Lesson 4: Non-Primitive Data Types

August 4, 2020

1 Lesson 4: Non-Primitive Data Types

In this lesson, we will learn about...

- Lists
- Tuples
- Sets
- Dictionaries

While there are many others, we will look at the four main **collection** data types. Many have roots in math. They can hold any data type, and even multiple types at once. They have many similarities but some important differences.

Type	Representation	Ordered?	Changeable / Mutable?	Duplicates?	Special Qualities
List	`[]`	Yes	Yes	Yes	
Tuple	`()`	Yes	No	Yes	
Set	`{}`	No	Yes	No	Unindexed
Dictionary	`{:}`	No	Yes	No	Indexed by key

2 List

This is the closest data type to the traditional array.

[1]:
```
fruits = ["apple", "banana", "orange"]

print(fruits)
print(len(fruits))
print(fruits[1])
print(fruits[-1])
print(fruits[0:2])
print("banana" in fruits)
print(fruits[0].upper())
print(fruits[1][1])
```

```
['apple', 'banana', 'orange']
3
banana
orange
['apple', 'banana']
True
APPLE
a
```

- You can use the same operators as strings to determine the length, get specific elements, and see if an element is in the list.
- Since this list stores strings, you can then even use string functions on the result!

[2]:
```
fruits = ["apple", "banana", "orange"]
veggies = ["carrot", "squash"]

fruits[1] = "pineapple"
print(fruits)

fruits.append("blueberry")
fruits.insert(1, "tomato")
print(fruits)

fruits.remove("apple")
print(fruits)

popped = fruits.pop()
print(fruits)
print(popped)

print(fruits + veggies)

fruits.clear()
print(fruits)
```

```
['apple', 'pineapple', 'orange']
['apple', 'tomato', 'pineapple', 'orange', 'blueberry']
['tomato', 'pineapple', 'orange', 'blueberry']
['tomato', 'pineapple', 'orange']
blueberry
['tomato', 'pineapple', 'orange', 'carrot', 'squash']
[]
```

- You can update an item a a given index by referencing its postiion and assigning a value
- You can `insert()` a value at a given index. This pushes other values back.
- You can `remove()` a given value. An error is thrown if it doesn't exist.
- You can `pop()` the last item (or specify an index) off the list. This also returns what it popped.
- Concatenate lists by adding them together.

- `clear()` a list to remove all its contents.

There are *many* more list methods available. A reference can be found here.

3 Tuple

These are unordered and unchangeable / immutable.

[3]:
```
fruits = ("apple", "banana", "orange")

print(fruits)
print(len(fruits))
print(fruits[1])
print(fruits[-1])
print(fruits[0:2])
print("banana" in fruits)
print(fruits[0].upper())
print(fruits[1][1])
```

```
('apple', 'banana', 'orange')
3
banana
orange
('apple', 'banana')
True
APPLE
a
```

- You can use the same operators as strings to determine the length, get specific elements, and see if an element is in the tuple.
- Since this tuple stores strings, you can then even use string functions on the result!

[4]:
```
fruits = ("apple", "banana", "orange")
veggies = ("carrot", "squash")

# fruits[1] = "pineapple" ## Illegal!
# print(fruits)

# fruits.append("blueberry") ## Illegal!
# fruits.insert(1, "tomato") ## Illegal!
# print(fruits)

# fruits.remove("apple") ## Illegal!
# print(fruits)

# popped = fruits.pop() ## Illegal!
# print(fruits)
```

```
# print(popped)

print(fruits + veggies)

# fruits.clear() ## Illegal!
# print(fruits)
```

```
('apple', 'banana', 'orange', 'carrot', 'squash')
```

- Concatenate tuples by adding them together.

There are only two methods available for tuples, `count()` and `index()`. Learn more here.

4 Sets

These have no order and do not allow duplicates. They are also unindexed.

[5]:
```
fruits = {"apple", "banana", "orange"}

print(fruits)
print(len(fruits))
# print(fruits[1]) ## Illegal!
# print(fruits[-1])  ## Illegal!
# print(fruits[0:2])  ## Illegal!
print("banana" in fruits)
# print(fruits[0].upper())  ## Illegal!
# print(fruits[1][1])  ## Illegal!
```

```
{'orange', 'banana', 'apple'}
3
True
```

- You can use the same operators as strings to see if an element is in the list.

[6]:
```
fruits = {"apple", "banana", "orange"}
veggies = {"carrot", "squash"}

# fruits[1] = "pineapple" ## Illegal!
# print(fruits)

fruits.add("blueberry")
# fruits.insert(1, "tomato")  ## Illegal!
print(fruits)

fruits.remove("apple")
print(fruits)

fruits.discard("lemon")
```

```
print(fruits)

popped = fruits.pop()
print(fruits)
print(popped)

union = fruits.union(veggies)
print(union)

fruits.clear()
print(fruits)
```

```
{'orange', 'blueberry', 'banana', 'apple'}
{'orange', 'blueberry', 'banana'}
{'orange', 'blueberry', 'banana'}
{'blueberry', 'banana'}
orange
{'carrot', 'blueberry', 'banana', 'squash'}
set()
```

- You can update an item a a given index by referencing its postiion and assigning a value
- You can `add()` a value.
- You can `remove()` a given value. An error is thrown if it doesn't exist.
- You can `discard()` a value. No error is thrown if it doesn't exist.
- You can `pop()` the last item off the set. This also returns what it popped. Remember that the last item is arbitrary.
- Several methods exist to do set operations, like `union()` and `intersection()`.
- `clear()` a set to remove all its contents.

There are *many* more set methods available. A reference can be found here.

5 Dictionary

A dictionary is a map between a string key and a value.

[7]:
```
colors = {"orange": "orange", "banana": "yellow", "apple": "red"}

print(colors)
print(colors.values())
print(colors.items())
print(len(colors))

print(colors["banana"])

print("banana" in colors)
print("yellow" in colors)
```

```
{'orange': 'orange', 'banana': 'yellow', 'apple': 'red'}
dict_values(['orange', 'yellow', 'red'])
dict_items([('orange', 'orange'), ('banana', 'yellow'), ('apple', 'red')])
3
yellow
True
False
```

- Use `values()` to get the actual values, or `items()` to get tuples of key-value pairs.
- You can use the `in` keyword to see if a **key** is in the dictionary. To check values, you will have to check the `values()`.

[8]:
```
colors = {"orange": "orange", "banana": "yellow", "apple": "red"}
colors2 = {"tomato": "red"}

colors["blueberry"] = "blue"
print(colors.items())

popped = colors.pop("apple")
print(colors.items())
print(popped)

popped = colors.popitem()
print(colors.items())
print(popped)

colors["more"] = colors2
print(colors.items())

colors.clear()
print(colors.items())
```

```
dict_items([('orange', 'orange'), ('banana', 'yellow'), ('apple', 'red'),
('blueberry', 'blue')])
dict_items([('orange', 'orange'), ('banana', 'yellow'), ('blueberry', 'blue')])
red
dict_items([('orange', 'orange'), ('banana', 'yellow')])
('blueberry', 'blue')
dict_items([('orange', 'orange'), ('banana', 'yellow'), ('more', {'tomato':
'red'})])
dict_items([])
```

- Add new items to the dictionary by assigning a value to the new key.
- `pop()` removes the item with the specified key, and returns its value.
- `popitem()` removes the last added item, and returns a tuple of the key and value.
- You can nest dictionaries by setting a key to have a dictionary as the value.
- `clear()` removes all items from the dictionary.

There are *many* more dictionary methods available. A reference can be found here.