

CL00: The Ideas Behind Coding

Today's Format is A Little Different...

- Little more lecture-y
- A little more vague

Why?

- A gentler introduction
- Want you to get a bigger picture of the little things we're going to talk about later
- **I don't expect you to be able to do any of these things tomorrow... that's what this class is for!**

Computational Thinking

- Strategic thought and problem-solving
- Can help perform a task better, faster, cheaper, etc.
- Examples:
 - Meal prepping
 - Making your class schedule
 - “Life Hacks”

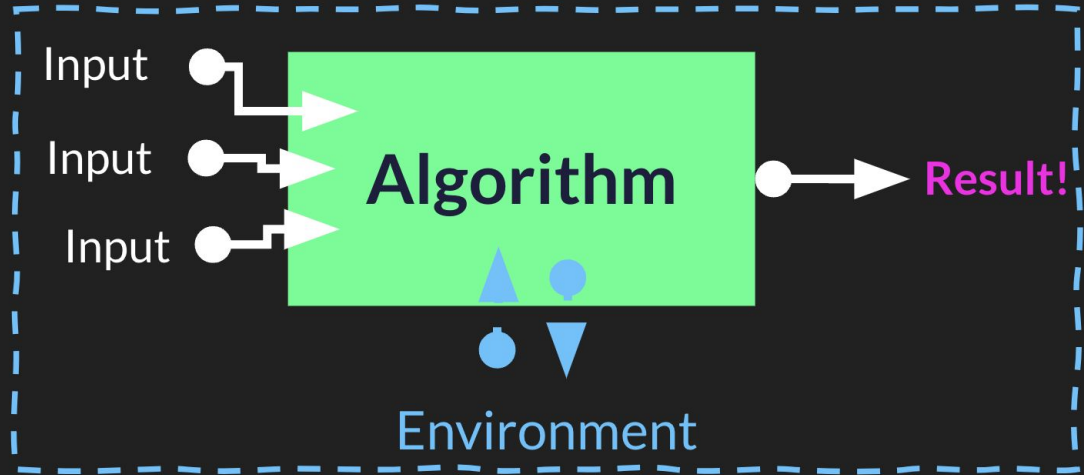
Algorithms

Input is data given to an algorithm

An **algorithm** is a series of steps

An algorithm **returns** some **result**

An algorithm *may* be influenced by its **environment** and it *may* produce side-effects which influence its environment.



Example: My dissertation



megapope

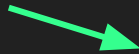
self driving cars aren't even hard to make lol
just program it not to hit stuff



ronpaulhdwallpapers

```
if(goingToHitStuff) {  
  dont();  
}
```

Algorithm



Discussion

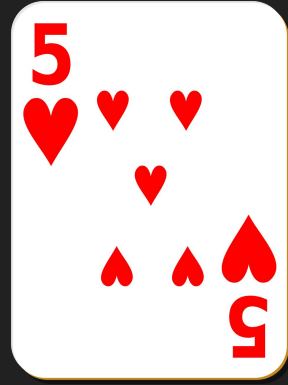
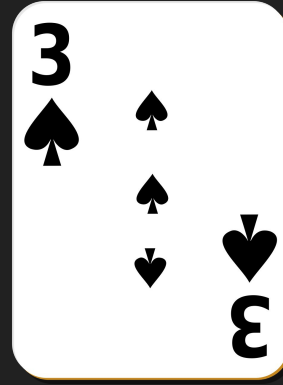
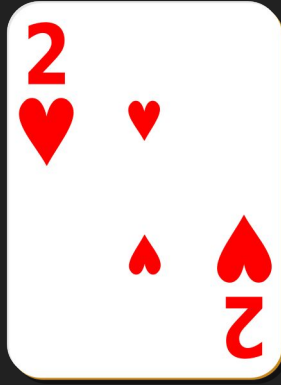
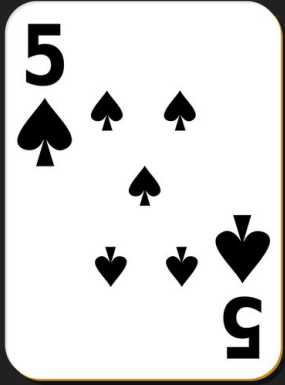
What are examples of computational thinking that you use day to day?

What kind of algorithms do you use to implement these ideas?

What is an algorithm?

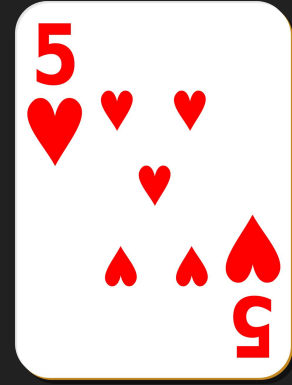
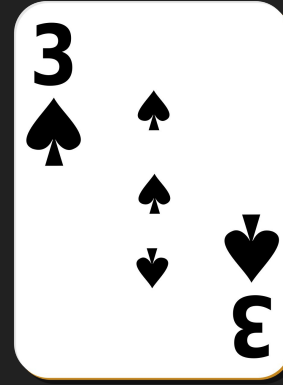
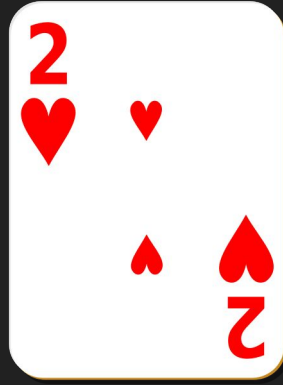
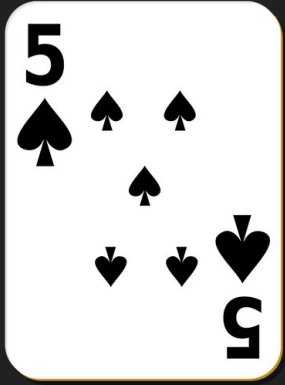
- A set of steps to solve a general problem
- Finite
- Can handle a problem of arbitrary size

Classic Algorithm: Sorting

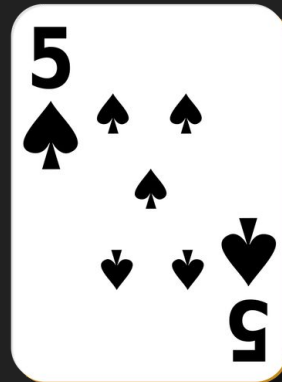
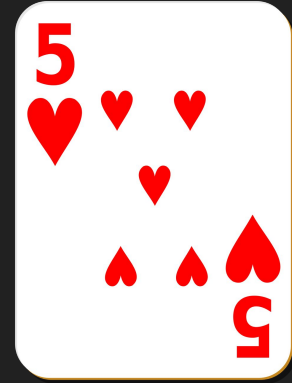
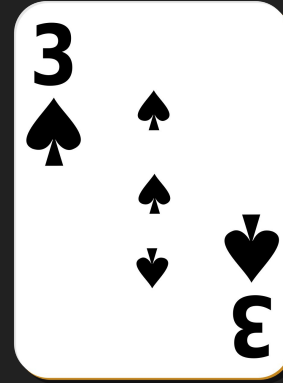
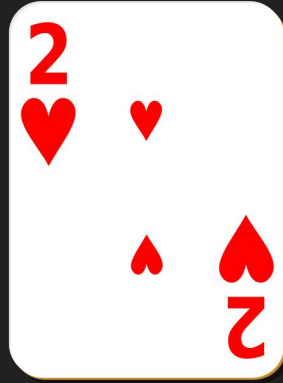


Instructions to sort these cards (or any set of cards) from least to greatest?

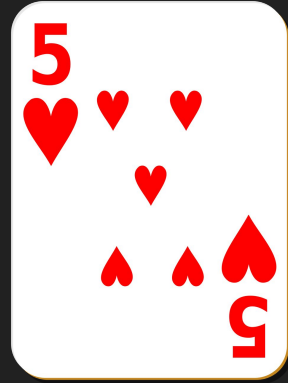
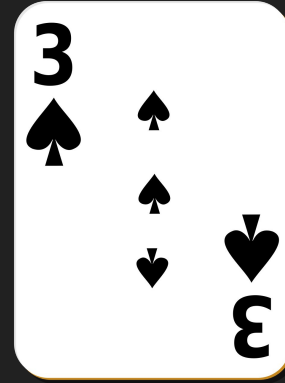
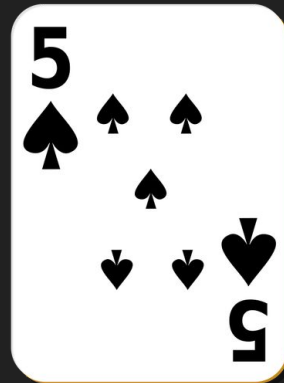
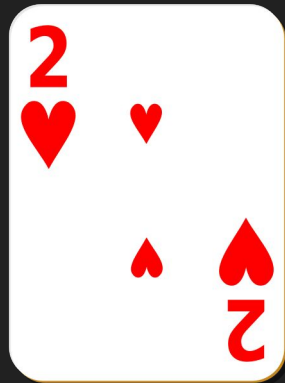
Insertion Sort: Build sorted deck.



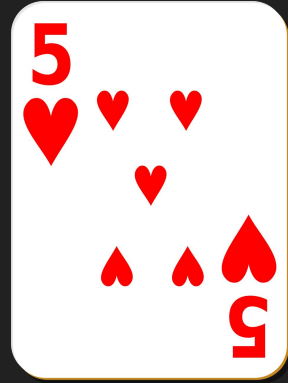
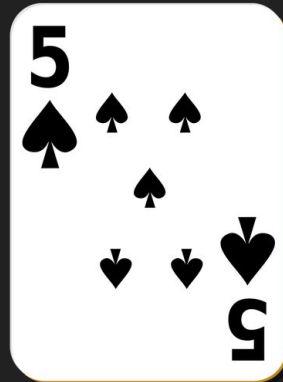
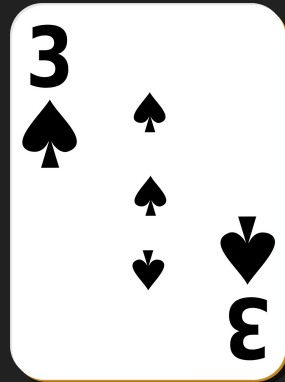
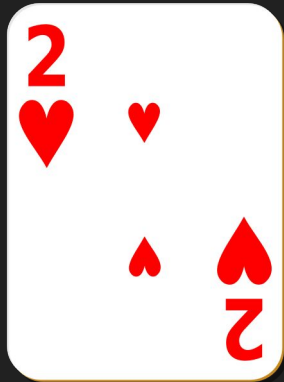
Insertion Sort: Build sorted deck.



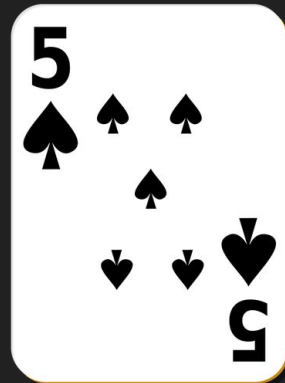
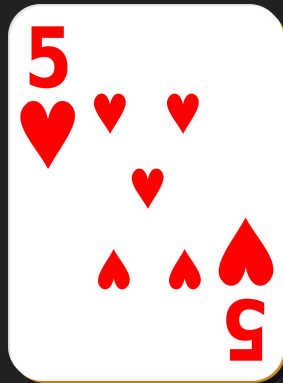
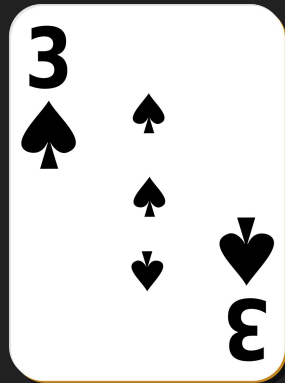
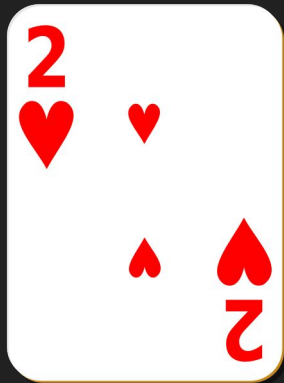
Insertion Sort: Build sorted deck.



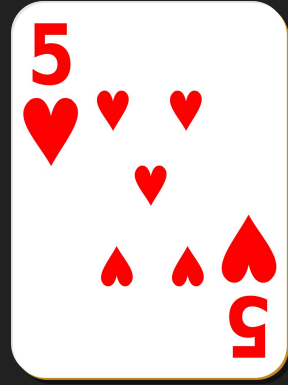
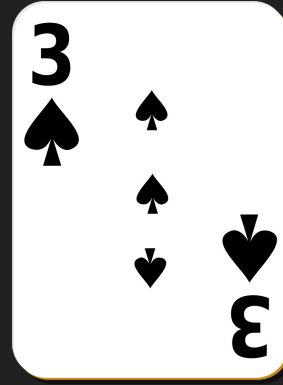
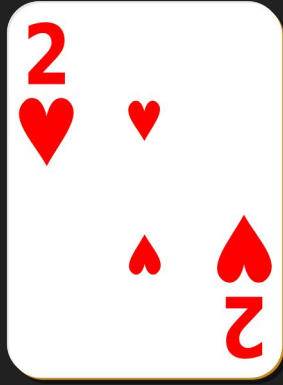
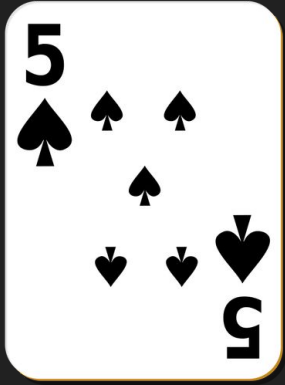
Insertion Sort: Build sorted deck.



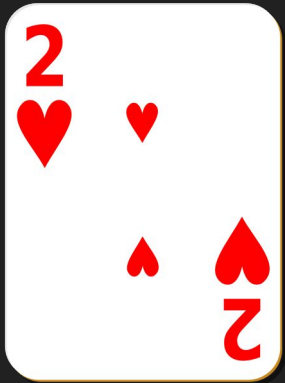
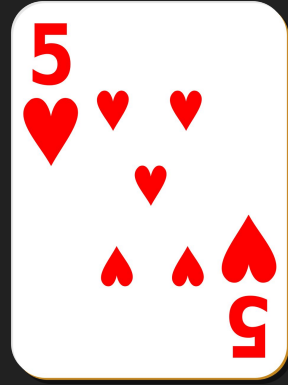
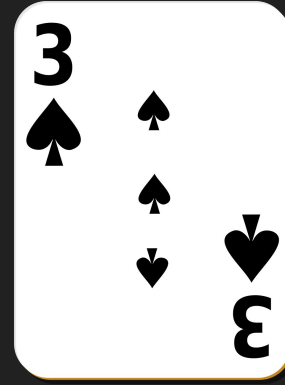
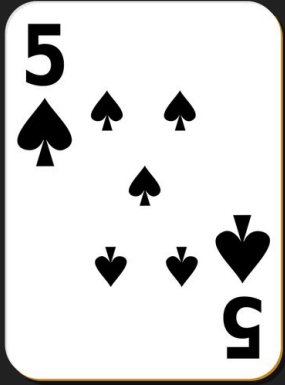
Insertion Sort: Build sorted deck.



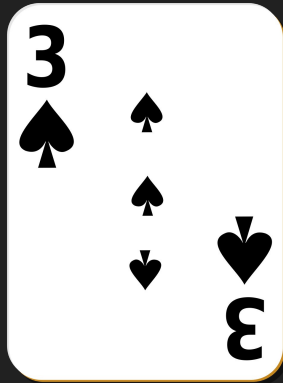
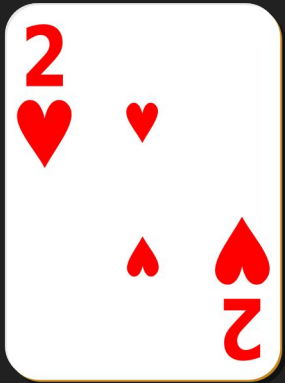
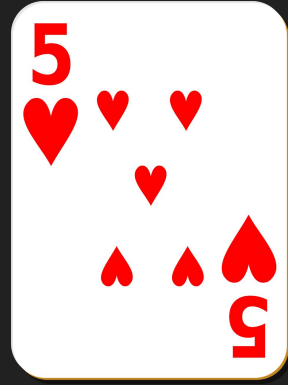
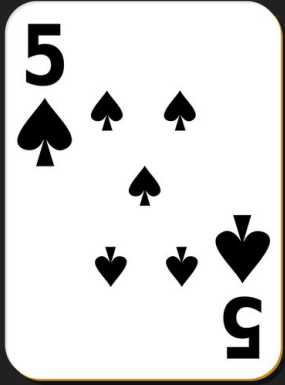
Selection Sort: repeatedly choose minimum



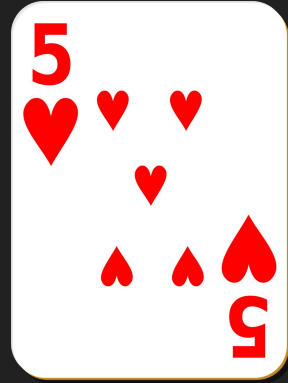
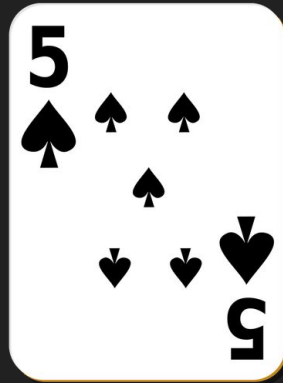
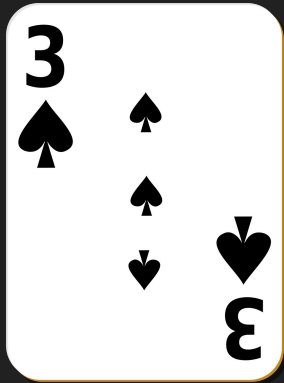
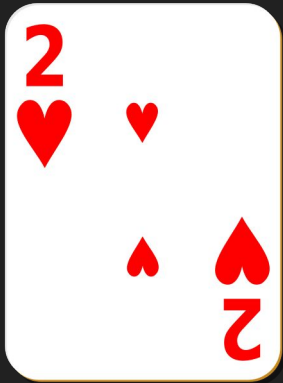
Selection Sort: repeatedly choose minimum



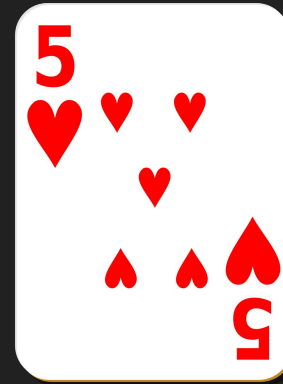
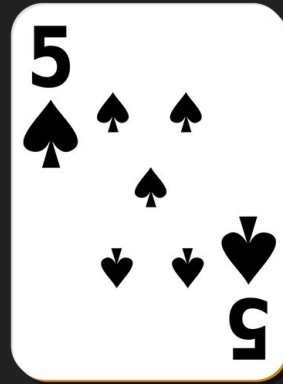
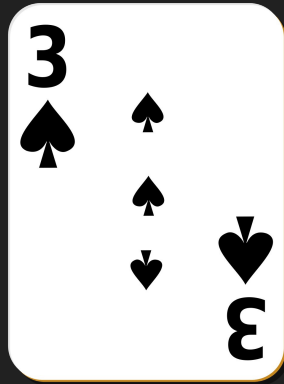
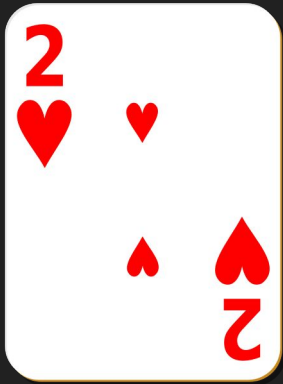
Selection Sort: repeatedly choose minimum



Selection Sort: repeatedly choose minimum



Selection Sort: repeatedly choose minimum



How do we express these steps?

Selection Sort:

- You're going to make a new, sorted deck, so let's call that our "new deck"
- From your old deck, repeatedly **find the card with the lowest value** and add it to the new deck until the old deck is empty

Point Of This Example

- Th

Back to Selection Sort...

Original instructions

- You're going to make a new, sorted deck, so let's call that our "new deck"
- From your old deck, repeatedly find the card with the lowest value and add it to the new deck until the old deck is empty

Pseudocode:

```
new_deck = new CardDeck()
```

Back to Selection Sort...

Original instructions

- You're going to make a new, sorted deck, so let's call that our "new deck"
- From your old deck, repeatedly find the card with the lowest value and add it to the new deck until the old deck is empty

Pseudocode:

```
new_deck = new CardDeck()
```



Assignment

Back to Selection Sort...

Original instructions

- You're going to make a new, sorted deck, so let's call that our "new deck"
- From your old deck, repeatedly find the card with the lowest value and add it to the new deck until the old deck is empty

Pseudocode:

```
new_deck = new CardDeck()
```

Repeatedly until old_deck is empty:

```
low_card = find_lowest_card(old_deck)
```

```
new_deck = new_deck + low_card
```

Back to Selection Sort...

Original instructions

- You're going to make a new, sorted deck, so let's call that our "new deck"
- From your old deck, repeatedly find the card with the lowest value and add it to the new deck until the old deck is empty

Pseudocode:

```
new_deck = new CardDeck()
```

Loop
↙
Repeatedly until old_deck is empty:

```
low_card = find_lowest_card(old_deck)
```

```
new_deck = new_deck + low_card
```


Back to Selection Sort...

Original instructions

- You're going to make a new, sorted deck, so let's call that our "new deck"
- From your old deck, repeatedly find the card with the lowest value and add it to the new deck until the old deck is empty

Pseudocode:

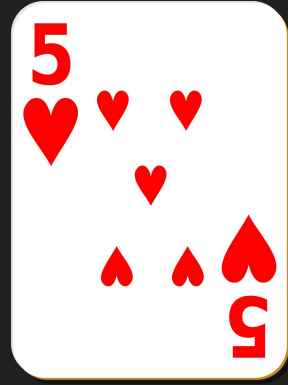
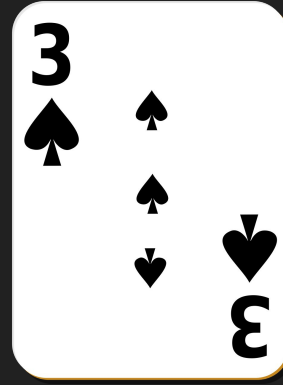
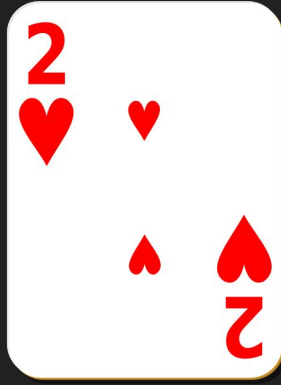
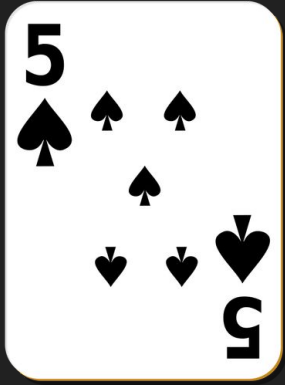
```
new_deck = new CardDeck()
```

Repeatedly until old_deck is empty:

```
low_card = find_lowest_card(old_deck)
```

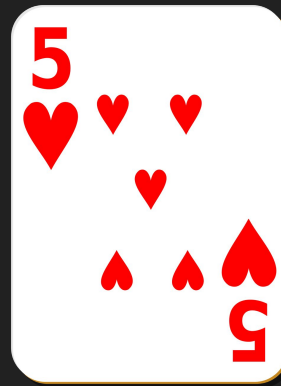
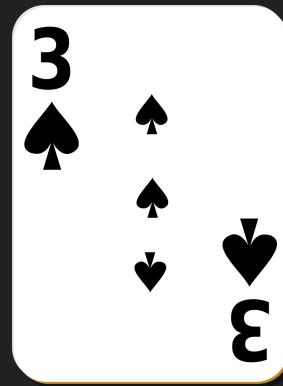
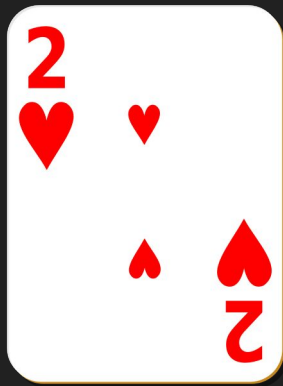
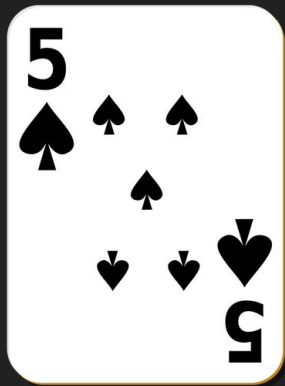
```
new_deck = new_deck + low_card
```

Finding the Lowest Card in a Deck

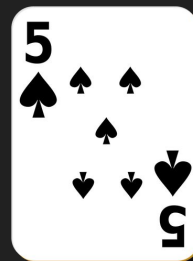


- Go from left to right
- Remember the lowest card you've seen *so far* and compare it to the next cards

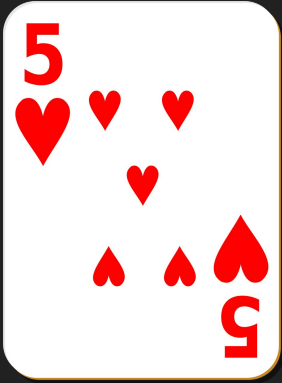
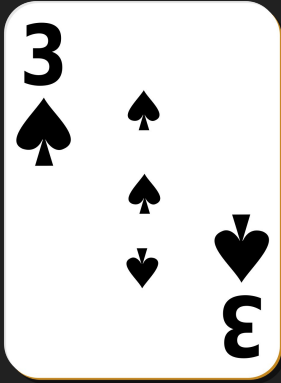
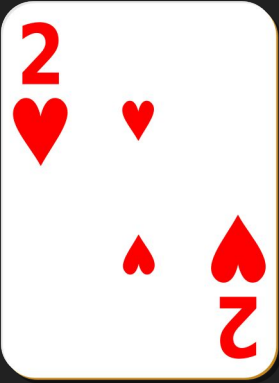
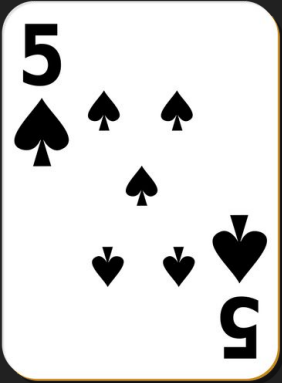
Finding the Lowest Card



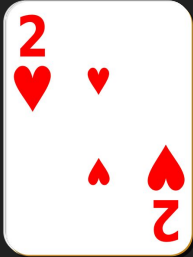
Low card:



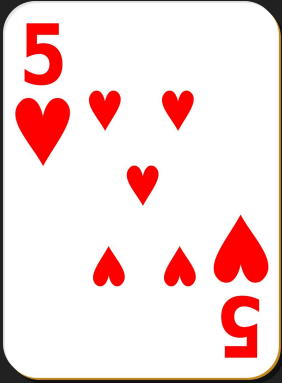
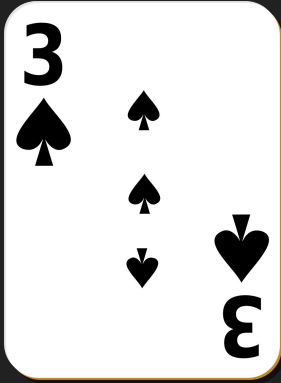
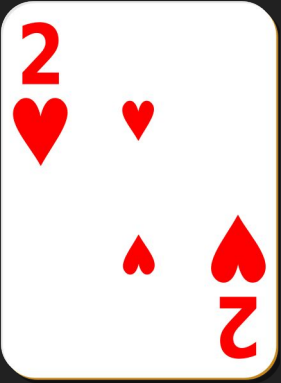
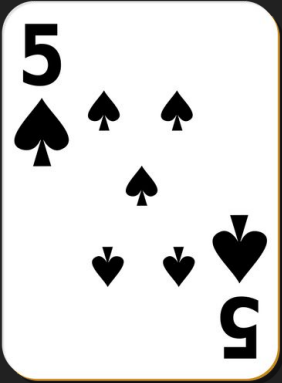
Finding the Lowest Card



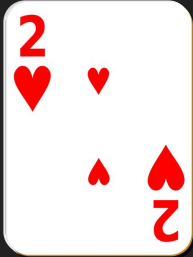
Low card:



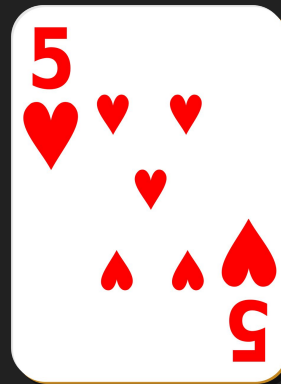
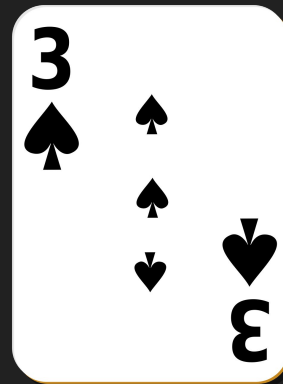
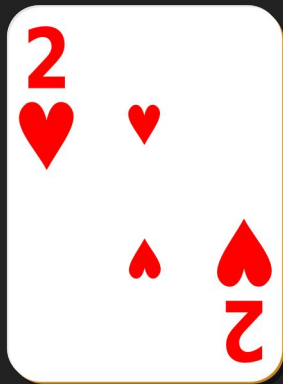
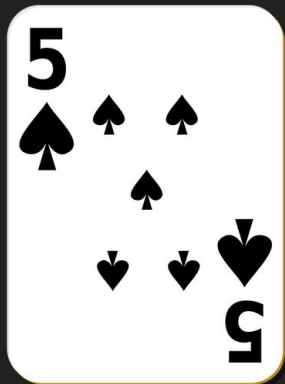
Finding the Lowest Card



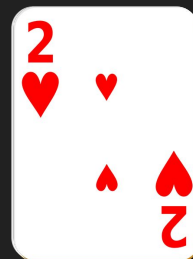
Low card:



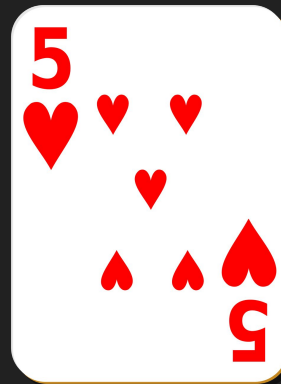
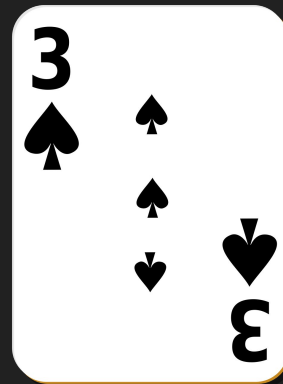
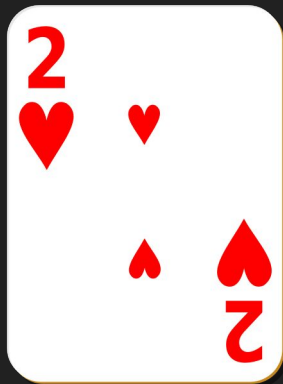
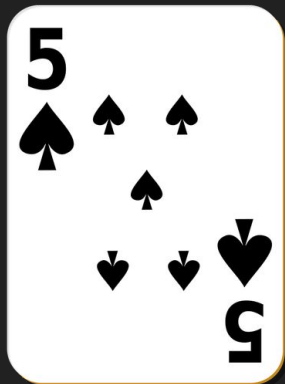
Finding the Lowest Card



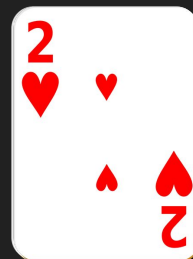
Low card:



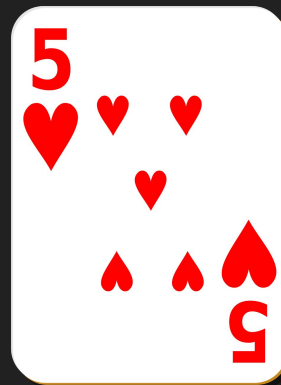
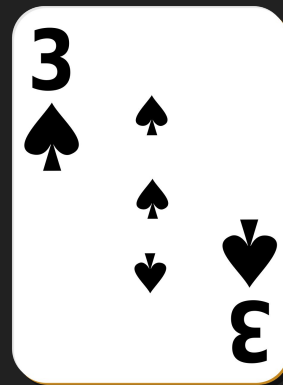
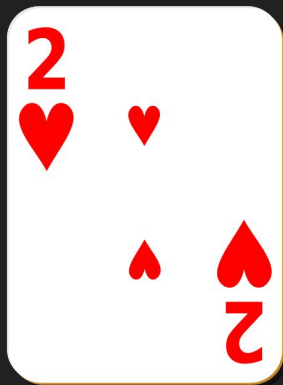
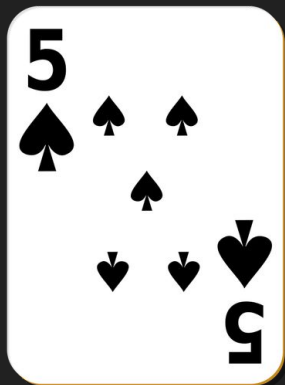
Finding the Lowest Card



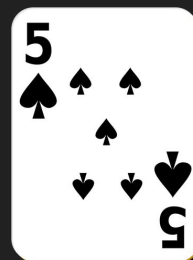
Low card:



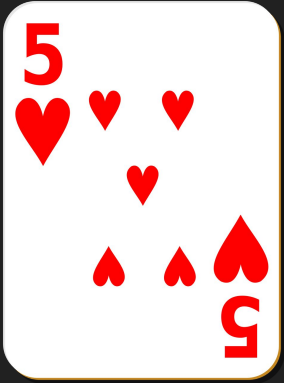
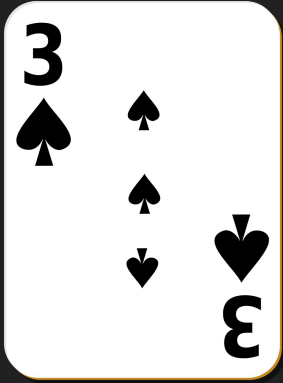
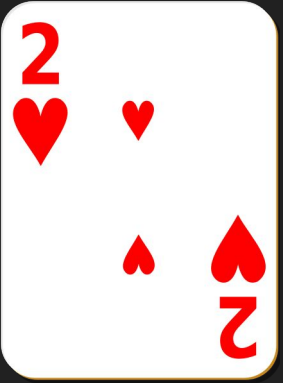
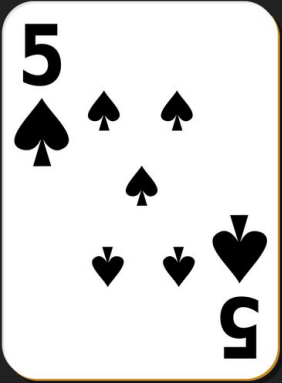
Finding the Lowest Card



Low card:



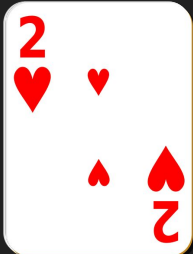
Finding the Lowest Card



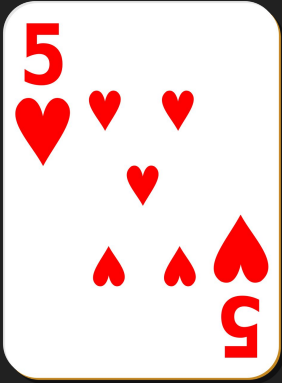
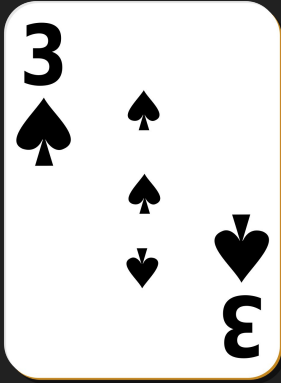
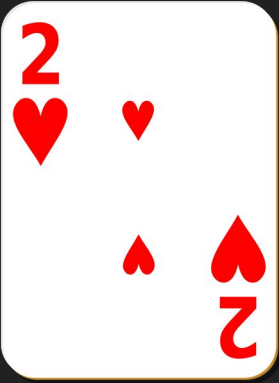
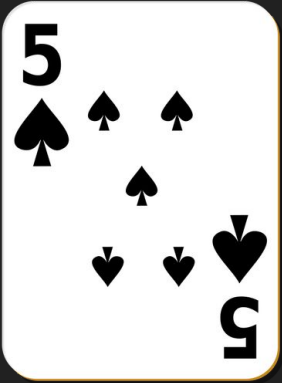
2 < 5?



Low card:

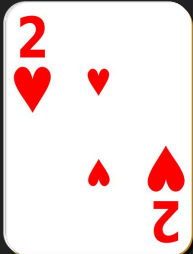


Finding the Lowest Card

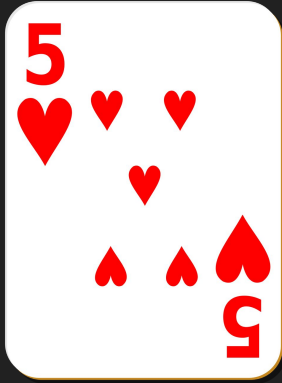
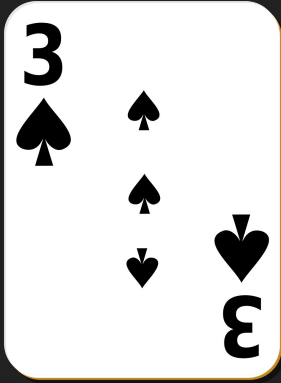
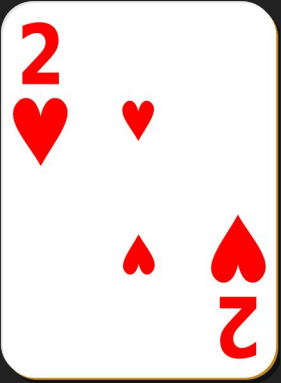
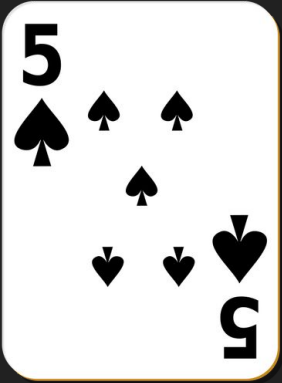


3 < 2? 

Low card:

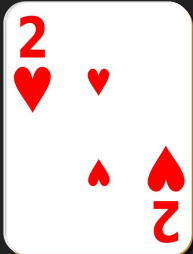


Finding the Lowest Card

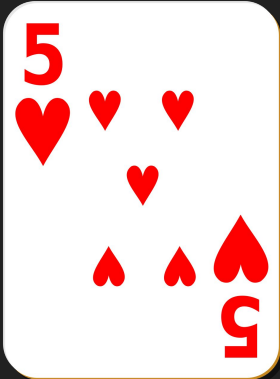
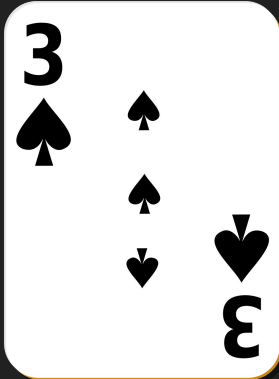
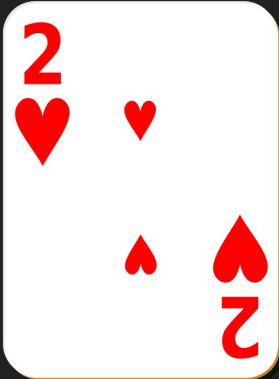
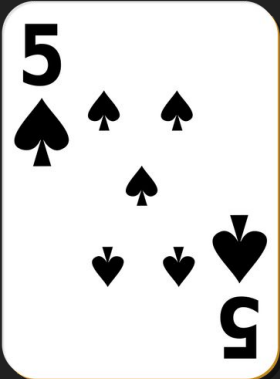


5 < 2? 

Low card:



Finding the Lowest Card

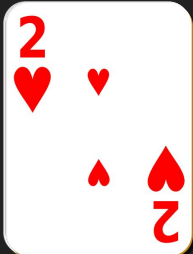


5 < 2?



Relational Operator

Low card:




Pseudocode

Looks like code, but simplified and readable.

Not meant to run on a computer.

Helps you outline what your algorithm is going to look like.

You should be able to expand on your pseudocode to help you write actual code!



If (going to hit stuff):
 dont()

Finding the Lowest Card Pseudocode

- Go from left to right
- Remember the lowest card you've seen *so far* and compare it to the next cards

Pseudocode:

Finding the Lowest Card Pseudocode

- Go from left to right
- Remember the lowest card you've seen *so far* and compare it to the next cards

Pseudocode:

```
lowest_card = first card in deck
```

Finding the Lowest Card Pseudocode

- Go from left to right
- Remember the lowest card you've seen *so far* and compare it to the next cards

Pseudocode:

lowest_card = first card in deck


Assignment

Finding the Lowest Card Pseudocode

- Go from left to right
- Remember the lowest card you've seen *so far* and compare it to the next cards

Pseudocode:

lowest_card = first card in deck



Assignment

(Week 1 concept)

Finding the Lowest Card Pseudocode

- Go from left to right
- Remember the lowest card you've seen *so far* and compare it to the next cards

Pseudocode:

lowest_card = first card in deck

Repeatedly until end of deck:

 if current_card < lowest_card:

 lowest_card = current_card

Finding the Lowest Card Pseudocode

- Go from left to right
- Remember the lowest card you've seen so *far* and compare it to the next cards

Pseudocode:

lowest_card = first card in deck

Repeatedly until end of deck:

 if current_card < lowest_card:

 lowest_card = current_card

Loop



Finding the Lowest Card Pseudocode

- Go from left to right
- Remember the lowest card you've seen so *far* and compare it to the next cards

Pseudocode:

lowest_card = first card in deck

Repeatedly until end of deck:

if current_card < lowest_card:

lowest_card = current_card

Loop

(Week 3
concept)

Finding the Lowest Card Pseudocode

- Go from left to right
- Remember the lowest card you've seen *so far* and compare it to the next cards

Pseudocode:

`lowest_card = first card in deck`

Repeatedly until end of deck:

`if current_card < lowest_card:`

`lowest_card = current_card`

Conditional



Finding the Lowest Card Pseudocode

- Go from left to right
- Remember the lowest card you've seen *so far* and compare it to the next cards

Pseudocode:

`lowest_card = first card in deck`

Repeatedly until end of deck:

`if current_card < lowest_card:`

`lowest_card = current_card`

Conditional

(Week 2 concept)



Finding the Lowest Card Pseudocode

- Go from left to right
- Remember the lowest card you've seen *so far* and compare it to the next cards

Pseudocode:

lowest_card = first card in deck

Repeatedly until end of deck:

if current_card < lowest_card:

lowest_card = current_card

**Relational
Operator**



Finding the Lowest Card Pseudocode

- Go from left to right
- Remember the lowest card you've seen *so far* and compare it to the next cards

Pseudocode:

lowest_card = first card in deck

Repeatedly until end of deck:

if current_card < lowest_card:

lowest_card = current_card

**Relational
Operator**

(Week 1 concept)

Finding the Lowest Card Pseudocode

- Go from left to right
- Remember the lowest card you've seen *so far* and compare it to the next cards

`find_lowcard(deck)`

`lowest_card = first card in deck`

Repeatedly until end of deck:

`if current_card < lowest_card:`

`lowest_card = current_card`

Function



Finding the Lowest Card Pseudocode

- Go from left to right
- Remember the lowest card you've seen *so far* and compare it to the next cards

`find_lowcard(deck)`

`lowest_card = first card in deck`

Repeatedly until end of deck:

`if current_card < lowest_card:`

`lowest_card = current_card`

Function

(Week 4 concept)

Takeaways

- Pseudocode: simple and readable version of algorithm that resembles code
- Assignment Operator: Assigns a variable some value
- Loop Statement: Repeatedly performs an action a fixed number of times
- Relational Operator: Compares two values
- Conditional Statement: A statement that only performs an action under certain conditions
- Function: Generalizes code to work for a generic input

Again, you don't need to know these right now, but I want you to have a point of reference when you do learn them!

What is an algorithm?

