (arithmetic add) so the loop will repeat 20 times. This program prints 20 different insults!

String Concatenation

"Concatenation" is just a posh word meaning "joining together one after another". The `+` symbol is used to concatenate string values in Python. It creates a new string value with the values stuck together. For example, `"A"+"B"` creates a new string value `"AB"`.

```
insult=person + " is a " + adjective + " " + thing  + "."
```

This will make a string something like "Harry is a hot banana." and store it in the `insult` variable.

We've now seen that + can mean at least two different things: with number values it means "arithmetic add", but with strings values it means "concatenation". What do you think happens if you try to add a number to a string? You can try it in the IDLE Shell, for example with `print("Hello"+1)`.

Summary: What have we seen in this lesson?

- IDLE Python Shell.
- Saving and running Python programs.
- The print() function.
- Understanding some Python error messages.
- Values: Strings, Numbers, Lists. Number arithmetic, string concatenation.
- Using variables for storing values.
- If/Else conditions.
- While loops with counters.
- Importing the 'random' module and using `random.choice()`.