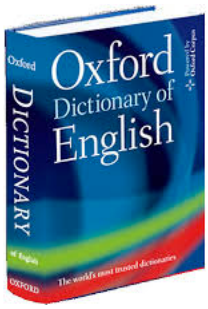
Module 4: Dictionary, Tuples and Methods



1

Dictionaries

- Collection of unordered objects stored/accessed through keys
- The key in a dictionary must me an immutable object
  - number, string, tuple, dictionary (we can have nested dictionaries)
- The value can be any object

```
D_num = {0:["Jake", "Joe"]}
```

```
D_str = {"name": ["Jake", "Joe"]}
```

```
D_tup = {("Jake", 1): [95,91,80]}
```

2

Dictionaries As Loop Target

Iterate over keys of dictionary:

REMEMBER: Dictionaries are not ordered!

```
D = {"Jake": [90,91], "Joe": [100,100], "Charlie": [99,100]}
```

```
for name in D:
    print(name)
```

Equivalent way to iterate over keys of dictionary:

```
D = {"Jake": [90,91], "Joe": [100,100], "Charlie": [99,100]}
```

```
for name in D.keys():
    print(name)
```

More on this later

3

Add Element to Existing Dictionary

```
D = {"Jake": [90,91], "Joe":[100,100], "Charlie": [99,100]}
```

```
D
```

```
{'Charlie': [99, 100], 'Jake': [90, 91], 'Joe': [100, 100]}
```

Name of dictionary we want to add to

```
#Add element for Ellen
D["Ellen"] = [75,80]
```

New key

New value

4

Why Dictionaries

Why Dictionaries?

- Let's us store and access info through something other than a number (index).
  - Let's say I wanted to store people's address somewhere in my code

With a dictionary:

```
Addresses = {"Jake": "67 Gleneden Ave",\
             "Joe": "10501 Streamview Ct." }
```

With a list:

```
Addresses = [["Jake","67 Gleneden Ave"],\
             ["Joe","10501 Streamview Ct."]]
```

5

Tuples

- Tuples are essentially immutable lists.
  - They can be slice and index and used in for loops, but you can't sort them.
- Since they are immutable, they can be keys in a dictionary, as we already saw.

```
a=(1,2)
type(a)
```

```
tuple
```

Notice parentheses instead of bracket!

```
#Concatenation
(1,2) + (3,4)
```

```
(1, 2, 3, 4)
```

```
#Single number
(4,)
```

```
(4,)
```

```
#Indexing works
a=(1,2,3,4)
a[1:3]
```

```
(2, 3)
```

6

Tuples

- Tuples are essentially immutable lists.
  - They can be slice and index and used in for loops, but you can't sort them.
- Since they are immutable, they can be keys in a dictionary, as we already saw.

```
t = (1,2,3,4)
t[1]=0
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-36-998ea75bd6ed> in <module>()
      1 t = (1,2,3,4)
----> 2 t[1]=0

TypeError: 'tuple' object does not support item assignment
```

Can't change an element of tuple because of immutability

7

Python Objects

- Python objects are **dynamically typed** – when we create a variable we don't have to say what type of object it will store.
- Python objects are wither **mutable** (can be changed) or **immutable** (cannot be changed)
- Python objects are **strongly typed** – there are built in type specific methods that help us manipulate objects.

8

String Methods

Replace method : global search and replace.

```
name = "Jaqe"
correct_name = name.replace("q", "k")

name
correct_name
```

```
'Jaqe'
'Jake'
```

name of string

method

9

String Methods

Find method : finds the first location of the given substring (or a -1 if it is not found).

```
sentence = "Hello World."

sentence.find('e')
sentence.find(' ')
```

```
1
5
```

10

String Methods

Split method : splits string into list, delimited by input.

```
line = 'I went to the store'
words = line.split(' ')

words
```

"split string by space"

```
['I', 'went', 'to', 'the', 'store']
```

11

String Methods

Strip method : Deletes input from both sides of string.

```
line = '.I went to the store.'
new_line  = line.strip(".")

new_line
line
```

```
'I went to the store'
'.I went to the store.'
```

12

String Methods

We can stack methods

```
line = '      I went to the store.'
new_line  = line.strip(" .").split(" ")

new_line
line
```

```
['I', 'went', 'to', 'the', 'store']
'     I went to the store.'
```

```
line = '      I went to the store.'
new_line_strip  = line.strip(" .")
final_line = new_line_strip.split(" ")
new_line_strip
final_line
```

```
'I went to the store'
['I', 'went', 'to', 'the', 'store']
```

13

List Methods

Append method : Add element to end of the list.

```
L = [1,2,3]
L+=[4]
L
```

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

```
L = [1,2,3]
L.append(4)
L
```

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

Since lists are mutable the methods change the object itself!

14

List Methods

Sort method : Sorts the elements in the list

```
L = [4,5,1]
L.sort()
L
```

```
[1, 4, 5]
```

You will lose the original ordering of L in this case

```
L = [4,5,1]
sorted_L = sorted(L)
sorted_L
L
```

```
[1, 4, 5]
[4, 5, 1]
```

15

List Methods

Index method : returns the index of first occurrence of the inputted element.

```
L = [4,5,1]
index_five = L.index(5)
index_five
```

```
1
```

If I want to use the index I have to store it....

What happens if the list does not have the inputted element?

16

Dictionary Methods

keys method : returns the keys as an interable.

```
D = {"Jake":1, "Joe":2}
D.keys()
list(D.keys())
```

```
dict_keys(['Jake', 'Joe'])
['Jake', 'Joe']
```

You can wrap in a list to a get indexable object

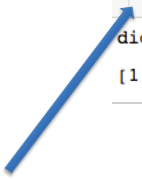
17

Dictionary Methods

values method : returns the values as an interable.

```
D = {"Jake":1, "Joe":2}
D.values()
list(D.values())
```

```
dict_values([1, 2])
[1, 2]
```



You can wrap in a list to a get indexable object

18

Dictionary Methods

get method : another way to access a value through a key

```
D = {"Jake":1, "Joe":2}
#Get value associated with key "Jake
D.get("Jake")
```

```
1
```

Main difference: Try to access key that does exist.

Won't cause code to crash

```
print(D.get("Steve"))
```

```
None
```

19