Collective Data Structures

lists, sets, dictionaries, tuples
Lists

Lists are sequences used to store collections of data.

Lists are ordered and their elements can be indexed by numbers.

The elements of a list are mutable. You can change individual elements directly.

You can create sub-lists using the slice operators.

Creating lists

To create a list, place all the elements inside a square bracket [ ] and separated them by commas.

The elements can be heterogeneous: integer, float, string, list etc.

```python
1  # empty list
2  empty_list = []
3  |
4  # list of numbers
5  rank_list = [1, 2, 3]
6  |
7  # list with multiple datatypes
8  mixed_list = ["Howdy", 8, (5 + 2j), 3.14]
```
Creating lists: list() constructor

The list() constructor returns a mutable sequence list of elements.

If no parameters are passed, it creates an empty list, otherwise, it creates a list of elements in the iterable.

```python
>>> l1 = [1, 2, 3]
>>> l1
[1, 2, 3]
>>> l2 = list([1, 2, 3])
>>> l2
[1, 2, 3]
```

The empty list is the list with no element.

Creating lists: list comprehension

List comprehension consists of an expression followed by a for statement inside square brackets.

```python
>>> odd_numbers = [2*n+1 for n in range(10)]
>>> print(odd_numbers)
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

A list comprehension can also contain if statements to filter elements.

```python
>>> odd_numbers = [n for n in range(20) if n % 2 == 1]
>>> print(odd_numbers)
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```
Creating lists

More examples:

```python
>>> l1 = [2, 4, 6, 8]
>>> l1
[2, 4, 6, 8]
>>> l2 = ['blue', 'red', 'yellow']
>>> l2
['blue', 'red', 'yellow']
>>> l3 = l1 + l2
>>> l3
[2, 4, 6, 8, 'blue', 'red', 'yellow]
>>> l3 += [10]
>>> l3
[2, 4, 6, 8, 'blue', 'red', 'yellow', 10]
>>> len(l3)
8
```

List: empty lists

The empty list is the list with no element.

```python
>>> l = []
>>> l
[]
>>> type(l)
<class 'list'>
```
Lists: nested lists

A list can also have lists as element. That list is a nested list.

```python
# list in a list
nested_list = ["arduino", [6, 4, 3.14], ['led']]

>>> nested_list
["arduino", [6, 4, 3.14], ['led']]
```

Lists: indexing

Like strings, each element in a list has an index. The index operator [] accesses an element in the list.

Negative indexing works the same way. Index -1 gets the last item, -2 the second last item etc.

<table>
<thead>
<tr>
<th>data1</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>123456</td>
<td>788754</td>
<td>987832</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data2</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>‘dog’</td>
<td>78</td>
<td>87.0</td>
<td>‘gorilla’</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>-4</th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>
Lists: slicing operators

To slice a range of elements in a list, or sub-list, use the slicing operators [:] or [::]

```
seq[: ]           # [seq[0], seq[1], ..., seq[-1] ]
seq[low: ]        # [seq[low], seq[low+1], ..., seq[-1] ]
seq[:high]        # [seq[0], seq[1], ..., seq[high-1]]
seq[low:high]     # [seq[low], seq[low+1], ..., seq[high-1]]
seq[:stride]      # [seq[0], seq[stride], ..., seq[-1] ]
seq[low::stride]  # [seq[low], seq[low+stride], ..., seq[-1] ]
seq[:high::stride]# [seq[0], seq[stride], ..., seq[high-1]]
seq[low:high::stride]# [seq[low], seq[low+stride], ..., seq[high-1]]
```

With a negative stride, the ordering changes since we are counting down.

```
seq[: -stride]    # [seq[-1], seq[-1-stride], ..., seq[0] ]
seq[high:: -stride]# [seq[high], seq[high-stride], ..., seq[0] ]
seq[:low:: -stride]# [seq[-1], seq[-1-stride], ..., seq[low+1]]
seq[high:low:: -stride]# [seq[high], seq[high-stride], ..., seq[low+1]]
```
Lists: slicing operators

To slice a range of elements in a list, or sub-list, use the slicing operators [:] or [::]

```python
device = list(input("Device name: "))
# elements 3rd to 5th
print(device[2:5])
# elements beginning to 4th
print(device[:4])
# elements 6th to end
print(device[5:])
# elements beginning to end
print(device[:])
```

```
>>> [evaluate lists_slicing.py]
Device name: microcontroller
['c', 'r', 'o']
['m', 't', 'c', 'r', 'o', 'c', 'o', 'n', 't', 'r']
['m', 't', 'c', 'r', 'o', 'c', 'o', 'n', 't', 'r', 'o', 'l', 'e', 'r']
```

Lists: slicing operators

To slice a range of elements in a list, or sub-list, use the slicing operators [:] or [::]

```python
>>> s = [1,2,3,4,5,6]
>>> s1 = s[:2]
>>> s1
[1, 3, 5]
>>> s2 = s[1::2]
>>> s2
[2, 4, 6]
>>> s3 = s[2:5:-1]
>>> s3
[]
>>> s4 = s[::-1]
>>> s4
[6, 5, 4, 3, 2, 1]
```
Lists: mutability

List are **mutable**: their elements can be changed **unlike strings**. How to change elements in a list?

```python
>>> l3 = [2, 'dog', 3, 'cat']
>>> l3[2] = 5
>>> l3
[2, 'dog', 5, 'cat']
```

Lists: append() and extend()

How to add elements to a list?

append() adds one element to a list.
extend() adds multiple items to a list.

```python
prime = [2, 3, 5, 7, 11]
print(prime)
prime.append([13])
print(prime)
prime.extend([17, 19, 23])
print(prime)
```

```python
[evaluate lists_append_extend.py]
[2, 3, 5, 7, 11]
[2, 3, 5, 7, 11, 13]
[2, 3, 5, 7, 11, 13, 17, 19, 23]
```
The `+` operator concatenate two lists.

The `*` operator repeats a list n times.

```
1 prime = [2, 3, 5]
2 print(prime)
3
4 prime = prime + [7, 11]
5 print(prime)
6
7 prime = prime * 2
8 print(prime)
```

```python
>>> [evaluate lists_operators.py]
[2, 3, 5]
[2, 3, 5, 7, 11]
[2, 3, 5, 7, 11, 2, 3, 5, 7, 11]
```

---

**Lists: insert()**

`list.insert(i, x)` inserts the element `x` at position `i`.

What are the outputs of the following commands?

```
> l1 = [2, 4, 6, 8]
> l1.insert(1, 3)
> l1
[2, 3, 4, 6, 8]
> l1.insert(-1, 10)
> l1
[2, 3, 4, 6, 10, 8]
> l1.insert(len(l1), 15)
```
Lists: insert()

`list.insert(i, x)` inserts the element `x` at position `i`.

What are the outputs of the following commands?

```
> l1 = [2, 4, 6, 8]
> l1.insert(1, 3)
[2, 3, 4, 6, 8]

> l1 = [2, 3, 4, 6, 8]
> l1.insert(-1, 10)
[2, 3, 4, 6, 10, 8]

> l1 = [2, 3, 4, 6, 10, 8]
> l1.insert(len(l1), 15)
[2, 3, 4, 6, 10, 8, 15]
```

Lists: delete del

del deletes one or more elements from a list.
del can also delete the list.

```
1  device = list(input("Device: "))
2  
3  # delete one character
4  del device[2]
5  print(device)
6  
7  # delete multiple items
8  del device[1:5]
9  print(device)
10 
11  # delete the entire list
12  del device
```

```
>>> [evaluate lists_delete.py]
Device: arduino
['a', 'r', 'u', 'i', 'n', 'o']
['a', 'o']
```
Lists: remove() and pop()

list.remove(x) removes the 1st element from the list whose value is x.
list.pop([i]) removes the element at the position i in the list, and returns it. If no index is specified, pop() removes and returns the last item in the list.

```python
device = list("arduino")
device.remove('u')
print(device)
print(device.pop(1))
print(device)
print(device.pop())
print(device)
device.clear()
print(device)

>>> [evaluate lists_remove_pop.py]
['a', 'r', 'd', 'i', 'n', 'o']
['a', 'd', 'i', 'n', 'o']
['a', 'd', 'i', 'n']
```

Lists: count() and index()

list.count(x) counts the number of times the element x appears.
list.index(x) returns the first index in the list where the item x is found.

```python
>>> l1
[2, 4, 6, 10, 8, 2, 4, 6, 10, 8, 8]
>>> l1.count(8)
3
>>> l1.index(8)
4
```
Lists: common methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>append()</td>
<td>Add an element to the end of the list</td>
</tr>
<tr>
<td>extend()</td>
<td>Add all elements of a list to the another list</td>
</tr>
<tr>
<td>insert()</td>
<td>Insert an item at the defined index</td>
</tr>
<tr>
<td>remove()</td>
<td>Removes an item from the list</td>
</tr>
<tr>
<td>pop()</td>
<td>Removes and returns an element at the given index</td>
</tr>
<tr>
<td>clear()</td>
<td>Removes all items from the list</td>
</tr>
<tr>
<td>index()</td>
<td>Returns the index of the first matched item</td>
</tr>
<tr>
<td>count()</td>
<td>Returns the count of number of items passed as an argument</td>
</tr>
<tr>
<td>sort()</td>
<td>Sort items in a list in ascending order</td>
</tr>
<tr>
<td>reverse()</td>
<td>Reverse the order of items in the list</td>
</tr>
<tr>
<td>copy()</td>
<td>Returns a shallow copy of the list</td>
</tr>
</tbody>
</table>

Lists: built-in functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all()</td>
<td>Return True if all elements of the list are true (or if the list is empty).</td>
</tr>
<tr>
<td>any()</td>
<td>Return True if any element of the list is true. If the list is empty, return False.</td>
</tr>
<tr>
<td>enumerate()</td>
<td>Return an enumerate object. It contains the index and value of all the items of list as a tuple.</td>
</tr>
<tr>
<td>len()</td>
<td>Return the length (the number of items) in the list.</td>
</tr>
<tr>
<td>list()</td>
<td>Convert an iterable (tuple, string, set, dictionary) to a list.</td>
</tr>
<tr>
<td>max()</td>
<td>Return the largest item in the list.</td>
</tr>
<tr>
<td>min()</td>
<td>Return the smallest item in the list.</td>
</tr>
<tr>
<td>sorted()</td>
<td>Return a new sorted list (does not sort the list itself).</td>
</tr>
<tr>
<td>sum()</td>
<td>Return the sum of all elements in the list.</td>
</tr>
</tbody>
</table>