Computational thinking

Computational thinking

1. Decomposition
   Can I divide this into sub-problems?

2. Pattern recognition
   Can I find repeating patterns?

3. Abstraction
   Can I create a model to design my program around (variables structure, functions structure)?

4. Algorithm design
   Can I create a recipe of programming steps for any of this?

Applying computational thinking

1. Start with a small human solvable version of the problem and solve it manually. This will help you get a better understanding of what needs to occur.
2. Keep working through the 4 computational thinking prompts until you devise a solution.
3. Test your proposed solution works on larger, real-world versions of the problem.
4. Adapt and repeat the process as necessary.

Example: Sorting numbers

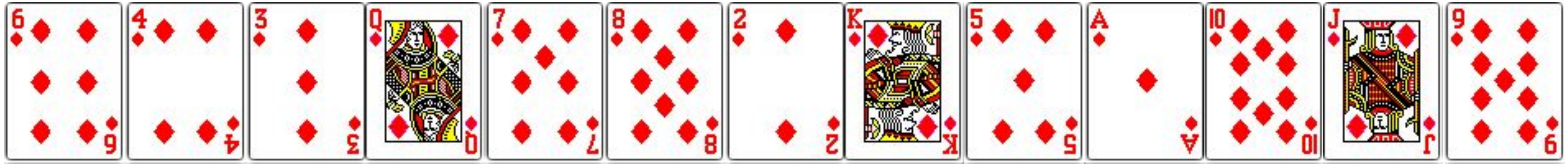
Challenge: Create a program that will sort a list of numbers into ascending order.

16	23	80	18	24	77	52	44	36	1	39	41	45	14	63	70	11	5
63	36	24	60	34	4	41	51	27	18	93	97	51	58	17	78	40	22
67	60	47	52	12	35	32	32	47	49	90	92	14	41	31	28	43	78
42	15	48	21	45	33	26	92	98	6	81	41	31	37	100	72	62	56
24	85	12	70	5	92	31	2	11	100	53	45	62	58	61	67	58	12
90	6	95	20	32	89	90	64	99	29	78	90	98	38	47	5	69	79
18	46	35	84	93	40	68	55	73	47	9	56	48	24	42	11	10	43
32	17	48	66	21	65	24	35	94	74	63	87	92	93	91	93	49	48
61	88	24	75	35	63	4	80	44	11	44	88	64	78	85	72	82	31
47	64	38	55	82	92	48	98	2	21	100	40	51	39	62	12	74	96
62	19	53	9	66	92	67	24	25	39	77	27	73	20	27	48	56	89

Getting started

First step: Create a human solvable version of the problem.

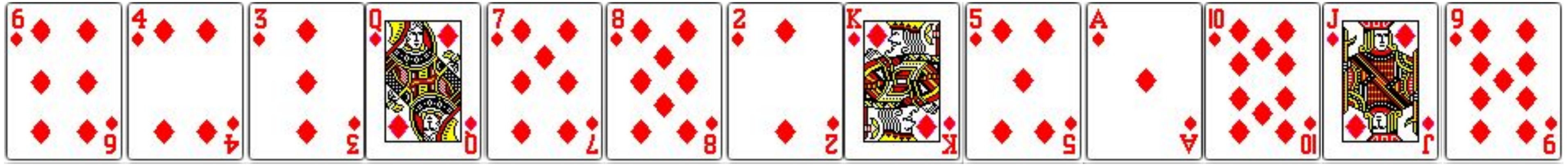
An enormous infinitely large set of numbers is a little overwhelming to think about. We know how to sort playing cards though.



Have a go

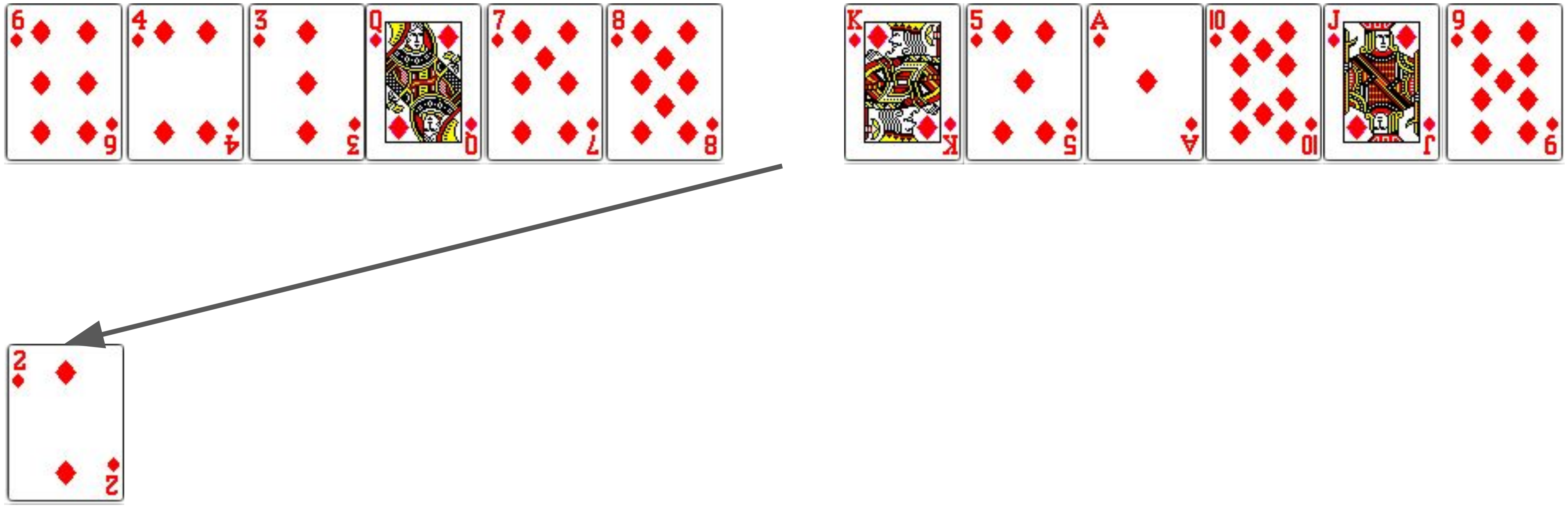
Task: 5 minutes with the person next to you. Write a set of steps for someone who had never sorted values, what would you write?

Remember: Your ultimate goal is not how to sort the given set of cards as presented, but to sort any possible set of numbers.

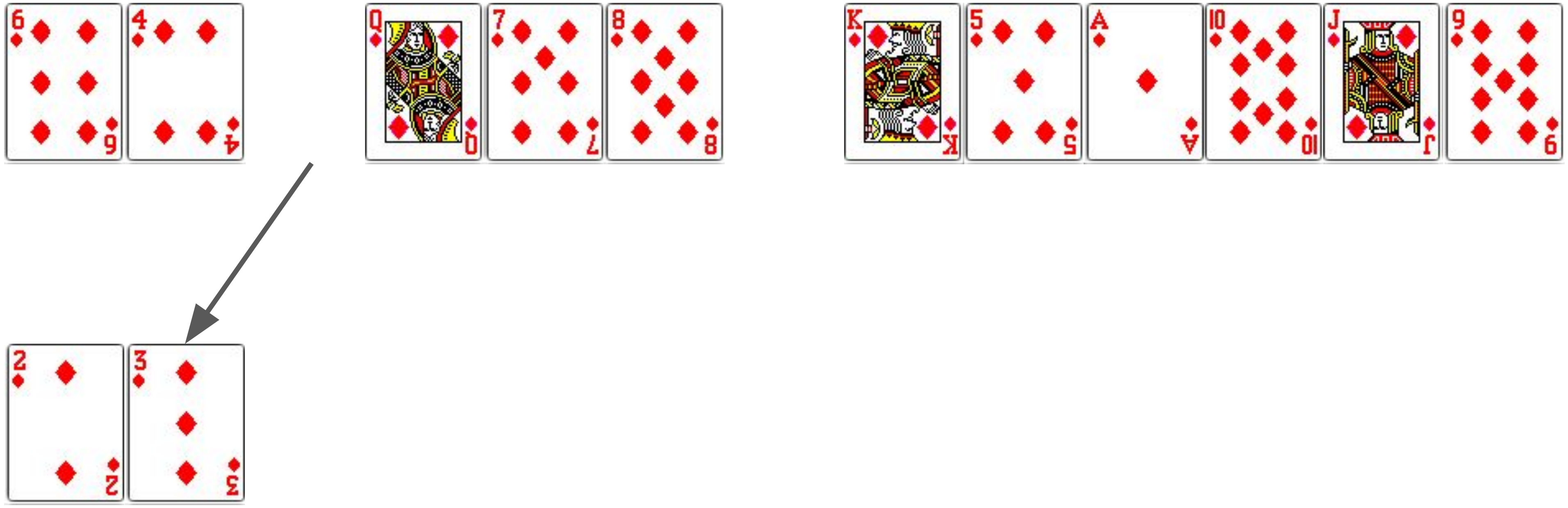


* assume Jack = 11, Queen = 12, King = 13, Ace = 14

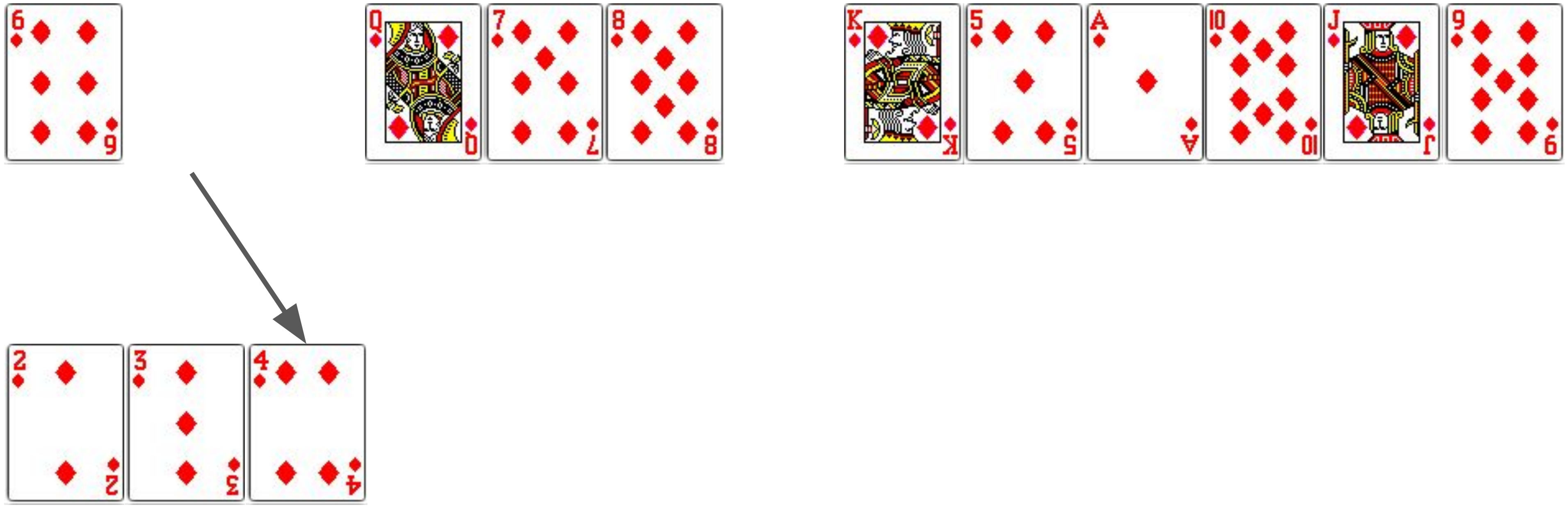
The human process...



The human process...



The human process...



But what did we actually do?

Search through all the cards
Identify 2 as the lowest
Move it to the sorted set
Search through all the cards
Identify 3 as the lowest
Move it to the sorted set
Search through all the cards
Identify 4 as the lowest
Move it to the sorted set
… etc

But what did we actually do?

Search through all the cards
Identify 2 as the lowest
Move it to the sorted set
Search through all the cards
Identify 3 as the lowest
Move it to the sorted set
Search through all the cards
Identify 4 as the lowest
Move it to the sorted set
… etc

The principle of **Abstraction** requires us to remove unnecessary complexity so we can create a general model or rule.

This sequence of steps is currently specific to our sample problem. How can we generalise it?

What did we actually do when we found the 2?

But what did we actually do?

Search through all the cards
Identify 2 as the lowest
Move it to the sorted set
Search through all the cards
Identify 3 as the lowest
Move it to the sorted set
Search through all the cards
Identify 4 as the lowest
Move it to the sorted set
… etc

The principle of **Pattern recognition** requires us to identify repeating patterns which this clearly is.

But what did we actually do?

```
While unsorted values remain:
    Search through all the cards
    Identify the lowest
    Move it to the sorted set
```

We have taken our original one “overwhelming” problem and broken it down into smaller sub-problems.

This is **Decomposition** at work!

I can now solve each of these separately. I continue decomposing my larger problem into smaller problems until I reach chunks I know how to program.

Turning this into code...

```
# The problem
values = [6, 4, 3, 12, 7, 8, 2, 13, 5, 14, 10, 11, 9]

# While unsorted values remain
    # Search through all the values
    # Identify the lowest
    # Move it to the sorted set
```

Note: For the purpose of this exercise, we are going to behave as though sort(), min() and max() do not exist. These are fairly trivial functions so if you can’t write your own alternatives to them there is a problem.

Turning this into code...

```
# The problem
values = [6, 4, 3, 12, 7, 8, 2, 13, 5, 14, 10, 11, 9]

# While unsorted values remain
while len(values) > 0:

    # Find the lowest value

    # Move it to the sorted set
```

This is fairly easy, so doesn’t need further decomposition.

What about the other steps?

Turning this into code...

```
# The problem
values = [6, 4, 3, 12, 7, 8, 2, 13, 5, 14, 10, 11, 9]

# While unsorted values remain
while len(values) > 0:

    # Find the lowest value

    # Move it to the sorted set
```

Not really sure about this yet, so I'll come back to it.

Let's move on to the next bit...

Turning this into code...

```
# The problem
values = [6, 4, 3, 12, 7, 8, 2, 13, 5, 14, 10, 11, 9]

# While unsorted values remain
while len(values) > 0:

    # Find the lowest value

    # Move it to the sorted set
```

Looking at the words "sorted set" it occurs to me I don't have a variable for that yet, so I need to create it.

Turning this into code...

```
# The problem
values = [6, 4, 3, 12, 7, 8, 2, 13, 5, 14, 10, 11, 9]
result = []

# While unsorted values remain
while len(values) > 0:

    # Find the lowest value

    # Move it to the sorted set
```

For my solution

Turning this into code...

```
# The problem
values = [6, 4, 3, 12, 7, 8, 2, 13, 5, 14, 10, 11, 9]
result = []

# While unsorted values remain
while len(values) > 0:

    # Find the lowest value

    # Move it to the sorted set
    result.append( lowest )
```

I know adding a value to the “sorted set” will be easy (once I know what the value is), so I’ll just create the statement now.

But I do realise it doesn’t remove it from the source values, so I turn this into a two step process...

Turning this into code...

```python
# The problem
values = [6, 4, 3, 12, 7, 8, 2, 13, 5, 14, 10, 11, 9]
result = []

# While unsorted values remain
while len(values) > 0:

    # Find the lowest value

    # Add it to the sorted set
    result.append( lowest )

    # Remove it from the source set
    values.remove( lowest )
```

That’s better

Turning this into code...

```
# The problem
values = [6, 4, 3, 12, 7, 8, 2, 13, 5, 14, 10, 11, 9]
result = []

# While unsorted values remain
while len(values) > 0:

    # Find the lowest value

    # Add it to the sorted set
    result.append( lowest )

    # Remove it from the source set
    values.remove( lowest )
```

Enough procrastination, how do solve this bit?

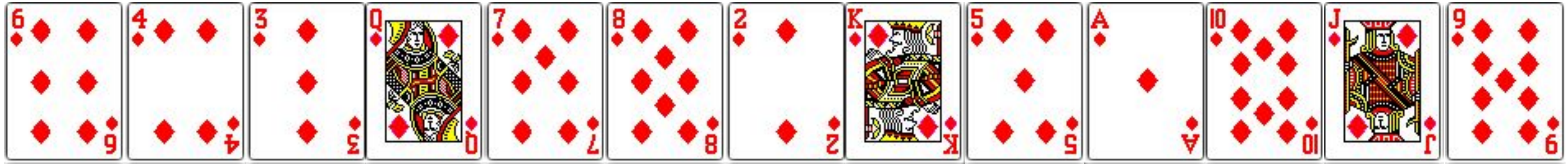
Return to our thinking steps...

Finding the lowest value

How did we identify the 2 as the lowest value?

Decomposition

Identify the steps.



Finding the lowest value

Start with the first card

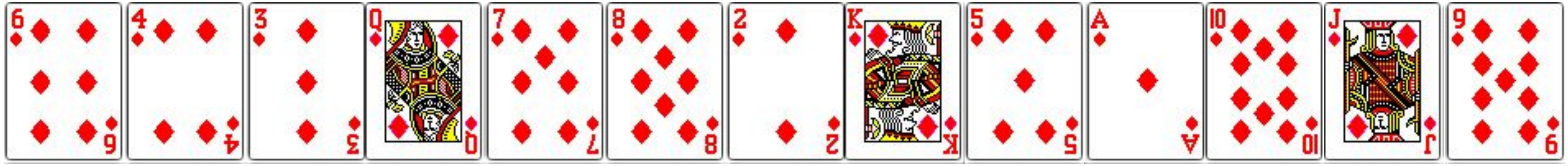
Is the 2nd card lower? If so, remember it as the lowest

Is the 3rd card lower? If so, remember it as the lowest

Is the 4th card lower? If so, remember it as the lowest

Abstraction

Create a generalised model.



Finding the lowest value

Pattern recognition.

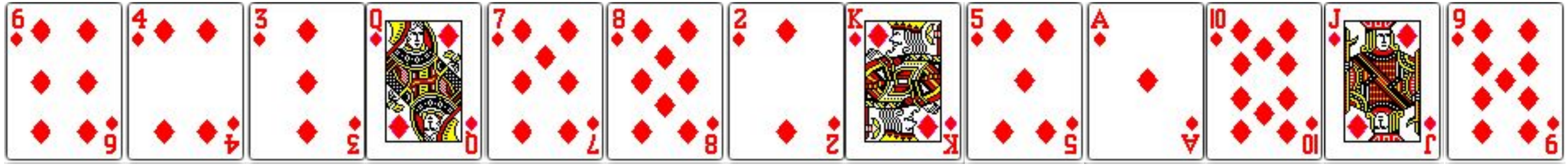
Identify repetition.

Start with the first card

Look at the next card. Is it lower? If so, remember it as the lowest

Look at the next card. Is it lower? If so, remember it as the lowest

Look at the next card. Is it lower? If so, remember it as the lowest



Finding the lowest value

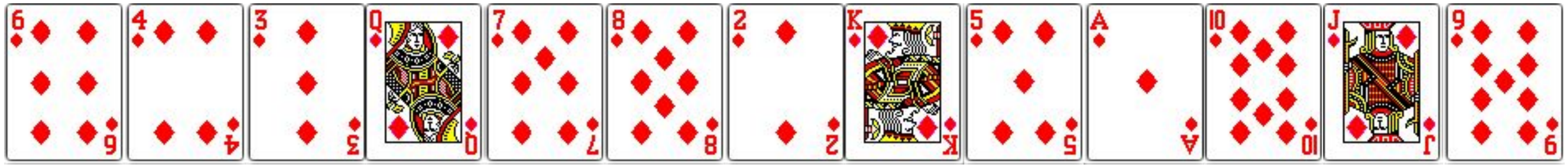
Start with the first card

For all the cards in our problem:

    Look at the next card. Is it lower? If so, remember it as the lowest

Algorithm design

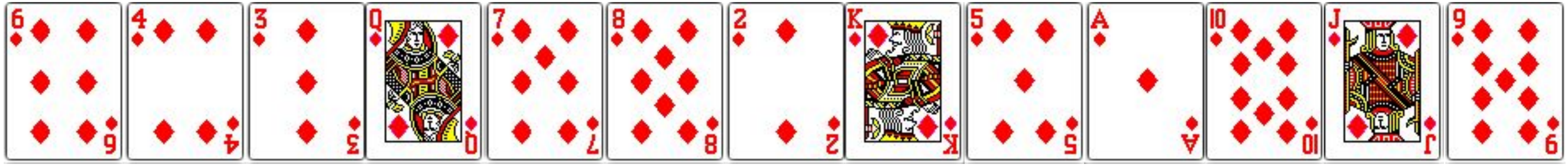
Document our sequence of steps.



Done!

Finding the lowest value

```
Start with the first card
For all the cards in our problem:
    Look at the next card
    Is it lower? If so,
        remember it as the lowest
```



Turning this into code...

```
# Start with the first value
# For all the values in our problem
    # Is it lower? If so...
        # Remember it as the lowest
```

Turning this into code...

```
# Start with the first value
lowest = values[0]

# For all the values in our problem
for n in values:

    # Is it lower? If so...
    if n < lowest:

        # Remember it as the lowest
        lowest = n
```

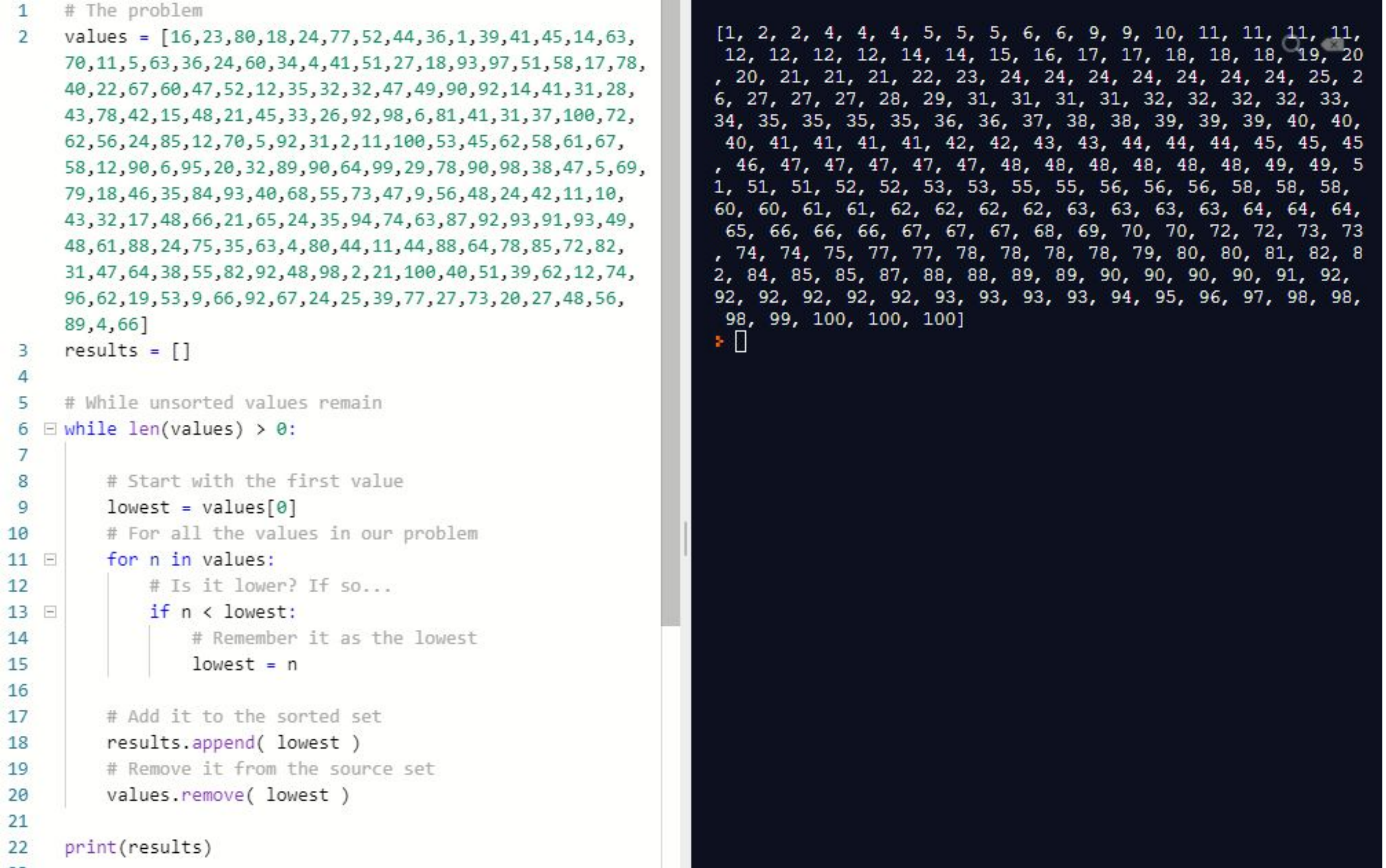
```
# The problem
values = [6, 4, 3, 12, 7, 8, 2, 13, 5, 14, 10, 11, 9]
results = []

# While unsorted values remain
while len(values) > 0:

    # Start with the first value
    lowest = values[0]
    # For all the values in our problem
    for n in values:
        # Is it lower? If so...
        if n < lowest:
            # Remember it as the lowest
            lowest = n

    # Add it to the sorted set
    results.append( lowest )
    # Remove it from the source set
    values.remove( lowest )
```

Solved!

Returning to these steps

1. Start with a small human solvable version of the problem.
2. Keep working through the 4 computational thinking prompts until you devise a solution.
3. **Test your proposed solution works on larger, real-world versions of the problem.**
4. Adapt and repeat the process as necessary.

Does our solution work for decimals? negatives? strings?



```python
# The problem
values = [16,23,80,18,24,77,52,44,36,1,39,41,45,14,63,
70,11,5,63,36,24,60,34,4,41,51,27,18,93,97,51,58,17,78,
40,22,67,60,47,52,12,35,32,32,47,49,90,92,14,41,31,28,
43,78,42,15,48,21,45,33,26,92,98,6,81,41,31,37,100,72,
62,56,24,85,12,70,5,92,31,2,11,100,53,45,62,58,61,67,
58,12,90,6,95,20,32,89,90,64,99,29,78,90,98,38,47,5,69,
79,18,46,35,84,93,40,68,55,73,47,9,56,48,24,42,11,10,
43,32,17,48,66,21,65,24,35,94,74,63,87,92,93,91,93,49,
48,61,88,24,75,35,63,4,80,44,11,44,88,64,78,85,72,82,
31,47,64,38,55,82,92,48,98,2,21,100,40,51,39,62,12,74,
96,62,19,53,9,66,92,67,24,25,39,77,27,73,20,27,48,56,
89,4,66]
results = []

# While unsorted values remain
while len(values) > 0:

    # Start with the first value
    lowest = values[0]
    # For all the values in our problem
    for n in values:
        # Is it lower? If so...
        if n < lowest:
            # Remember it as the lowest
            lowest = n

    # Add it to the sorted set
    results.append( lowest )
    # Remove it from the source set
    values.remove( lowest )

print(results)
```

```
[1, 2, 2, 4, 4, 4, 5, 5, 5, 6, 6, 9, 9, 10, 11, 11, 11, 11,
 12, 12, 12, 12, 14, 14, 15, 16, 17, 17, 18, 18, 18, 19, 20
, 20, 21, 21, 21, 22, 23, 24, 24, 24, 24, 24, 24, 24, 25, 2
6, 27, 27, 27, 28, 29, 31, 31, 31, 31, 32, 32, 32, 32, 33,
34, 35, 35, 35, 35, 36, 36, 37, 38, 38, 39, 39, 39, 40, 40,
 40, 41, 41, 41, 41, 42, 42, 43, 43, 44, 44, 44, 45, 45, 45
, 46, 47, 47, 47, 47, 47, 48, 48, 48, 48, 48, 48, 49, 49, 5
1, 51, 51, 52, 52, 53, 53, 55, 55, 56, 56, 56, 58, 58, 58,
60, 60, 61, 61, 62, 62, 62, 62, 63, 63, 63, 63, 64, 64, 64,
 65, 66, 66, 66, 67, 67, 67, 68, 69, 70, 70, 72, 72, 73, 73
, 74, 74, 75, 77, 77, 78, 78, 78, 78, 79, 80, 80, 81, 82, 8
2, 84, 85, 85, 87, 88, 88, 89, 89, 90, 90, 90, 90, 91, 92,
92, 92, 92, 92, 92, 93, 93, 93, 93, 94, 95, 96, 97, 98, 98,
 98, 99, 100, 100, 100]
>
```

Review: Computational thinking

1. Decomposition
   Can I divide this into sub-problems?

2. Pattern recognition
   Can I find repeating patterns?

3. Abstraction
   Can generalise this to make an overall rule?

4. Algorithm design
   Can I design the programming steps for any of this?

Some other suggestions for beginners

1.	Just start	(A blank screen can be overwhelming)
2.	Don’t start at the start	(It’s not a novel)
3.	Start with something you know	(Such as the UI)
4.	Don’t be afraid to Google	(Prioritise forums such as stackoverflow)
5.	Test & print a lot	

Your turn! Caesar’s cipher

Julius Caesar’s “famous” cipher. It was a substitution method where each letter is replaced by another letter a fixed distance along the alphabet.

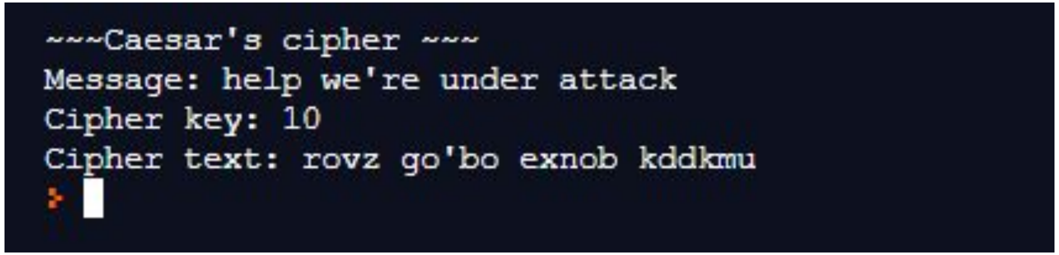
For example if the “cipher key” is 3, then a letter “a” will move up three places to become the letter “d”.

Use the computational thinking process to design an implementation of the Caesar cipher.

This does not require anything we have not already done in class. The challenge is it is a combination of several of those skills in one problem.

Please do not google Caesar cipher, it will defeat the point of the exercise.

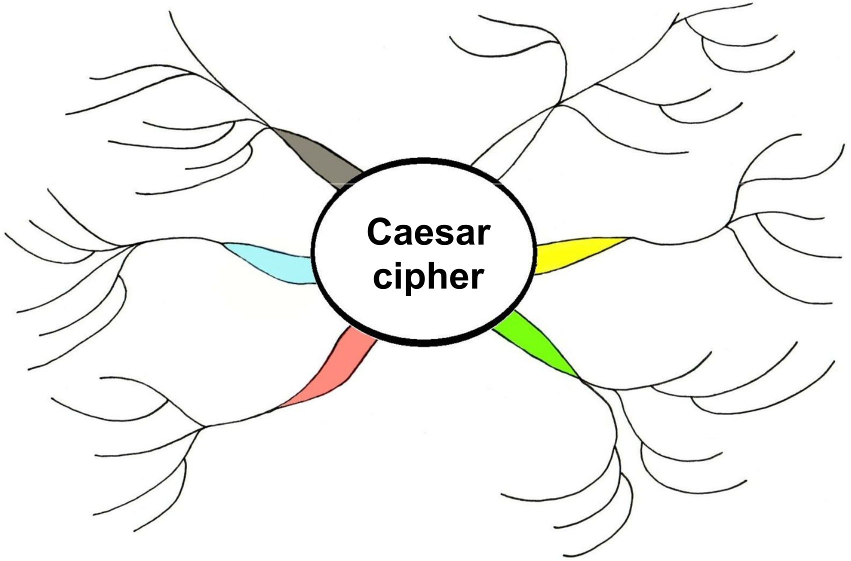
Example input	Cipher key	Example output
attack	3	dwwdfm
defend the castle	1	efgfoe uif dbtumf
captain my captain	5	hfuyfns rd hfuyfns



```
~~~Caesar's cipher ~~~
Message: help we're under attack
Cipher key: 10
Cipher text: rovz go'bo exnob kddkmu
>
```

Your turn! Caesar’s cipher

1. Start with a small human solvable version of the problem.
2. Document your human solvable test run.
3. Create a brainstorm deconstructing the problem into it’s smaller and smaller components.
4. Add your recipe of steps as Python comments into repl.
5. <u>**Then**</u> you may attempt to program it.



Submit where you got to before next lesson.
Don’t worry if you don’t solve it, but do submit your progress (including the test run and brainstorm and your algorithm “recipe of steps”).
I don’t suggest spending more than an hour on it.