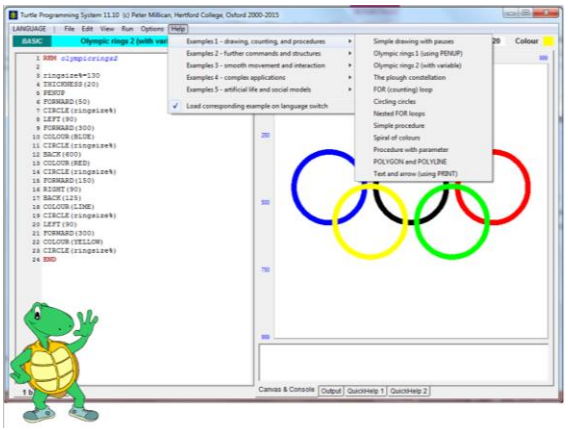
Transitioning To Text

Teacher Notes to support Tenderfoot Unit 1: How Computers Do Stuff – A conceptual approach to programming (part 2)

A practical introduction to Turtle System, reinforcing key concepts and constructs.


Turtle Programming System 11.10 (c) Peter Millican, Hertford College, Oxford 2000-2015
LANGUAGE
File
Edit
View
Run
Options
Help
BASIC
Olympic rings 2 (with vari
Examples 1 - drawing, counting, and procedures
Examples 2 - further commands and structures
Examples 3 - smooth movement and interaction
Examples 4 - complex applications
Examples 5 - artificial life and social models
Load corresponding example on language switch
Simple drawing with pauses
Olympic rings 1 (using PENUP)
Olympic rings 2 (with variable)
The plough constellation
FOR (counting) loop
Circling circles
Nested FOR loops
Simple procedure
Spiral of colours
Procedure with parameter
POLYGON and POLYLINE
Text and arrow (using PRINT)
Colour
Canvas & Console
Output
QuickHelp 1
QuickHelp 2

```
REM olympicrings2

ringsize%=130
THICKNESS(20)
PENUP
FORWARD(50)
CIRCLE(ringsize%)
LEFT(90)
FORWARD(300)
COLOUR(BLUE)
CIRCLE(ringsize%)
BACK(600)
COLOUR(RED)
CIRCLE(ringsize%)
FORWARD(150)
RIGHT(90)
BACK(125)
COLOUR(LIME)
CIRCLE(ringsize%)
LEFT(90)
FORWARD(300)
COLOUR(YELLOW)
CIRCLE(ringsize%)
END
```

Preparation required:
Turtle System available on all computers.
Telling The Time activity sheet for all pupils (optional).

A Note About Pedagogy

As with Part 1 of this Unit, we'll try to emphasise computational thinking, drawing out the 'big 3' constructs and 4 key concepts underpinning all the activities. Let's start by reiterating some lessons relating to pedagogy. We've already acknowledged the legacy of Seymour Papert, Logo, turtle graphics and his ideas about pedagogy. At its heart is the notion of constructionism: that children construct mental models in the course of building things. By mentally predicting then observing outcomes their knowledge is refined. They learn through doing but before children create, there is much value in class discussion, to model ways develop their investigatory powers.

We'll use an environment developed by Peter Millican (Oxford University) called Turtle System, designed specifically to support introductory programming through exploration of graphics. It is free and requires no installation – it runs as a standalone executable (currently Windows only). The software, User and Teacher guides are included in the resources, but check here: www.turtle.ox.ac.uk/ for later versions.

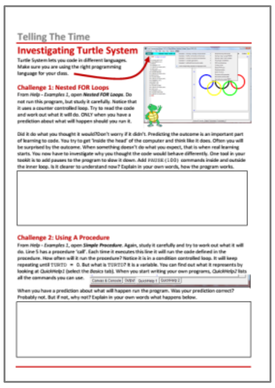
Turtle supports programs written in three 'real' programming languages (Java is under development). We'll use Basic for our examples but you may prefer to look at the Pascal or Python variants. Programs are loaded (or written) in the left window, whilst the output is observed on the canvas on the right, when the run button is clicked. The package comes with many example programs and the option to compare the same program in all three languages. We can use the examples as whole class exercises to get children to read the code, predict what will happen and demonstrate simple ways to investigate the behaviour.

The first example is a FOR loop. Display the code to the class. What will happen when this program is run? A red circle outlined in black is repeatedly drawn, spiralling outward in ever increasing steps. Here we can see the power involved in exploring code that develops patterns. The effect in the second example, Circling Circles is even more pronounced if you observe the program running. You will see the shape building as the loop executes. Again, encourage students to predict, as a class, then run it and observe.

It needs to be spelt out to students why you insist on a prediction, before observing the code running. When you READ code, you are putting yourself in the position of the computer. You are trying to think about how the computer thinks. You are thinking about thinking. When you OBSERVE the code running, unless you already have a mental model of what you expect, there is little gained form the experience. If you have a mental model – or prediction, then the observation confirms you were right, or contradicts it. If your prediction is wrong, you have to reach for an intellectual toolkit to reason about why it was wrong. You need to debug; to employ logical deduction. You need to observe and experiment. In employing these techniques you hone the ability to systematically investigate. You are learning to learn.

Put this way, children will embrace the approach, as they appreciate its value in improving their capacity to learn. A wrong prediction is something to celebrate – real learning takes place when your mental model is challenged. In more general terms; when your programs don't work, that is no reason to become disheartened. It is something to celebrate – an opportunity to debug and to learn! Alan Kay, who featured in Part 1, once famously said: 'If you don't aim for 90% failure, you aren't aiming high enough.'

Investigating Code



Telling The Time

Investigating Turtle System

Turtle System lets you code in different languages. Make sure you are using the right programming language for your class.

Challenge 1: Nested FOR Loops

From *Help - Examples 1*, open ***Nested FOR Loops***. Do not run this program, but study it carefully. Notice that it uses a counter controlled loop. Try to read the code and work out what it will do. ONLY when you have a prediction about what will happen should you run it.

Did it do what you thought it would?Don't worry if it didn't. Predicting the outcome is an important part of learning to code. You try to get 'inside the head' of the computer and think like it does. Often you will be surprised by the outcome. When something doesn't do what you expect, that is when real learning starts. You now have to investigate why you thought the code would behave differently. One tool in your toolkit is to add pauses to the program to slow it down. Add `PAUSE(100)` commands inside and outside the inner loop. Is it clearer to understand now? Explain in your own words, how the program works.

Challenge 2: Using A Procedure

From *Help - Examples 1*, open ***Simple Procedure***. Again, study it carefully and try to work out what it will do. Line 5 has a procedure 'call'. Each time it executes this line it will run the code defined in the procedure. How often will it run the procedure? Notice it is in a condition controlled loop. It will keep repeating until `TURTD = 0`. But what is `TURTD`? It is a variable. You can find out what it represents by looking at *QuickHelp1* (select the *Basics* tab). When you start writing your own programs, *QuickHelp2* lists all the commands you can use.

When you have a prediction about what will happen run the program. Was your prediction correct? Probably not. But if not, why not? Explain in your own words what happens below.

The series of exercises develop student's ability to read, predict and analyse code. You can work through them as a class exercise or an activity sheet is included for pupil's self-study. In Part 1, we highlighted the cognitive challenge of nesting constructs inside each other. ***Nested FOR Loops*** is an example using two repetition constructs. With clear visual output we can begin to appreciate how nested constructs execute. Adding a pause both within and outside the inner loop, and observing what happens should make the stages of execution clear.

We also highlighted in Part 1 the benefits of procedural abstraction. By defining procedures we can hide the detail involved within a procedure definition. This make complex code much easier to manage and understand. ***Simple Procedure*** shows procedure definition in Basic. Notice in this case the definition comes at the end of the program (unlike other languages). The procedure is called in the main program above. Look at the code carefully. This time we have an example of a condition controlled loop. When does the loop end? When the variable TURTD% = 0. But it isn't obvious where the variable is set, or changed? The tabs at the bottom give access to the help. Quick Help 1 / Basic tab explains that the TURT variables are built in. Having pointed this out, let the students predict what will happen BEFORE running the program and observing what actually happens. Was their initial prediction correct? Probably not. But if not, why not? Encourage them to articulate their thoughts.

One of the advantages of a purpose built beginners environment is that by having a small, restricted command set, help is much more accessible. Encourage students to begin to use Help to investigate, and then modify programs. Quick Help2 lists all the commands on the left.

Suggest students study the next example, ***Spiral of Colours***, which is reminiscent of our earlier squiral. Ensure they predict what will happen before having their mental model confirmed or challenged when they run the program. Was their prediction right? Can they modify the code so it draws a Squiral?

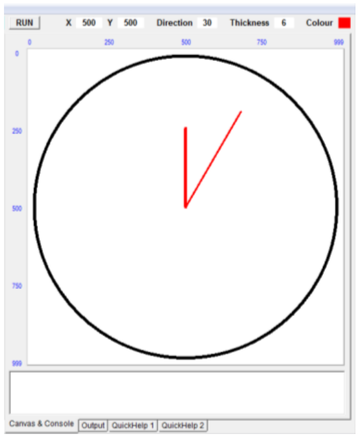
As a question and answer activity, ask the students to explain each line of code. If a student translation of a line is vague, ask others to sharpen it. If it is still not precise, model the correct the answer. Turn it into a collective challenge to hone the precision of the code reading.

Procedure With Parameter uses the same procedure definition as before but adds a parameter. We can generalise procedures with parameters. We met the idea in the RoboMind exercises in Part 1. Parameters allow a procedure to be called multiple times with different values. It is another cognitive leap and you may wish to leave this as an extension discussion for the very able. The dialogue below is offered as a model.

- When the procedure is called, what VALUE is passed to the parameter len%?
  Answer: On the first iteration 360 + 100 + 460.
- Can anyone precisely explain what will happen on the first call of this procedure?
  Answer: A line of a random colour will be drawn for 460 units. A blot of radius 23 is placed on the canvas and the turtle moves back 460 steps.
- What happens after that?
  Answer: The turtle turns 61 degrees clockwise before the procedure is called again.
- What value is passed to len% on the second iteration? Answer: 359 + 100 = 459 …

… and so on until you are confident they have grasped it. Now get them to predict what will happen when the program is run. Even though many will follow the way the procedure calls work, they will still be surprised at the result, so encourage a further discussion to try to explain what they have just observed. As a starting prompt, ask why there are 6 lines spiralling into the centre, rather than the one suggested in the code. Answer: There isn't. The reason lies in how far the turtle turns before being called again (61 degrees). If this isn't understood you could get a protractor and draw it out, or add appropriate pause statements, so the execution is slowed down. What this is, in effect, is a Catherine Wheel effect.

Telling the Time



X 500 Y 500 Direction 30 Thickness 6 Colour 🟥

Having looked at a variety of short programs students will have familiarity with all the commands required to tackle the following task. The challenge is to write a program to display an analogue clock, whose minute hand ticks round. It isn't complex, and does not require the use of parameters. At the risk of too much repetition, getting to grips with programming requires repeated exposure to the same concepts in a variety of contexts. So this represents a first attempt to write a program in a 'proper' programming language. The expectation is to use Help to identify the commands and generally work in a more independent fashion.

The key to a successful program is planning. The problem needs to be decomposed, perhaps as a group activity. It could break down into i) displaying a clock face and two hands ii) getting the minute hand to rotate and iii) rotating the hour hand when the minute hand has completed 60 'ticks'. Each tick does not need to take a minute!

Few people write before they can read. Now it is their turn to write code, insist on the same systematic method as before. Write a line of code, predict and observe. Reason logically about what you observe. If it doesn't work, use Help, or insert tests and pauses to confirm your theories. Fix it before moving on. There is no need to define procedures on the first iteration of this task. More confident students may wish to define procedures, but there is no need. Their code could be refactored to include them in a second iteration.

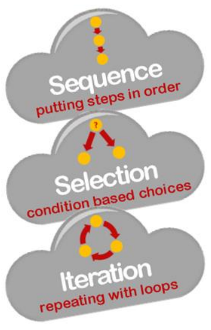
As their first attempt at writing a text based program, students will inevitably hit the syntax barrier. One of the great strengths of Turtle System, is that error messages are very precisely targeted, stating what the problem is and showing exactly where to find the problem. The presentation gives various examples of the clarity of the error messages. Notice in each case that the source of the error is underlined.

Configuring Turtle System

Turtle System comes with many program examples. We've looked at a few and they are very useful teaching aids. However, you may not want students to have access to them all. For example, a solution to Telling the Time is given in the second set of examples. You may also want to add your own to the Examples for them to investigate. Teachers can do all of this via the Power Users Menu. It is worth familiarising yourselves with this so you can configure a version of Turtle for the students, and have a separate version for teachers. Once the Power User Menu is available, the options under Help allow you to Save all the Help examples to the Turtle System folder. Each language folder created, has all the example files in it. This can then be modified, adding programs, or removing others.

The examples will list files only whilst in alphabetic sequence. The presentation demonstrates the effect of renaming the file D_theplough, so the alphabetic sequence only runs A to C. With the option for the Help menu to reflect the file structure, the restricted menu is shown. This feature allows quick reconfiguration and reinstatement. Note that if you wish to add your own programs to list in the menu, you must use the .tgx option in the Save As dialogue. You may wish to make other changes too, perhaps restricting to one language and disabling the Power Users Options. When finished, save them to an options file. Now when students open Turtle System, they can apply the settings from the options file.

A Conceptual Approach



With any programming environment there are lots of important elements that need to be mastered to make best use of it. When teaching it is easy to get bogged down in these details. Teachers and students need to keep 'coming up for air'. Get into the habit of pausing, taking a few deep breaths and refocusing on the key concepts.