Lists

Key Concepts:

- List Data Structure
- List Operators
- Side Effects on Lists

List Data Structure

Simple Types and Complex Types

```
>>> a = 6
>>> type(a)
<class 'int'>
>>> type(3.5)
<class 'float'>
>>> type(True)
<class 'bool'>
```

- Simple types have a single value.
- We often need to manipulate more complex "data structures" like lists, sets, multi-dimensional arrays, etc.

Python List Type

- An ordered collection of elements (objects)
- Associated with an identifier
- Can grow (or shrink) dynamically
- Elements can be of different types

Examples:

```
Weekend = ["Saturday", "Sunday"]
Multiple3 = [3, 6, 9, 12]
Roman = [[1, 'I'], [2, 'II'], [3, 'III'], [4, 'IV']]
iv = 4
Mixed = ["One", 2, 3.0, iv]
Empty = []
```

List Operators

Accessing Elements:

```
Weekend = ["Saturday", "Sunday"]
print(Weekend[0])  # Output: 'Saturday'
```

Appending Elements:

```
Multiple3 = [3, 6, 9]
Multiple3.append(21)
print(Multiple3)  # Output: [3, 6, 9, 21]
```

List Length:

```
Multiple3 = [3, 6, 9, 15, 21, 24, 27]
print(len(Multiple3))  # Output: 7
```

Printing a List:

```
l1 = [1, 2, 3]
print(l1)  # Output: [1, 2, 3]
```

Checking if an Element is in a List:

```
def search(elem, l):
    if elem in l:
        return True
    else:
        return False

my_list = [2, 5, 8, 12, 17, 25]
found = search(12, my_list)
print(found)  # Output: True
found = search(6, my_list)
print(found)  # Output: False
```

Inserting Elements:

```
Multiple3 = [3, 6, 9, 21]
Multiple3.insert(3, 15)
print(Multiple3)  # Output: [3, 6, 9, 15, 21]
```

Extending a List:

```
Multiple3.extend([24, 27])
print(Multiple3)  # Output: [3, 6, 9, 15, 21, 24, 27]
```

Popping Elements:

```
a = Multiple3.pop(0)
print(a)  # Output: 3
print(Multiple3)  # Output: [6, 9, 15, 21, 24, 27]
```

Removing Elements:

```
Multiple3.remove(24)
print(Multiple3)  # Output: [6, 9, 15, 21, 27]
```

Equality Operator:

```
list1 = [1, 2, 3]
list2 = list1
list2.append("bip")
# Both list1 and list2 are modified because they refer to the same
list.
```

Copying Lists:

```
import copy

list1 = [1, 2, 3]
list3 = list(list1)
list3.append("bip")
# list1 is not modified
# Deep copy example
list4 = copy.deepcopy(list1)
list4.append("bip")
# list1 is not modified
```

Side Effects on Lists

A function can modify a list passed as an argument, independently of its return value. This is a new form of side effect (besides print, input, turtle, etc.).

Example:

```
def add_start_end(l, start_elem, end_elem):
    l.insert(0, start_elem)
    l.append(end_elem)

my_list = [2, 3, 4]
add_start_end(my_list, 1, 5)
print(my_list)  # Output: [1, 2, 3, 4, 5]
```

Avoiding Undesired Side Effects

Copy the list if you do not want it to be modified by the function.

Example:

```
import random

def add_random(l):
    """Returns a list obtained from l by adding a random integer
between 5 and 10."""
    x = random.randint(5, 10)
    new_list = list(l)  # copy the list
    new_list.append(x)
    return new_list

original_list = [1, 2, 3]
new_list = add_random(original_list)
```

```
print(original_list)  # Output: [1, 2, 3]
print(new_list)  # Output: [1, 2, 3, <random_number>]
```

Common Mistake

Example:

```
import random

def add_random(l):
    """Returns a list obtained from l by adding a random integer
between 5 and 10."""
    x = random.randint(5, 10)
    new_list = l  # This does not copy the list
    new_list.append(x)
    return new_list

original_list = [1, 2, 3]
new_list = add_random(original_list)
print(original_list)  # Output: [1, 2, 3, <random_number>]
print(new_list)  # Output: [1, 2, 3, <random_number>]
```