UCL/CAS Training for Teachers

Algorithms and Programming Module 1

`1. forward fd1`	↑
`2. left turn lt`	←
`3. right turn rt`	→

“Would you tell me, please, which way I ought to go from here?"
"That depends a good deal on where you want to get to."
"I don't much care where –"
"Then it doesn't matter which way you go.”
— Lewis Carroll, Alice in Wonderland.

UNPLUGGED PROGRAMMING
A RATIONALE

- Addressed to Teachers

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

INTRODUCTION

- This is the first part of an exemplar short course for Teachers developed from a CAS Training Course for Master Teachers delivered in the Department of Computer Science at University College London September 2014 - July 2015.
- **Workbooks are for teachers; Missions and activities are for pupils (and teachers!).**
- **if you know your left from your right and can 'Walk the Talk' or 'Chalk the Talk', you can start to solve drawing problems by programming.**

OUR AIMS

1. To introduce pupils to the art of programming before sitting at a computer.
2. To find a way to support learning about programming, in preparation for and prior to the other difficulties faced by teachers and pupils when tackling the pupil-computer interface.
3. The whole unplugged programming course, Workbooks 1-5, is one of transition to programming at the screen, assuming no previous knowledge of Algorithms and Programming, starting with the same three instructions to drive a pet/robot/sprite/turtle which UPL, Scratch, Logo and Python all employ.
4. 'Unplugged' programming is a way to learn how to program, away from the computer. This part of the course is designed to precede, and/or scaffold alongside, learning to program at the computer specifically in Scratch, Python, Logo and Coffeescript, using a single sprite or turtle.
5. To test out how different strategies in ***Computer Science pedagogy*** may encourage ***creativity, experimentation, discovery, independence, perseverance, learning from mistakes…*** and pinpoint and encourage ***Computational Thinking;*** thus making learning effective, exciting and enjoyable. (Well, maybe not all at the same time!)

   There are many thought processes which can be considered to be ***Computational Thinking*** (see http://www.ispython.com/computational-thinking/); we will use a simple but comprehensive subset: the following five elements as guidelines in our approach to cover the ground.
6. Calling on Computational thinking to devise this Course in programming and Algorithms, we have set up a simple model to apply the 5 computational thinking elements to solving problems using programming. See 'Computational Thinking Applied to Programming' below. We are writing/building code which explicitly illustrates how all five elements underpin learning to program. Both the *content* and *the process* are important to understanding. In Seymour Papert's words:

   "making and making sense; hands on and minds in".

7. Using an unplugged simplified approach to start, frees us to learn about the art and principles of programming and how to employ some of the fundamental control structures in the design of our programs, without having to engage with the demands of the computer environment of a programming language until we are ready. When we are

ready, the programs we have written serve as an *already coded* exact pseudo-code and are mapped directly to Scratch 2.0 instructions in a program. (or to Python 3, Coffeescript or Logo if we so choose).
We take a project-driven approach to learning in the cross-curricula area of geometric shapes in our first pathway, using pattern recognition, symmetry in geometry, and a minimal need for correct programming language syntax, especially in the oral, and paper-and-pencil arenas of unplugged programming.

❖ **If you want to skip this rationale of Workbooks and how to use them, until later, and get straight in on the action, go to Workbook 1.**

WORKBOOKS, MISSIONS, ACTIVITIES AND EXPLANATIONS -- ROLE OF THE TEACHER

❖ Workbooks are designed for teachers, and are designed to be stand-alone booklets. In sequence, they constitute an integrated basic course in Algorithms and Programming underpinned by computational thinking, and introducing teaching and learning methods in computing, using projects from cross-curricula topics.

❖ The sections in the body of the text, prefaced by the diamond bullet point, are specifically addressed to Teachers.

Role of the Teacher. The role of the teacher is vital in making the missions successful. This may include providing input at the beginning, to introduce, demonstrate (particularly in unplugged 'Walk the talk') and amplify the explanation for a mission or a specific activity as work proceeds.
Missions, activities, explanations included in the workbooks may be extracted from workbooks and are aimed at learners in the range KS2-KS4, but may require modification to make them suitable for a particular group.
The missions are graded from zero * to 5*. Learners should be able to complete up to 1* in the course of a first reading of the workbooks and missions. Missions and activities marked ** to ***** are increasingly more difficult in order to challenge different levels of understanding and experience.

SUMMARY OF THE UNPLUGGED COURSE

In these Workbooks 1-5 on unplugged programming, we introduce the program control structures: ***sequence, function,*** and ***repetition*** in order to tackle the projects presented.
Workbooks 6 – 8 are introductions to Scratch 2.0, Python 3 and Coffeescript respectively and are designed to enable students to undertake the projects in Workbooks 1-5 by programming at the screen in the chosen programming language.

COMPUTATIONAL THINKING: THE UNDERPINNING

1. ***Algorithmic Thinking –*** thinking through the strategy and steps required to solve a problem(AT)

2. ***Decomposition* –** breaking a problem down into its smaller components(D)
3. ***Abstraction* –** reducing and hiding complexity by using or creating tools/models
4. ***Generalization* --** adapting solutions so that they can be used to solve a wider range of problems(G)
5. ***Evaluation* –** assessing whether a program or technique works correctly, efficiently and by other criteria e.g. maintains good programming practice.(E)

COMPUTATIONAL THINKING APPLIED TO PROGRAMMING

❖ How the five elements of ***Computational Thinking*** (ADAGE) underpin the programming process will be addressed in more detail throughout the course.

A PROJECT-DRIVEN APPROACH

The four tasks to be undertaken are:

- project 1: build programs to draw a (restricted) alphabet of capital letters and write simple words. (Workbooks 3,4)
- project 2: build a program to draw a 3 x 3 array of squares, (Workbook 4). These tasks are simplifications or first steps in solving real practical drawing problems.
- project 3: ***generalise*** -- build a program to draw a board for a game with a square array of squares, say 8 x 8 for draughts/chess, 9 x 9 for Sudoku, 10 x 10 for snakes and ladders, 13 x 13 for crosswords…
- project 4: ***generalise*** – build a program to draw a rectangular array of squares/rectangles for different purposes e.g for aqado in the AQA GCSE controlled assessment 2015

❖ The first problem (project 1) arose in an attempt to design iconic first letters for naming pupils' work.
The second (project 2) as a result of wanting to play Sudoku more effectively offline, by simply printing a 9 x 9 array of squares larger than that offered in newspapers and books. This project turned out to have practical use in providing an array of squares as a background to illustrate diagrams in this Workbook for cracking the code by drawing on squared paper, and as a starter for the board in the AQA controlled assessment for GCSE 2015.
In our approach to these tasks we are already encountering aspects of ***computational thinking.*** Reducing (and hiding) the complexity of a task, by devising a simplified model of a problem, breaking it down into distinct components, is a common approach to solving problems and embodies thinking in terms of ***abstraction*** and ***decomposition***. ***Generalising*** (project 3 and 4) in programming terms, is to restructure a program, if necessary, usually reshaping it as a function with parameters, so that it provides a solution to a wider range of problems than the original program.

A FRUITFUL EXAMPLE

- This section of the Course on unplugged programming has taken shape, and is offered as a fruitful example because different solutions to the projects set, arrived at by teachers and pupils,
    - illustrate ***Computational Thinking*** in detail,
    - lend themselves to harnessing three of the five control structures of programming: ***sequence, functions and repetition,***
    - enable learning and practice of ***algorithmic thinking*** and ***fundamentals of programming***
    - do without the added distracting complication of handling the pupil-computer interface at the same time
    - make it an effective launch-pad for transition into Scratch 2.0, Python 3, Logo and Coffeescript either 'unplugged' or at the Computer interface.

MAKING THE COURSE YOUR OWN

- It is intended that this course (in particular the missions in each workbook) can be fashioned by Teachers to suit their own teaching style and adapted to suit their pupils in the KS2-KS3 range. Introducing Scratch or Python, unplugged, or at the screen, can take place at any point during or after the unplugged elements of this course, when the teacher judges it to be right.

CRACKING THE CODE

- We use our spoken or written program to get an immediate response from our imaginary pet/robot in very much the same way that a Scratch or Python program can drive a sprite or turtle portrayed on the screen. Pupils can work in pairs using UPL (Unplugged Programming Language) in a number of ways. Using the literacy precept "read before you write"; we start with:
    1. ***crack the Code (decomposition).*** Break down the code of a program into components by using 2 or 3 below in order to get an idea of the whole program.
    2. 'Walk the talk': one pupil 'talks' the instructions, the other 'walks' the corresponding figure
    3. 'Draw the talk': one pupil 'talks' or 'reads' the instructions, the other 'draws' the figure on squared paper provided

Later, we use a combination of 2, 3 interactively to construct a program and write it on paper.

A SOFTWARE TOOL : UP1.PY --- A KEYPRESS ALTERNATIVE

As a reinforcement to learning, we can also make use of up1.py, a software tool, which complements the unplugged approach, (download from ispython.com/software/) used to demonstrate the motion of the pet/robot or to 'crack the code' by pressing the arrow keys corresponding to the instructions in the program. It's a way of getting used to the technology that drives a pet/robot/sprite/turtle without having to build a program in Scratch or write a Python program. This unplugged course in programming serves equally well as a precursor for, or as part of the transition process to programming at the computer. The pet/robot and its tools matching up exactly with the sprite of

Scratch 2.0 and the turtle shared by Python, Logo and Coffeescript. You may decide to start up with Scratch 2.0 or Python 3 at any point, but it will save building fairly lengthy programs, if you introduce ***sequence, repetition*** and ***functions*** unplugged, before moving to the screen.

WHERE ARE WE GOING?

- In the pursuit of symmetrical geometry, we need to be on the lookout for patterns of repeated code that help to determine the physical components of the drawing ***(decomposition)*** and to make use of the ***repeat*** control structure to simplify our programs.
  We start with a few simple instructions/functions, and the idea of a RETURN program to get our class ***thinking algorithmically*** and used to programming unplugged. The instructions we use in UPL are all (system) functions in Scratch and Python, standing for quite complex machine code, which is hidden away ***(abstraction).*** We develop and use parameters in functions ***(generalisation),*** right from the start in both the system instruction **`forward1 (fd1)`** and later when we build our own user functions (tools).

- With the simple idea of a RETURN program, we employ ideas of pattern and symmetry to introduce 'action geometry', again unplugged, to develop a familiarity with geometrical shapes before moving online with our programming.
  When we harness the tools of the programming language, including the ones we create ourselves -- through functions -- we are able to explore and develop the exciting area of polygons, stars, spirals, regular and irregular shapes, and motion in the natural world. We are then able to design our own unique coloured patterns in a way that would not be possible without programming a computer.

PROGRAMMING IN SCRATCH 2.0 AND/OR PYTHON 3

We are aiming by the end of Module 1 to build programs to produce our own unique patterns through symmetric rotation and translation and reflection of shapes. See Figure 1 for examples.

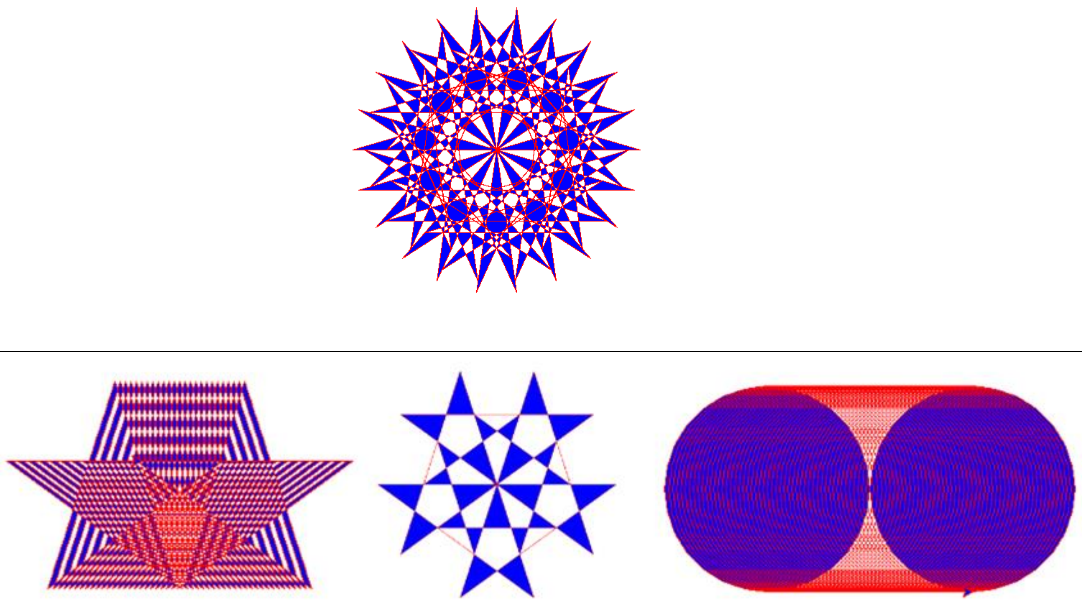

Figure 1. "legs-eleven", "shooting star", "starring a star" and "rock'n rolling a circle".