

Lists and Dictionaries

Cont....

MAD 102

Nested lists

- Lists can contain lists.
 - These are known as nested lists.

The diagram illustrates a nested list structure. A code snippet is shown with three lines of code. The first line is a comment. The second line defines a list named 'crews' containing three sublists. The third line is a comment. Brackets are used to label the components: a large bracket under the entire list is labeled 'Main List', and three smaller brackets above the sublists are labeled 'List 0', 'List 1', and 'List 2' respectively. The word 'Nested Lists' is centered above the sublists.

```
-  
3 crews = [['Mal', 'Washburne', 'Zoe'], ['Han', 'Chewie'], ['Kirk', 'Spock', 'McCoy']]  
4
```

Nested Lists

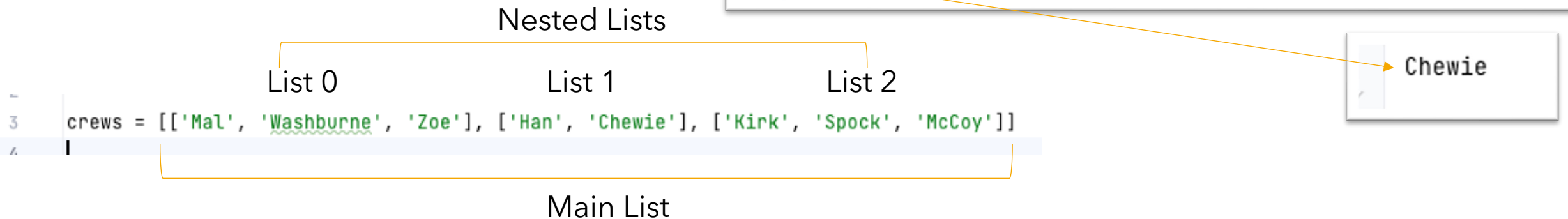
List 0 List 1 List 2

Main List

Nested Lists

- To access the nested list, use the index to get the location of the nested list, and another index value to get the element in the nested list
 - For example – to access Chewie – it is in list index 1, at index position 1

```
3 crews = [['Mal', 'Washburne', 'Zoe'], ['Han', 'Chewie'], ['Kirk', 'Spock', 'McCoy']]
4 print(crews[1][1])
```



Nested Lists

- Can use a for loop, with a nested for loop, to iterate over the entire contents


```
3 crews = [['Mal', 'Washburne', 'Zoe'], ['Han', 'Chewie'], ['Kirk', 'Spock', 'McCoy']]
4
5 → for position, crew in enumerate(crews):
6     print('=' * 20)
7     print(f'Crew #{position + 1}')
8     print('=' * 20)
9     → for member in crew:
10         print(member)
11
```

```
=====
Crew #1
=====
Mal
Washburne
Zoe
=====
Crew #2
=====
Han
Chewie
=====
Crew #3
=====
Kirk
Spock
McCoy
```

List slicing

- Using **slice notation**, you can create new lists with just the elements you want
- **list[firstIndex : lastIndex]** – where the **firstIndex** is included, but the **lastIndex** is not
- -ve values start at the end

```
3 crew = ['Kirk', 'Spock', 'McCoy', 'Uhura', 'Checkov', 'Scotty', 'Sulu']  
4 print(crew[0:3])
```



```
['Kirk', 'Spock', 'McCoy']
```

List slicing

- Slice notation allows you to not provide an index as part of the argument values
 - Absence of a value means – *from start or to end*

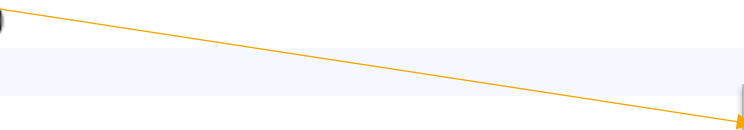
list[: 2] – *means from the start to index 2*

list[3:] – *means from the index position 4 to the end*

List slicing

- Slice notation also allows for a **stride** value
 - Indicates how many elements are skipped

```
-  
3 crew = ['Kirk', 'Spock', 'McCoy', 'Uhura', 'Checkov', 'Scotty', 'Sulu']  
4 print(crew[::2])  
5
```



```
['Kirk', 'McCoy', 'Checkov', 'Sulu']
```



List comprehensions

- A construct called list comprehensions allows you to modify every element in a list the same way
 - The construct iterates over a list
 - Modifies each element
 - Returns a new list of modified elements

List comprehensions

- Made up of three components
 1. Expression used to evaluate each element
 2. Loop variable to bind the the current element
 3. Iterable object to iterate over
- Is surrounded by []
- Contains the keywords for and in to separate the expression from the loop and the loop variable from the iterable object

List comprehensions

- We want to increase all our prices by 5%

```
3 prices = [23.45, 22.99, 19.99, 9.95]
4
5 price_increase = [price * 1.05 for price in prices]
6
7 print(price_increase)
8
```

```
[24.6225, 24.139499999999998, 20.9895, 10.4475]
```

List comprehensions

- List comprehensions can be applied based on a condition
 - Remember - this makes a new list, but only uses the elements that meet the current condition

```
prices = [23.45, 22.99, 19.99, 9.95]
```

```
price_increase = [price * 1.05 for price in prices if price > 20]
```

```
print(price_increase)
```

```
[24.6225, 24.139499999999998]
```

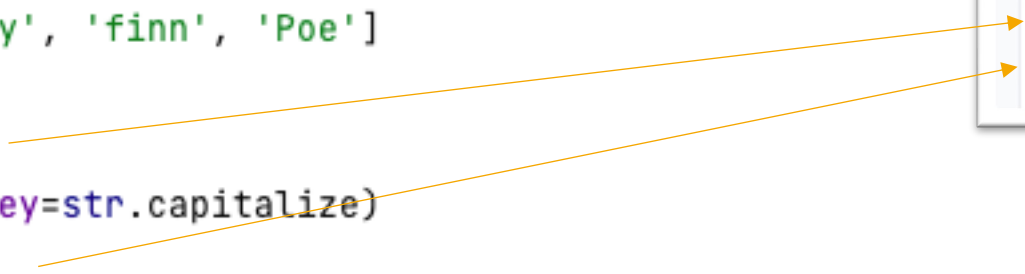
List sorting

- Two methods can be used for sorting a list
 - The **sort()** method will place the list in alphabetical or numerical order
 - The **sorted()** method provides the same sorting - but it returns a new list instead of just modifying it in place
- Optional key argument that specified a function to be applied to each element **prior** to being compared
 - Useful to use `.lower` or `.upper` or `.capitalize` as a key

List sorting

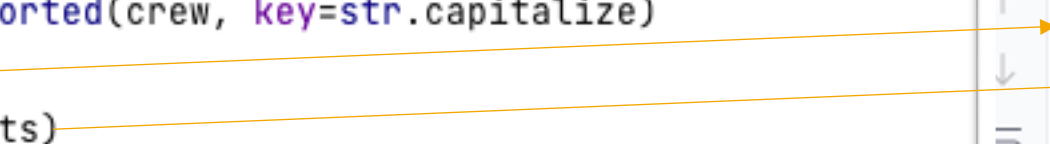
- Note order without the capitalize and with

```
3 crew = ['rey', 'finn', 'Poe']
4 crew.sort()
5 print(crew)
6 crew.sort(key=str.capitalize)
7 print(crew)
```



```
['Poe', 'finn', 'rey']
['finn', 'Poe', 'rey']
```

```
3 crew = ['rey', 'finn', 'Poe']
4 results = sorted(crew, key=str.capitalize)
5 print(crew)
6 print(results)
```

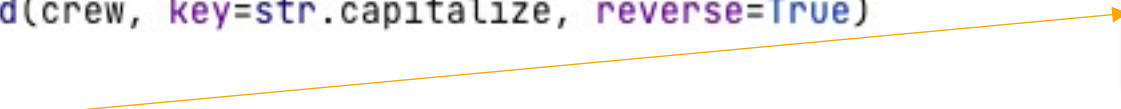


```
['rey', 'finn', 'Poe']
['finn', 'Poe', 'rey']
```

List sorting

- To sort highest to lowest or reverse alphabetical, use the reverse keyword with a value of true

```
3 crew = ['rey', 'finn', 'Poe']  
4 results = sorted(crew, key=str.capitalize, reverse=True)  
5 print(crew)  
6 print(results)  
7
```



```
['rey', 'finn', 'Poe']  
['rey', 'Poe', 'finn']
```

Dictionaries

- Container object that store content in key and value pairs
 - The dict type implements a dictionary in Python
- Dictionaries are created by placing key : value in { }
- Using dict() function – passing in keys assigned to values or an array of tuples

```
example = {'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander'}  
function_example = dict(Kirk='Captain', Spock='Commander', McCoy='Lt. Commander')  
function_example_with_tuples = dict([('Kirk', 'Captain'), ('Spock', 'Commander'), ('McCoy', 'Lt. Commander')])  
  
print(example)  
print(function_example)  
print(function_example_with_tuples)
```

```
{'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander'}  
{'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander'}  
{'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander'}
```

Dictionaries

- Access elements using the key to get the value

```
example = {'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander'}  
print(example['Kirk'])
```

Captain

- Add an entry using the key (will modify if the key exists)

```
example = {'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander'}  
example['Sulu'] = 'Lieutenant'  
print(example)
```

```
{'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander', 'Sulu': 'Lieutenant'}
```


Dictionaries

- Delete the entry using del and the key

```
3 example = {'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander', 'Sulu': 'Lieutenant'}  
4 del example['Sulu']  
5 print(example)
```

```
{'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander'}
```

- Checks to see if the key is present using in

```
example = {'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander', 'Sulu': 'Lieutenant'}  
print('Sulu' in example)
```

```
True
```

Common dictionary methods

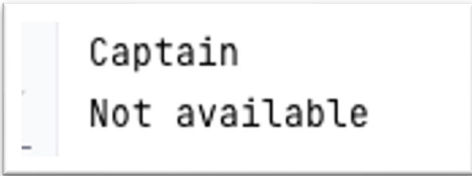
- The **clear()** method will remove all items from a dictionary

```
example = {'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander', 'Sulu': 'Lieutenant'}  
example.clear()  
print(example)
```



- The get(key, default) method reads the value associated with the key – or returns the default value

```
3 example = {'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander', 'Sulu': 'Lieutenant'}  
4 print(example.get('Kirk', 'Not available'))  
5 print(example.get('Chapel', 'Not available'))  
/
```



Common dictionary methods

- The **update()** method allows you to merge two dictionaries into one

```
3 example = {'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander'}
4 example.update({'Sulu': 'Lieutenant'})
5 print(example)
```

```
{'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander', 'Sulu': 'Lieutenant'}
```

- The **pop(key, default)** removes and returns the value or the default if not found

```
3 example = {'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander'}
4 print(example.pop('Kirk', 'Not found'))
5 print(example.pop('Scott', 'Not found'))
```

```
Captain
Not found
;
```

Iterating over a dictionary

- The for loop can be used to iterate over a dictionary
 - Ordering is determined by the hash value created by the Python interpreter for the key values

```
3 example = {'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander'}  
4 for key in example:  
5     print(key)
```

Kirk
Spock
McCoy

```
example = {'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander'}  
for key in example:  
    print(example[key])
```

Captain
Commander
Lt. Commander

Iterating over a dictionary

- A **view object** is created by three useful dictionary methods
 - This is read-only access to keys and values
- They are the **items()**, **keys()**, **values()** methods
 - Items() returns a tuple of key and value
 - Keys() returns just the keys
 - Values returns just the values

Iterating over a dictionary

- Handling items

```
3 example = {'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander'}  
4 for name, rank in example.items():  
5     print(f'{name} holds the rank of {rank}.')  
6 |
```

```
Kirk holds the rank of Captain.  
Spock holds the rank of Commander.  
McCoy holds the rank of Lt. Commander.
```

Iterating over a dictionary

- Handling keys

```
3 example = {'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander'}  
4 for name in example.keys():  
5     print(f'{name} is part of the crew.')  
6
```

```
Kirk is part of the crew.  
Spock is part of the crew.  
McCoy is part of the crew.
```

Iterating over a dictionary

- Handling values

```
3 example = {'Kirk': 'Captain', 'Spock': 'Commander', 'McCoy': 'Lt. Commander'}
4 for rank in example.values():
5     print(f'{rank} is a rank on our crew.')
6
```

```
Captain is a rank on our crew.
Commander is a rank on our crew.
Lt. Commander is a rank on our crew.
```


Dictionary nesting

- Dictionaries can be nested

```
students = {'Harry': {'Potions': 88, 'Dark Arts': 83}, 'Hermione': {'Dark Arts': 100, 'Potions': 100},  
           'Ron': {'Potions': 67, 'Dark Arts': 74}}
```

- Lookup values using keys

```
3 students = {'Harry': {'Potions': 88, 'Dark Arts': 83}, 'Hermione': {'Dark Arts': 100, 'Potions': 100},  
4           'Ron': {'Potions': 67, 'Dark Arts': 74}}  
5  
6 print(students['Harry']['Potions'])
```

88