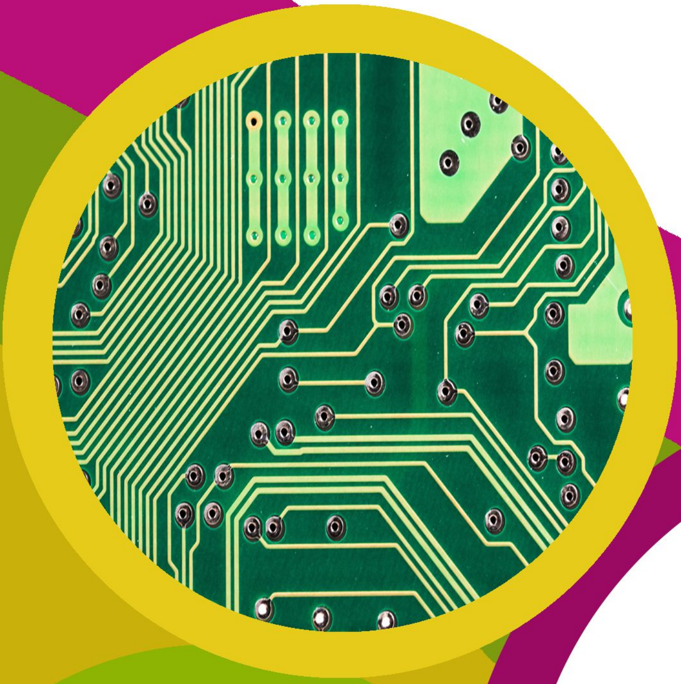
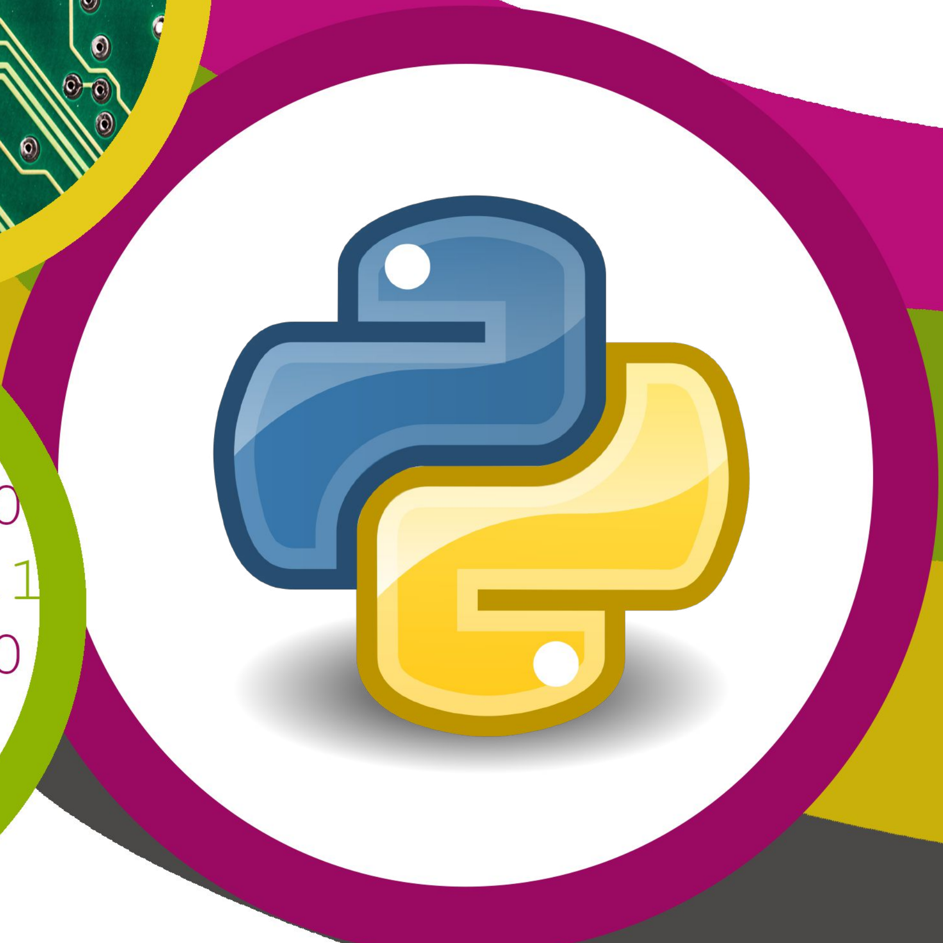
technocamps

Algorithms II Session Plan





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
i.t.wales
Prifysgol Aberystwyth University
Prifysgol Glyndŵr Wrecsam | Wrexham Glyndŵr University
University of South Wales Prifysgol De Cymru

Workshop Schedule

Introduction - 10 minutes

Algorithms Session 1 Recap - 1 hour

Flowcharts, Selection - 20 minutes

Subroutines in Python - 1 hour 10 minutes

Radioactive Decay - 35 minutes

Lists in Python - 25 minutes

Sort and Search Algorithms - 45 minutes

Conclusion - 10 minutes

Post-Day Questionnaires - 10 minutes

Note: These are estimated times, these will vary between classes, schools etc. so times will need to be adjusted accordingly.

Total: 4 hours 45 minutes

Preparation

1. Ensure all computers have Python 3 installed and ready to be used.
2. Print out Algorithm II workbooks and cheatsheet, one for each student attending the workshop.
3. Each student will need a pencil for drawing the flowcharts.

Learning Outcomes

1. Improved knowledge of subroutines using functions in Python.
2. Greater experience of designing, writing, and implementing algorithms to solve real world mathematical problems.
3. Greater experience in applying algorithms to other STEM subjects.

Attendee Prerequisites

1. Intermediate experience in Python programming.

Session Plan Key

In this session plan we use the following colours to differentiate the types of activities:

- **Yellow - Explain.** Teachers should explain the slide/example to the class.
- **Green - Discuss.** Teachers should start an open discussion with the class to get them to feedback some answers/ideas.
- **Purple - Activity.** Students are expected to complete an activity whether it be in their workbooks or on the computer, followed by a discussion of their solutions.
- **Green - Introduction/Conclusion.** The introduction/conclusion is also colour coded green. Teachers should hand out materials in the introduction and conclude the day and collect materials at the end.

Introduction

Begin with introductions, and a brief explanation of the Technocamps programme, before handing out pre-day questionnaires to be filled out by the students and teacher.

Activity: Recap Algorithms

What is an Algorithm? – Students should write their definition of an algorithm in their workbooks.

Discuss: Algorithms

Ask for answers from students to see if they remember the definition of algorithms:

- Can anyone remember the definition of algorithm?
- What is important to remember when writing an algorithm?
- Where do we use algorithms in everyday life?

Talk about how today is about applying those skills learned in Algorithms I in programming.

Discuss: Recap Decomposition

Ask if anyone can remember the definition for decomposition?
Remind them of the activity they did about the decomposition of creating a video game:

- What needs to be done first?
- What is the objective of the game?
- Characters, Worlds, what can the Characters do?
- Is it a single-player or multiplayer game?
- How do the Characters interact with the World/other Characters?

Explain: Recap Decomposition

Remember the outcome of the Lego activity in Algorithms I, which showed how important it is to give clear, simple instructions with enough details, especially to a computer which cannot ask questions. It highlights the importance of decomposition when faced with a task which may seem complex at first, but can be achieved by breaking the problem down into smaller steps.

Discuss: Recap Abstraction

Ask the students if they remember the definition of abstraction:

- Can anyone remember the definition for abstraction?

Explain: Recap Abstraction

Remind the students about abstraction:

- Why is abstraction useful? To whom?
- Remind them again that the concept of abstraction is everywhere; teachers use abstraction in how much detail they teach at different academic levels. Remind them of the difference in detail in an animal cell and the inside of an atom at GCSE and A Level.
- In the concept of a car – someone who builds a car needs to understand how each and every part works. The driver only needs to know how to use the steering wheel, gearstick, brakes, clutch and accelerator. They do not need to know the inner workings of the engine.

Discuss: Recap Subroutines

Ask the students if they remember the concept of subroutines:

- Can anyone remember the concept of subroutines?

Explain: Recap Subroutines

Explain subroutines using the simple flow chart.
Highlight the fact that the subroutine is the section on the right which seems separated from the main body of the flow chart. Also mention that when the 'addTwoToAge' is executed it activates the subroutine on the right and when it finishes it goes back to where the 'addTwoToAge' is called and carries on with the rest of the flowchart.

Explain: Functions in Python

Explain that subroutines are implemented as functions in Python. Explain that a function is a subroutine that usually takes in one or more values (sometimes no input at all) from the program and returns a value back. Talk about the advantages of functions, i.e. reduces the redundancy of code and duplications. Also explain how the functions can be accessed any number of times as required by the program. A function can be called by other functions and can be executed whenever needed. It saves time and effort for the programmer.

Explain: Functions in Python - Example

Show the students an example of a complete function in Python (the one on the slides). If time allows show the students the program running. Change the values provided as an input when testing and step by step illustration of what is happening. The step by step illustration of the flow of the program is also provided in the following slide.

Explain: Functions in Python - main()

Once the cubeVolume function is explained, explain how the program can be extended to include steps that call the cubeVolume function from another function called main() and how the Python interpreter will use main() as the entry point for the program.

Explain: Functions Implementation

Show them the maths question which asks students to work out the hypotenuse of a right-angled triangle. We have three points, A, B and C. We know the distance from A to B and B to C. We are asked to calculate the distance from A to C.
Let us create a function that can calculate the hypotenuse for any right-angled triangle.

The students should think of an equation that will solve this question.

Activity: Calculating the Hypotenuse

Students should write an equation in their workbooks to calculate the hypotenuse. They can use the equation to solve the above problem so they can check if the program they write later, works correctly. Check if the equation is $h = \sqrt{(a^2 + b^2)}$

Explain: Functions Implementation

Explain the main() function. Inside this function is where our normal code goes and also where we call other functions.

Explain how the main function is the first function in our code. It is the main body of the program and that all other functions are subroutines.

Explain that when we create our program we always start by creating the main function first using the reserved word **def**. As shown in the snippet below.

```
#A program that calculates the hypotenuse.

def main():

#Main entry to the program

main()
```

Explain: Define Hypotenuse Function

Now that they have created their main function they can start creating the function that will calculate the hypotenuse.
First, explain that the function has three parts:

1. **Name:** Name of the function.
   Explain the importance of naming the function by its purpose as they would when naming variables. Come up with examples of good and bad examples of function names such as: canVolume instead of just cV etc. Also remind the students of camelCasing. For example we can use **hypotenuseCalculator** as the name of our function.

2. **Parameter(s):** Variables that provide input to the function.
   Explain what input variables or function parameters are. Explain that the parameters are input values that the function requires to carry out its task, without these it cannot complete the task. In this case the hypotenuseCalculator needs two inputs: input a and input b which are the two known sides of the right-angled triangle. Also mention, how these variables are used only in the context of the function and not outside it. Explain that not all functions need inputs, some either do not have any or need any to perform certain tasks.

3. **Body:** Block of code that processes the input(s) and returns a value.
   This is the main body of the function where all the calculations are performed. For example the body of our hypotenuseCalculator function will be

```
hypotenuse = (a ** 2 + b ** 2) ** 0.5
return hypotenuse
```

Explain: Define Hypotenuse Function (continued)

Explain that the function definition starts with the reserved word **def** (which stands for define) followed by the name of the function and possible inputs required by that function. Our hypotenuseCalculator function should look like this:

```
def hypotenuseCalculator(a, b):

    hypotenuse = (a ** 2 + b ** 2) ** 0.5

    return hypotenuse
```

Explain: User Output - Refresher

Remind the students that **print("...")** is a function that prints the given output data to the console e.g. the following print function prints "Python is fun." in the console.

```
print("Python is fun.”)
```

Also, explain how we can combine multiple texts using a comma as a separator.

```
a = 5

b = 10

print("a = ", a, " b = ", b)
```

Explain: User Input - Refresher

Remind the students that **input("...")** is a function that prints the given question to the console and waits for the user to provide an input.
The following input function prints "Is Python fun? [Y/N] " on the console and expects the user to provide an answer.

```
input("Is Python fun? [Y/N]  ")
```

Remind the students that the program will not proceed until the user has given an input.

Explain: Input as a String

Remind the students that user input comes into Python as a string, which means that when they type the number 10 on the keyboard, Python saves the number 10 into a variable as a string, not as a number.

Explain how these two statements are different in how the computer processes them.

```
age = 10 #This is a number
age = "10" #This is a text/string
```

The result of an input("...") function is always a string. To convert it to a number (int, float etc.) we need to use the appropriate converter function.

Explain: Converter Functions

Explain how the **int** function converts a string or a number into a whole number (an integer), which means that everything after the decimal point is dropped.

```
int(123.456) = 123

int('123') = 123
```

Also, explain how the **float** function converts a string or a number into a floating-point number, which is a number with a decimal place.

```
float(12) = 12.0

float("123.456") = 123.456
```

Explain: Define Main Function

Once the students have understood how to create the hypotenuse function, all that is needed to be done is to call that function with the input values inside the main function.

```
def main():
    ab = int(input("Enter AB length: "))

    bc = int(input("Enter BC length: "))

    ac = hypotenuseCalculator(ab, bc)

    print("Length of AC is: ", ac)
```

Functions Implementation

Explain: Hypotenuse Program

Show the whole hypotenuse program to the students. If time permits, execute the program and show the various inputs and output.

```
# A program that calculates the hypotenuse.

def main():

    ab = int(input("Enter AB length: "))

    bc = int(input("Enter BC length: "))

    ac = hypotenuseCalculator(ab, bc)

    print("Length of AC is: ", ac)

def hypotenuseCalculator(a, b):

    h = (a ** 2 + b ** 2) ** 0.5

    return h

main()
```

The following slide shows the flow of execution of the above hypotenuse program.

Activity: Currency Converter

Explain the activity and walk through the steps of the flowchart on the board, explaining the steps and decision stages of the program. Students are encouraged to follow the flowchart in their workbook to help them understand it and then attempt to implement the code in Python.

main():

1. Ask the user to enter "1" to convert £ to € or "2" to convert € to £.
2. Call the function that performs the conversions with these inputs.

currencyConverter():

1. Takes in two inputs: user's choice of 1 or 2, value to be converted.
2. The function then performs the calculation (Note: research must be done to find out the conversion rates for pounds and euros.)
3. Returns the result in two decimal places.

Extension: Implement an extra choice to perform conversions into dollars, implement a third parameter variable which is the exchange rate.

Important: when using the input() function it is important to use the int() function to cast the string entered by the user into an integer. Most students forget this when using input() and asking for a positive number.
If necessary highlight the importance of not having any magic numbers and the use of constant variables.

Activity: Currency Converter Solution

Below is the full Python code for this task:

```
# A program that calculates the conversions of Pounds to
Euros and vice versa.
def main():
    choice = int(input("Enter (1) for Pounds to Euros
               and " + "(2) for Euros to Pounds: "))
    amount = int(input("Enter the value to convert: "))
    convertedAmount = poundsAndEuroConverter(choice,
                                                   amount)
    print("Converted Currency = ", convertedAmount)

def poundsAndEuroConverter(option,value):
    EXCHANGE_RATE = 1.14
    if (option == 1):
        result = value * EXCHANGE_RATE
    else:
        result = value / EXCHANGE_RATE
    result = round(result, 2)
    return result

main()
```

Activity: Temperature Converter

Implement a program which prompts the user to enter a temperature in degrees Celsius and then converts this to degrees Fahrenheit. Implement the program in Python using functions.

The program should produce the following output:

Degrees Celsius: xx.xx

Degrees Fahrenheit: yy.yy

Where xx.xx and yy.yy are the temperatures to 2 decimal places displayed in Celsius and Fahrenheit respectively. Hint: To convert the temperature in degrees Celsius to degrees Fahrenheit use the following equation:

Fahrenheit = (Celsius x 9/5) + 32

Hint: Be aware of integer division!

Extension: Convert from Fahrenheit to Celsius. (Students may use the internet to find the equation for this, or rearrange it themselves).

Activity: Temperature Converter Solution

Below is the full Python code for this task:

```
# Degrees Converter

def main():
    value = float(input("Enter Celsius value: "))
    result = celsiusToFahrenheit(value)
    print(result)

def celsiusToFahrenheit(celsius):
    fahrenheit = (celsius * (9/5)) + 32
    output = ("Degrees Celsius: %0.2f \nDegrees
     Fahrenheit: %0.2f " % (celsius, fahrenheit))
    return output

main()
```

Important to note: To set the output in 2 decimal place you must use the String format **"%0.2f" % variable** as shown in the code above.

Activity: Jacket Potato

A jacket potato vendor has asked you to write a program that will calculate prices for his shop. The customer will be asked two questions: would they like either a medium or large size jacket potato, and the number of toppings they would like.

Implement in Python, a program which prompts the user to enter the letter "M" for medium and "L" for large. Next ask the user for the number of toppings they would like to have. The total price is calculated according to the following table:

	Up to 2 toppings	3 or more toppings
Medium Jacket Potato	£2.50 + 50p / topping	£2.50 + 40p / topping
Large Jacket Potato	£3.50 + 55p / topping	£3.50 + 45p / topping

The code should then print the total cost of their order. An example run might be:

Medium (M) or Large (L) jacket potato: L

Number of toppings: 3

Total cost: 4.85 pounds

The cost is calculated as 3.50 + (0.45 * 3) = 4.85.

Hint: Loops are not required for this program, but nested if statements are.

Activity: Jacket Potato Solution

Below is the full Python code for this task:

```
def jacketPotatoCalculator(size, toppings):
    cost = 0
    MEDIUM = 2.50
    LARGE = 3.50
    MEDIUM_UP_TO_2 = 0.50
    MEDIUM_THREE_OR_MORE = 0.40
    LARGE_UP_TO_2 = 0.55
    LARGE_THREE_OR_MORE = 0.45

    if (size == "M"):
        cost = MEDIUM
        if (toppings <= 2):
            cost = cost + (toppings * MEDIUM_UP_TO_2)
        else:
            cost = cost + (toppings *
                                    MEDIUM_THREE_OR_MORE)
    else:
        cost = LARGE
        if (toppings <= 2):
            cost = cost + (toppings * LARGE_UP_TO_2)
        else:
```

Activity: Jacket Potato Solution (continued)

```
            cost = cost + (toppings *
                                        LARGE_THREE_OR_MORE)
    output = "Total cost %0.2f" % cost + " pounds"
    return output


# Jacket Potato Calculator

def main():
    choice = input("(M) for medium and (L) for large
                                        sized jackets: ")
    toppings = int(input("Number of toppings: "))
    result = jacketPotatoCalculator(choice, toppings)
    print(result)

main()
```

Activity: Lottery

The national lottery has contacted you to make a new lottery game.
The game will ask the user how many weeks they want to play and for 3 numbers they want to select; First between 1-10, second between 11-20 and third between 21-30.

If they match 1 number they win £10, 2 numbers £500, 3 numbers £1,000,000.

Implement in Python, a program which prompts the user to ask how many weeks they wish to play. Then enter in 3 numbers. 1-10, 11-20 and 21-30.

Example:

Week 1:

Winning numbers were: 5, 14, 24

Your numbers were: 6, 11, 29

Sadly no win this week

Week 2:

Winning numbers were: 7, 11, 25

Your numbers were: 6, 11, 29

You win £10!

…

Hint: Loops are required (for loops) with if statements for winning Students will need to import random in order to generate the random numbers. Students will also need to reset numbersMatched after each week

Activity: Lottery Solution

```
from random import randint
numbersMatched = 0

numOfTickets = int(input("How many weeks do you want to play?"))
num1 = int(input("Please enter in your first number (1-10)"))
num2 = int(input("Please enter your second number (11-20)"))
num3 = int(input("Please enter in your final number(21-30)"))

for i in range(numOfTickets) :
    winningNum1 = randint(1,10)
    winningNum2 = randint(11,20)
    winningNum3 = randint(21,30)

    print("Week: ", i+1)

    if num1 == winningNum1:
        numbersMatched += 1

    if num2 == winningNum2:
        numbersMatched += 1

    if num3 == winningNum3:
        numbersMatched += 1
```

Activity: Lottery Solution (continued)

```
print("The winning numbers were: ", winningNum1, winningNum2,
winningNum3)
    if numbersMatched == 0:
        print("You won nothing")

    if numbersMatched == 1:
        print("You won £10")

    if numbersMatched == 2:
        print("You won £500")

    if numbersMatched == 3:
        print(“Well Done! you won £1,000,000")

    numbersMatched = 0
```

Extension Activity: Lottery

Students can try to add in ticket costs. The game can then include profit, amount lost or even run until a profit is made.

Activity: Counting Heads

Ask the students to create a program that allows them to repeatedly throw multiple coins and each time they've been thrown to remove them if they land on heads.

The program should display the amount of throws and how many coins are remaining after each throw.

Activity: Visualising The Program

Take the results from the program and add them to the excel sheet. Visualise the number of coins left after every throw in the form of a line graph.

Activity: Counting Heads Solution

```
import random

initialNumberOfCoins = int(input("Please enter the
initial number of coins: "))
currentNumberOfCoins = initialNumberOfCoins

throws = 0

print("\nThrow = %d\nNumber of Coins remaining =
%d\n" %(throws, currentNumberOfCoins))
throws += 1
while currentNumberOfCoins > 0:
    for coin in range(0, currentNumberOfCoins):
        headsOrTails = random.randint(0, 1)
        if headsOrTails == 1:
            currentNumberOfCoins -= 1
    print("Throw = %d\nNumber of Coins remaining =
%d\n" %(throws, currentNumberOfCoins))
    throws += 1
```

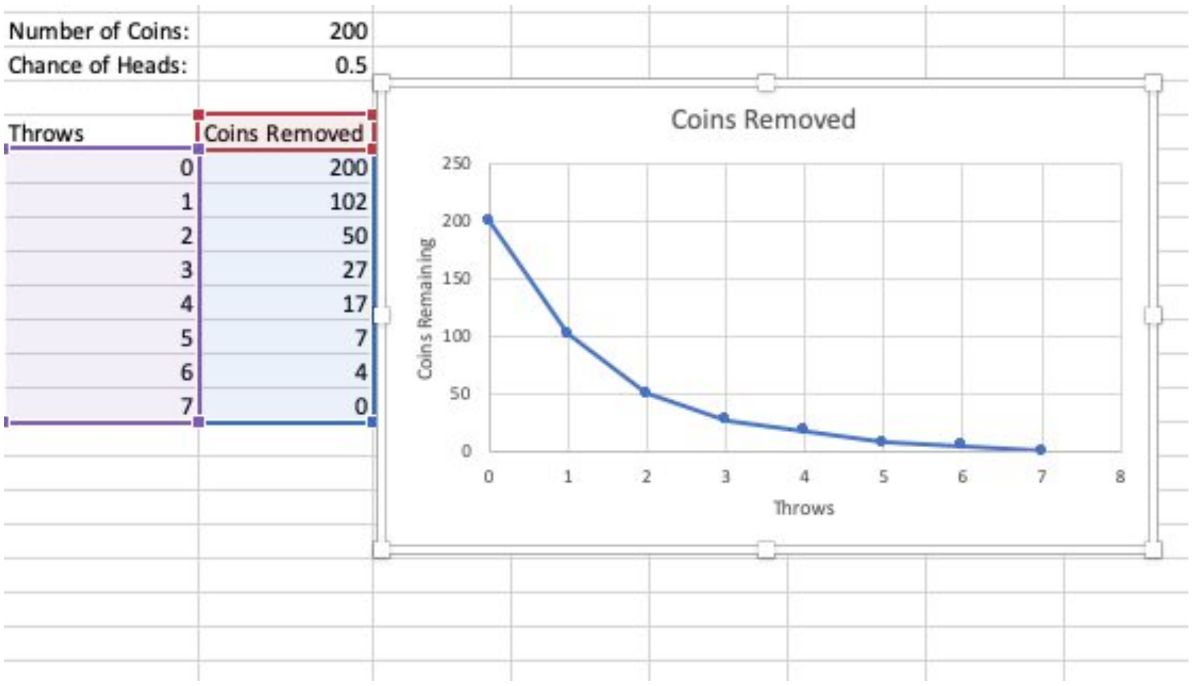
Activity: Counting Heads Extension Solution

```
import random

initialNumberOfCoins = int(input("Please enter the initial number
of coins: "))
currentNumberOfCoins = initialNumberOfCoins

throws = 0
halfCoinsRemoved = False

print("\nThrow = %d\nNumber of Coins remaining = %d\n" %(throws,
currentNumberOfCoins))
throws += 1
while currentNumberOfCoins > 0:
    for coin in range(0, currentNumberOfCoins):
        headsOrTails = random.randint(0, 1)
        if headsOrTails == 1:
            currentNumberOfCoins -= 1
        print("Throw = %d\nNumber of Coins remaining = %d\n" %
(throws, currentNumberOfCoins))
    if halfCoinsRemoved == False:
        if currentNumberOfCoins <= initialNumberOfCoins/2:
            halfRemovedThrow = throws
            halfCoinsRemoved = True
    throws += 1

print("Half of the coins were removed after throw %d." %
(halfRemovedThrow))
```

Activity: Visualising Half-life

The students can take the results from the program and add them to an excel sheet. They need to add the Throws in a column and Coins Removed in the next column. Then they should use these two columns to create a line chart with Throws along the x-axis and Coins removed along the y-axis. This will help them visualise the removal of coins over a period of time using a line-graph.

An example graph with number of atoms set to 500 and the chance of Heads set to 1/2 is given below.



Number of Coins:	200
Chance of Heads:	0.5

Throws	Coins Removed
0	200
1	102
2	50
3	27
4	17
5	7
6	4
7	0

Explain: Radioactive Decay & Half-Life

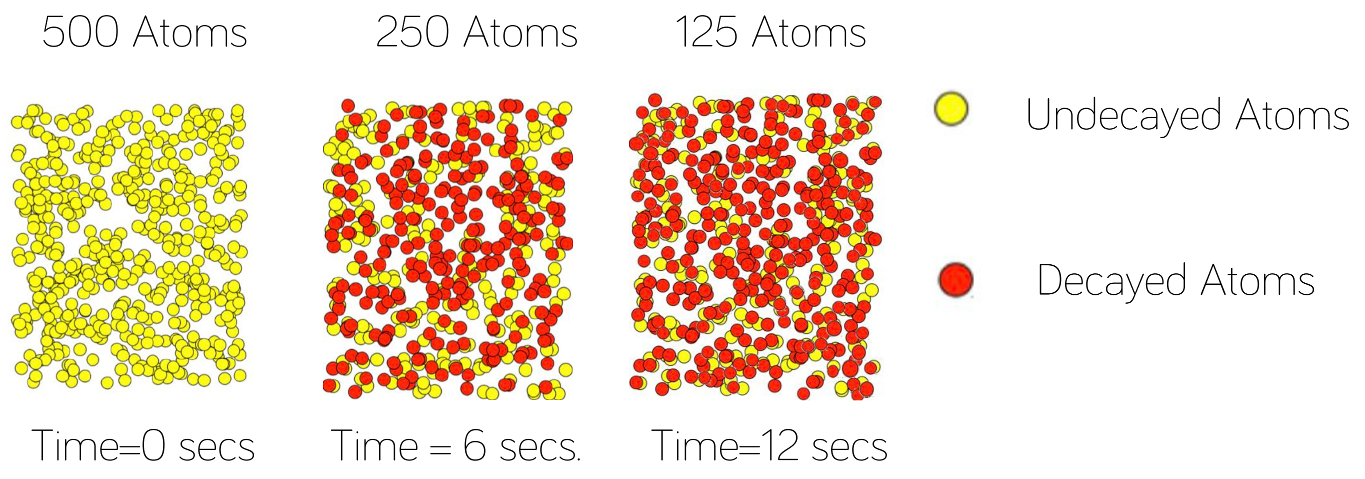
Explain to the pupils that they have just simulated a complex problem in Nuclear physics!

An atom's nucleus can only be stable if it has a certain amount of neutrons for the amount of protons it has. Nuclei with too many, or too few neutrons, do exist naturally but are unstable and will decay by emitting radiation.

Explain: Half-Life

Explain how half-life is the average amount of time it takes for the number of undecayed atoms in a sample to halve.

For example: If we begin with 500 atoms of an element. The amount of time it takes for 250 of these atoms to decay is the half- life of that sample.



Explain that for this sample, the half-life is 6 seconds as the atoms decayed to half of its value in 6 seconds.

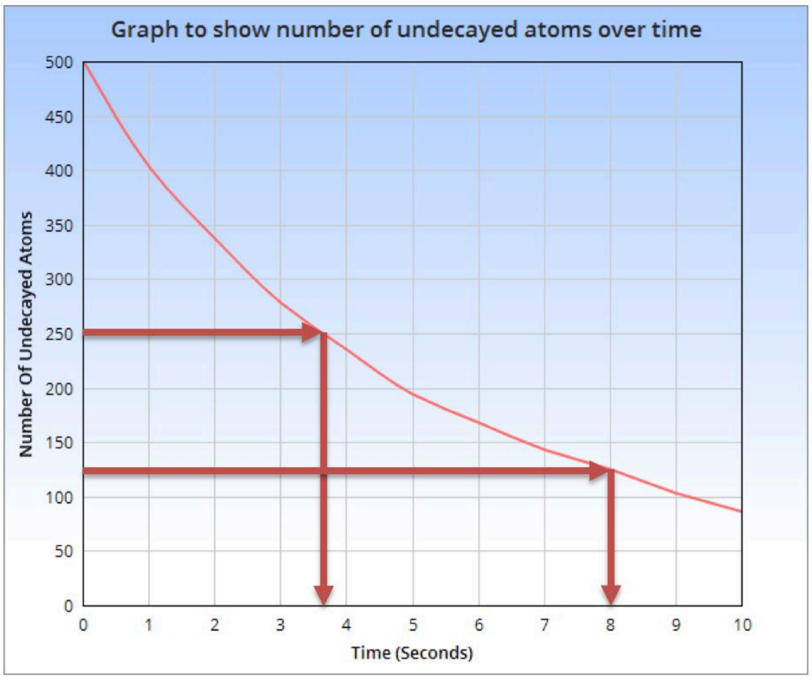
Explain: Half-Life Visualised

Display the graph to the class and get the students to figure out and draw the lines on to figure out the half life time in seconds. i.e. the time to go from 400 to 200 atoms would be around 4 seconds as well, and 200 to 100 would also be 4 seconds.

It is never perfect because radioactive decay is random, but the half-life is a good estimate.

Different elements have different half life times.

Note: Having an understanding of Half-lives of elements is important and useful in applications such as Carbon-Dating, Radioactive Tracers and for safely disposing of radioactive waste.



Explain: Half-Life, Carbon-Dating and Isotopes

Knowing how long a sample will remain at a certain level of radioactivity helps us decide which radioactive elements should be used in different situations.
Since the isotope Carbon-14 has a long half-life scientists and archeologists can use half-life to figure out approximately how old an organic object is. This is known as carbon-dating.

Explain: Radioactive Decay & Half-life

Explain that radioactive decay is the process which an unstable nucleus goes through due to an imbalance between the numbers of protons and neutrons. It does this in three main ways:

Alpha Decay:	The nucleus emits a group of two protons and two neutrons known as an alpha particle or a Helium nucleus.
Beta Decay:	A neutron transforms into a proton and emit a fast-moving electron (from inside the nucleus.)
Gamma Radiation:	A gamma ray is an electromagnetic wave, usually released after Alpha or Beta Decay has taken place.

Note that the radioactive decay is a completely random process. Explain that an atom could decay within a second, or might never decay at all.

There are 3 main types of radioactive decay. These are Alpha, Beta and Gamma. They differ in strengths with Alpha being stopped by paper, Beta by a sheet of metal and Gamma with heavily dense materials.

Explain: Medical Tracers

Mention to the students how radioactive decay is used in different situations; such as x-rays, sterilise medical equipment and to produce energy etc.

Ask the students what they think x-rays are most similar to (Gamma They are both electromagnetic waves i.e. forms of light)

Ask the students which could be used to for measuring thickness of toilet paper? (Alpha as it is stopped by a certain thickness of paper)

Activity: Recap Lists

What are Lists? Ask the students to write down what they know about Lists. Have they heard about them? What can a list contain? etc.

Explain: What are Lists?

Explain that a computer program often needs to store a sequence of values and then process them. For example if we had to store this sequence of values, how many
variables would we need?

32 54 67.5 29 35 80 115 44.5 100 65

This is where lists become very useful, saving us time and making the process of storing a sequence of values much easier.

List definition: A List is a collection of values which is ordered and changeable. Explain how the lists are only a container to store the values. We can change the order and the sequence of the values in the list.

Explain: Lists in Python

Explain how there are two ways of initialising a list in Python. Explain that one initialises an empty list and another initialises a list with values. We might need an empty list if we want to add items at runtime (as part of the program execution). If we already know the items that need to be part of the list, then we can use the second option to hardcode it in the program.

Explain that each item in a list has a corresponding index number associated with it. This index number is a non-negative whole number starting with the number 0. We use this index number to access an item in the list.

index	**0**	**1**	**2**	**3**	**4**	**5**	**6**
values	32	54	66	28	39	87	111

Explain that the values/items in a list need not be in any particular order, but they usually belong to the same type. In other words, they usually have all numbers or all text in them.

Explain: Accessing Lists in Python

Explain how we can use index numbers in Python to access an item from a list. For accessing the third item, we need to use the index number 2 as the index numbers start from 0. Similarly, to access the first item we use the index number 0.

Ask the students what would happen if we provide an index number which is out of the range of the list. For example in the given list, we have 7 items and hence the index numbers range from 0 to 6. If we use the index number 7, Python will throw a runtime error.

Explain: Replacing Values in a List

Explain how we can use the index numbers again to replace a value in a list. We first identify the index number of the value that needs to be changed and then set the value in the corresponding index number to a different value. In our example, we will be replace the 6th element with the index number 5 which has the value 87 to 88.

Also explain to the students that if we try to update/replace a value in a list with the index number beyond the given range, then we will get a runtime syntax error. This is explained using list boundaries.

Explain: List Boundaries

Explain that the students have to be careful that the index number stays within the valid range. Most mistakes made when using lists happen because of the invalid index range. Attempting to access an element whose index number is not within the valid index range is called an out-of-range error or a bounds error which in turn causes run-time exception. The following example will throw an IndexError at run-time because we are trying to access an index number out of range of the list.

Explain: Length of a List

Explain that if we want to find the number of elements in the list, which is also called as the length of the list, we can use the **len()** function on the lists.

Explain: Iterating a List

Explain that iterating a list is the process of accessing each and every element of the list in sequence.

A **for** loop is ideal to iterate through the items in a list. We use a variable in the for loop that corresponds to the index number.

The **range(n)** function yields the numbers 0, 1, … n-1 and **range(a, b)** returns a, a+1, …,b-1 up to but not including the last number (b). For e.g. range(5) returns 0, 1, 2, 3, 4 where as range(2,5) will return 2, 3, 4.

The combination of the for-loop and the range() function allows you to iterate through the list easily. In our example, we will be using range(n) function where n will be the length of the list. The len() function will be used to get the length of the list. This example prints all the elements of the list in the console.

Activity: Review Lists

Students should answer the questions on lists in their workbooks.

Explain: Linear Search

Recap linear search ensuring all students understand how it is performed. A linear search is a simple search process where a list is searched sequentially until the required value is found.

Explain: Binary Search

Recap binary search ensuring all students understand how it is performed. Emphasize the importance of the list being sorted before performing binary search.
A binary search algorithm (also knowns as half-interval search) is the process in which:

- The middle value in a sorted list is inspected to see if it matches the search value.
- If the middle value is greater than the search value, the upper half of the list is discarded.
- If the middle value is less than the search value, the lower half of the list is discarded.
- This process is repeated, with the list halving in size each time until the search value is found.

Activity: Binary Search in Python

Students should create a flowchart in their workbooks listing all the variables they will need and then attempt to implement the flowchart in Python.

The full Python code is given below:

```
# Binary Search
def binarySearch(sortedList, item):

    first = 0

    last = len(sortedList) - 1

    found = False
    while first <= last and not found:
        midpoint = round((first + last) / 2)

        if sortedList[midpoint] == item:

                found = True

        else:
                if item < sortedList[midpoint]:

                    last = midpoint - 1
                else:
                    first = midpoint + 1
    return found
```

Activity: Binary Search in Python (continued)

```
# Binary Search main entry
def main():
    mySortedList =

    [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
    itemToFind = 7
    print(binarySearch(mySortedList,itemToFind))

main()
```

Explain: Bubble Sort

Recap bubble sort ensuring students understand how it is performed. From left to right, compare two numbers, swap if needed. Repeat until all numbers are in correct order.

Activity: Bubble Sort in Python

Students should create a flowchart in their workbook listing all the variables they will need and then attempt to implement the flowchart in Python.

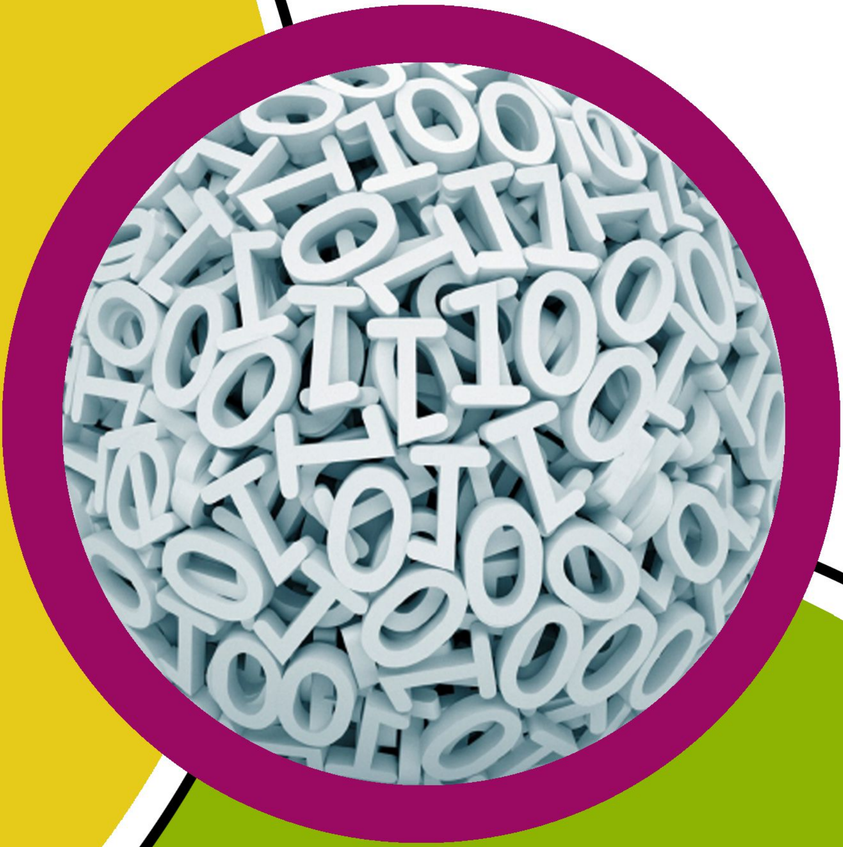
The full Python code is given below:

```
# Bubble sort function
def bubbleSort(aList):
    n = len(aList)
    swapped = False
    while n > 0:
          swapped = False
          for i in range(1, n):
              if aList[i-1] > aList[i]:
                     temp = aList[i]
                     aList[i] = aList[i-1]
                     aList[i-1] = temp
#aList[i], aList[i-1] = aList[i-1],aList[i]
                     swapped = True

          n=n-1

    return alist
```

Activity: Bubble Sort in Python

```
# Bubble sort main
def main():
    unorderedList =
        [34,23,56,89,23,43,55,75,4,2,6,10,11]
    print(bubbleSort(unorderedList))

main()
```

Important note: The bubble sort program may be difficult to understand. It can be explained by going through the program line by line and performing a dry run hand tracing if confusion arises.



technocamps

@Technocamps

Find us on Facebook