LISTS

COMPUTER SCIENCE MENTORS CS 88

February 15th to 20th

1 Lists

Introduction In Python, *lists* are ordered collections of whatever values we want, be it numbers, strings, functions, or even other lists! Each value stored inside a list is called an *element*. We can create lists by using square braces.

```
>>> foods = ['apple', 'oranges', 'banana', 'milk', 'cookies']
>>> print(foods)
['apple', 'oranges', 'banana', 'milk', 'cookies']
```

Accessing elements Lists are zero-indexed: to access the first element, we must access the element at index 0; to access the ith element, we must index at $i - 1$. We can also index with negative numbers. This begins indexing at the end of the list, so the index -1 is equivalent to the index `len(list) - 1` and index -2 is the same as `len(list) - 2`.
Examples:

```
>>> foods[0]
'apple'
>>> foods[2]
'banana'
>>> foods[-3]
'banana'
```

Sequences also have a notion of length, the number of items stored in the sequence. In Python, we can check how long a sequence is with the `len` built-in function. We can also check if an item exists within a list with the `in` statement.

```
>>> poke_team = ['Meowth', 'Mewtwo']
>>> len(poke_team)
2
>>> 'Meowth' in poke_team
True
>>> 'Pikachu' in poke_team
False
```

2 List Comprehension

A **list comprehension** is a compact way to create a list whose elements are the results of applying a fixed expression to elements in another sequence: `[<map exp> for <name> in <iter exp> if <filter exp>]`

It might be helpful to note that we can rewrite a list comprehension as an equivalent for statement. See the example to the right.

Let's break down an example:

```
[x * x - 3 for x in [1, 2, 3, 4, 5] if x % 2 == 1]
```

In this list comprehension, we are creating a new list after performing a series of operations to our initial sequence `[1, 2, 3, 4, 5]`. We only keep the elements that satisfy the filter expression `x % 2 == 1` (`1`, `3`, and `5`). For each retained element, we apply the map expression `x*x - 3` before adding it to the new list that we are creating, resulting in the output `[-2, 6, 22]`.

Note: The `if` clause in a list comprehension is optional.

3 WWPD

1. What would Python display?

```
>>> a = [1, 5, 4, [2, 3], 3]
>>> print(a[0], a[-1])

>>> len(a)

>>> 2 in a

>>> 4 in a

>>> a[3][0]



>>> print(print("Welcome to"), print("CS 88"))
```

4 Code Writing Questions

2. Write a function that takes in a list and prints the elements in the list at indices that are divisible by 3.

```
def every_third(lst):
    """
    >>> lst1 = [1, 4, 7, 9, 6, 3, 2, 10, 5]
    >>> every_third(lst1)
    1
    9
    2
    >>> lst2 = [5, 3, 1, 7]
    >>> every_third(lst2)
    5
    7
    >>> lst3 = [4, 7]
    >>> every_third(lst3)
    4
    """
```

3. Write a function that returns the sum of all even numbers from 2 to n.

   (Hint: If n = 10, return 2 + 4 + 6 + 8 + 10)

   Assume n is always greater than or equal to 2.

```
def sum_even_to(n):
    """
    >>> sum_even_to(6) # 2 + 4 + 6
    12
    >>> sum_even_to(10) # 2 + 4 + 6 + 8 + 10
    30
    >>> sum_even_to(11) # Still 2 + 4 + 6 + 8 + 10
    30
    """
```

4. Write a function that takes in a list of numbers and returns a list containing only the even numbers from the given list. Use a list comprehension.

```
def only_even(lst):
    """
    >>> lst1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    >>> only_even(lst1)
    [2, 4, 6, 8]
    >>> lst2 = [5, 3, 1, 7]
    >>> only_even(lst2)
    []
    >>> lst3 = [4, 7, 10]
    >>> only_even(lst3)
    [4, 10]
    """
```

5 Optional Challenging problem!

5. Write a function that returns the longest string in a list of strings. You can assume the list has at least one string. If there are multiple strings with the same length, it does not matter which one the function returns.

```
def longest_string(lst):
    """
    >>> food = ["pie", "burgers", "mashed potatoes", "fries"]
    >>> longest_string(food)
    'mashed potatoes'
    >>> colors = ["green", "red", "purple", "turquoise"]
    >>> longest_string(colors)
    'turquoise'
    """
```

6. Draw the environment diagram that results from running the following code.

```
bless, up = 3, 5
another = [1, 2, 3, 4]
one = another[1:]

another[bless] = up
another.append(one.remove(2))
another[another[0]] = one
one[another[0]] = another[1]
one = one + [another.pop(3)]
another[1] = one[1][1][0]
one.append([one.pop(1)])
```