

# Wokingham Code Club online

## Summary of Python session Saturday 19th September 2020

### Python from the beginning - session 2

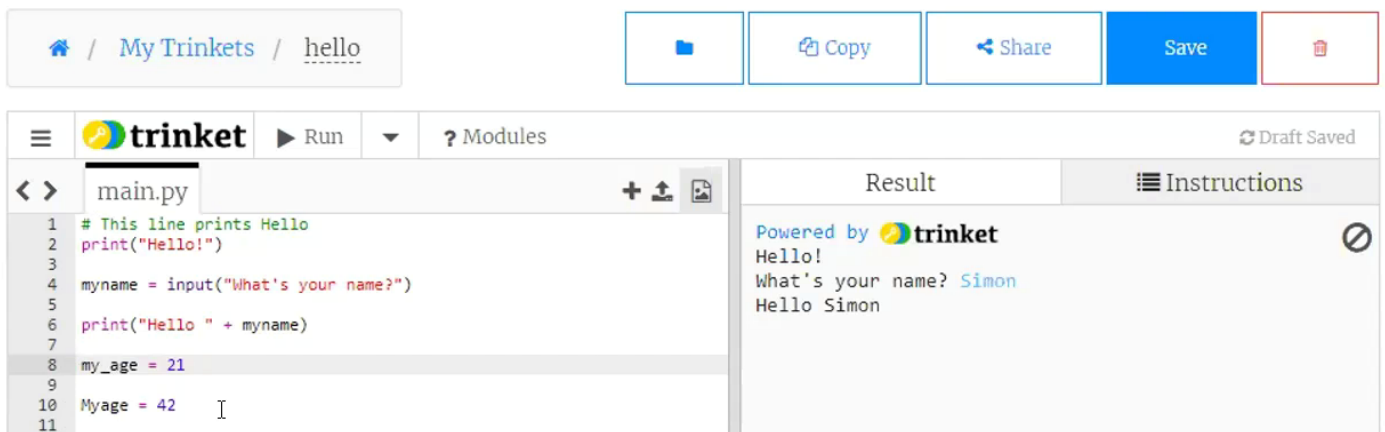
#### 0. Catching up

To catch up on what happened in **Python from the beginning - session 1** watch this youtube video

<https://youtu.be/7JIm-EYIsTQ>

Session 2 picks up from the end of this video.

Here is the trinket window at the end of session 1



Main Python coding points from session 1:

#### 0.1 Making and printing text

To create a *text string* - a bit of text which Python recognises as text - put the text inside quotes - you can use single or double quotes but don't mix them for the same text string. The Python `print()` function will cause whatever you put inside the brackets to be printed in the results window:

```
print("Hello!")
```

or

```
print('Hello!')
```

#### 0.2 Making variables

##### Naming variables

To make a variable you must first give it a **name**. Variable names must start with a letter or the underscore character, cannot start with a number, and can only contain alpha-numeric characters and underscores (A-z, 0-9, and `_`). You can't put a space in a variable name, but often people use the underscore character to show that the variable name consists of several words.

Python is very fussy about capital letters and lower case letters. If I make a variable called `Myname` Python will think this is an entirely different variable from `myname`.

To make a variable in Python you simply write a line of code which sets the variable to a value.

### Giving variables a value

To give a variable a value you type the name of the variable, then the equals sign, then the value that you want to set the variable to.

If you set a variable equal to a piece of text Python will know this is a *text* variable or *string* variable. If you set it equal to a number than Python will know it is a *numeric* variable. There are things you can do with text variables that you can't do with numeric variables, and the other way round.

Here is a line of code

```
myname = "Simon"
```

this creates a variable called `myname` and set the variable equal to the text string "Simon". Python knows this is a string variable.

```
my_age = 21
```

This code creates a numeric variable called `my_age` and sets it equal to 21.

At the beginning of learning Python our variables will only contain text or numbers, but as you learn more Python you will find that variables can be set to a whole range of different things.

### 0.3 `input()` to get user to type in something

The Python function `input()` asks the user to type in something. Inside the brackets you can put a text string, called a *prompt* string, which will appear on the screen to tell the user exactly what input the code wants. Normally `input()` is used with a variable name:

```
1 #!/bin/python3
2
3 # This line prints Hello
4 print("Hello!")
5
6 myname = input("What's your name?")
7
```

The last line writes "What's your name?" in the results window, then whatever the user types this will be set to the value of the variable `myname`.

## 1. Getting Python

The different ways you can access Python on your computer were covered in session 1, but there is also a very useful webpage which summarises this:

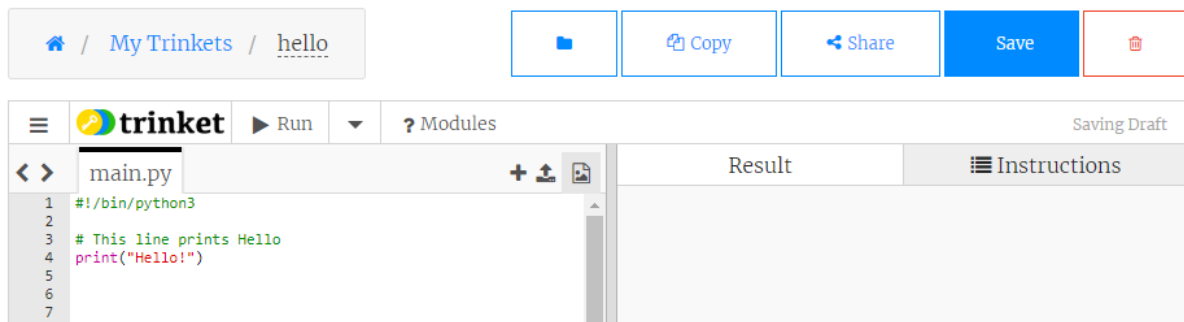
<https://projects.raspberrypi.org/en/projects/python-install-options>

## 2. Confusion between Python 2 and Python 3

Python has been around since the 1980s, and since then it has been revised and updated many times. Unfortunately this has led to some confusion. Python was given a major overhaul in the year 2000 when version 2 was released; then in 2008 a major new version, Python 3, came out. This version did some things rather differently to Python 2, so that some commands work differently depending on whether you are using Python 2 or Python 3. In the code club we will **always use Python 3** and we would advise you to do the same, but the trinket version of Python still uses some Python 2 commands *unless we tell it to use Python 3*.

To tell your trinket Python code to use Python 3 **always add this line** at the very beginning of all your projects:

```
#!/bin/python3
```



When you look on the internet for help with your Python code **make sure the webpage is talking about Python 3 and not Python 2.** Usually it will state at the top which version of Python it refers to.

### 3. Python ignores blank lines

You can put as many blank lines as you like in your Python code - Python will just ignore these. This makes it easy to divide your code up into different sections - separated by blank lines - to make the structure of your code easier to understand.

If you also add comments (lines beginning with the hash character #) at the beginning of each section you will find it much easier to locate any problems ("bugs") in your code if it doesn't work properly.

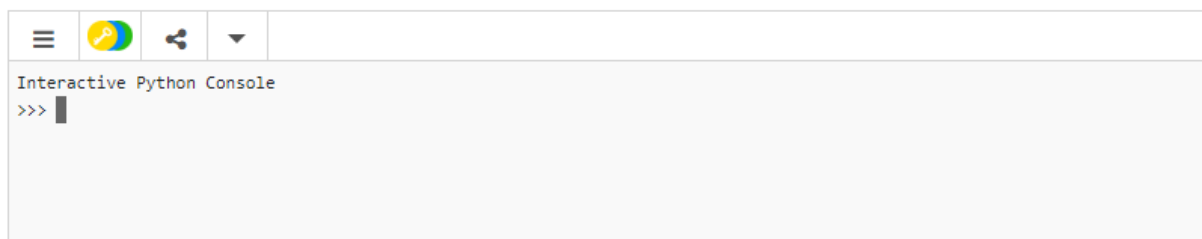
### 4. Doing sums and the Python console

#### 4.1 Python console

If you open a new tab in your browser and go to this link

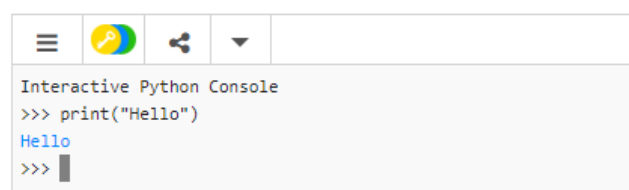
<https://trinket.io/console>

you will find a webpage looking like this:



where you can type things after the three "greater than" signs >>>. This is the *Python Console*, or *Python Shell*, and it's a window where you can test out bits of code before you actually write them into a programme and run them.

You could use this window to test out print functions



or to do sums

```
Interactive Python Console
>>> print("Hello")
Hello
>>> 2 + 3
5
>>> 
```

## 4.2 Rules for sums

You can see above that Python uses the normal plus sign `+` to add numbers together. It also uses the short dash `-` as a minus sign, the asterisk `*` to indicate multiplication and the slash `/` for division.

```
Interactive Python Console
>>> print("Hello")
Hello
>>> 2 + 3
5
>>> 5 - 3
2
>>> 3 * 5
15
>>> 10/2
5.0
>>> 2 + 3 * 4
```

Look at the sum typed in the last line above. What do you think the answer would be?

If you thought the answer would be 20 you would be wrong. The answer is 14. This is because Python uses a set of rules for the order of doing types of calculation (you may have already come across these rules) and the main rule is that you do multiplication or division **before** addition or subtraction. Python calculates `3 * 4` first, to get 12, then adds 2 to get 14.

If you want to instruct Python to carry out the addition first then you use brackets - because calculations inside brackets are done **first**:

```
>>> 2 + 3 * 4
14
>>> (2 + 3) * 4
20
>>> 
```

## 4.3 Addition with text

You can use a plus sign to add text strings as well as numbers! This doesn't do a mathematical addition, but it *joins* one text string on to the end of another:

```
>>>
>>> h = "Hello, "
>>> w = "World"
>>> print(h + w)
Hello, World
>>> 
```

Here we have used the Python console to make two string variables and used the plus sign to join them together.

## 5. Converting text variables to numbers and numeric variables to text

In Python you can do mathematical sums with numbers but not with text. So, obviously, you can do sums with numeric variables but not with text variables.

### 5.1 `input()` function always produces text

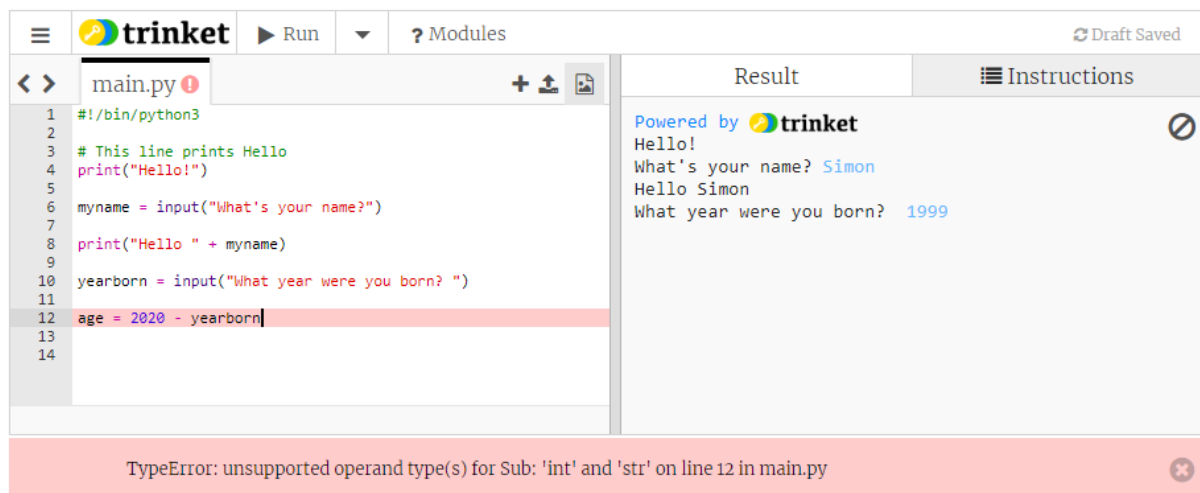
In Python, whenever you use the `input()` function Python takes whatever input the user types as *text*, **even if the user types a number**. This is important to remember, as it can lead to code not working properly although it looks to be correct. Here is a bit of code using `input()`

```
9
10 yearborn = input("What year were you born? ")
11
```

The user is asked to type in their year of birth, and this is put into a variable called `yearborn`. Let's say the user types 1999.

This is the start of a bit of code to calculate how old a person is, using the year they were born. If you were born in 1999 you can calculate your age in 2020 with a subtraction sum `2020 - 1999`.

But if you use the variable `yearborn` to calculate the person's age you will get an error when you run your code:



The screenshot shows the Trinket Python IDE interface. On the left, the code editor displays a file named `main.py` with the following code:

```
1 #!/bin/python3
2
3 # This line prints Hello
4 print("Hello!")
5
6 myname = input("What's your name?")
7
8 print("Hello " + myname)
9
10 yearborn = input("What year were you born? ")
11
12 age = 2020 - yearborn
13
14
```

On the right, the 'Result' pane shows the output of the program:

```
Powered by trinket
Hello!
What's your name? Simon
Hello Simon
What year were you born? 1999
```

At the bottom, a red error banner displays the following message:

```
TypeError: unsupported operand type(s) for Sub: 'int' and 'str' on line 12 in main.py
```

Here, the problem is that the variable `yearborn` is a *text* variable - it doesn't contain the number 1999, it contains the text "1999", so Python *can't* use it in the mathematical calculation

```
a = 2020 - yearborn.
```

### 5.2 Converting a text variable into numeric variable

Luckily there is a Python function `int()` which converts a text variable into a numeric variable.

**int** is short for integer, which is the mathematical word for a whole number. This code

```
yearborn = int(yearborn)
```

will take the text value of variable `yearborn`, convert it into an integer using the `int()` function and put the number into the same variable `yearborn` - which is now a numeric variable and can be used in a sum.

The variable `age` contains the answer to the sum, so this is a *numeric* variable as it contains a number.

The screenshot shows the Trinket IDE interface. On the left, a code editor displays a Python script in `main.py`. The script includes a shebang, a comment, a `print` statement for 'Hello!', an `input` statement for a name, a `print` statement for 'Hello ' followed by the name, another `input` statement for a birth year, an `int` conversion, and an `age` calculation. On the right, the 'Result' pane shows the output of the script, which matches the input provided in the code.

```
1 #!/bin/python3
2
3 # This line prints Hello
4 print("Hello!")
5
6 myname = input("What's your name?")
7
8 print("Hello " + myname)
9
10 yearborn = input("What year were you born? ")
11
12 yearborn = int(yearborn)
13
14 age = 2020 - yearborn
15 print(age)
16
```

Result: Powered by trinket  
Hello!  
What's your name? Simon  
Hello Simon  
What year were you born? 1999  
21

Let's use the plus sign to join together a bit of text and the person's age in the print line at the end of this code, so that it will print out Your age is 21, instead of just 21. We'll change the last line to

```
print("Your age is " + age)
```

This screenshot shows the same Trinket IDE interface, but the last line of the script has been changed to `print("Your age is " + age)`. A red box highlights this line. The 'Result' pane shows the same output as before. At the bottom of the IDE, a red error message is displayed: `TypeError: cannot concatenate 'str' and 'int' objects on line 15 in main.py`.

```
1 #!/bin/python3
2
3 # This line prints Hello
4 print("Hello!")
5
6 myname = input("What's your name?")
7
8 print("Hello " + myname)
9
10 yearborn = input("What year were you born? ")
11
12 yearborn = int(yearborn)
13
14 age = 2020 - yearborn
15 print("Your age is " + age)
16
```

Result: Powered by trinket  
Hello!  
What's your name? Simon  
Hello Simon  
What year were you born? 1999

TypeError: cannot concatenate 'str' and 'int' objects on line 15 in main.py

Now we've hit another problem. This time it's because Python won't let you use the plus sign to join together a bit of text and a numeric variable like `age`.

### 5.3 Converting a numeric variable to text

Luckily there is a Python function `str()` which does the opposite to the `int()` function - it converts a numeric variable into text:

```
age = str(age)
```

This takes the numeric value of variable `age`, converts it into a text string using the `str()` function and put the text into the same variable `age` - which is now a text variable and can be used with a plus sign to join it to more text.

The screenshot shows the Trinket Python IDE interface. On the left, a code editor displays a Python script in a file named `main.py`. The script includes a shebang line, a comment, a `print` statement, an `input` statement for a name, another `print` statement, an `input` statement for a birth year, an `int` conversion, an age calculation, a string conversion, and a final `print` statement. On the right, the 'Result' pane shows the output of the script, which includes the 'Hello!' message, the name 'Simon', and the calculated age '21'.

```

1 #!/bin/python3
2
3 # This line prints Hello
4 print("Hello!")
5
6 myname = input("What's your name?")
7
8 print("Hello " + myname)
9
10 yearborn = input("What year were you born? ")
11 yearborn = int(yearborn)
12
13 age = 2020 - yearborn
14 age = str(age)
15
16 print("Your age is " + age)
17

```

Result: Powered by trinket  
Hello!  
What's your name? Simon  
Hello Simon  
What year were you born? 1999  
Your age is 21

## 5.4 Using commas in a `print()` function

There is another, slightly easier, way of printing `Your age is 21`, and that's by using a comma inside the brackets in the `print()` function:

```
print("Your age is ", age)
```

Here it doesn't matter if `age` is a text variable or a numeric variable, because `print()` is not trying to join it to the text string.

## 6. `if...else` blocks in Python

If you try out the Python code to calculate your age it will give you the wrong answer if you haven't had your birthday yet this year. But we can adapt the programme to make sure it always gives the right answer using a Python `if...else` statement. You may have come across `if` blocks in Scratch.

First we need more information - has the user had a birthday this year - we can use `input()` to get this.

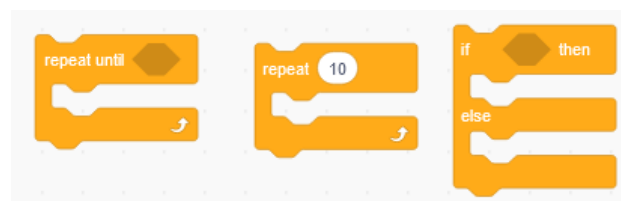
```
birthday_yet = input("Have you had your birthday this year (y or n)? ")
```

This will ask the user whether they've had a birthday this year and put whatever they type into a text variable called `birthday_yet`. Notice how the prompt string also tells the user how they should reply - we want an answer which either is "y" or "n" but nothing else. We will write the code assuming the user has typed one of these letters.

### 6.1 Making *chunks*, or *blocks* of code in Scratch

A *block* of code is a series of lines of code that will run together whenever a particular instruction is given, or a particular situation occurs.

In Scratch you use chunks of code where the shape of the Scratch blocks is like this:



These Scratch blocks are shaped like a sort of container, with a top and bottom boundary, and whatever code you put inside the container only happens under certain circumstances. In the case of the `if` block you put a *condition* inside the diamond shaped hole and the chunk of code inside will run if the condition is true. The code inside the `else` container will run if the condition is not true.

## 6.2 Making *blocks* of code in Python

Because Python code is all typed in text it can't use a picture to define blocks, so it needs a different way of showing where a block of code begins and ends. The Python way of doing this is

- the line of code which *starts* the block must **end** with a colon :
- all the following lines which are part of the block must be **indented**, which means the lines have to start with some space characters
- after the end of the block the next lines of code are **not indented**.

We will show this with the Python `if` statement. We want to check if the user typed "y" when asked "Have you had your birthday this year (y or n)?" . The Python code is

```
if birthday_yet == "y":  
    age = 2020 - yearborn
```

The first line starts with the word `if`, then it does a comparison - is the variable `birthday_yet` equal to the string of text "y"? Python always uses a *double equals sign* for comparing two values, because in Python the single equals sign is only used when setting the value of a variable.

Note the colon at the end of this line - that signals that we are starting a block of code.

If the answer is "true" (the variable is equal to "y") then we can calculate the user's age just as we did before. In this case our block consists of one line of code, and notice that this line is *indented* - it has a couple of spaces at the beginning.

What if the user typed "n" in answer to the question? In this case our age calculation will give an answer which is too big by 1, so we need a different age calculation where we take off 1 from the answer. To do this in Python we need to make the `if` statement more complex by adding another block of code which begins with the word `else`: (ending with a colon). The code inside the `if` block runs if the condition is true, and the code inside the `else` block runs if the condition is not true, so our age calculation for the `else` block will be `age = 2020 - yearborn - 1`.

Here is the complete `if...else` code:

```
if birthday_yet == "y":  
    age = 2020 - yearborn  
else:  
    age = 2020 - yearborn - 1
```

Notice the word `else` is *not* indented - it is not part of the `if` block of code.

After we've computed the user's age we can now convert the variable `age` from a number to a text string and print the answer. These lines of code are *not* indented - they are not part of the `else` block of code:

```
if birthday_yet == "y":  
    age = 2020 - yearborn  
else:  
    age = 2020 - yearborn - 1  
  
age = str(age)  
print("Your age is " + age)
```

The trinket window below shows the code working, and shows that Python has computed a different age because the user hasn't yet had a birthday this year:



trinket

Run

Modules

Draft Saved

main.py

```
1 #!/bin/python3
2
3 # This line prints Hello
4 print("Hello!")
5
6 myname = input("What's your name?")
7
8 print("Hello " + myname)
9
10 yearborn = input("What year were you born? ")
11 yearborn = int(yearborn)
12
13 birthday_yet = input("Have you had your birthday this year (y or n)? ")
14
15 if birthday_yet == "y":
16     age = 2020 - yearborn
17 else:
18     age = 2020 - yearborn - 1
19
20 age = str(age)
21 print("Your age is " + age)
```

Result

Instructions

Powered by trinket

Hello!  
What's your name? Simon  
Hello Simon  
What year were you born? 1999  
Have you had your birthday this year (y or n)? n  
Your age is 20

## 7. Next steps

We have covered the basic steps of printing text in the Result window and getting input from the user, doing sums, converting between text and numbers and applying conditions. Next we will show how Python can create something rather like Scratch sprites and move them around the screen.