Computational Thinking with Programming

@cse_bennett @csebennett

Tuple Operations

- **Accessing the Tuple**
- **Updating the Tuple**
- **Unpack Tuple**
- Loop Tuple
- Join Tuple
- Tuple Methods

Accessing Tuple

- You can access tuple items by referring to the index number, inside square brackets:

```
>>> a = (“Bennett", “University", “Computer")
>>> print(a[1])
University
```

- **Negative Indexing:** Negative indexing means start from the end. -1 refers to the last item, -2 refers to the second last item etc.

```
>>> print(a[-1]) #Print the last item of the tuple
Computer
```

- **Range (Slicing) of Indexes:** You can specify a range of indexes by specifying where to start and where to end the range. First item has index 0.
- When specifying a range, the return value will be a new tuple with the specified items.

```
>>>tup = ("apple", "banana", "cherry","orange", "kiwi", "melon", "mango")
>>>print(tup[2:5]) # Return the 3rd, 4th, and 5th item, index 2 (included) and index 5 (excluded)
```

Accessing Tuple (Cont..)

- **Range of Indexes:** By leaving out the start value, the range will start at the first item.
- By leaving out the end value, the range will go on to the end of the list:

```
>>>tup = ("apple", "banana", "cherry","orange", "kiwi", "melon", "mango")
>>>print(tup[:4]) #This example returns the items from the beginning to, but NOT included, "kiwi“
>>>print(tup[2:]) #This example returns the items from "cherry" and to the end
```

- **Range of Negative Indexes:** Specify negative indexes if you want to start the search from the end of the tuple

```
>>>tup = ("apple", "banana", "cherry","orange", "kiwi", "melon", "mango")
>>>print(tup[-4:-1]) #This example returns the items from index -4 (included) to index -1 (excluded)
```

- **Check if Item Exists:** To determine if a specified item is present in a tuple use the in keyword:

```
>>>tup = ("apple", "banana", "cherry")
>>>if "apple" in tup:
>>>    print("Yes, 'apple' is in the fruits tuple")
```

Updating Tuple

- Once a tuple is created, you cannot change, add, or remove items once the tuple is created. Tuples are **unchangeable**, or **immutable** as it also is called.
- But you can convert the tuple into a list, change the list, and convert it back into a tuple.
- **Change Tuple Values:**

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)     #Output: ("apple", "kiwi", "cherry")
```

- **Add Items:** Once a tuple is created, you cannot add items to it.
- You can also not add items using **x**.append("orange") # This will raise an error.
- Instead, you can convert it into a list, add your item(s), and convert it back into a tuple.

```
tup = ("apple", "banana", "cherry")
y = list(tup)
y.append("orange")
tup = tuple(y)  #Output: ("apple", “banana", "cherry“, “orange”)
```

Updating Tuple (Cont..)

- **Remove Items:** Tuples are **unchangeable**, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items.
- **Example:** Convert the tuple into a list, remove "apple", and convert it back into a tuple.

```
x = ("apple", "banana", "cherry")
y = list(x)
y.remove("apple")
x = tuple(y) #Output: (“banana", "cherry“)
```

- Although, you can delete the tuple completely. The `del` keyword can delete the tuple completely.

```
x = ("apple", "banana", "cherry")
del x
print(x) #this will raise an error because the tuple no longer exists
```

Unpacking Tuple

- When we create a tuple, we normally assign values to it. This is called "packing" a tuple.

```
fruits = ("apple", "banana", "cherry") #Packing a tuple
```

- But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking“.

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits #Unpacking a tuple
print(green)
print(yellow)
print(red)
```

- **Note:** If the number of variables is less than the number of values, you can add an asterix (*) to the variable name and the values will be assigned to the variable as a list.

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")
(green, yellow, *red) = fruits
print(green)    # apple
print(yellow)   # banana
print(red)      # ['cherry', 'strawberry', 'raspberry']
```

Looping/Iterating Tuple

- You can loop through the tuple items by using a for loop.

```
tup = ("apple", "banana", "cherry")
for x in tup:
      print(x)
```

```
Output:
apple
banana
cherry
```

- You can also loop through the tuple items by referring to their index number. Use the range() and len() functions to create a suitable iterable.

```
tup = ("apple", "banana", "cherry")
for x in range(len(tup)):
      print(x)
```

```
Output:
apple
banana
cherry
```

You can loop through the list items by using a while loop.

```
x = ("apple", "banana", "cherry")
i = 0
while i < len(x):
  print(x[i])
  i = i + 1
```

```
Output:
apple
banana
cherry
```

Join Tuples

- **Join two tuples:** To join two or more tuples you can use the + operator.

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

```
Output:
('a', 'b', 'c', 1, 2, 3)
```

- **Multiply Tuples:** If you want to multiply the content of a tuple a given number of times, you can use the * operator.

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

```
Output:
('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
```

Tuple Methods

- Python has two built-in methods that you can use on tuples.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Thank You