Name: Zach Caldwell
Class: CS 112-01
Date: March 7, 2017

| **Cues:** | **Notes:** |
|---|---|
| Method definition | 1. A *method* is like a function inside of an object, and it is given implicit access to the object instance on which it is called, meaning that methods generally do not return a new object of the same type. |
| | 2. List methods: |
| index() list method | a. index(): Gives the list position (the index) of a given item if it is in the list; throws a ValueError otherwise. Example:<br>`>>> spam = ['hello', 'hi']`<br>`>>> spam.index('hi')`<br>`1` |
| append() list method | b. append(): Adds a value to the end of the list. Example (assume that the same variables are used throughout this document):<br>`>>> spam.append('howdy')`<br>`>>> spam`<br>`['hello', 'hi', 'howdy']` |
| insert() list method | c. insert(): Inserts a value at a given position in the list. Example:<br>`>>> spam.insert(2, 'hey')`<br>`>>> spam`<br>`['hello', 'hi', 'hey', 'howdy']` |
| remove() list method | d. remove(): If the given item exists in the list, this removes it; a ValueError is thrown otherwise. Example:<br>`>>> spam.remove('hi')`<br>`>>> spam`<br>`['hello', 'hey', 'howdy']` |
| sort() list method | e. sort(): If all the items in the list can be reasonably compared, then this sorts the list values by ASCII / numerical order; otherwise a TypeError is thrown. Can also be passed a `reverse` argument to sort in reverse. Example:<br>`>>> spam.sort(reverse=True)`<br>`>>> spam`<br>`['howdy', 'hey', 'hello']` |
| | 3. More on strings: |
| Single quotes in strings | a. Strings can be encapsulated within single (‘) or double (“) quote marks; however, a string can only *contain* single quote marks if the string is given in double quotes. Example:<br>`>>> cat_desc = "This is not Bob's cat."` |
| Escape sequences | b. Unusual or otherwise impossible characters or codes can also be added to a string by using a *character escape*, which is a backslash (\) followed by the character. Example:<br>`>>> cat_desc = 'This is not Bob\'s cat.'` |
| ‘Raw’ strings | c. Strings can be created as *raw* (or literal) by placing an ‘r’ in front of the string, which makes Python treat any escape sequences as literal string characters. Example:<br>`>>> print(r'That\'s Bob\'s cat')`<br>`That\'s Bob\'s cat` |
| Multi-line strings | d. Multi-line strings can be created by using either triple quotes (‘’’) or the newline escape character (\n). Example: |

Name: Zach Caldwell
Class: CS 112-01
Date: March 7, 2017

| Cues: | Notes: |
|---|---|
| | **>>> print('''Dear Alice,\nI’ve found your cat.**<br>**... Sincerely, Bob''')**<br>**Dear Alice,**<br>**I’ve found your cat.**<br>**Sincerely, Bob** |
| Strings as lists | e. Similar to lists, strings can also be indexed, sliced, and used with the ‘in’ and ‘not in’ keywords, with the same syntax. |
| String case | f. Strings can be converted to all lowercase with lower(), uppercase with upper(), and can have their ‘case-ness’ tested with islower() and isupper(). Example:<br>**>>> cat_desc.upper()**<br>**‘THIS IS NOT BOB’S CAT.’**<br>**>>> cat_desc.islower()**<br>**False** |
| startswith(), endswith() | g. The beginning and ending contents of strings can be tested with startswith() and endswith(), respectively. Example:<br>**>>> cat_desc.endswith(‘CAT.’)**<br>**True** |
| join() | h. Strings can be *joined* (concatenated) into a new string by calling join() on the desired separator value and passing in a list of target strings. Example:<br>**>>> ‘-’.join([‘Totally’, ‘not’, ‘my’, ‘cat.’])**<br>**'Totally-not-my-cat.'** |
| split() | i. Strings can be *split* (the opposite of joined) into a list of component sub-strings by using the split() method. Example:<br>**>>> cat_desc.split(' ')**<br>**['This', 'is', 'not', "Bob's", 'cat.']** |

**Summary/Reflection:**