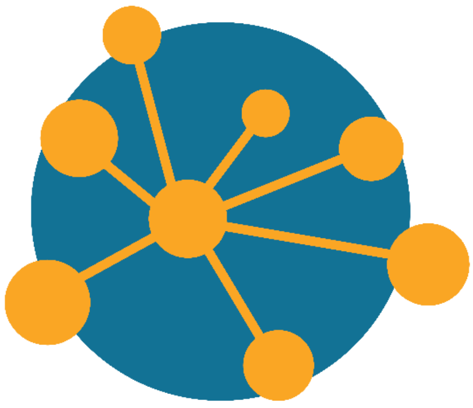
Digital Technologies in focus:

Supporting implementation of Digital Technologies



Python tutorial

AUSTRALIAN CURRICULUM, ASSESSMENT AND REPORTING AUTHORITY

Initiative of and funded by the Australian Government Department of Education and Training

Acknowledgements

The Curriculum group of ACARA has developed this professional learning program to support teachers in the 3-year National Innovation and Science Agenda (NISA) project known as Digital Technologies in focus: Supporting implementation of Digital Technologies.

For more information contact:

Julie King

Project Lead, Digital Technologies in focus

Curriculum Specialist, Technologies

julie.king@acara.edu.au

TABLE OF CONTENTS **Page**

GETTING STARTED WITH PYTHON

Python is one of more than 250 high-level programming languages, many of which are referred to as general-purpose programming languages. These types of languages, also known as text-based languages, require the coder to key letters, numbers and symbols to issue instructions. General-purpose programming languages use formal syntax, unlike visual languages, where syntax is 'hidden' in blocks of code. In the Australian Curriculum: Digital Technologies, students are required to study a general-purpose programming language at Years 7 and 8. In particular, students should be able to implement programs that include branching (conditions) and iterations, such as the 'for' loop. These will be covered in this tutorial.

This tutorial will help you understand the basics of Python.

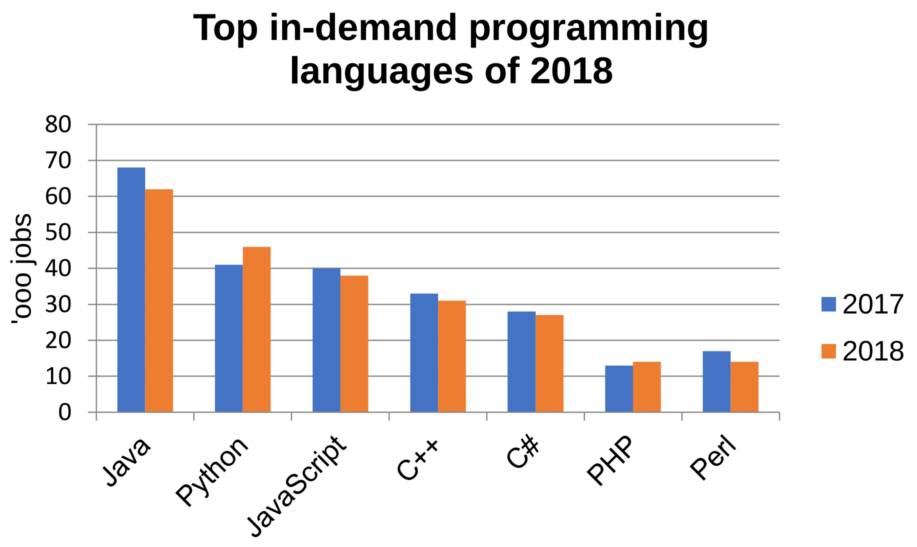
About Python

Python is a simple and incredibly readable programming language, as it closely resembles the English language. It's a great language for beginners all the way up to seasoned professionals.

Python is used in many universities and schools, and as a consequence there are lots of libraries created for Python related to fields, such as mathematics, physics and natural processing.

Python is also used widely in industry. According to the US job website *Indeed*, Python and PHP are the only programming languages that have experienced an increase in the demand for job postings from 2017 to 2018, as shown in the following chart.

CHART 1: JOB POSTINGS CONTAINING TOP PROGRAMMING LANGUAGES



Reference: 'The 7 most in-demand programming languages of 2018', *Coding Dojo*, 13 December 2017, **https://www.codingdojo.com/blog/7-most-in-demand-programming-languages-of-2018/**

What to expect

In this tutorial of 16 programs, you will learn about:

- inputs and outputs
- variables and their scope
- strings and numbers
- lists or arrays
- loops
- conditionals (if, elif, else)
- functions
- return values
- modules
- import libraries
- random numbers
- integrated development environments (IDEs).

Loading Python

First, you need to put Python on your computer. You will load two programs – Python itself as well as a Python Integrated Development Environment (IDE), which will help you with your coding.

First, let's load Python itself.

Mac users

Download the latest version from Python at **www.python.org/downloads/mac-osx/**

The latest version is usually at the top of the list. The Mac OS comes with Python 2.7 included. This is fine as a starter, but it has some different syntax to the newer versions and this tutorial uses the newer (3.x) versions.

PC users

Download the latest version from Python at **www.python.org/downloads/windows/**

The latest version is usually at the top of the list.

If you have another PC operating system such as Linux, visit Python at **www.python.org/downloads/** to see if your operating system is supported.

Installing (Mac and PC)

Once downloaded, click or double-click to install the package and follow the prompts. Just choosing the defaults will be fine.

Python is all you really need; however, you will also install PyCharm so you begin to learn about the wonderful world of programming and the sort of support that can be built into the programming environment.

Note: It is important to load the Python program first, before downloading and installing PyCharm.

When you are ready (after installing Python), if you have a Mac, go to Jet Brains at
www.jetbrains.com/pycharm/download/#section=mac

or, if you have a PC visit Jet Brains at
www.jetbrains.com/pycharm/download/#section=windows

In either case, choose the ‘Community’ download as it is free, unless you are willing to pay for the professional version.

Once the PyCharm download has been completed, click or double-click to install the software. Again choose the defaults and you should have no issues.

PROGRAMS

There are 16 programs in this tutorial, some more extensive than others. Initially the instructions for each program are detailed and supported by images, but this gives way to briefer instructions, which should coincide with your growing confidence with Python.

Program 1: Introductory code

You are going to create your first program by running the Python program. When you load the program – look for IDLE Python – a window will appear, which is headed 'Python 3.6.5 Shell'.

You could do a lot just here, like maths or print statements.

Type:

13 + 29 and press enter.

Now try:

16/4 and see what happens.

Finally try:

16/0. Any comments?

You've made your first mistake – the good news is you can ignore all but the last line. Computers don't like dividing by zero.

Type:

print ("Hello World") and press enter.

Try it with single quotation marks (' ') instead of doubles. Try it without the ( ) brackets – another error. If you don't get an error message here it means you have an earlier version of Python (2.7ish). It's no big deal but there are some differences.

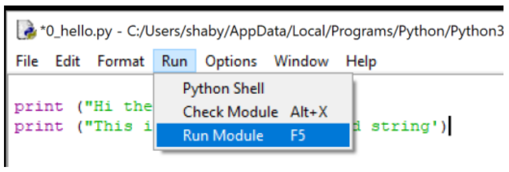
Go to **File/New file**. A new window will appear called 'Untitled'. This is where you will put your code.

Type the following two lines.

print ("Hi there")

print ("This is" + ' a concatenated string')

Then go to 'Run/Run Module', as shown in Figure 1.



```
print ("Hi the
print ("This i                    d string')
```

Figure 1: Python prompt to run module

A window will appear asking you to save your work. To keep it all together you will create a new folder called ‘Workshop’ and put your program there, as shown in Figure 2.

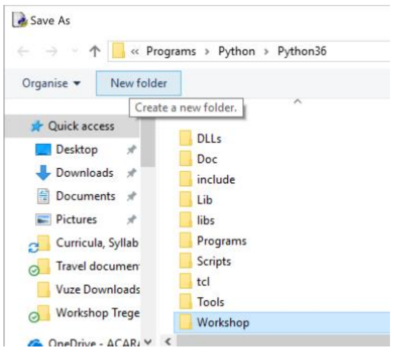
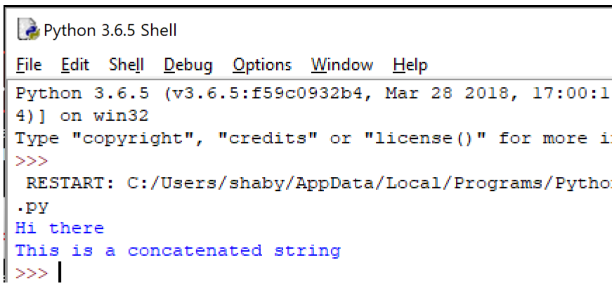


Figure 2: Python prompt to create a new folder

Tip: Make sure you know where you are saving your programs as it makes it easier to come back to them later.

After the program saves, you should see on the ‘Shell’ window, the result of your coding, as shown in Figure 3.



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:1
4)] on win32
Type "copyright", "credits" or "license()" for more i
>>>
 RESTART: C:/Users/shaby/AppData/Local/Programs/Pytho
.py
Hi there
This is a concatenated string
>>> |
```

Figure 3: The Shell window shows the result of coding

Loading a different integrated development environment (IDE)

Congratulations! You have written your first code. Now you are going to write the code again using the other program that you downloaded called ‘PyCharm’. PyCharm is an integrated development environment (IDE) – a place where you can write code.

PyCharm doesn’t have the Python ‘engine’ so it will not work without the Python program (the one you just used) having been loaded onto the computer first. The Python engine has its own IDE called IDLE.

Why are you doing this? It’s to show that there are differences in IDEs, and also to give you a choice.

Locate the shortcut that looks like the one shown in Figure 4 or search for 'Jetbrains' in your programs and load it from there.

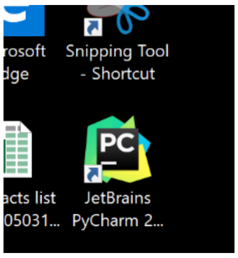


Figure 4: Icon to access professional developer tools from JetBrains

You might need to tell the program which interpreter to use. This is important. You want to use the one that came with Python that you downloaded, so you need to map to that if it isn't already mapped.

When the program loads it looks like Figure 5.

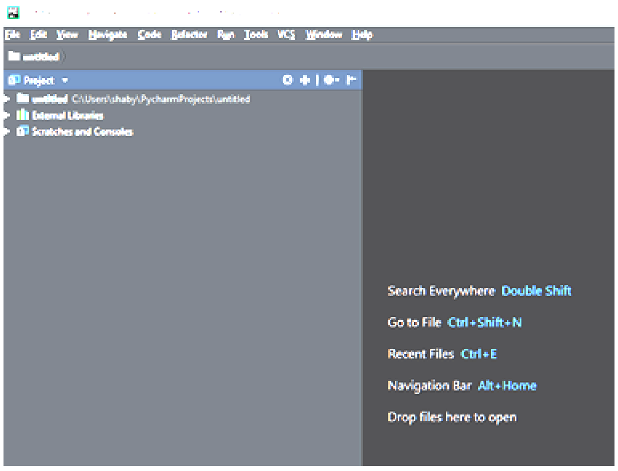
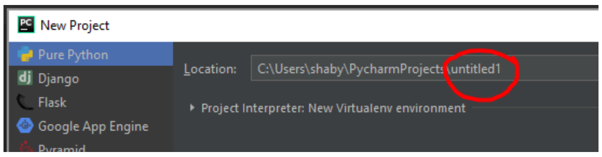


Figure 5: Screen image when program is loaded

Note: You get one chance to easily rename the project from 'untitled' to something like 'Workshops'; however, it is not essential that you do so.

When you start, the program wants to know where you want to save your work. It calls it 'untitled' after a path to a folder called 'PyCharm Projects', as shown in Figure 6. Just rename the end part to 'Workshops'.

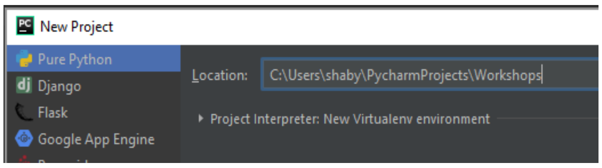


Location: C:\Users\shaby\PycharmProjects\untitled1

Project Interpreter: New Virtualenv environment

Figure 6: Python example of stating the location for a saved program

You are doing this so that you know where to find your programs – it is good practice to do so but it is not a problem if you leave it at ‘untitled’. Figure 7 shows the renamed location.



C:\Users\shaby\PycharmProjects\Workshops

Figure 7: Python example showing the renamed location for a saved program

Now you are ready to create your first program again.

Here it is if you don’t remember:

print ("Hi there")

print ("This is" + ' a concatenated string')

To do this in PyCharm, right-click on the Workshops (or untitled) folder and choose **New/Python file**, as shown in Figure 8.

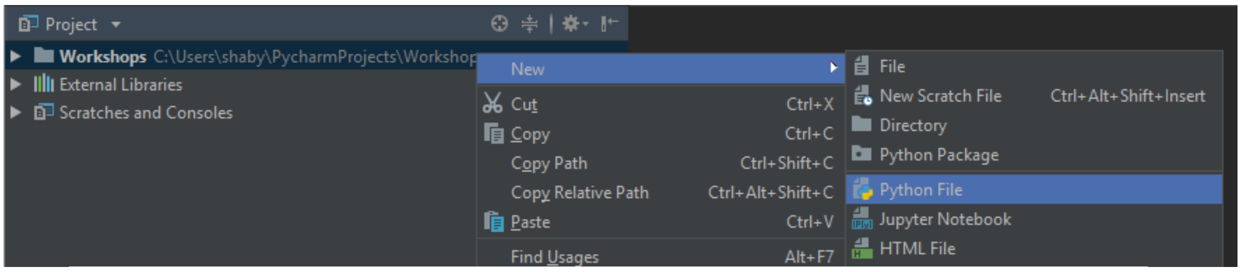
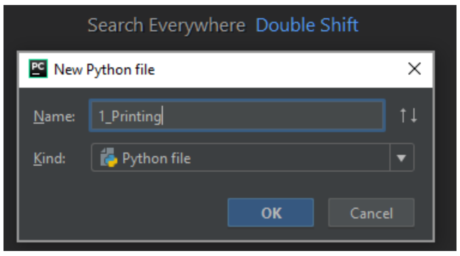


Figure 8: Python techniques to create a new file in PyCharm

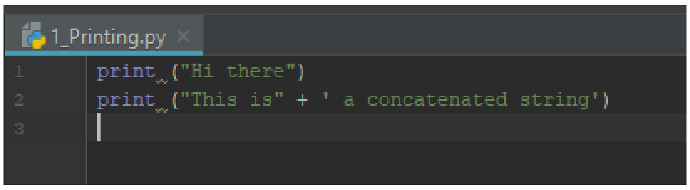
You’ll be asked for a name – call it ‘1_Printing’, as shown in Figure 9. Click OK to enter the name.



Name: 1_Printing
Kind: Python file

Figure 9: Naming a new file in PyCharm

The coding area then comes up where you can type in your code. It will look like Figure 10.



1_Printing.py

```
print ("Hi there")
print ("This is" + ' a concatenated string')
```

Figure 10: Coding area for file ‘1_Printing.py’

Hopefully you noticed a couple of nice things as you typed. What were they?

IDEs are designed to help the programmer. Some help more than others. You need to choose which one you are going to work with for the rest of this tutorial and probably stick with it. The PyCharm IDE will be used in this tutorial as it has another nice feature that you will see in a moment.

To get the program to run, you need to right-click the **1_Printing.py** tab at the top of the coding window and choose **Run '1_Printing',** as shown in Figure 11.

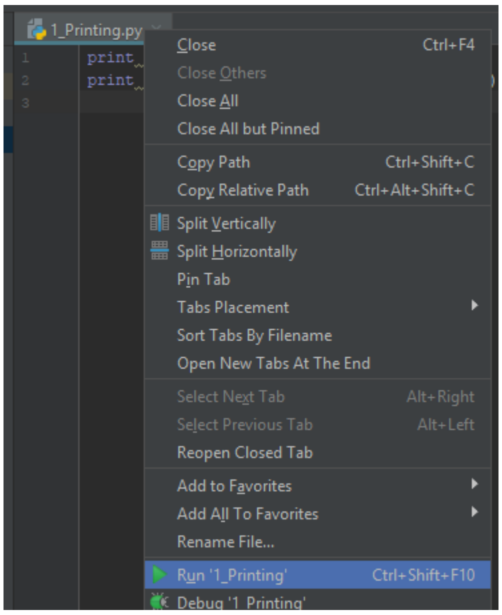
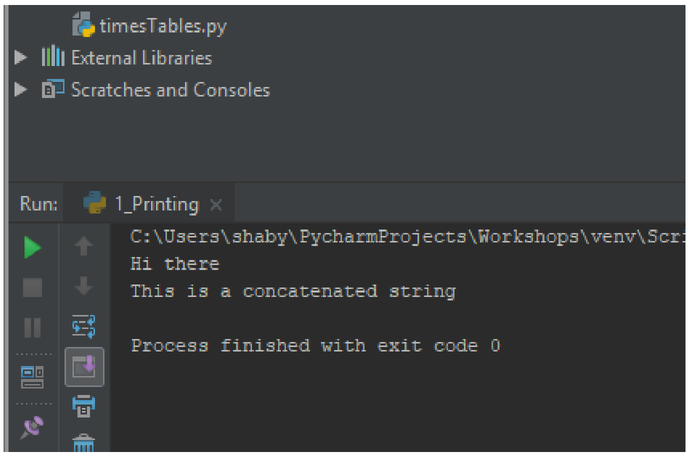


Figure 11: Techniques for getting '1_Printing' to run

You will notice that a new section appears on the IDE, which shows you the results of your code. There is no second window to deal with and get confused about.

If you make changes, then the second time you run the code you can just use the 'play' triangle on the right of the screen at the top or the left of the result window at the bottom, as shown in Figure 12 (over page).

Remember: When running for the first time, right-click the tab at the top – after that you can press the green play triangle*.*



```
C:\Users\shaby\PycharmProjects\Workshops\venv\Scr
Hi there
This is a concatenated string

Process finished with exit code 0
```

Figure 12: The green triangle is used to run a program on its second or subsequent runs

You have written your first code twice, in totally different looking environments.

You can now concentrate on the actual coding to learn about Python, rather than the environment that you are going to use.

Notes

- In **Python 3,** print statements are followed by **( )** with either **‘** or **“** surrounding the words or string. If you start with **‘** you need to end with **’**.
- Concatenating means to attach two strings together. You use the **+** to concatenate a couple of strings. Surround each string with either **‘ ’** or **“ ”** but not a mixture of them.

Program 2: Variables

Variables are placeholders that store values using a name you can remember.

Copy the code below Figure 13 into a new file. If you cannot remember how to create a new file, refer to Figure 13.

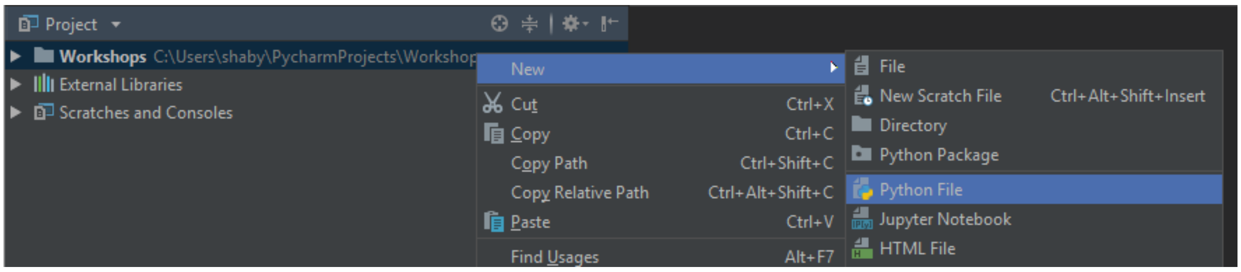


Figure 13: Python processes to create a new file

Type the following code.

```
todays_date = "13 June"
print (todays_date + " is the date today")
```

Then run the program.

Program 3: A bit about mathematics

This demonstrates mixing strings with numbers.

Type the following code.

```
product = 7 * 6
print (product)
print ("... is the answer to the meaning of life")
```

Type the following code. It’s okay in the same program file.

```
product = 7 * 6
print (str (product) + "... is the answer to the meaning of life")
```

An error has occurred because you are trying to concatenate two different data types and this doesn’t work.

The following code fixes the error. It’s okay in the same program file.

```
product = 7 * 6
print(str(product) + "...is the answer to the meaning of life")
print(product,"...is the answer to the meaning of life")
```

Note: The comma allows the two different types of data to be handled by the same print command. You are putting the string and the number next to each other but not ‘adding’ them together.

Type the following code. It’s okay in the same program file.

```
product = 5 * 7
print (product)
remainder = 1398 % 11
print (remainder)
```

This demonstrates *modulus*. This gives you the remainder of a division sum, which is useful for differentiating between odd and even, determining leap years and factors.

To do: Try seeing if order of operations works in Python.

Program 4: Arrays or lists

Type the following code in a new file**.**

```
names = ["Kim", "Beth", "Gerry", "Kerry", "Julie", "Brendan"]
print (names[0])
print (names [2:4])
print (names [2:])
print (len(names))
age = [40, 40, 45, 25, 52, 50]
print (names[0]+ str (age [0]))
```

Note that the items in the list are inside **[ ]** square brackets. This is different from the **( )** curved parentheses that are used for things like print statements.

To do: Add a few more names to the array. Maybe research ‘append’ and see if you can add an append line to the above code.

Program 5: Conditional statements – if, elif, else

Type the following code in a new file.

```
colour = "red"

if colour == "red":
   print ("You must stop")
elif colour == "amber":
   print ("You must go slower")
elif colour is "green":
   print ("You must keep moving")
else:
   print ("You must use caution")
```

Note the use of **==** (double equal sign). In Python and many other languages, the **==** has the same meaning as you would use in mathematics, like if **x equals "red".**

A single **=** (equal sign) is used to assign a value to a *variable*. So counter **= 5** will mean that a variable called 'counter' is given the value 5.

In the above example, colour is first given the value red and then the program checks to see if the *variable* colour equals red, which it does.

Also note the use of indentation – this is important in Python. In this example, it means that if colour **== "red"** then do everything underneath that is indented. If it is not indented it won't be part of that condition. Python expects indents after *if, elif and else statements* and also when using *loops*.

To do: Change the value of colour to check if the other conditions work.

Program 6: One type of looping – for loop

Type the following code in a new file and run it.

```
#reads the first list and reports back what it finds

names = ['Kim', "Beth", "Gerry", "Kerry", "Julie", "Brendan"]
age = [40, 40, 45, 25, 52, 50]
for i in names:
    print(i)
```

Type the following code under the code above.

```
#reads both lists and reports back what it finds
for i,k in zip(names,age):
    print(i,k)
```

The *zip* command groups the two different lists together so you can read through them at the same time.

To do: Research while loops *in Python and see if you can create a simple* while loop *instead of the* for loop*.*

Program 7: Range, break, counting through a loop

Type the following code in a new file.

```
MagicNumber=42

for n in range(101):
    if n is MagicNumber:
        print('You found the magic number',MagicNumber)
        break
    else:
        print(n)
    for n in range(101):
        if n % 4 is 0 and n != 0:
            print('Divisible by four...' , n)
    for n in range(1,101):
        if n % 4 is 0:
            print('Divisible by four...' , n)
```

Note: != means the same as ‘not equal to’. Some languages use **<>** to indicate the same thing.

To do: Print out a list of leap years between 1900 and today. Note that 1900 wasn’t a leap year.

Program 8: Functions including variables and coder comments

Type the following code in a new file.

```
#define a function – this one will convert bitcoins to Aussie dollars
def bitcoin_to_au(btc):
    amount = btc * 8966.85
    print(amount)

#now you call or use the function a few times:
bitcoin_to_au(1)
bitcoin_to_au(100)
bitcoin_to_au(3.5)
```

Type the following code below the above code.

```
#another example using 2 variables
def square_area(l,b):
    area = l * b
    print(area)

square_area(3,4)
square_area(5,7)
```

Type the following code below the above code.

```
#a final example using unlimited variables

def perimeter(*args):
    total = 0
    for i in args:
        total += i
    print('the perimeter is', total)

perimeter(2,3,4,5,6,7,8,19)
perimeter(43,54,23)
```

Note: (*args) means for all the arguments supplied. Each number is called an argument for some reason. It will work with 1 or 1,000 arguments, which is useful for example when you don't know how many sides there are in a shape when you want to calculate its perimeter.

To do: Try changing bitcoins to another currency or work between two of your favourite currencies.

Program 9: Returning values

Type the following code in a new file.

```
def non_creepy_dating_age(my_age):
    partner_age = my_age/2 + 7
    return partner_age

shaby_limit = non_creepy_dating_age(53)
shabyMum_limit = non_creepy_dating_age(83)
print('Shaby can date people',shaby_limit," or older")
print('Shaby\'s mum can date people ', shabyMum_limit, 'or older' )
```

Notes:

- The backward slash in shaby\'s to ignore that particular **'** single inverted comma.
- The function or procedure names can be very long.
- The function **non_creepy_dating_age()** expects a value to be included. That value is stored in a *variable* called **my_age**.

To do: Try removing the \ and see what happens.

Program 10: Global v local variables – variable scope

Type the following code in a new file.

```
pi = 3.14159

def circleArea(radius):
    pi = 50000
    area = pi * radius * radius
    print(area)

def circleCirc(radius):
    circ = pi * radius * 2
    print(circ)

circleArea(7)
circleCirc(7)
```

Note: The first **pi** is outside of the two functions being defined – it is a *variable* known by all the program so is called global. The **pi** inside **def circleArea(radius)** is a new *variable* also called **pi** (not very good programming technique), which has a different value. This *variable* is only known inside the **circleArea**. A really big number has been chosen just to make obvious the differentiation between which pi is being used.

To do: Add to this program to work out the volume of a sphere as well.

Program 11: User input

Type the following code in a new file.

```
name = input('Type your name and press enter: ')
print('Hello there, ' + name + '. Pleased to meet you.')
```

Note: A full stop has been added at the start of the second string followed by a space to make the output in line with English writing convention.

To do: Add to this program to ask for your age as well or where you were born and get it to respond to that information.

Program 12: Tables

Type the following code in a new file*.*

```
def tables(number,repetitions):
   for i in range(1, repetitions):
      print(number, " times ", i, 'is equal to ', number * i)


tables(3,23)
```

To do: Try changing the arguments when calling or running tables to two different numbers.

Note: This program is fairly restrictive because it will only allow for the tables and number of repetitions that have been hard coded into the code. Program 13 improves on this. Going through this whole process would be useful to students to see how programmers improve on their code, then run, then improve again and so on.

Program 13: Tables with user input

Type the following code in a new file.

```
def tables(number,repetitions):
    for i in range(1, repetitions+1):
        print(number, " times ", i, 'is equal to ', number * i)

number = input('Type what times tables you want to print out here:')
repetitions = input('Type how high up you want the tables to go: ')

tables(int(number), int(repetitions))
```

Program 14: Creating and importing a ‘library of code’ or module

This is like Program 13 but it imports a file of programs. There are two files here, and the second imports and refers to the first.

Note: It is important that the following two files (file 1 and file 2) are saved as *separate* programs.

Type the following code in a new file (file 1).

```
#file 1 timesTables
def tables(number,repetitions):
    for i in range(1, repetitions+1):
        print(number, " times ", i, 'is equal to ', number * i)
```

Type the following code in a new file (file 2)

```
#file 2 tablesImport
import timesTables

number = input('Type what times tables you want to print out here:')
repetitions = input('Type how high up you want the tables to go: ')

timesTables.tables(int(number), int(repetitions))
```

Program 15: Random number guessing game

Type the following code in a new file.

```
# Using random numbers for a guessing game

import random

print('This program will get you to guess a number between 1 and 10'.)
print('You will only get 6 guesses.')

SecretNumber = random.randrange(1,11)

print(SecretNumber)

for count in range(6):
    myGuess = int(input('Type your guess and hit enter: '))
    if count is 5:
        print('You didn\'t guess it in time. The secret number was ',
        SecretNumber)
    elif myGuess is SecretNumber:
        print('You guessed it!', SecretNumber, ' is the secret number. It took you
        ',count + 1, 'guesses to get the number')
        break
    elif myGuess != SecretNumber:
        print ('That\'s not it and you have', 5 - count , 'guesses to go.')
```

Note: Random is a library of code that someone else has coded for us. It has an attribute or function (something it knows how to do), which allows you to define a range called **randrange()**. You put the range inside the parentheses separated by a comma. You can use the attribute by referring to the library, then a full stop, then the attribute or function – so **random.randrange(100,1001)**. Think about what the range here will choose between.

Program 16: Improving the guessing game

Type the following code in a new file.

```
import random

print('This program will get you to guess a number between 1 and 64.')
print('You will only get 6 guesses.')

SecretNumber = random.randrange(1, 65)

print(SecretNumber)

for count in range(6):
    myGuess = int(input('Type your guess and hit enter: '))
    if myGuess is SecretNumber:
        print('You guessed it!', SecretNumber, ' is the secret number. It took you
        ',count + 1, 'guesses to get the number')
        break
    elif count is 5:
        print('You didn\'t guess it in time. The secret number was ',
        SecretNumber)

    elif myGuess > SecretNumber:
        print('Lower and you have', 5 - count, 'guesses to go.')
    else:
        print('Higher and you have ', 5 - count, 'guesses to go.')
```

SUMMARY

You have made significant progress in learning Python. Through this tutorial you have been able to:

- create loops (‘for loop’ in particular)
- create conditional statements (if elif else)
- import libraries
- create your own library of code and import it into another program
- get user input
- work with functions, such as random and modulus
- work with variables (global and local)
- work with arrays or lists.

There is lots more to know, and Google search is your friend in knowing more. You can also use sites, such as Lynda at **www.lynda.com/**, Grok Learning at **https://groklearning.com/** or Cosmo Learning at **https://cosmolearning.org/topics/python/** to work through some informative tutorials related to Python (and many other coding languages as well).