4PINT (Intermediate Python)

Topic 10 – Data Structures (Part 1)

Lists | Tuples | Sets

Government of South Australia

List Data Structure

```
# Creating a List
my_subjects = ["4PYI", "4JAB","4DBB", "4UML"]
print(my_subjects) ['4PYI', '4JAB', '4DBB', '4UML']
```

Note

In many other programming languages, you would use a type called an *array* to store a sequence of data. An array has a fixed size. A Python list's size is flexible. It can grow and shrink on demand.

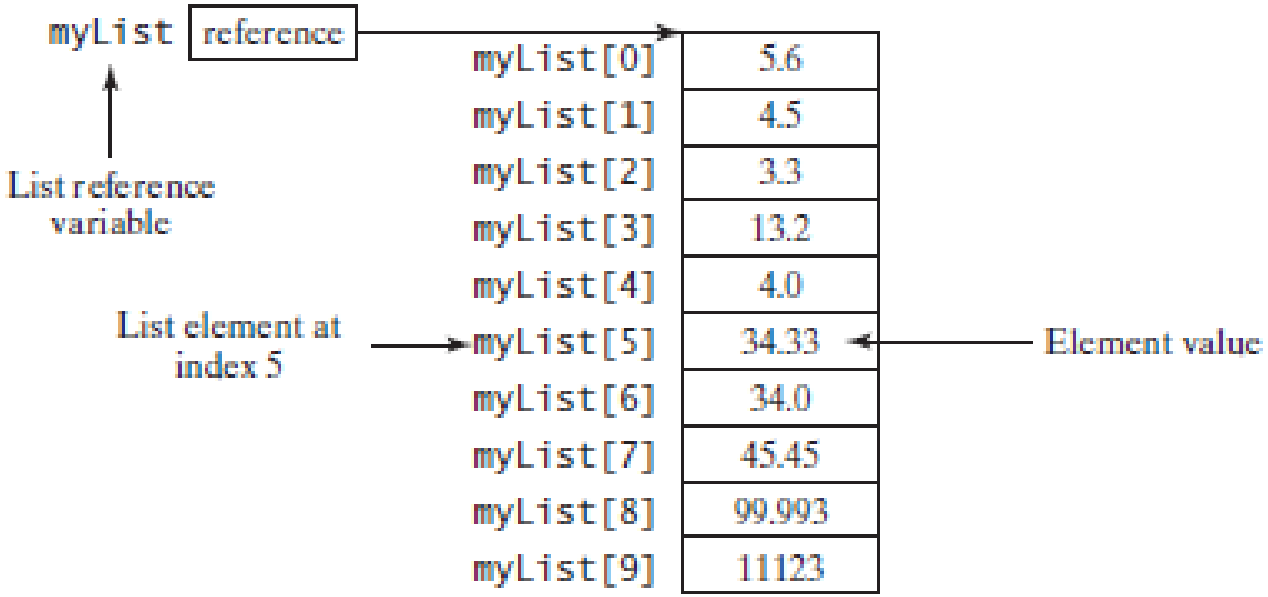
Operation	*Description*
`x in s`	True if element x is in sequence s.
`x not in s`	True if element x is not in sequence s.
`s1 + s2`	Concatenates two sequences s1 and s2.
`s * n, n * s`	n copies of sequence s concatenated.
`s[i]`	ith element in sequence s.
`s[i : j]`	Slice of sequence s from index i to j − 1.
`len(s)`	Length of sequence s, i.e., the number of elements in s.
`min(s)`	Smallest element in sequence s.
`max(s)`	Largest element in sequence s.
`sum(s)`	Sum of all numbers in sequence s.
`for loop`	Traverses elements from left to right in a `for` loop.
`<, <=, >, >=, =, !=`	Compares two sequences.

List functions

1	**append()** This method is used to add an element at the end of the list
2	**clear()** This method is used to remove all the elements from the list
3	**copy()** This method is used to return a copy of the list
4	**count()** This method is used to return the number of elements with the specified value
5	**extend()** This method is used to add elements of a list (or any iterable), to the end of the current list
6	**index()** This method is used to return the index of the first element with the specified value
7	**insert()** This method is used to add an element at the specified position
8	**pop()** This method is used to remove the element at the specified position
9	**remove()** This method is used to remove the item with the specified value
10	**reverse()** This method is used to reverse the order of the list
11	**sort()** This method is used to sort the list

Using Index Operator

```
myList = [5.6, 4.5, 3.3, 13.2, 4.0, 34.33, 34.0, 45.45, 99.993, 11123]
```



Lists contd..

Python List is one of the Python Collections which is ordered and changeable

```
fruitList = ["apple", "banana", "cherry“, ”orange”, ”kiwi”, ”melon”, ”mango”]

print(fruitList[1]) # What is the output?
print(fruitList[-1]) # What is the output?
print(fruitList[2:5]) # What is the output?
print(fruitList[:4]) # What is the output?
print(fruitList[2:]) # What is the output?
print(fruitList[-4:-1]) # What is the output?
```

Modifying List (Adding an item)

```
fruitList[1] = ‘blackcurrent”
print(fruitList) # What does the list look like?
```

Lists (contd..)

Loop through a List

```
fruitList = ["apple", "banana", "cherry“, ”orange”, ”kiwi”, ”melon”, ”mango”]
for fruit in fruitList:
    print(fruit)
```

Check if Item Exists

```
if “apple” in fruitList:
    print(“We have apples”)
```

List Length

```
    print(len(fruitList)) # What is the output?
```

Add Items

```
    fruitList.append(“pear”) # What does the list look like?
    fruitList.insert(1, “berry”) #What does the list look like?
```

Remove Item

```
    fruitList.remove(“banana”)
```

Delete Item

```
    del fruitList(0) # Which item is deleted?
```

The +,* and in/not in Operators

```
>>> list1 = [2, 3]
>>> list2 = [1, 9]
>>> list3 = list1 + list2
>>> list3
[2, 3, 1, 9]
>>>
>>> list4 = 3 * list1
>>> list4
[2, 3, 2, 3, 2, 3]
>>>
```

```
>>> list1 = [2, 3, 5, 2, 33, 21]
>>> 2 in list1
True
```

```
>>> 2 not in list1
False
>>>
```

Comparing Lists

- Can use >,>=,<,<=,== and != Operators
- Both Lists must contain the same types of elements
- Comparisons use *lexicographical* ordering:
  - If ist two elements differ, outcome determined
  - If they are equal, the next two are compared..

```
>>> list1 = ["green", "red", "blue"]
>>> list2 = ["red", "blue", "green"]
>>> list2 == list1
False
>>> list2 != list1
True
>>> list2 >= list1
False
>>> list2 > list1
False
>>> list2 < list1
True
>>> list2 <= list1
True
>>>
```

List Comprehensions

It consist of brackets containing an expression followed a **for** clause, then zero or more **for** or **if** clauses

The Comprehension produces a list with the results from evaluating the expression

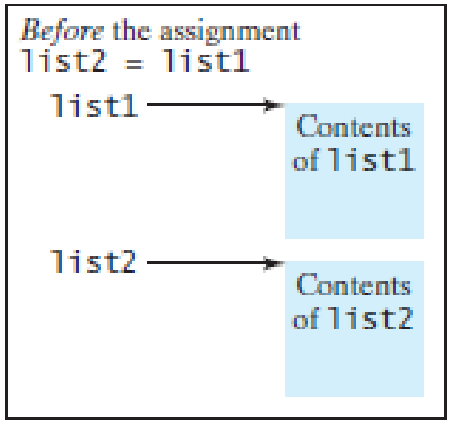
Examples:

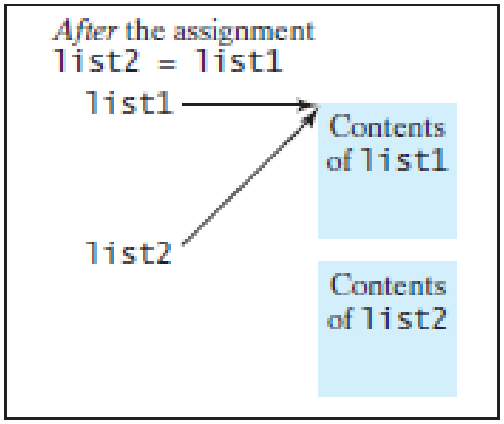
```
list1 = [x for x in range(5)]
# Returns a list of [0,1,2,3,4]
list2 = [0.5 * x for x in list1]
# Returns a list of [0.0,0.5,1.5,2.0]
list3 = [x for x in list2 if x < 1.5]
# Returns a list of [0.0,0.5,1.0]
```

Copying Lists

If you want to duplicate a list use the '=' operator:

```
list2 = list1
```





To copy the data from one list to another and keep both lists, you have to copy individual elements from the source list to the target list:

```
list2 = [x for x in list1]
```

or simply:

```
list2 = [] + list1
```

Passing Lists to Functions

When passing a list to a function, the contents of the list may change after the function call, since the list is a mutable object.

Example:

```
def main():
    x = 1 # x is an int variable
    y = [1, 2, 3] # y is a list

    m(x, y) # Invoke m with arguments x and y

    print("x is", x)
    print("y[0] is", y[0])

def m(number, numbers):
    number = 1001 # Assign a new value to number
    numbers[0] = 5555 # Assign a new value to numbers[0]

main() # Call the main function
```

```
x is 1
y[0] is 5555
```

Passing a list as a default argument

```
def add(x, lst = []):
    if x not in lst:
        lst.append(x)

    return lst

def main():
    list1 = add(1)
    print(list1)

    list2 = add(2)
    print(list2)

    list3 = add(3, [11, 12, 13, 14])
    print(list3)

    list4 = add(4)
    print(list4)

main()
```

```
[1]
[1, 2]
[11, 12, 13, 14, 3]
[1, 2, 4]
```

If you require the default list to be **[ ]** for every function call:

```
def add(x, lst = None):
    if lst == None:
        lst = []
    if x not in lst:
        lst.append(x)

    return lst
```

Tuples

- Like lists, but elements are fixed and once created you cannot add new elements,delete or replace elements.
- More efficient than lists
- Creating a Tuple:

```
t1 = () # Create an empty tuple

t2 = (1, 3, 5) # Create a tuple with three elements

# Create a tuple from a list
t3 = tuple([2 * x for x in range(1, 5)])
# Create a tuple from a string
t4 = tuple("abac") # t4 is ['a', 'b', 'a', 'c']
```

- Can use functions such as len, min,max and sum
- Can use for loop and in and not in operators
- If an individual element in a Tuple is mutable, it can be changed

tafeSA Government of South Australia

Sets

- Like lists but no duplicate elements allowed
- More efficient than lists
- Creating a Set:

```
s1 = set() # Create an empty set
s2 = {1, 3, 5} # Create a set with three elements
s3 = set([1, 3, 5]) # Create a set from a tuple

# Create a set from a list
s4 = set([x * 2 for x in range(1, 10)])
```

- Can create a list or tuple from a set:

```
list(set) tuple(set).
```

- Can create a set from a string

```
s5 = set("abac") # s5 is {'a', 'b', 'c'}
```

Note:Even though char 'a' appears twice in the string, it appears only once in a set

- Each element in a set must be Hashable
  - Each object in Python has a *hash* value
  - An object is *hashable* if its hash value never changes during its lifetime (lists are not hashable)

tafeSA
Government of South Australia

Manipulating and Accessing Sets

- Can add and remove elements using:
  add(e) or remove(e)
- Can use len, min,max and sum functions on a set and for loop to traverse elements
- Can use in or not in operators
- The remove(e) method will throw a KeyError exception if element e is not in set

```
>>> s1 = {1, 2, 4}
>>> s1.add(6)
>>> s1
{1, 2, 4, 6}
>>> len(s1)
4
>>> max(s1)
6
```

```
>>> min(s1)
1
>>> sum(s1)
13
>>> 3 in s1
False
>>> s1.remove(4)
>>> s1
{1, 2, 6}
```

Note

The `remove(e)` method will throw a `KeyError` exception if the element to be removed is not in the set.

Subset and Superset

- A set s1 is a subset of s2 if every element in s1 is also in s2.
- Use the s1.issubset(s2) to determine if s1 is a subset of s2
- Example:

```
>>> s1 = {1, 2, 4}
>>> s2 = {1, 4, 5, 2, 6}
>>> s1.issubset(s2) # s1 is a subset of s2
True
>>>
```

- A set s1 is a superset of s2 if every element in s2 is also in s1.
- Use the s1.issubset(s2) to determine if s1 is a subset of s2
- Example:

```
>>> s1 = {1, 2, 4}
>>> s2 = {1, 4, 5, 2, 6}
>>> s2.issuperset(s1) # s2 is a superset of s1
True
>>>
```

Equality Test

- You can use the == and != operators to test if two sets contain the same elements (regardless of order)

```
>>> s1 = {1, 2, 4}
>>> s2 = {1, 4, 2}
>>> s1 == s2
True
>>> s1 != s2
False
>>>
```

- Makes no sense to compare sets with >,>=,<=,< operators as the elements in a set are not ordered
- When used in sets it has special meaning:
  - `s1 < s2` returns `True` if `s1` is a proper subset of `s2`.
  - `s1 <= s2` returns `True` if `s1` is a subset of `s2`.
  - `s1 > s2` returns `True` if `s1` is a proper superset of `s2`.
  - `s1 >= s2` returns `True` if `s1` is a superset of `s2`.

Note

If `s1` is a proper subset of `s2`, every element in `s1` is also in `s2`, and at least one element in `s2` is not in `s1`. If `s1` is a proper subset of `s2`, `s2` is a proper superset of `s1`.

Set Operations

- Union – to combibe elements in two sets together using the union( ) or | operator

```
>>> s1 = {1, 2, 4}
>>> s2 = {1, 3, 5}
>>> s1.union(s2)
{1, 2, 3, 4, 5}
>>>
>>> s1 | s2
{1, 2, 3, 4, 5}
>>>
```

- Intersection – shows only elements in both sets using intersaction () or & operator

```
>>> s1 = {1, 2, 4}
>>> s2 = {1, 3, 5}
>>> s1.intersection(s2)
{1}
>>>
>>> s1 & s2
{1}
>>>
```

- Difference – display elements contained in set1 but not in set2 using difference( ) or - operator

```
>>> s1 = {1, 2, 4}
>>> s2 = {1, 3, 5}
>>> s1.difference(s2)
{2, 4}
```

```
>>> s1 - s2
{2, 4}
```

tafeSA
Government of South Australia

Comparing the Performance of Sets and Lists

- Sets are more efficient than lists for the in operator and for the remove( ) method
- Elements in a List can be accessed using the index operator
- Sets do not support the index operator because elements in a Set are unordered
- To traverse elements in a Set use a for loop
- Run the Python program SetListPerformanceTest.py that compares the performances between a Set and List using the in operator and the remove( ) method- Sample output below

```
To test if 10000 elements are in the set
The runtime is 5 milliseconds

To test if 10000 elements are in the list
The runtime is 4274 milliseconds

To remove 10000 elements from the set
The runtime is 7 milliseconds

To remove 10000 elements from the list
The runtime is 1853 milliseconds
```

Next Week – Data Structures 2
Dictionary