Junior Knights Python I - Week 9

Strings & Lists Review

Agenda

String Review

String Methods

Lists

String Review

- Strings in Python are surrounded by either single quotation marks ‘ ’ or double quotation marks “ ”
  - ‘hello’ is the same as “hello”
- You can use quotes inside a string, as long as they don’t match the quotes surrounding the string:
  - print(“It’s alright”)
  - print(“He is called ‘Johnny’”)
    print(‘He is called “Johnny”’)
- You can assign a multiline string to a variable by using three double quotes or single quotes:
  - a = “””This is a
    multiline string.”””
- Concatenate Strings:
  - a = "Hello"
    b = "World"
    print(a + " " + b)

```
a = "Hello"
b = "World"
print(a + b)
```

String Review

- Strings are Arrays, which means square brackets can be used to access elements of the string
- Python does not have a character data type, a single character is a string with a length 1
- Getting length of a string: **`len()`**
- Determines if a certain phrase or character is present in a string using keyword **`in`** and returns a boolean

Example 1:

```
a = "Hello, World!"
print(a[1])                    #
Output?
```

Example 2:

```
a = "Hello!"
print(len(a))                  #
Output?
```

Example 3:

```
a = "Hello, World!"
print("Hello" in a)          # Output?
```

Slicing Strings

- Slicing returns a range of characters. Stop indexes are not inclusive.

Technique	Syntax	Example
Slicing from a start to a stop	`a[start:stop]`	`s = "Hello, World!"` `print(s[7:12])`
Slicing from the start:	`a[:stop]`	`s = "Hello, World!"` `print(s[:5])`
Slicing to the end:	`a[start:]`	`s = "Hello, World!"` `print(s[7:])`
Negative indexing starts slice from end	`a[-start:-end]`	`s = "Hello, World!"` `print(s[-6:-1])`

Formatting Strings

- Ways to print a string:
  - ```
    age = 36
    print("I am " + age)
    ```
  - ```
    age = 36
    print(f"I am {age}")
    ```
- Modifying with decimals
  - ```
    price = 50
    print(f"The price is ${price:.2f}")
    ```

\\	Backslash
\n	New Line
\t	Tab
\b	Backspace
    ```

String Methods

- Python has built-in string functions that don't require imports.

Function	Purpose
`.upper()`	Returns string in upper case
`.lower()`	Returns string in lower case
`.strip()`	Removes whitespace from beginning or end
`.replace()`	Replaces a string with another string
`.split()`	Splits string into substrings if it finds instances of the separator
`.find()` / `.index()`	Searches for value and returns where it was found

Quick Practice

1. Convert the following string to both uppercase and lowercase and print the results:
   a. `text = "Python is fun!"`
2. Remove the whitespace from the following string:
   a. `text = "   Hello, World!   "`
3. Replace "apples" in the following string with "bananas":
   a. `text = "I love apples!"`
4. Split the sentence into a list of words and print the result:
   a. `text = "Python is my favorite language"`
5. Find and print the index of the word "fox":
   a. `text = "The quick brown fox jumps over the lazy dog"`

Live Coding

List Review

- Lists are used to store multiple items in a single variable. Lists are indexed, allow duplicate values, and are changeable. They can contain any data type and can mix data types

- Ways to create a list
  - Standard:
    - `my_list = [1, 2, 3, 4, 5]`
  - Using list() constructor. *Useful for converting from tuples, sets, or strings:*
    - `my_list = list((1, 2, 3, 4, 5))`
    - `my_list = list(range(5))`
    - `my_list = list("hello")`
  - Using list comprehensions – *creating a list from an expression:*
    - `my_list = [x*2 for x in range(5)]`
  - Using .split() on a string
    - `my_list = "Junior Knights".split()`
  - Using * for repetition
    - `my_list = [0] * 5`

List Review

- Lists are used to store multiple items in a single variable. Lists are indexed, allow duplicate values, and are changeable. They can contain any data type and can mix data types

- Ways to create a list
  - Standard:
    - `my_list = [1, 2, 3, 4, 5]  # [1, 2, 3, 4, 5]`
  - Using list() constructor. *Useful for converting from tuples, sets, or strings:*
    - `my_list = list((1, 2, 3, 4, 5))  # [1, 2, 3, 4, 5]`
    - `my_list = list(range(5))  # [0, 1, 2, 3, 4]`
    - `my_list = list("hello")  # ['h', 'e', 'l', 'l', 'o']`
  - Using list comprehensions – *creating a list from an expression:*
    - `my_list = [x*2 for x in range(5)]  # [0, 2, 4, 6, 8]`
  - Using .split() on a string
    - `my_list = "Junior Knights".split()  # ['Junior',`

Accessing List Items

- Lists are accessed by using indexes, where the first element is [0]
  - `my_list = [1, 2, 3, 4, 5]`
  - `print(my_list[1])`
- Accessing lists with negative indexing starts from the end:
  - `print(my_list[-1])`
  - `print(my_list[-2])`
- Slicing to get a range of indexes, stop is not inclusive
  - `print(my_list[1:3])`
- Checking if an item exists
  - `print(0 in my_list)`

Accessing List Items

- Lists are accessed by using indexes, where the first element is [0]
  - `my_list = [1, 2, 3, 4, 5]`
    `print(my_list[1])` `# Output = 2`
- Accessing lists with negative indexing starts from the end:
  - `print(my_list[-1])` `# Output = 5`
  - `print(my_list[-2])` `# Output = 4`
- Slicing to get a range of indexes, stop is not inclusive
  - `print(my_list[1:3])` `# Output = [2, 3]`
- Checking if an item exists
  - `print(0 in my_list)` `# Output = False`

Changing List Items

- Refer to the index number to change a specific item:
  - `my_list = [1, 2, 3, 4, 5]`
    `my_list[1] = 7`
- Change a range of item values using slicing. You can insert more or less items than you replace and the remaining items will move accordingly:
  - `my_list[1:3] = [71, 72]`
  - `my_list[1:3] = [45]`
- Insert a new list item, without replacing the existing values, using insert() method
  - `my_list.insert(2, 99)`
- Add list items by using append()
  - `my_list.append(17)`
- Append elements from another list to current list using extend()
  - `other_list = [4, 3, 2, 1]`
    `my_list.extend(other_list)`

Changing & Adding List Items

- Refer to the index number to change a specific item:
  - `my_list = [1, 2, 3, 4, 5]`
    `my_list[1] = 7` `# [1, 7, 3, 4, 5]`
- Change a range of item values using slicing. You can insert more or less items than you replace and the remaining items will move accordingly:
  - `my_list[1:3] = [71, 72]` `# [1, 71, 72, 4, 5]`
  - `my_list[1:3] = [45]` `# [1, 45, 4, 5]`
- Insert a new list item, without replacing the existing values, using insert() method
  - `my_list.insert(2, 99)` `# [1, 2, 99, 3, 4, 5]`
- Add list items by using append()
  - `my_list.append(17)` `# [1, 2, 3, 4, 5, 17]`
- Append elements from another list to current list using extend()
  - `other_list = [4, 3, 2, 1]`
    `my_list.extend(other_list)` `# [1, 2, 3, 4, 5, 4, 3, 2, 1]`

Deleting List Items

- Use the remove() method to remove a specified item
  - `my_list = [1, 2, 2, 3, 4, 5]`
  - `my_list.remove(4)`
- If there are duplicates, remove() removes the first occurrence
  - `my_list.remove(2)`
- Remove specific index with pop() method
  - `my_list.pop(0)`
- del keyword can remove a specific index or an entire list
  - `del my_list[0]`
  - `del my_list`
- clear() empties the list without deleting it
  - `my_list.clear()`

Nested Lists

- Nested lists are used to store multiple groups of things in different categories.
  - If a list is a box where you can store things, nested lists are smaller boxes within that box. Each smaller box can have its own elements, which can even be yet another box.
- Defining a nested list:
  - ```
    nested_list = [
    [1, 2, 3],                  # First smaller box
    [4, 5, 6],                  # Second smaller box
    ['a', 'b', 'c']   # Third smaller box
    ]
    ```
- To access a nested list, you need to select the bigger box (outer list) then the smaller box (inner list)
  - `nested_list[0]` `# [1, 2, 3]`
  - `nested_list[1][2]` `# 6`
    ```

List Methods

- Python has a set of built-in methods that you can use on lists.

Method	Purpose
`copy()`	Returns a copy of the list
`reverse()`	Reverses the order of the list
`sort()`	Sorts the list

Live Coding