Introduction to programming

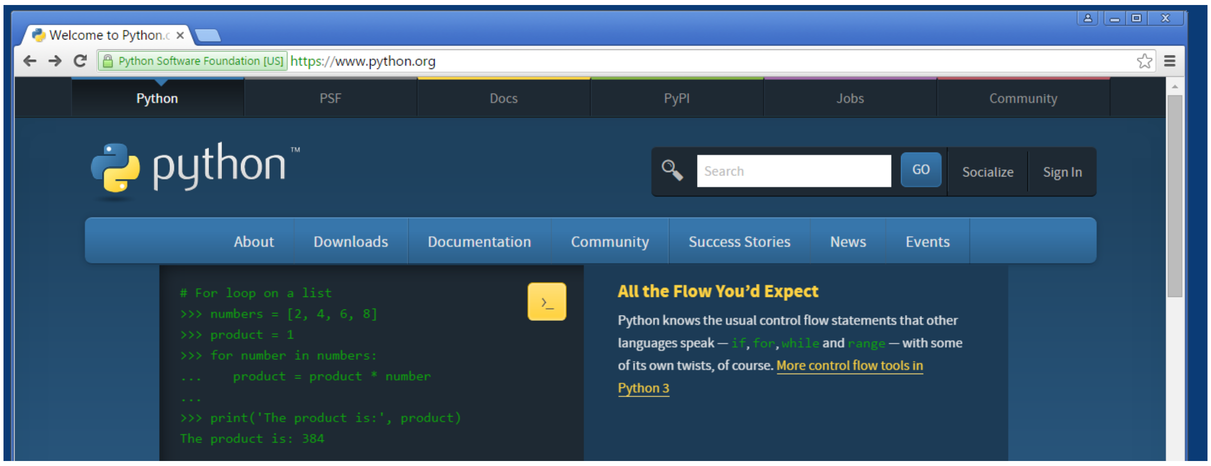
Input, output and variables

Displaying output

One of the key things you will need many of your programs to do is to provide feedback to users. We have already seen this in our first program that we wrote last week where we displayed **Hello World** on the screen using the **print** keyword. This is fine when writing programs that are going to be run from the command prompt as the text that is output fits in with the environment that you are working in.

Screenshot of IDLE hello world

Another way to convey information to users is by using a **Graphical User Interface** (GUI). We use GUIs every day when working with our computers; most programs that you work with will have a GUI to display information to you and also for you to provide input. GUIs generally provide a simpler interface for users of a program by providing buttons, images etc. Programs run from the command line are generally menu driven and can require more expertise to operate.



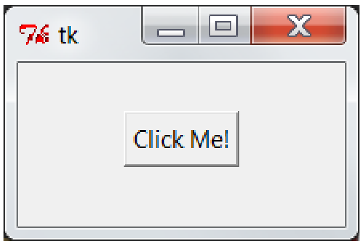
```
# For loop on a list
>>> numbers = [2, 4, 6, 8]
>>> product = 1
>>> for number in numbers:
...     product = product * number
...
>>> print('The product is:', product)
The product is: 384
```

All the Flow You'd Expect

Python knows the usual control flow statements that other languages speak — if, for, while and range — with some of its own twists, of course. More control flow tools in Python 3

Using the GUI of Chrome to browse the web

Python has a few options when it comes to creating GUIs for programs. An easy, built in option is to use is a library called **tkinter.**



You can create complex applications with tkinter but the above shows a simple example of how you can create a GUI using Python. Try copying the Python program below and running it. You will see it creates the same window with the **Click Me!** button (the button will not do anything so feel free to click it!).

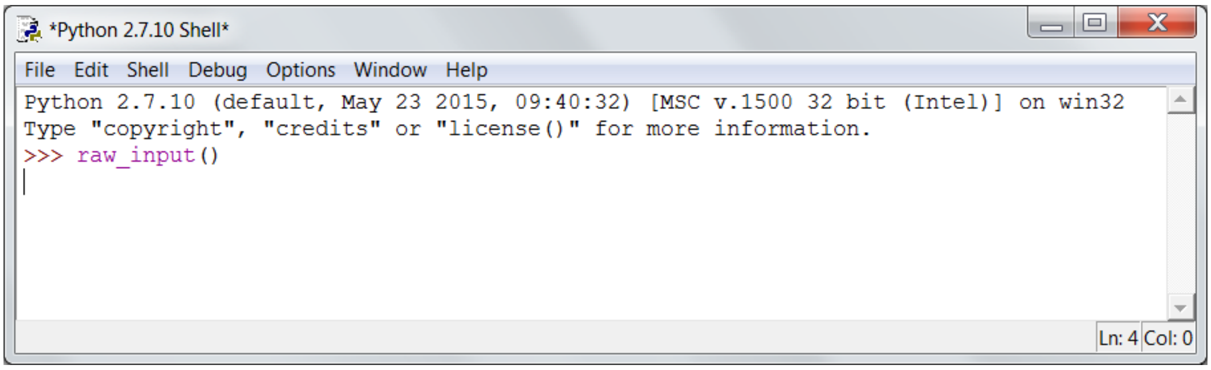
```
import Tkinter
window = Tkinter.Tk()
btn = Tkinter.Button(window, text="Click Me!")
btn.pack(pady=30)
window.mainloop()
```

There are several new concepts in the GUI program above so do not worry about understanding it at this point.

Receiving input

In the same way there are different ways for a program to display out, there are several ways for a program to receive input. A user can type input directly into a program when it is running or click a button within a GUI but also a program can access information stored in files or databases.

A simple way for us to supply a Python program with some input is to use the built in **raw_input** function. In IDLE, you can simply type raw_input() and IDLE will pause waiting for you to type something.



```
Python 2.7.10 (default, May 23 2015, 09:40:32) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> raw_input()
```

Anything you type will simply be repeated back to you on the screen but this is the basics for giving information to Python. When a program runs that has a raw_input function the program will pause to take the input and then substitute the value that is typed into the program. For instance we could combine raw_input with a print statement to display a custom message.

```
print "Hello, %s", raw_input()
```

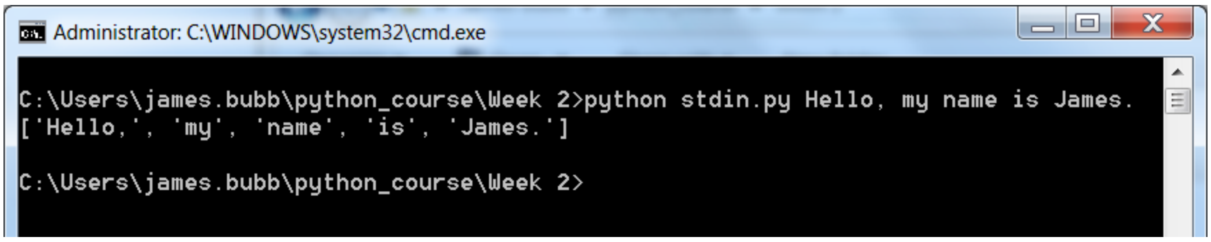
Try the program out and you will see that the program pauses before displaying the **Hello,** message as it needs to know what you want to output as the string.

When run from the command prompt, a Python program will have access to **standard input.** Which is data provided directly on the command prompt either from the user or from the result of another command or file. To illustrate this, here is a simple program that prints out any text that is supplied after a program is run from the command prompt.

```
import sys
print sys.argv[1::]
```

The program can then be run on the command prompt, assuming that the program has been saved as **stdin.py** by typing: **python stdin.py <text>**

The resulting output looks a little strange as it has been converted to an **array** (we will look at arrays in later weeks).



```
C:\Users\james.bubb\python_course\Week 2>python stdin.py Hello, my name is James.
['Hello,', 'my', 'name', 'is', 'James.']

C:\Users\james.bubb\python_course\Week 2>
```

Variables

In order for our programs to perform useful actions they need to remember values that are provided to it, either from a file, user input or the result of a calculation for example. These values are stored in **variables** which are locations in a computer's memory. You can think of variables as buckets which hold either, numbers, text, files, objects and many other things. In Python, the value of what is stored in these 'buckets' can be changed at any time, swapped with a totally different item and if it is a numeric value, mathematical operations can be applied to them. To create a variable in Python we simply specify a name to refer to the variable and the value you want to store.

```
myName = 'James'
```

The above is an example of a **string** variable but as mentioned, there are other types of data we can store in our variables.

Data type overview

Data type	Explanation	Examples
String	A single letter, space, number, symbol or longer combination of these. Need to be enclosed in quotes (either ‘ or “)	‘James’, ‘Pa55w0rd’, ‘Python programming!’, ’52 Shepherds Brook Rd’
Number	Either a whole number (integer) or a decimal number (with a decimal point)	123, 0000, 3.14, 1.11
Boolean	Either true or false, like a binary 1 or 0. Do not need quotes	true, false

Object	A representation of a real world or abstract object such as a **bike** or a **file** on the computer. They contain other variables and actions (methods).	`class Bike:`
		`    gear = 1`
		`    def changeGear(self,g):`
		`        self.gear = g`

Using variables with print
You have already used strings when writing our simple programs that **print** information to the user.

```
print "Hello World!"
```

The text inside the double quotes is the string. If you want to print the contents of a variable you can put the variable name after the **print** keyword.

```
myName = "James"
print myName
```

Try creating some variables in IDLE and then printing out their values. This also works for other data types such as numbers and Booleans.

You might want to combine strings together with other strings or variables to create custom messages to the user. This can be done by adding or **concatenating** the desired variables together.

```
myName = "James"
print "Hi my name is " + myName
```

In Python you have to be careful about combining data types together. For example if you wanted to extend the above program to display your age:

```
myName = "James"
myAge = 21
print "Hi my name is " + myName + " and I am " + myAge + " years old."
```

You will encounter an error. This is because Python does not allow you to mix the string variable and integer variable (myAge) when using the **print** keyword. This is an example of Python's **strong typing**. We can fix this by telling Python we want to use the myAge integer variable as a string using the **str()** function.

```
myName = "James"
myAge = 21
print "Hi myname is " + myName + " and I am " + str(myAge) + " years old."
```

Try running the above programming entering your name and age to display the message.

Storing user input
Displaying messages to the user with information you already have is fairly trivial but with most programs you write you will not have information from the user until they run the program. This information could be retrieved from a file, a database, an Application Programming Interface (API) or directly from the user entering information. For simplicity at this point, let us look at retrieving and storing information directly from the user. You will remember we can receive input from the user using **raw_input()**. To store what the user types in, you can create a variable to catch the result.

```
print "Hi, what's your name?"
name = raw_input()
print "Hi, " + name + ".  How are you today?"
```

To store and retrieve user input from databases and files in Python, we need to use Objects that Python provides to open connections to the data source which we can then read or write to. You will learn more about files and databases in later lessons.

Arithmetic and variables

In last week's lesson you saw how to do simple sums in Python. Now that you know how to store numbers in variables you can perform simple arithmetic operations with them.

```
firstNum = 8
secondNum = 5

print firstNum + secondNum
print 10 - secondNum
print firstNum * secondNum
print firstNum / 8
```

Try running the above program to see the result of the sums. As you can see you can use either literal numbers or variables to perform the calculations.

In most programming languages you can use increment (to increase) or decrement (to decrease) operators to modify a variable's value by 1. For example in Java:

```
int number = 10;
number++;
```

will result in number being incrementing to 11;

Python does not have support for ++ and – to increase and decrease a variables value but you can achieve this by reassigning the value of the variable. So to add one to a variable:

```
number = 10
number = number + 1
```

You are basically saying, let the variable referenced by the name **number** equal the same value as **number** is already and then add 1 to that value. Python lets you write this in a shorter way:

```
number = 10
number += 1
```

This will have the same result as the first programming. You can do this with all of the arithmetic operators e.g. -=, *=, /=

Boolean expressions

As mentioned in the data types section earlier, a boolean value is one that is either **true** or **false**. A Boolean expression is the result of a statement that can either be true or false. Some real world examples (which possibly could be argued!) could be:

- The sky is blue – *TRUE*
- Water is made from hydrogen and oxygen – *TRUE*
- Ice is warm – *FALSE*
- You have money in your bank account – *FALSE*

When writing a program you might want to consider whether a certain expression is true. For example:

- User is logged in
- Password is correct
- Player has lives left
- Connection is active

These can all be checked with a Boolean expression which is often referred to as a **condition**. There are several operators you can use when writing conditions.

Operator	Explanation	Examples
>	Greater than. If the value on the left hand side is larger than the value on the right then the result is **True**.	2 > 1 ***TRUE*** 5 > 10 ***FALSE*** 3 > 2 ***TRUE***
<	Less than. If the value on the left hand side is smaller than the value on the right then the result is **True**.	2 < 1 ***FALSE*** 5 < 10 ***TRUE*** 3 < 2 ***FALSE***
==	Equals. Results in **True** if the values on both the left and right are the same. Note that there must be the double equals ==	2 == 1 ***FALSE*** 5 == 5 ***TRUE*** 'password == 'password' ***TRUE***
!=	Does not equal. Results in **True** if the values on the left and right do not match.	2 != 1 ***TRUE*** 5 != 5 ***FALSE*** 'password != 'password' ***FALSE***

You can use Boolean conditions to check how literal values compare to each other but more it is more useful to check the values of variables.

```
livesLeft = 3
print livesLeft > 0
```

Because the variable **livesLeft** has a value which is greater than 0 the print statement results in **True** being displayed. Another example, checking that a string a user enters is equal to the required password:

```
password = raw_input("Enter your password:")
print "Correct password:" + str(password == "letmein")
```

Boolean conditions are used extensively within most programs you will write. In later weeks you will see how they can be used to make decisions based on the outcome of the condition.