

Name: _____
Class: _____
Teacher: _____

Computing Concepts

**Year 7 Digital Technologies
2019 Term 1**

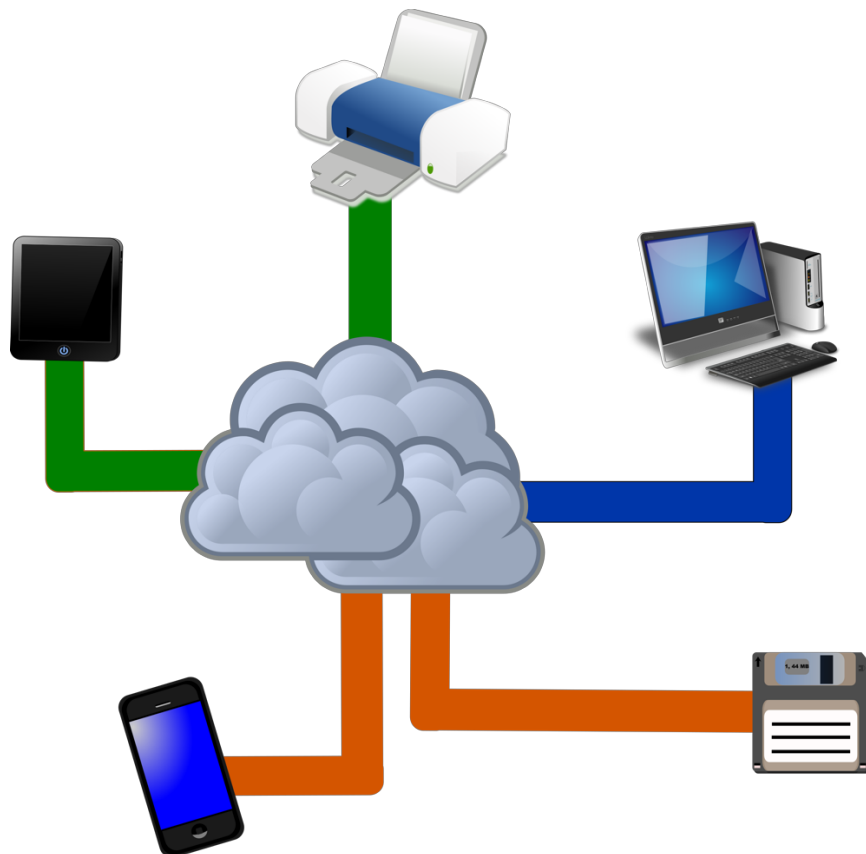


Image Source:
https://openclipart.org/image/2400px/svg_to_png/170263/Cloud-computing.png

Computing Concepts

Year 7 Digital Technologies Term 1

This term, we are introducing you to a range of fundamental concepts in Computer Science.

This booklet includes all the resources you need for classroom lessons, so that you will be prepared for the next unit, Coding in Python.

Contents

Google Drive Fundamentals

Introduction to Algorithms

Flowcharts, Algorithm Design & Pseudocode

Activity Guide – Binary Message Devices

Activity Guide – Coordination and Binary Messages

Activity Guides – Broadcast Battleship



Google Drive Fundamentals

Now that you have successfully logged into your CGS Google Drive account you can use this to store files and documents for all your subjects.

Q1. Briefly explain why storing your files in a cloud-based storage provider, like Google Drive, is beneficial. **Make sure you write in full sentences, professionally and completely with correct spelling and grammar.**

Q2. Google Drive provides a variety of different file formats. Provide two examples of how you could possibly use each file format in one of your subjects this year. Bullet point responses are okay for this question.

Google Docs:

-
-

Google Sheets:

-
-

Google Slides:

-
-

BONUS if you can provide examples for using Google Forms:

-
-

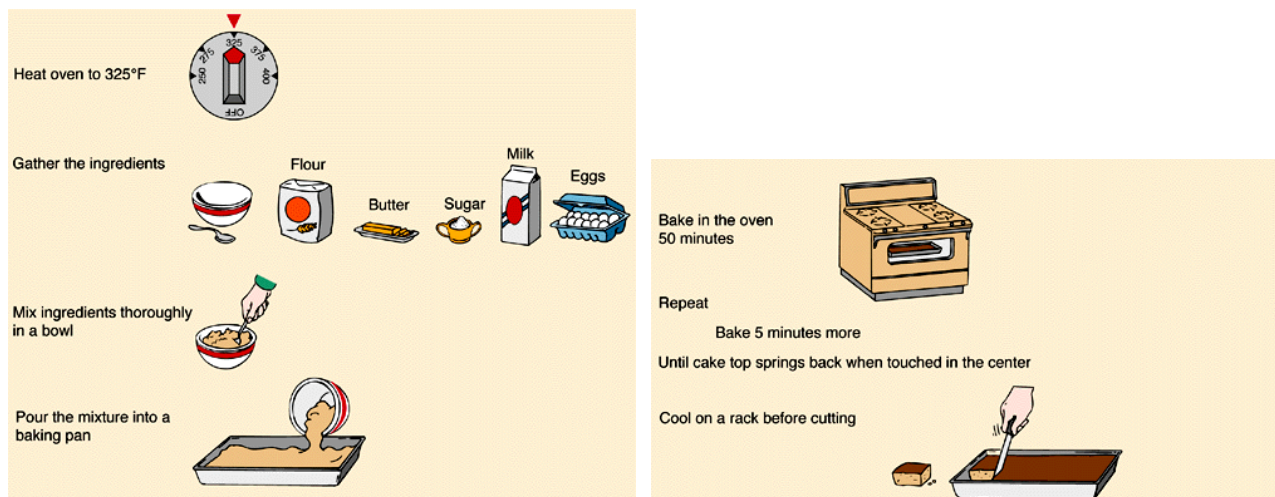
Introduction to Computational Thinking

Digital Technologies is not about learning to write code - it's learning the skills of **computational thinking**. That said, computational thinking is most often used to write computer software to solve a particular problem.

What is *computational thinking*? Wikipedia provides a great definition: *Computational thinking is the thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer—human or machine—can effectively carry out.*¹

We express a solution to a problem as an algorithm. What is an algorithm?

Algorithm Example: Baking a cake²



What are characteristics of a good algorithm? Bullet points are okay to answer this question.

-
-
-
-

¹ https://en.wikipedia.org/wiki/Computational_thinking

² <https://www.wiley.com/college/busin/icmis/oakman/outline/chap05/slides/algor.htm>

When we develop an algorithm to solve a problem we need to provide very precise information. This is easier said than done, particularly if attempting to instruct a computer which (unlike a human) cannot read between the lines and is far less forgiving if ambiguous instructions are provided. Computers are very compliant, they will do exactly what you tell them to do.



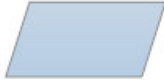


To demonstrate this, you are going to write an algorithm to instruct your teacher to make a vegemite sandwich. They are going to follow your instructions exactly! Your teacher will show you what items are available to make the sandwich, and you can only use those along with instructions to your teacher.

Sometimes when we test algorithms, the results are different from what we expected!

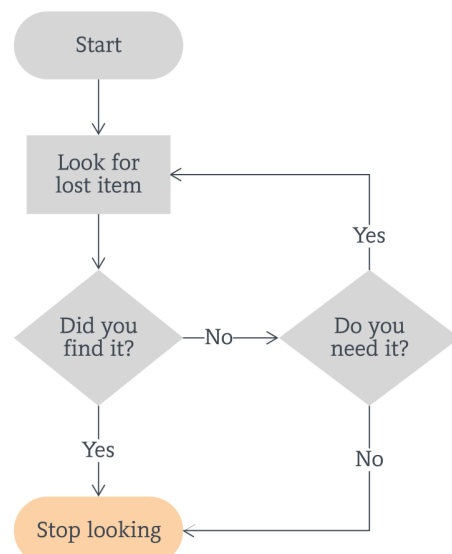
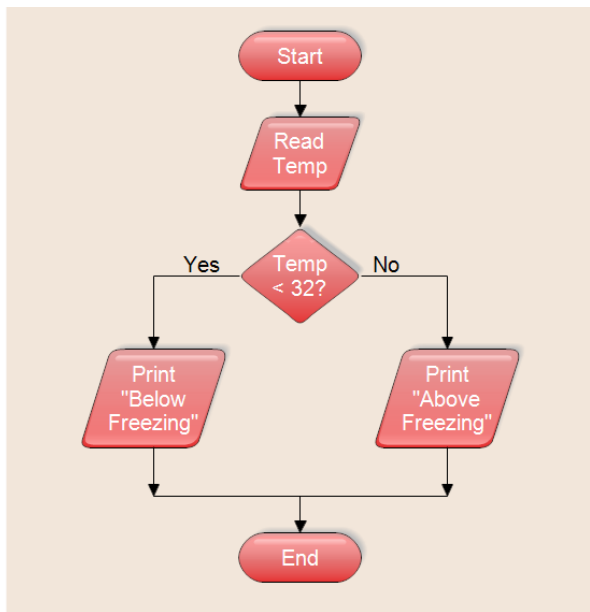
What could you improve in your algorithm, after testing it? You may like to write a new version of your algorithm here.

Flowcharts

Flowcharts are a visual translation of an algorithm, which define the flow of a program from one step to the next. They consist of a few basic shapes that have particular meanings:

| Symbol | Name | Function |
|---|--------------|--|
|  | Start/end | An oval represents a start or end point |
|  | Arrows | A line is a connector that shows relationships between the representative shapes |
|  | Input/Output | A parallelogram represents input or output |
|  | Process | A rectangle represents a process |
|  | Decision | A diamond indicates a decision |

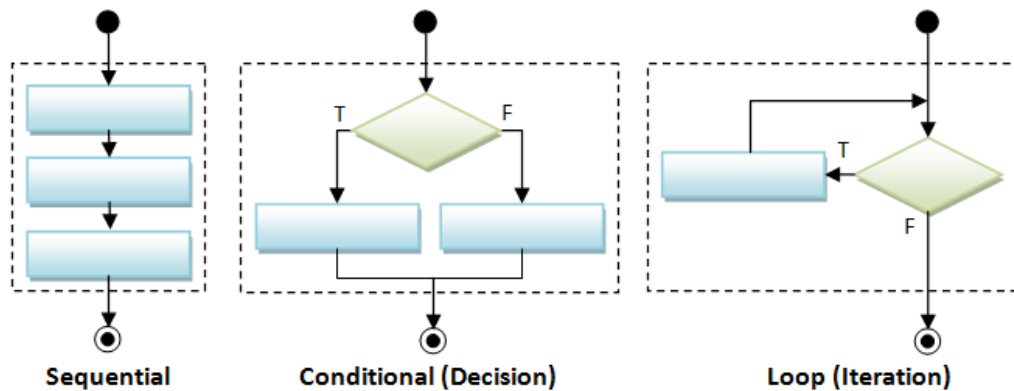
Flowchart Examples: Freezing Check / Look for lost item



Notice in the above examples that an algorithm can follow a different path, when a *decision* is made. An algorithm can also repeat a process, as you can see in the second example. In programming, this is often called a *loop*.

Algorithm design

When designing algorithms, you should consider three fundamental control structures:



Sequential

This means that a program will be completed in a certain order.

The algorithm below wouldn't work so well. Write out the steps in the correct order.

Brushing your teeth

Rinse your mouth out with water

Put the toothbrush in your mouth

Scrub for two minutes

Apply toothpaste to toothbrush

Conditional (Decision)

This means that during a program, a decision is made based on a certain condition.

Here is one way to improve the above algorithm. How would you draw the steps in a flowchart? Note that a decision should always have at least two arrows, each pointing to a different point.

Check if toothpaste is empty

If toothpaste is empty (condition)

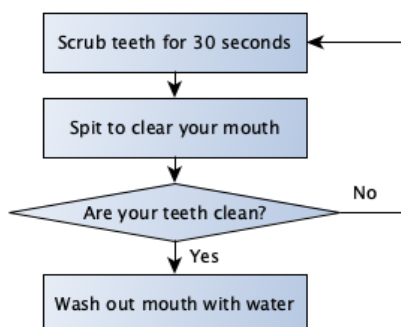
Open new toothpaste

End if

Apply toothpaste to toothbrush

LOOP (iteration)

This means that during a program, a decision could be made to loop back to a previous step.



To the left are some steps in a flowchart that could be used to modify the algorithm.

Fill out the missing instructions in the algorithm below.

Loop while teeth are not clean

(1)

(2)

End loop

(3)

Pseudocode

Pseudocode is a high-level description of an algorithm. It uses similar structures as actual programming languages, except that they read more like standard English to make it easier to understand.

The algorithm examples on the previous page about brushing your teeth are actually examples of pseudocode. Here's another example. Have a go at drawing a complete flowchart. Remember to use your start/end symbols!

Note that a decision should always have at least two arrows, and sometimes it can have more than two arrows if required.

Getting ready for school

Put books in bag

If it's summer season

Put hat into bag

End if

If it's really cold

Put on a winter jacket

Else if it's just a little cool

Put on a light jacket

Else

Just wear a shirt

End if

Pick up bag and leave for school

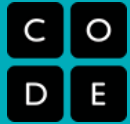
Your own algorithm

If you have time, use the space below to create an algorithm and/or flowchart for one of the following (or choose your own topic):

- How to cook eggs
- How to use a microwave to heat up food
- How to choose a new movie or TV show to watch

Name(s) _____ Period _____ Date _____

Activity Guide - Binary Message Devices



Scenario: You are going to build a device out of classroom supplies to send information to a classmate on the other side of the room. There are some basic rules and constraints:

- **Stay on your side.** You may not walk to the other side of the room.
- **No language.** That means no writing or talking to communicate.
- **No projectiles!**

Challenge 1: Simple Binary Message (state A or B)

Time Limit: 5 mins

- Choose the **binary question** your device will be used to answer.
- **Create a device** using classroom items to send a simple binary message - state A or B.
- **Try to make it fail-proof.** Consider a few obstacles. Would it still work if...
 - There was something in between you and your partner?
 - You couldn't see your partner?
 - You were in a loud room?
 - Your partner wasn't paying attention?

Record how to use your device to send a state A / B in the table below

Your Binary Question: _____

| Message | How to send with your device |
|-----------|------------------------------|
| A: | |
| B: | |

Check-in with the Teacher

- Demonstrate the messaging systems
- Record this information about your device in your journal using a table similar to the example above.
- Your teacher may use the rubric to assess your device.

Challenge 2: Complex Messages (4 possible messages)

Time Limit: 5 mins

Not all questions have only two possible answers. Your new **challenge is to invent a way** to use your device to send an answer to a question that has **4 possible answers!** Think about these things:

- Should you modify your device?
- Should you use it in a different way?
- Should you make a new device entirely?

You've got 5 minutes! GO!

After you've done some testing, make a note below about how to use your device to send 4 possible messages. (enough that another person could pick up your device and use it).

| Message | How to send with your device |
|---------|------------------------------|
| | |
| | |
| | |
| | |

Challenge 3: Complex Messages (8 possible messages)

Time Limit: 5 mins

What if you wanted to ask an even more complex question with **8 possible answers?**

Just as before update your device and test it out. Record how to use your device in the table below.

| Message | How to send with your device |
|---------|------------------------------|
| | |
| | |
| | |
| | |
| | |
| | |
| | |

Challenge 4: Complex Messages (16 - n possible messages)

Time Limit: 5 mins

Could we keep increasing the number of messages forever? Could our devices be used for questions with 16, 32, or 1,000,000 possible responses? Some things to think about...

- Our alphabet only has 26-letters, yet we can spell many words
- Our number system only has 10 digits yet we can represent many numbers
- Think back to your simple two-state device. Could you simply use it differently, rather than modifying it?

Discuss with your partner

- How could you use your device to respond to much more complex questions (for example one with 1,000 possible responses).
- Use the space below to **describe the system you develop** in such a way that another group could pick up your device and use it to send messages this way.

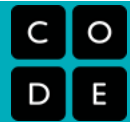
Class Discussion

Follow your teacher's instructions for presenting your work to the class. You might need to:

- Describe how your system works
- Provide a simple example of your system
- Do a live demonstration of your system being used

Name(s) _____ Period _____ Date _____

Activity Guide - Coordination and Binary Messages



Develop your Protocol: You and your partner will need to send a 2-bit message back and forth on the Internet Simulator. One partner will have a secret 2-bit message (for example BA). When your teacher says “Go” that partner will send the message using the Internet Simulator. The second partner will then send the same message back. At the end you’ll check that the correct secret message was successfully sent back and forth.

Your Protocol: You will need to agree on rules, or a “protocol” to make this message exchange work. Develop your protocol in the space below. Make sure you consider:

- How will you know when the exchange is supposed to begin?
- How will you know whose turn it is to send or receive the message?
- How will you coordinate your actions?

Practice: Practice sending a 2-bit message back and forth (4 bits sent in total). Try to get your time as low as possible without making mistakes. Record your fastest time in the space below.

Bits Transmitted: 4 Time in Seconds: _____

Challenge: Extend your protocol so that it can send more bits. Can you make it work for 4-bit messages or 8-bit messages? Keep improving your protocol so that you can send more bits as quickly as possible without making mistakes.

Bit Rate: A bit rate is a measure of how fast a system transmits bits. You can calculate your protocol’s bit rate by dividing the number of bits sent by the amount of time it takes. Calculate your bit rate for one of your fastest runs of your protocol. Note: If you send 4 bits back and forth, you’ve actually transmitted 8 bits.

Bits Transmitted: _____ Time in Seconds: _____ Bit rate: _____ bits/sec

Activity Guide - Broadcast Battleship Rules



Overview

Today you'll be playing a crazy version of the classic board game, Battleship. In the classic game, the board is 10x10, but here we play with a 3x3 board. The classic game is a guessing game for two people. Here are the rules for our multi-person version of the game.

Goal: Win the most games!

Game Objective

- The object of Battleship is to try and sink another player's ship before they sink yours.
- The other player's ship is somewhere on his/her board
- Each turn, you try to guess your opponent's ship location by calling out a coordinate on the grid.
- The other player also tries to hit your ship by calling out coordinates.
- You maintain and mark two grids:
 1. Your *Ship Boards* - where you record shots fired against you.
 2. Your *Target Boards* - where you record the shots you fired against others.

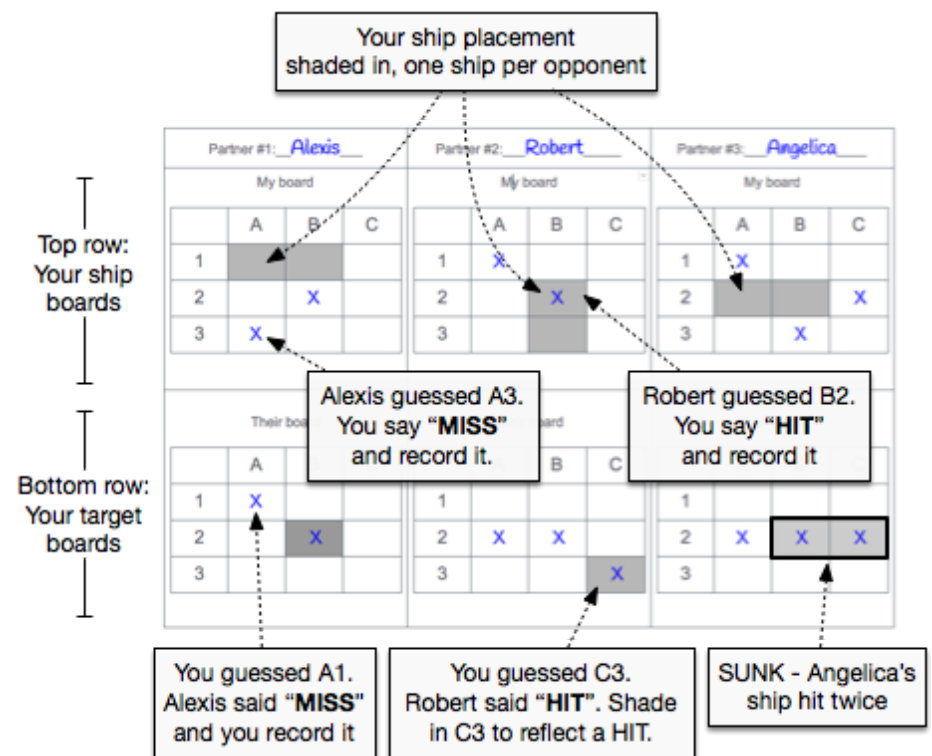
Starting a New Game

- Each player places a ship on their board - one ship for each opponent
- A ship occupies two contiguous squares on the grid - vertically or horizontally. NO Diagonal placement.
- Once the guessing begins, the players may not move the ships.

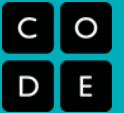
Playing the Game

The diagram shows an example of a game in progress between 4 players - you, Alexis, Robert and Angelica. You can see how you might mark the board.

- Players try to guess their opponent's ship locations by calling out coordinates.
- The opponent responds with "HIT" or "MISS" as appropriate.
- Both players should mark their grid with separate markings for hit or miss.
- When both of the squares that represent one ship have been hit, that should be announced as "HIT AND SUNK!". As soon as all of one player's ships have been sunk, the game ends.



Activity Guide - Broadcast Battleship Game Board



Battleship Directions:

1. Write in the names of your partners on the lines below.
2. Shade in boxes for your ships' locations in the "My board" sections. Ships are 2 units long and are either horizontal/vertical, no diagonals.
3. You can have a different ship placement for each opponent.
4. Don't show your board to your opponents! Record each hit with an "X" and miss with an "O."

| | | | | | | | | | | | |
|-------------------|---|---|---|-------------------|---|---|---|-------------------|---|---|---|
| Partner #1: _____ | | | | Partner #2: _____ | | | | Partner #3: _____ | | | |
| My board | | | | My board | | | | My board | | | |
| | A | B | C | | A | B | C | | A | B | C |
| 1 | | | | 1 | | | | 1 | | | |
| 2 | | | | 2 | | | | 2 | | | |
| 3 | | | | 3 | | | | 3 | | | |
| | | | | | | | | | | | |
| Their board | | | | Their board | | | | Their board | | | |
| | A | B | C | | A | B | C | | A | B | C |
| 1 | | | | 1 | | | | 1 | | | |
| 2 | | | | 2 | | | | 2 | | | |
| 3 | | | | 3 | | | | 3 | | | |
| | | | | | | | | | | | |

Name(s) _____ Period _____ Date _____

Activity Guide: Invent a Binary Protocol for Battleship



Overview

Previously you came up with a method for exchanging messages on an open broadcast channel to play multiple games of Battleship at once. Now that you've played Battleship this way, with your group or with a partner, **describe an efficient binary protocol for playing a 3-person game of Battleship that can be played accurately over the Internet Simulator.**

Let "efficient" mean that your protocol uses the smallest reasonable number of bits (0s and 1s) to make messages for Battleship that still contain all of the necessary information for playing the game.

Directions

1. List all of the information (data) that you will need to communicate in order to play Battleship. *You will need to express all of this with your protocol, so think about how you can express each piece of data in binary.*

Example Grids for Player A

Player A's ship on the board

| | A | B | C |
|---|---|---|---|
| 1 | | | |
| 2 | | X | |
| 3 | X | | |

Player A's guesses

| | A | B | C |
|---|---|---|---|
| 1 | | | X |
| 2 | X | | |
| 3 | X | | |

2. Make a diagram or chart that explains the protocol, what the bits represent, and how to use the protocol to play Battleship. Don't worry about coming up with a "correct" protocol -- just one that works! Make sure that the diagram shows some example encodings of things that can happen during play.

While developing your protocol here are some **questions and suggestions** to think about:

- Remember that ultimately you are making a communication protocol for a computer to read, so if it's hard for a human to decipher, but would be easy to describe for a computer, that's fine.
- A message you send over the Internet is just a string of 0s and 1s -- so your protocol should indicate how the bits work (i.e. what is the binary representation?)
- You can also invent other rules or standards of play that would help make the protocol work well.
- Think about using numeric addresses rather than people's names.
- Think about what else you can communicate with binary to reduce the total number of bits. How "efficient" can you make your message? How small can you make the size of a message?
- You might consider testing out your protocol with the Internet Simulator to see if it's viable.

Rubric

| Criteria | Yes | No | Comments |
|---|-----|----|----------|
| All data required for a Battleship message is represented as a binary protocol. | | | |
| An example of a Battleship message is given and explained. | | | |
| All data required for a Battleship response is represented as a binary protocol. | | | |
| An example of a Battleship response is given and explained. | | | |
| The overall protocol is explained clearly. | | | |

Name(s) _____ Period _____ Date _____

Activity Guide - LEGO® Instructions



Challenge

Create instructions your classmates could use to reproduce a simple arrangement of LEGO® blocks. Do the following:

1. Create Your LEGO Arrangement

You will be given 5-6 LEGO blocks which you should connect into a single arrangement. Try to choose something interesting or challenging to test your instruction-giving abilities.

2. Record Your Arrangement

Record your arrangement somehow so you can recall it later. Make a simple drawing, take a photo, etc. You'll want an exact record, so make sure you pay attention to color!

3. Write Instructions

In the space provided below, write a clear and precise set of instructions your classmates could follow to build this arrangement on their own, without diagrams or pictures. That is, your instructions may only use words, so you cannot use pictures to help you.

Test Your Algorithm

Exchange algorithms and blocks with another group and try to follow the instructions provided to create the correct arrangement. Afterwards, confirm whether you succeeded by using the other group's image.

Reflection

Once you've had an opportunity to test one another's instructions, respond to the following reflection questions.

- Were you always able to create the intended arrangement? Were your instructions as clear as you thought?
- Why do you think we are running into these miscommunications? Is it really the fault of your classmates or is something else going on?

Name(s) _____ Period _____ Date _____

Activity - Minimum Card Algorithm



“Human Machine” Algorithms

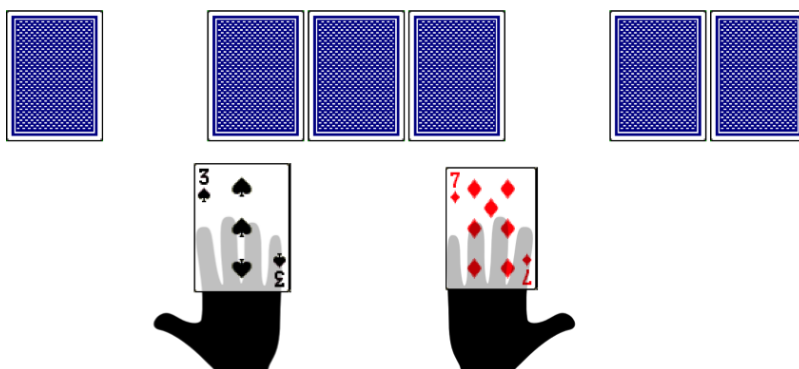
We often get started thinking about algorithms by trying to rigorously act them out ourselves as a sort of “Human Machine”. When acting as a machine, we can keep the limitations of a computer in mind.

In this activity, you’ll design an algorithm to find the smallest item in a list. Obviously, if we were really writing instructions for a person, we could simply tell them: “find the smallest item in a list.” But that won’t work for a computer.

We need to describe the *process* that a person must go through when they are finding the smallest item. What are they *really* doing?

Setup and Rules:

- We’ll use playing cards face down on the table to represent a list of items. Start with 8 random cards face down in a row.
- Any card on the table **must** be face down.
- When acting as the machine, you can pick up a card with either hand, but *each hand can only hold one card at a time*.
- You can look at and compare the values of any cards you are holding to determine which one is greater than the other.
- You can put a card back down on the table (face down), but once a card is face down on the table, you cannot remember (or memorize) its value or position in the list.



Task:

Write an algorithm to find the card with the lowest value in the row of cards.

- Goal: The algorithm must have a clear end to it. The last instruction should be to say: “I found it!” and hold up the card with the lowest value.
- The algorithm should be written so that it would theoretically work for any number of cards (1 or 1 million).
- Write your algorithm out on paper as a clear list of instructions in “pseudocode.” Your instructions can refer to the values on cards, and a person’s hands, etc., but you must invent a systematic way for finding the smallest card.

My Algorithm To Find Minimum Card

Write your algorithm below. We suggest writing it out as a numbered list of instructions to make the sequence clear.

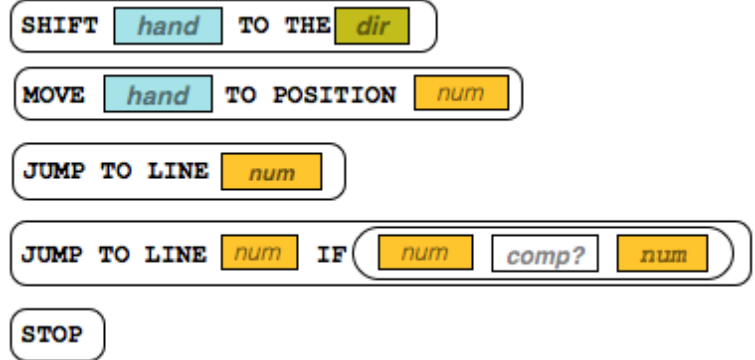
1.

The “Human Machine” Language

Here are the beginnings of a more formalized low-level language you can use to create programs for a “Human Machine” to solve problems with playing cards.

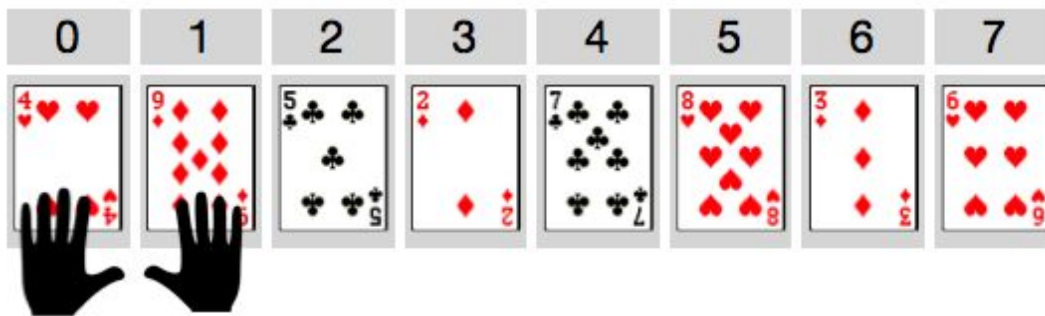
To simplify things we'll get rid of the need to pick cards up and put them down. Instead leave cards face up and just touch them. The 5 commands you can use are shown to the right. **See the [Reference Guide](#) on the next page for descriptions of what these commands do.**

Some of these commands might seem unusual, but we can write programs with just these commands to control the “human machine's” hands to touch or pick up the cards, look at their values, and move left or right down the row of cards.



Standard Card Setup

You should assume this standard initial setup. Here is a diagram for an 8-card setup:



- There will be some number of cards with random values, lined up in a row, face up.
- Positions are numbered starting at 0 and increasing for however many cards there are.
- The left and right hands start at positions 0 and 1 respectively.

Try out some example programs

Get to know the Human Machine Language by acting out the examples on the following page with a partner. For each of the examples on the next page you should:

- Lay out a row of **8 cards** in front of you to test out the program.
- Have one partner read the instructions in sequence starting at line 1, and the other partner act out each command as the human machine.
- Use the [code reference](#) to answer your questions and verify you're interpreting the code correctly.
- Give a brief description of what the program does, or its ending state.

NOTES:

- Some of the programs are very simple
- Some of the programs might not ever stop
- The point is simply to practice using the language and executing commands as a “Human Machine”

| Example Program | What does it do? |
|--|---|
| <pre> 1 SHIFT RH TO THE R 2 SHIFT RH TO THE R 3 SHIFT RH TO THE R 4 SHIFT RH TO THE R 5 SHIFT RH TO THE R 6 SHIFT RH TO THE R 7 STOP </pre> | |
| <pre> 1 SHIFT RH TO THE R 2 JUMP TO LINE 1 3 STOP </pre> | <p><i>Note: this one has a problem, can you find it?</i></p> |
| <pre> 1 SHIFT RH TO THE R 2 JUMP TO LINE 1 IF RHPos ne 7 3 STOP </pre> | |
| <pre> 1 MOVE RH TO POSITION 7 2 SHIFT LH TO THE R 3 SHIFT RH TO THE L 4 JUMP TO LINE 2 IF RHPos gt LHPos 5 STOP </pre> | |
| <pre> 1 JUMP TO LINE 5 IF LHCard eq 9 2 SHIFT LH TO THE R 3 MOVE RH TO POSITION LHPos 4 JUMP TO LINE 1 5 STOP </pre> | <p><i>Note: there is a potential problem with this one too. But only in certain circumstances. Can you find it?</i></p> |

Human Machine Code Reference Guide

Hands, Values and Direction

There are some short-hand abbreviations for referring to the human machine, the cards, positions, and directions of movement.

Hands - The Human Machine has hands! You can refer to a specific hand abbreviated **LH** or **RH** (left hand or right hand).

Values - Each hand has two values you can refer to:

1. **LHPos**, **RHPos** - The hand's position in the list (a number)
2. **LHCard**, **RHCard** - The value of the card the hand is holding (a number)

Direction - There are two directions **R** and **L** (right and left) that hands can move along the row of cards.

Hands

| | |
|-----------|------------|
| RH | Right Hand |
| LH | Left Hand |

Values

| | | |
|---------------|---------------|----------------------|
| LHPos | RHPos | Position in the list |
| LHCard | RHCard | Value on the card |

Directions

| | |
|----------|-------|
| R | Right |
| L | Left |

Commands

| Description | Examples |
|--|--|
| <div>SHIFT hand TO THE dir</div> <p>Shift the given hand one position to the right or left along the row of cards.</p> | <div>SHIFT LH TO THE R</div> |
| <div>MOVE hand TO POSITION num</div> <p>Move a given hand to a specific position number in the row of cards.</p> | <div>MOVE RH TO POSITION 4</div> <div>MOVE LH TO POSITION RHPos</div> |
| <div>JUMP TO LINE num</div> <p>Jump to a specific line number in the program and continue execution from that point.</p> | <div>JUMP TO LINE 1</div> |
| <div>JUMP TO LINE num IF num comp? num</div> <p>Jump to line but ONLY IF the comparison of two numbers is <i>true</i>. If the comparison is <i>false</i> then just proceed onto the next line of code.</p> <ul style="list-style-type: none">• For numbers, you can use integers or any of the hand values RHCard, LHCard, RHPos, LHPos• For comparisons you can use eq, ne, lt, gt, (equal, not equal, less than, greater than) | <div>JUMP TO LINE 4 IF LHCard eq 7</div> <div>JUMP TO LINE 2 IF LHCard lt RHCard</div> <div>JUMP TO LINE 7 IF RHPos gt 9</div> |
| <div>STOP</div> <p>End of program. Stop doing anything, stop executing lines of code.</p> | <p><i>This should be the last line of code in the program, or on a line that is jumped to when you want the program to stop.</i></p> |

Challenge: Find Min

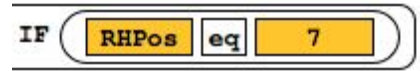
Using only the Human Machine Language design a program to find the card with the smallest value in the list of cards.

Goal: When the program stops, the left hand should be touching the card with the smallest value.

Hint: How do you know you're at one end of the list or the other?

Use the hand position values to check whether the position is 0 or the largest position in the list - you can assume that you know how big the list is ahead of time.

For example, if the last position is 7, then the comparison: **IF RHPos eq 7** would tell you that the right hand was as the end of the list.



Write your program in the space provided below

| | |
|----|--|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |
| 10 | |

Command Cut Outs

Print out this sheet and cut out each command to use as lines of code in the template provided on the previous page. Alternatively, you can just write the commands by hand into the template.

SHIFT TO THE

SHIFT TO THE

JUMP TO LINE STOP

JUMP TO LINE STOP

MOVE TO POSITION

MOVE TO POSITION

SHIFT TO THE

SHIFT TO THE

JUMP TO LINE STOP

JUMP TO LINE STOP

MOVE TO POSITION

MOVE TO POSITION

JUMP TO LINE IF

JUMP TO LINE IF

JUMP TO LINE IF

JUMP TO LINE IF