





# What is an Algorithm?

**Algorithm**  $\stackrel{\text{def}}{=}$  steps to result

-  Ordered
-  Unambiguous
-  Computable
-  Terminating: halt in finite time

**Thinking about the definition**  
Parts-whole: determine relationships

# What is an Algorithm? Thinking About Definition

## Whole-parts relationships

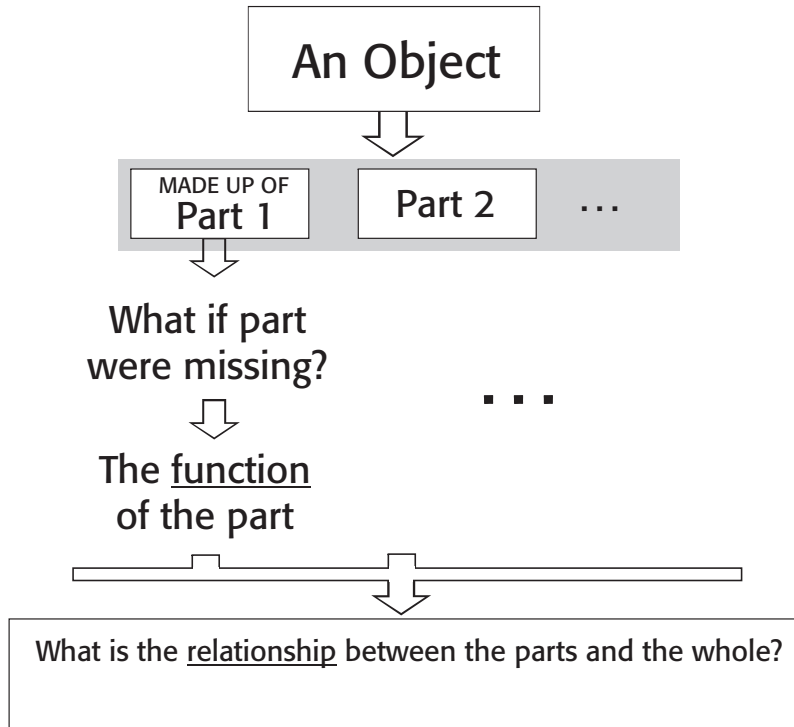
We are able to understand an object by identifying its parts and analyzing their roles in how the object works



### Example: hand

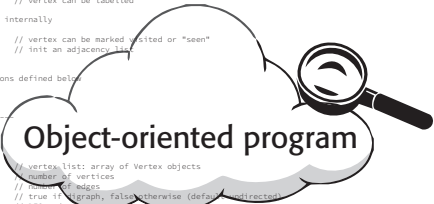
Parts: fingers, palm, nails, opposable thumb (?), ...

# Whole-Parts Thinking Map



# Second Exercise

```
// -----  
// Vertex object constructor  
function Vertex(v)  
{  
  // user input fields  
  this.label = v.label; // vertex can be labelled  
  // more fields to initialize internally  
  this.visit = false; // vertex can be marked visited or "seen"  
  this.adjacent = new List(); // init an adjacency list  
  // -----  
  // member methods use functions defined below  
}  
// -----  
// Graph object constructor  
function Graph()  
{  
  // vertex list: array of Vertex objects  
  this.vert = [];  
  this.nv; // number of vertices  
  this.ne; // number of edges  
  this.digraph = false; // true = graph, false otherwise (default = undirected)  
  this.dfs_order = []; // DFS order output  
  this.bfs_order = []; // BFS order output  
  this.label = ""; // identification string to label graph  
  // -----  
  // member methods use functions below  
  this.read_graph = better_input; // default input reader method  
  this.print_graph = better_output; // better printer function  
  this.add_edge = add_edge;  
  this.dfs = dfs; // DFS a connected component  
  this.bfs = bfs; // BFS a connected component  
}
```



Call operations on objects

Example: List



Execute a method

Example: .sort(), .reverse()

In CPCS-324, we determine all vertex pairs connected by a directed path in a digraph by calling a **method** of a graph object that implements an efficient algorithm to find the transitive closure of the graph.

We focus on the method



An algorithm provides a *method* for an object to behave or communicate (express) properties.

⇒ **Where do algorithms fit?**

JavaScript has C-like basic syntax (like java) and can directly run a procedure in a convenient setting.

⇒ **Programming component**


⇒ **Exercise walkthrough**



# Data Structures Review

⇒ **Algorithms process data**

⇒ **Where does DS fit?**

⇒  **Section 1.4: quick review on demand**

Revisit basic data arrangements and re-examine implications in terms of solutions to computing problems.



# Algorithmic Solution

- ⇒ **Problem instance**
- ⇒ **Pseudocode**

“ a good algorithm is a result of repeated effort and rework ”



## Development activities

Work small cases, id problem, and specify its **instances**.



**Understand problem**

Decisions include compu- tational device, solution type, data structure(s), and design strategy.



**Decide**



**Specify algorithm**



**Prove correctness**

Characterize **efficiency, simplicity and general- ity**.



**Analyze (determine characteristics)**



**Code solution**