Tao of Computational Thinking in Programming

(A Session for Teachers)

Rationale

Our Core Approach: Project-based and Problem-Solving Oriented

1 Develop an ongoing pedagogy in computing (ispython.com/pedagogy) guided by: current research findings, classroom experience, existing expertise in other disciplines. And remain open to further action-research in the classroom.
2 Select a cross-curricula problem domain, interesting and fun for pupils to explore, and fertile in problems to solve
3 Choose a basic appropriate programming toolbox in which to design and create user-tools for problem solving in this domain.
4 Implement the simple, but general, paradigm for problem solving in computing:
program(toolbox + user-created tools) = solutions
5 Harness computational thinking in achieving a human solution in pursuit of a program solution.

Example 1: Teaching at the KS2/KS3 Interface for beginners in Python 3

Here we are embarking on a path to teach beginners -- at least in Python 3 – computational thinking in programming. The pedagogy we employ, choice of problem domain, examples, programming toolbox, and computational thinking is geared to delivering the start of a pathway, which is suitable for teaching beginners in the KS2/KS3 spectrum.

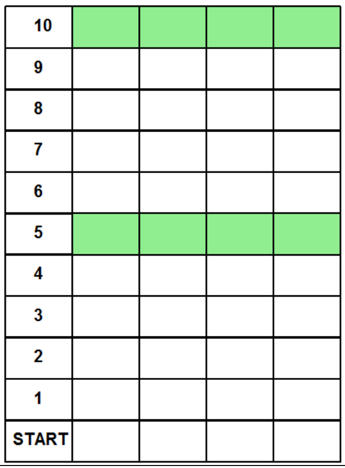
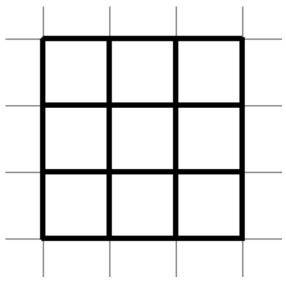
1. Our pedagogy, which we try to make explicit, in the process of delivery for teachers, for this problem includes:
    - Investigating human solutions with pencil and paper
    - Unplugged programming: human robot ‘Walking the Talk’ solutions
    - Enquiry-led learning
    - Read code before you write code
    - Opportunities for experimenting and learning by doing
    - Guidance in graded problems to promote computational thinking in programming
    - Looking at different solutions (generalising and evaluating)
    - Transferring our drawing/walking solutions and our robot to the screen with

- ❖ ISPY (red arrow) – push-button scaffolded programming for beginners
- ❖ Or Scratch (sprite)
- ❖ Or Python (turtle)

2. We have chosen a 2-D rectilinear grid as our first problem domain. The advantages of this and the follow-up domain are:

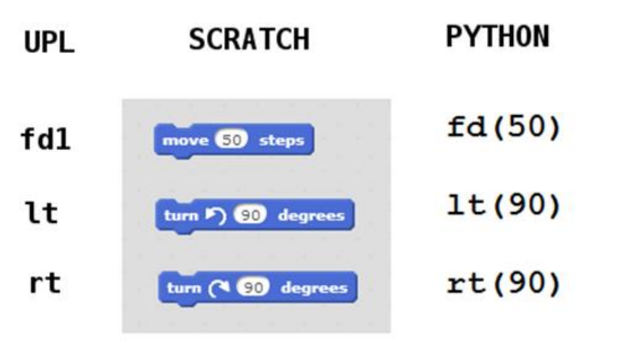
- that the developing programming produces a visual graphical response. (Enhanced to be an immediate interactive response with ISPY, see $5).
- Symmetry and pattern in drawings can be used in computational thinking to link symmetry and pattern in the code via the programming control structures of *repetition and functions.*

The specific problem we start with: build a program to draw the Aqado game board in Figure 1. (part of an AQA GCSE 2015 controlled assessment assignment).

Figure 1: AQADO board and our simplified model.	
	
Aquado Board	A Simplified 3x3 Grid

3. We use the 3 basic motion instructions for line drawing with Seymour Papert's turtle or the Scratch sprite together with 3 programming control structures: *sequence, repetition and functions* as our programming toolbox.

Figure 2. Toolbox Instructions



UPL	SCRATCH	PYTHON
fd1	move 50 steps	fd(50)
lt	turn ↶ 90 degrees	lt(90)
rt	turn ↷ 90 degrees	rt(90)

Basic robot instructions in unplugged programming language (UPL) with corresponding instructions in Scratch and Python

Let's Go: The Session in Practice for Teachers

Computational Thinking for a Human Solution

a) Show Aqado and reduced model. Figure 1. We are going to look at solutions to a simpler problem. (Decomposition: Simplifying with a model of bare essentials of a line drawing of a simple 3x3 grid of squares --- hiding a lot of the detail of the problem. (Abstraction)
b) Give out paper and pencil. Ask each person just to draw the 3x3 grid model.
c) Ask them to describe how they did it. (Their algorithm for a human solution).
d) Draw their attention to the symmetry in the grid. Ask them to draw the 3x3 grid again 'systematically' as if they might be asked to do a 4x4 or a 10x10 grid…(Generalisation)
e) In small groups ask them for 3 or 4 really different solutions.
f) Give them a hint: What do you see in the model as potential building blocks/templates to repeat in order to draw the model 'systematically'.
g) Ask them to choose one solution to program which seems easiest -- maybe to generalise. (Evaluation)

Our Unplugged Programming Tool box:

'Walking the Talk' with the 3 instructions, in UPL or (unplugged programming language) to drive the robot. These instructions are common to Scratch, Python, (and other languages which implement the turtle). Together with the universal programming control structures of *sequence, repetition and functions*.

Computational Thinking for a Computer Solution

We could get the computer to mimic the human solution exactly? Or we can try to harness directly the properties and capacities of our toolbox, by experimenting with the instructions and introducing verbally the programming control structures *sequence, repetition and functions.*

Find out if everybody knows what characterises a robot/sprite/turtle/ in the plane when we want it to move? (Enquiry-led learning). Answer: (position, direction)

Ask everybody (who agrees to) to stand up and be a robot. Get the participants to inhabit the robot. A robot faces to the right as the handler views them, and chooses enough space in front and to their left to allow at least one pace forward or to the left. (If you think of the floor as a page on which they leave a trail (a drawing), they are standing at the bottom of the page and the top of the page is to their left).

Figure 3. The system commands for the robot.

`forward1, fd1`	↑
`left turn, lt`	←
`right turn, rt`	→

"Begin at the beginning," the King said, very gravely,

"and go on till you come to the end: then stop."

--- Lewis Carroll, Alice's Adventures in Wonderland &Through the Looking-Glass

Functions/Procedures/Subroutines

The instructions **`left turn or shortly: lt;  right turn: rt`** are functions. They stand for fairly complex code (machine code) which is hidden (abstraction). (There is no parameter for these instructions the turn is through 90 degrees, understood – this changes later when we move off the grid to an extended problem area – the whole 2-D plane).

`forward4, fd4` is move forward 4 paces and draw as you go, --- again a function this time with a parameter --- the number of paces -- 4.

Get the robots moving --- give the commands:

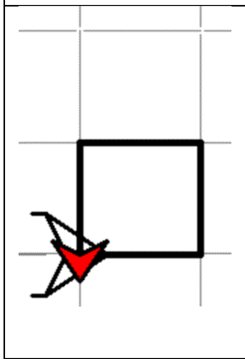
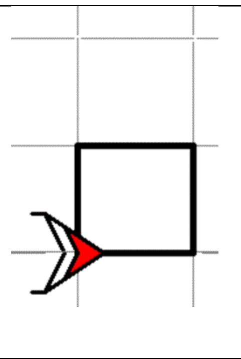
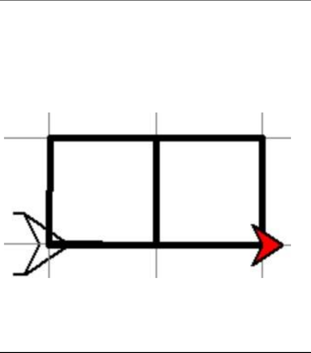
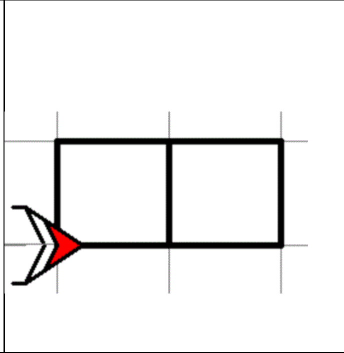
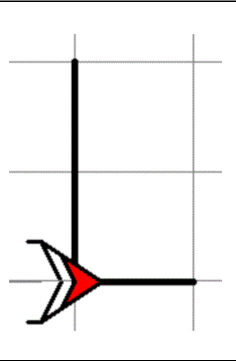
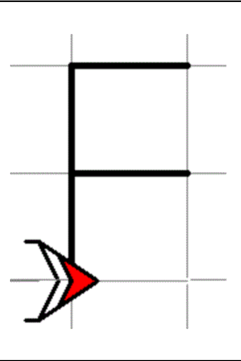
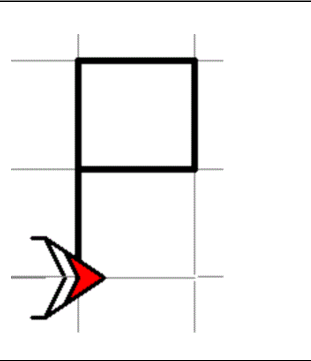
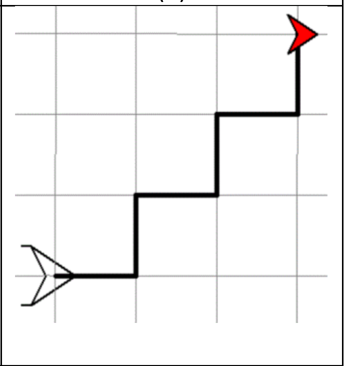
```
fd1 lt
fd1 lt
fd1 lt
fd1 lt............................(1)
```

with hesitation after each line.

Ask them to describe what happened.

Looking for: Repetition -- Symmetry in the shape drawn and in the code (achieved with the last `lt` command). And an example of a return program. (One where the robot returns to the starting position and starting direction)

Figure 4. Repetition and Return programs;

			
(a)	(b)	(c)	(d)
			
(e)	(f)	(g)	(h)

```
fd1 lt
fd1 lt
fd1 lt
fd1        ..............................(2)
```

What is the difference between (1) and (2)? And between Figure 4(a) and 4(b)? (Both draw a square but…)

Programming languages contain **`repeat`** structures. So we ask the robots to recognise their 'wired in' **`repeat`** ability.

Give the command (with intonation to insert the brackets!):

```
repeat 4[fd1 lt]..................................(3)
```

and watch them obey.

Then call upon their 'wired in' understanding of functions and name the code **`square1 or sq1`** (for short when we go to the screen), a function with a parameter 'pace', which could be 2, 3 … paces along a side of a grid, in a similar vein to the function forward1, 2, 3…

paces. (In unplugged we have to define a user function by the code we name but, helpfully, not with a formal syntax definition for the function or its parameter).

{** this notation we introduce in later sessions

More formally, we use '→' as our symbol for 'defining' a function with a parameter **`p`** in writing:

```
squarep or sqp → repeat 4[fdp lt]..................................(4)
```

Where **`p`** can take values (arguments) **`1, 2, 3`**...

** }

Give the command: **`square2.`** (See Figure 5 below for same command in ISPY)

(For robot practice, divide into pairs, handler and robot, and the handler uses instructions to the robot partner to draw some capital letters: e.g. I, L, F, E, T, P or combinations of square and write down the corresponding return programs. The handler can, in each case,

1. build the code for the letter from the three instructions,
2. by taking note of symmetry in the drawing, use the repeat control structure in the program,
3. by identifying building blocks for the drawing, include user-defined functions in the program.

Examples: For Figure 4(e) L the code might be:

```
fd1 lt lt fd1 rt fd2..........................(5)
```

or as a return program

```
fd1 lt lt fd1 rt fd2   lt lt fd2 lt...............................(6)
```

and for Figure 4(f) F the code might be

```
lt repeat 2[fd1 rt fd1 lt lt fd1 rt]...... . ..................(7)
```

or as a return program

```
lt repeat 2[fd1 rt fd1 lt lt fd1 rt] lt lt fd1 lt fd2 lt.........(8)
```

and for Figure 4(g) P the code might be

```
lt fd1 rt sq1..........................................................................................(9)
```

or as a return program

```
lt fd1 rt sq1   rt fd1 lt...........................................................(10)
```

The above code introduces and illustrates our programming paradigm

program(toolbox + user-created tools) = solutions

by making use of:

- the toolbox instructions **`fd, lt and rt`**

- `repeat programming control structure`
- `functions:` user-defined function `sq`

(Each letter could then be named as a user defined function).

Try the remainder of the examples in Figure 4, identifying symmetry and building blocks in the drawings to use repeat and user-defined functions in the programs.

We can define two more building blocks as user-defined functions:

a (return) line and a (return) rectangle to help us with alternative solutions for the 3x3 grid:

```
fetchp (fep) → fdp lt lt fdp lt lt..........................................(11)
rectanglep (rep) → repeat 2 [fdp lt fd1 lt].................(12)
```

(a rectangle is here defined to have a variable length parameter: **p** paces, and a fixed breadth of 1 pace).

For letter E, the code might be

```
fe1 repeat 2[lt fd1 rt fe1]......................................................(13)
```

These are more examples of user-defined functions --- the building blocks identified in the discussed solutions. This illustrates the problem solving paradigm: Choose your toolbox for the problem, and create more tools (functions) to represent the physical building blocks for your drawings as you need them.

Results: Teacher devised solutions of 3x3 model written after 'Walking the talk':

Solution 1:	`fe3`
	`repeat 3[lt fd1 rt fe3]`
	`rt`
	`repeat 3[lt fd1 rt fe3]`

Solution 2:	`sq3`
	`repeat 2[fd1 lt fe3 rt]`
	`fd1 lt`
	`repeat 2[fd1 lt fe3 rt]`

Solution 3:	`repeat 3[re3 lt fd1 rt]`
	`lt fd1 lt lt`
	`repeat 3[re3 lt fd1 rt]`

Solution 4:	`repeat 3[re3 lt fd1 rt]`
	`lt fd1 rt fd1 rt`
	`re3`

Solution 5:	`rowsq → repeat 3[sq1 fd1]`
	`repeat 4[rowsq lt]`

Solution 5 defines a function for a row of squares **`rowsq`** using an existing user-function **`sq`**. Clever and works for 3x3. Not easily generalizable. How would you adapt **`rowsq`** to write a program that was easier to generalise? (Return program?)(Plenty more solutions! Lots of discussion, but all teachers agreed they could build the programs in upl, Scratch or Python.)

Graded Challenges:

Why would you draw a grid? Board Games challenges in UPL (ISPY) include: 3x3 Noughts and Crosses; 4x4, 5x5, 6x6 Boggle (a spelling game on the grid); 6x6 Conway's Game of Life; 7x7 Othello; 8x8 Chess and Draughts; 9x9 Sudoku; 10x10 Snakes and Ladders; 11x5 Aqado; 13x13 crosswords. Other contenders: Ludo, Monopoly, Scrabble, Go, Connect 4. Project 2 Pixels and Colour backgrounds, National flags, Bar scarves, Pixel pictures …

Further Challenges:

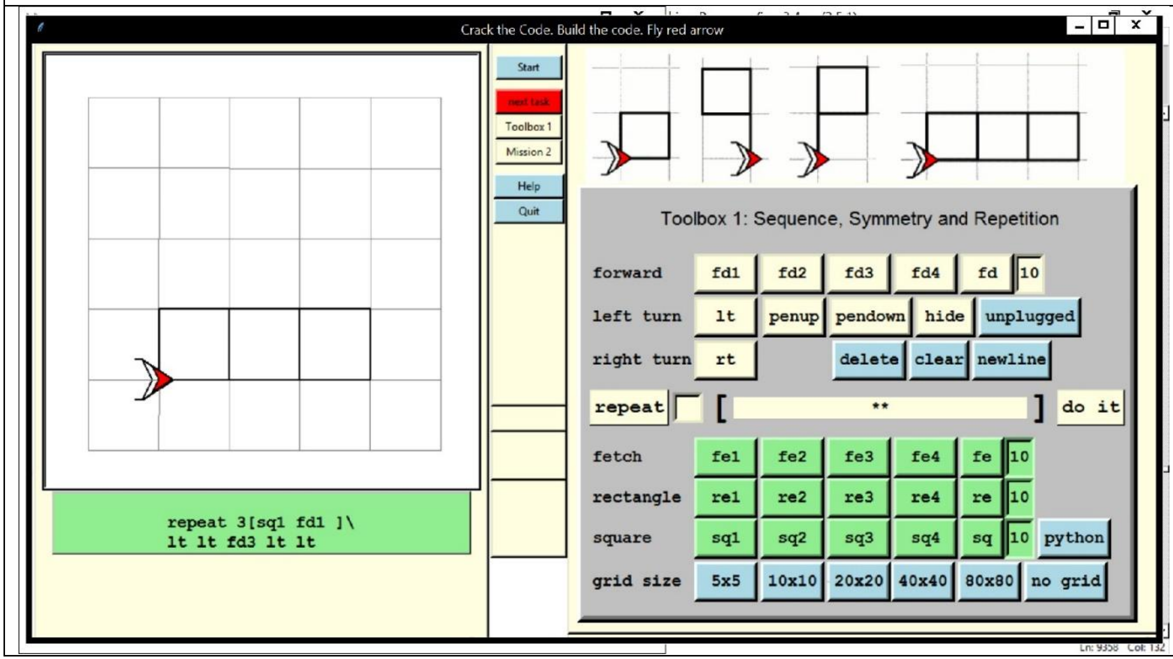
move on to challenges in the open 2-D plane: polygons, stars, circles, curves, spirals, patterns, linear transformations: translation, rotation, reflection; action geometry, transitional geometry. Solar system and planetary motion.

Move on to graded missions in unplugged programming in ISPY and Push-Python (when live!):

ispython.com/ispy

All these problems are programming tasks and could be solved in Scratch 2, or Python 3, but ISPY has an integrated graded problem set and is incrementally interactive with no syntax errors, and no 'save and run' procedure and a delete button for semantic or logical errors, ideal as a transition phase and for learning Python 3. See Figure 5.

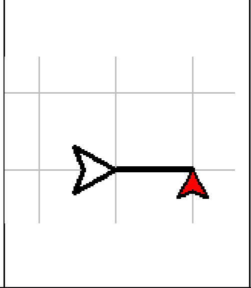
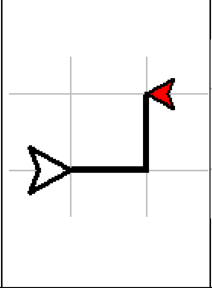
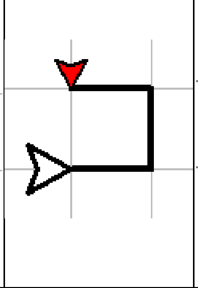
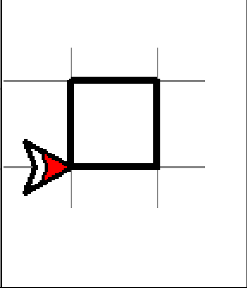
Figure 5. Snapshot of ISPY in action.
Push-button code for drawing the example of three squares as a Return program. The code is recorded instruction by instruction as the push-button program is built. The push-buttons lower down(green) represent the user-defined tools (functions) for project 1.



```
repeat 3[sq1 fd1 ]\
lt lt fd3 lt lt
```

Reference: Sequence, Repetition and Functions in Unplugged Programming

Figure 6. Symmetry in the code and symmetry in the 'action geometry' of drawing

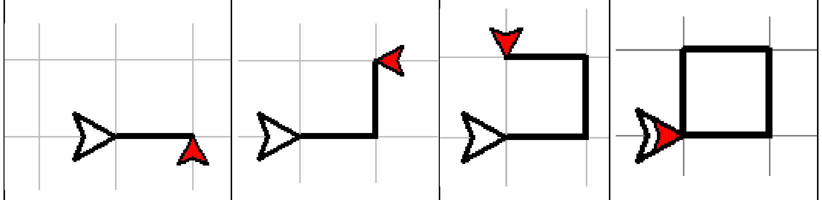
`fd1 lt`
`fd1 lt`
`fd1 lt`
`fd1 lt`
 	
`repeat 4[fd1 lt}`
				

Code for unit square	Building block: the unit square (Return program) Representation of the four symmetric repeats of **`[fd1 lt]`**			

Figure 7. Defining the square function more formally in UPL

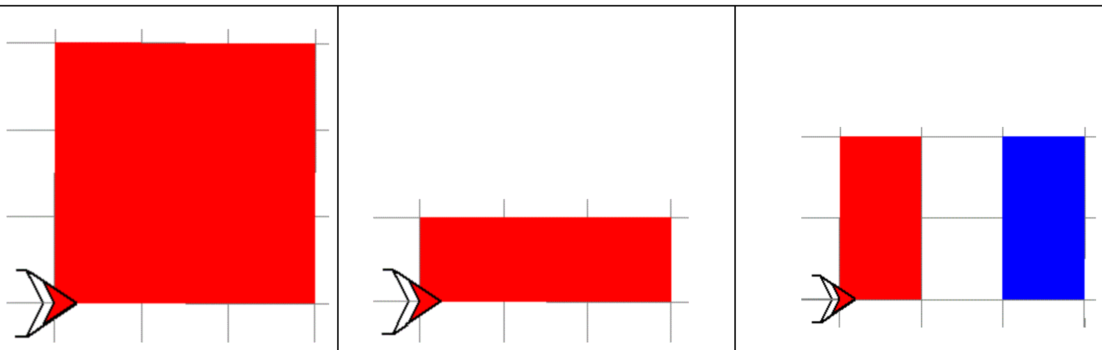
```
squarep or
sqp →    fdp lt
         fdp lt
         fdp lt
         fdp lt

→ repeat 4[fdp lt}
```



Next Challenge: Pixels and Colouring a background on a   grid in ISPY (Toolbox 2)

Figure 8. Colouring examples

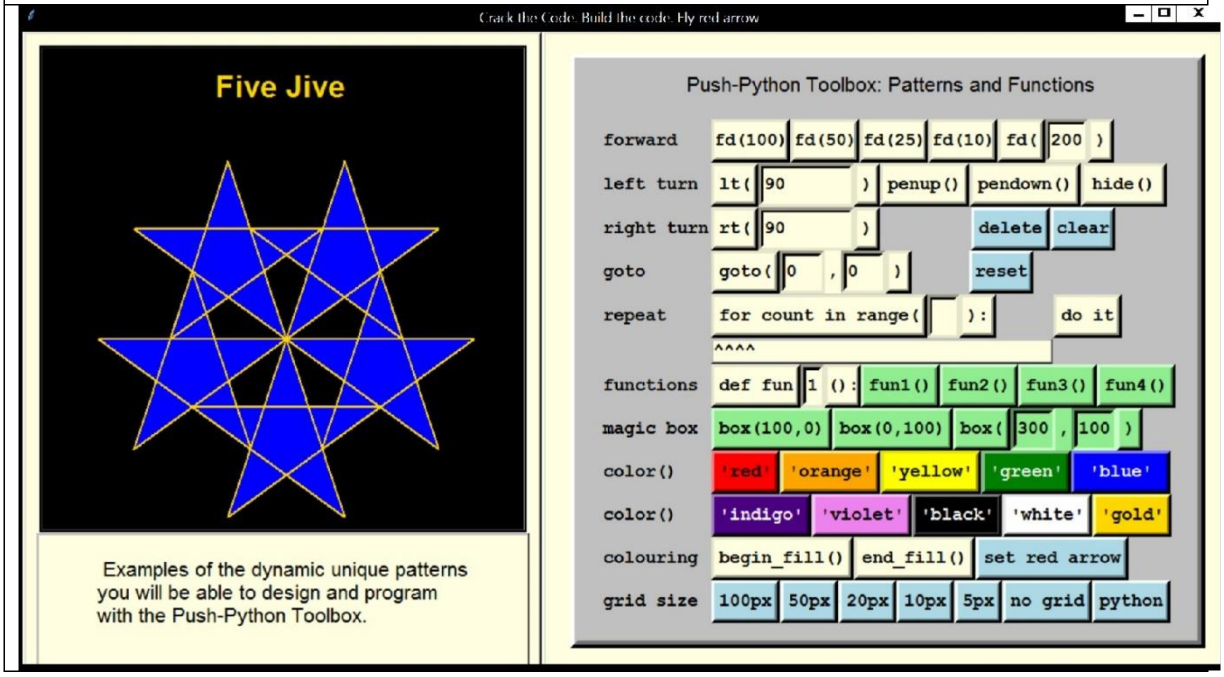


Push-Python

When we are ready, (at any time) we can switch to Push-Python, (Figure 9) a stand alone, push-button subset of Python 3, in which the buttons on the pad and the programs generated and recorded is Python 3.

This software technology will provide an introduction to computational thinking and ‘programming unplugged’ and for pupils a scaffolded transition to programming in Python 3.

Figure 9. Push-Python Prototype (Opening Screen)



Examples of the dynamic unique patterns you will be able to design and program with the Push-Python Toolbox.

References

The website for teachers on which the full Course Tao of Computational Thinking in Programming (a work in progress) is being developed is **ispython.com.**

The url for the Course is: **ispython.com/tao**

This booklet can be down loaded as a pdf file (in colour) from ispython.com/tao.pdf