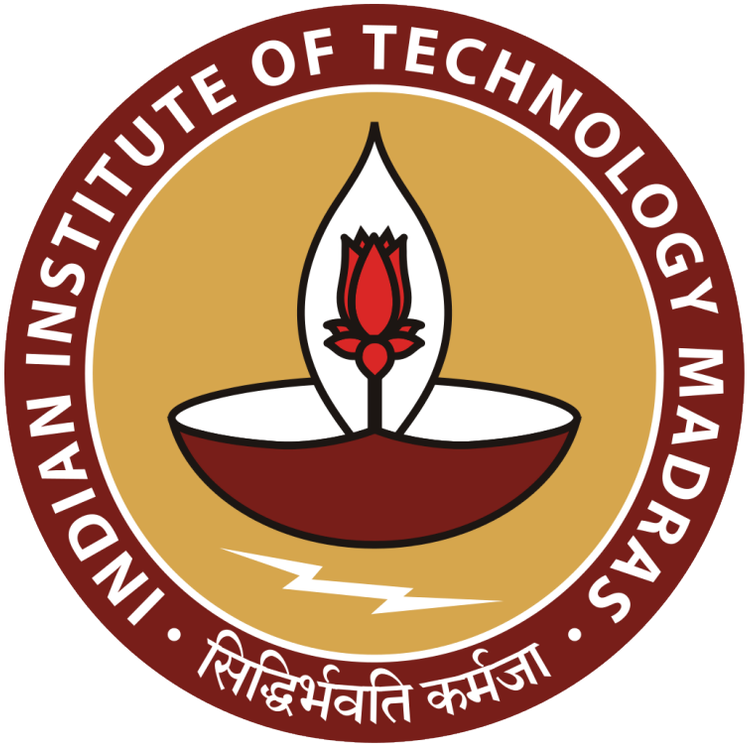


IIT Madras

ONLINE DEGREE

Summary of concepts introduced in weeks 1-4

Iterators and Variables

- The **iterator** is the most commonly used pattern in computational thinking
- Represents the procedure of doing some task repeatedly
  - requires an initialisation step,
  - the steps for the task that needs to be repeated,
  - and a way to determine when to stop the iteration

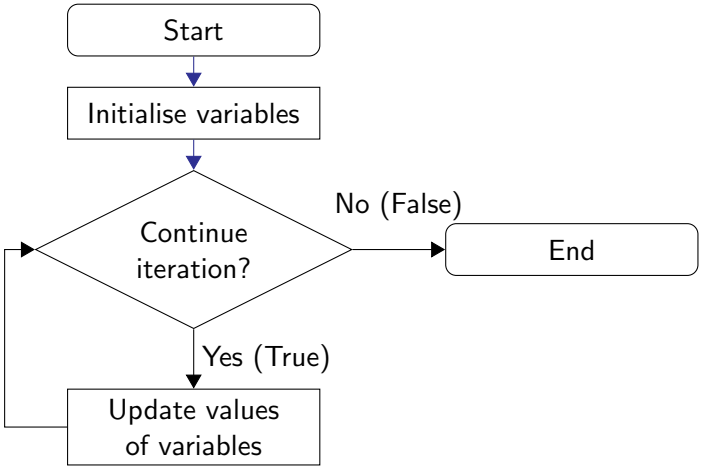
Iterators and Variables

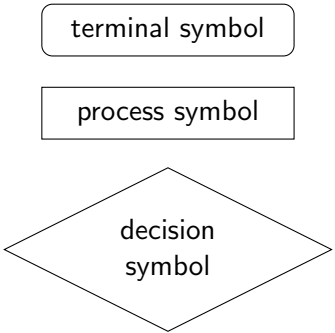
- The **iterator** is the most commonly used pattern in computational thinking
- Represents the procedure of doing some task repeatedly
  - requires an initialisation step,
  - the steps for the task that needs to be repeated,
  - and a way to determine when to stop the iteration
- **Variables** keep track of intermediate values during the iteration
  - Variables are given starting values at the initialisation step
  - At each repeated step, the variable values are updated

Iterators and Variables

- The **iterator** is the most commonly used pattern in computational thinking
- Represents the procedure of doing some task repeatedly
  - requires an initialisation step,
  - the steps for the task that needs to be repeated,
  - and a way to determine when to stop the iteration
- **Variables** keep track of intermediate values during the iteration
  - Variables are given starting values at the initialisation step
  - At each repeated step, the variable values are updated
- Initialisation and updates of variables are done through **assignment statements**

Iterator represented as a flowchart

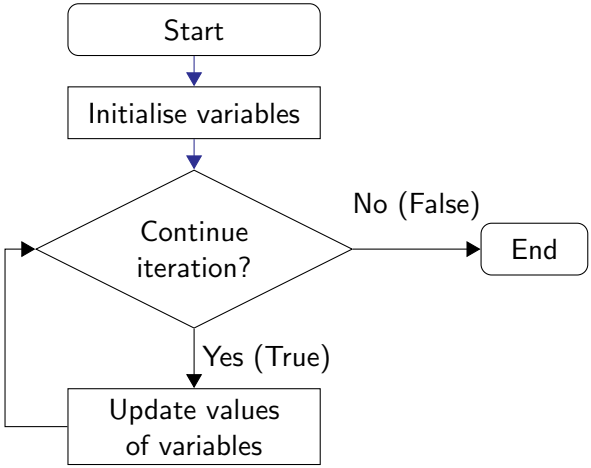




$\longrightarrow$ show progress from one step to another

Iteration expressed through pseudocode

```
Initialise variables
while (Continue with Iteration?) {
  Update values of variables
}
```



Iteration to systematically go through a set of items

Initialise variables

while (Pile 1 has more cards) {

  Pick a card **X** from Pile 1

  Move **X** to Pile 2

Update values of variables

}

The set of items need to have well defined values

- Sanity of different data fields of the item

  ... leads us to the concept of **datatypes**, which clearly identifies the values and allowed operations
- Basic data types - **boolean, integer, character**
- Add to this **string** data type
- **Subtypes** put more constraints on the values and operations allowed
- Lists and Records are two ways of creating bigger bundles of data
- In a **list** all data items typically have the same datatype
- Whereas, a **record** has multiple named fields, each can be of a different datatype

Iteration with Filtering

- **Filtering** makes a decision at each repeated step whether to process an item or not
- This introduces a decision step within the iteration loop

Iteration with Filtering

- **Filtering** makes a decision at each repeated step whether to process an item or not
- This introduces a decision step within the iteration loop
- Expressed in pseudocode, it would look something like this:

```
Initialise variables
while (Continue with Iteration?) {
  ...
  if (condition is satisfied?) {
    Update some variables
  }
  ...
}
Prepare final results from variable values
```

Iteration with Filtering

- **Filtering** makes a decision at each repeated step whether to process an item or not

Iteration with Filtering

- **Filtering** makes a decision at each repeated step whether to process an item or not
- The filtering condition can compare the item values with a constant

Iteration with Filtering

- **Filtering** makes a decision at each repeated step whether to process an item or not
- The filtering condition can compare the item values with a constant

  $\Rightarrow$The filtering condition does not change after each iteration step (is constant)

Iteration with Filtering

- **Filtering** makes a decision at each repeated step whether to process an item or not
- The filtering condition can compare the item values with a constant

  $\Rightarrow$The filtering condition does not change after each iteration step (is constant)
  - Example: Count, Sum

Iteration with Filtering

- **Filtering** makes a decision at each repeated step whether to process an item or not
- The filtering condition can compare the item values with a constant
  - $\Rightarrow$The filtering condition does not change after each iteration step (is constant)
  - Example: Count, Sum
- Or, it could compare item values with a variable

Iteration with Filtering

- **Filtering** makes a decision at each repeated step whether to process an item or not
- The filtering condition can compare the item values with a constant

  $\Rightarrow$The filtering condition does not change after each iteration step (is constant)
  - Example: Count, Sum
- Or, it could compare item values with a variable

  $\Rightarrow$The filtering condition changes after an iteration step

Iteration with Filtering

- **Filtering** makes a decision at each repeated step whether to process an item or not
- The filtering condition can compare the item values with a constant

  $\Rightarrow$The filtering condition does not change after each iteration step (is constant)
  - Example: Count, Sum
- Or, it could compare item values with a variable

  $\Rightarrow$The filtering condition changes after an iteration step
  - Example: max

Procedures and parameters

- Sometimes we have to write the same piece of code again and again with small differences

Procedures and parameters

- Sometimes we have to write the same piece of code again and again with small differences
- A piece of pseudocode can be converted into a **procedure** by separating it out from the rest of the code

Procedures and parameters

- Sometimes we have to write the same piece of code again and again with small differences
- A piece of pseudocode can be converted into a **procedure** by separating it out from the rest of the code
- Some variables (or constants) used in this piece of code can be replaced by a **parameter** variable

Procedures and parameters

- Sometimes we have to write the same piece of code again and again with small differences
- A piece of pseudocode can be converted into a **procedure** by separating it out from the rest of the code
- Some variables (or constants) used in this piece of code can be replaced by a **parameter** variable
- Instead of writing the code again with a small difference, we now just have to make a **call** to the procedure with a different parameter value

Procedures and parameters

- Sometimes we have to write the same piece of code again and again with small differences
- A piece of pseudocode can be converted into a **procedure** by separating it out from the rest of the code
- Some variables (or constants) used in this piece of code can be replaced by a **parameter** variable
- Instead of writing the code again with a small difference, we now just have to make a **call** to the procedure with a different parameter value
- e.g. finding max for each subject

Accumulation through Iteration

- The most common use of an iterator is to create an aggregate value (**accumulation**) from the available values
- Simple examples of this are count, sum, average
- We could also apply filtering while doing accumulation - e.g. sum of boys marks
- We could also collect a list of elements - e.g. list of students with max marks in a subject

Doing two iterations - one after another

- Use the first iteration to do some accumulation
- The variables in which these accumulations are done can be called accumulators

Doing two iterations - one after another

- Use the first iteration to do some accumulation
- The variables in which these accumulations are done can be called accumulators
- Second iteration can do filtering using the accumulator variables
- e.g. find above average students - average is an accumulator from the first iteration

Doing two iterations - one after another

- Use the first iteration to do some accumulation
- The variables in which these accumulations are done can be called accumulators
- Second iteration can do filtering using the accumulator variables
- e.g. find above average students - average is an accumulator from the first iteration
- This establishes a relationship between any element and the aggregate of all elements
- e.g. find out the more frequently occuring word, higher spending customers, etc

Doing two iterations - one nested within another

- If we need to go beyond the relationship between an element and the aggregate of all elements

Doing two iterations - one nested within another

- If we need to go beyond the relationship between an element and the aggregate of all elements

  ... to expressing a relationship between any two elements

Doing two iterations - one nested within another

- If we need to go beyond the relationship between an element and the aggregate of all elements

  ... to expressing a relationship between any two elements
- We will need to do one iteration within another

Doing two iterations - one nested within another

- If we need to go beyond the relationship between an element and the aggregate of all elements

  ... to expressing a relationship between any two elements
- We will need to do one iteration within another
- e.g. find out if two students have the same birth day and month

Doing two iterations - one nested within another

- If we need to go beyond the relationship between an element and the aggregate of all elements

  ... to expressing a relationship between any two elements
- We will need to do one iteration within another
- e.g. find out if two students have the same birth day and month
- Nested iterations are costly in terms of number of computations required

Doing two iterations - one nested within another

- If we need to go beyond the relationship between an element and the aggregate of all elements

  ... to expressing a relationship between any two elements
- We will need to do one iteration within another
- e.g. find out if two students have the same birth day and month
- Nested iterations are costly in terms of number of computations required
- We could reduce the number of comparisons by using **binning** wherever possible

Doing two iterations - one nested within another

- If we need to go beyond the relationship between an element and the aggregate of all elements

  ... to expressing a relationship between any two elements
- We will need to do one iteration within another
- e.g. find out if two students have the same birth day and month
- Nested iterations are costly in terms of number of computations required
- We could reduce the number of comparisons by using **binning** wherever possible
- How do we store such relationships?

Doing two iterations - one nested within another

- If we need to go beyond the relationship between an element and the aggregate of all elements

  ... to expressing a relationship between any two elements
- We will need to do one iteration within another
- e.g. find out if two students have the same birth day and month
- Nested iterations are costly in terms of number of computations required
- We could reduce the number of comparisons by using **binning** wherever possible
- How do we store such relationships?
- To be discussed in the next 4 weeks