Introduction Lesson: Developing the Pseudocode - Part 1

Teacher says: We have been working on identifying Prime numbers and Composite numbers. What would you say to someone who asked you what the differences were between Prime and Composite numbers?

Teacher says: We have been making Factor Trees. What would you say to someone who asked you what are Factor Trees?

Note: The pseudocode developed for creating Factor Trees in the previous math lessons should be available at this time for this coding lesson.

Note: Try to keep past pseudocodes &/or flowcharts available and posted in your classroom as posters or PDF for on-line referral. They can be used as reference as well as to show increased efficiency to decompose math algorithms and as examples of computational thinking.

Remember these two examples are possible pseudocode lists and should not be shown to your students. They are for you, the teacher, to have some idea of how to proceed. A mathematical algorithm or a formula is standard but this activity is to capture your students thinking in their own words. Over time the repetition of decomposition in this visual format will reveal your students' thinking as more efficient and the use of more mathematical terms and the names (and descriptions) of coding blocks.

Here are two examples of what you might have developed over time with your students:

Early example of pseudocode (students beginning to learn to express their math thinks when decomposing an algorithm)	**Developing example of pseudocode** (students beginning to learn to express their math thinks when decomposing an algorithm)
● Draw lines from numbers to a bigger number ● If the number can have two parts to multiply together, draw lines from those numbers up to that number	●Get a composite number to make a factor tree ●Remember the number (*) ●Think of the number as a product of two factors ●Think what two numbers multiplied together have that product ●How to think of those numbers? - Remember a multiplication fact with the same product (go directly to #)

● Print those 2 factors, one at each end of the line ● If one of those factor numbers can have two parts to multiply together, draw two lines from that number ● Print those 2 factors ●When no more parts then factor tree is done	If I can't think of those numbers, I … - Make arrays of blocks and the number of rows and columns will be the 2 factors - Multiply two numbers together and if this product is greater than the product I want, I change the factor(s) to be less; if this product is less than the product I want I change the factor(s) to be more - Remember a division fact that has its dividend as the same number as the product I want (#) ●Remember the two factors ●Print the two factors underneath the product, the lesser one to the left, and the other to the right ●Draw lines to each factor as branches (that's where the Tree name comes from) ●**If a factor is** a Prime Number (A whole number greater than 1 that has only two factors, itself and 1) circle it ●**If a factor isn't** a Prime Number, then it is Composite and has two factors ●Remember this number ●Repeat from (#) ●If both factors are Prime numbers STOP ●List all the prime factors (A factor that is a prime number.)

Why is there no advanced example?

An advanced or "perfect" PseudoCode will be one that you and your students develop together and work out/debug the problem areas (consider having a second column and adding the coding blocks that match the actions). It might only be "perfect" after the coding has been completed and tested out (Data Analysis and Data Representation & Abstraction) (see Glossary) and you and your students returned to the pseudocode to make more edits.

Might this advanced "perfect" pseudocode look exactly like an action list a different class creates?

Probably not, if there are complex mathematical algorithms in the stack. Possibly but not always, if the coding is a simple stack of a mathematical operation.

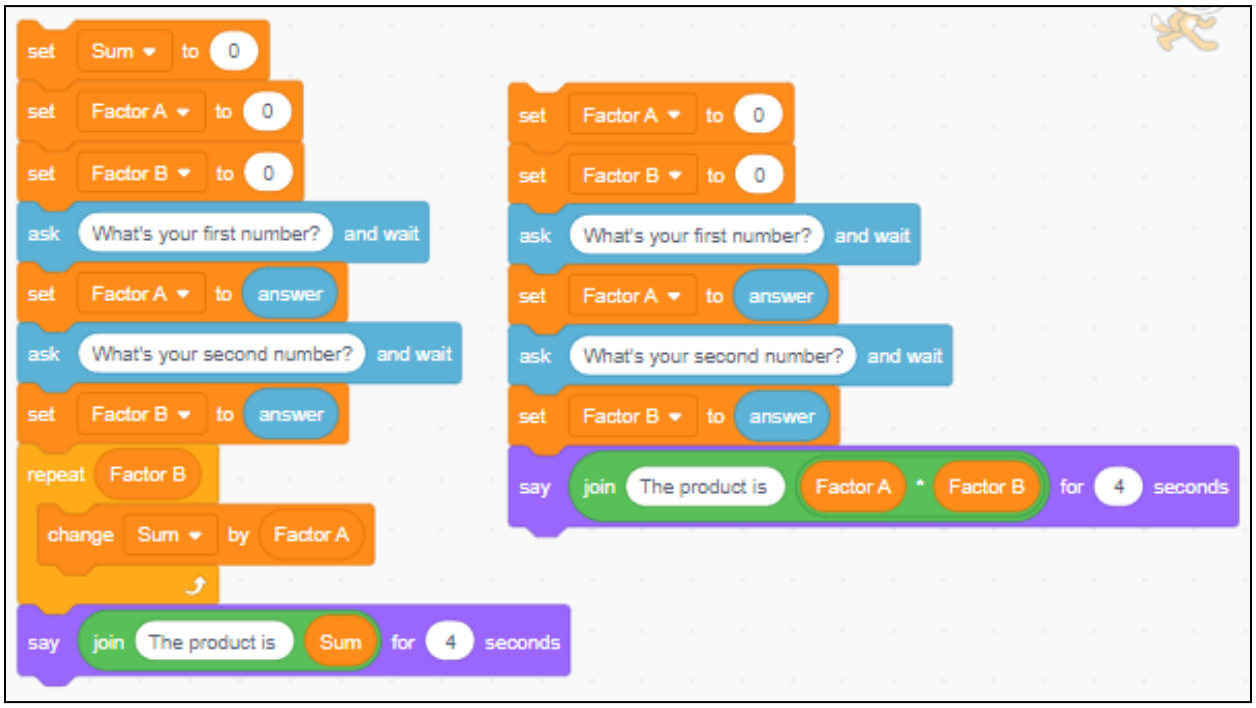
Compare the coding for:

- What a product might be when multiplying two numbers together and adding the same number over and over.
- Coding can mirror the relationship between addition and multiplication, but which code is efficient and is one always used over the other?

- The answer/output is the same for each stack, but is one method better than the other?
- What about the thinking required to create both stacks?

These are the questions that will make the pseudocode discussions rich and highly impactful.

Coding to Add	Coding to Multiply
3 variables, Repeat block, total 10 blocks	2 variables, multiply Operator block, total 7 blocks



Scratch Wiki Links:

[Multiple Operator Block Wiki Article]

[Add Operator Block Wiki Article]