12-13 Strings, Lists and Dictionaries

We already know string data type in Python. Let’s see what more can we do with strings.

12-13 Strings, Lists and Dictionaries

We already know string data type in Python. Let's see what more can we do with strings.

Recall that string is immutable type.
Therefore,

```
url = "https://www.logees.com/"
url[1] = 'p'
```

will not work!

```
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    url[1]='p'
TypeError: 'str' object does not support item assignment
```

12 Strings

String slicing

```
>>> url = "https://www.logees.com/"
>>> url[8:]
'www.logees.com/'
```

12 Strings

String slicing

```
>>> url = "https://www.logees.com/"
>>> url[8:]
'www.logees.com/'
```

```
url[8:]
```

start *stop (next after)*

12 Strings

String slicing

```
>>> url = "https://www.logees.com/"
>>> url[8:]
'www.logees.com/'
```

```
url[8:]
```

start *stop*
(next after)

```
>>> url = "https://www.logees.com/"
>>> url[12:18]
'logees'
```

12 Strings

String slicing

This operation creates a new string!

Hence, if you'd like a copy of the string:

```
>>> a = "BCC"
>>> b = a[:]
```

b is a brand new string "BCC"

```
>>> b
"BCC"
```

12 Strings

Finding and replacing

```
>>> phrase = "There are two options here: you get two apples or you get three oranges"
>>> phrase.replace("two","three")
'There are three options here: you get three apples or you get three oranges'
```

12 Strings

Finding and replacing

```
>>> phrase = "There are two options here: you get two apples or you get three oranges"
>>> phrase.replace("two","three")
'There are three options here: you get three apples or you get three oranges'
```

`replace(old,new)`
- returns a copy of the string with all occurrences of the substring `old` replaced by the string `new`.
The `old` and `new` arguments may be string variables or string literals.

12 Strings

Finding and replacing

```
>>> phrase = "There are two options here: you get two apples or you get three oranges"
>>> phrase.replace("two","three")
'There are three options here: you get three apples or you get three oranges'
```

`replace(old,new,count)`

- same as the call with two parameters, except only replaces the first `count` occurrences of `old`

12 Strings

Finding and counting occurrences

```
>>> phrase = "The robocalls come when you are driving and they bother you at night. It doesn't matter if you're in bed or in a meeting. Here's the worst news: There is really no way for you to stop them. I know this because for the past few years, I have been bombarded with robocalls alerting me that I owe student loans, or that I won a free vacation, or that I am being audited by the Internal Revenue Service."
>>> phrase.find("spam")
-1
```

12 Strings

Finding and counting occurrences

```
>>> phrase = "The robocalls come when you are driving and they bother you at night. It doesn't matter if you're in bed or in a meeting. Here's the worst news: There is really no way for you to stop them. I know this because for the past few years, I have been bombarded with robocalls alerting me that I owe student loans, or that I won a free vacation, or that I am being audited by the Internal Revenue Service."
>>> phrase.find("if")
88
```

12 Strings

Finding and counting occurrences

```
>>> phrase = "The robocalls come when you are driving and they bother you at night. It doesn't matter if you're in bed or in a meeting. Here's the worst news: There is really no way for you to stop them. I know this because for the past few years, I have been bombarded with robocalls alerting me that I owe student loans, or that I won a free vacation, or that I am being audited by the Internal Revenue Service."
>>> phrase.count("a")
25
```

12 Strings

Comparing strings

```
>>> fruit1 = "apple"
>>> fruit2 = "pear"
>>> fruit3 = "apple"

>>> fruit1 > fruit2
False

>>> fruit1 == fruit3
True
```

12 Strings

check a string value

isalnum() - returns True if all characters in the string are lowercase or uppercase letters, or the numbers 0-9.

isdigit() - returns True if all characters are the numbers 0-9.

islower() - returns True if all characters are lowercase letters.

isupper() - return True if all cased characters are uppercase letters.

isspace() - return True if all characters are whitespace.

startswith(x) - return True if the string starts with x.

endswith(x) - return True if the string ends with x.

12 Strings

Split and join

```
>>> birthdate = "12/23/1985"
>>> birthdate.split("/")
['12', '23', '1985']
```

12 Strings

Split and join

```
>>> birthdate = "12/23/1985"
>>> birthdate.split("/")
['12', '23', '1985']

>>> names = "Jane Janet Jasmine Janine"
>>> names.split()
['Jane', 'Janet', 'Jasmine', 'Janine']
```

12 Strings

Split and join

```
>>> birthdate = "12/23/1985"
>>> birthdate.split("/")
['12', '23', '1985']

>>> names = "Jane Janet Jasmine Janine"
>>> names.split()
['Jane', 'Janet', 'Jasmine', 'Janine']
```

`split(<parameter>)`
- the string method that splits a string up into a list of tokens, using separator `<parameter>` (by default, it is a *whitespace*)

12 Strings

Split and join

```
>>> names = ["Anna","Maria"]
>>> "-".join(names)
'Anna-Maria'

>>> data = ["Jane","yahoo.com"]
>>> "@".join(data)
'Jane@yahoo.com'
```

The `join()` method performs the inverse operation of split() by joining a list of strings together to create a single string.

12 Strings

In-class activity 1-3

13 Lists and Dictionaries

We already know that a Python *list* is a container, in which the elements are ordered from left to the right.

```
>>> myList = [ “Hello”, 123, “my”, 1.2, 1978]
positions:          0      1      2      3      4

>>> len(myList)
5
```

5 elements in the *list* `myList`

Lists and Dictionaries

We already know that a Python *list* is a container, in which the elements are ordered from left to the right.

```
>>> myList = [ “Hello”, 123, “my”, 1.2, 1978]
positions:          0      1      2      3      4

>>> len(myList)
5
```

5 elements in the *list* `myList`

Let’s explore more about Python lists!

Lists and Dictionaries

```
list() function:

>>> myList = list(“Hello”)
>>> myList
[‘H’, ’e’, ’l’, ’l’, ’o’]
```

Lists and Dictionaries

`list()` function:

```
>>> myList = list("Hello")
>>> myList
['H', 'e', 'l', 'l', 'o']
```

Python lists are *mutable*, i.e. can be modified:

```
>>> m = [1,2,7,3,4]
>>> m[2] = 10
>>> print(m)
[1,2,10,3,4]
>>> m.append(12)
>>> m
[1,2,10,3,4,12]
```

Lists and Dictionaries

More operators for work with Python lists:

List slicing:

```
>>> m = [1,2,7,3,4,7,0]
>>> k = m[2:6]
```

Lists and Dictionaries

More operators for work with Python lists:

List slicing:

```
          0 1  2  3 4  5  6
>>> m = [1,2,7,3,4,7,0]
>>> k = m[2:6]
>>> print(k)
[7,3,4,7]
```

Lists and Dictionaries

More operators for work with Python lists:

List slicing:

```
>>> m = [1,2,7,3,4,7,0]
>>> k = m[2:6]
>>> print(k)
[7,3,4,7]
```

Lists concatenation:

```
>>> m = [1, 2, 7]
>>> k = [4.5, 10, 1.2, 5.6]
>>> n = m + k
>>> print(n)
[1, 2, 7, 4.5, 10, 1.2, 5.6]
```

Lists and Dictionaries

More operators for work with Python lists:

Deleting an element at a position in the list:

```
>>> m = [1,2,7,3,4,7,0]
>>> del m[5]
>>> print(m)
[1,2,7,3,4,0]
```

Lists and Dictionaries

More operators for work with Python lists:

Deleting an element at a position in the list:

```
>>> m = [1,2,7,3,4,7,0]
>>> del m[5]
>>> print(m)
[1,2,7,3,4,0]
```

Deleting an element at a position in the list:

```
>>> m = [1,2,7,3,4,7,0]
>>> m.pop(2)
>>> print(m)
[1,2,3,4,7,0]
```

- similar to `del m[2]`

Lists and Dictionaries

More operators for work with Python lists:

Deleting an element at from the end of the list:

```
>>> m = [1,2,7,3,4,7,0]
>>> m.pop()
>>> print(m)
[1,2,7,3,4,7]
```

Lists and Dictionaries

More operators for work with Python lists:

Deleting an element at from the end of the list:

```
>>> m = [1,2,7,3,4,7,0]
>>> m.pop()
>>> print(m)
[1,2,7,3,4,7]
```

Deleting the first occurrence of a value from the list:

```
>>> m = [1,2,7,3,4,7,0]
>>> m.remove(7)
>>> print(m)
[1,2,3,4,7,0]
```

Lists and Dictionaries

Adding elements to the list:

Appending an element at the end of the list:

```
>>> m = [1,2,7,3,4,7,0]
>>> m.append(9)
>>> print(m)
[1,2,7,3,4,7,0,9]
```

Lists and Dictionaries

Adding elements to the list:

Appending an element at the end of the list:

```
>>> m = [1,2,7,3,4,7,0]
>>> m.append(9)
>>> print(m)
[1,2,7,3,4,7,0,9]
```

Inserting an element before a position in the list:

```
>>> m = [1,2,7,3,4,7,0]
>>> m.insert(3,17) # position, value
>>> print(m)
[1,2,7,17,3,4,7,0]
```

Lists and Dictionaries

Adding elements to the list:

Extending a list:

```
>>> m = [1,2,7,3,4,7,0]
>>> n = [‘a’,’b’,20]
>>> m.extend(n)
>>> print(m)
[1,2,7,3,4,7,0,‘a’,’b’,20]
```

Lists and Dictionaries

Modifying elements of the list:

sorting the list:

```
>>> m = [1,12,7,13,4,7,0]
>>> m.sort()
>>> print(m)
[0,1,4,7,7,12,13]
```

Lists and Dictionaries

More operations on the list:

Finding the position of the element in the list (first occurrence):

```
>>> m = [1,12,7,13,4,7,0]
>>> m.index(7)
2
```

Lists and Dictionaries

More operations on the list:

Finding the position of the element in the list (first occurrence):

```
>>> m = [1,12,7,13,4,7,0]
>>> m.index(7)
2
```

Counting the number of occurrences of a value in the list:

```
>>> m = [7,1,2,7,3,4,7,0]
>>> m.count(7)
3
```

Lists and Dictionaries

Iterating over the elements of the list:

Assume we are given a list `Numbers` and we want to print *the squares of all the elements in the list*.

Lists and Dictionaries

Iterating over the elements of the list:

Assume we are given a list `Numbers` and we want to print *the squares of all the elements in the list*.

```
def squares(Numbers):
   for item in Numbers:
      print(item*item)

b = [9, 1.1, 0, -4, 2]
squares(b)
```

Lists and Dictionaries

Iterating over the elements of the list:

Assume we are given a list `Numbers` and we want to print *the squares of all the elements in the list*.

```
def squares(Numbers):
    for item in Numbers:
        print(item*item)

b = [9, 1.1, 0, -4, 2]
squares(b)
```

Output:

```
81
1.44
0
16
4
```

Lists and Dictionaries

Iterating over the elements of the list:

Assume we are given a list `Names` and we want to print *the sorted list of names with enumeration*.

Lists and Dictionaries

Iterating over the elements of the list:

Assume we are given a list `Names` and we want to print *the sorted list of names with enumeration*.

```
def sortNames(Names):
   Names.sort()

   for i,name in enumerate(Names):
      print(i+1,name)

a = ['Mary','Tom','Paul','Liza','Andrew',
'Jack','Destiny','Dan']
sortNames(a)
```

Lists and Dictionaries

Iterating over the elements of the list:

Assume we are given a list `Names` and we want to print *the sorted list of names with enumeration*.

```
def sortNames(Names):
   Names.sort()

   for i,name in enumerate(Names):
      print(i+1,name)

a = ['Mary','Tom','Paul','Liza','
'Jack','Destiny','Dan']
sortNames(a)
```

Output:

```
1 Andrew
2 Dan
3 Destiny
4 Jack
5 Liza
6 Mary
7 Paul
8 Tom
```

Lists and Dictionaries

Iterating over the elements of the list:

Assume we are given a list of decimal values and we would like to round them off to the nearest integer value.

Lists and Dictionaries

Iterating over the elements of the list:

Assume we are given a list of decimal values and we would like to round them off to the nearest integer value.

```
def roundOff(values):
   for i in range(len(values)):
      values[i] = round(values[i])

a = [1.34, 6.56, 8.76, 9.1]
rouondOff(a)
print(a)
```

Lists and Dictionaries

Iterating over the elements of the list:

Assume we are given a list of decimal values and we would like to round them off to the nearest integer value.

```
def roundOff(values):
    for i in range(len(values)):
        values[i] = round(values[i])

a = [1.34, 6.56, 8.76, 9.1]
rouondOff(a)
print(a)
```

Output:

[1, 7, 9, 9]

Lists and Dictionaries

In-class activity 4-7