Lists

CS 1111
Introduction to Programming
Spring 2019

[The Coder’s Apprentice, §12]
Overview: Lists

• List = ordered sequence of values

• **Mutable** data type

• Because of the ordering, an element in a list can be referred by its **index**.

• Indices start at **zero**
## Ordered Collection

<table>
<thead>
<tr>
<th>Collection</th>
<th>What it can hold</th>
<th>Syntax to create</th>
<th>Access element</th>
<th>Mutable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>characters</td>
<td>“…..”</td>
<td>[index]</td>
<td>Immutable</td>
</tr>
<tr>
<td>range</td>
<td>int</td>
<td>range(start, stop, step) Note: start, stop, step must be int</td>
<td>[index]</td>
<td>Immutable</td>
</tr>
<tr>
<td>list</td>
<td>anything, any type</td>
<td>[ e1, e2, ... ]</td>
<td>[index]</td>
<td>Mutable</td>
</tr>
</tbody>
</table>
Creating Lists

```python
animals = ['cow', 'dog', 'horse']  # create a new list
print(animals)

animals1 = []  # create an empty list
print(animals)

animals1 = ['cow', 'horse']
animals2 = ['dog']
animals3 = animals1 + animals2  # concatenate lists
print(animals3)
```
in

```python
list = [5, 7, 9, 11, 15]
print(7 in list)
print(3 in list)
print(3 not in list)
```

**in** is a keyword and can be used to check if the element is in the list or string before trying to get its index.
Accessing Items in Lists

animals = ['cow', 'dog', 'horse']  # create a new list

print(animals[2])  # access a particular item
animals[2] = 'duck'  # update a particular item
print(animals[0])  # indices start from zero
print(animals[-1])  # negative numbers start from the end of the list

print('The ' + animals[0] + ' and the ' + animals[2] + ' sleep in the barn.')
Length of Lists

animals = ['dog', 'cat', 'bird']
counter = 0
while counter < len(animals):
    print(animals[counter])
    counter = counter + 1
print(animals)

`len()` returns the length of a list (i.e., the number of items in a list)
# Adding Items to Lists

```python
animals = ['cow', 'dog', 'horse']  # create a new list
animals.append('deer')  # add item to a list
print(animals[2])  # access a particular item
animals[2] = 'duck'  # update a particular item
print(animals[0])  # indices start from zero
print(animals[2])  # negative numbers start from the end of the list
print('The ' + animals[0] + ' and the ' + animals[2] + ' sleep in the barn.')
animals.insert(2, 'pig')
print(animals)
```

- `append(element)` adds an element to the end of a list, return None
- `insert(index, element)` adds an element to a particular position of a list, return None
Removing Items from Lists

animals = ['cow', 'dog', 'horse', 'sheep', 'pig']
print(animals)
del animals[3]  # remove by index
print(animals)
print(animals.pop())  # remove the last element, and return its value
print(animals.pop(1))  # remove by index, and return its value
print(animals)
animals.remove('horse')  # remove by item / element
print(animals)

**del** deletes an element at a particular position

**pop()** removes the last element from the list and return its value

**pop(index)** removes an element at a particular position and return its value; raise IndexError if an index is out of range

**remove(element)** removes a particular element, return None; raise ValueError if an element does not exist
animals = ['cow', 'dog', 'horse', 'sheep', 'pig']
animals.sort()
print('sorted animals =', animals)

# another way to print (notice a space after "=")
print('sorted animals = ' + str(animals))

animals.reverse()
print('reversed animals = ', animals)

**sort()** rearranges the items of a list (in ascending order), return None

**reverse()** reverses the order of the items in the list, return None
index(element)

small = [1, 2, 3]
print(small.index(2))

index(element) returns an index of an element, raise ValueError if an element is not found

Note: You’ll need to check if the element is in the list before trying to get its index
**list**(*collection*)

```python
letters = 'ABCDEFG'
print(list(letters))
```

**list**(*collection*) converts a given collection into a list, return a list
Slicing and Returning Part of a List with [ : ]

```python
list = [5, 7, 9, 11, 15]
print(list)
print(list[1:4])
print(list[1:])
print(list[:4])
print(list[::])
print(list[::1])
print(type(list[2:4]))
```
How are Lists Represented in Memory?

- Primitive types are stored directly
- Complex types (such as lists) are stored indirectly
- Trace through code

```python
num = 5
grades = [97, 86, 91, num, 88]
num = 33
big = [23, grades, num, 7]
print(big)
grades[1] = 87
grades.append(6)
big[2] = grades
print(big)
```

- What happens when we assign a variable to a list? (in memory)
  - Only the memory address is assigned; the list is not copied
Tracing through Code with Lists

• **Rule 1**
  • Variables and items on the heap are stored in separate locations.

• **Rule 2**
  • A primitive type is stored directly with its variable.
  • A complex type has its variable store a **memory address**.
    • A memory address refers to a location on the heap where the actual data is stored.

• **Rule 3**
  • Every assignment begins by either creating a variable space (and heap location, if necessary), or emptying out the existing contents of a variable space (**but not the heap**!).
  • Copying either a value or memory address from one box into the other.
  • A variable or memory location must only store either numbers/booleans, or a memory address, **never** the name of a variable.
Tracing through Code with Lists

num = 5
grades = [97, 86, 91, num, 88]
num = 33
big = [23, grades, num, 7]
print(big)
grades[1] = 87
grades.append(6)
big[2] = grades
print(big)
print(grades)

Variables
num
grades
big

Heap
Output

A100 of 1
A200 of 2

Variables
num
grades
big

Heap
Output

A100
A200

[23, [97, 86, 91, 5, 88], 33, 7]
[23, [97, 87, 91, 5, 88, 6], [97, 87, 91, 5, 88, 6], 7]
[97, 87, 91, 5, 88, 6]
Two Dimensional List (List of Lists)

```python
list1 = [5, 7, 9, 11, 15]
list_of_lists = [['cow', 'horse'], [list1], [4, 5, 6]]

print (list_of_lists[0])    # access a particular list
print (list_of_lists[0][1]) # access a particular item

print(len(list_of_lists))
print(len(list_of_lists[1]))
```

Summary

• Must know (based on exam2 topic list, as of 03/04/2019)
  • element in lst
  • lst.append(value)
  • lst.insert(index, value)
  • lst.remove(value)
  • lst.pop(index)
  • lst.sort()
  • lst.index(element)
  • lst[start:end]
  • list(collection)