**Session 1 Notes**

**Getting started**

In this course we will use the RaspberryPi and the Zumo 32U4. The Zumo needs to be programmed using a version of the C language developed by Arduino while the RaspberryPi can be programmed using Python. We will be using an open source library developed by lecturers here at CDU which allows us to program only in Python and to send the commands to the Zumo, which means that you will only need to learn the language Python and the library will take care of the translation of the commands into C.

In this first session we will simply be learning the basics of Python and the RaspberryPi, in the next session we will start using the Zumo board and commence some simple robotics. You could practise your Python on any computer, check the support materials for more information.

The place where most people start programming is the traditional Hello World. This means we get the computer to print these words on the screen. So that is where we will start. If we can do this it means that everything is set up properly and we can concentrate on learning Python.

Turn on the RaspberryPi which has already be loaded up with the programs that we need for this course. Go to the Menu and select Programming – Geany Programmer’s Editor. The screen should look like this:



Before we go any further there are two checks that we need to make to ensure that everything will work. Go to Build -> Set Build Commands and make sure that both the Compile and the Execute section are set to Python3. This makes sure that Geany is using the correct version of Python. Also go to Edit -> Preferences. Then go to the Edit->Indentation section. The width should be 4 and the Type should be Spaces. The means that we can use the tab and enter keys and Geany will put spaces into our file (which Python requires).

In the untitled section, which is a text editor, type in the following:

#! /usr/bin/env python3

print("Hello world")

Now we need to save it so go to File-Save as, first create a directory (folder) for your files and then save it using the file name hello.py. To run the file inside Geany press the Run button (it looks like an arrow). This should open a terminal window and put the word Hello World on the screen.

You can also run this command by navigating to the file in the GUI and double clicking the file name or by navigating to the folder in the terminal and typing in the command

python3 hello.py

So how does this work.

#! /usr/bin/env python3

This is a command for Unix based the operating systems (Linux and Macintosh but not Windows) which says where the Python executable file is located in the system. Most times this information is not needed and can often be left off, however it may be important one day so it is a good idea to have it as the first line of all files. More information about this command can be found here <http://stackoverflow.com/questions/2429511/why-do-people-write-usr-bin-env-python-on-the-first-line-of-a-python-script>

**print** – is a built in Python command (actually it is a function) which will output the data it is given to the console. On the Raspberry Pi the console is the Terminal. On Windows machines the console is the DOS window, while some IDEs (Integrated Development Environments) like Wing IDE or PyCharm have their own console built in. Geany uses the terminal.

**()** – these are used in Python to provide the function with the data it needs. When using the round brackets with functions (like here) the data is called **arguments (or parameters)**, you often see the two terms used (wrongly) interchangeably. If there is more than one argument/parameter they are separated by commas. There is an actual difference between the two terms (arguments and parameters) in programming and it might be worth searching for the StackOverflow discussion if you are interested, basically they are parameters when you create a function and arguments when you call a function, but we are getting ahead of ourselves.

**"Hello World"** – this is the data. Since it is in quotation marks Python knows that this data is ordinary words. In computer terms this type of data is called a **String**. The print function knows that **Strings** are printed out exactly as they are presented and will place a new line at the end. In Python **Strings** can be marked with a single quote ', double quote " or triple quote """ or '''.

See if you can change the code so it outputs your name instead of Hello World.

*NB. An essential part of programming is learning exactly what the commands (either built in or in the libraries) actually do and what arguments are required. All well developed languages have a wide range of these* ***libraries*** *of commands and part of mastering a language is understanding how to use these commands. No course can ever cover all the commands, it is up to you to learn how to find this information when you need it. What we will cover is the pattern of forming a command. Learning a language involves understanding its patterns.*

*One solution is to buy a good reference book about the language, a second solution is to read the documentation that comes with the libraries, a third option is to learn to use search engines on the internet. Many people start learning a language by buying a good introductory book. Some can also be the reference book, but most don’t have enough information to do both jobs well.*

Now let’s try to get the computer to add two numbers together using Python.

print(5+6)

Change your code to the code above and try to run it. What happened?

**print** works with numbers as well as words. In Python there are two basic types of numbers, **Integer** and **Float**. An **Integer** is a whole number, much like the everyday number that we normally use. We will discuss **Float** later since it looks like a decimal but is actually very different. 5 and 6 are both **Integers** so Python works out the result.

If you want Python to print both the question and the answer you must tell it to print both, however you can’t mix your data types so you need to separate the data using commas

print("5+6= ", 5+6)

Try getting the computer to do all four possible calculations (addition, subtraction, multiplication and division). You now have the start of a calculator!!

**Floats**

In Python **Integers** can be any size so it is a good data type for day to day numbers, however for decimals we need another type of data called a **Float**. These use IEEE 754 single-precision binary Floating point format which is built into the computers CPU. What this means in practice is that **Floats** are not accurate. Try

print(.1+.2)

Python will use **Integers** where it can and will use **Floats** if **Integers** don’t work. If you give it **Integers** and **Floats** the answer will be in **Floats**. For division it will always use **Floats**.

**Functions**

**Functions** are the basic building blocks to programming so let’s create one now. Change your code to the following and run it:

def fun1():

 print("Hello World")

fun()

You should get an error like this:



When Python runs any program it will print an error message when it comes to a command that it does not know. The problem here is in on line 6. We created a function correctly on lines 3 and 4, but we gave it the name fun1. On line 6 we tried to call this function but gave it the wrong name which the error message is trying to tell us. Change line 6 to the correct name and run it again.

We have done something important here, we have create our own function called fun1 and also run it. Let’s look at how it was created on line 3.

**def** – This is a Python key word which means that on this line we are creating a function. A function is a small block of code which can perform a small task. It is particularly useful when we want to run the task many times in our programming. Think of functions as the bricks that we use to build a wall (our program).

**fun1** – this is the name of the function and can be almost anything that follows the naming rules of Python. These rules apply to any name that we create.

The naming rules for Python are:

1. Names must start with a letter, and can include letters and numbers.
2. Names cannot contain spaces.
3. Names should be meaningful

*There are also a lot of conventions around names, but generally keep names short and meaningful. Python has its own style guide which does cover names called PEP 8 (*[*https://www.python.org/dev/peps/pep-0008/*](https://www.python.org/dev/peps/pep-0008/)*).*

*It also has its own Zen (*[*https://www.python.org/dev/peps/pep-0020/*](https://www.python.org/dev/peps/pep-0020/)*)*

(Looking at Pep8 and the Zen fun1 is a bad name since it does not mean anything so is not very readable).

**()** – we saw this before when we discussed **print**. It’s where you put the data that the function can use. Currently we are not going to need any data so we can leave this blank, we will explain how to bring data into a function soon.

**:** - This creates a block of code, which is indented. These blocks are the foundations of programming. The secret to writing good code is writing good blocks. Ideally they should be short, do one thing and do it well.

Here we are giving the block a name. We can put as many Python commands in here as we like, including library functions and our own functions. Blocks are finished when the indentation finishes. Indented blocks like this is used for other Python commands as well.

Now let’s get our function do some real work. Change fun1 to the following

def fun1():

 return 5+6

print(fun1())

The only new thing here is the keyword **return** which tells Python that this is the data that is being sent back from this function. In the last line, all of fun1() is replaced with the actual data that comes from fun1. Run this to see the result.

**Variables**

Using the actual numbers is not good practise in programming, it is far better to use variables. Variable are names which stand in the place of numbers and other types of data and are very useful when you don’t know what numbers (data) will be used. For example if we are building a calculator, as we will later, the numbers will come from the user at runtime.

Create a new function called adding\_machine which will be called immediately below by entering the following:

def adding\_machine(){

 num\_a = 5

 num\_b = 6

 total = num\_a + num\_b

 print("The total of ", num\_a, "and", num\_b, "is", total)

adding\_machine()

Study this carefully and see if you can work out how to use this as a basis for other functions to do calculations beside addition, with the correct message.

*NB there are a number of ways to write names, however programmers usually follow a naming convention to make this easier. There are two basic conventions either use the underscore \_ to\_separate\_the\_words (which is the convention for PHP and Python) or could also use camelCase (which is the convention for Java and C).*

*One of the signs of a good program is that it should be easy to read and understand for someone who is familiar with the language. This means that the variables names should tell you what the data is and the function names should say what they do. Many programming languages and companies have style guides to help people make their code readable to other programmers. As mentioned earlier, Python has Pep8.*

This is the start of the calculator.

**num\_a = 5** - num\_a is a variable which only exists inside the function, you can’t use it outside the function. These types of variable are called local variables and most programmers write their programs using as many local variables as possible.

What happens is the Python reserves a small area of memory and assigns to that area of memory the name given to the variable. When that name is used later on in the program the value that is contained in that area of memory is used. Here the name is num\_a and the value that is placed into the memory is 5. The best way to think about this is as a box. On the outside is a label (num\_a) and on the inside is the data 5. Python does not care what type of data is placed in the box, however there must always be something, you can’t have a variable with no data.

The rest of this code you should now be able to work out. To build a calculator you first need to build a number of functions to handle each of the separate calculations.

Building a calculator as using our current pattern will lead to a lot of duplicate code. Although each machine that we build to do the calculation (adding\_machine, subtraction\_machine etc) will need their own results and display, we only need one set of starting numbers which can be shared with all the machines (functions). To do this we need a way to pass information into the functions. This is the job of parameters. Change the first line of the adding\_machine to

def adding\_machine(num\_a, num\_b):

and then you can delete the first two lines of the function. num\_a and num\_b are now **parameters** of the function. In Python to create a parameter you only need to give it a name to be used inside the function. If you have more than one parameter you separate them with commas. Parameters are local variables which receive their data from outside the function. Python does not care what type of data is passed as a parameter, it’s up to the programmer to work that out. To use this function change the call:

adding\_machine(5,6)

Now our adding\_machine is more useful since it can be called with any numbers, for example

adding\_machine(5,6)

adding\_machine(-4,26)

adding\_machine(1000,4545)

addingMachine(3\*100,6)

You should now be able to build three more calculation machines (functions) for subtraction, multiplication and division.

**User Data**

We now have four machines to calculate our answer, however to get them to work we are still entering the data as part of the program. We need a way to get the data from the user. In Python that command is input. Type the following into a new file

name = input("What is your name? ")

print("Hello", name)

There are a couple of new things here but run this program to see how it works.

**name = input("What is your name?** **" )**  – **input** is another built-in function in Python. It takes information from the keyboard and puts the data into the variable **name**. The data that you give to **input** (as an argument) is displayed on the console.

So now that we know how to get information from the keyboard, let’s write a program to run our adding machine. Type the following and run it:

def adding\_machine(num\_a, num\_b):

 return num\_a + num\_b

num1 = input("First number ")

num2 = input("Second number ")

print(adding\_machine(num1 + num2))

*This does not work as expected, however Python is doing the best that it can. It is doing exactly what you told it to do, not what you wanted it to do.*

The problem here is an issue of data types. We know that we expect to have a number but what does Python know about the data coming from the keyboard? It could be a number or it could be a **String**, it does not know. So to be safe it always assumes that it is a **String**.

There is a command in Python to turn a **String** into an **Integer**, that function is **int**(). Change the following:

num1 = input("First number ")

num1 = int(num1)

num2 = int(input("Second number "))

This shows both ways this could be done, on a separate line or by chaining the commands together.

We are almost there at the calculator, however we need to get Python to make a decision about which machine to use.

**Making Decisions**

Now we need one more Python command to be able to complete a simple calculator program. We need a way to choose which of our own functions to use. The basic way to make choices in programming uses the key word **if**.

Create a file and enter the following:

name = input("What is your name? ")

if name == "Charlie":

 print("Welcome back Charlie, how was golf.")

if name == "Gloria":

 print("Hi Gloria, where's my wallet?")

print("Good to see you!", name)

See if you can work out what it will do.

The new things here are in the **if** statements. The first one is followed by **name == "Charlie"**. This is an **expression** which must be true or false. In this case the **==** means that we compare **name** and "Charlie" so see if they are the same. We use **==** not **=** since **=** already has a use, it is used to give variables their value, called **assignment**.

You now have all the building blocks you need to make your first program, a simple calculator. Building the actual calculator will be your first Practical where we will introduce you to Scrums and Sprints to organise how to get the software written.

**Glossary**

|  |  |
| --- | --- |
| Concepts | Meaning |
| Command | These are key words used by the language that perform a function for the language. It is also possible to create your own commands to be used in your programming. Most languages have libraries of commands that have already been built for you to use as well as those you build yourself. |
| Functions | The most common way to create your own commands. These can accept data, process data in some way and can return the changed data to be used in other parts of the program. In a well structured program almost everything is done in small single purpose functions. |
| Arguments | These are bits of data that are used by functions, so that they have information to work with. |
| Parameters | These are the place holder variables that are used in function definitions. These are replaced by the arguments that are used when the function is run. Often the two terms are used interchangeably. |
| String | Data type, String refers to ordinary words.  |
| Integers | Data type, int refers to whole numbers.  |
| Float | Data type, Float used for decimals, but they are not accurate.  |
| Variable | Way of representing data for the program to work on. |
| Libraries | These contain commands that have been developed and tested and are ready to use. Many of the libraries have been written by the people who originally developed the language while other libraries have been developed by companies or individuals that use the language (Google, Yahoo, Apache etc have all developed extensive language libraries for a wide variety of languages). You can also develop your own library of commands. |
| Assignment | Uses = to give a variable name a value |
| Expression | An operation which must be true or false |
|  |  |
| **Python Commands** | Meaning |
| print | Python function which will show the data on the console |
| def | Python key word used to define your own functions |
| input | Python function which gets data from the keyboard |
| int | Python function which changes a String to its Integer value |
| if | Python keyword used to build decision making structures, must be followed by an expression which is either true or false |