**ICTPRG301**

**Lesson 1 Notes**

**Getting started**

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.

In this course we will use Wing IDE101 for our demonstrations. There are separate instructions in the support materials on how to install this program. There are also instructions on how to install and use the Eclipse IDE if you would prefer to use that. You can also just use a text editor or any other IDE of your choice, however we have not provided instructions for these options.

Load up Wing IDE 101 and you should see this screen.



Python shell

Python program editor

The top section works like a simple text editor and is where you create your Python programs. The section in the bottom right is a Python shell. It is where you can test commands. In the Python shell type in :

print("Hello World")

and press enter. It prints the words Hello World in the console.

This is how it works:

print – is a Python **command** which will print out the contents to the console. If used in the Python shell the console is part the shell, if used in a program it will open a console for you or use the one already opened.

() – these are used in Python to provide the command with the data it needs. This data is called **arguments or parameters**. In the case of print, the arguments (parameters) are the data that will be printed. 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)

“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 command knows that Strings are printed out exactly as they are presented.

By typing this into the shell we are just testing how the command works. This gives us confidence that we know what the command will do as part of a program. Let’s create a small program now to do the same thing.

Go to the File menu and press New. This creates a new file. In the file type

print(‘Hello World’)

(NB this time you use single quotes not double quotes. Python allows you to use either so long and you are consistent)

Then save the file to your folder. (I used the file name lesson\_11). Wing should look like this



and the folder where it was saved should look like this:



Wing put the extension .py on the file. Double click the file and if you are quick you will see the console open and print the words, then close again. (There is a way to keep the console open which will be explained at the end of this session.)

In Wing there is a way to run the program to the shell, this is done with the run button, which looks like this:  on the tool bar. Press this button and the shell resets and runs your program. It will take a few moments.

**Activity 1.1**

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 parameters 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. Most 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 write some more code. In the shell write

5+6

Now before you run the code work out what will appear in the console. Then run the code, were you right?

The shell prints out the answer. Python is a calculator and works out the answer before it prints it.

To get your program to print the answer you must tell it to do this. In the program write

print(5+6)

Then run the program again.

If you want Python to print both the question and the answer you must tell it to print both:

print(‘5+6=’,5+6)

the comma is used to separate the parts of the print statement when you are joining two types of data together into one display.

**Activity 1.2**

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

**VARIABLES**

Using the actual numbers is not good practise in programming, it is far better to use variables. Variable are words or letters 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 in this lesson, the numbers will come from the user at runtime.

Create a new file called lesson\_12 and enter the following:

number\_a = 5
number\_b = 6
total = number\_a + number\_b
print("The total of ", number\_a, " and ", number\_b ", is ", total)

**Activity 1.3**

Study this carefully and see if you can work out how to change it so that it can do other calculations beside addition, with the correct message.

*NB there are a number of ways to write programs, however programmers usually follow a naming convention to make this easier. The convention for python is to use the underscore \_ to separate\_the\_words (which is the standard for PHP). Be consistent. You could also use camelCase which is the convention for Java and C languages*

*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 command names should say what they do. Many programming language and companies have style guides to help people make their code readable to other programmers. The two most important for Python are PEP 8 and PEP 20, found at* [*http://www.python.org/dev/peps/pep-0008/*](http://www.python.org/dev/peps/pep-0008/) *and* [*http://www.python.org/dev/peps/pep-0020/*](http://www.python.org/dev/peps/pep-0020/)

This is the start of the calculator, however we are still entering the information into the program, what we want is for the user to enter the value. To do this we need another command called input.

Clear the last file and type this in:

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

print('Hello ',name)

run this file and see what happens. Run this from the file and see what happens.

Now let’s use all these commands to develop a small calculator program that adds two numbers together and gives us the answer:

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

print('Hello ',name)

print('Let me add some numbers')

num\_a = input('What is the first number')

num\_b = input('What is the second number')

print('Your total is ', num\_a + num\_b)

input('Press enter to continue')

What is wrong, why doesn’t this work (did you notice the last statement, this holds the console open until we want to close it so this file can be run by double clicking the file as well as within Wing).

The problem is with the variable num\_a and num\_b. You know that you want them to be a number (in programing whole numbers are called Integers), but how does Python know this. The short answer is that it doesn’t know, it works it out when the variable is given a value. Since the value comes from the input command (that is it comes from what you type) Python does not know what you are going to type. However for Python the safe way of assigning a value is to use the data type String.

There is a command to turn a String into a number. The command is int(). Add these lines after the input statements

num\_a = int(num\_a)

num\_b = int(num\_b)

Now the program works as you intended. (Try typing in a word instead of a number and see what happens, you will learn how to fix this in lesson 6)

You have now seen three of Python’s build in commands (print, input and int). Here is the list of all these commands (Python calls these commands functions):

Built-in Functions (also found at <http://docs.python.org/py3k/library/functions.html#ascii>)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [abs()](http://docs.python.org/py3k/library/functions.html#abs) | [dict()](http://docs.python.org/py3k/library/functions.html#func-dict) | [help()](http://docs.python.org/py3k/library/functions.html#help) | [min()](http://docs.python.org/py3k/library/functions.html#min) | [setattr()](http://docs.python.org/py3k/library/functions.html#setattr) |
| [all()](http://docs.python.org/py3k/library/functions.html#all) | [dir()](http://docs.python.org/py3k/library/functions.html#dir) | [hex()](http://docs.python.org/py3k/library/functions.html#hex) | [next()](http://docs.python.org/py3k/library/functions.html#next) | [slice()](http://docs.python.org/py3k/library/functions.html#slice) |
| [any()](http://docs.python.org/py3k/library/functions.html#any) | [divmod()](http://docs.python.org/py3k/library/functions.html#divmod) | [id()](http://docs.python.org/py3k/library/functions.html#id) | [object()](http://docs.python.org/py3k/library/functions.html#object) | [sorted()](http://docs.python.org/py3k/library/functions.html#sorted) |
| [ascii()](http://docs.python.org/py3k/library/functions.html#ascii) | [enumerate()](http://docs.python.org/py3k/library/functions.html#enumerate) | [input()](http://docs.python.org/py3k/library/functions.html#input) | [oct()](http://docs.python.org/py3k/library/functions.html#oct) | [staticmethod()](http://docs.python.org/py3k/library/functions.html#staticmethod) |
| [bin()](http://docs.python.org/py3k/library/functions.html#bin) | [eval()](http://docs.python.org/py3k/library/functions.html#eval) | [int()](http://docs.python.org/py3k/library/functions.html#int) | [open()](http://docs.python.org/py3k/library/functions.html#open) | [str()](http://docs.python.org/py3k/library/functions.html#str) |
| [bool()](http://docs.python.org/py3k/library/functions.html#bool) | [exec()](http://docs.python.org/py3k/library/functions.html#exec) | [isinstance()](http://docs.python.org/py3k/library/functions.html#isinstance) | [ord()](http://docs.python.org/py3k/library/functions.html#ord) | [sum()](http://docs.python.org/py3k/library/functions.html#sum) |
| [bytearray()](http://docs.python.org/py3k/library/functions.html#bytearray) | [filter()](http://docs.python.org/py3k/library/functions.html#filter) | [issubclass()](http://docs.python.org/py3k/library/functions.html#issubclass) | [pow()](http://docs.python.org/py3k/library/functions.html#pow) | [super()](http://docs.python.org/py3k/library/functions.html#super) |
| [bytes()](http://docs.python.org/py3k/library/functions.html#bytes) | [float()](http://docs.python.org/py3k/library/functions.html#float) | [iter()](http://docs.python.org/py3k/library/functions.html#iter) | [print()](http://docs.python.org/py3k/library/functions.html#print) | [**tuple()**](http://docs.python.org/py3k/library/functions.html#tuple) |
| [callable()](http://docs.python.org/py3k/library/functions.html#callable) | [format()](http://docs.python.org/py3k/library/functions.html#format) | [len()](http://docs.python.org/py3k/library/functions.html#len) | [property()](http://docs.python.org/py3k/library/functions.html#property) | [type()](http://docs.python.org/py3k/library/functions.html#type) |
| [chr()](http://docs.python.org/py3k/library/functions.html#chr) | [frozenset()](http://docs.python.org/py3k/library/functions.html#func-frozenset) | [list()](http://docs.python.org/py3k/library/functions.html#list) | [range()](http://docs.python.org/py3k/library/functions.html#range) | [vars()](http://docs.python.org/py3k/library/functions.html#vars) |
| [classmethod()](http://docs.python.org/py3k/library/functions.html#classmethod) | [getattr()](http://docs.python.org/py3k/library/functions.html#getattr) | [locals()](http://docs.python.org/py3k/library/functions.html#locals) | [repr()](http://docs.python.org/py3k/library/functions.html#repr) | [zip()](http://docs.python.org/py3k/library/functions.html#zip) |
| [compile()](http://docs.python.org/py3k/library/functions.html#compile) | [globals()](http://docs.python.org/py3k/library/functions.html#globals) | [map()](http://docs.python.org/py3k/library/functions.html#map) | [reversed()](http://docs.python.org/py3k/library/functions.html#reversed) | [\_\_import\_\_()](http://docs.python.org/py3k/library/functions.html#__import__) |
| [complex()](http://docs.python.org/py3k/library/functions.html#complex) | [hasattr()](http://docs.python.org/py3k/library/functions.html#hasattr) | [max()](http://docs.python.org/py3k/library/functions.html#max) | [round()](http://docs.python.org/py3k/library/functions.html#round) |  |
| [delattr()](http://docs.python.org/py3k/library/functions.html#delattr) | [hash()](http://docs.python.org/py3k/library/functions.html#hash) | [memoryview()](http://docs.python.org/py3k/library/functions.html#func-memoryview) | [set()](http://docs.python.org/py3k/library/functions.html#func-set)  |  |

In addition to these functions there is also a list of word that Python understands and can be used as commands. These are called **Key words** or Reserve words and cannot be used as variable or function names. The complete list is:

<http://en.wikipedia.org/wiki/Python_syntax_and_semantics#Keywords>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| and | del | from | None | True |
| as | elif | global | nonlocal | try |
| assert | else | if | not | while |
| break | except | import | or | with |
| class | False | in | pass | yield |
| continue | finally | is | raise |  |
| def | for | lambda | return |  |

NB all these are in lower case except True, False and None.

These are the basic building blocks of the language, everything else are commands developed by other people or by you. So let’s create our own command. Create a new file called lesson\_13 and enter the following:

def neil(text):

 print(“Neil says: “,text)

neil("Hello World")

This is a stupid program, but it does show how commands are constructed. We have created a command called neil which modifies the print command. Let’s look at it line by line.

def neil(text):

The command **def** is a Python Key word. This command is used to create commands.
**neil** is the name we give to our command
**()** holds the parameters
**text** is a variable name (parameter)that will be used within the command that we are creating. When the command is run text will change to whatever we set at the time.
**:** is very important. It means that we are finished defining the name of the command and that what follows is the actual command structure we want run every time the command is called.

 print(“Neil says: “, text)

This the block of Python code that we want to run each time that command neil is called. The Python commands here you should now understand. However note carefully the setting out. Python indents the block by two spaces (or a combination of two spaces NOT TABS). Everything that stays within the indentation pattern is part of the block. The IDE Wing (or Notepad++ if set up for Python) creates the block using the TAB key, however other text editors may not. Whatever you use the spacing must be consistent. This is particularly important if you move a file from one text editor to another, it may not have the spacing set properly.

nei(“Hello World”) this is outside the block so is not part of the command block. It actually runs the command and takes the argument “Hello World” which it knows is a Strings because they have quotes around them and places the argument in the block of code replace the parameter text wherever it appears in the block.

*NB Python functions must be defined before they are used. If you swap the code around so it is used before it is defined you will get an error. Getting errors like this is not a disaster, they actually become your friend. These errors tell you exactly what is wrong with your code and hint as to how to fix it. Creating code follows a cycle:*

1. *write some code*
2. *look at the results (errors)*
3. *fix the errors,*
4. *get exactly what you want*
5. *write some more code*

*Errors in code are often called bugs, and* ***Debugging*** *is an important skill in becoming a good programmer.*

Activity 1.4

See if you can work out what this does?

neil (5+6)

Does it: A. print out Neil says: 5+6
 B. print out Neil says: 11
 C. print out 11
 D. Throws an error

Work out and answer then try it for yourself to see if you were right.

The functions you create yourself can have more than one parameter, these are separated by commas. Modify the file like this as see how this works.

def neil(text1, text2):

 print(“Neil says: “, text1,text2)

neil("Hello", " World")

neil(5+6, " is the answer")

*NB Python does not care whether the argument is a String or an integer. It works this out when the function is used, it is up to you to use it properly.*

Now that we can make our own commands 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 commands to use. The basic way to make choices in programming uses the key word if.

Create a new file (lesson\_14) and enter the following:

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

if name=='Charlie':

 print('Good to see you again ',name)

if name=='Gloria':

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

print('Welcome back')

*NB the if statement is followed by a block of code so it follows the same rules as the def statement using spaces to show where the block starts and ends. Also note the == two equals signs. One equals is used to set the variable to that data (technical term for this is assignment), two equals is use to compare two bits of data to see if they are the same. Also notice that we needed an apostrophe for Gloria, by using double quotes we can use a single quote inside.*

Run this to see if you understand how it works.

Whew, you now have all the building blocks you need to make your first program, a simple calculator. You know the following commands:

int(), input(), print() and the key words def and if

Activity 1.5

That should be all you need to make a calculator. The flow of the program is:

Create four commands using def for the four operations (Optional but good practise)

Ask the user for the first number (using input)

Ask the user for the second number

Ask the user for the type of operation

Use if to work out which command to use

Good Luck!!!

**Glossary**

|  |  |  |
| --- | --- | --- |
| Lesson | Term | Meaning |
| One | Command | These are key words or phrases 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. |
|  | Arguments | These are bits of data that are used by commands, 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, there are a wide variety of types of data, String refers to ordinary words. Some other types include Integer (int) which refers to ordinary numbers, float which refers to numbers which have decimal values, boolean which refers to two valued True and False, etc |
|  | 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. |
|  | Variable | Way of representing data for the program to work on. |
|  | Key Word | Key words are the basic building blocks of any language. These words are reserved so they cannot be used as variable or command names. Most languages do not have many key words. |
|  | Debugging | All programs have errors because they are written by humans. Machines don’t make errors, they do exactly what you tell them, which often is not what you want or thought you actually told them. These errors can be simple syntax error or complex logic errors. The name programmers give to these errors is bugs. Some programmers estimate that 80% of the time they spend on programming is spent on testing and debugging, especially when they have to work on code which they did not write.The only programmers who don’t make errors (bugs) are those who don’t write code, all the rest of us make errors all the time; which is why we spent so long testing our code. |
|  |  |  |