Python Data Structures

Prof. Audrey Mbogho
Fall 2020
USIU-Africa

Python Data Structures

- We will look at Python's in-build data structures, namely:
  - List
  - Tuple
  - Dictionary
  - Set

Creating Lists

- A list is an ordered sequence of items in a pair of square brackets.
- An empty list is created using a pair of square brackets with nothing in between.
- Use square brackets to index into the list.

```
>>> shopping_list = []
>>> shopping_list = ['milk', 'honey', 'onions', 'tomatoes',
'eggs']
>>> shopping_list
['milk', 'honey', 'onions', 'tomatoes', 'eggs']
>>> shopping_list[0]
'milk'
```

List Methods

- A list object has methods for manipulating it.
- For example the append method lets you add an item to the end of the list.
- Type help(list) to see other methods and their usage.

```
>>> fruits = ["apple", "banana", "cherries", "dates"]
>>> fruits.append("mango")
>>> fruits
['apple', 'banana', 'cherries', 'dates', 'mango']
```

Sorting Lists

- You can sort the list in place. This is possible because a list object is mutable.
- In the example, we also reverse the list.

```
>>> numbers = [9, 2, 6, 3, 8, 7]
>>> numbers
[9, 2, 6, 3, 8, 7]
>>> numbers.sort()
>>> numbers
[2, 3, 6, 7, 8, 9]
>>> numbers.reverse()
>>> numbers
[9, 8, 7, 6, 3, 2]
```

Processing List Elements

- The following loop uses the indexing operator to access all the items in the list.

```
>>> for i in range(len(fruits)):
        print(fruits[i])


apple
banana
cherries
dates
mango
```

Looping through a list

- To access each element of a list, it is not necessary to use the indexing operator. A simpler way to do this is demonstrated below:

```
>>> for fruit in fruits:
        print(fruit)


apple
banana
cherries
dates
mango
```

Exercise

- Write a function that performs the above task. The function takes a list as an argument and returns the average of the numbers in the list.

```
>>> def total(lst):
        sum = 0
        for item in lst:
            sum = sum + item
        return sum

>>> numbers = [4, 3, 2, 6]
>>> print(total(numbers))
15
```

Nested Lists

- The elements of a list can be any type of Python object, including other lists.
- This is similar to what is known as a multidimensional array in other languages.

```
>>> shopping = [['oranges', 'apples', 'guavas'], ['spinach', 'cabbage', 'broccoli', 'sukuma']]
>>> shopping[0][2]
'guavas'
>>> shopping[1][1]
'cabbage'
```

List Comprehensions

- A list comprehension consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses.
- This creates a new list resulting from evaluating the expression in the context of the for and if clauses which follow it.

List Comprehensions

- The code on the left shows a list created normally using a loop.
- The code on the right achieves the same result more concisely using a list comprehension.

```
>>> for x in range(10):
        squares.append(x**2)


>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
>>> squares = [x**2 for x in range(10)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List Comprehensions

- The following loop combines the elements of two lists if they are not equal. The listcomp does the same thing more elegantly.

```
>>> for x in [1, 2, 3]:
        for y in [3, 1, 4]:
            if x != y:
                combs.append((x, y))


>>> combs
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

```
>>> [(x, y) for x in [1, 2, 3] for y in [3, 1, 4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

List Comprehensions

- More examples

```
>>> vec = [-4, -2, 0, 2, 4]
>>> # create a new list with the values doubled
>>> [x*2 for x in vec]
[-8, -4, 0, 4, 8]
>>> # filter the list to exclude negative numbers
>>> [x for x in vec if x >= 0]
[0, 2, 4]
>>> # apply a function to all the elements
>>> [abs(x) for x in vec]
[4, 2, 0, 2, 4]
>>> # call a method on each element
>>> freshfruit = ['  banana', '  loganberry ', 'passion fruit  ']
>>> [weapon.strip() for weapon in freshfruit]
['banana', 'loganberry', 'passion fruit']
>>> # create a list of 2-tuples like (number, square)
>>> [(x, x**2) for x in range(6)]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
```

Tuples

- A tuple is like a list except you enclose it in a pair of round brackets (parentheses), and, more important, a tuple immutable.
- So, a tuple does not have any methods that can change it.
- You can access the elements of a tuple in the same way you do a list.

Cycling through a tuple

```
>>> vowels = ('a', 'e', 'i', 'o', 'u')
>>> for vowel in vowels:
        print(vowel, end=" ")


a e i o u

>>> for i in range(5):
        print(vowels[i], end=" ")


a e i o u
>>>
```

Tuples

- An empty tuple is similarly created using an empty pair of round brackets.
- A tuple with a single element is a bit odd. It must be followed by a comma.
- This is because any object enclosed in parentheses evaluates to that item.

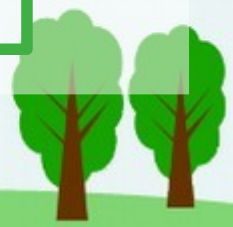
```
>>> shopping_list = ()
>>> shopping_list = ('milk',)
>>> shopping_list[0]
'milk'
```

Dictionaries

- A dictionary stores key-value pairs. To access a value, use square brackets and the key.

```
>>> shopping_list = {'apples': 5, 'mangoes': 2, 'tea': 1, 'carrots':
10}
>>> shopping_list['mangoes']
2
>>> for key in shopping_list:
        print(key, shopping_list[key])


apples 5
mangoes 2
tea 1
carrots 10
```



Sequences

- Lists, tuples and strings are all sequences
- Sequences support the following operations:
  - Membership
  - Indexing
  - Slicing
- Built-in functions like len, min and max apply to all sequences.
- We have discussed membership and indexing. We now look at slicing.

Slicing

- A sequence is numbered from 0 to n-1, where n is the length of the sequence.
- A slice is a portion of the sequence obtained using indexing and a colon.

```
>>> greeting = "Hello"
>>> greeting[1:3]
'el'
>>> greeting[2:]
'llo'
>>> greeting[:3]
'Hel'
```

Slicing

- A sequence also has another numbering from the end of the sequence to the beginning.
- The indices go from -1 to -n.
- These indices can also be used for slicing.

0	1	2	3	4
h	e	l	l	o
-5	-2	-3	-4	-1

```
>>> continents = ('Africa', 'Asia', 'Europe',
'North America', 'South America')
>>> continents[-4:]
('Asia', 'Europe', 'North America', 'South
America')
>>> continents[-4:-2]
('Asia', 'Europe')
```

Tutorials

- Recommended Python Tutorials:

  https://python.swaroopch.com/

  http://anh.cs.luc.edu/python/hands-on/3.1/Hands-onPythonTutorial.pdf