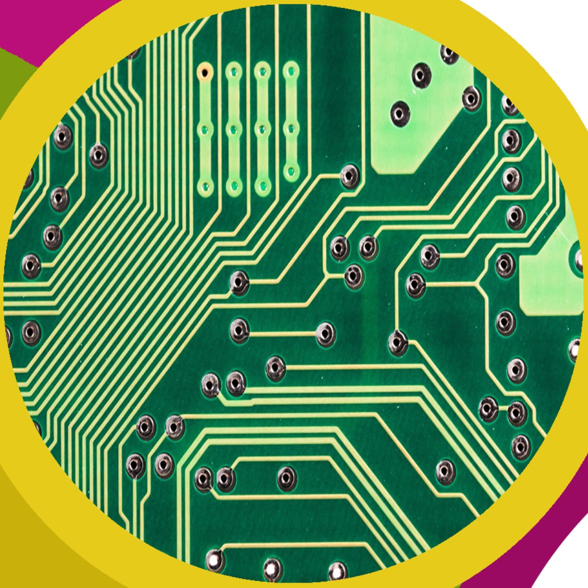
technocamps

Python Mathematics Teacher Guidance



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

Links to Science and Technology AoLE

Computing:

(PS3) I can use conditional statements to add control and decision-making to algorithms.
(PS3) I can identify repeating patterns and use loops to make my algorithms more concise.
(PS3) I can explain and debug algorithms.

(PS2) I can follow algorithms to determine their purpose and predict outcomes.

Links to Other AoLEs

Mathematics and Numeracy:

Geometry:

(PS3) I can explore and consolidate my understanding of the properties of two-dimensional shapes to include the number of sides and symmetry.
(PS3) I can demonstrate my understanding of angle as a measure of rotation and I can recognise, name and describe types of angles.

Algebra:

(PS2) I have explored patterns of numbers and shape. I can recognise, copy and generate sequences of numbers and visual patterns.

The Four Purposes and Cross-Curricular Skills

This resource provides opportunities for **Critical Thinking and Problem Solving** throughout. Learners are able to recognise potential problems with algorithms which are not in the correct order and when debugging any errors they encounter when programming.

Learners are also allowed to show **Creativity and Innovation** when they are asked to write algorithms to produce their own shapes and patterns, as well as experimenting with variables to produce unique patterns.

The **Data and Computational Thinking** strand of the **DCF** is also prevalent with opportunities for learners to understand the importance of the order of statements, when creating and refining their algorithms. They will be able to detect errors and use iteration to improve efficiency of their code.

Why Is Learning This Important?

This resource allows learners to develop their understanding of geometry through angles and symmetry of shapes, by allowing them to decompose the creation of shapes into a sequence of instructions utilising the "turtle" library in Python. Alongside this, learners can develop their understanding of programming and debugging algorithms in a modern and popular programming language before experimenting and exploring how modifying their algorithms can produce unique outputs in a creative way. There is opportunity to link this with geography by exploring the flags of countries which can be produced by sequencing instructions.

Suggested Approaches Key

In this suggested approach 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 session and collect materials at the end.

Introduction

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

Explain: Topics Covered Today

We will be learning about geometrical shapes, a topic within Mathematics. We will also see why we are using Python and we will create programs that draw geometrical shapes for us.

Activity: Draw the Image

Provide a set of instructions and ask the students to follow the instructions in the given order to draw an image.

Discuss: Draw the Image

Towards the end of the activity, ask the students to describe their image and check how many students drew the same image.
If there were different images discuss how given the same instructions, there were differences in results.
Discuss what would happen if the computer was given the same instructions. Would it have drawn one of the students images?

Explain: What is Programming?

Programming is telling a computer what to do using a set of ordered instructions. The set of ordered instructions is called an **algorithm**. The language used to tell the computer what to do is called a **programming language**.
Unless we give the computer step by step instructions to draw a shape, it would not know how to draw it. It will follow the instructions in the given order and the given format.
If it fails to create the expected shape, it is the fault of the instructions provided and not the computer.

Discuss: Lines of Symmetry

Ask the students what they remember about symmetry and lines of symmetry in geometry. Ask if they can name some shapes (2D) which have lines of symmetry.

Explain: Lines of Symmetry

A 2D shape is symmetrical if a line can be drawn through it so that either side of the line looks exactly the same. That line is called the line of symmetry.

A square, which is a regular polygon, has 4 lines of symmetry. A rectangle has 2 lines of symmetry. A regular pentagon has 5 lines of symmetry. A kite and a trapezium have 1 line of symmetry. Some geometrical shapes have no lines of symmetry, for example a parallelogram.

Discuss: Angles?

Ask the students about the term angles in geometry and what are the various angles associated with different 2D shapes.

Geometry Refresher

Explain: Angles

An angle is formed when two lines meet around their common point. There are 360º in a full rotation. As the angle increases, the name changes.
Acute angles are those between 0º and 90º
A right angle is 90º
Obtuse angles are those between 90º and 180º
Reflex angles are those between 180º and 360º
One full rotation contains 360º

Activity: Geometry

Identify the 2D shapes from their given properties on sides, angles and symmetry.

Explain: Geometry in Programming

We know that programming is about instructing the computer to do what we want it to do.
If we want it to draw some shapes for us then we have to provide exact instructions on how to draw those shapes using sides and angles.
In order to do that we need to use a programming language which the computer understands to provide our instructions and the computer will interpret it and covert it into an image.
Clear instructions are essential. If we provide incorrect commands or if the computer does not recognise a command, it will complain.

Activity: What is Python?

Ask the students to write down what they think of when they hear the word Python.

Explain: Python

Python in a computer science context is not a snake or the Harry Potter language “Parseltongue”.
Python is a programming language which tells the computer what to do through the use of algorithms.
Python is free.
Python is easy to learn, read and code in.
Python is interactive and portable.
Python is considered a high level programming language and flexible.

High level means that the instructions are similar to English language instructions. It is easier to read by a human but not necessarily by the computer. For a computer to understand a high level language, it uses a compiler or interpreter to convert the English-like instructions to binary commands.

The Python language was created by Guido Van Rossum in 1991 and is being used now more than ever. He liked Monty Python’s Flying Circus TV series a lot, so named the language after that.

Explain: Turtle Graphics in Python

Python has a lot of libraries. A library is a set of commands for a specific purpose. For example the Turtle library we will be using today has commands related to graphics. It has the necessary commands to create some fun mathematics graphics. We will be using these to create some of the 2D shapes that we have just talked about.

Activity: Draw a Square

Assume you are a turtle and you are given two commands:

a) **forward x**: you can only move forward x steps. E.g. forward 100
b) **right y**: you can only turn right (clockwise) at y angle. E.g. right 90

Using these two instructions try to draw a square. Remember the angles between the sides in the square and the number of sides.

Explain: Square - Instructions

Let us go through the instructions to draw a square using those two commands:

forward 100: move forward 100 steps
right 90: turn 90 degrees and wait

This draws the first side of the square.

forward 100: move forward 100 steps
right 90: turn 90 degrees and wait

This draws the second side of the square.

forward 100: move forward 100 steps
right 90: turn 90 degrees and wait

This draws the third side of the square

forward 100: move forward 100 steps
right 90: turn 90 degrees and wait

This draws the fourth side and completes the square.

Explain: Square - The Turtle Way

Now let us convert the instructions to the commands that Turtle library has and see how it is written.

In Turtle we write these commands using the format **command(value)**. For example forward(100) instead of forward 100 and right(90) instead of right 90.

Since a square has four sides we repeat the commands, move forward and turn right, four times.

Explain: First Turtle Program

Now we finally add a couple more lines of code to get our first Turtle program.

The **import** keyword tells Python that we are using the specified library and functions from that library. It is important to have all the import statements at the beginning of your program.

The functions or the commands provided by the library should be called using their name dot-suffixed after the module name. In our case we create a new turtle from the Turtle library and call it pen. It can be called anything as long as it is not called Turtle (with a capital).
We then use this to call all the functions. Hence we have to use pen.forward(100). If we call forward(100) without the pen. (pen followed by a dot) then Python will not know what to do with the command. We then write all out other commands to draw the square using the pen object.

Activity: First Turtle Program

You will have to open a program called IDLE which is used for Python programming. Create a new file, copy the contents of the Square program that is provided for you. Save it as **square.py** and run the module. Once you have completed the square program, change the code so that it will create a rectangle.
Reference programs: **Square.py** and **Rectangle.py**.

Turtle Commands

Explain: Turtle Commands

Now that we have seen how the turtle library can draw a simple square for us, let us have a look at some more commands that the turtle library supports.
Apart from right and forward, it also supports:

left(__)	to turn left (anticlockwise)
backward(__)	to move backwards
color("colour name")	to set a colour for the lines that the turtle will draw
fillcolor("colour name")	to fill the shape marked by begin and end fill commands with the colour given
begin_fill()	marks the start of the area to be filled
end_fill()	marks the end of the area being filled

Explain: Square with Colours

Using the commands that were listed, let us modify the program that draws the square to add some colours. This program will draw the square using the colour yellow and fills the square in green.
We have to ensure that we give the **begin_fill()** command to mark the starting point of our shape and **end_fill()** command to mark the end point of our shape.
If we give the begin_fill() command without the end_fill() command or vice versa it will not fill the shape with colour.

Activity: Colour the Shapes

Let us modify the square program to draw the square with red lines and fill it in green.
We need to use the color(), fillcolor(), begin_fill() and end_fill() commands to achieve this. Once that is completed, we can modify the rectangle program to draw the rectangle with orange lines and fill it in purple.

Reference Program: **SquareWithColour.py**

Discuss: Pen Commands

Here is a scenario:
I want to draw a square in red and fill it with green. I move my turtle forward once the square is drawn and then draw a rectangle next to my square in black and fill it with blue. In between the square and the rectangle I see a red line. Why do you think that is?

Explain: Penup and Pendown

When we are drawing on paper, the pen/pencil is down on the paper. When we have to draw two squares next to each other, we draw a square, lift the pen/pencil up, move it a little further across the paper and then place the pen/pencil down and start drawing again.

In our turtle programming, by default **the pen is down** as it has to draw the lines when moving forward and backward.

When we finished drawing the square the pen was still down when we moved forward to draw our rectangle and since the square was drawn using red colour the line was also in red.

We could solve it by using the **penup()** and **pendown()** commands. Once the square is complete, we call the penup() command, move forward and then call the pendown() and start drawing the rectangle. This avoids the red line between the two shapes.

Note the lines with **#**, these are called **comments**. These are used for us humans to understand what we doing in the program and for anyone reading our code. Python will ignore these lines as they don't mean anything to it. It is good practice to write comments whilst writing any code.

Activity: Pen Exercises

Let us use the penup and pendown commands to draw two shapes of your choice next to each other with some space in between them. You can use colours of your own choice to fill them too.

Reference Program: **TrianglesWithPenCommands.py**

Activity: Fun Flags

Let us use the commands we have learned so far to draw one or more of these lifeguard flags or a flag of the world.
On a good day when you go to the beach, you will see one or more of these flags in the area to indicated the swimming section and where the lifeguards are on duty.
Choose one or more flags and program the turtle to draw them for you.

Explain: Repeating Instructions

Reviewing our first Square program in Turtle, we can see that there are some commands that get repeated more than once.
For example the commands forward(100) and right(90) are written once and then repeated three times to complete the square.
This process is called **iteration**.

Iteration

Explain: Iteration

Iteration is the process where one or more steps are repeated more than once. This process is also known as **looping**.
There are various types of loops. We are going to be looking at loops or iterations which are based on a specific range. In other words, loops where the steps or commands are repeated a specific number of times.

Explain: For Loops

We are looking at **For** loops in Python which allow you to repeat a set of instructions a specified number of times.
An example of how we translate our Square program using loops can be seen in the slides. The lines in green are rewritten using iteration.

Explain: Dissecting the Loop

A **for** loop consists of the following format:

```
for <loop variable> in <iteratable>:
        <loop body>
```

In our session, we will be looking at **range(<begin>, <end>, <stride>)**. The stride indicates the amount to skip between values. For example, range(2, 10, 2) will result in a list of numbers starting from 2 and ending in 10, but each number step will be 2. So the resulting list will be 2, 4, 6 and 8. The <end> number is not included in the list.

For simplicity, we will use the easiest form of range which is range(<end>).

Note: if <begin> value is not specified it will start with 0 and if <stride> value is not specified it will default to 1.
In our example, range(4) would result in the loop variable taking values 0, 1, 2 and 3 thus executing the statements in the body of the loop four times.

Ensure that the **for** and **in** keywords are not missed.

The loop variable is any variable name which can be used as the counter variable for the loop. We usually use **i** as the loop variable as "i" indicates the index of the counter. The loop body is any set of commands which needs to be executed iteratively one or more times.

Explain: Indentation in Loops

It is also important to ensure that the loop body is indented by a few spaces (use tab). Python identifiers the loop body by the indentation. If the indentation is missed, Python will not be able to recognise the loop body which will result in the syntax error “expected indented block” during runtime.

Activity: Let’s Loop

A couple of programs are provided where the instructions/commands are repeated more than once. The students have to go through the program and replace the repeated set of commands with a for loop that has been discussed earlier.

Reference Program: **SquareWithLoops.py**

Explain: The Three Octagons

A sample program which uses all the commands of the Turtle that have been discussed so far (movement commands, pen commands, colour commands and the for loop) should be shown. This program draws three octagons next to each other to form a pattern. Each octagon is drawn using a loop.

Activity: Loopy Designs

We ask the students to create some designs like the three octagons using the commands learnt so far.

Reference Program: **ThreeOctagons.py**

Explain: Loop it with a Variable

As an extension to the for loops, it is also possible to use the loop variable used in the definition, in the loop body and create some designs. An example program and its output are shown. Alternatively, we can also run the **LoopitWithVariable.py** and show the live example of the program.

Extension Activity: Loop it with a Variable

We ask the students to create some designs using the loop variables in the loop body just like the one shown in the previous slide. We can also show the previous slides example for their reference.

Explain: 2D Shapes

We shall quickly revise some of our regular 2D polygon shapes. Each of these shapes, triangle, square, pentagon etc., have a different number of sides. They range from 3 — 10 sides. What about their angles? There are two types of angles for any regular 2D polygon: Interior Angles and Exterior Angles. We will do a quick review of the angles in the next few slides.
Note: We will be looking at only regular polygons for our exercises.

Explain: Exterior Angles

An exterior angle is the angle outside the polygon made by extending one of the sides. The sum of all exterior angles is 360^o. How did we arrive at 360^o?

If we were to place a pencil horizontally along the side of the polygon and then begin rotate it around the shape, following the sides, we would notice that the pencil completes one full rotation by the time we return to the original position.
We know there are 360^o one full rotation (a circle) so the sum of the exterior angles must be 360^o as well.

Explain: Size of Each Exterior Angle

All the interior angles of a regular polygon are equal, hence all the exterior angles will be equal too. If we want to know the size of each exterior angle, then we use the formula **360º / number of sides**. Since the total sum of all exterior angles is 360º, we can get the size of each individual exterior angle by dividing 360º by the number of sides.

Explain: Applying Exterior Angles

Where do we apply these exterior angles? Let us assume that we start a car and that the car travels in a square for full rotation, turning four times. For each turn, the angle of turn would be 360º/4 = 90º.

Activity: Angles

Now that we know the interior and exterior angles for each of the regular 2D polygons, let us complete the given table by identifying the 2D polygon and filling in the number of sides, the total sum of exterior angles and the size of each exterior angle for each of the given regular polygons.

Discuss: Angle Facts

Let us go through the shapes and their exterior angles. Can we rewrite our Turtle program to create any or all of these 2D shapes based on their number of sides?

Explain: Automate the Angles

In order to automate the 2D shapes based on the number of sides, we need to ask the user the number of sides and store that in a variable. The loop will iterate for the given number of sides. We will also use that variable in our calculation of our exterior angle in the loop body which is 360º / number of sides.

A sample program which automates the angles is shown and the key commands that are used to in the automation are highlighted. The first key command is an input command which is used to accept the number of sides from the user and the value is stored in a variable called sides.

Explain: Input and Variables

input() is a command used to ask the user a question and we can then use the answer in the Python program. The structure of the input commands is **input(<question>)**. If the question is missed, then Python will wait for the user to enter the data without displaying any information.

It is always good practice let the users know what the input is about. Any input given by the user will be in a text format, even if they enter a number! If we want to use the answer/input provided by the user, we need to store that information in a variable.

Since the input provided by the user is in text format, we cannot use it directly in our mathematical calculations. We need to convert them into numbers before we can use the formula 360º/number of sides. In order to do so, we can use the **int()** function which is an integer function which takes data in text format and coverts it to data in number format.

Explain: How Does It Work?

A screenshot of what the input() function does when the Python program is run is shown so that the students can see how the program works and what screen they should use to enter the number of sides.

Alternatively, we can use the **AutomatedShapes.py** program to show a live demonstration of the same commands.

Explain: Result

A screenshot of the result of the AutomatedShapes program is shown along with the prompt screen where the user has entered the number of sides. This slide is shown to emphasise that there will be two windows in cases where there is an input involved. When the turtle program is executed/run, apart from the prompt window it will also open a drawing window. The user has to ensure that the give the input in the prompt window so that the required shape will be drawn on the drawing window.

Activity: Draw Any Shape

The code similar to that used to do the automate shapes from the previous slides is given but is jumbled up. The students have to rearrange the code in the correct order based on what we have learned so far.

Extension Activity: Colour the Shape

As an extension to the automated shapes, add another input which asks the user for the desired colour of their shape. Store the colour in a variable called **usercolour** and use it in the fillcolor() command.

Explain: Conditionals

We have seen a series of statements executed in a sequence and also a sequence of statements executed in an iteration. What if we want to change the flow of how the program works? For example, if we want to change the order of execution of a command based on a condition. In such cases we use conditionals.

We use conditions in day to day life all the time. For example, if it is raining outside then take and umbrella. In this example, the action of taking an umbrella is based on the condition of raining.

Here is another example. If my homework is completed then I play games on my Xbox else I sit and complete my homework. In this example, if the condition of completion of the homework is true, then the person can go play on their Xbox, but if the condition is false, i.e. if they haven't completed the homework, then they have to go and complete it.

Apart from specifying what should be done in the case that the condition is met, we have also specified what should be done in the case that the condition is **not met**. In any conditional there can be only two outcomes: **True** or **False**.

Explain: Python Conditionals

How do we write conditional statements in Python? The format is as follows:

```
if (condition):
            then do these things if the condition is True
else:
            then do these things if the condition is False
```

The indentations of the True conditional statements and the False conditional statements are important just like we did it in the loop body of an iteration. If we do not do the indentation, then Python won't know which statements belong to True condition and which belong to False.

In our example of automating the shapes, if the user gives the input as 4 sides, we have to decide if we want to draw a square or a rectangle as both of them have 4 sides. To achieve that, we ask the length of two sides in two variables and use them to draw the 4 - sided shape.

Explain: Conditionals

The conditional statement is written after the keyword **if**. The condition is any expression that can result in a True/False outcome. It is important to ensure that the condition ends with a colon (:). The condition is followed by the statements that need to be executed in case the condition is True and these should be indented.
The conditional else just has the keyword **else** followed by the colon (:). This is then followed by the statements that need to be executed as part of the False outcome of the condition.

Explain: Using Conditionals

The automated shape program is enhanced to use conditionals. If the user input is 4 then we ask the user to input side1 and side2 and draw a square or rectangle using the two sides given. But if the user input is not 4 then we use the formula 360º/sides to draw the shape.

What would happen if side1 and side2 are set to the same value?
Turtle would draw a square. If side1 and side2 are set to different values then Turtle would draw a rectangle instead.

Ensure that the indentations are in place.

Activity: Conditional Shapes

Reorder the automated shapes program to add the conditions that were discussed.

Reference Program: **AutomatedShapesWithConditions.py**

Explain: Stamp

When we look at post (not that we use it much anymore, maybe except during Christmas or other such festivals) we see various stamps on them.

These stamps tell us a tale. They tell us where our post has been. For each of the different places it has been (mostly different countries) there will be a stamp on it giving us the route it has travelled.

In Turtle programming, when the turtle is moving forward or backward it doesn't leave any footprints. We don't know where the turtle has been. If we want to know the route the turtle has taken, we need to use stamps.

Stamping leaves an impression of the turtle on the screen and it works even when the pen is up (i.e. not in drawing mode). We need to use a command called **stamp()**. A simple example using stamp is given. This creates a square shape with a turtle stamped at each corner.

Activity: Stamp the Turtles

A sample code has been given which the students can modify to create random shapes and also create stamps of the turtle on the screen for each shape.

Reference Program: **SquareWithLoopsAndaStamp.py**

Activity: Turtle Review

We come to the end of our workshop. Before we do a final program, let us do a quick review of what we have learnt today and see if we can update the worksheet by identifying the correct lines of the program for the given concept.

Activity: The House

As part of our first activity of the day, the starter activity, the students were asked to draw an image based on a series of instructions. Students "should" have drawn an image of a very simple house.
That activity used only simple geometrical 2D shapes to create the house.
Now students should use their Turtle programming knowledge to write a program which draws the house for them.

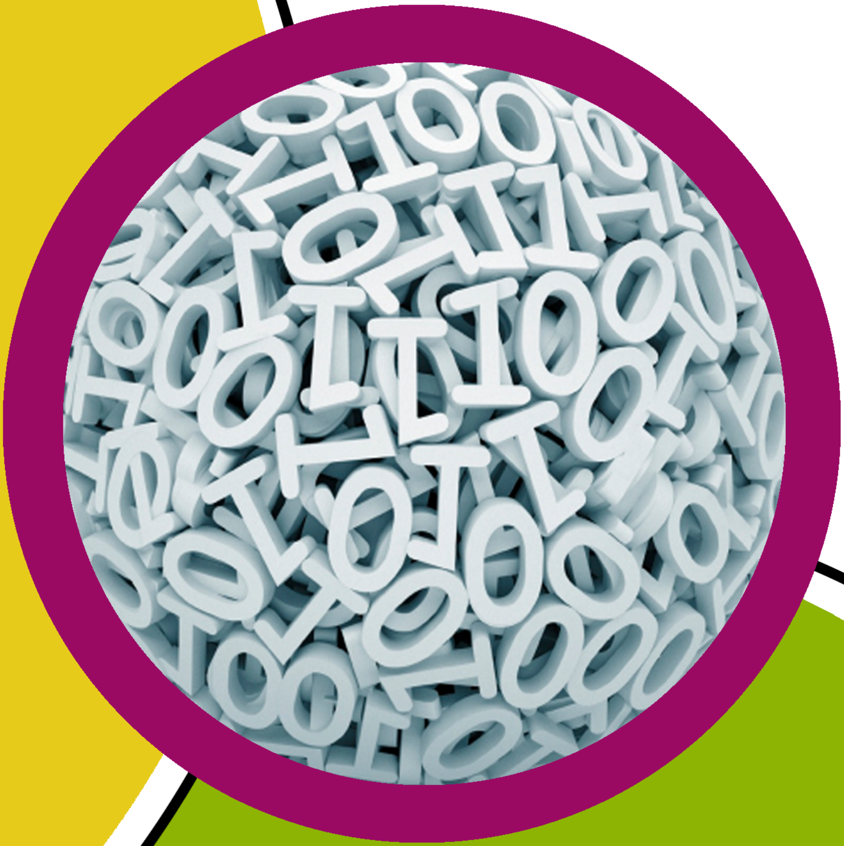
We need to ensure that we give correct and clear commands for the turtle to draw.

Differentiating for Learners

- Opportunity to challenge learners by suggesting more difficult country flags to recreate.
- It may be of use to provide some learners with the code whilst giving others snippets of code to be assembled while others need to use what they have learnt in prior activities to write algorithms without any code provision.
- Most programming environments allow customisation of the user interface and other accessibility features to support all learners.

Where To Go Next

- Opportunity to teach functions in Python, for example, defining a function to draw any regular polygon by inputting a number of sides.
- Another example could be defining a function for drawing a house, and then repeatedly calling the function to draw a row of houses. The functions could allow input of arguments to control the size and number of houses in the row.
- A follow-up task would be to ask pupils to create a drawing with multiple shapes, asking them to demonstrate and reflect on the benefits of compartmentalising the code for individual shapes.



technocamps

@Technocamps

Find us on Facebook