Please refer to the online lesson plan on the DT Hub to access all website links and additional resources.

## Learning hook

1. Begin the lesson by explaining to students that we’re going to be writing a program that will allow us to determine the winner of a representative election – it could even be the election you run for student leaders at your school! But first, we need to understand the different ways that elections can occur. For that, watch the Counting your vote video from the Australian Electoral Commission.

As another means of representation, provide a handout to complement the interactive. The handout could list key terms and definitions as well as illustrate (in simple terms) how to do ‘First to the Post’ and Preferential Voting tallying.

1. Depending on how you want to run the lesson, you may want to skip straight to the voting system explanations. However, it doesn’t hurt to give students a better understanding of the Australian democratic process anyway.

As a means of expressing themselves you can also demonstrate ‘First to the Post’ voting by asking students to raise their hand to vote for two familiar, similar options for example:

* Drinks: Coke or Pepsi?
* Fast foods: McDonalds or KFC?
* Music: hip-hop or Indi rock?

1. After exploring the interactive, discuss with students which voting system is used at your school when voting for student leaders. Why is that system used? Is it fair? How closely does it resemble one of the systems used in voting for the Australian parliament?

As a means of engagement and recruiting interest explore your school voting systems:

1. What could be done to improve the school’s voting system?

2. Is it completed online?

3. How can someone with a visual impairment vote in this system? For example, Will printed materials be made in large or high-contrast print or is a screen reader available when reading online text?

1. Regardless of which system is used, one of the most important aspects of ensuring that an election is fair and appropriately conducted is to have a process for counting the votes accurately. When votes are collected using a secret ballot process on paper ballot forms, what are some of the potential issues? Have students come up with their own lists of issues and suggest possible solutions. They should include things such as:
   * Time consuming (use computers!)
   * Possible to miscount (have someone check each vote after it is counted; use computers!)
   * Potential for tampering with physical votes during counting (have multiple people check each vote  
     Use computers!)
   * Have multiple people agree on the interpretation; use computers!)
2. Ask the students to volunteer their issues. When they identify one, ask the group if they can think of a solution to the problem (possible solutions are listed above below each point as a guide). Students should identify that using a computer to count votes would solve most (if not all) of the issues they’ve identified. And that’s what we’re going to do!

## Learning map and outcomes

Share the learning intentions for the activity. Tell students that the class will be:

* identifying the requirements of our software so that it accurately counts votes and determines a winner using our chosen voting system
* designing and implementing appropriate algorithms to allow us to automate the vote counting process
* considering some of the issues and potential risks or problems our solution might introduce if it were to be adopted for larger, public elections.

Learners have a clear understanding of what they are learning and can see how it fits into the bigger context.

## Learning input

Before they start defining their requirements, it is important students have completely understood the different voting systems being used.

The AEC Education site provides a resource called *Topic 3 – What’s your vote worth?* in their series on voting and elections. The introductory activity in this resource is an exploration of a hypothetical student election, and the associated worksheet is available as *Topic 3 – Blackline Master 1* from the download link on the AEC Democracy Rules page.

Using the teacher guide for the introductory activity, have students work through BLM1 with particular emphasis on the secret ballot elections (ie one first past the post, the other preferential).

Over the course of this module, students will develop programs that correctly determine the winner of an election using two different types of voting systems:

1. First past the post
2. Preferential voting

Implementation of the first past the post solution will give them an opportunity to familiarise themselves with the implementation of an algorithm based on a relatively simple set of requirements.

Provide an outline or plan for students to use and tick off steps (algorithm) as they complete them, these may include:

* Devise a vote and agree on options
* Ask a group to vote on the options provided
* Record the votes
* Count up and tally the votes for each option
* Order the voting options
* The option with the most votes wins.
* Announce the winner.

## From there, students who develop confidence working with Python and with algorithm design can independently pursue development of the preferential algorithm, while support may be necessary to have all students develop an appropriate solution that takes into consideration any additional needs.



**Provide multiple means of engagement**

1. Ideally, students should create a program using Python code to calculate election results from a comma separated file.

2. Students could instead use a different programming language or a program like Microsoft Access or Excel to do the tallying.

3. Alternatively, you could use a paper-based method with tallying votes by hand to demonstrate how voting was done before computers.

## Learning construction

1. Explain to students that for the purposes of our program, the votes will have already been cast by voters and that the data has been stored in a comma separated text file, where each line represents a single vote with candidates listed in order of preference. Votes have been de-identified already so we don’t need to worry about that stage – we are simply going to use the data stored in the text file to work out who the winner of the election is.
2. Tell students that their first task is to break down the steps that a program will need to perform to determine the winner in a 'first past the post' style election.

One easy way to represent the steps involved is to use a flowchart. Have them produce their flowchart on paper with a pencil (so it can be easily changed) in landscape orientation.

Encourage students to think carefully about what the program needs to do, and try to be as explicit about each step as possible. You may want to consider giving them the 'starter' flowchart provided, but make it clear to them that they will need to provide more detail for it to be useful.

1. Consider walking them through the following process as a guide.   
   Tell them:
   * 'There are a lot of assumed steps missing from the flow chart. Let’s have a think about what’s missing from the part where we read from the file.
   * At no point do we specify that we are going to be reading the file a line at a time (which makes sense since each vote is on a single line). Let’s add that, and change our loop slightly so that we repeat the part where we read a line, then check to see if it is the end of the file.
   * Next, we say that we will store the first preference in the votes list, but we don’t actually have a votes list yet, so we’ll make that at the start of the program.
   * Before we can put the value in the list, we need to extract it from the line. This means that we have to split the line up at the commas, and then store the first value.
   * When we store the vote, we’ll be appending it to the list so that it gets added to the other votes we have already seen.
   * Now we’ve got a more clearly defined process for reading the contents of our file!'

They should then end up with something that looks more like the 'starter 2' flowchart. If they need suggestions about what other things need more detail, ask them how they will count the number of votes for each person, and how they will then work out which number of votes is the largest. Both of these things will need to be expanded.

* + There are going to be variations that students come up with that will still yield the correct result - there is no single 'perfect' answer. In the end though, it should provide detail similar to that presented in the 'final' flowchart provided.
  + Note that the final version is not the most efficient or 'best' solution (**it doesn’t cope with a draw**) so that’s an interesting point of discussion you can have with students when/if you show them this answer.
  + Your students might come up with a better algorithm, and you should encourage them to solve the draw problem (the solution requires adding the winner(s) to a list by matching all candidates who have the highest number of votes, then printing each winner’s name out).

1. The final step is to convert their algorithm into code. The sample *voting\_fpp* py file provided shows you how the algorithm expressed as a flowchart is converted into Python code.
   * Have students start their .py file by writing out the steps of their algorithm as code comments in the appropriate order, then translating these into Python instructions.
   * A correctly formatted *votes\_FINAL* sample file has also been provided, with the winner of the first past the post style election being Candidate Green.

 **Provide multiple means of engagement**

Instead of digital programming, the students can role play a voting scenario where the “program” has to decide what to do when each student votes. The “program” could just be the paper-based flowchart that you and the students step through as each student votes.

1. Students can then progress on to the preferential voting algorithm. Two sample flowcharts and solutions have been provided on the online lesson plan on the DT Hub: First past post: Flowcharts and Preferential: Flowcharts
   * one set where the algorithm uses 'buckets' to count votes (which is most analogous to the actual method used to count votes) then redistribute them as candidates are eliminated
   * the other which tracks eliminated candidates and processes each vote independently every round.

Both are acceptable approaches.

1. As in the previous example, the provided algorithms do not cope with a draw, but this time this is even more important – draws in earlier rounds that are not dealt with properly can lead to different results, and you’ll see this in action by running the code provided.  
   The fix to this is to ensure **all**candidates with the lowest vote are eliminated each round (not just one of them), and to check that if elimination of all candidates occurs in any round, the election should be declared a draw.

Resources: Refer to these Python files on the online lesson plan on the DT Hub: Preferential: code, First past post: Code, Preferential: code (buckets)

## Learning demo

1. When students have completed their own voting programs, have them come up with their own set of votes (ie their own votes.txt file) that they can share with other people in the class to test each other’s algorithms.
   * They should always get the same answer, even if their code is different.
   * When two students get different answers, the challenge then becomes working out whose algorithm is performing correctly, what the problems are with the incorrect one and how they can be corrected.
2. As a final reflection and consolidation activity, ask them to write a 300–500 word response to this statement:

The voting systems used in Australian elections at all levels of government can be defined in algorithmic terms, so the counting of votes can be performed easily and reliably using computer software. However, modern elections still take place using paper ballots and secret ballot boxes across the country, where people present physically to the polling place in their local community. What are some of the reasons, both technical and social, that might prevent Australia from moving to a fully online, electronic voting system?

1. Have them share their responses with other students through either an online collaboration tool (such as Google Docs or Padlet), a Post-it thought wall or in small-group discussions. Compile a class list of possible reasons based on their responses.

## Learning reflection

Students reflect on their learning in this activity using the *Triangle-Circle-Square* strategy where they:

* Triangle: identify three important points they have discovered from the lesson (these can be about any aspect of the activity they choose)
* Square: write down anything that 'squares' with their thinking (ie that they agree with). This might be a good way of consolidating their thinking around the merits and risks of voting electronically
* Circle: what thoughts are still 'circling' in their head? Are there any questions they still have about either writing and implementing algorithms, or about voting generally?

You can use the responses from this activity to identify any areas that might need extra attention in future lessons, and to get a sense of how far students have progressed in their understanding.

## Resources

* Australian Electoral Commission: 'Counting your vote' interactive
* Australian Electoral Commission: 'What’s your vote worth?' resource
* Australian Electoral Commission: 'Blackline Master 1' (zip) resource
* Python file: Voting\_FINAL
* Python file: Preferential: code
* Python file: Preferential: code (buckets)
* Python file: First past post: Code
* Sample flowchart: First past post
* Sample flowchart: Preferentials
* Triangle-Circle-Square reflection piece

## Curriculum links

| Links with Digital Technologies Curriculum Area | |
| --- | --- |
| **Strand** | **Content Description** |
| **Processes and Production Skills** | Acquire, store and validate data from a range of sources using software, including spreadsheets and databases (AC9TDI8P01) .  Define and decompose real-world problems with design criteria and by creating user stories (AC9TDI8P04) .  Design algorithms involving nested control structures and represent them using flowcharts and pseudocode (AC9TDI8P05) .  Evaluate existing and student solutions against the design criteria, user stories and possible future impact (AC9TDI8P10) . |

## Assessment

### Formative assessment

The way the activities have been structured provides ample opportunities for formative assessment while the learning takes place. The list below identifies the various stages in the activities where student progress can be evaluated. Note that this does not need to be a formal process, but feedback to students is always encouraged and these opportunities naturally occur in the lesson sequence.

* Student responses to teacher prompts
* Correctness and detail present in flowcharts
* Ability to articulate requirements of software explicitly throughout the design and implementation process
* Translation of algorithm design into code during implementation
* Student progress and engagement after direct instruction or intervention during program development
* Peer collaboration
* Thoroughness of testing and correctness of algorithm when program is executed
* Depth of thinking and analysis associated with exploration of electronic voting
* Contributions to class collaborative activities
* Reflection on learning through the Triangle-Square-Circle activity

Where students are struggling to articulate their progress towards the learning goals, teachers should consider intervention strategies that will support the students to overcome sticking points, or to address any skill or knowledge gaps the student may have.

### Summative assessment rubric

Teachers should always present summative assessment rubrics that are tailored to meet the specific learning goals identified for any assessment item. These should be explicit, identify the specific knowledge, understanding and skills that should be demonstrated, and use measurable outcomes and language that students can understand.  
The rubric below is not meant to be used as a task rubric – it has been produced to assist teachers to determine the appropriate level of knowledge and understanding of the key concepts presented in the activities for student achievement at each grade level. This should be used alongside the Australian Curriculum achievement standards and any school or jurisdiction guidelines for summative assessment purposes.  
Think of the depth of knowledge and understanding as being a continuum – students who are operating at a relational level, for example, will also be able to demonstrate the multi-structural knowledge acquisition presented.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Depth of knowledge** | | | **Depth of understanding** | |
| **Criteria** | **Pre-structural** | **Uni-structural** | **Multi-structural** | **Relational** | **Extended abstract** |
| **Defining problems and solutions** | States the problem being addressed and that solutions can be created using software | Understands that to create a digital solution to a problem you must first define that problem in clear terms | Can identify how a suggested solution addresses specific aspects of the problem that has been defined | Explains the role each aspect of the solution plays in solving the problem, and any interdependence between them | Draws on their knowledge of the solution and the context in which is it used to identify constraints, limitations and indirect consequences of its implementation |
| **Algorithm design** | States the algorithm in very broad and ambiguous terms | Is able to provide a clear sequence of steps that perform the algorithm, but does so in broad terms | Identifies the appropriate sequence of steps in a correct algorithm with sufficient detail to ensure it can be repeated | Design of the algorithm is correct, unambiguous and uses appropriate metalanguage to describe steps in the process | Can generalise the algorithm such that it works for varied data and rare or exception cases, and is applicable in additional contexts |
| **Algorithm implementation** | Attempts are made to implement the algorithm but significant assistance is required to do so | Can implement some aspects of the algorithm but needs assistance to structure code correctly and manage control structures | Can translate the main elements of the algorithm into code with minimal assistance and generate correct answers with common data. Comments are added to the code to explain steps | Correct translation of the algorithm with no assistance. Generates correct answers from data sets that are unfamiliar or uncommon. Code is documented | Algorithm implementation is well structured and minimises repetition. Code deals with rare / exception cases and is well documented |
| **Interactions and impact** | States that computer programs are beneficial in unspecific, broad terms | Is able to state examples of the impact and use of the algorithm covered in class | Uses general terms to describe how the algorithm is beneficial and can be used in other, familiar contexts | Draws on personal experience to explain both benefits and potential risks as a direct result of the program / algorithm | Uses evidence and experience to present the positive and negative social and technical implications of the program / algorithm in a variety of contexts |
| **Approximate grade level** | E | D | C | B | A |