Computer Science Principles

Unit 3: Algorithms and Programming

LECTURE 7: ALGORITHM AND LIBRARY (U6/U7)

DR. ERIC CHOU

IEEE SENIOR MEMBER

Topics

1. Variables and Assignments
2. Data Abstraction
3. Mathematical Expressions
4. Strings
5. Boolean Expressions
6. Conditionals
7. Nested Conditionals
8. Iteration
9. Developing Algorithms
10. Lists
11. Binary Search
12. Calling Procedures
13. Developing Procedures
14. Parts of a Procedure
15. **Libraries**
16. **Random Values**
17. **Simulation**
18. **Algorithmic Efficiency**
19. **Undecidable Problems**

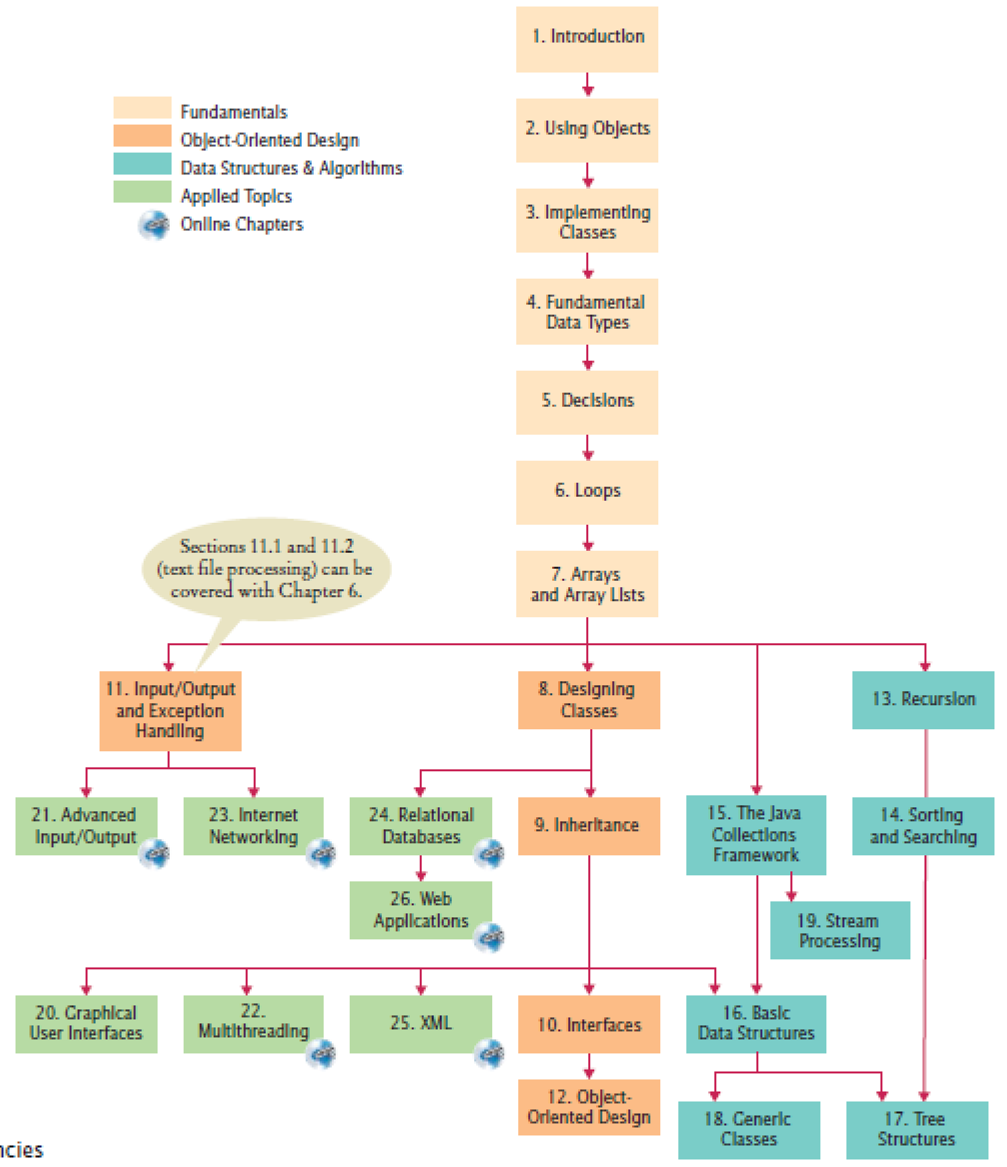
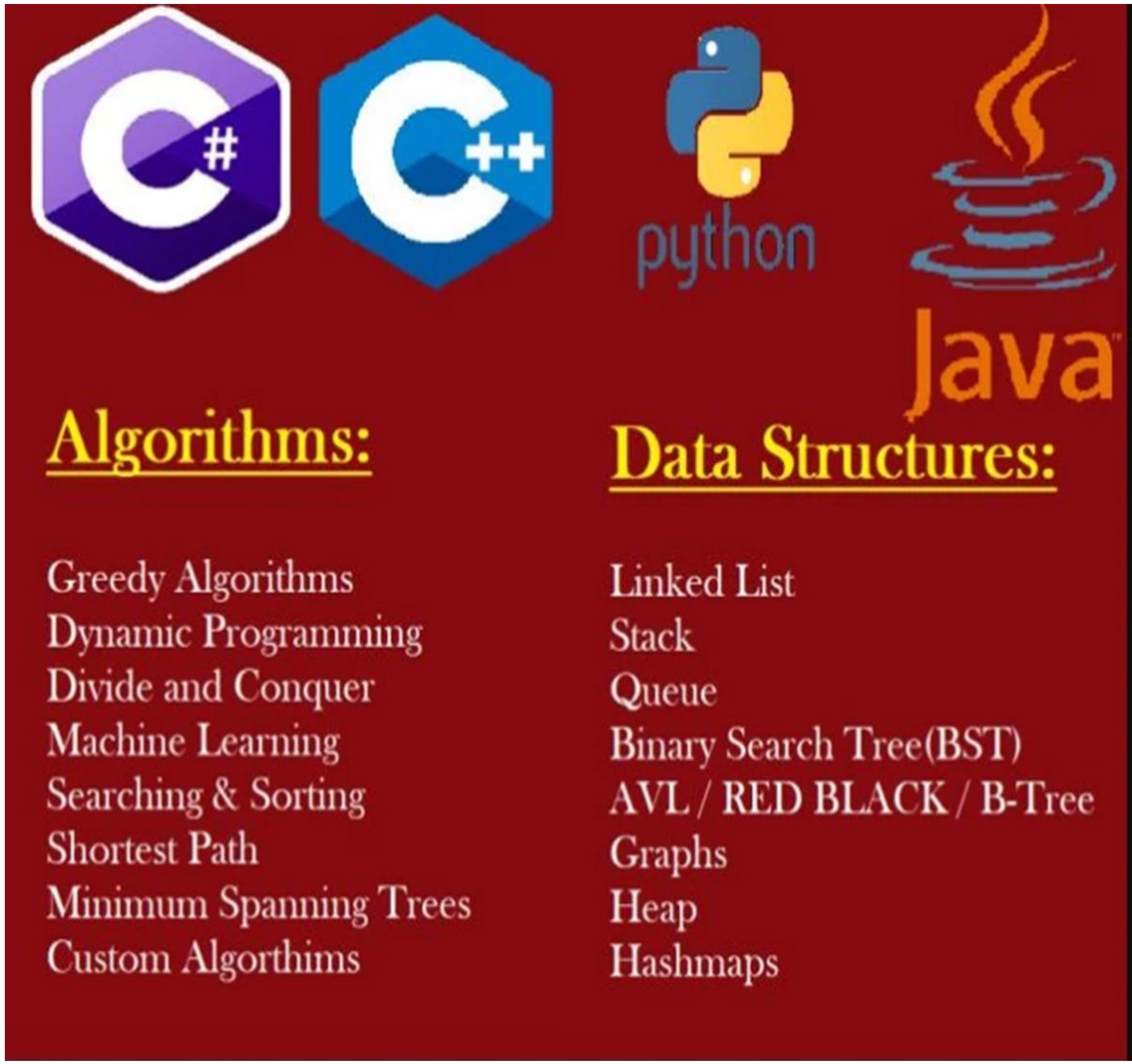

Fundamentals
Object-Oriented Design
Data Structures & Algorithms
Applied Topics
Online Chapters
1. Introduction
2. Using Objects
3. Implementing Classes
4. Fundamental Data Types
5. Decisions
6. Loops
7. Arrays and Array Lists
Sections 11.1 and 11.2 (text file processing) can be covered with Chapter 6.
11. Input/Output and Exception Handling
8. Designing Classes
13. Recursion
21. Advanced Input/Output
23. Internet Networking
24. Relational Databases
9. Inheritance
15. The Java Collections Framework
14. Sorting and Searching
26. Web Applications
19. Stream Processing
20. Graphical User Interfaces
22. Multithreading
25. XML
10. Interfaces
16. Basic Data Structures
12. Object-Oriented Design
18. Generic Classes
17. Tree Structures

Figure 1
Chapter
Dependencies


python
Java

Algorithms:

Greedy Algorithms
Dynamic Programming
Divide and Conquer
Machine Learning
Searching & Sorting
Shortest Path
Minimum Spanning Trees
Custom Algorthims

Data Structures:

Linked List
Stack
Queue
Binary Search Tree(BST)
AVL / RED BLACK / B-Tree
Graphs
Heap
Hashmaps

Learning Channel

Library

LECTURE 1

Libraries

- You don't have to define every procedure you use in a program. If you're using already-written code, you can import it into your program. In Python, you do this by importing the module it's contained in. This importing usually takes the form of a line of text at the top of your program that looks like this:

```
from location import module
```

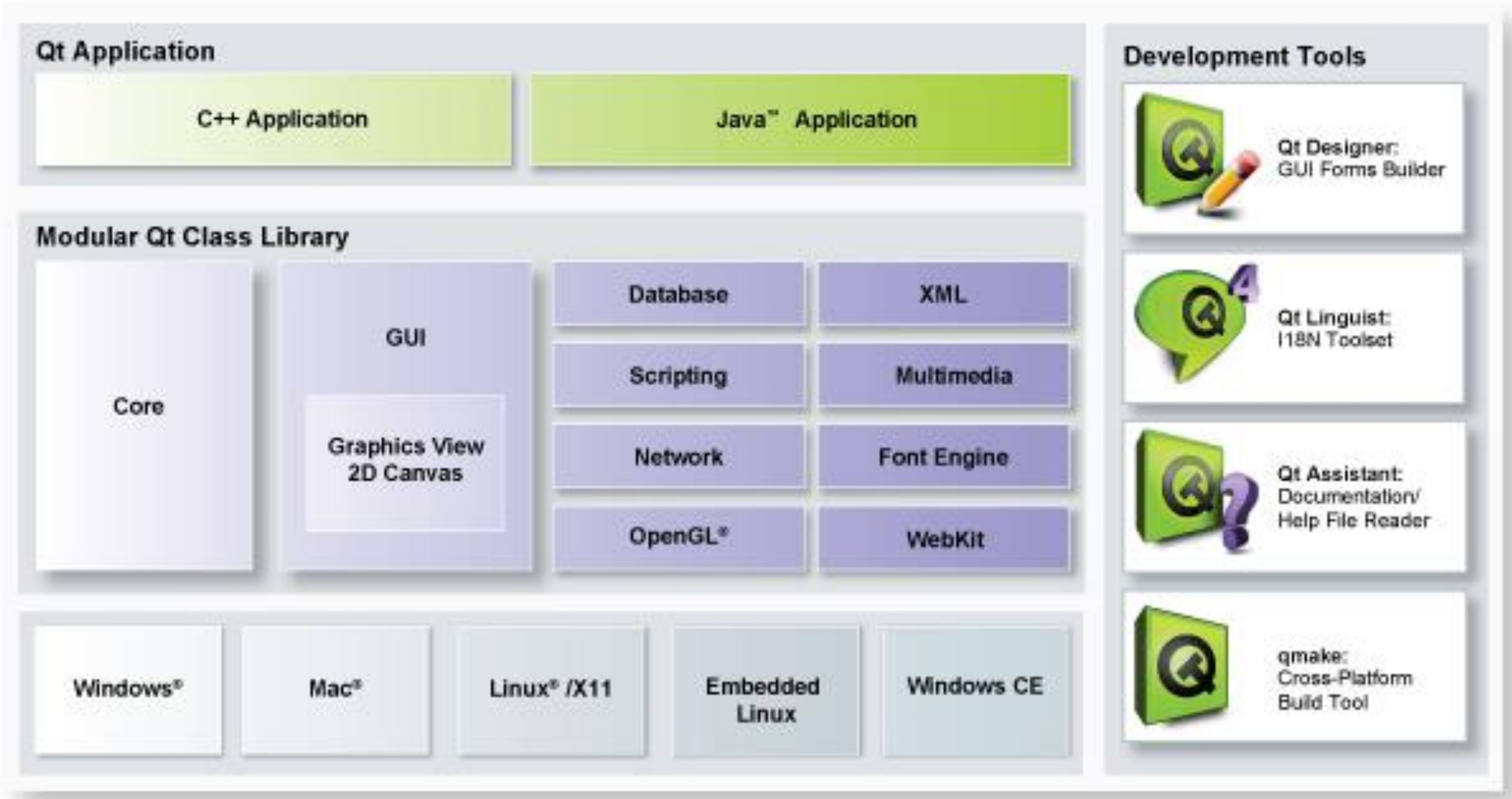
- There are several places where already-written modules come from.

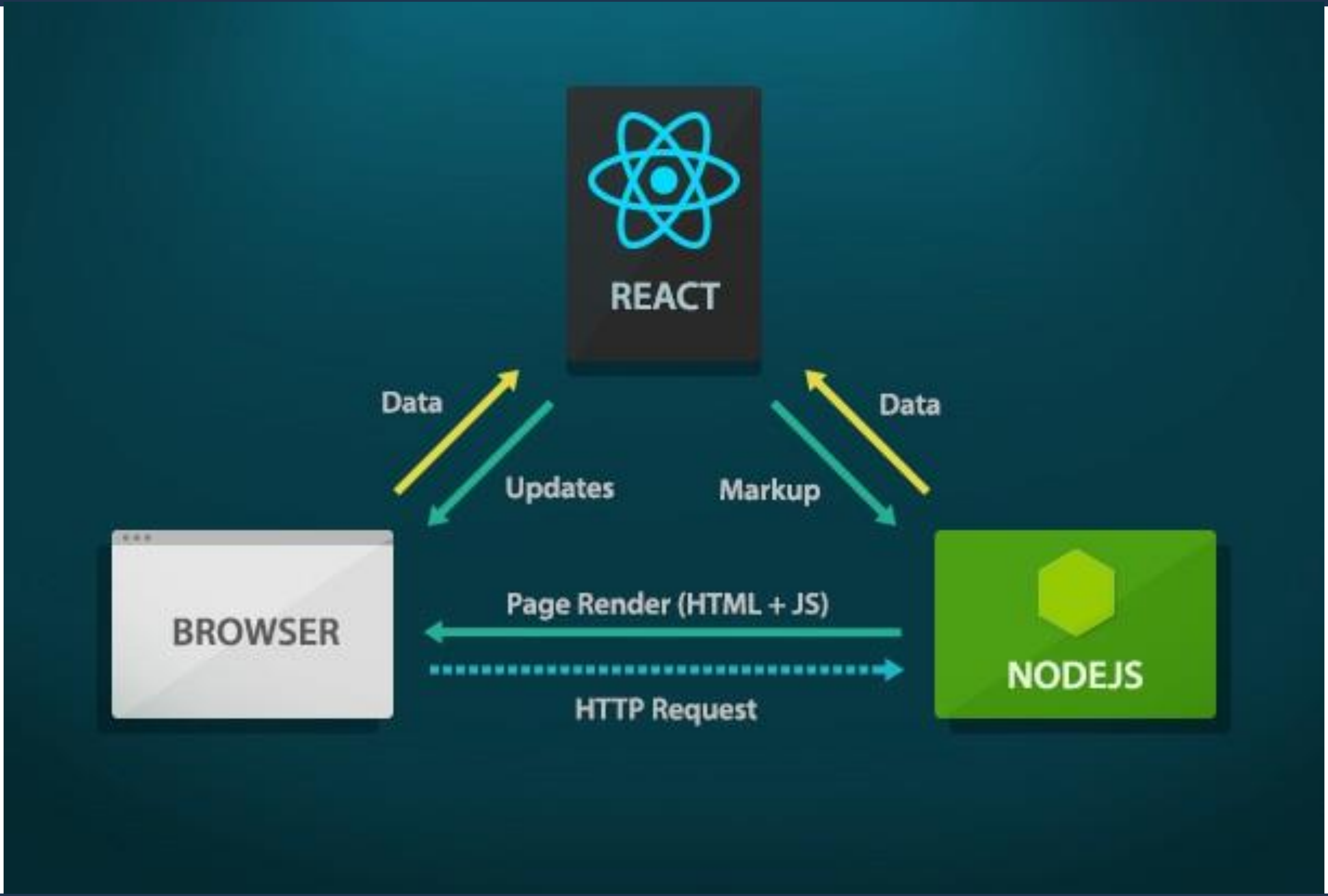
API Software Libraries

- A **software library** contains already-developed procedures that you can use in creating your own programs. You don't have to write new procedures to, for example, display images or find derivatives in your program; someone's already written those procedures and put them in a library for the public to use.
- There are libraries for everything under the sun. Here are some Python examples:
    - **Pillow** allows you to work with images.
    - **Matplotlib** allows you to make 2D graphs and plots.
- You can also **import** some of your own previously written code into a new program.

API Software Libraries

- An Application Program Interface, or **API**, contains specifications for how the procedures in a library behave and can be used. It allows the imported procedures from the library to interact with the rest of your code.
- In order to make the most of both APIs and Libraries, both need to be well documented. Libraries are, at their heart, a collection of other people's code, and it would be difficult to understand how the procedures should be used without documentation. APIs explain how two separate pieces of software interact with each other, and they also need documentation to keep this communication going.


Qt Application
C++ Application
Java™ Application
Modular Qt Class Library
Core
GUI
Graphics View 2D Canvas
Database
XML
Scripting
Multimedia
Network
Font Engine
OpenGL®
WebKit
Windows®
Mac®
Linux® /X11
Embedded Linux
Windows CE
Development Tools
Qt Designer: GUI Forms Builder
Qt Linguist: I18N Toolset
Qt Assistant: Documentation/ Help File Reader
qmake: Cross-Platform Build Tool


REACT
Data
Updates
Markup
Data
BROWSER
Page Render (HTML + JS)
HTTP Request
NODEJS

Python Flask Framework

- Setting Up the Environment
- Beginning with Flask
- Variable Rules
- URL Building
- HTTP Methods

Random Values

LECTURE 2

Generating Random Numbers

- Many coding languages provide a way to generate random numbers, and College Board's Pseudocode is no exception. It's a good tool for many programs.

Text: `RANDOM(a, b)` Block: RANDOM $\boxed{\texttt{a, b}}$	Generates and returns a random integer from `a` to `b`, including `a` and `b`. Each result is equally likely to occur. For example, `RANDOM(1, 3)` could return `1`, `2`, or `3`.

Python

```
import random
c = random.randint(a,b)
```

Notice how you have to import the random module into your program in order to gain access to the random generator.

Simulation

LECTURE 3

Simulations

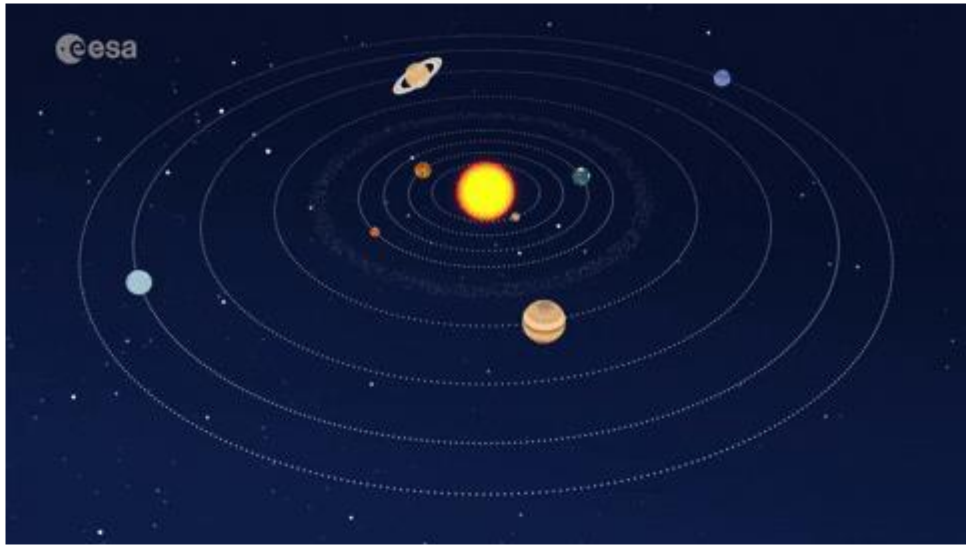
- **Simulations** are simplifications of complex objects (like the planets) or phenomena (like tornadoes) for a stated goal. They often use varying sets of values to reflect how a phenomenon changes. Using a computer, we can simulate everything from a **science lab** to a nuclear explosion to a zombie apocalypse, and computer simulations are used in industries like weather forecasting and financial planning.
- In order to develop a simulation, you have to remove certain real world details (like language barriers in a historical simulation event) or simplify how something functions.

Abstraction in Simulations

- Simplifying details to highlight a main point, where have we heard this before? 🤔
- That's right, simulations are an example of abstraction.
- There are many benefits to creating a simulation. They can be used to represent real-world events and conditions, like the force of gravity or the atmospheric conditions of a battle, so you can investigate and draw conclusions about them without dealing with some of the complications of the real world. Simulations are the most useful when observing the phenomenon in real life would be impractical, like if what you wanted to study was too big (Big Bang, continental drift) or too small (atoms, elements).

Abstraction in Simulations

- However, simulations also have some disadvantages. They run the risk of being too simple or conveying the wrong message about what you're trying to study (simulating the planets with tennis balls, for example, may lead people to think they're closer to each other and more similarly sized than they actually are.)



Abstraction in Simulations

- Simulations may also contain bias based on what the simulation creator chose to include or exclude.
- Random number generators can help simulate real-world variability in these simulations: it's a little like rolling a pair of dice.

Algorithm Efficiency

LECTURE 3

Undecidable Problems

LECTURE 4

Undecidable Problems

- A **decidable problem** is a decision problem (one that has a yes/no answer) where an algorithm can be written to produce a correct output for all inputs.
- If an algorithm can't be written that's always capable of providing a correct yes or no answer, it's an **undecidable problem**. An undecidable problem might be able to be solved in some cases, but not in all of them.
- The classic example of an undecidable problem is the halting problem, created by Alan Turing in 1936. The halting problem asks that if a computer is given a random program, can an algorithm ever be written that will answer the question, will this program ever stop running?, for all programs? By proving that there wasn't, Turing demonstrated that some problems can't be completely solved with an algorithm.

Algorithms Solve Problems

LESSON 1 [UNIT 6 CODE.ORG]

Algorithm

SECTION 1

The Need for Algorithms

LECTURE 1

Objectives

- The main purpose of the lesson is to connect the acts of writing "code" and designing algorithms, and to take some steps towards programming with code.
- **Algorithm** - A precise sequence of instructions for processes that can be executed by a computer

Problem Solving

Step 1	• Understand the problem • Identify program input and output
Step 2	• Design the solution (algorithm)
Step 3	• Writing a program in a programming language to match the algorithm steps

Human-Machine FindMin

ACTIVITY

Activity – Human/Machine

- In this activity you're going to pretend that you are a "**Human Machine**" that operates on playing cards on the table.
- 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.

Activity – Human/Machine

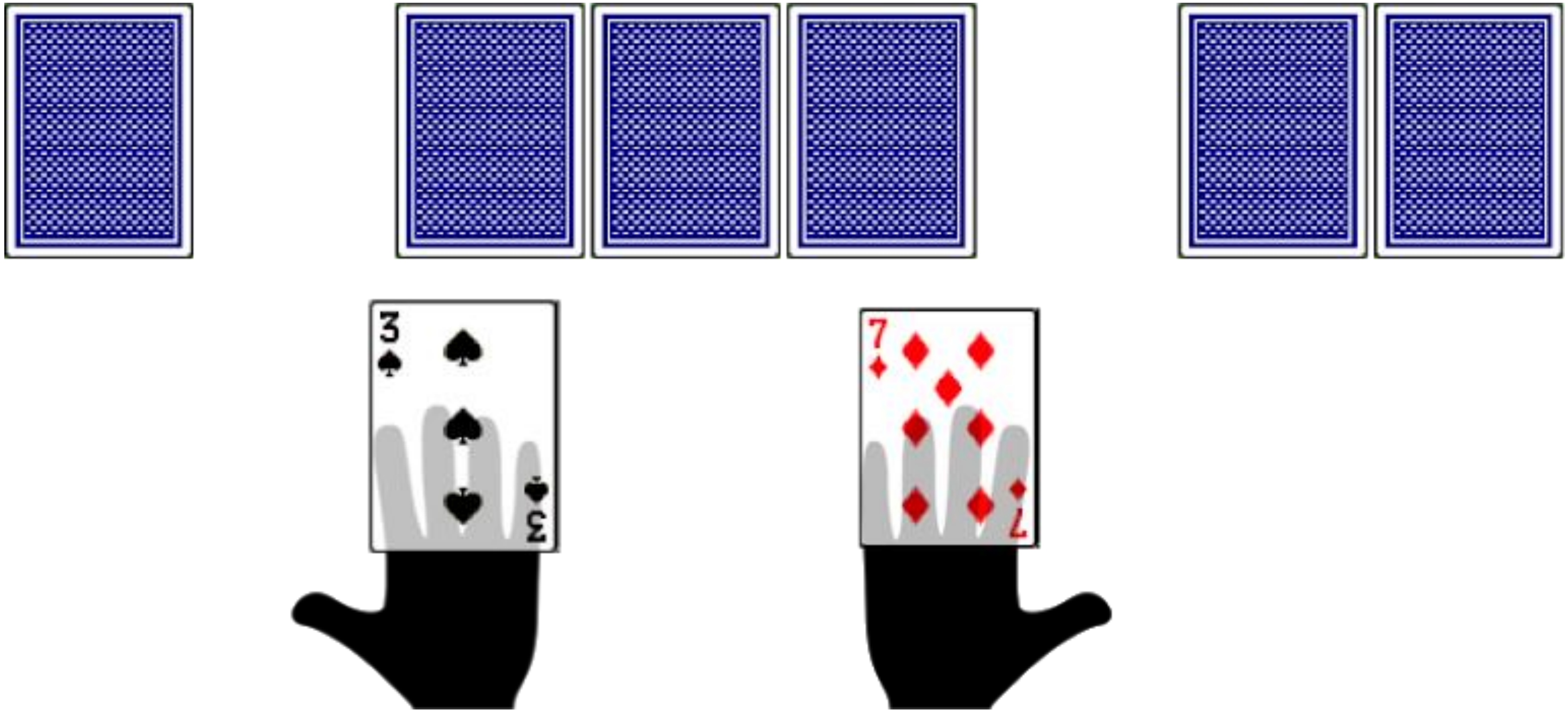
- In this activity, you'll design an algorithm to find the smallest item in a list. Obviously, if we were really writing instructions for a person, we could simply tell them: "find the smallest item in a list." But that won't work for a computer.
- We need to describe the process that a person must go through when they are finding the smallest item. What are they really doing?

Activity

- Everyone gets Minimum Card Algorithm - Activity Guide
- Each pair gets a deck of cards.
- Or you can use Online-Deck-of-Cards

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.


3
7

Task:

- **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.

Activity

- Get clear on the task, rules, instructions
- With a partner act out an algorithm
- Write down the steps

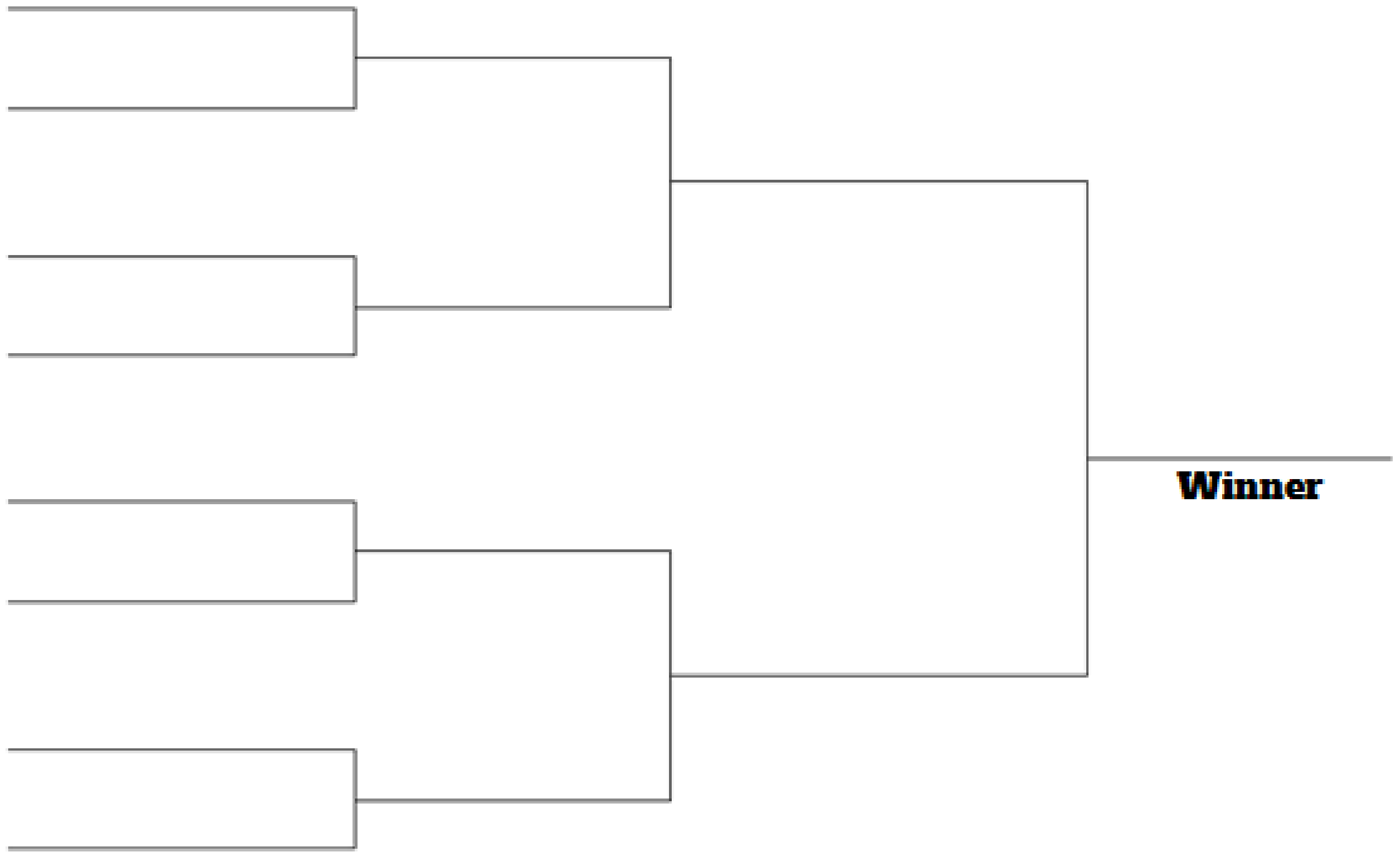
Discussion

- How do you know when to stop?
- Do your instructions state where and how to start?
- Is it clear where to put cards back down after you've picked them up?

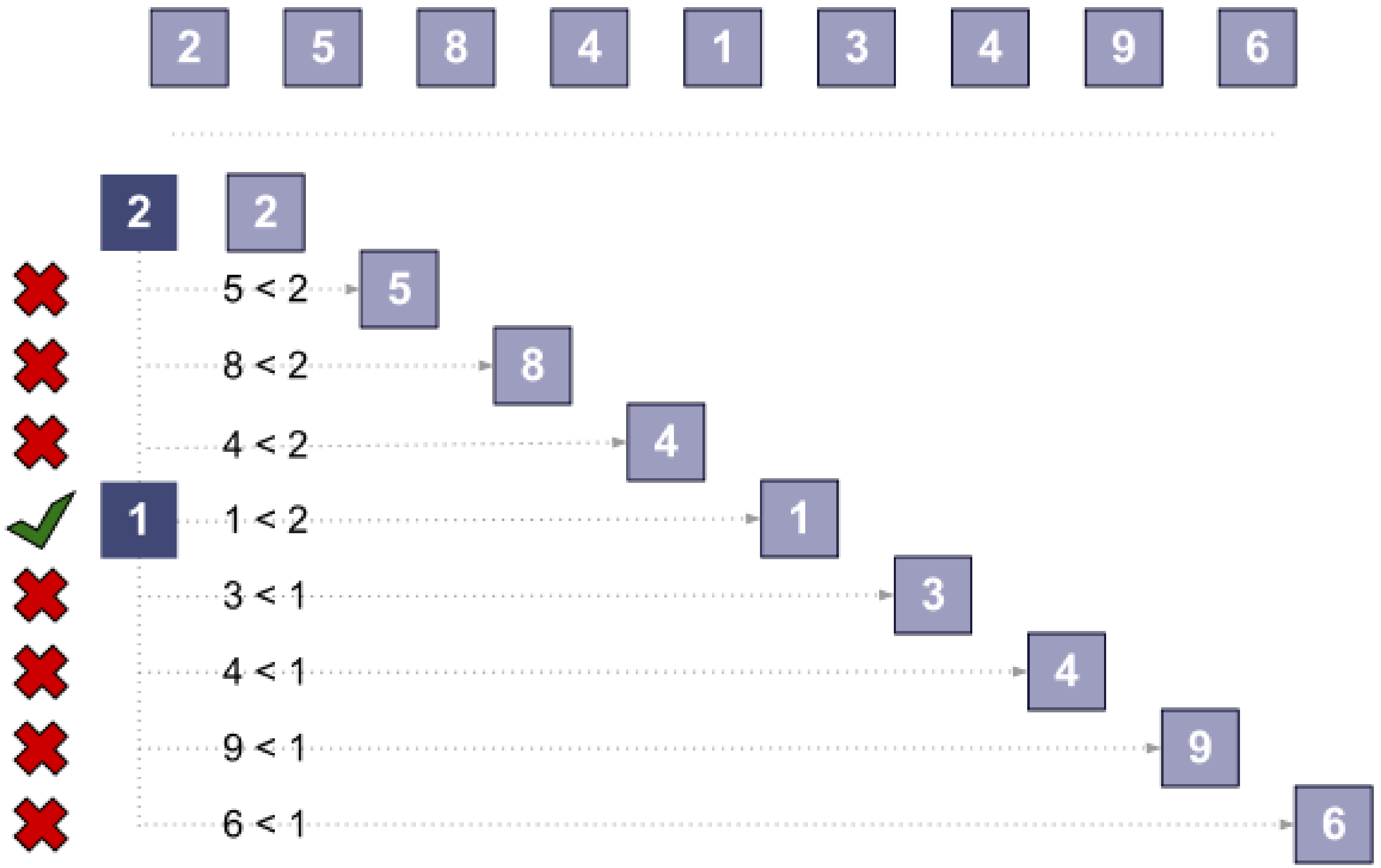
Discussion

- As we look at these algorithms you came up with, we can see they are not all the same.
- However, there are common things that you are all making the human machine do and commonalities in some of your instructions.
- Can we define a language of common Human Machine commands for moving cards around? What are the commands or actions most of these instructions have in common?"

8 Team Single Elimination



PrintYourBrackets.com



1 is the minimum

The “Human Machine” Language

ACTIVITY

Activity

- 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.

Commands for the Card Game

Did you have commands like:

- **SHIFT (hand)** - some form of shifting hands one position down the row left or right
- **MOVE (hand)** - some form of moving a hand directly to a particular card based on its position in the list or to the position of one of the other hands.
- **COMPARE** - some way to compare cards and do something based on the result like: "if card in right hand is less than card in left hand then...“
- **GO TO LINE** - some way to jump to an earlier or later line in the program
- **PICK CARD UP/PUT CARD DOWN** - some way to do this that also makes clear where to put a card back down. Typically something like: "Put right hand card down into the right-most open space in the row of cards" NOTE: we don't need this command for the next activity so just acknowledging the need is fine.

SHIFT *hand* TO THE *dir*

MOVE *hand* TO POSITION *num*

JUMP TO LINE *num*

JUMP TO LINE *num* IF *num* *comp?* *num*

STOP

Human Machine Language (Block Programming)

- Everyone gets Human Machine Language - Activity Guide
- Make pairs

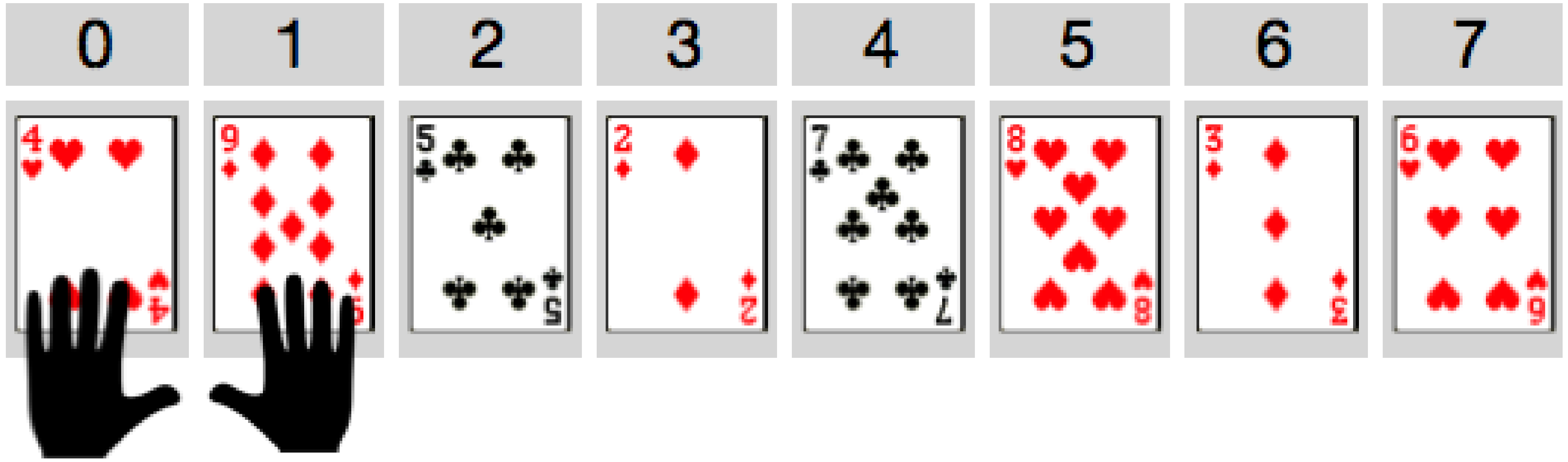
  Each pair gets at least 8 cards

Human Machine Language (I)

1. Introduce Human Machine Language
2. Read the first page.
3. Clarify the instructions and setup.
4. Get with your partner

Setup and Rules

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



Setup and Rules

- 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.

Human-Machine Language (II)

Execute the Example Programs

1. You should try to figure out the example programs with your partner, making use of the code reference guide.
2. One partner reads, the other acts as the human machine.
3. Jot notes about what the program does.

Human-Machine Language (III)

Verify that you get the gist of how the language works.

1. For each example can you describe what it did.
2. Review the last example which had a problem.
3. Does everyone understand the problem?
4. Do you have a solution?

FindMin

- This problem we identified with the last example speaks to the art and science of algorithm design and what can make it so interesting.
- The question is: can we fix the problem without adding more commands to the language? Yes.
- If we can fix a problem without extending the language, that's a good thing. We can focus our attention on designing algorithms given these constraints.
- Let's try to write **Challenge**: FindMin using the Human Machine Language...

Challenge: Find Min with the Human Machine Language

- First identify what's different about the problem setup for the Human Machine Language:
  - All cards are face up
  - Card positions have numbers
  - Don't need to pick up cards or put them down
    There is actually no way to move cards at all - only hands
  - The ending state is well defined - left hand touching the min card.
  - Now use the Human Machine Language to write the algorithm for finding the minimum number of cards.

Activity

- You can just write the code, or you can use the cutout strips of the commands and write values into the boxes.
- Take some time to work things out with your partners.
- It may take some time to get oriented and understand the task.

Activity

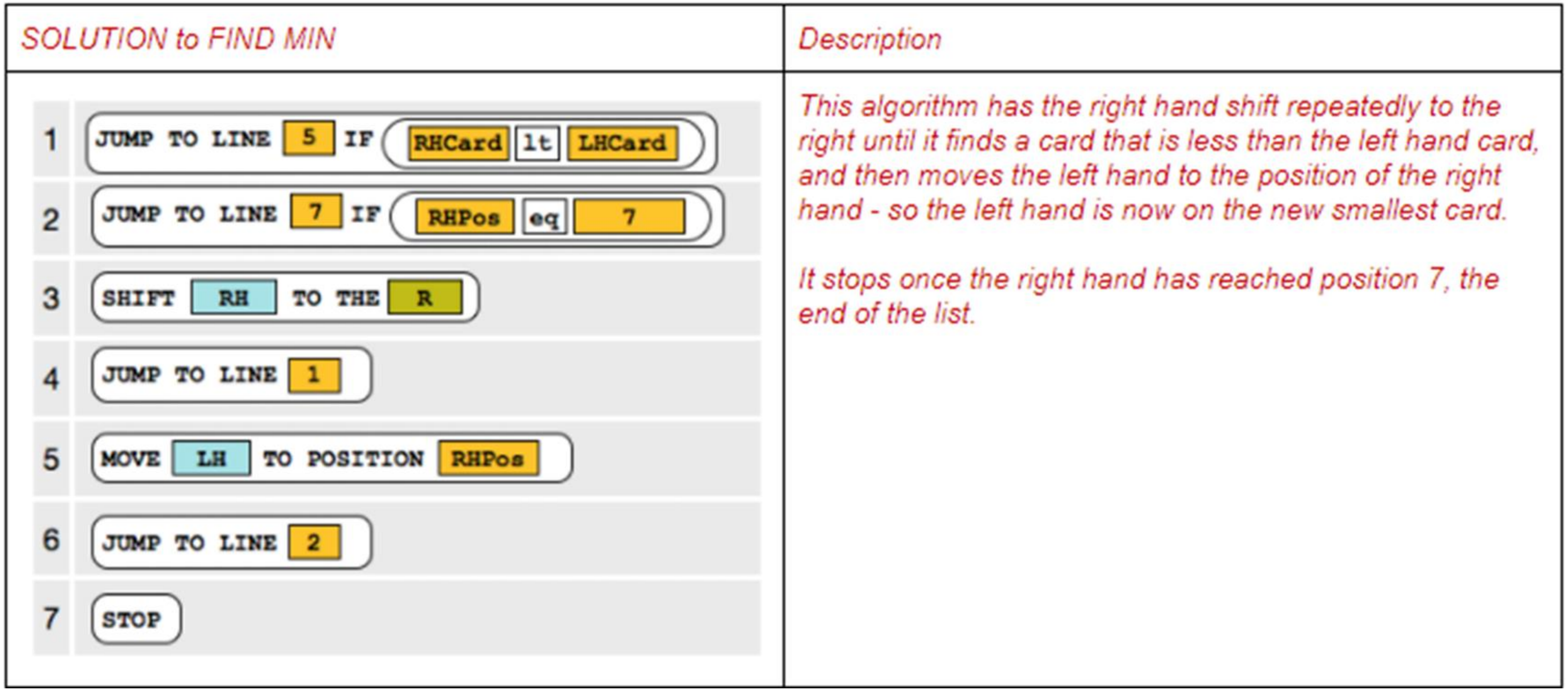
- Put pairs together to compare solutions.
- Each group should test out the other group's code by acting as the human machine.
- During the comparison note any differences in your classmates’ approach.
- There are several ways to go about it

Programming

Putting Algorithms to Block Programming Language

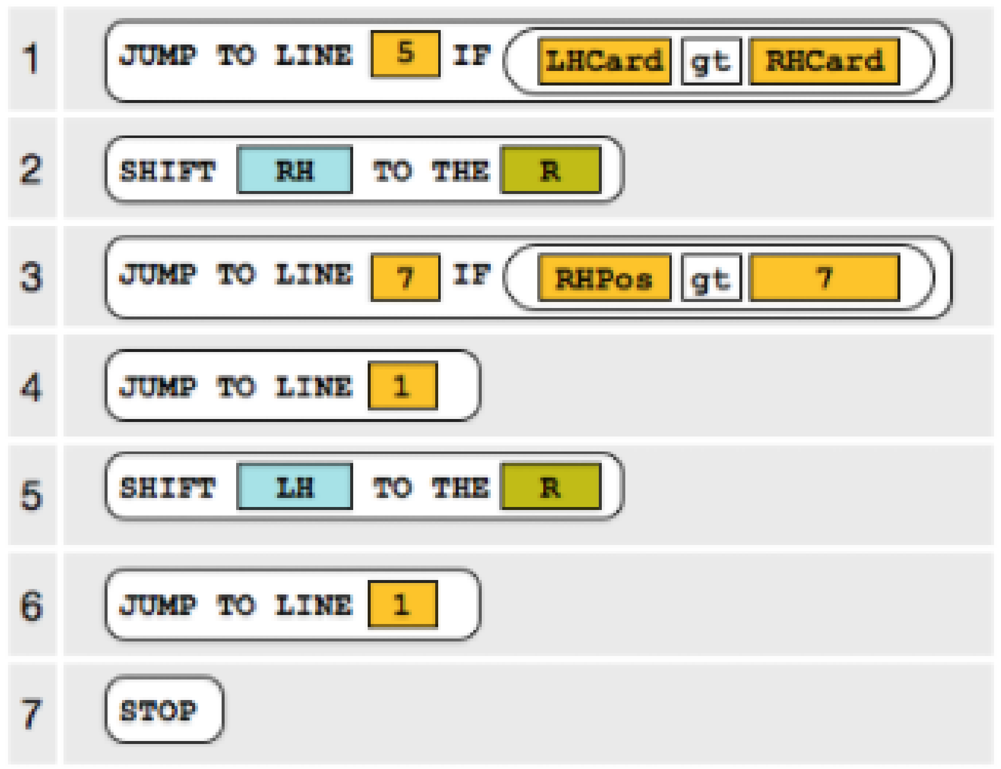
ACTIVITY

Put Algorithm to Block Program



SOLUTION to FIND MIN	Description
1 JUMP TO LINE 5 IF (RHCard lt LHCard) 2 JUMP TO LINE 7 IF (RHPos eq 7) 3 SHIFT RH TO THE R 4 JUMP TO LINE 1 5 MOVE LH TO POSITION RHPos 6 JUMP TO LINE 2 7 STOP	This algorithm has the right hand shift repeatedly to the right until it finds a card that is less than the left hand card, and then moves the left hand to the position of the right hand - so the left hand is now on the new smallest card. It stops once the right hand has reached position 7, the end of the list.

Put Algorithm to Block Program



```
1  JUMP TO LINE 5 IF (LHCard gt RHCard)
2  SHIFT RH TO THE R
3  JUMP TO LINE 7 IF (RHPos gt 7)
4  JUMP TO LINE 1
5  SHIFT LH TO THE R
6  JUMP TO LINE 1
7  STOP
```

This algorithm takes an interesting (perhaps unconventional) approach.

It repeatedly shifts the right hand to the right until it finds a card that is less than the card in the left hand, and then repeatedly shifts the left hand until it's touching the right hand. Then it repeats that process until the right hand moves past the end of the row.

The way it shifts the left hand involves a clever piece of logic on line 1: as long as the left hand card is strictly greater than the right hand card, shift right.

Code Studio

- Fill out assessments and reflections in Code Studio.

Creativity in Algorithms

LESSON 3

Objectives

- The purpose of this lesson is to see what "creativity in algorithms" means.

Vocabulary

- **Algorithm** - A precise sequence of instructions for processes that can be executed by a computer
- **Iterate** - To repeat in order to achieve, or get closer to, a desired goal.
- **Selection** - A generic term for a type of programming statement (usually an if-statement) that uses a Boolean condition to determine, or select, whether or not to run a certain block of statements.
- **Sequencing** - Putting commands in correct order so computers can read the commands.

Vocabulary

- **Boolean** – when you have two choices, on or off, yes or no.
- **Creativity** - has to do with both the process you invent (an algorithm) to solve a new problem in clever ways that can be executed by a machine.

Creativity

- **Creativity** often means combining or using algorithms you know as part of a solution to a new problem.
- Different algorithms can be developed to solve the same problem

Different programs (or code) can be written to implement the same algorithm.

Program Sequence

- Let's learn a little program sequence:

What is the answer to this program?

```
X = 2
X = 5
X = X + 1
```

The “Human Machine” Language

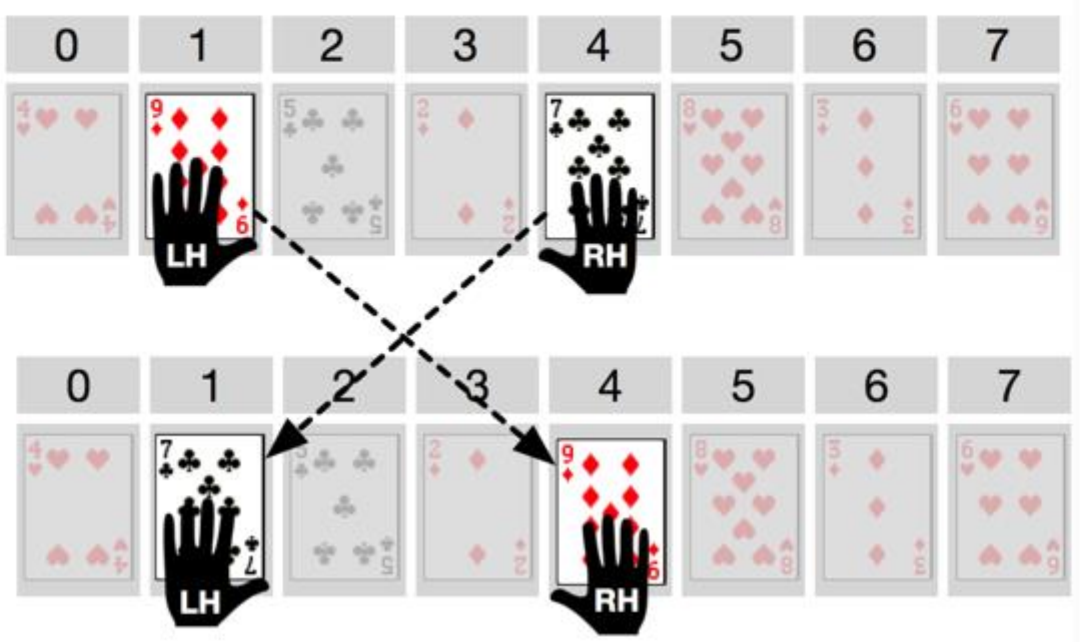
ACTIVITY – PART 2

Activity guide - Human Machine Language - Part 2: Min To Front

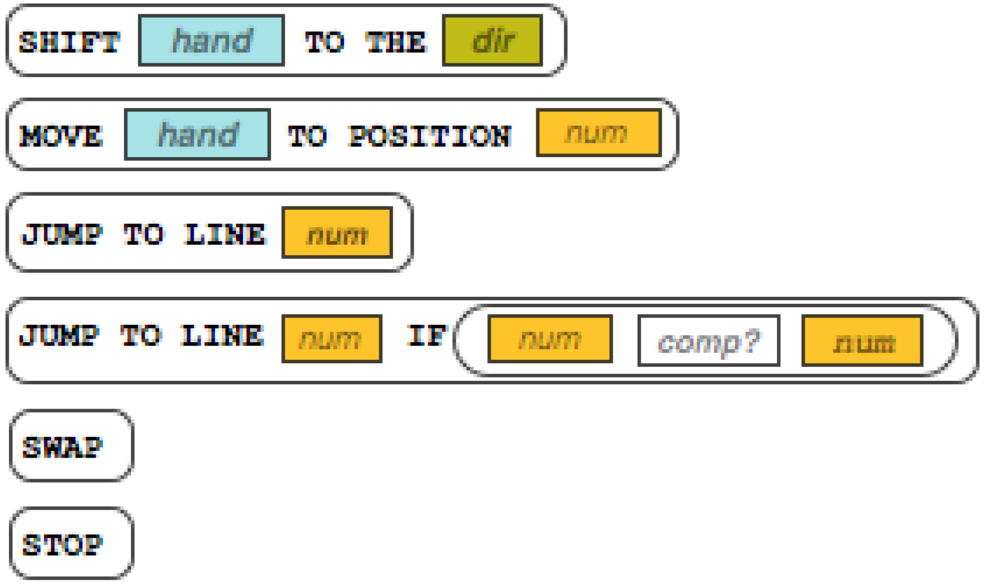
- Review the first page of the activity guide and the addition of the "**swap**" command.
- Here's what the example program does:
  END STATE: the order of the cards has been reversed
  - It does this by first moving the right hand to the end of the list, then shifting each hand progressively toward the middle of the row, swapping cards each time.
  - The program stops once the hands have crossed over each other (by checking if `RHPos < LHPos`)

SWAP

Swap the positions of the cards currently being touched by the left and right hands. After a swap the cards have changed positions but hands return to original position.

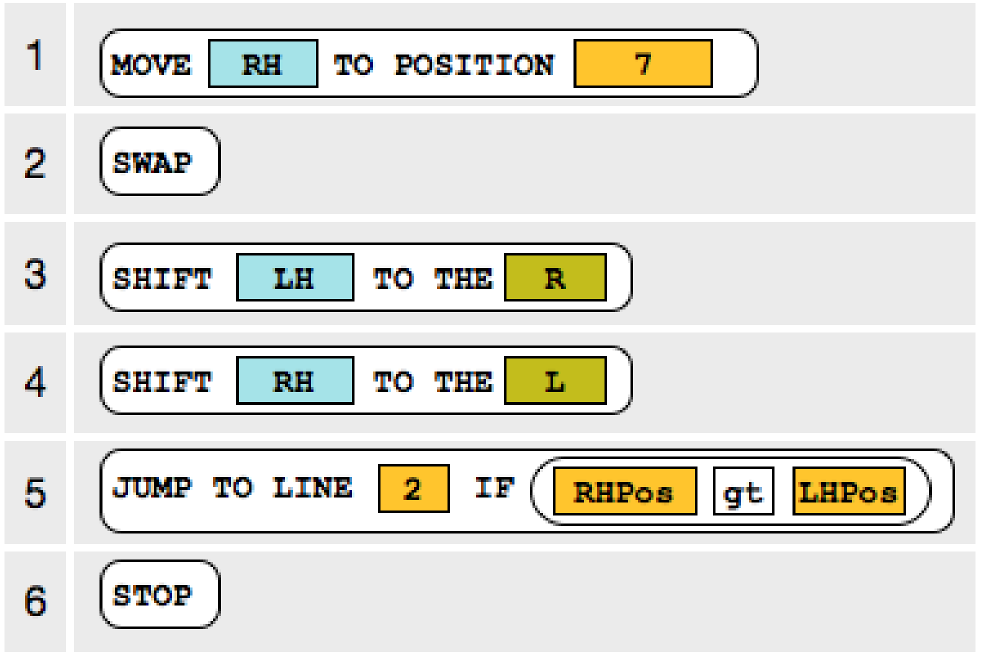


- We're going to add one command to the Human Machine Language called **SWAP** - see description below.
- All of the other commands are still available to you. So, there are 6 commands total in the language now.
- The human machine action is: pick up the cards, exchange the cards in hand, and return hands to original position in the list with the other card.


SHIFT hand TO THE dir
MOVE hand TO POSITION num
JUMP TO LINE num
JUMP TO LINE num IF (num comp? num)
SWAP
STOP

Try an example with Swap

Trace the program below with a partner and describe what it does.



```
1  MOVE RH TO POSITION 7
2  SWAP
3  SHIFT LH TO THE R
4  SHIFT RH TO THE L
5  JUMP TO LINE 2 IF (RHPos gt LHPos)
6  STOP
```

What does this program do?

Move right hand to the seventh card

Swap the cards that are being touched

Move left hand to the right

Move right hand to the left

If the right hands position is greater than the left hands position then swap again.

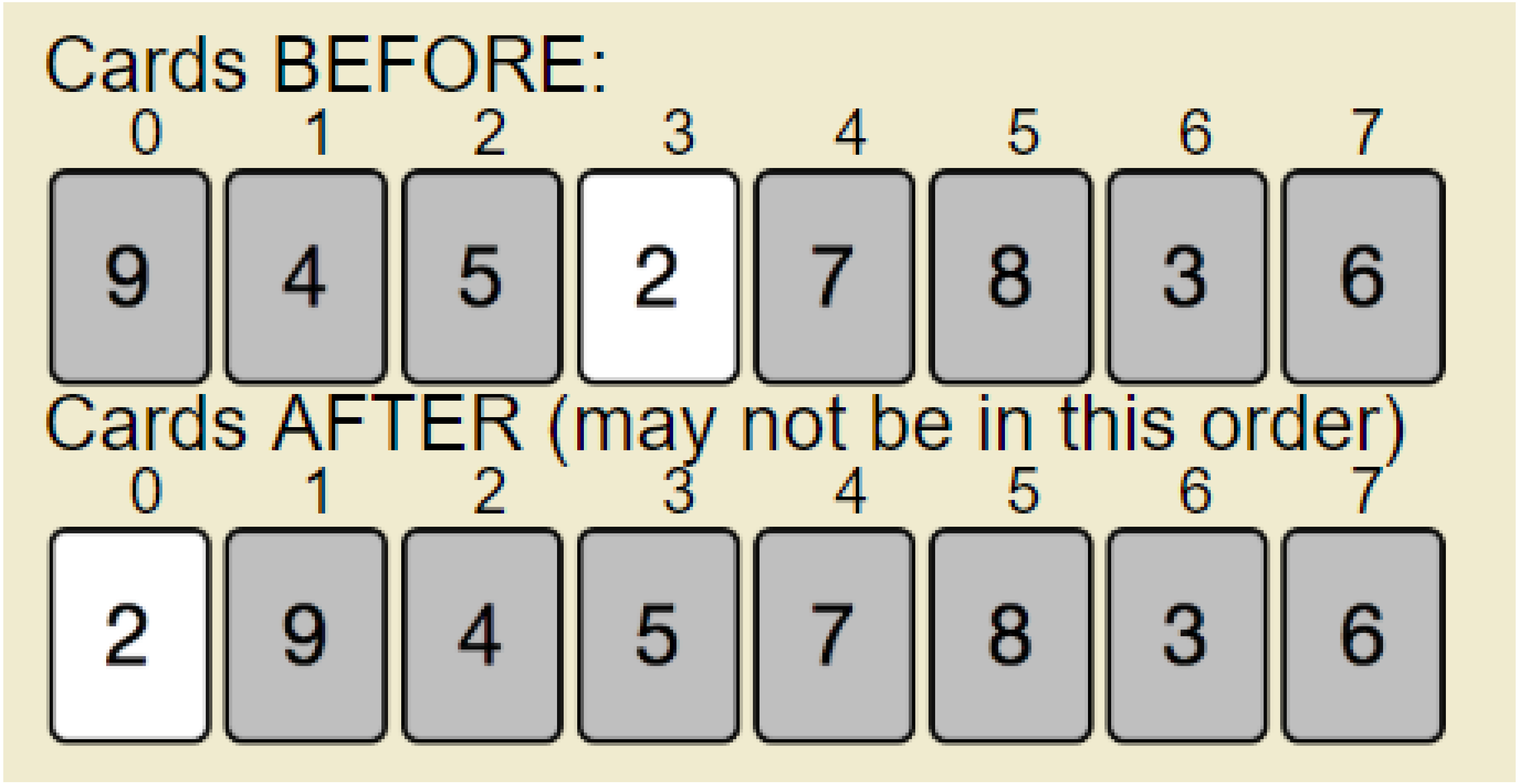
If not stop

Challenge: Min-to-Front

- The challenge is to find the min card and swap it to the front of the list, keeping the order of the rest of the cards the same.

Challenge: Min To Front

- Using only the Human Machine Language design an algorithm to find the smallest card and move it to the front of the list (position 0). All of the other cards must remain in their original relative ordering.
- **END STATE:** When the program stops, the smallest card should be in position 0. The ending positions of the hands do not matter, the ending positions of the other cards do not matter. As a challenge: try to move the min-to-front and have all other cards be in their original relative ordering.



Cards BEFORE:

0	1	2	3	4	5	6	7
9	4	5	2	7	8	3	6

Cards AFTER (may not be in this order)

0	1	2	3	4	5	6	7
2	9	4	5	7	8	3	6

1	Move left hand to position 3
2	Move right hand to position 0
3	Swap
4	Move left hand to position 1
5	Move right hand to position 2
6	Swap
7	Move left hand to position 3
8	Move right hand to position 1
9	Swap
10	

(If you need more lines, just keep going).

IDEA: Solve move-to-front first

- Remember: "Algorithms can be combined to make new algorithms“
- You should know a solution to find min, so you can put that out of mind for a minute.
- So, start by assuming that you've found the min card, and writing a procedure to move some card to the front of the list, by swapping.
- Once you've got that you can tack the two together

IDEA: Don't be afraid to invent a completely new algorithm

- get creative - you might need or want to invent a whole new algorithm
- have groups trade algorithms to test out each others' solutions

The CSP Framework states

4.1.1A Sequencing, selection, and iteration are building blocks of algorithms.
4.1.2G Every algorithm can be constructed using only sequencing, selection, and iteration.

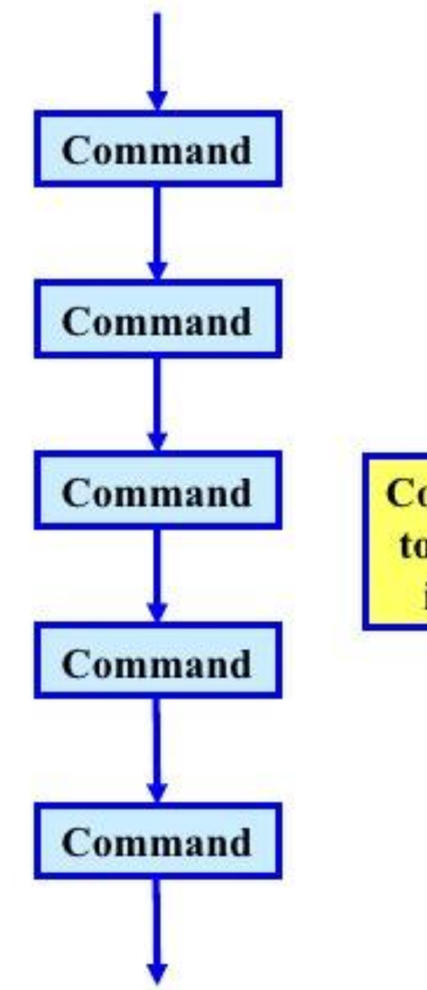
Structured Programming

LECTURE

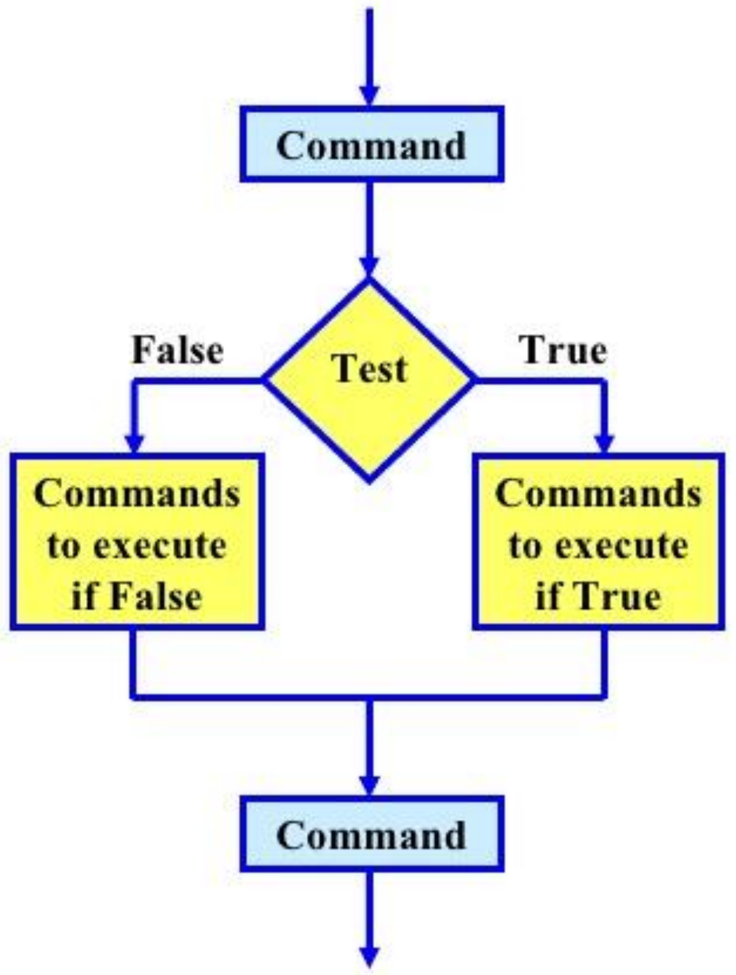
Structured Programming

- If these statements are true then we should be able to identify these elements of **sequencing, selection and iteration** in our **Find-Min** and **Min-to-Front** algorithms.
- I'll give you a quick definition of each and you tell me if or where we saw it in our Human Machine Language programs.

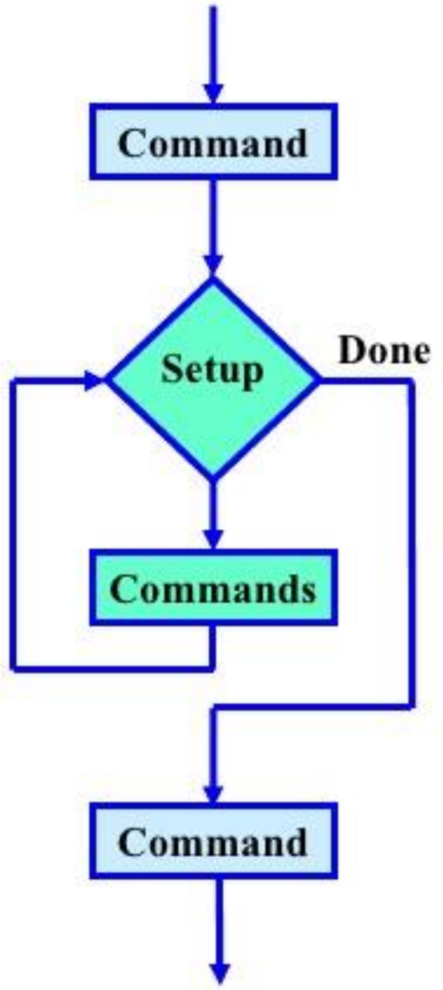
Flowcharts for sequential, selection, and iterative control structures



Sequential Structure
(straight-line structure)



Example Selection Structure
(decision or branching structure)



Iterative Structure
(looping structure)

Sequencing

- **“4.1.1B Sequencing** is the application of each step of an algorithm in the order in which the statements are given." -- Does our human machine language have sequencing?
- **Sequencing** is so fundamental to programming it sometimes goes without saying. In our lesson, the sequencing is simply implied by the fact that we number the instructions with the intent to execute them in order.

Selection

- "4.1.1C **Selection** uses a [true-false] condition to determine which of two parts of an algorithm is used." -- Where did we see "**selection**" in our human machine language programs?
- The JUMP…IF command in the Human Machine Language is a form of selection. It gives us a way to compare two things (numbers) and take action if the comparison is true, or simply proceed with the sequence if false.
- NOTE: **Selection** is also known as “branching” most commonly seen in if-statements in programs.

Iteration

- "4.1.1D **Iteration** is the repetition of part of an algorithm until a condition is met or for a specified number of times." -- Where did we see iteration in our human machine language programs?
- The JUMP command (as well as JUMP...IF) in the Human Machine Language allows us to move to a different point in the program and start executing from there. This allows us to re-use lines of code, and this is a form of **iteration** or **looping**.
- NOTE: **Iteration** is also known as "looping" in most programming languages.

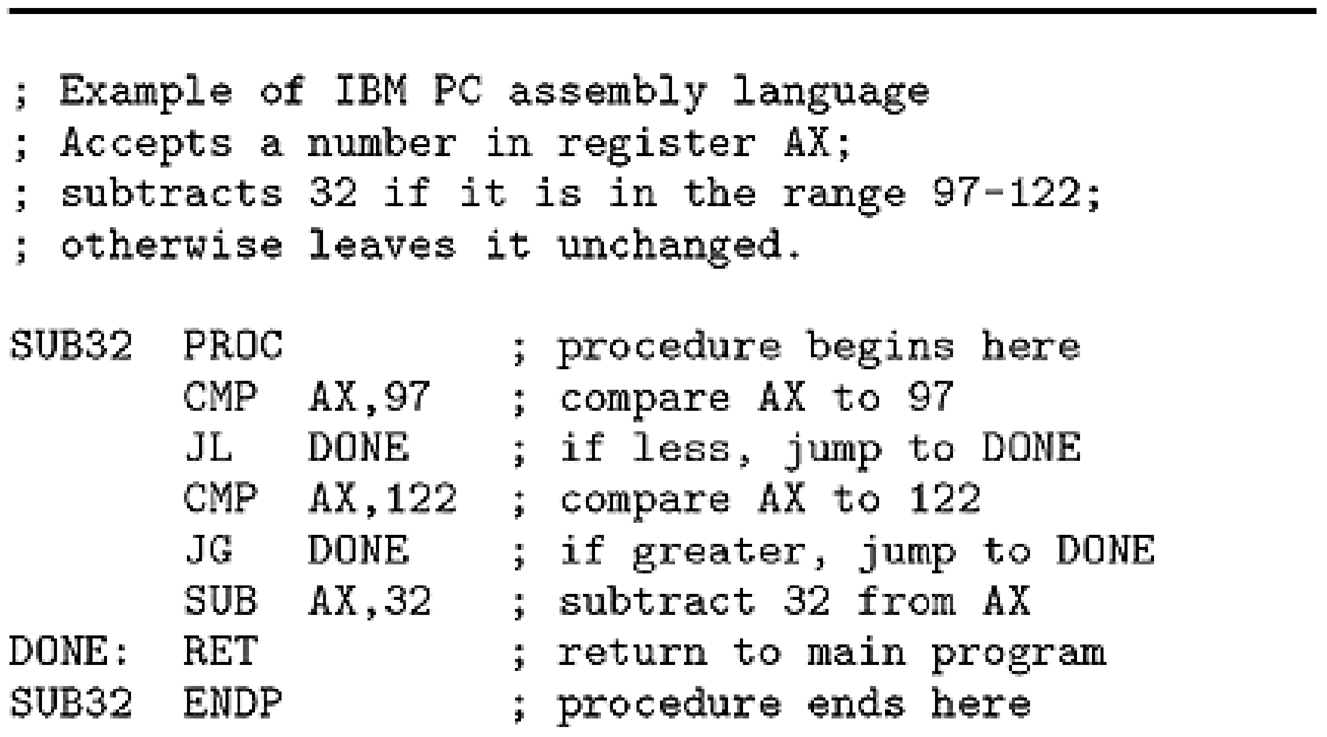
Development of Algorithm

- Important points:
  `Algorithms can be combined to make new algorithms`

- Low-Level languages exist - most basic, primitive, set of commands to control a computer. The Human Machine Language is similar to something called **Assembly Language**

Assembly Language

Here is an example of a simple program written in IBM Assembly Language.

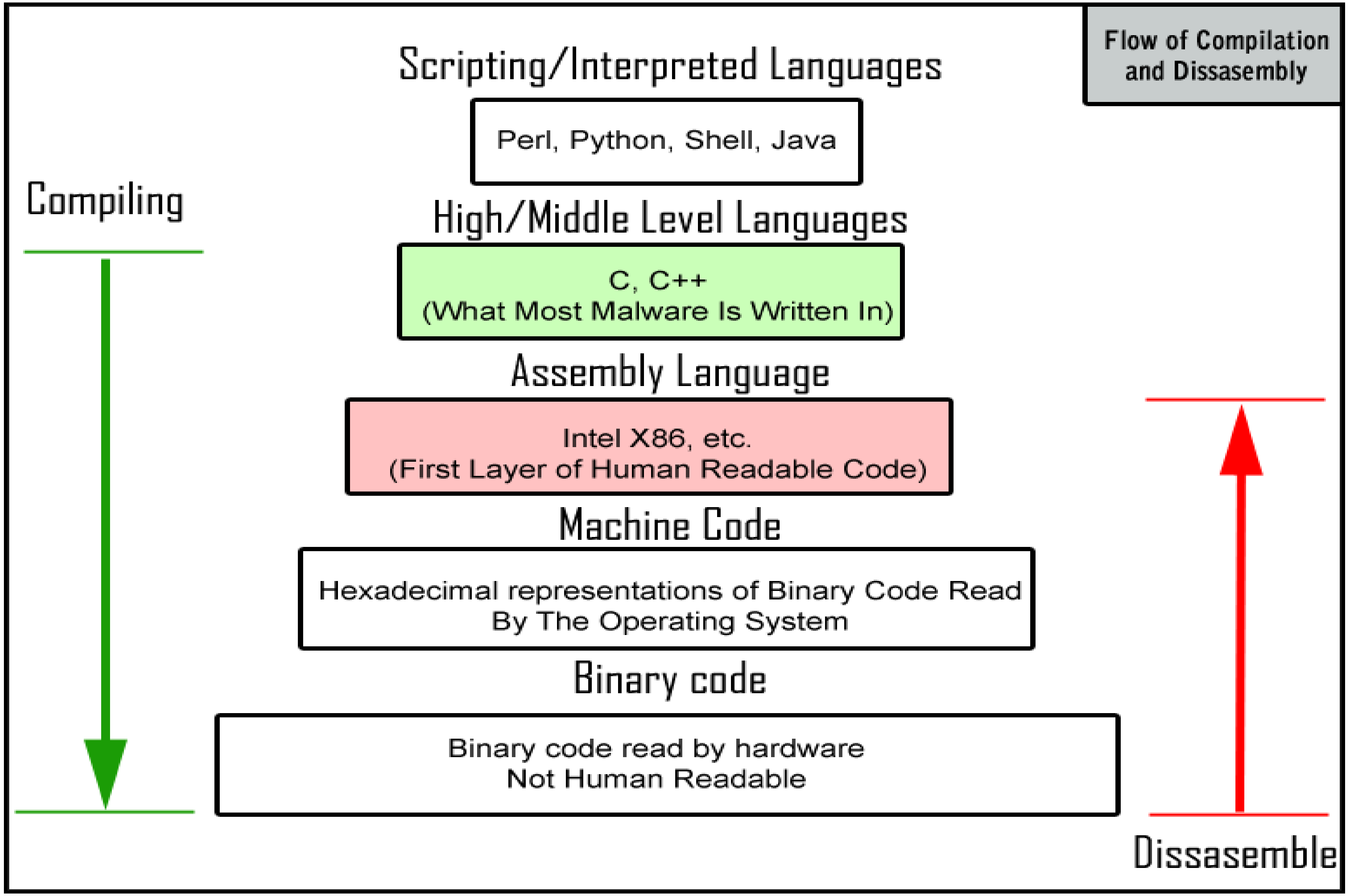


```
; Example of IBM PC assembly language
; Accepts a number in register AX;
; subtracts 32 if it is in the range 97-122;
; otherwise leaves it unchanged.

SUB32  PROC           ; procedure begins here
       CMP  AX,97     ; compare AX to 97
       JL   DONE      ; if less, jump to DONE
       CMP  AX,122    ; compare AX to 122
       JG   DONE      ; if greater, jump to DONE
       SUB  AX,32     ; subtract 32 from AX
DONE:  RET            ; return to main program
SUB32  ENDP           ; procedure ends here
```

Programming Language

- From the CSP Framework: 2.2.3C Code in a programming language is often translated into code in another (lower level) language to be executed on a computer.
- The Human Machine Language is a "low level" language because the commands are very primitive and tie directly specific functions of the "human machine".


Flow of Compilation and Dissasembly
Scripting/Interpreted Languages
Perl, Python, Shell, Java
Compiling
High/Middle Level Languages
C, C++
(What Most Malware Is Written In)
Assembly Language
Intel X86, etc.
(First Layer of Human Readable Code)
Machine Code
Hexadecimal representations of Binary Code Read By The Operating System
Binary code
Binary code read by hardware
Not Human Readable
Dissasemble

Magical Powers

- Learning to program is really learning how to think in terms of algorithms and processes. And it can be really fun and addicting. It also can make you feel like you have **magical powers**.

Algorithm Efficiency

LESSON 2 [UNIT 6 CODE.ORG]

Unreasonable Time

LESSON 3 [UNIT 6 CODE.ORG]

The Limits of Algorithms

LESSON 4 [UNIT 6 CODE.ORG]

Parallel and Distributed Algorithms

LESSON 5 [UNIT 6 CODE.ORG]

Parameters and Return Value

SECTION 2

Parameters and Return Explore

LESSON 2 [UNIT 7 CODE.ORG]

Parameters and Return Explore

LESSON 2 [UNIT 7 CODE.ORG]

Parameters and Return Practice

LESSON 3 [UNIT 7 CODE.ORG]

Parameters and Return Make

LESSON 4 [UNIT 7 CODE.ORG]

Library

SECTION 3

Library Explore

LESSON 2 [UNIT 7 CODE.ORG]

Library Investigate

LESSON 6 [UNIT 7 CODE.ORG]

Library Practice

LESSON 7 [UNIT 7 CODE.ORG]

Library Make

LESSON 8-10 [UNIT 7 CODE.ORG]