Compound Data vs. List

- With compound data, we have the ability to store multiple pieces of related information together
  - These pieces of information can be of different types
  - E.g., Student's have a name (str), an age (int), a major (str), a GPA (float), etc.

- Lists can store multiple pieces of information but these pieces of information all have to be the same conceptual (and actual) type
  - E.g., a list can contain multiple pieces of Student data to represent everyone in CPSC 103
  - E.g., You would **not** have a list of student names and majors all mixed up together. Yes they are both strings and you could technically put all the strings together in a list but conceptually, it doesn't make sense to mix these two unrelated pieces of data together

What is a list?

- Arbitrary sized means you don't know how many data items you have (not that it has to be a very big number of items!)

- Compare these two situations
  1. How would you represent the number of people coming on a hike?

  2. How would you represent the names of all the people coming on a hike?

List[str]

```
from typing import List  # Must appear once at the top of the file when List is used.

# List[str]
# interp. a list of strings

L0 = []
L1 = ["orange", "hi", "truck"]

@typecheck
def fn_for_los(los: List[str]) -> ...: # template based on arbitrary-sized
    # description of the accumulator
    acc = ...        # type: ...
    for s in los:
        acc = ...(s, acc)

    return ...(acc)
```

List[str] Data Defintion (part 1)

```
# List[str]
# interp. a list of strings

L0 = []
L1 = ["orange", "hi", "truck"]
```

This is where we indicate what types of values will be stored in our list.

Make sure it matches with what we actually put in the list in our examples.

List[str] Data Defintion (part 1)

```
# List[str]
# interp. a list of strings

L0 = []
L1 = ["orange", "hi", "truck"]
```

An empty list is still a list (it's just a list with nothing in it).

You should always have an empty list as an example to ensure that your code can work for all types of lists- lists with things in it and lists with nothing in it.

List[str] Data Defintion (part 2)

```
@typecheck
# template based on arbitrary-sized
def fn_for_los(los: List[str]) -> ...:
    # description of the accumulator
    acc = ...        # type: …


    for s in los:
        acc = ...(s, acc)

  return ...(acc)
```

List[str] Data Defintion (part 2)

```
@typecheck
# template based on arbitrary-sized
def fn_for_los(los: List[str]) -> ...:
    # description of the accumulator
    acc = ...        # type: …


    for s in los:
        acc = ...(s, acc)

    return ...(acc)
```

Accumulators

- Accumulators are **variables** that store information collected during your for-loop iterations
  - Often they store information about items previously seen in the list

- You don't have to always call it acc! It can be any name.

- You can have zero or more (not must limited to one!) accumulators in a function

- You can *sometimes* rewrite functions to use or not use accumulators.

Accumulator Example 1

Task: Design a function to check if the number 10 exists in the following list [1, 2, 5, 8, 24, 6].

Do you need an accumulator? If so, how many accumulators do you need?

Answer: None! You don't need to store any information about previous items found in the list.

The second you find the number 10, you are done your task!

Accumulator Example 2

Task: Count the number of times "UBC" appears in the following list:
["apple", "oranges", "UBC", "pears", "UBC"]

Do you need an accumulator? If so, how many accumulators do you need?

Answer: You need 1 accumulator. It stores how many times you have seen UBC so far in the list.

Accumulator Example 3

Task: Find the difference between the maximum and minimum numbers in the list [1, 2, 5, 8, 24, 6].

Do you need an accumulator? If so, how many accumulators do you need?

Answer: 2 accumulators! You need to keep track of the largest and smallest number you have seen in the list so far. Since each variable only stores one piece of information, you need two variables to store the two numbers.

List[str] Data Defintion (part 2)

```python
@typecheck
# template based on arbitrary-sized
def fn_for_los(los: List[str]) -> ...:
    # description of the accumulator
    acc = ...        # type: …


    for s in los:
        acc = ...(s, acc)

    return ...(acc)
```

Super Quick Review: Code Execution

Code executes line by line from top to bottom.

```
x = 1     ⟵
y = 2
x = x+y
y = 4
```

Super Quick Review: Code Execution

Code executes line by line from top to bottom.

```
x = 1
y = 2        ⬅
x = x+y
y = 4
```

Super Quick Review: Code Execution

Code executes line by line from top to bottom.

```
x = 1
y = 2
x = x+y    ⬅
y = 4
```

Super Quick Review: Code Execution

Code executes line by line from top to bottom.

```
x = 1
y = 2
x = x+y
y = 4
```

Super Quick Review: Code Execution

Code executes line by line from top to bottom.

```
x = 1
y = 2
x = x+y
y = 4
```

For loops allow you to execute some lines of code multiple times without having to type everything out repeatedly. The lines of code in the for-loop body will be executed repeatedly until you reach a return statement or you run out of items to look at in the list.

For Loop Basics

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

This is the body of the for loop .

Code that is indented under the "for" without being interrupted by a line of code that is indented the same as the "for" is considered as being part of the for-loop body.

For Loop Basics
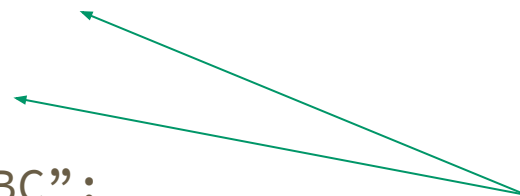
```python
def contains_ubc(unis: List[str]) -> bool:

    for u in unis:

        if u == "UBC":

            return True

    return False
```

This specifies the list we are going to look at. It's not just a random name.

For Loop Basics

```python
def contains_ubc(unis: List[str]) -> bool:

    for u in unis:

        if u == "UBC":

            return True

    return False
```

This is a variable that will change values as the for loop iterates.

Make sure this variable name has not been previously used in this function.

Pre-Class Question #6: For Loops

How many times is the body of the for loop executed when contains_ubc is called with the input ["McGill", "SFU", "UBC", "MIT", "Harvard"]?

```
def contains_ubc(unis: List[str]) -> bool:

    for u in unis:

        if u == "UBC":

            return True

    return False
```

Someone makes a call to contains unis. For example:

```
universities = ["McGill", "SFU", "UBC", "MIT", "Harvard"]
contains_ubc(universities)
```

Pre-Class Question: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

unis

| McGill | SFU | UBC | MIT | Harvard |

Pre-Class Question: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

unis

McGill	SFU	UBC	MIT	Harvard

Pre-Class Question: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

unis

McGill	SFU	UBC	MIT	Harvard

Pre-Class Question: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

If condition is not true, therefore the if-loop code is not executed

unis

McGill	SFU	UBC	MIT	Harvard

u

First Iteration

Pre-Class Question: For Loops

```python
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

← No more code in the for-loop body to execute

unis

McGill	SFU	UBC	MIT	Harvard

↑
u

First Iteration

Pre-Class Question: For Loops

```python
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

unis

McGill	SFU	UBC	MIT	Harvard

u

Second Iteration

Pre-Class Question: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	SFU	UBC	MIT	Harvard

u

Second Iteration

Pre-Class Question: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

unis

McGill	SFU	UBC	MIT	Harvard

u

Second Iteration

Pre-Class Question: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

unis

McGill	SFU	UBC	MIT	Harvard

u

Third Iteration

Pre-Class Question: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":        ⬅

            return True

    return False
```

If condition is true!

unis

McGill	SFU	UBC	MIT	Harvard

⬆

u

Third Iteration

Pre-Class Question: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

Function finishes when it reaches a return statement

unis

McGill	SFU	UBC	MIT	Harvard

u

Third Iteration

Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

How many times does the for loop run?

Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

u

First Iteration

Exercise 1: For Loops

```python
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

u

First Iteration

Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

u

First Iteration

Exercise 1: For Loops

```python
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

u

Second Iteration
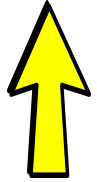
Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":    ⬅

            return True


    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

⬆

u

Second Iteration

Exercise 1: For Loops

```python
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

u

Second Iteration
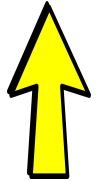
Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

u

Third Iteration

Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":      ⬅

            return True

    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

u

Third Iteration

Exercise 1: For Loops

```python
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

u

Third Iteration

Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

u

Fourth Iteration

Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

u

Fourth Iteration

Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

u

Fourth Iteration

Exercise 1: For Loops

```python
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

u

Fifth Iteration

Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

u

Fifth Iteration

Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

u

Fifth Iteration

Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

u

No more items in the list so the for loop ends and the next line of code after the for loop executes

Exercise 1: For Loops

```python
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

u

No more items in the list so the for loop ends and the next line of code after the for loop executes

Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	SFU	UoT	MIT	Harvard

How many times does the for loop run?

5 times

Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	UBC	UBC	MIT	Harvard

How many times does the for loop run?

Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":    ⬅

            return True

    return False
```

unis

McGill	UBC	UBC	MIT	Harvard

⬆

u

First Iteration

Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

unis

| McGill | UBC | UBC | MIT | Harvard |

u

First iteration

Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	UBC	UBC	MIT	Harvard

u

Second Iteration

Exercise 1: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

unis

McGill	UBC	UBC	MIT	Harvard

u

Second Iteration

Exercise 2: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True

    return False
```

unis

McGill	UBC	UBC	MIT	Harvard

u

Second Iteration

Exercise 2: For Loops

```
def contains_ubc(unis):

    for u in unis:

        if u == "UBC":

            return True


    return False
```

unis

McGill	UBC	UBC	MIT	Harvard

How many times does the for loop run?

2 times

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```

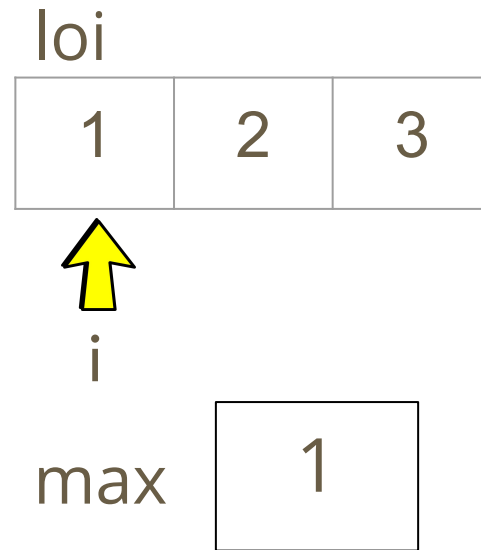Someone makes a call to the maximum function:

```
nums = [1, 2, 3]
maximum(nums)
```

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int      ⬅

    for i in loi:

        if i > max:

            max = i

    return max
```

loi

1	2	3

max

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```

loi

1	2	3

max   -1

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```
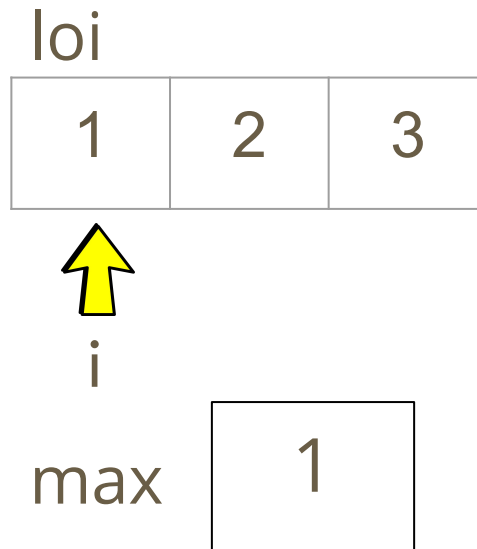
loi

1	2	3

i
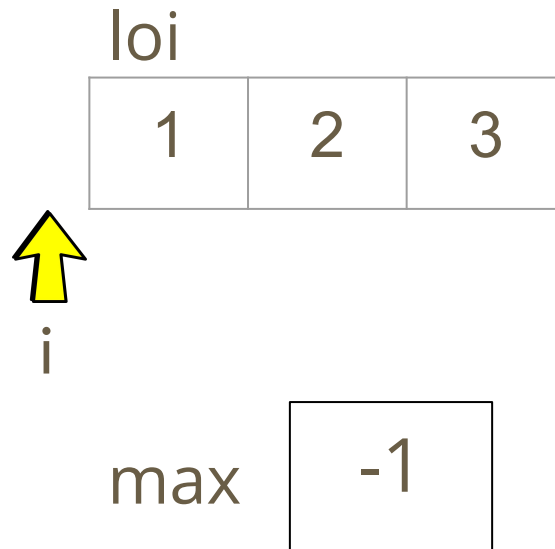
max   -1

Just before the
first iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```
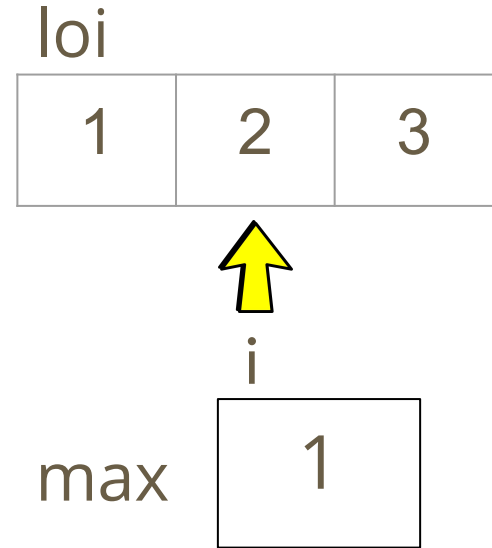
loi

| 1 | 2 | 3 |

i

max | -1 |

First iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:    ⬅

            max = i

    return max
```

loi

| 1 | 2 | 3 |

⬆

i

max   -1

First iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```
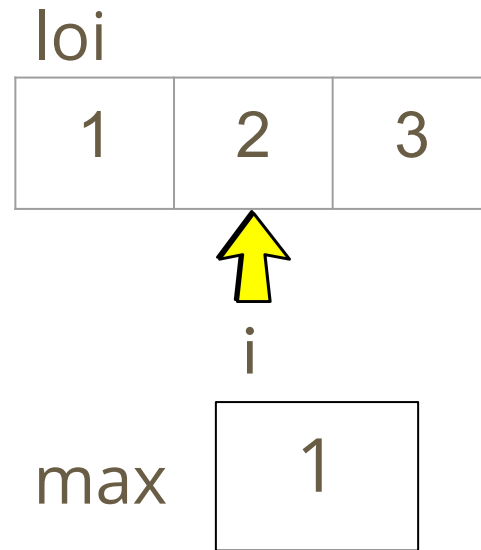
loi

1	2	3

i

max -1

First iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```
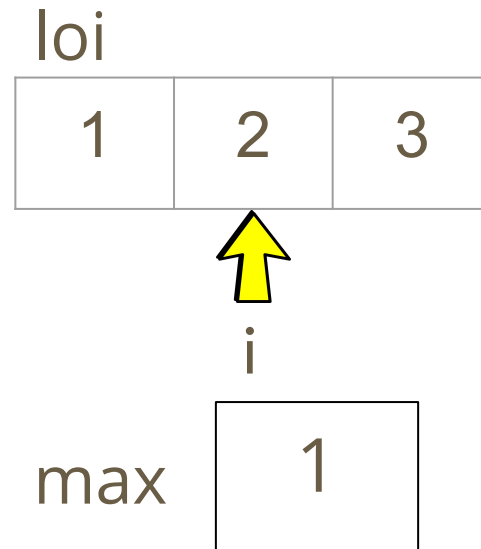
loi

| 1 | 2 | 3 |

i

max   1

First iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```
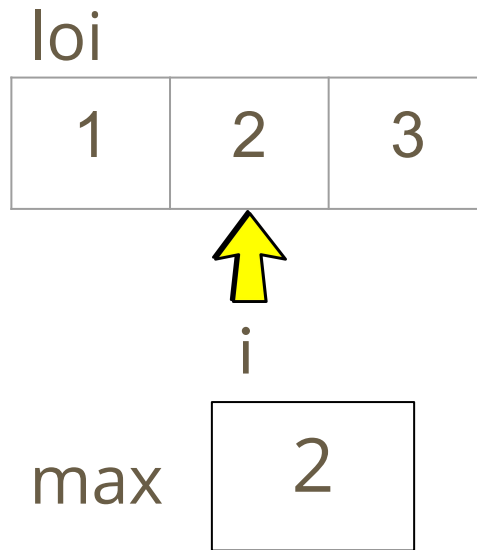
loi

| 1 | 2 | 3 |

i

max | 1 |

First iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```

loi

| 1 | 2 | 3 |

i

max | 1 |

Starting the
second iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```
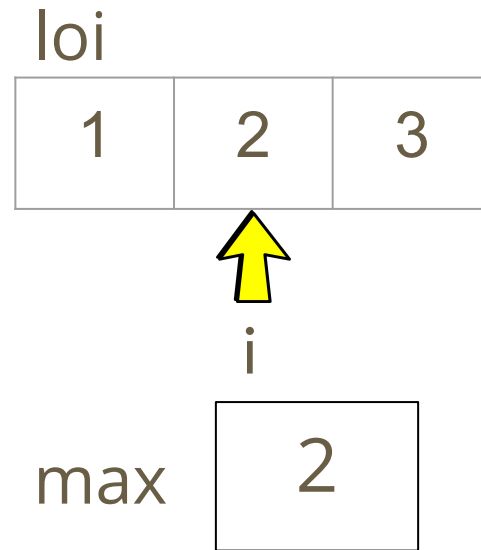
loi

1	2	3

i

max 1

Second iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```
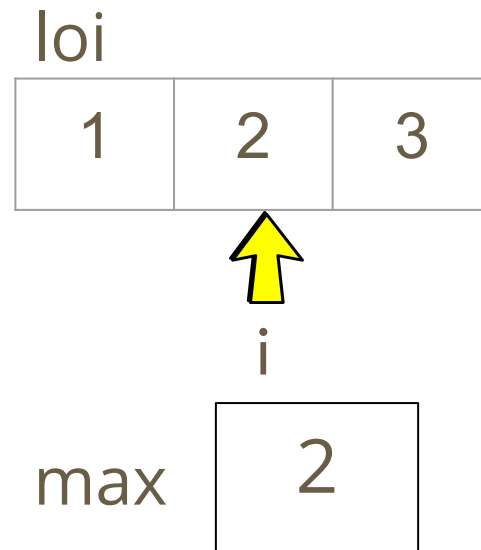
loi

1	2	3

i

max 1

Second iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```python
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i       ⬅

    return max
```

loi

1	2	3

⬆

i

max [1]

Second iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```
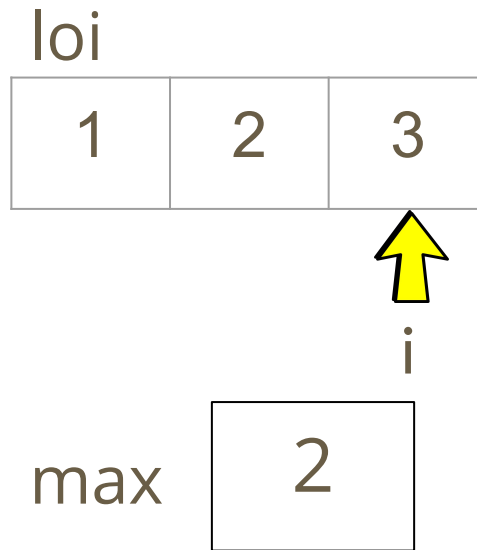
loi

1	2	3

i

max 2

Second iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```

loi

1	2	3

i

max

2

Second iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```

loi

| 1 | 2 | 3 |

i

max 2

Starting the third
iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```
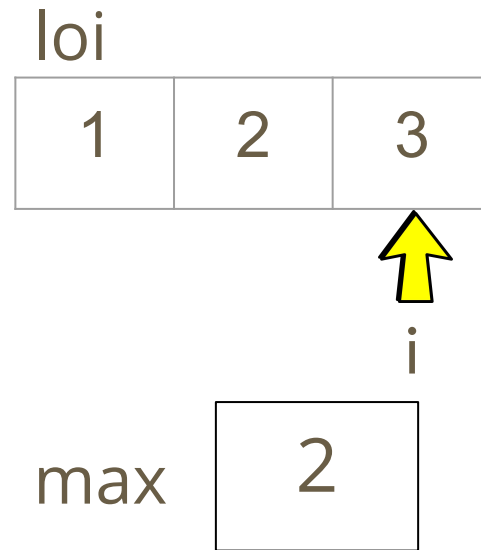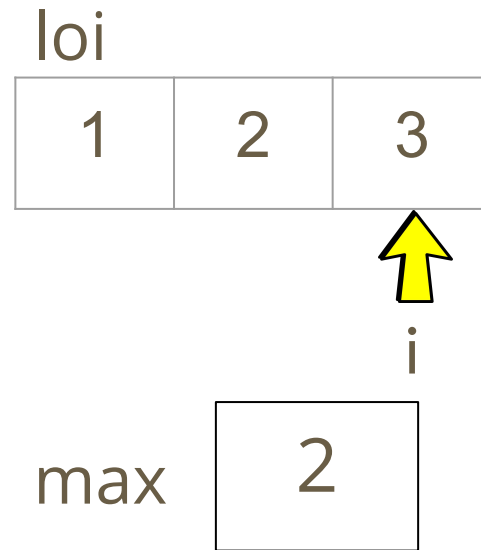
loi

1	2	3

i

max   2

Starting the third
iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```
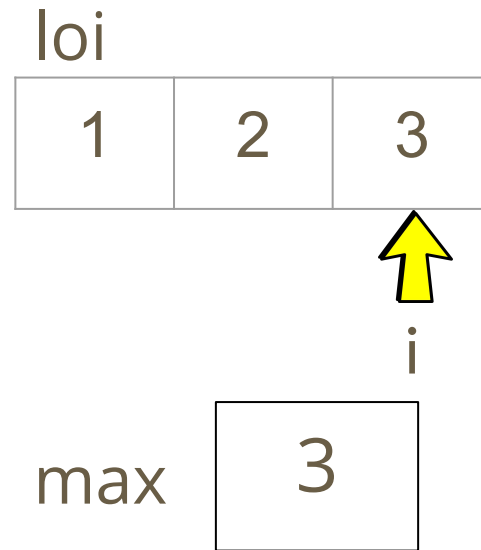
loi

1	2	3

i

max   2

Third iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i    ⬅

    return max
```

loi

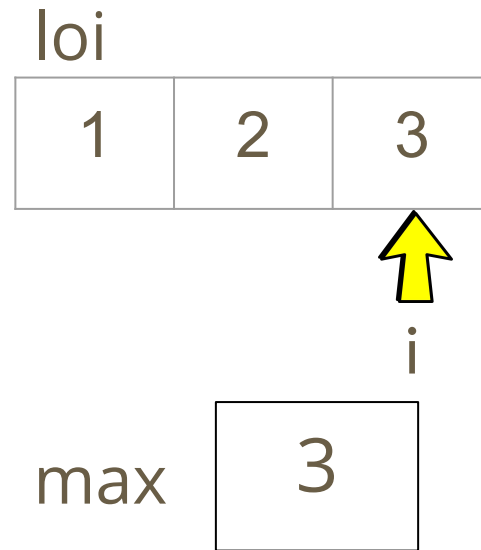| 1	2	3

⬆
i

max 2

Third iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```

loi

| 1 | 2 | 3 |

i

max   3

Third iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```

loi

1	2	3

i

max   3

Third iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```
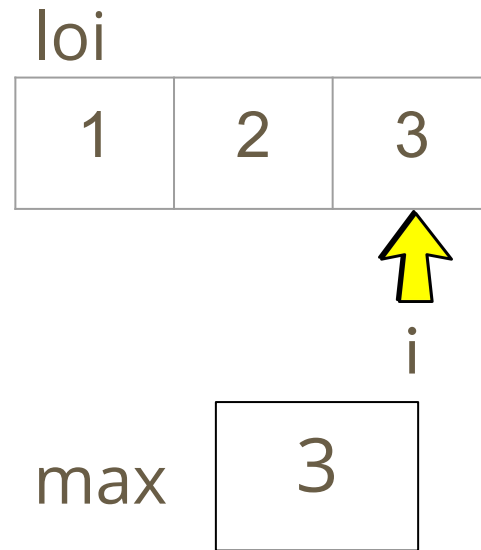
loi

1	2	3

i

max 3

Just before
fourth iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```

loi

1	2	3

i

max 3

Just before
fourth iteration

Exercise 3: For Loop Tracing with an Accumulator

From Q#1 on the worksheet:

```
def maximum(loi: List[int]) -> int:
    max = -1 # type: int

    for i in loi:

        if i > max:

            max = i

    return max
```

loi

| 1 | 2 | 3 |

i

max

3

No more things in the list. For loop done!