

Session 5:

Patrick Baker: Sharing of Teachers best ideas from teaching the new CS NCEA material (1.44, 1.45, 1.46)

Facilitated Discussion:

What we were teaching wasn't really preparing kids for the next levels.

Decided to teach 1.40, 1.41, 1.44, 1.45, 1.46 -> this was too many in hindsight.

What worked and didn't:

This is all open comments and feedback...

1.44

Teach as Research based report, found kids were not up to it and could not understand it. They needed some practical examples. Bought scales, canisters etc. Once the kids were doing it physically they could understand it.

Write down whats happening, then use cards, draw sets.

As soon as we got them off the straight theory they started to understand it much more.

Regurgitating theory doesn't develop understanding, but when they examined each others cell phones they could then demonstrate understanding at a higher level.

Others...

HCI -> putting out several example language and structure more clearly explained and the kids were more confident.

Bubble sort outside using large bits of paper. A4 paper with numbers and having kids sort themselves.

Battleships: Linear, sequential or lucky. Careful you don't use too many examples to confuse them. 2 algorithms is right number. eg. Linear and binary.

Students found it hard to explain in own words.

Natural Language vs. Programming Languages vs. Programming Languages.
Spelling unimportant in algorithm devl. but highly specific and accurate in programming.

Natural Language ----> Code

Range of instructions in different areas, kids sort into the different areas.

Testing each others language and algorithms, students can run through simple algorithms and be critical in a positive way.

Taking inverses;

eg traffic light. explain it without the picture. shows language is not precise enough.

Eg, draw a rectangle, put a circle in it, put ... Students ended up with 20 different versions of a traffic light.

Compilers:

Look up on internet, simple codes, machine code. Logo code. Low level vs. high level code.

Used Pascal, showed the compiler running, if there was a syntax error it will not compile. Students could see compiling in action.

In Python will only go up to the line where the error was. Can still do other things. A compiled language compiles the whole thing, and cant if there is a problem, an interpretive code will run up to that point but may not display the error.

As the kids work through this they need to demonstrate understanding.