code-it.co.uk

Fortune telling programming designed for upper KS2 or lower KS3

This is the first draft of a programming module that I have taught in many different ways.

It uses an adapted version of PRIMM a programming teaching method advocated by Sue Sentence Senior lecturer in Computer Science Education at Kings College London. You can read her article on PRIMM here https://blogs.kcl.ac.uk/cser/2017/09/01/primm-a-structured-approach-to-teaching-programming/

PRIMM stands for **Predict, Run, Investigate, Modify, Make**

In my first trial of this method I used the following sheet for pupils to **predict** what they thought the programming did.

We then **ran** the code as a class and tried out all of the options

We then used the second sheet to get inside the mind of the fictional programmer, both why she wrote the code and what types of things she was looking to do next with it (**investigate**).

There are a few targeted questions as I have found that not everyone reads everything thoroughly without some reason to read.

I then challenged pupils to build their own **modified** version of a fortune telling program using as much or as little of my example as they wanted to.

I deviated here from Sues classic model in that I didn't give pupils the code. I might have if this was text based.

I have only attempted this with classes that have used conditional selection, simple variables and simple lists before.

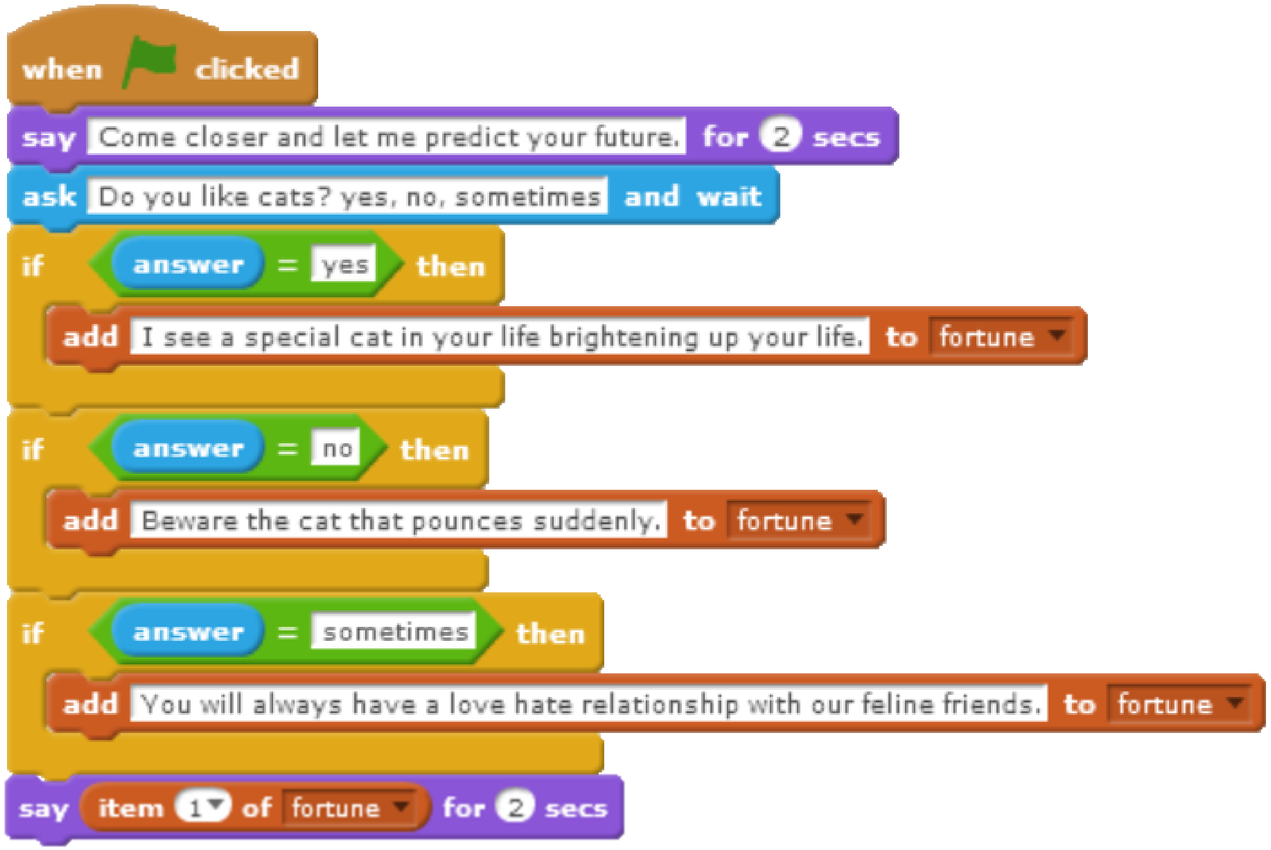
I think they have to have some understanding of the code elements involved before they can use this method.

Initial results have been very positive, my thanks to Sue for a really useful addition to programming pedagogue.

Phil Bagge 28th September 2017

code-it.co.uk

Predicting



```
when green flag clicked
say [Come closer and let me predict your future.] for (2) secs
ask [Do you like cats? yes, no, sometimes] and wait
if <(answer) = [yes]> then
    add [I see a special cat in your life brightening up your life.] to [fortune v]
end
if <(answer) = [no]> then
    add [Beware the cat that pounces suddenly.] to [fortune v]
end
if <(answer) = [sometimes]> then
    add [You will always have a love hate relationship with our feline friends.] to [fortune v]
end
say (item (1 v) of [fortune v]) for (2) secs
```

What do you think this program does?

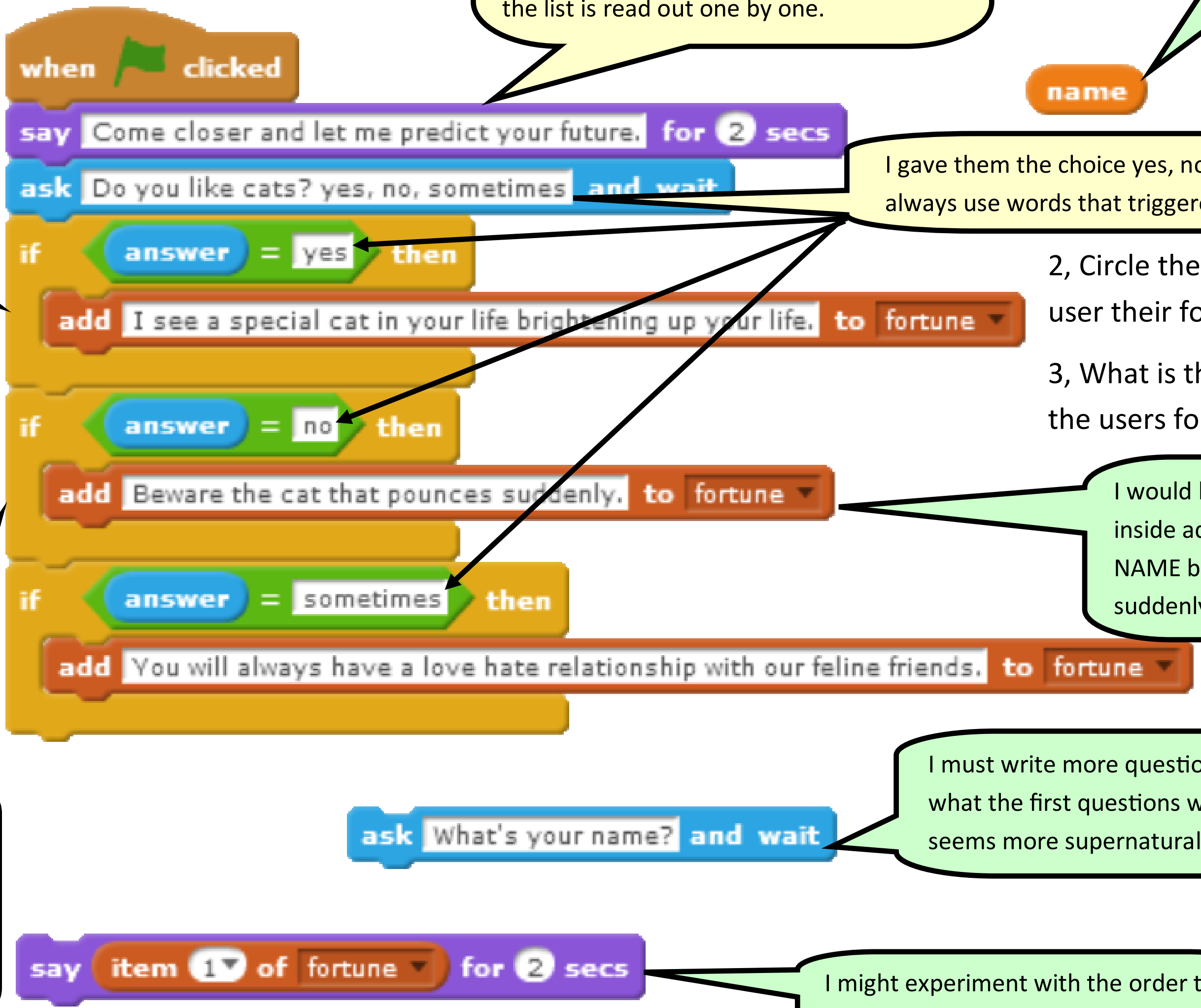
code-it.co.uk

I am writing a program that looks like it predicts the future. It asks the user questions and depending on their answer it puts my prepared responses into a list called fortune. At the end of the program the list is read out one by one.

It would be great if I asked the user what their name is and stored this in a variable so I can use it across the program and at the end when telling their future.

I have used lots of **if then conditional selection blocks** so I can let the user have lots of possible answers.

I gave them the choice yes, no or sometimes as user didn't always use words that triggered the conditional selection blocks



```
when [flag] clicked
say [Come closer and let me predict your future.] for (2) secs
ask [Do you like cats? yes, no, sometimes] and wait
if <(answer) = [yes]> then
    add [I see a special cat in your life brightening up your life.] to [fortune v]
if <(answer) = [no]> then
    add [Beware the cat that pounces suddenly.] to [fortune v]
if <(answer) = [sometimes]> then
    add [You will always have a love hate relationship with our feline friends.] to [fortune v]

(name)

ask [What's your name?] and wait

say (item (1 v) of [fortune v]) for (2) secs
```

1, If the user typed in no to the first question what would be added to the list?

2, Circle the code section that will tell the user their fortune

3, What is the name of the list that stores the users fortune?

I would love to use the name variable inside adding things to the fortune list. NAME beware the cat that pounces suddenly

When I tested this with users sometimes they didn't spell some of my answer options correctly. I would like to force them to answer the question again if they choose to write something else.

I must write more questions so that users forget what the first questions were and the fortune seems more supernatural.

I might experiment with the order the program tells the fortune.

Marina has started to write a fortune telling program. She has jotted down some of her reasons for adding code in yellow and some of her things she might do next in green.

Where Next?

Explore other programming planning at code-it.co.uk/csplanning

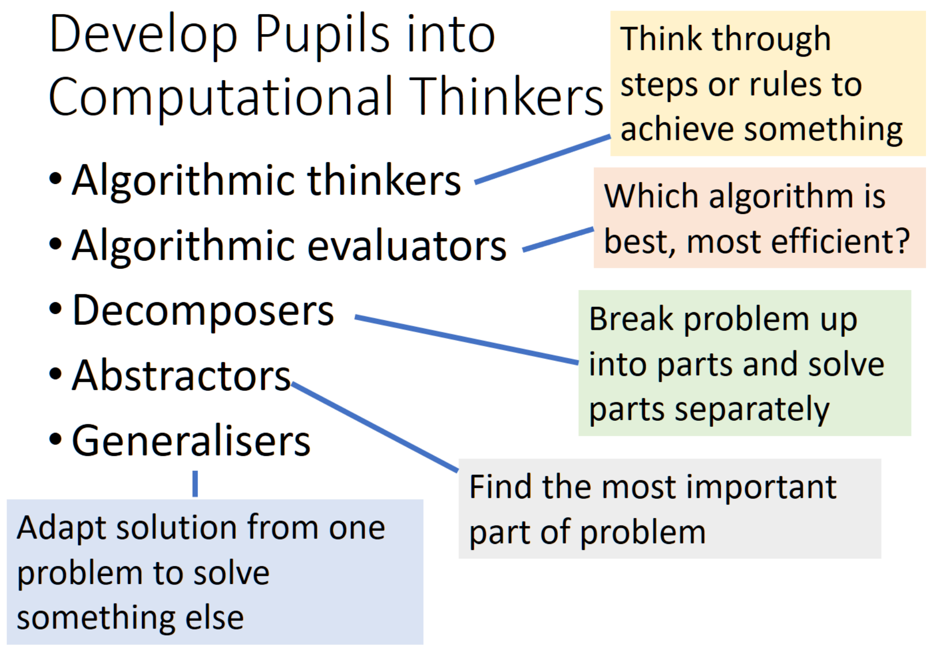
Design your own programming planning using computational thinking & problem solving skills

You can follow me on Twitter

@baggiepr

You can find lots of free resources at code-it.co.uk

I enjoy sharing my journey with fellow educators around the world



Develop Pupils into Computational Thinkers

- Algorithmic thinkers — Think through steps or rules to achieve something
- Algorithmic evaluators — Which algorithm is best, most efficient?
- Decomposers — Break problem up into parts and solve parts separately
- Abstractors — Find the most important part of problem
- Generalisers — Adapt solution from one problem to solve something else

Complexity	I can break complex problems into parts	I can discover / concentrate on the most important part of a problem	I can explain how I used decomposition & abstraction
Ambiguity	I recognise there is more than one way to solve a problem	I recognise there is more than one way to describe a problem	I can explain how I managed ambiguity
Open Ended	I look for a range of solution to the same problem	I don't just accept the first solution	I can describe how a project can be extended
Adapt	I can adapt existing ideas to solve new problems	I can identify patterns in problems & solutions	I can explain how I adapted a solution to solve a new problem
Evaluate	I can evaluate my solutions against a set criteria	I can design criteria to evaluate my creations	I can explain how evaluation helped me improve a project
Experiment & Debug	I can develop, test and debug until a product is refined	I repeatedly experiment through making, testing & debugging	I can explain how using the iterative cycle improves my work
Persist	I can persevere even if the solution is not obvious	I learn from setbacks and don't let them put me off	I can describe how I overcame problems
Communicate	I can contribute useful ideas to a partner or group	I can encourage others to share their ideas	I can lead using all the people talent in my group

Decomposition 🟥 Abstraction 🟨 Generalisation 🟦 Algorithmic Evaluation 🟩 Algorithm 🟧

Adapted from Behaviour Rubric created with @MarkDorling and linked at http://code-it.co.uk/attitudes/ Adapted with permission