LISTS

Objectives

- Describe, create and access a list data structure
- Use built in methods to modify and copy lists
- Iterate over lists using loops and list comprehensions
- Work with nested lists to build more complex data structures

What is a List?

It's just a collection or grouping of items!

How are Lists useful?

A fundamental data structure for organizing collections of items

```
first_task = "Install Python"
second_task = "Learn Python"
third_task = "Take a break"
```

No ordering!

What Lists Look Like

A fundamental data structure for organizing data

```
tasks = ["Install Python", "Learn Python", "Take a break"]
```

Comma separated values

```
first_task = "Install Python"
second_task = "Learn Python"
third_task = "Take a break"

tasks = [first_task, second_task, third_task]
```

How Many Elements Exist?

Let's use our first built in function for lists - **len**

```
tasks = ["Install Python", "Learn Python", "Take a break"]

len(tasks) # 3
```

Another Way to Make a List

Using another built in function called `list()`

```
tasks = list(range(1, 4))
tasks
[1, 2, 3]
```

We'll use this technique of list-making more later with several other data types!

Accessing Values in a List

```
friends = ["Ashley", "Matt", "Michael"]
```

Like ranges, lists ALWAYS start counting at zero.
So the first element lives at index 0.

```
print(friends[0]) # 'Ashley'
print(friends[2]) # 'Michael'
print(friends[3]) # IndexError
```

Accessing Values from the End

```
friends = ["Ashley", "Matt", "Michael"]
```

You can use a negative number to index backwards

```
print(friends[-1]) # 'Michael'
print(friends[-3]) # 'Ashley'
print(friends[-4]) # IndexError
```

Check if a Value is in a List

```
friends = ["Ashley", "Matt", "Michael"]

"Ashley" in friends # True

"Colt" in friends # False
```

Accessing All Values in a List

We could print out each value....

```
numbers = [1,2,3,4]

print(numbers[0]) # 1
print(numbers[1]) # 2
print(numbers[2]) # 3
print(numbers[3]) # 4
```

Can we do better?

Accessing All Values in a List

There are a few ways - let's start with a for loop!

```
numbers = [1,2,3,4]

for number in numbers:
    print(number)

# 1
# 2
# 3
# 4
```

Accessing All Values in a List

Now let's use a while loop!

```
numbers = [1, 2, 3, 4]
i = 0

while i < len(numbers):
    print(numbers[i])
    i += 1

# 1
# 2
# 3
# 4
```

List Methods

Working with lists is very common - there are quite a few things we can do!

append

Add an item to the end of the list.

```
first_list = [1, 2, 3, 4]

first_list.append(5)

print(first_list) # [1, 2, 3, 4, 5]
```

extend

Add to the end of a list all values passed to extend

```
first_list = [1, 2, 3, 4]

first_list.append(5, 6, 7, 8) # does not work!

first_list.append([5, 6, 7, 8])

print(first_list) # [1, 2, 3, 4,  [5, 6, 7, 8]]

correct_list = [1, 2, 3, 4]

correct_list.extend([5, 6, 7, 8])

print(correct_list) # [1, 2, 3, 4, 5, 6, 7, 8]
```

insert

Insert an item at a given position.

```
first_list = [1, 2, 3, 4]

first_list.insert(2, 'Hi!')

print(first_list) # [1, 2, 'Hi!', 3, 4]

first_list.insert(-1, 'The end!')

print(first_list) # [1, 2, 'Hi!', 3, 'The end!', 4]
```

clear

Remove all items from the list.

```
first_list = [1, 2, 3, 4]

first_list.clear()

print(first_list) # []
```

pop

- Remove the item at the given position in the list, and return it.
- If no index is specified, removes & returns last item in the list.

```
first_list = [1, 2, 3, 4]

first_list.pop() # 4

first_list.pop(1) # 2
```

remove

- Remove the first item from the list whose value is *x*.
- Throws a ValueError if the item is not found.

```
first_list = [1, 2, 3, 4, 4, 4]

first_list.remove(2)

print(first_list) # [1, 3, 4, 4, 4]

first_list.remove(4)

print(first_list) # [1, 3, 4, 4]
```

del

Deletes a value from a list.

```
first_list = [1, 2, 3, 4]

del first_list[3]

print(first_list) # [1, 2, 3]

del first_list[1]

print(first_list) # [1, 3]
```

YOUR TURN

index

returns the index of the specified item in the list

```
numbers = [5, 6, 7, 8, 9, 10]

numbers.index(6) # 1
numbers.index(9) # 4
```

Can specify start and end

```
numbers = [5, 5, 6, 7, 5, 8, 8, 9, 10]

numbers.index(5) # 0
numbers.index(5, 1) # 1
numbers.index(5, 2) # 4

numbers.index(8, 6, 8) # 6
```

count

return the number of times *x* appears in the list

```
numbers = [1, 2, 3, 4, 3, 2, 1, 4, 10, 2]

numbers.count(2) # 3
numbers.count(21) # 0
numbers.count(3) # 2
```

reverse

reverse the elements of the list (in-place)

```
first_list = [1, 2, 3, 4]

first_list.reverse()

print(first_list) # [4, 3, 2, 1]
```

sort

sort the items of the list (in-place)

```
another_list = [6, 4, 1, 2, 5]

another_list.sort()

print(another_list) # [1, 2, 4, 5, 6]
```

join

- technically a **String method** that takes an iterable argument
- **concatenates** (combines) a copy of the base string **between** each item of the iterable
- returns a new string
- can be used to make sentences out of a list of words by joining on a space, for instance:

```
words = ['Coding', 'Is', 'Fun!']

' '.join(words) # 'Coding is Fun!'
```

another example:

```
name = ['Mr', "Steele"]

'. '.join(name) # 'Mr. Steele'
```

List Methods *continued*

But wait....there's more!

Try these out as well!

YOUR TURN

Slicing

Make new lists using slices of the old list!

```
some_list[start:end:step]
```

First Parameter for Slice: start

what index to start slicing from

```
first_list = [1, 2, 3, 4]

first_list[1:] # [2, 3, 4]

first_list[3:] # [4]
```

If you enter a negative number, it will start the slice that many back from the end

```
first_list[-1:] # [4]

first_list[-3:] # [2, 3, 4]
```

Second Parameter for Slice: end

The index to copy up to (exclusive counting).

```
first_list = [1, 2, 3, 4]

first_list[:2] # [1, 2]

first_list[:4] # [1, 2, 3, 4]

first_list[1:3] # [2, 3]
```

With negative numbers, how many items to exclude from the end (i.e. indexing by counting backwards)

```
first_list[:-1] # [1, 2, 3]

first_list[1:-1] # [2, 3]
```

Third Parameter for Slice: step

- "step" in Python is basically the number to count at a time
- same as step with range!
- for example, a step of 2 counts every other number (1, 3, 5)

```
first_list = [1, 2, 3, 4, 5, 6]

first_list[1::2] # [2, 4, 6]

first_list[::2] # [1, 3, 5]
```

with negative numbers, reverse the order

```
first_list[1::-1] # [2, 1]

first_list[:1:-1] # [6, 5, 4, 3]

first_list[2::-1] # [3, 2, 1]
```

Tricks with Slices

Reversing lists / strings

```
string = "This is fun!"

string[::-1]
```

Modifying portions of lists

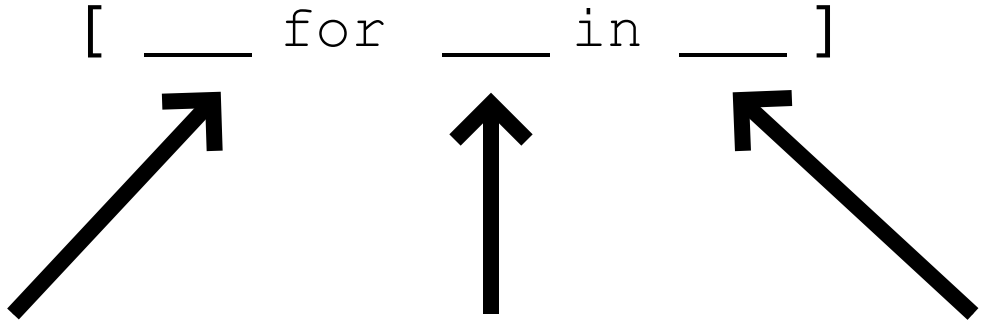
```
numbers = [1, 2, 3, 4, 5]
numbers[1:3] = ['a','b','c']

print(a) # [1, 'a', 'b', 'c', 4, 5]
```

YOUR TURN

List Comprehension

What does it do?

the syntax



List Comprehension vs Looping

```
numbers = [1, 2, 3, 4, 5]
doubled_numbers = []

for num in numbers:
    doubled_number = num * 2
    doubled_numbers.append(doubled_number)

print(doubled_numbers) # [2, 4, 6, 8, 10]
```

```
numbers = [1, 2, 3, 4, 5]

doubled_numbers = [num * 2 for num in numbers]

print(doubled_numbers) # [2, 4, 6, 8, 10]
```

List Comprehension Examples

```
name = 'colt'

[char.upper() for char in name] # ['C', 'O', 'L', 'T']
```

```
friends = ['ashley', 'matt', 'michael']

[friend[0].upper() for friend in friends] # ['Ashley', 'Matt', 'Michael']
```

More Examples

```
[num*10 for num in range(1,6)] # [10, 20, 30, 40, 50]
```

```
[bool(val) for val in [0, [], '']] # [False, False, False]
```

```
numbers = [1, 2, 3, 4, 5]

string_list = [str(num) for num in numbers]

print(string_list) # ['1', '2', '3', '4', '5']
```

LC with Conditional Logic

```
numbers = [1, 2, 3, 4, 5, 6]

evens = [num for num in numbers if num % 2 == 0]

odds = [num for num in numbers if num % 2 != 0]
```

```
[num*2 if num % 2 == 0 else num/2 for num in numbers]

# [0.5, 4, 1.5, 8, 2.5, 12]
```

```
with_vowels = "This is so much fun!"

''.join(char for char in with_vowels if char not in "aeiou")

# "Ths s s mch fn!"
```

YOUR
TURN

Nested Lists

Lists can contain any kind of element, even other lists!

```
nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Why?

Complex data structures - matricies

Game Boards / Mazes

Rows and Columns for visualizations, tabulation and grouping data

Accessing Nested Lists

```
nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

nested_list[0][1] # 2

nested_list[1][-1] # 6
```

Printing Values in Nested Lists

```
nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

for l in nested_list:
    for val in l:
        print(val)


# 1
# 2
# 3
# 4
# 5
# 6
# 7
# 8
```

Nested List Comprehension

```
nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

[[print(val) for val in l] for l in nested_list]

# 1
# 2
# 3
# 4
# 5
# 6
# 7
# 8
```

Another Example

```
board = [[num for num in range(1,4)] for val in range(1,4)]

print(board) # [[1, 2, 3], [1, 2, 3], [1, 2, 3]]

[["X" if num % 2 != 0 else "O" for num in range(1,4)] for val in range(1,4)]

# [['X', 'O', 'X'], ['X', 'O', 'X'], ['X', 'O', 'X']]
```

YOUR TURN

Swapping Values

```
names = ["James", "Michelle"]

names[0], names[1] = names[1], names[0]

print(names) # ['Michelle', 'James']
```

When Do You Need to Swap?

shuffling

sorting

algorithms

Recap

- lists are fundamental data structures for ordered information
- lists can be include any type, even other lists!
- we can modify lists using a variety of methods
- slices are quite useful when making copies of lists
- list comprehension is used everywhere when iterating over lists, strings, ranges and even more data types!
- nested lists are essential for building more complex data structures like matrices, game boards and mazes
- swapping is quite useful when shuffling or sorting

YOUR TURN