Debugging Progression Steps for primary coding

Teacher responses to encourage debugging in red

1. Pupil is passive and expects teacher or peer to fix their code often characterised by very general statements like 'I am stuck' or 'I don't get it' where no effort has been made to determine where in the code the mistake is or even what aspect they are stuck on. (It is not my job to fix your code, can you tell me exactly what your problem is, what does it do that is wrong?)
2. Pupil can spot there is a problem and can describe what that it is or what it is doing that is wrong. (Well done, can you point to where in the code that is happening?)
3. Pupils start to develop simple pre-reading debugging strategies
    a. Walking through or tracing steps if code is logo, movement or turtle (bee-bot) orientated
        i. Ticking off steps that work to arrive at code with bug (Have you ticked of all the code which works?)
        ii. Walking through route to determine commands that are correct (Have you tried walking through the route and checking your code as you go?)
        iii. Moving model through route to determine commands that are correct (Have you tried moving the model/robot through the route and checking your instructions one by one?)
        iv. Directing another child to move through code sequence whilst checking code step by step (Why don't you work together, read your instructions slowly whilst your partner carries them out)
        v. Reading Bee-bot symbols one at a time as turtle (bee-bot) carries out the instructions
    b. Comparing their code with that of their neighbour or teacher to see if it is the same
        i. Looking for simple colour differences if using blocks (Are the colours of your blocks the same as the colours of teachers or neighbours?)
        ii. Looking for different shapes (Are your code blocks the same shape as teachers or neighbours)
        iii. Looking for things that are missing (How many lines of code have you got, is that the same as teachers or neighbours?)
        iv. Looking for too many things (How many lines of code have you got, is that the same as teachers or neighbours?)
        v. Looking for missing gaps (FD 40 FD40) (Have you got a space between the command and the number?)
        vi. Looking for patterns that are different (Do all of your if commands line up underneath each other? This is hard to give examples as all patterns are different)

4. Pupils start to develop reading strategies
    a. Pupils read code aloud to see if it does what they wanted it to do. (Have you read it aloud to see if it does what you wanted it to do?)
    b. Pupils read code to a partner to see if it sounds right (Why not read your code aloud to your partner to see if it sounds right or is in the right order?)
    c. Pupils read code to see if it the same as teacher or peer (Why not read teachers or peers code. Now read your code, are they the same?)
    d. Teachers read code written in abbreviations in full (Read your code in full, not using the abbreviations, aloud, does it still make sense?)
5. Pupils ask questions of the code that doesn't work. These can take the form of asking a question such as if I do x why doesn't y happen? Or why doesn't this code section work all of the time? (Teachers often need to model these before pupils will think to ask them themselves)
6. Pupils step through more complex code which could include loops, selection and variables. They describe to a peer or teacher what is happening in each stage. (Pupils will only do this if it is modelled regularly by teachers. Can you describe to your partner what is happening in each step of your code, point to it as you describe it)
    a. A variation is to list what is happening at each stage. This is particularly useful when dealing with multiple variables or variables which change within loops. (Can you list what happening inside each variable after each repeat?)
7. Pupils use strategies such as divide and conquer. Breaking up longer sequences of code and running parts of it separately to try and find out where the error is. In the case of multi-threaded programming such as multiple Scratch blocks running each block separately to try and isolate the bug. This strategy can be difficult to use if parts of the code are dependent on other parts functioning correctly.
(Can you save your work and then split it up into parts and run each part separately to try to find the bug) (Run each block one at a time, is there any block that is not working as you thought it should?)

Please Note

- Pupils don't move uniformly through this progression and encouraging a variety of strategies is important. Sometimes a pre-reading strategy may be the most useful in debugging complex code.
- This is a general progression and some pupils will find some strategies more useful than others.
- I suspect these are not the only strategies but are good to start with please feel free to email me or tweet any others that I may have missed that are broadly primary and I will add them to the list.
- It is very important that teachers don't panic when they can't see an error. The worst reaction is to take over from the pupil and fix their code for them. Encourage the

pupil to work through a range of strategies one at a time and come back and let you know if that fixed the problem.

- Although I have written this for code a lot of these strategies can be transferred into debugging algorithm design as every program is an algorithm turned into code.

Please feel free to use or adapt this but please credit the source if you re-publish an adaptation.