**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. Click the Windows button, search for **Python IDLE**, and then click **Python 3 (IDLE)**.

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?") #file age.py**

**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?") #file age1.py**

**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?") #file age2.py**

**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.

**First Principles**

*Encapsulation* – if you want to print in Python, you must use the **print** function. This function controls how you can display information, and you must use it properly to send output to the screen. You don’t have direct access to the computer’s display. The print command controls how you can produce screen output.

*Abstraction* – when you create a **variable** in Python, the value you store in the variable is stored in memory. You only need to give the variable a name and a value. The details are hidden from you. You don’t see how many bytes of memory the variable requires, and you don’t need to know the memory addresses that are used. Those are details that you really don’t need to know.

*Information Hiding* – when you create a **string** in Python, it is multiple characters. Each character requires one byte of memory. How those bytes are arranged is hidden from you. All you do is specify the characters in the string, and Python takes care of the details about how to make sure the characters are stored in the right order.

**GenCyber Cybersecurity Concepts:**

* *Defense in Depth*: allow multiple layers or levels of security in your programs so if one layer fails, another layer of security is already in place to stop the attack. Ex: company data is secured by a firewall, passwords and encryption.
* *Confidentiality*: information is not disclosed to unauthorized users. Ex: medical information of a patient at a hospital, bank account information.
* *Integrity*: system has not been modified or destroyed in an unauthorized manner. Ex: computer system is virus-free and uncompromised.
* *Availability*: information is accessible and usable upon demand. Ex: Student grades can be viewed by student and principal but modified by the teacher.
* *Think Like an Adversary*: Always be prepared to think ahead and be defensive. Ex: virus and malware attacks and prevention
* *Keep It Simple*: Try to keep your programs simple and easy to follow. Ex: Complex coding is not easy to understand and can have too many bugs.

**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 #file testStar.py**

**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.

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

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

**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(1,100)**

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(1,100) 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(1,20)\*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 displays the GenCyber Concepts and its definition randomly or ask the user to choose based on a menu.

**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. Create a Simple Calculator by writing your own functions.
3. Write a program to display monthly or yearly calendar.
4. Challenge: Write your **own hangman game**.