code-it.co.uk

omputer  cience

Spelling Tester

Program Aim Can the student design a spelling test algorithm based on primary pupil and teacher needs. Can they convert this algorithm into Scratch code before adapting it to work in Python 3

Computational Thinking

-Designing an algorithm to work out steps pupil and teacher take when conducting a look cover write check spelling drill

-Evaluating the algorithm to see if it works and meets the needs of the pupil and teacher adapting it if need-ed.

-Decomposition, breaking program into stages to solve separately

Computational Doing

-Repetition -debugging

-Using a list

-Converting from one programming language to another

Dependencies

-Recommend pupils cover Maths Quiz with extra challenges in Scratch http://code-it.co.uk/scratch/mathsquiz/mathsquizoverview.html

-Recommend pupils cover Random module in Scratch as good intro to simple list use http://code-it.co.uk/scratch/randomword/randomword_overview.html

-Recommend pupils have been taught some basic Python 3 at least three modules from http://www.pythoncode.co.uk/ which include the Poem and Adventure game or equivalent from another source.

Resources

Portable Python

http://portablepython.com/wiki/Download/

Other Adult Support

Make sure other adults are on board with providing hints and strategies not solutions for pupils

Debugging Strategies

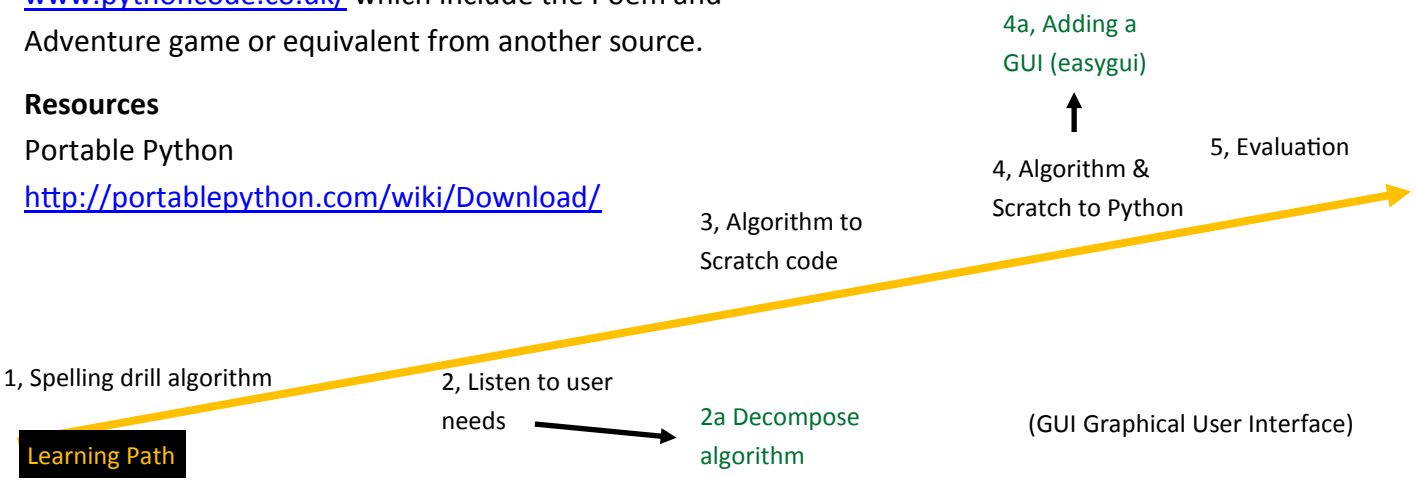
Algorithm Conversion to Code

Refer back to the algorithm. Which part of the algorithm are you trying to convert into code? Is their any key words in the algorithm that help you understand what you are trying to do?

Debugging

Can you break the code up into sections and check to see which parts work and which don't? If you decomposed the project in 2a this can help.

Python specific debugging help see http://www.pythoncode.co.uk/debug



(GUI Graphical User Interface)

code-it.co.uk

omputer  cience

Spelling Tester p2

1, Spelling Drill Algorithm

Remind pupils of all those spelling tests they used to do in primary school. Can they remember Look Cover Write Check. Remind them how they used to have a list of words which they would then cover before writing it down and finally checking to see if their spelling was correct. Who learnt them like this? Who used other variations?

After they have discussed this inform them that their task is to write a spelling test algorithm before turning this into code. They can discuss this in pairs but need to complete the activity themselves.

Remind them that an algorithm is a detailed list of instructions that accomplishes something. An algorithm is written in ordinary language so they are not allowed to program at this point.

If typing the algorithm encourage them to use numbered bullet points to help them organise their work 1.1.1 numbered list format works well.

Hint:- If pupils stick to sequence only in their algorithms ask them what aspects are repeated and how they could make the code more efficient using less code?

2, Listen to user needs

An important part of writing good software is to respond to the needs of the user. Give your pupils the transcript of the teachers discussion with the software researcher.

It is useful to ask pupils how they adapted their algorithm in response to the users needs part way through this section.

It can also help to share part completed algorithms that show detail and good structure.

A detailed example algorithm can be found on the main planning page

If students are struggling to grasp the concept it can be useful for them to view other programmers attempts to solve this problem http://www.amblesideprimary.com/ambleweb/lookcover/lookcover.html or http://www.bbc.co.uk/skillswise/game/en20memo-game-look-say-cover-write-check . They could also find pictures of this in Google Images and descriptions on the web.

2a, Decompose Algorithm

Can you break the algorithm up into sections? What does each section do? You could add section headers. We call this decomposition, breaking up a problem into manageable chunks. Is this useful for this type of program? Can you solve the chunks in any order? Although decomposition can be a very useful computational thinking skill it is less useful for a simple linear program like this but knowing when not to use it is important.

code-it.co.uk

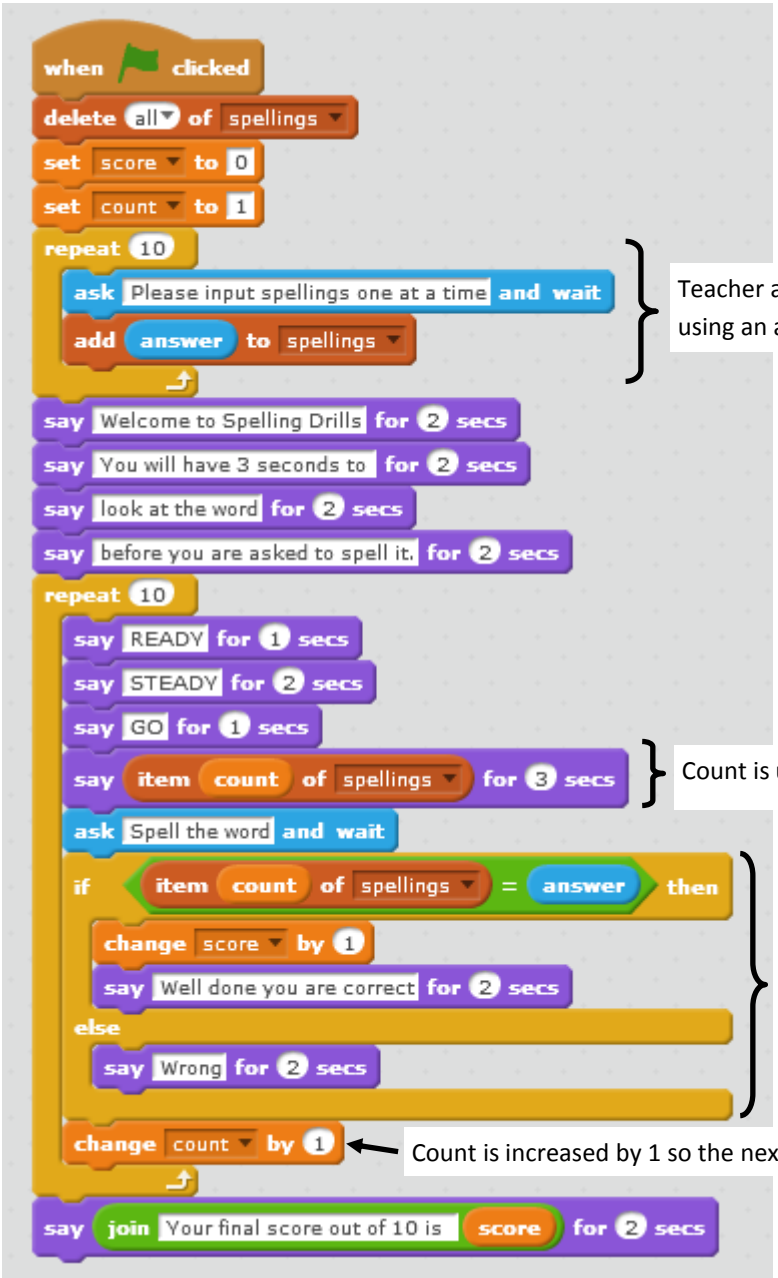
omputer cience

Spelling Tester p3

3 Convert algorithm into Scratch code

If pupils have completed the precursor lessons mentioned on page one and a detailed algorithm such as the one in the example then they should be able to build the Scratch program independently. There are some hint cards for those who get stuck but don't give these out unless pupils can say which part of the algorithm they are stuck on.

This is an example of one way this could be built it is by no means the only way



Teacher adding words into the spelling list for the class to learn using an ask input block

Hint
Iterating through the words
Ask pupils how the list orders the words?
Answer it numbers them
Ask them if numbers can be replaced by anything? (Answer a variable)
Could this be used in anyway?
You could use a pot and pencils to demonstrate adding to a variable

Count is used to work through the spelling list one at a time

The spelling word at first position on the list is checked against the user answer typed in and contained within blue answer block.

If they are the same (=) then a point is gained on the score and the user is congratulated. If they are not the same the user is told that they are wrong.

Count is increased by 1 so the next word is accessed on the list

code-it.co.uk

omputer  cience

Spelling Tester p4

4, Algorithm & Scratch to Python

All the code needed is on the Python Dictionary although pupils are welcome to use any code they have learnt from anywhere else.

http://pythondictionary.co.uk

Variables

http://pythondictionary.co.uk

Lists

http://pythondictionary.co.uk

Time Delay

http://pythondictionary.co.uk

Loops

http://pythondictionary.co.uk

Selection

http://pythondictionary.co.uk

print

http://pythondictionary.co.uk

```
import time
score=0
spellings=[]
for i in range(3):
    sp=input("Type in your spelling")
    spellings.append(sp)
print("Welcome to Spelling Drills")
time.sleep(2)
print("You have three seconds to read and learn the word")
time.sleep(3)
for i in range(3):
    print("Ready")
    time.sleep(1)
    print("Steady")
    time.sleep(1)
    sp1=spellings.pop()
    print("Your word is",sp1)
    answer=input("Type the spelling")
    if answer==sp1:
        print("Correct")
        time.sleep(2)
        score=score+1
    else:
        print("Incorrect")
        time.sleep(2)
        print("The correct spelling was",sp1)
        time.sleep(3)
    print("Your score is",score)
print("Your final score is",score)
```

This imports the time library that allows you to use the time commands

An empty list called spellings is created

This loop allows the teacher to input three words. These are put into the variable called **sp** before being added to the spelling list using append

This pauses the program for 3 seconds. Sleep will accept decimal fractions of a second.

The last word is added into the variable **sp1** and at the same time pop deletes it from the list so that it is not the last word anymore

Conditional selection is used to compare the users **answer** with the word inside **sp1** if they are the same a point is given. If they are not the same the correct spelling is given.

The real problem with this program is that the original spelling is still available on screen when the user is asked to try and recall it from memory. To overcome this problem we need some form of GUI, Graphical User Interface.

code-it.co.uk

omputer  cience

Spelling Tester p5

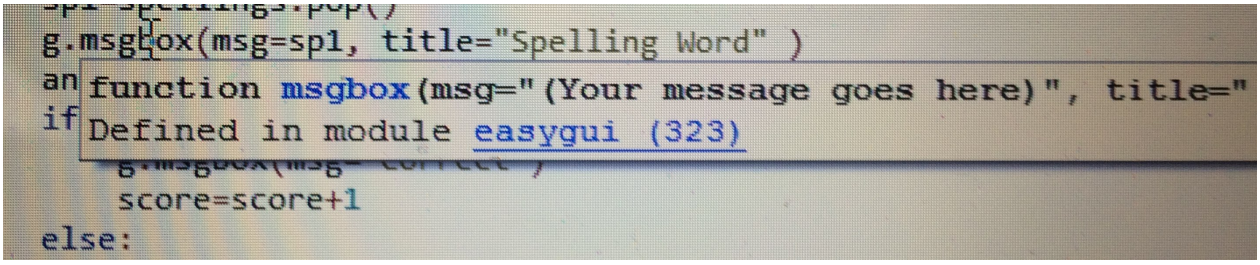
4a, Adding a GUI easygui

The most commonly used GUI in Python is Tkinter however this is a massive leap from the simplicity of Scratch. Easygui is a simple GUI that you can import from http://pythondictionary.code-it.co.uk/easygui it doesn't work well in idle. Message boxes [msgbox] are good for reporting simple information to the user and enterboxes [enterbox] are good for entering data into a variable. There are lots of other windows that could also be used detailed on http://pythondictionary.code-it.co.uk/

```
#easygui version
import time
import easygui as g
count=0
score=0
name=""
spellings=[]
for i in range(3):
    sp=input("Type in your spelling")
    spellings.append(sp)
g.msgbox(msg="Welcome to Spelling Drills")
g.msgbox(msg="Read the word and click ok when you are ready to be tested")
for i in range(3):
    sp1=spellings.pop()
    g.msgbox(msg=sp1, title="Spelling Word" )
    answer=g.enterbox("Type the spelling")
    if answer==sp1:
        g.msgbox(msg="Correct")
        score=score+1
    else:
        g.msgbox(msg="Incorrect")
        g.msgbox(msg=sp1, title="The correct spelling is")
g.msgbox(msg=score, title="Your final score is")
```

easygui needs to be pasted into Portable Python 3.YOURVERSION\App\Lib\site-packages for the easygui code to work

Most easygui windows have more options that those we have used. When the option is selected or hovered over the function of these are revealed.



code-it.co.uk

omputer  cience

Spelling Tester p6

5, Evaluation

A really good thing to do would be to go and try your programs out with some real primary pupils. Can you prepare four questions to ask them before hand which will help you collect their responses. Could you record their answers? What worked well? What do you need to improve/debug?

Time to respond to these comments is important.