**Programming in Python**

Python is a widely-used programming language that is available on most modern operating systems and computers. On the Pi, Python is a great way to write software and games, and to control the General Purpose Input/Output (GPIO) pins. Your First Python Program “Hello, World!” is a short program that people often write as their first program in a new environment or language. It is a test program, and only displays the message “Hello, World!” to the user. In Python, you can write this program with one line of code, and it is a good way of introducing you to the integrated development environment (IDE) and how to run Python programs.

**Python Basics**

**Activity 1: Run a simple Python command**

1. Start Thonny by clicking on the Raspberry Pi icon followed by Programming > Thonny Python IDE. Write your program in the top pane, click File > Save as... to save it, and click Run > Run current script to execute the program. Output will appear in the bottom interpreter pane.

2. Type **print("Hello, World!")** and then press **Enter**. You can use apostrophes (‘) instead of quotation marks (“) if you wish.

The first thing to note is that the window is titled “Python Shell”. The Python Shell works like the Linux terminal – it runs commands that you type here when you press the Enter key.

Python programs (or “scripts”) are text files that contain all of the commands that you want to run. You can use any text editor to create these files, but there is one built-in to the Python packages on the Pi.

**Python Conditionals**

**Activity 2: Using Conditional Statements**

1. The power of computer programming is in its ability to make decisions. We call these decisions *conditional statements*.

2. Here is a python script that makes a decision. Type it exactly as you see it in the Python editor:

**age=input("How old are you?")**

**age=int(age)**

**if age < 18:**

**print("You are a minor.")**

**else:**

**print("You are an adult.")**

**input(“Press any key to continue.”)**

3. Click **File**, and then click **Save As**.

4. Save the file to your desktop with the name **age.py**.

The *if* statement introduces a conditional. Note a few important things about this conditional:

* Both the *if* and *else* lines end with a colon. This is required.
* Both the *print* statements are indented one space. They must be indented one or more spaces.
* You can have multiple statements after the *if* or *else* lines, but they must be indented.
* Only one of the *print* statements will be run. The first will display if the condition is true (age is less than 18). The second will be displayed if the condition is false (age is not less than 18).
* The *input* statement causes the program to pause so the user can see the output. Without this statement, programs run from the terminal prompt would disappear before the user could read the output.

Test your program from the terminal prompt at least four times. Once with an age that is less than 18, once with an age that is greater than 18, and once with an age that is equal to 18. Lastly, try entering a word instead of a number. You will notice that your program fails. We’ll deal with that issue soon. It’s important to test in multiple ways!

**Comments in Python**

As your Python scripts become more complicated, you can put in explanations of what parts of your code do. This is to help you in case you need to a change a program that you wrote a long time ago. It also helps if somebody else has to understand your code. Any line that begins with # is a comment and is ignored by the Python interpreter.

**Activity 3: Error Checking – Numeric Input**

1. It’s important to check that the user enters valid data. When we are asking for an age, we should make sure the input is correct. First, it must be a number. Modify your Python script so it checks for valid input. Here’s how:

**age=input("How old are you?")**

**if age.isdigit():**

**age=int(age)**

**else:**

**print(“Your input needs to be a number!”)**

**input(“Press any key to continue.”)**

**quit()**

**if age < 18:**

**print("You are a minor.")**

**else:**

**print("You are an adult.")  
input(“Press any key to continue.”)**

Note the following things about this program

* The statement age.isdigit() is true if age is a number.
* The statement quit() terminates the program because the user did not enter a number.

**Activity 4: Error Checking – Reasonable Input**

1. It’s important to check that the user enters valid data. When we are asking for an age, we should make sure the input is reasonable. First, it must be a number. Second, it should be a positive number. Third, we probably shouldn’t believe someone who says they are 512 years old!
2. Modify your Python script so it checks for valid input. Here’s how:

**age=input("How old are you?")**

**if age.isdigit():**

**age=int(age)**

**else:**

**print(“Your input needs to be a number!”)**

**input(“Press any key to continue.”)**

**quit()**

**if age > 0 and age < 130:**

**if age < 18:**

**print("You are a minor.")**

**else:**

**print("You are an adult.")  
else:  
 print(“Your age has to be between 0 and 130”)  
input(“Press any key to continue.”)**

Note the following things about this program

* The statement *if age > 0 and age < 130*: is true only if both conditions are true. Age must be both greater than 0 and less than 130.
* Note that we have an if statement inside an if statement. This is called a “nested if.” The statement *if age < 18* will be executed only if the age is > 0 and the age is < 130.

**Advanced Python concepts**

We will learn three important programming concepts:

1. functions: perform a specific task
2. loops: repeat a block of code multiple times
3. Random number:

**Activity # 5: Drawing a five-pointed star** 

Here is a sample code to create a five-pointed star. We are making use of use of functions and loop in Python. Functions are written by programmers to perform a specific task. Functions begin with the word def followed by the function name and normally receive some data (called arguments or parameters) to perform the task. Function header ends with : In this case, the function is called to draw the star.

for loops allow to repeat the code below to loop as many times as you specify in the range function.

**Indentation of code is very important. Pay attention as you practice the following code:**

import turtle

def main():

t = turtle.Turtle()

t.hideturtle()

sideLength = 200

drawFivePointStar(t, 0, 0, sideLength)

def drawFivePointStar(t, x, y, sideLength):

#Drawing begins at (x,y) and moves in a north-east direction.

t.up()

t.goto(x,y)

t.left(36)

t.down()

for i in range(5):

t.forward(sideLength)

t.left(144) # 144 = 180-36 why??

main()

Save the above code and run the program using IDLE or Raspberry Pi.

**Exercise:** Can you draw a filled rectangle, filled circle, filled star?

Hint: There are fillcolor and begin\_fill functions in Python.

When learning Python and you might become curious about how to find help when you need it. Asides from doing google searches that may or may not help you, there's a useful function called help() built into python that can assist you in understanding different aspects of Python or additional libraries.

Let's say you're using Python's Turtle module, which allows you to create cool graphics and animations. Sometimes, you might wonder what functions and methods are available in the Turtle module that you can use to draw, move the turtle, or change its appearance. You can reference perhaps a printed manual but if you do not have one to access that's where the help() function comes in handy! It's like having a friendly guide who can provide you with information about different things you can do with Turtle.

To use the help() function with Turtle, you just need to import the Turtle module at the beginning of your program. Once you've imported Turtle, you can call help(turtle) to get a list of available methods and functions in the Turtle module. When you run the code, the help documentation will appear, showing you a list of methods and their descriptions. You can explore this list to see what each method does and how you can use it to create fantastic drawings and animations with Turtle.

Try the following in the shell:

>>> from turtle import Turtle

>>> t = Turtle()

>>> help(t)

After running that last command, you will get a message “Squeezed text(1150 lines).” Double click this box to expand out the text. This will show you all the information on the entire Turtle Class. This can be helpful to explore and look for methods in a new library or class you are wanting to work with.

Sometimes however, we do not need the entire list of available commands to a library and instead just want some information about a specific method or property. If that is the case and you know the method or property, you can ask for just help on that specific item.

For instance if I wanted to learn more about how to make Turtle write specific words at its current location in a program I could do what I did above but for the help command, I would instead type the following.

>>> help(t.write)

I would then get a list of the parameters the method could handle and what each of their purposes are.

Remember, if you ever need assistance or want to learn more about a specific method, the help() function is always there to guide you. It's like having a knowledgeable friend right by your side.

**Random number generator**

In Python, just like in almost any other OOP language, chances are that you'll find yourself needing to generate a random number at some point. Check out the code snippet below to see how it works to generate a number between 1 and 100.

import random

for x in range(10):

print random.randint(1,101)

The code above will print 10 random values of numbers between 1 and 100. The second line, for x in range(10), determines how many values will be printed (when you use range(x), the number that you use in place of x will be the amount of values that you'll have printed. if you want 20 values, use range(20). use range(5) if you only want 5 values returned, etc.). Then the third line: print random.randint(1,101) will automatically select a random integer between 1 and 100 for you. The process is fairly simple.

What if, however, you wanted to select a random integer that was between 1 and 100 but also a multiple of five? This is a little more complicated. The process is the same, but you'll need to use a little more arithmetic to make sure that the random integer is in fact a multiple of five. Check out the code below:

import random

for x in range(10):

print (random.randint(1,21)\*5)

This code generates a random number between 1 and 20, and then multiply that number by 5.

**What is the highest number generated here?**

****

**Activity #6: Generate random numbers**

Play around with the above code yourself and see if you can generate a random number between 1-100 where every number generated is a multiple of ten!

**Lab Exercise #1**

Write a Python program that asks a user for two numbers. The first is the number of students in the class, and the second is the cost of a drone. The program should then tell the user how much it costs to buy a drone for every student in the class. Write your program in phases:

1. First, ask for two numbers, convert them to integers, and calculate the cost. Display the cost for the user. Don’t worry about error checking.
2. Next modify your program to check for numeric input. Be sure both values entered by the user are numbers. If they aren’t the program displays a message and stops. If they are, the program calculates the cost. Test your program with valid and invalid input.
3. Next, check for reasonable input. Both numbers must be greater than zero. If they aren’t, display an error message. If they are, calculate the cost.

**GenCyber Principles that programmers follow the most: Keep It Simple.**

**Keep computer programming and system design simple without compromising the ability to protect information from unauthorized access.**

**Notes about programming style:**

1. You should include comments in your code to explain what it does. It’s also a good idea to include your name.
2. When asking for input, display a helpful and meaningful message.
3. Use meaningful variable names! If you are asking for someone’s age, use a variable named age or something similar.
4. Make sure your output looks good and is understandable to the user. Spell your words correctly and include blank spaces and punctuation where they belong.

**Check your project for completion:**

1. Does your program ask for both required pieces of information (number of students and price) and store each in a variable?
2. Does your program correctly calculate the total cost?
3. Does your program clearly display the total cost?
4. Does your program use meaningful variable names?
5. Does your program have helpful and descriptive comments?
6. Run your program two different ways: from the Python IDE and from a Linux terminal.
7. Show your program to your team leader to check it out!

**Learning More**

Want to learn more about Python? There are lots of free resources out there. Python is the most popular language for colleges in introductory programming courses. A great site for learning more about Python is:

<https://www.python.org/>

Here you can download Python, learn all about the language, and discuss Python with other programmers.

**Python Extra Challenges**

1. Write a joke telling program such as knock knock joke or any joke.
2. Ask user to guess a randomly selected number. Allow hints such as too high or too low or getting warm or cold. Count how many guesses it took for the user to guess.
3. Create a Simple Calculator by writing your own functions.
4. Write a program to display monthly or yearly calendar.
5. Challenge: Write your own hangman game.