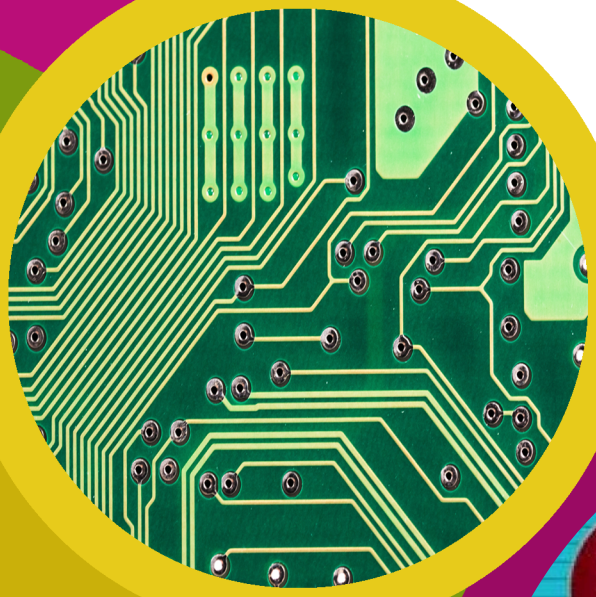
technocamps

Using Python across the CfW



UNDEB EWROPEAIDD EUROPEAN UNION
Llywodraeth Cymru Welsh Government
Cronfa Gymdeithasol Ewrop European Social Fund

Prifysgol Abertawe Swansea University
CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD
PRIFYSGOL BANGOR UNIVERSITY
Cardiff Metropolitan University | Prifysgol Metropolitan Caerdydd
University of South Wales Prifysgol De Cymru
i.t.wales
PRIFYSGOL ABERYSTWYTH UNIVERSITY
PRIFYSGOL glyndŵr Wrecsam | Wrexham glyndŵr UNIVERSITY

Overview

Coding can be implemented across all the Areas of Learning and Experience, reinforcing learning in the classroom and improving digital literacy in the process.

In today's world digital literacy is an essential skill for learners to develop. The technological requirements for jobs are ever increasing, and a strong start in digital skills will prepare learners and give them an advantage.

Online Resources

Digital Resources:

https://tc1.me/educonf22resources

Youtube Tutorials:

https://tc1.me/progacrosscurriculum

More ideas to Program

Health and Wellbeing
- Food Pyramid
- Pong

Mathematics and Numeracy
- Drawing Shapes
- Estimating Pi

Science and Technology
- States of Matter
- Water Cycle

Languages, Literacy and Communication
- Translating Quiz
- Pronouns Quiz

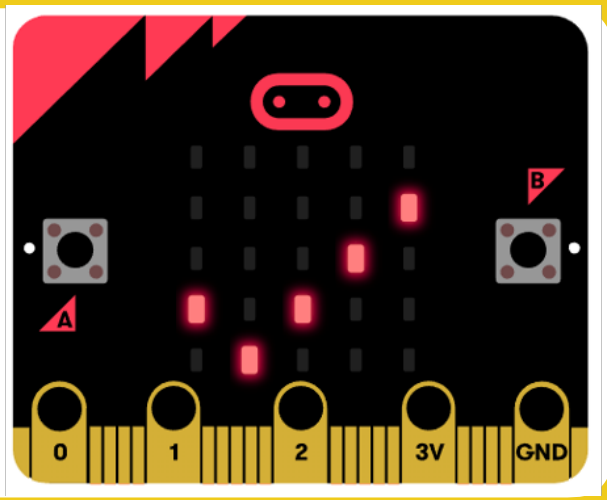
Expressive Arts
- Algorithmic Art
- Matching Art Styles

Humanities
- Interactive Timeline
- Migration Simulation

Micro:bit Pedometer

This is example code on how you can use a BBC Micro:bit to make a pedometer.

The following pages will provide you with the code required to get a BBC Micro:bit to count the steps you take.



Importing Libraries

Firstly go to the following website:
https://python.microbit.org/v/3
This allows you to interact with with a simulated BBC Micro:bit. Delete all the default code you see.

To use the BBC Micro:bit to count our steps for us, we will need to import one library. We will be using a variety of functions from the library, so we can go ahead an import the whole library.

```
from microbit import *
```

This will import all the libraries needed for our Python code to interact with the BBC Micro:bit.

Creating the Pedometer

Firstly we import our Micro:bit library, and import all methods from it (as shown by the *).

This code then declares and sets a variable called steps to 0. This will keep track of how many steps we take.

We now start a loop, with a condition which will always be true. This is done to ensure our code continues to run forever.

Within this loop we add an if statement. This is a conditional, that when true, the code which is placed after it is exectued. In this case, we are checking if the accelerometer has detected the device has been shaken.

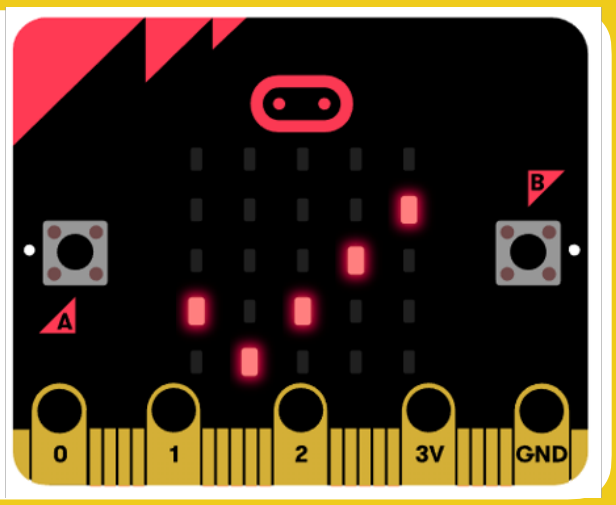
If the device has been shaken, we add one to our steps variable, and we then display the current amount of steps.

```
from microbit import *

steps = 0

while True:
    if accelerometer.was_gesture('shake'):
        steps += 1
        display.show(steps)
```

Micro:bit Music Maker

This is example code on how you can use a BBC Micro:bit to compose and create music.

The following pages will provide you with the code required to get a BBC Micro:bit to play the song Aderyn Melyn.



Importing Libraries

Firstly go to the following website:
https://python.microbit.org/v/3
This allows you to interact with with a simulated BBC Micro:bit. Delete all the default code you see.

To use the BBC Micro:bit to create music for us, we will need to import one library and a function. We will be using a variety of functions from the library, so we can go ahead and import the whole library.

```
from microbit import *
import music
```

Firstly, we are importing the microbit library. This allows us to use Python to control and interact with our BBC Micro:bit.
Secondly, we are importing the music library. We can use the "play" function and provide it with the pitch & duration of each note.

Creating Notes

We create notes using python using the music library we imported earlier. The play function requires a maximum of two pieces of information: the note, and the duration. Its not essential to give the duration, but we must always provide a note.

The below piece of code plays a C4 note (Middle C), for 4 bars (the standard length for a note with the play function).

```
music.play(['C4:4'])
```

We are not limited to just one note per line, we can string multiple notes together, to be played one after another.

```
music.play(['C4:4', 'B4', 'D3:3'])
```

We can also add rests into our code, which will give us a rest in-between notes. Notice the rest in-between notes B4 & D3.

```
music.play(['C4:4', 'B4', 'R', 'D3:3'])
```

Finally, we can introduce loops, to repeat our notes multiple times. The following code repeats all 3 notes in sequence 2 times, resulting in 6 notes in total: C4, B4, D3, C4, B4, D3.

```
for i in range(2):
    music.play(['C4', 'B4', 'D3'])
```

Aderyn Melyn

Here is an example program for playing the song "Yellow Bird, Up High in Banana Tree" or "Aderyn Melyn".

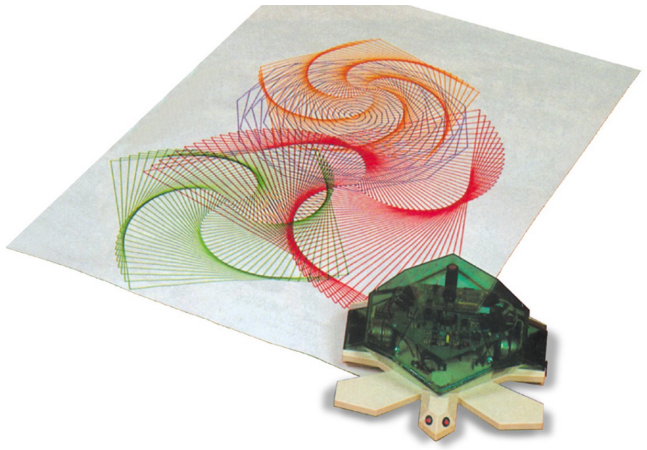
```
from microbit import *
import music


for i in range(2):

      music.play(['C4:4', 'C5:12', 'B4:4', 'C5:2',
'C5:2','R', 'A4:4', 'A#4:2', 'A#4', 'A#4', 'A#4',
'A#4', 'A#4', 'C5', 'A4:4', 'A4', 'R:2'])

for i in range(2):

      music.play(['D4:4', 'D4:2', 'G4:3', 'A#4:2',
'D5', 'R','C4:4', 'C4:2', 'E4:3', 'F4:2', 'G4',
'R'])

music.play(['C4', 'C4:3', 'A#4:4', 'A4:2',
'G4:3','R:1', 'F4:5'])
```

Note: Some of the lines are too long to fit in one in this workbook, you should not put new lines before finishing the play function. This is true for the code that follows as well.

Turtle Art

This is example code for creating shapes and art via python using the Turtle Library

The following pages will show two programs: how to draw a square, and a how to make a spirograph.



Importing Libraries and Accessing the Turtle

To start making a turtle program, access this website (it is possible to do this in IDLE if you have installed Python on your device):

https://www.pythonsandbox.com/turtle

Once on the website, delete any code in the terminal on the left of the screen, we will be making ours from scratch!

To use the Turtle to make shapes and art for us, we firstly need to import the turtle library.

```
import turtle
```

Drawing a Spirograph

Firstly we import our library and create variables. The library import allows us to access the turtle functions, while the variables allow us to edit the type of turtle we will be using. We will call our turtle 'pen' since we are drawing, and will make it the shape of a turtle, and make it draw in green.

```
import turtle
pen = turtle.Turtle()
pen.shape("turtle")
pen.color("green")
```

Now we can give explicit commands to our turtle. Commands such as forward make the turtle move in a straight line, to a set number of 'units'. We can also make our turtle rotate either left or right, using degrees.

```
pen.forward(100)
pen.right(90)
pen.forward(100)
pen.right(90)
pen.forward(100)
pen.right(90)
pen.forward(100)
pen.right(90)
```

The result will be a turtle which will move in a square for us.

Tip: Think of the turtle as having 'tank controls' it can only go forwards or backwards, and must pivot on the spot to change direction.

Drawing a Spirograph

Firstly we import our library and variables, as we did before.

```
import turtle
pen = turtle.Turtle()
pen.shape("turtle")
pen.color("green")
```

Now unlike last time, we will be making a loop to repeat our code multiple times. We have specified we want our code to repeat 350 times. The variable i will be used to count up to 350, starting from 1, and increasing my 1 every interation.

This means initially our turtle will only move 1 unit, but it will slowly get bigger and bigger, moving more and more as the code continues.

```
for i in range(350):
    pen.forward(i)
    pen.right(98)
```

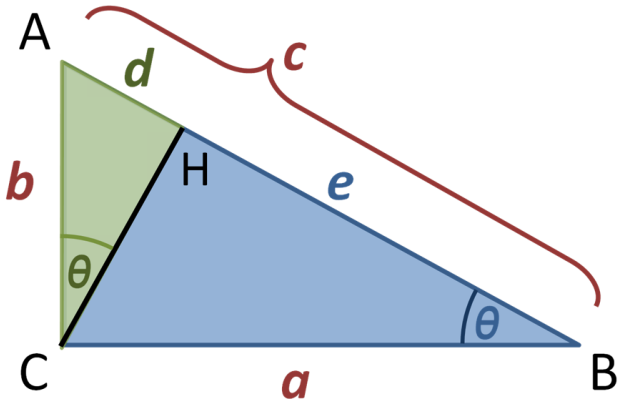
The result will be a turtle which will make a spirograph for us.

Why not try tinkering with some of the numbers? What happens if you change the angle?

Pythagoras

This is an example code for a Pythagorean theorem calculator in Python.

Each page will provide you with the next step in the program.



Importing Libraries

To use the Pythagorean theorem to calculate the lengths and angles of a right triangle we will need to import some functions from the python **math** library.

```
from math import sqrt, asin, acos, atan

print('Pythagorean theorem calculator! Calculate your triangle sides and angles')
print('Assume the sides are a, o and h, where h is the hypotenuse')

formula = input('Which side (a, o, h) do you wish to calculate?')
```

The program begins by printing an explanation of its function, and asking the user to input which side it should calculate.

Calculating Side A

To calculate the length of '**a**' (the side adjacent to the angle θ) an **if** statement is used to check the chosen user input.

The program then asks the user for the length of sides **o** and **h**.

Then the length is calculated using the rearranged Pythagorean theorem.

Using **$\sin^{-1}$** the angle theta is calculated.

Both values are printed along with a string describing the value (**Note**: **\n** is used to insert a new line in Python).

```
if formula == 'a':
      sideO = int(input('Input the length of side o: '))
      sideH = int(input('Input the length of side h: '))

      sideA = sqrt((sideH * sideH) - (sideO * sideO))

      theta = asin(sideO / sideH)

      print('\nThe length of side a is : ' + str(sideA))
      print('\nThe angle given by sin in radians is: ' + str(theta))
```

Calculating Side B

To calculate the length of '**o**' (the side opposite to the angle θ) an **elif** statement is used to add a condition to the if statement. The program then asks the user for the length of sides **a** and **h**.

Then the length is calculated using the rearranged Pythagorean theorem.

Using **cos^{-1}** the angle theta is calculated.

Both values are printed along with a string describing the value.

```
elif formula == 'o':
     sideA = int(input('Input the length of side a: '))
     sideH = int(input('Input the length of side h: '))

     sideO = sqrt((sideH * sideH) - (sideA * sideA))

     theta = acos(sideA / sideH)

     print('\nThe length of side a is : ' + str(sideO))
     print('\nThe angle given by sin in radians is: ' + str(theta))
```

Calculating Side C

To calculate the length of '**h**' (the hypotenuse) an **elif** statement is used to add another condition to the if statement. The program then asks the user for the length of sides **a** and **o**.

Then the length is calculated using the Pythagorean theorem. Using **$\tan^{-1}$** the angle theta is calculated. Both values are printed along with a string describing the value.

An **else** statement is included to ensure that a valid input has been given.

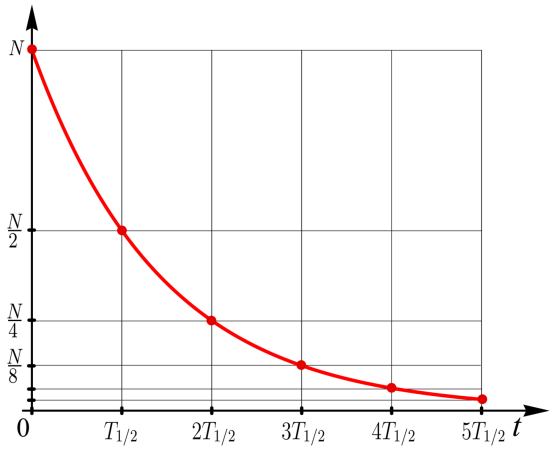
```
elif formula == 'h':
     sideA = int(input('Input the length of side a: '))
     sideO = int(input('Input the length of side o: '))

     sideH = sqrt((sideA * sideA) + (sideO * sideO))

     theta = atan(sideO / sideA)

     print('\nThe length of side a is : ' + str(sideH))
     print('\nThe angle given by sin in radians is: ' + str(theta))

else:
     print('Please select a side from a, o or h')
```

Random Exponential Decay

This is an example code for an Exponential Decay Simulator in Python.

The program will run a random simulation and output the results, these can then be plotted in Excel.

Each page will provide you with the next step in the program.



Importing Libraries

To simulate the random decay of radioactive nuclei we will need to import some functions from the python math and science libraries.

```
from math import log
from random import randint
```

Libraries can either be fully imported where each function called will also require the library to be called (using import _),
or specific functions may be imported from a library, which can then be called directly (using from _ import _).

Initialising

To initialise the program the user is asked to input the number of nuclei at the beginning of the simulation and the probability of a nucleus decaying each second.

These values are saved as an integer and a float respectively, so that the number of nuclei must be inputted as a full number where the probability can take any value.

Some initial values are defined so that the simulation begins at 0 seconds with the half life unknown (the reason this is required will become clear).

Note: The red lines are comments within the code. Comments in Python can be added with a #.

```
# Initial Number of nuclei
initialNuclei = int(input('What is the starting
number of nuclei?'))
currentNuclei = initialNuclei

# The probability that a given nucleus will decay
probability = float(input('\nEnter a percentage value
for the chance of a nucleus decaying each second i.e.
50: '))

# Setting initial values for variables
seconds = 0
halflife = 0
halflifeFound = False
```

Main Program

The main body of the program consists of a while loop which will run until all the nuclei have decayed.

Each time the loop runs the number of nuclei remaining is printed and the number of seconds passed increases.

The for loop runs through each nucleus remaining and generates a random float number between 0 and 100.

If the random number is smaller than the probability of decay, then the nucleus decays and the number of nuclei is decreased by 1.

```
while currentNuclei > 0:

# Print the number of nuclei remaining and
increase the time by 1

    print(currentNuclei)
    seconds += 1

# Consider each nucleus in turn and decide whether
it decays or not

    for nucleus in range(currentNuclei):
        rand = randint(0,100)
        if rand < probability:
            currentNuclei -= 1
```

Half Life Estimate

As an extension to the main program the half life can be estimated. This is done within the while loop.

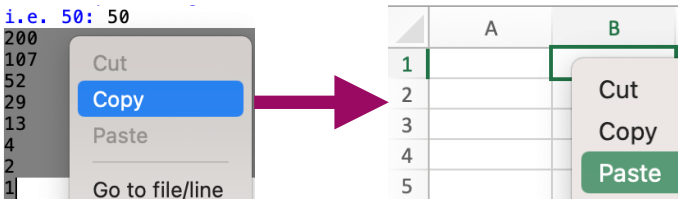
If the number of nuclei remaining is less than half the initial number then an estimate value of the halflife is taken. So only one value is taken, an and clause is included to check a Boolean value which is then changed.

The estimate for half life is printed along with the second before to give a range for when the halflife occurred. This is written outside of the while loop.

```
# Store the value of the halflife when more than
half of the nuclei have decayed

    if currentNuclei < round(initialNuclei / 2)
and halflifeFound == False:
        halflife = seconds
        halflifeFound = True

print('The half life for this sample is between %d
and %d seconds.' %(halflife - 1, halflife))
```

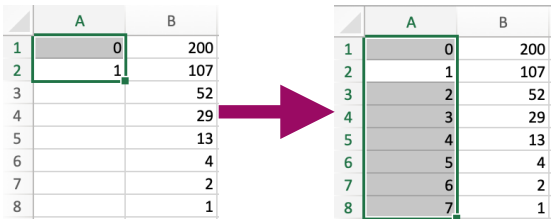
Plotting Data

The values outputted in the Python Shell can be copied into Excel in one by highlighting them all, copying and pasting in Excel.



Paste the values in column B, so that it plots as the y axis. Further results can be added in the following columns, to plot multiple simulations.

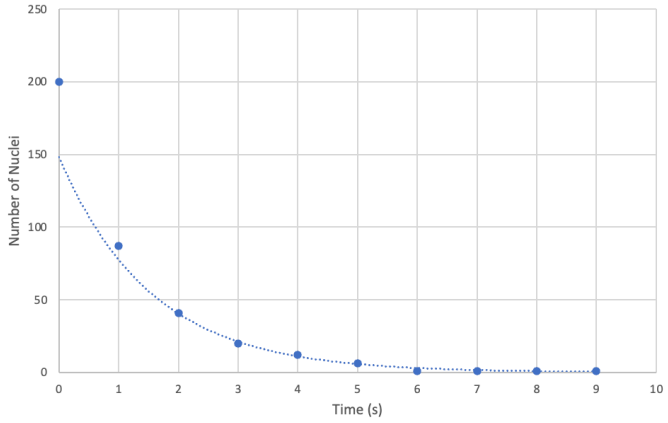
In column A add the seconds passed from 0. This can be done by entering 0 and 1, selecting both, then dragging the bottom corner.



	A	B
1	0	200
2	1	107
3		52
4		29
5		13
6		4
7		2
8		1

	A	B
1	0	200
2	1	107
3	2	52
4	3	29
5	4	13
6	5	4
7	6	2
8	7	1

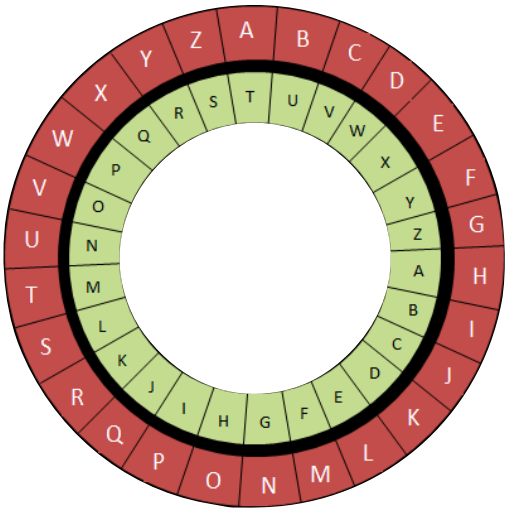
Select all the data and plot through Insert > Charts > X Y (Scatter).
Add an Exponential Trendline from Add Chart Element to fit the data.
A trendline could be fitted to the average result of many simulations.



Caesar Cipher

This is an example code for a Caesar Cipher program in Python.

Each page will provide you with a possible function to implement, increasing the complexity of the program.



Defining Variables

To build a working Caesar Cipher we will need to define the set of symbols (characters) we will be using and shifting to encrypt our text.

```
global alpha,bet
alpha = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
bet = 'abcdefghijklmnopqrstuvwxyz'
```

In this example two variables were created for upper and lower case letters. However we could include another for numbers, another for punctuation marks, or alternatively include them all in one large cipher variable!

The variables have been made **global** so that they can be used in any of the following functions, avoiding having to pass them to each one.

Encrypt Function

This function will encrypt any plaintext fed into it using key specified and the global variables.

```
def encrypt(text,key):
    result = ''

    for letter in text:

        if letter.isupper():
            num = alpha.find(letter)
            num += key

            if num > 25:
                num -= len(alpha)
            result += alpha[num]

        elif letter.islower():
            num = bet.find(letter)
            num += key

            if num > 25:
                num -= len(bet)
            result += bet[num]

        else:
            result += letter

    return result
```

Note: There is no call or print command in this function - to call and print the function we could use something like print(encrypt('My Plain Text' , 5))

Decrypt Function

This function will decrypt any encrypted text fed into it using key specified and the global variables.

```
def decrypt(text,key):
    result = ''

    for letter in text:

        if letter.isupper():
            num = alpha.find(letter)
            num -= key

            if num < 0:
                num += len(alpha)
            result += alpha[num]

        elif letter.islower():
            num = bet.find(letter)
            num -= key

            if num < 0:
                num += len(bet)
            result += bet[num]

        else:
            result += letter

    return result
```

Note: There is no call or print command in this function - to call and print we could use something like print(decrypt('Rd Uqfns Yjcy' , 5))

Hack Function

This function will hack any encrypted text fed into it using a brute force method to run through each symbol in the global variables.

This function contains one extra for loop to try every key possible and then print out the results.

```
def hack(text):
    for key in range(len(alpha)):
        result = ''

        for letter in text:
            if letter in alpha:
                num = alpha.find(letter)
                num -= key
                if num < 0:
                    num += len(alpha)
                result += alpha[num]

            elif letter in bet:
                num = bet.find(letter)
                num -= key
                if num < 0:
                    num += len(bet)
                result += bet[num]

            else:
                result += letter

        print('Hacking key #%s: %s' % (key, result))
```

Main Menu Function

This function is an extension task to create a unified menu for the program with all three functions accessible

.

The function first asks for an input (1, 2 or 3) to choose a function, then asks for the information needed to run each function.

Note: The **encrypt** and **decrypt** functions are run from within the print command.

```
def main():
    print('Welcome to the Caesar Cipher!\n')
    program = input('What would you like to do? \n1. Encrypt \n2. Decrypt \n3. Hack \n')

    if program == '1':
        text = input('Enter the text to encrypt: ')
        key = int(input('Enter the encryption key: '))
        print('\nPlain Text: ' + text + 'Encryption key: ' + str(key) + 'Encrypted Text: ' + encrypt(text,key))

    elif program == '2':
        text = input('Enter the encrypted text to decrypt: ')
        key = int(input('Enter the encryption key: '))
        print('\nEncrypted Text: ' + text + 'Encryption key: ' + str(key) + 'Plain Text: ' + decrypt(text,key))
```

Main Menu Function

This is the remainder of the main menu function.

As the **hack** function already prints out each iteration as it runs, there is no need to include it within a print command.
An **else** statement is included to ensure a valid input is given, then **main()** is called so that the main menu loops indefinitely!

Note: main() is called once more at the end, outside of the function, to begin the program and open the menu!

```
    elif program == '3':
        text = input('Enter the encrypted text to crack: ')
        print(' \n' )
        hack(text)

    else:
        print('MUST ENTER A VALID OPTION BETWEEN 1 AND 3')

    main()

main()
```

technocamps

@Technocamps

Find us on Facebook