CS051A
INTRO TO COMPUTER SCIENCE WITH TOPICS IN AI

6: Sequences

Alexandra Papoutsaki
she/her/hers
Lectures

Zilong Ye
he/him/his
Labs
Lecture 6: Sequences

- Lists
- Sequences
- Tuples
A program that contains a set of functions for reading in scores and calculating various statistics on them.

```python
# scores-list.py
# A set of functions for reading in scores and calculating
# various statistics from the input scores.

def get_scores():
    """
    Reads user input of numerical scores as floats into a list
    and returns the list
    :return: None
    """
```
scores-list.py main - What does it do?

- First, it prompts the user to enter a list of scores one at a time
  - Uses a while loop that keeps asking the user for a new score. What is the exit condition?
    - Checks to see if the line is empty: `while line != ""`
- Then, calculate various statistics based on what was entered. How are we calculating these statistics?
  - Average?
    - We could keep track of the sum and the total number of scores entered and divide them at the end.
  - Max (min)?
    - Keep track of the largest (smallest) score seen so far. Each time a new one is entered, see if it's larger (smaller). If so, update the largest (smallest).
  - Median?
    - The challenge with median is that we can't calculate it until we have all of the scores. We need to sort them and then find the middle score.
  - Why can't we do this using int/float variables?
    - We don't know how many scores are going to be entered. Even if we did, if we had 100 students in the class, we'd need 100 variables!
Lists

- **List**: a data structure.
  - **Data structure**: a way of storing and organizing data.

- Lists allow us to store multiple values using only a single variable to refer to them!

- Creating lists: provide elements separated by comma and enclosed in square brackets.

- Lists are a type and represent a value, just like `float`, `int`, `bool` and `str`. We can assign them to variables, print them, etc.

```python
>>> [7, 4, 3, 6, 1, 2]
[7, 4, 3, 6, 1, 2]
>>> 10
10
>>> [10]
[10]
>>> my_list = [7, 4, 3, 6, 1, 2]
>>> my_list
[7, 4, 3, 6, 1, 2]
>>> type(my_list)
<class 'list'>
```
Accessing Lists

- `[]`: creates an empty list.
- We can access a particular value in the list by using the `[]` to "index" into the list.
  - Indexing starts at 0!
- Be careful of index out of range errors!
  - We can only index from 0...length-1.
- Negative indexing counts back from the end of the list.

```python
>>> my_list = [7, 4, 3, 6, 1, 2]
>>> my_list[3]
6
>>> my_list[0]
7
>>> my_list[20]
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/...
    exec(code, self.locals)
  File "<input>", line 1, in <module>
IndexError: list index out of range
>>> my_list[-1]
2
>>> type(my_list[3])
<class 'int'>
```
Storing other things in lists

- A list is a contiguous set of spaces in memory.
  
  ```
  >>> ['this', 'is', 'a', 'list', 'of', 'strings']
  >>> list_of_strings = ['this', 'is', 'a', 'list', 'of', 'strings']
  >>> list_of_strings[0]
  'this'
  >>> [1, 5.0, 'my string']
  >>> mixed_list = [1, 5.0, 'my string']
  >>> type(mixed_list[0])
  <class 'int'>
  >>> type(mixed_list[1])
  <class 'float'>
  >>> type(mixed_list[2])
  <class 'str'>
  ```

- We can store anything in each of these spaces.

- In general, it's a good idea to have lists be homogeneous, i.e. be of the same type.
Slicing

- Sometimes, we want more than just one item from the list (this is called *slicing*).

- We can specify a range in the square brackets, [], using the colon (:)
  
  - `list[start:end]` will return a new list with the elements from `start` index through `end-1`.
  
  - `list[start:]` will return a new list with the elements from `start` to the end of the list.
  
  - `list[:end]` will return a new list with the elements from 0 through `end-1`.

```python
>>> list_of_numbers = [32, 4, -1, 15, -20]
>>> list_of_numbers[0:3]
[32, 4, -1]
>>> list_of_numbers[1:4]
[4, -1, 15]
>>> list_of_numbers[1:]
[4, -1, 15, -20]
>>> list_of_numbers[:2]
[32, 4]
>>> list_of_numbers[:]
[32, 4, -1, 15, -20]
>>> list_of_numbers[1:1]
[]
>>> list_of_numbers[-3:-1]
[-1, 15]
```
Looping over lists

- We can use the for loop to iterate over each item in the list.
- This is often called a "foreach" loop, i.e. for each item in the list, do an iteration of the loop.

```python
>>> list_of_numbers = [32, 4, -1, 15, -20]
>>> for value in list_of_numbers:
...    print(value)
...32
...4
...-1
...15
...-20
```
Practice time

- Write a function called `sum` that returns the sum of all the values in a list of numbers.

```python
>>> def sum(numbers):
...     total = 0
...     for val in numbers:
...         total += val
...     return total
...
>>> sum([13, -2, 47, 9, -5])
62
```
Calculating the average of a list - the inelegant way

```python
def inelegant_average(scores):
    """
    Calculates the average of the values in list scores in an inelegant way
    :param scores: (list) a list of numbers that correspond to scores
    :return: (float) the average of the values in scores
    """

    sum = 0.0
    count = 0

    for score in scores:
        sum += score
        count += 1

    return sum / count
```
Calculating the average of a list - the elegant way

```python
def average(scores):
    """
    Calculates the average of the values in list scores in an elegant way
    :param scores: (list) a list of numbers that correspond to scores
    :return: (float) the average of the values in scores
    """
    return sum(scores) / len(scores)
```
Built-in functions over lists

- Length of list
  ```python
  >>> list_of_numbers = [32, 4, -1, 15, -20]
  >>> len(list_of_numbers)
  5
  ```

- Max of list
  ```python
  >>> list_of_numbers = [32, 4, -1, 15, -20]
  >>> max(list_of_numbers)
  32
  ```

- Min of list
  ```python
  >>> list_of_numbers = [32, 4, -1, 15, -20]
  >>> min(list_of_numbers)
  -20
  ```

- Sum of list
  ```python
  >>> list_of_numbers = [32, 4, -1, 15, -20]
  >>> sum(list_of_numbers)
  30
  ```
List methods

- Lists are objects therefore have methods.
  - **Object**: a software bundle that consists of properties and behavior. Behavior is controlled by **methods**.
  - We call a method of an object using the **dot operator**.
  - **Syntax**: `myList.someMethod(argument)`
  - [https://docs.python.org/3/tutorial/datastructures.html](https://docs.python.org/3/tutorial/datastructures.html)
  - `help([])`
  - `help(list)`
append

- Adds a value at the end of a list.

```python
>>> list_of_numbers = [32, 4, -1, 15, -20]
>>> list_of_numbers.append(47)
>>> list_of_numbers
[32, 4, -1, 15, -20, 47]
```

- Notice that append does not return a new list, it just modifies the existing list!
pop

- Removes a value from the end of a list and returns it.
  ```python
  >>> list_of_numbers = [32, 4, -1, 15, -20, 47]
  >>> list_of_numbers.pop()
  47
  >>> list_of_numbers
  [32, 4, -1, 15, -20]
  ```

- Notice that `pop` both modifies the list and returns the last value. If you want to use this value, you need to store it.
  ```python
  >>> popped = list_of_numbers.pop()
  >>> popped
  -20
  >>> list_of_numbers
  [32, 4, -1, 15]
  ```

- `pop` also has another version where you can specify the index.
  ```python
  >>> list_of_numbers.pop(1)
  4
  >>> list_of_numbers
  [32, -1, 15]
  ```
**insert**

- Inserts a value at a specific index.

  ```python
  >>> list_of_numbers
  [32, -1, 15]
  >>> list_of_numbers.insert(2, 100)
  >>> list_of_numbers
  [32, -1, 100, 15]
  ```

- Notice that `insert` does not return a new list but modifies the underlying one.
sort

- Sorts a list in ascending order.

```python
>>> list_of_numbers
[32, -1, 100, 15]
>>> list_of_numbers.sort()
>>> list_of_numbers
[-1, 15, 32, 100]

>>> list_of_strings
['this', 'is', 'a', 'list', 'of', 'strings']
>>> list_of_strings.sort()
>>> list_of_strings
['a', 'is', 'list', 'of', 'strings', 'this']
```

- Again, `sort` does not return a new list but modifies the underlying one.
There is a function called `get_scores`. It gets the scores and returns them as a list.

- starts with an empty list,
- uses `append` to add them on to the end of the list,
- returns the list when the loop finishes.

**median function**

- sorts the values
  - notice again that sort does NOT return a value, but sorts the list that it is called on.
- returns the middle entry
Lists are mutable

- We can change (or mutate) the values in a list.
- Notice that many of the methods that we call on lists change the list itself.
- We can mutate lists with methods, but we can also change particular indices.

```python
>>> list_of_numbers
[-1, 15, 32, 100]
>>> list_of_strings[2] = 100
>>> list_of_numbers
[-1, 15, 32, 100]
```
Lecture 6: Sequences

- Lists
- Sequences
- Tuples
Sequences

- Lists are part of a general category of data structures called **sequences**.
- Sequences represent a... sequence of things.
- **All** sequences support a number of shared behavior.
  - The ability to index using `[]`.
  - The ability to slice using `[:]`.
  - A number of built-in functions: `len`, `max`, `min`.
  - The ability to iterate over them with a for loop.
- We've actually seen one other sequence. Strings!
Strings as sequences

- We can do all sorts of sequence-like things to strings!
- Strings, however, are immutable! We cannot mutate them.

```python
>>> fruit = "banana"
>>> fruit[4]
'n'
>>> fruit[2:5]
'nan'
>>> len(fruit)
6
>>> for letter in fruit:
    ...     print(letter)
    ...
  b
  a
  n
  a
  n
  a
```

```
>>> fruit[4] = "c"
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/code.py", line 90, in runcode
    exec(code, self.locals)
  File "<input>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```
What does the list-to-string function do?

- Creates a list from a string:
  - Takes as input a list. A list of almost any type, as long as we can call `str()` on.
  - Concatenates all the items in the list into a single string.
  - `result` starts out as the empty string.
  - It iterates through each item in the list and concatenates them on to the `result`.
  - Returns the entire `result` list minus the last element (which is " ")
Alternate way of iterating over lists

```python
>>> for letter in fruit:
...     print(letter)
...
banana
banana
banana

>>> for i in range(len(fruit)):
...     print(fruit[i])
...
banana
banana
banana
```
Write a function called `multiply_lists` that takes two lists of numbers and creates a new list with the values pairwise multiplied. E.g.,

```python
def multiply_lists(list1, list2):
    """
    Creates a new list that is the result of the multiplication of two equal
    :param list1: (list) the first list of numbers
    :param list2: (list) the second list of numbers
    :return: (list) a list where each index corresponds to the multiplication in list1 and list2
    """
    result = []
    if len(list1) != len(list2):
        print("Error: lists are not of equal length!")
    else:
        for i in range(len(list1)):
            result.append(list1[i] * list2[i])
    return result

>>> list1 = [1, 2, 1, 2]
>>> list2 = [1, 2, 3, 4]
>>> multiply_lists(list1, list2)
[1, 4, 3, 8]
```
Lecture 6: Sequences

- Lists
- Sequences
- Tuples
Tuples

- **Tuple**: an immutable list. Type of sequence.

- Tuples can be created using parentheses (instead of []).

```python
>>> my_tuple = (1, 2, 3, 4)
>>> my_tuple
(1, 2, 3, 4)

>>> another_tuple = ("a", "b", "c", "d")
>>> another_tuple
(\'a\', \'b\', \'c\', \'d\')
```

- Notice that when they print out, they also show using parentheses.
Tuples as immutable sequences

```python
>>> my_tuple[0]
1
>>> my_tuple[3]
4
>>> for val in my_tuple:
...     print(val)
...
1
2
3
4
>>> my_tuple[1:3]
(2, 3)
>>> my_tuple[0] = 1
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/code", line 1, in <module>
    TypeError: 'tuple' object does not support item assignment
```
Unpacking tuples

- If we know how many items are in a tuple, we can "unpack" it into individual variables.

```python
>>> my_tuple = (1, 2, 3)
>>> my_tuple
(1, 2, 3)
>>> (x, y, z) = my_tuple
>>> x
1
>>> y
2
>>> z
3
>>> (x, y, z) = (10, 11, 12)
>>> x
10
>>> y
11
>>> z
12
>>> x, y, z = "apple", "banana", "pineapple"
>>> x
'apple'
>>> y
'banana'
>>> z
'pineapple'
```
_movies.py_

- Tuples are useful for representing data with fixed entries.
- Look at the `print_movies` function in _movies.py_.
  - It iterates over the list, just like any other list.
  - `movie_pair` is a tuple (each entry in the list is a tuple). We unpack the tuple to get at the two values in the tuple.
    - We also could have written `movie_pair[0]` and `movie_pair[1]` (see `print_movies2`), though unpacking is much cleaner.
  - Once we have the two values, we can print them out
    - `	` is a special character that represents a tab (like `\n`, which represents the end of line character)
- Look at the `print_movies3` function.
  - We can unpack the two values of the tuple *in* the for loop. Any of the variants is fine for this class!
get_movie_score function

- What does the get_movie_score function do?
  - Takes two parameters, a movie database and a movie title.
  - It iterates through the movie database and tries to find the matching title.
    - If it finds it, it returns the score.
    - If it doesn't find it, it will iterate through all of the movie entries, finish the for loop and return -1.0
Practice time

Write a function called `my_max` that takes a list of positive numbers and returns the largest one.

```python
>>> def my_max(numbers):
...     max = -1
...     for num in numbers:
...         if num > max:
...             max = num
...     return max
```

Key idea: have a variable that keeps track of the largest number seen so far. At each iteration, compare the current number to `max`, if it's bigger, update the `max` value.

Why initialize it to -1? We need to initialize it to something that is smaller than any of the values. We could also have done something like `max = numbers[0]` (assuming that the input would have at least one value).
get_highest_rated_movie function

- What does the get_highest_rated_movie function do?
  - Very similar idea to my_max function.
    - We’re finding the largest score.
    - We also keep track of the movie with the highest score so that we can return that at the end.
Practice time

- Write a function called `get_movies_above_threshold` that takes as input a movie database and a critic score threshold and returns all of the movies above that threshold.

```python
def get_movies_above_threshold(movie_db, threshold):
    
    """
    Given a database and a threshold critic score, it returns a list of movies with scores above the threshold.
    :param movie_db: (list) a list of tuples that correspond to movies (str) and scores (float)
    :param threshold: (num) the threshold critic score to filter movies by.
    :return: (list) a list of movie titles that have critic scores higher than the threshold
    """
    movies_above = []

    for (movie, score) in movie_db:
        if score >= threshold:
            movies_above.append(movie)

    return movies_above
```
Resources

- Textbook: Chapters 9 and 10
- scores-list.py
- more-lists.py
- movies.py

Practice Problems

- Practice 4 (solution)

Homework

- Assignment 3