Thinking like a computer (or a robot)

Purpose

The purpose of this exercise is to get students thinking about how one needs to tell a computer to do something, which is basically what computer programming is. *Computers are very precise - they only do exactly what you tell them.* This is good for a programmer because computers are predictable and will act exactly the way they have been programmed. But it can also be a challenge because computers don't have any "common sense," so that serious problems can result if the instructions they are given do not carefully address every step and situation that may occur.

Overview

In this activity, students will need to address the following concerns:

- Write specific instructions to tell a robot how to prepare a common food item or do some simple task, like lifting an object and moving it somewhere else.

- Be as specific as possible. See the example below for lifting up a pencil and moving it to a different spot on a table. (Feel free to copy some of the instructions wholesale if you like them; it's good programming practice to reuse code.)

- Try something similar, like lifting another object of a different size and shape and moving it, or lifting the pencil and putting it on a different table. (These are fine examples, no need to think of another, and if you do, keep it simple! See how many steps just to lift and move a small object!)

- Students can assume that all of the ingredients are available and have not run out. For extra credit, consider how you would amend your program so that it would respond appropriately if you had to move the pencil from the table top and put it in a cup on a different table.

- Finally, make a list of the different commands that the robot needs to understand.

Example: Lifting and repositioning a pencil on a table

Assume the robot starts one meter in front of the edge of the table

1. Face the table.
2. Walk forward on meter to the table.
3. Stop when at the table.
4. If the pencil is not in front of you, identify if to the left or right of you.
5. If the pencil is in front of you, stay still and go to step 6; If to the right, step sideways to the right until pencil is in front of you, else if to the left, step sideways to the left until the pencil is in front of you.
6. Reach out with your hand straight out in front of you.
7. Stop reaching out your hand when it directly over the pencil.
8. Lower your hand until it touches the pencil.
9. Position two fingers on either side of the pencil.
10. Bring the two fingers towards each other so they squeeze the pencil.
11. Hold the pencil between the two fingers.
12. Lift your arm back to a position straight out in front of you.
13. Take three steps to your right and stop.
14. Lower your arm until the pencil touches the table top.
15. Move the two fingers apart to let go of the pencil.
16. Lift your arm back to a position straight out in front of you.
17. Move away from the table.
18. End.

Examples of formal Commands you may or may not consider using:

- **Walk** forward
- **Step** to one side
- **Hold** object
- **Let go** of object
- **Place** object1 **on** object2
- **Go into** room
- **Remember** something
- **Lift** object
- **Lower** object
- **Find** object **in** location
- **Return** object **to** location

In the technology/computer world, the list of instructions a program spells out for a computer is called an ***algorithm.***

For the teacher:

It is very useful and fun for the class to spend about 5-10 minutes after all students have had a chance to make their instruction list for some simple action, and **have a volunteer come to the front of the room and be the robot.**

The volunteer should be instructed to **take every command of someone's list as literal as possible**. So if an instruction is "pick up a pencil" the robot might pick it up as high as they can, instead of just above a piece of paper.

To be clear for the robot, the instruction would have to be detailed to the point of saying how high above the paper, and get the coordinates of where to position the pencil with respect to the paper, and so on.

It is the minute details of instructions we want the students to understand and appreciate if they are going to take on anything related to computer programming, or at the very least to have some clue as to what a computer program is, and what an algorithm is.

Student Outcomes

- Students will become aware of the mindset they will need when solving certain types of problems or when computer programming.
- Students will also begin to *improve problem solving skills* because they will be thinking of problems or tasks in much greater and finer detail than they typically need to do in a standard math problem.
- Students will begin to learn a new way of detailed communication of instructions, whether it is for simple or complex tasks.

Time

Students will need about 30 minutes to complete this activity.

Level

This can be an opening exercise for any age group of students who are going to begin to learn about programming. This exercise is independent of any programming language.

Materials and Tools

Students will just need a notebook to write their instructions, or a computer to type the instructions.

Preparation
The teacher will want to consider whether students need to use a computer to type this exercise or not. The teacher will also want to consider how in-depth the class should discuss an example before trying the activity.

Prerequisite
This is for beginning programming students. If students are experienced in any type of programming, there likely is no need to do this activity.

Background
Computer programming in any language is built upon systematic, detailed, and logical progressions of instructions that tell the computer what to do. The list of instructions needed to figure out some problem or solve some set of equations is **an algorithm**, and for many students who have never thought this way, this type of thinking is difficult.

The use of this type of exercise is that many students become better problem solvers in general, whether it is in a math class or science class, because programming and getting a program to work properly requires the student to think through a solution of a problem in more detail than they are probably used to.

Teaching Notes
It will be useful to have a whole class discussion after students complete their work. Students may share the lists of instructions for a given task, and through discussion other students may find missing steps that could be essential to completing the task correctly. A teacher may also take a list of instructions and ask what the consequences would be if particular steps were missing. Thinking about consequences is an important part of problem solving, and this type of exercise is good exposure and practice for students.

One recommendation is to have a student in front of the class who is a robot, and give the rest of the class a task they need the robot to do. It could be anything: lifting a book and moving it to another position in the room, taking a sip of water, writing something on paper or on a chalkboard, etc. Tell the student playing the robot to take every instruction from classmates as literal as possible (e.g. if someone says “lift a piece of chalk” the robot will continue lifting it as high as possible, since no lift height was provided by the user). This is great fun and really drives home the point of *being as specific with instructions as possible*. This is how one must write a computer program.

Assessment
Teachers may choose to look through the student work, or it may be interesting to let students grade each others. It will get them thinking about other problems, and they may be able to determine if a set of instructions is complete or not by picturing themselves having to follow the instructions – often they will have questions for the author of the list when a series of steps is unclear or incomplete for the task.