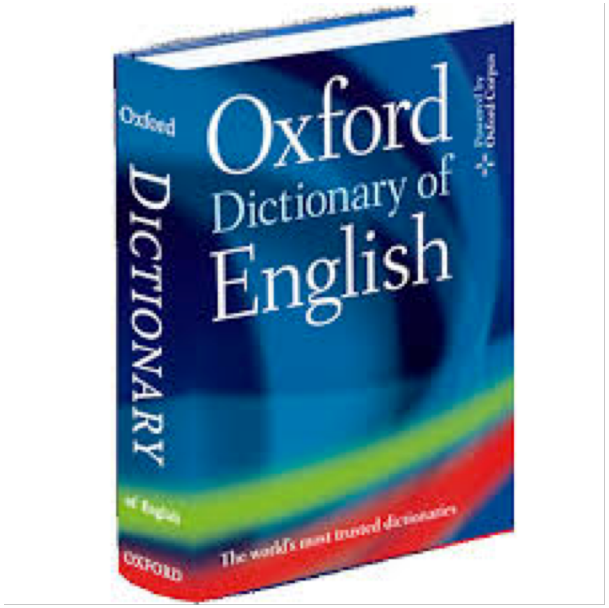
Dictionary/Tuples and Methods



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]}
```

Dictionaries

- 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]}
```

Dictionaries

- 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 and it is access through its key.

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

```
['Jake', 'Joe']
```

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

```
['Jake', 'Joe']
```

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

```
[95, 91, 80]
```

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)
```

```
Jake
Joe
Charlie
```

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

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

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

```
D
```

```
{'Charlie': [99, 100], 'Ellen': [75, 80], 'Jake': [90, 91], 'Joe': [100, 100]}
```

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."]]
```

Why Dictionaries

With a dictionary:

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

To get Joe's address:

```
Addresses["Joe"]
```

Why Dictionaries

With a list:

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

To get Joe's address (imagine there were hundreds of addresses):

Why Dictionaries

With a list:

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

To get Joe's address (imagine there were hundreds of addresses):

```
for address in Addresses:
    name = address[0]
    if name == "Joe":
        joe_address = address[1]
```

A lot more tedious!

Why Dictionaries

Let's say I want to create a dictionary which stores the number of times each word appears in the list:

```
L = ["Jake", "Jake", "Jonny", "Tarik", "Tarik", "Katy"]
```

Why Dictionaries

Let's say I want to create a dictionary which stores the number of times each word appears in the list:

```
L = ["Jake", "Jake", "Jonny", "Tarik", "Tarik", "Katy"]
```

How would you do this with just lists?

Why Dictionaries

Let's say I want to create a dictionary which stores the number of times each word appears in the list:

$$\text{D\_counts} = \{\ \}$$

```
L = ["Jake", "Jake", "Jonny", "Tarik", "Tarik", "Katy"]

D_counts = {}

for name in L:
    if name in D_counts.keys():
        D_counts[name]+=1
    else:
        D_counts[name] =1
```

Why Dictionaries

Let's say I want to create a dictionary which stores the number of times each word appears in the list:

D_counts = { }
name = "Jake"

```
L = ["Jake", "Jake", "Jonny", "Tarik", "Tarik", "Katy"]

D_counts = {}

for name in L:
    if name in D_counts.keys():
        D_counts[name]+=1
    else:
        D_counts[name] =1
```

Why Dictionaries

Let's say I want to create a dictionary which stores the number of times each word appears in the list:

D_counts = { }
name = "Jake"

```
L = ["Jake", "Jake", "Jonny", "Tarik", "Tarik", "Katy"]

D_counts = {}

for name in L:
    if name in D_counts.keys():
        D_counts[name]+=1
    else:
        D_counts[name] =1
```

Why Dictionaries

Let's say I want to create a dictionary which stores the number of times each word appears in the list:

$$\text{D\_counts} = \{\text{“Jake”} : 1\}$$
$$\text{name} = \text{“Jake”}$$

```
L = ["Jake", "Jake", "Jonny", "Tarik", "Tarik", "Katy"]

D_counts = {}

for name in L:
    if name in D_counts.keys():
        D_counts[name]+=1
    else:
        D_counts[name] =1
```

Why Dictionaries

Let's say I want to create a dictionary which stores the number of times each word appears in the list:

$$\text{D\_counts} = \{\text{“Jake”} : 1\}$$
$$\text{name} = \text{“Jake”}$$

```
L = ["Jake", "Jake", "Jonny", "Tarik", "Tarik", "Katy"]

D_counts = {}

for name in L:
    if name in D_counts.keys():
        D_counts[name]+=1
    else:
        D_counts[name] =1
```

Why Dictionaries

Let's say I want to create a dictionary which stores the number of times each word appears in the list:

$$\text{D\_counts} = \{\text{“Jake”} : 1\}$$
$$\text{name} = \text{“Jake”}$$

```
L = ["Jake", "Jake", "Jonny", "Tarik", "Tarik", "Katy"]

D_counts = {}

for name in L:
    if name in D_counts.keys():
        D_counts[name]+=1
    else:
        D_counts[name] =1
```

Why Dictionaries

Let's say I want to create a dictionary which stores the number of times each word appears in the list:

$$\text{D\_counts} = \{\text{“Jake”} : 2\}$$
$$\text{name} = \text{“Jake”}$$

```
L = ["Jake", "Jake", "Jonny", "Tarik", "Tarik", "Katy"]

D_counts = {}

for name in L:
    if name in D_counts.keys():
        D_counts[name]+=1
    else:
        D_counts[name] =1
```

Why Dictionaries

Let's say I want to create a dictionary which stores the number of times each word appears in the list:

$$\text{D\_counts} = \{\text{“Jake”} : 2\}$$
$$\text{name} = \text{“Jonny”}$$

```
L = ["Jake", "Jake", "Jonny", "Tarik", "Tarik", "Katy"]

D_counts = {}

for name in L:
    if name in D_counts.keys():
        D_counts[name]+=1
    else:
        D_counts[name] =1
```

Why Dictionaries

Let's say I want to create a dictionary which stores the number of times each word appears in the list:

$$\text{D\_counts} = \{\text{“Jake”} : 2\}$$
$$\text{name} = \text{“Jonny”}$$

```
L = ["Jake", "Jake", "Jonny", "Tarik", "Tarik", "Katy"]

D_counts = {}

for name in L:
    if name in D_counts.keys():
        D_counts[name]+=1
    else:
        D_counts[name] =1
```

Why Dictionaries

Let's say I want to create a dictionary which stores the number of times each word appears in the list:

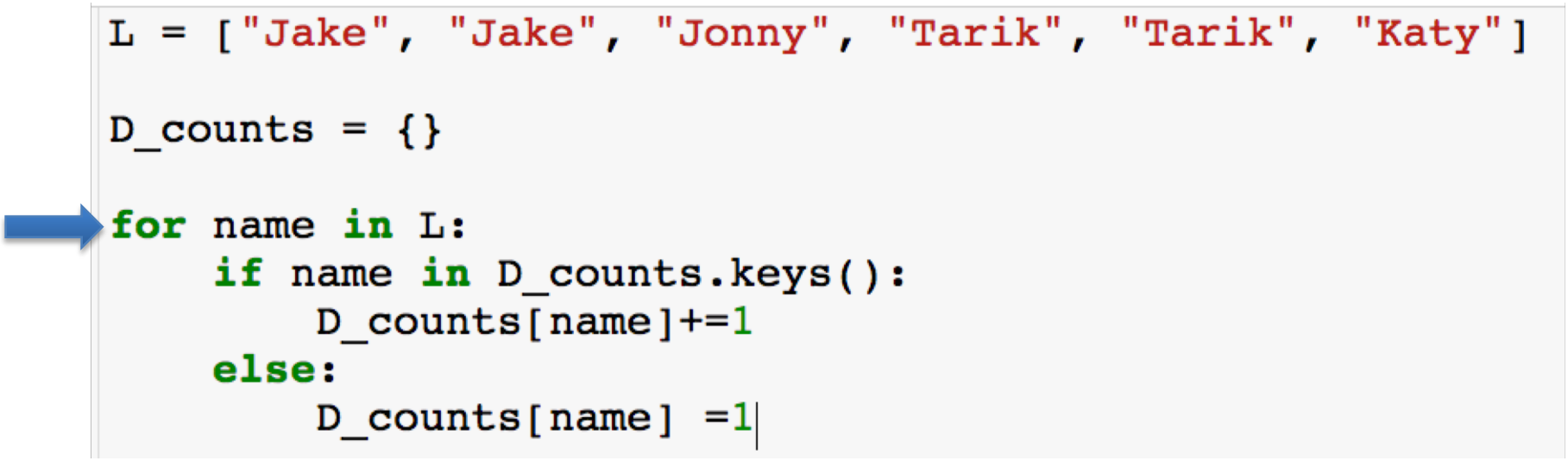
$$\text{D\_counts} = \{\text{“Jake”} : 2, \text{“Jonny”} : 1\}$$
$$\text{name} = \text{“Jonny”}$$

```
L = ["Jake", "Jake", "Jonny", "Tarik", "Tarik", "Katy"]

D_counts = {}

for name in L:
    if name in D_counts.keys():
        D_counts[name]+=1
    else:
        D_counts[name] =1
```

Why Dictionaries

Let's say I want to create a dictionary which stores the number of times each word appears in the list:

$$\text{D\_counts} = \{\text{“Jake”} : 2, \text{“Jonny”} : 1\}$$
$$\text{name} = \text{“Tarik”}$$



```
L = ["Jake", "Jake", "Jonny", "Tarik", "Tarik", "Katy"]

D_counts = {}

for name in L:
    if name in D_counts.keys():
        D_counts[name]+=1
    else:
        D_counts[name] =1
```

And so on….

Why Dictionaries

Let’s say I want to compute each person’s score

```
scores_list = [["Jake", 10], ["Joe",5], ["Jake", 12],\
               ["Ellen", 20], ["Ellen", 2], ["Joe", 14]]
```

Why Dictionaries

Let's say I want to compute each person's score

```
scores_list = [["Jake", 10], ["Joe",5], ["Jake", 12],\
               ["Ellen", 20], ["Ellen", 2], ["Joe", 14]]
```

How would you do this with just lists?

Why Dictionaries

Let's say I want to compute each person's score

```
scores_list = [["Jake", 10], ["Joe",5], ["Jake", 12],\
               ["Ellen", 20], ["Ellen", 2], ["Joe", 14]]
```

With dictionary:

```
scores_D = {}

for pair in scores_list:

    name = pair[0]
    score = pair[1]

    if name in scores_D:
        scores_D[name]+=score
    else:
        scores_D[name] = score

scores_D
```

```
{'Ellen': 22, 'Jake': 22, 'Joe': 19}
```

Why Dictionaries

Let's say I want to compute each person's score

```
scores_list = [["Jake", 10], ["Joe",5], ["Jake", 12],\
               ["Ellen", 20], ["Ellen", 2], ["Joe", 14]]
```

With dictionary:

Unnecessary, but makes code more readable

```
scores_D = {}

for pair in scores_list:

    name = pair[0]
    score = pair[1]

    if name in scores_D:
        scores_D[name]+=score
    else:
        scores_D[name] = score

scores_D
```

```
{'Ellen': 22, 'Jake': 22, 'Joe': 19}
```

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.

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
```

```
#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)
```

Notice parentheses instead of bracket!

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

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.

```
#Tubles can be object in for loop
a =(1,2,3,4)
total=0
for i in a:
    total+=i
total
```

```
10
```

Can be used in for loops!

Python Objects

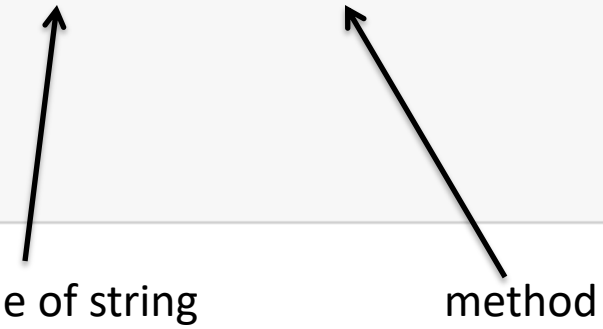
- Python objects are **dynamically typed** – when we create a variable we don't have to say what type of object it will store.
- Python objects are wither **mutable** (can be changed) or **immutable** (cannot be changed)
- Python objects are **strongly typed** – there are built in type specific methods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

String Methods

Replace method : global search and replace.

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

name
correct_name
```

```
'Jaqe'

'Jake'
```



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
```

String Methods

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

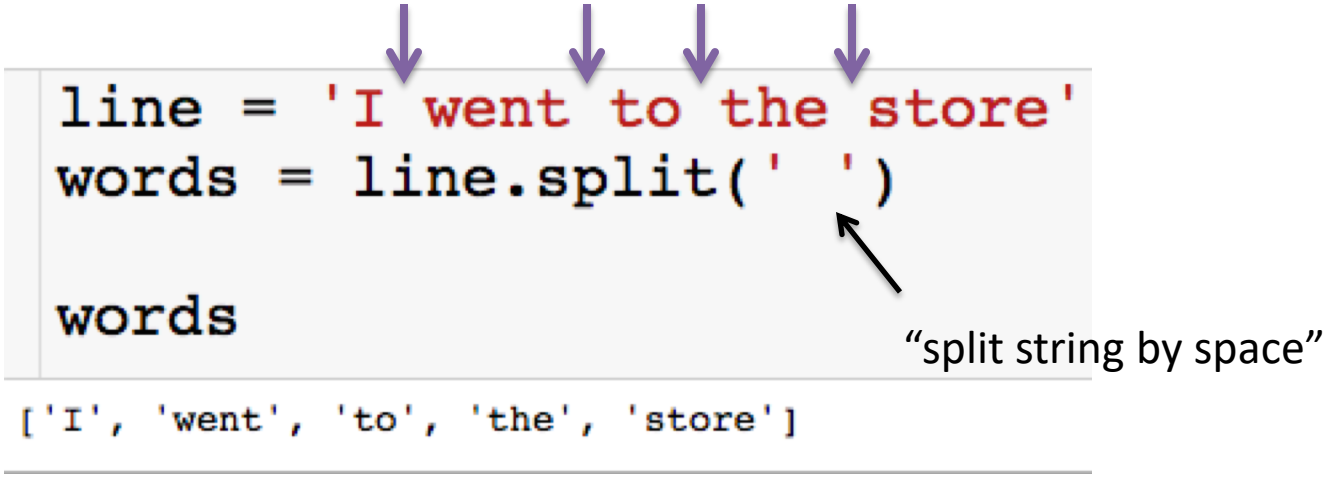
```
line = 'I went to the store'
words = line.split(' ')

words
```

"split string by space"

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

String Methods

Split method : splits string into list, delimited by input



String Methods

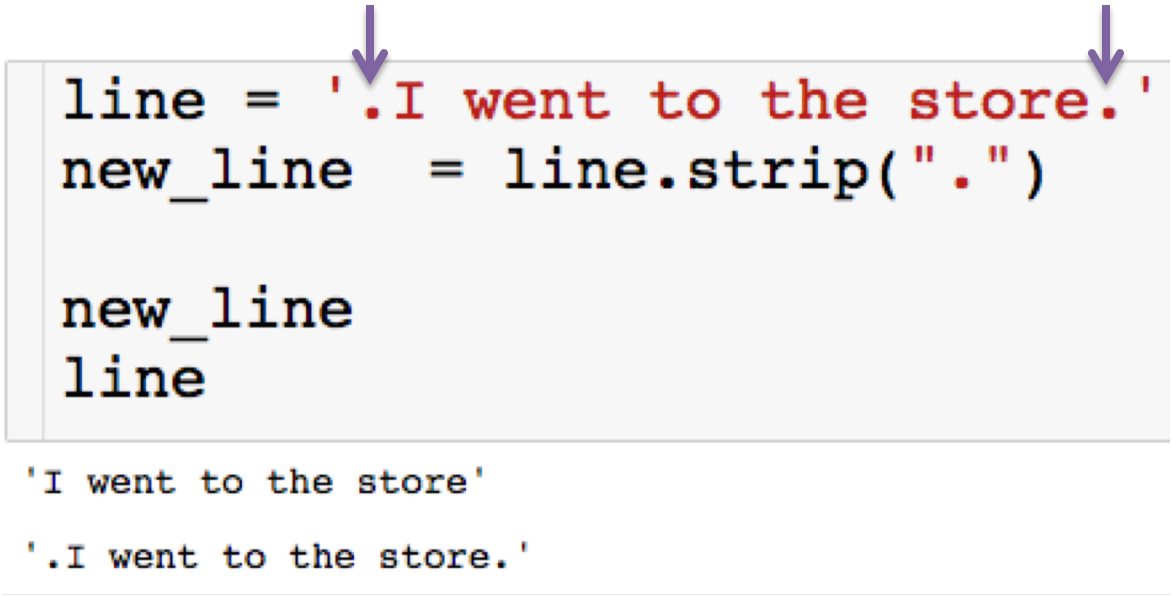
Strip method : Deletes input from both sides of string.

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

new_line
line
```

```
'I went to the store'

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

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.'
```

“Get rid of periods on the left and right of the string”

String Methods

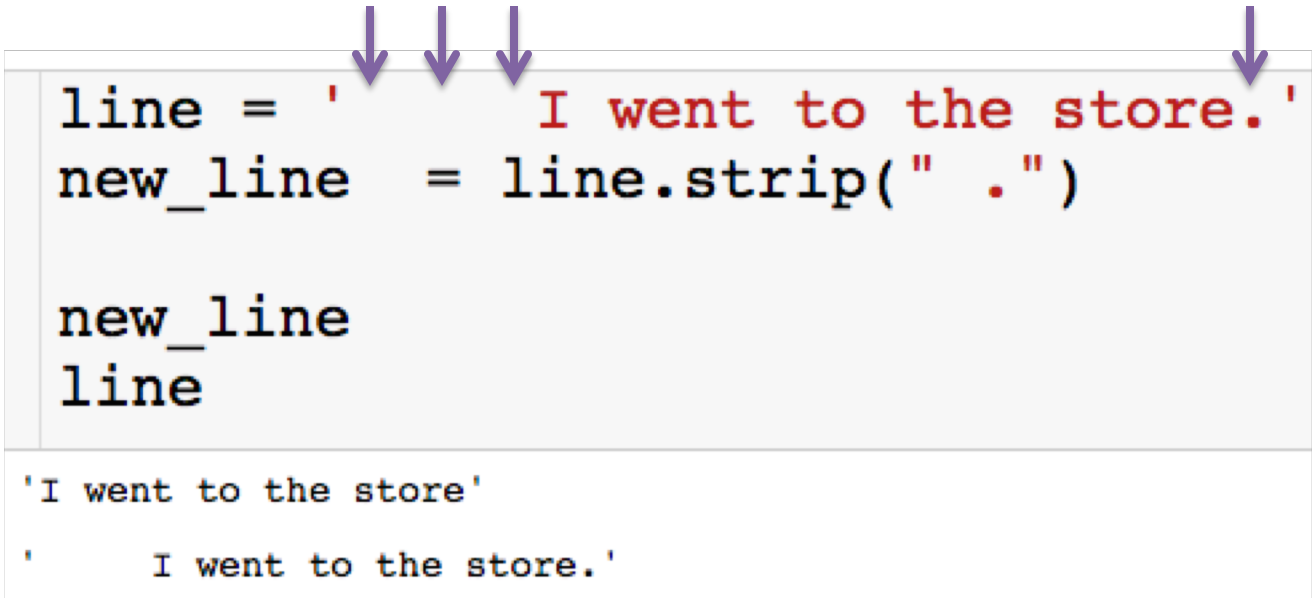
Strip method : Deletes input from both sides of string.

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

new_line
line
```

```
'I went to the store'

'     I went to the store.'
```

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.'
```

"Get rid of spaces and periods from the left and right of the string."

String Methods

We can stack string 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.'
```

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']
```

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]
```

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
```

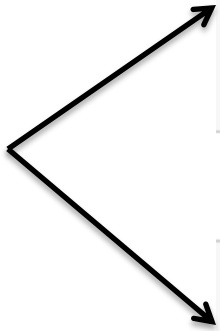
```
[1, 2, 3, 4]
```

Since lists are mutable the methods change the object itself!

List Methods

Append method : Add element to end of the list.

Do these really do the same thing?



```
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!

List Methods

Sort method : Sorts the elements in the list

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

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

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

Built in Sort Function

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

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

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

Built in Sort Function

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

```
[1, 4, 5]

[4, 5, 1]
```

Unlike the sort() method, running the sorted function on L does not change L, It just returns a sorted version of L that we can store!

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....

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?

List Methods

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

```
L = [4,5,1]
index_five = L.index(10)
index_five
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-23-7d6b0bfccc08> in <module>()
      1 L = [4,5,1]
----> 2 index_five = L.index(10)
      3 index_five

ValueError: 10 is not in list
```

We get an error!

Question

How do I find all indices of a given element?

```
L = [4,5,1,5,12,3,4,1,5,7,8]
```

Question

How do I find all indices of a given element?

```
L = [4,5,1,5,12,3,4,1,5,7,8]
```

Answer 1: For loop

```
find_fives =[]
for i in range(len(L)):
    if L[i]==5:
        find_fives.append(i)

find_fives
```

```
[1, 3, 8]
```

Question

How do I find all indices of a given element?

```
L = [4,5,1,5,12,3,4,1,5,7,8]
```

Answer 2: List comprehension (more on this later…)

```
find_fives = [i for i in range(len(L)) if L[i] == 5]
find_fives
```

```
[1, 3, 8]
```

List Methods

We cannot stack list methods

```
L = [4,5,1,5,12,3,4,1,5,7]

L.sort().index(5)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-28-11ce6a038eb9> in <module>()
      1 L = [4,5,1,5,12,3,4,1,5,7]
      2
----> 3 L.sort().index(5)

AttributeError: 'NoneType' object has no attribute 'index'
```

This is because list methods do not return anything, they change the list inplace.

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

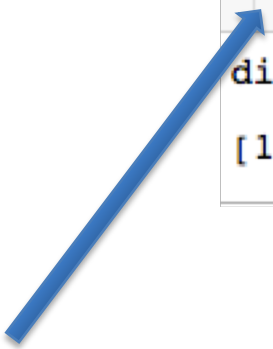
Dictionary Methods

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

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]
```

Can put values on object in four loop:

```
D = {"Jake":1, "Joe":2}
for num in D.values():
    print(num)
```

```
1
2
```

Dictionary Methods

We can check whether a key or value exists as follows:

```
D = {"Jake": "Charlie", "Joe":"Griffey"}
D
```

```
{'Jake': 'Charlie', 'Joe': 'Griffey'}
```

```
"Jake" in D.keys()
```

```
True
```

```
"Clemy" in D.values()
```

```
False
```

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
```

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.

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.

```
D["Steve"]
```

```
---------------------------
KeyError
<ipython-input-9-f4219aed6c
----> 1 D["Steve"]

KeyError: 'Steve'
```

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
```