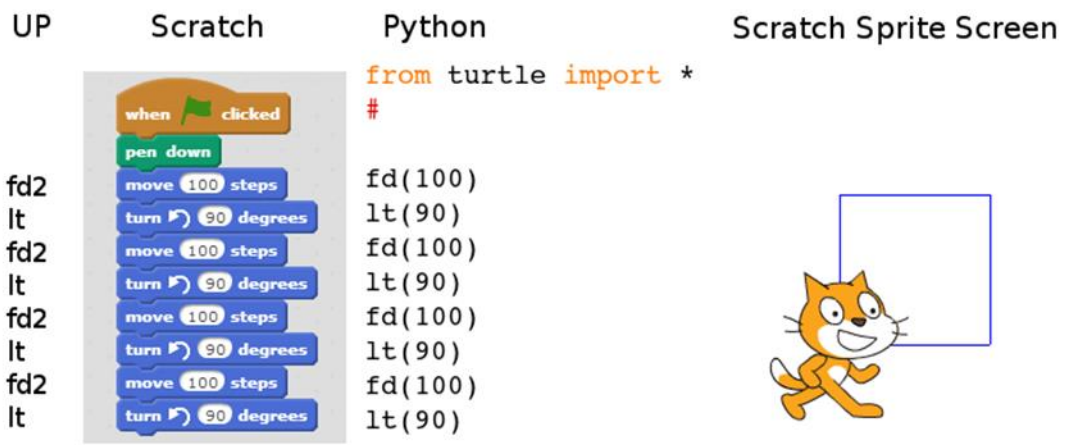
COMPUTING AT SCHOOL
EDUCATE · ENGAGE · ENCOURAGE
In collaboration with BCS, The Chartered Institute for IT

UCL ENGINEERING
Change the world

UCL/CAS Training for Teachers
Algorithms and Programming Module 1



WORKBOOK 11

SCRATCH: A WAY TO LOGO AND PYTHON FOR TEACHERS

Exemplar learning and teaching pathways in computational thinking from scratch, practically applied to algorithms and practising the 5 fundamental control structures of programming in Scratch 2.0/Python 3 in procedural and event-driven paradigms of programming.

- Addressed to Teachers
- Workbooks 8 – 13
- Activities are graded: easy to hard – 0 to 5*. You should attempt every activity marked without a star or marked with one *.

COMPUTING AT SCHOOL
EDUCATE · ENGAGE · ENCOURAGE
In collaboration with BCS, The Chartered Institute for IT

UCL ENGINEERING
Change the world

UCL/CAS Training for Teachers
Algorithms and Programming Module 1

CONTENTS

OVERVIEW OF PATHWAY ONE: GEOMETRIC SHAPES AND PATTERNS

- We have prefaced this pathway with an introduction to the Reflective Practitioner and Pedagogy in Computer Science with a view to you developing and extending your own model of teaching and learning in traversing this pathway.
- We first focus on learning and teaching practical programming unplugged that is away from the computer.
- We then go through some basics in the background of subject shapes, so we are more prepared to succeed when we hit the screen.
- We introduce Scratch 2.0 as our learner vehicle, and use it as a pseudo-code to Python 3, although it is possible to go directly to Python 3 as Scratch and Python versions run in parallel.
- Our first approach is to enter the world of regular shapes and patterns illustrating and illuminating them with the three control structures we have already met: ***Sequence, Repetition and Function***
- We round off with a project to build an App with the control structures ***communication and selection.***
- In practice, we engage immediately and explicitly with the use of functions because they constitute the building blocks of programming -- even the program instructions to drive sprites/turtles are examples of (system) functions with parameters, which we can use to experiment.

MISSION 1: SEQUENCE AND *REPETITION* IN UPL, SCRATCH, LOGO AND PYTHON

- We use Scratch to develop and run our programs and adopt it as a pseudo-code to transition to a Python program. The Python program in the text is the exact equivalent of the Scratch program mirroring exactly the control structures and instructions in the program.
- When you want your class to make the transition to Python, the mapping of the correctly working Scratch program to the Python 3 version is 1-1 with instructions and an automatic formulation with the 5 control structures. The task becomes, in the first instance, one of managing the Idle environment for Python 3 and getting the syntax of Python 3 correct so that the Python version runs correctly. See Workbook 7: Plug in to Python 3 and Workbook 6: Plug in to Scratch 2.0.

SEQUENCE

Sequence is setting up the instructions in a program in the right order usually set out one after the other, reading left to right and top to bottom.

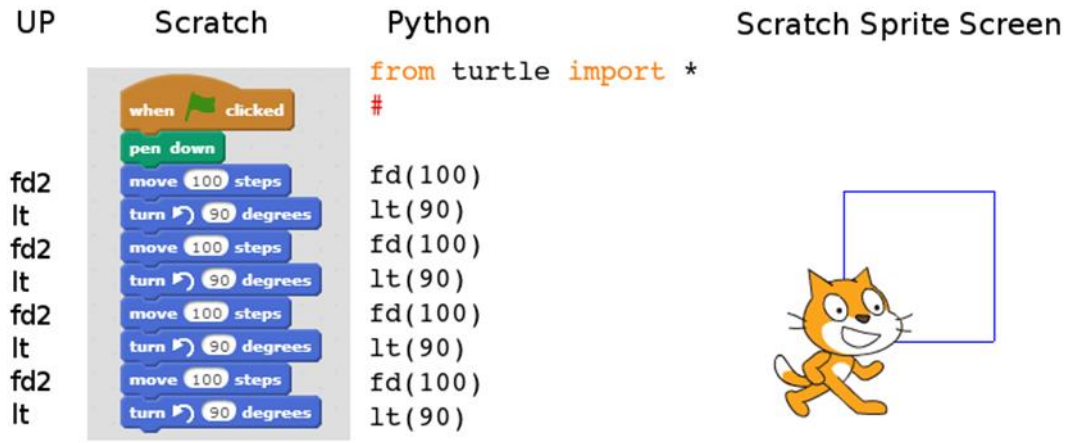


Figure 4. Programs for drawing a square in UP(Logo), Scratch and Python

In all the programs in Figure 4, they read in order from top to bottom, and the drawing/motion performed is a square. Logo and UPL are written horizontally. We have entered the UPL program vertically for ease of comparison.

SCRATCH 2.0

From here on, we are first going to use Scratch to develop, test and run any program. We won't have to worry about making any syntax errors, e.g. spelling mistakes, misplaced brackets, formatting, in Scratch, because the program pieces are already written and fit together under each other in ***sequence*** like interlocking jig-saw pieces. When the Scratch program we have constructed is working, we then map it one-to-one onto the Python equivalent, dealing with any syntax or formatting errors in Python as they arise. To start up Scratch, for this project work see Workbook 6: Plug in to Scratch.

MISSION 2 REPETITION

To recap from unplugged programming earlier: quite often, the idea of repetition is used to exploit the occurrence of pattern in the code, which in turn reflects the pattern on the screen. In program(1) below, there is a ***repetition*** of pattern in the code to draw a square. We emphasised the importance of a RETURN program. (It is possible to draw a square without the last `lt` in program(1). It wouldn't be a RETURN program then. Why not?) If your code looked something like this in UPL:

```
fd2 lt    fd2 lt    fd2 lt    fd2 lt.....program(1)
```

(we have introduced some extra spacing to emphasize the pattern) it is evident that 2 instructions `fd2 lt` are repeated 4 times corresponding to the four sides of the square. We first place the instructions in a vertical line to accentuate the repetition

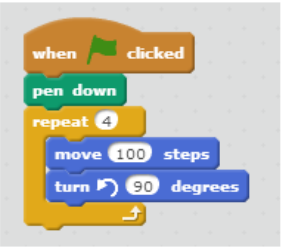
```
fd2 lt
fd2 lt
fd2 lt
fd2 lt.....program(1b)
```

We then introduced the **repeat** control structure in UPL as

```
repeat 4 [fd2 lt].....program(1c)
```

The equivalent repeat structure is handled in Scratch by the repeat structure ' orange C, or Jaws', and in Python by a 'for-loop', see Figure 5 below.

Scratch Python UP(Logo)




```
from turtle import *
#
for i in range(4):
    fd(100)
    lt(90)
```

repeat 4[fd2 lt]

Figure 5. *Repetition* in Scratch, Python and UP(Logo)

The result is shorter, simplified programs in both Scratch and Python, which achieve the same effect as the programs in Figure 4.

- Note that in drawing the square, the sprite returns to its original position and direction and is therefore a RETURN program. In addition, you can see the sprite turn a backward somersault as it goes round the square thereby executing a SPIN(360). (We can see this more demonstrably by introducing a wait instruction in the loop to slow the sprite down. See Figure 6).

EXPERIMENTATION, INDUCTION AND ENQUIRY-BASED LEARNING - WRITTEN AND ORAL

- Make a note of how you would/would not use questioning to challenge and support your class in Activity 1 and the rest of this section.

EXPERIMENTING WITH SCRATCH/PYTHON PROGRAMS

Set up and run the Scratch or Python program in Figure 5.

1. When you have run it successfully to draw a square, ***experiment*** with the angle of turn in the program in Figure 5, by substituting an angle 45 for the 90 degrees, and vary the repeat value (4) in the program to take values 5,6, 7… , to see if you can return the sprite to its starting point O, (and facing in the starting direction), thus completing a RETURN program. If the return is completed, the program also completes a SPIN(360) and generates a simple regular polygon. You may need to reduce the length of the side of the polygon to 50 pixels(steps) in order keep the diagram on the screen. What is the repeat value that works for 45 degrees? How many sides has the polygon got?
2. *Now ***experiment*** with a different angle of turn in the program of Figure 5.
   a. 60 degrees
   b. 30 degrees
   c. 40 degrees
   d. 36 degrees

And vary the repeat value from 4, 5 … to see if you can complete a polygon by returning to O and facing in the original direction. (A RETURN program)

3. *What is the repeat value, and the number of sides of the polygon in 1a. – 1d.?
4. **What happens if we choose 63 degrees? Can you explain why?

MISSION 3 PROGRAMMING FOR POLYGONS AND CIRCLES

1. *Complete the first 3 columns of Table 1.
2. *What shape is the last entry in the table?

Table 1

Turning Angle	Repeat value	Regular Shape polygon	Shorthand for repeat loop	Length of side
90	4	Square	(4, 360/4)	100
45	6, 7…?		(8, 360/8)	50
60	?			50
30	?			40
36				40
63		None	N/A	50
?	5	pentagon	(5, 360/5)	75
?	10			?
?	7	heptagon		50
4	90	?		5

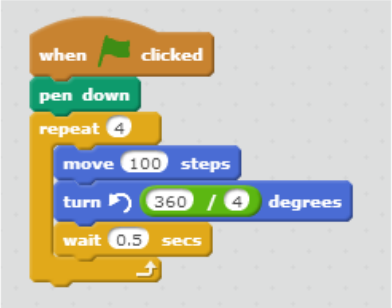
MISSION 4 SHORTHAND NOTATION FOR SPIN(360) ROTATIONS

Note that in each successful polygon you produce, the sprite has executed a SPIN(360). It enables us, with the results from the Table 1 to use **`(4, 360/4)`** as a shorthand form for the **`repeat`** loop which produces a square. Note the symmetry of 4 and 360/4. {It is assumed that 360/4 means a left turn through (360/4) i.e. **`lt(360/4)`** unless stated otherwise}. This special category of repeat loops we will use later in rotational symmetry. This one is the 'square' loop; there will be others. The program to draw a square of side length 100 pixels in shorthand notation would be

`(4, fd(100) 360/4)`.....**`program(1s)`**

The program is easily transcribed into Scratch and Python. See Figure 6. And we will employ this linear shorthand notation, where appropriate, for special repeat loops and programs and extend the notation to describe translations when we are dealing with pattern generation later.

Scratch **Python** **Shorthand Program**




```
from turtle import *
#
for count in range(4):
    fd(100)
    lt(360/4)
```

(4, fd(100), 360/4)

Figure 6. Scratch, Python, and Shorthand Programs for the repeat loop for a square with side length 100 pixels

* SHORTHAND FORMS FOR REPEAT LOOPS AND PROGRAMS

1. * (***generalising***). Fill in shorthand forms for the other shapes in Table 1. (Use the format 360/5 instead of 72, for example, in programming your repeat loop in Scratch and Python whenever it fits). The shorthand form comes in handy for the heptagon. Why? (Let the computer do the calculation!)

2. *Give an example of a shorthand form for a circle.
3. *** What is the shorthand form for a polygon with n sides***?(generalising)***

MISSION 5: 90-SIDED POLYGON: A CIRCLE

The last entry in Table 1: if you make the repeat value 90, the turning angle 4 degrees and the length of side 5 pixels, in shorthand our program would be

```
SPIN(360):   (90, fd(5), 360/90).....program(2s)
```

And you get a 90-sided regular polygon. What does it look like?

As long as the shorthand is of this SPIN(360) form, the shape will be a simple regular polygon. In this case it consists of 90 small (5 pixel) straight sides (just visible), and after each one the sprite/turtle turns through the same small angle 360/90 degrees until it returns to its starting position and direction. Just as it did for all the polygons with a relatively small number of sides.

The shorthand form `(90, fd(5), 360/90)` for this well-known shape, the circle, we can use in our pattern generation later.

1. ***Experiment** with the Scratch/Python program for program(2s). What happens if you change fd(5) to fd(3) or fd(6) or fd(25)?
2. *Describe physically what happens -- Crack the Code -- with program(3s). Is it a SPIN(360) program?

   ```
   (180, fd(3), 360/180).....program(3s)
   ```

3. *And again for program(4s)?

   ```
   (360, fd(2), 360/360).....program(4s)
   ```

4. ****If you have a simple polygon, say an octagon, drawn by a sprite/turtle starting at O(0, 0):

   ```
   (8, fd(50), 360/8).....program(5s)
   ```

   How would you locate the centre C(x,y) of the octagon, and the length of CO? We have already found the angles at the centre of a polygon. But you need some basic trigonometry to solve this one.
5. *****A circle on the screen! If the starting point of the sprite/turtle is O(0, 0). What is the radius of the circle? And the coordinates of its centre C(x, y)? Use the fact that the circle is a polygon, and Activity 5.4 to help you.
6. ***** Using results of Activity 5.5 above, or otherwise, write a program to draw a simple regular hexagon (polygon with 6 sides) and the circle that goes through the 6 vertices of the polygon.
7. *****Do the same for a heptagon (7-sided polygon).

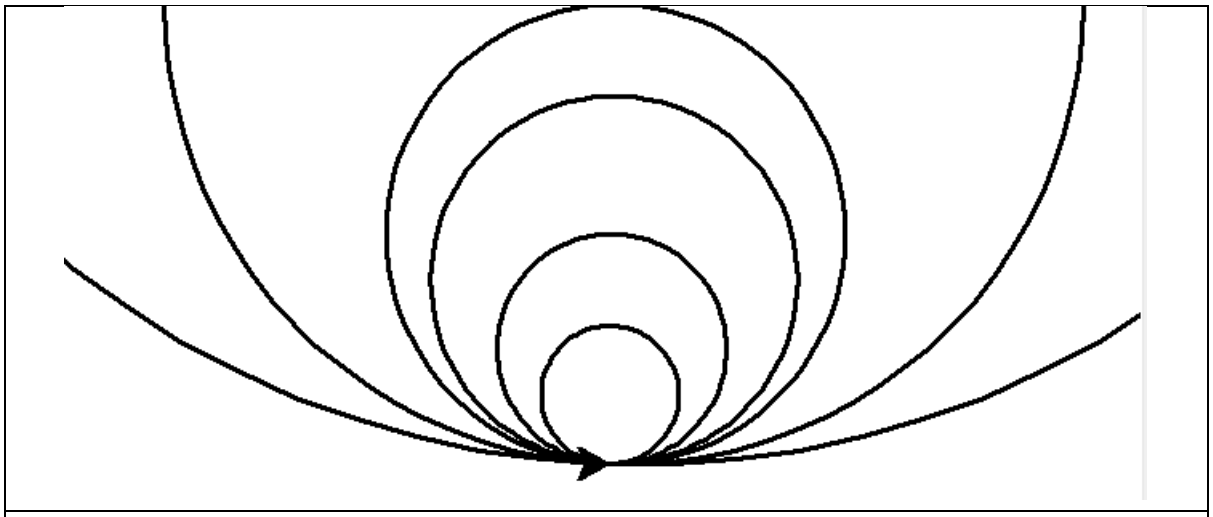


Figure 7. `(90 fd(3)360/90)`.....`program(2s)` *with fd( ) taking arguments 3, 5, 8, 10, 20, 40 pixels*

As you can see in Figure 7, even the 90 sided polygon with a straight edge of 40 pixels (0.5 inches) still manages to look like a circle.

Program in Scratch/Python?

MISSION 6: STARLIGHT ENQUIRY

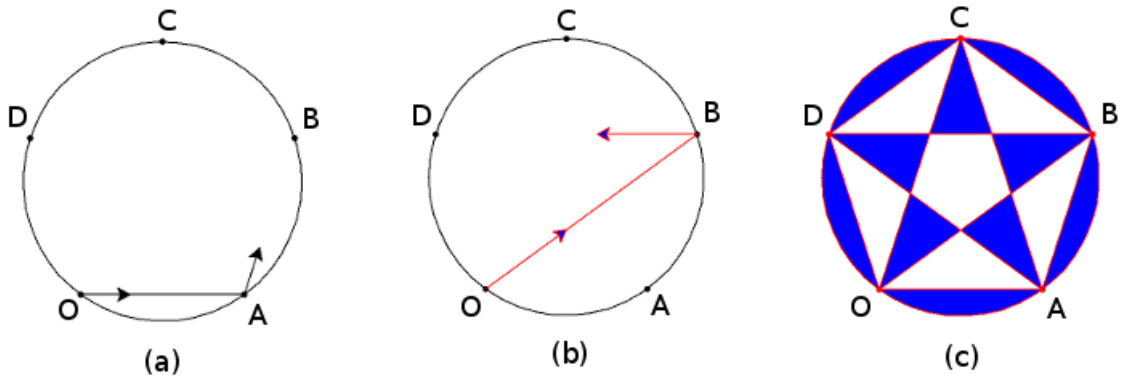


Figure 8. ***action geometry:*** **for the pentagon and pentagram**

In Workbook 6, unplugged, we experimented with joining up the points in Figure 8 (a), (b). In 8(a) we joined up OABCO and turned the corner at O to face A again. In the process we turned through 360 degrees and completed a SPIN(360) and the result was a regular pentagon. In Figure 6, we changed the program in Scratch (or Python) until we reached a repeat value of 5 and an angle turning value of 360/5. The shorthand form of the program for the pentagon is:

`(5, fd(100), 360/5)`.....`program(6s)`

- How would you write **`program(6s)`** in Scratch or Python? See Figure 6 for the programs for the square.
- Now the tricky bit. A few questions! Let's try some reasoning from symmetry and induction?
  1. In the pentagram (5-star), how many lines do we draw: OB, BD … ?
  2. Are they all of equal length?
  3. When we turn, do we turn through the same angle?
  4. Is this angle bigger/more than the angle of turn for the pentagon? (360/5)
  5. How many times do we turn through an angle in the course of drawing the pentagram? (Don't forget to count the turn at the finish by facing the starting direction OB)
  6. How much do we SPIN around as we go through the drawing?

For example, when the moon goes around the earth it SPINS ON ITSELF 360 degrees --- which is why on earth we only see one side of the moon. On the other hand the earth on its journey around the sun SPINS around 365 and a quarter times on its axis. In our journey round the pentagram we definitely SPIN more than 360 degrees but how much more?

7. If we find out how much we spin in total, how do we find out what angle we turn through at each vertex?

Let's act it out on the pet/robot walk? Or try the moon walk around the earth first. Or watch the program starpower.py. And move on to Workbook 12.