Lists, List Comprehension, Strings and Multiple Inputs

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

Feedback

How comfortable are you with material so far

Green: Very Comfortable

Yellow: Moderately Comfortable

Red: Not comfortable

Structured Data Types

- Looked at scalar data types - which have simple values
  - E.g. integer, real numbers, boolean
  - These data have no components/parts within them
- Python also has data types in which the data object is compound, i.e. a collection of scalar (structured) data items combined into one object
- In such data types, you can access the full data object, or its components / items
- In python there are some built-in structured types: Lists, Strings, Sets, Tuples, Dictionaries

- We will first discuss lists and strings (strings often not considered in the same category)
- Will then discuss how to take multiple inputs from user

Lists

Lists

- Lists are used to store multiple items in one variable
- List items are ***ordered*** - there is a notion of ith item
- Items can be of any type (incl structured types), and can be mixed - though generally lists of type of items is used
- List items are changeable; duplicate values are allowed
- Lists are created using square brackets, e.g.
  - `L1 = [1, 4, 5, 2, 9, 5]`
  - `L2 = ["apple", "banana"]`
  - `L3 = [1, 2.0, "str", 3, 5]`
  - `L4 = [] #Empty List`
- List items are indexed, the first item has index [0], the second item has index [1] etc.
- `L1[0]` is 1; `L1[2]` is 5, `L2[1]` is "banana", `L3[3]` is 3
- String objects: those within "" - for now treat them as one object

Accessing a List Item : Indexing

- To access an Individual item stored in a list we use indexing
- The position of any item in list (with the first item as 0) is known as its index. This index can be used to access an item in the list.
- Usage : `L[i]` returns the **i**th item in the list where **i** starts from 0.
- Most programming languages use positive index i.e. `L[i]` accesses the **i**th item, with indexes starting from 0.
- Python also allows for negative index. The items are accessed from the end of the list; -1 means last item, -2 last but one….
- Example : `L3 = [1, 2.0, "str", 3, 5]`
  - `L3[0]` is 1
  - `L3[1]` is 2.0
  - `L3[2]` is "str"
  - `L3[-1]` is 5
  - `L3[-2]` is 3

Accessing Multiple List Items : Slicing

- We can also extract a portion of a list. Operation is known as Slicing.
- Syntax : `L[i:j]` # returns a list which is sub-list of L for index range i:j (i.e. includes i, but not j, no. of items is : j-i)
  - Omitting i starts the slicing at the beginning of the list i.e `L[:j]`
  - Omitting j stops the slicing at the end of the list i.e `L[i:]`
  - A third parameter step is available to determine the jump size `L[start:stop:step]`. Similar to range() fn.
- Example : `L3 = [1, 2.0, "str", 3, 5]`
  - `L3[1:3]` returns a list `[2.0, "str"]`
  - `L3[:2]` is `[1, 2.0]`
  - `L3[3:]` is `[3,5]`
  - `L3[-4:-2]` is list from 4th item from end to 2nd item; `[2.0, "str"] (note: -4 included, -2 is not; no of items is: -2 - (-4) = 2;`
  - `L3[1:4:2] returns [2.0,3]`

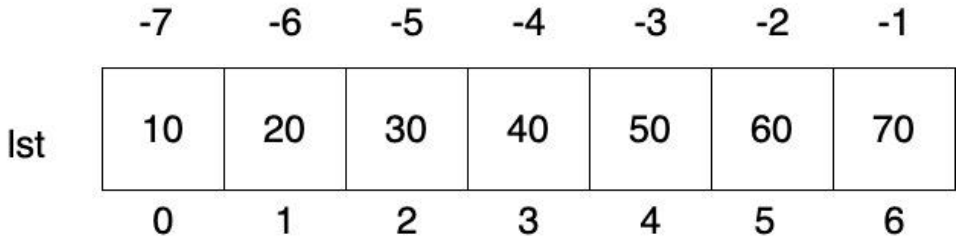
List Slicing Examples

- Used to access a particular range of elements in a list
- Uses the slicing operator i.e. colon(:)

  lst[start : stop : step]

- Examples :

```
print(lst[1:4])
[20, 30, 40]
print(lst[1:5:2])
[20, 40]
print(lst[2:])
[30, 40, 50, 60, 70]
```



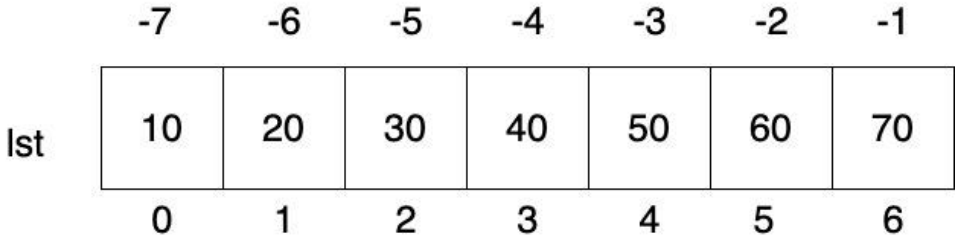
List Slicing Examples

- Used to access a particular range of elements in a list
- Uses the slicing operator i.e. colon(:)

  lst[start : stop : step]

- Examples :

```
print(lst[:4])
[10, 20, 30, 40]
print(lst[:-6:-2])
[70, 50, 30]
print(lst[-5:-2: 2])
[30, 50]
```



Quiz (ungraded)

Write down your answer

L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Q1: What is the value of L[3]

Q2: Value of L[2:5]

Q3: L[-2:-5]

Q4: L[-5: -2]

We will do them in python shell

Change List Items

- We can use the index to specify an item - assign it a new value
- L[i] = new_val # item L[i] will now be new_val
- Example :
  - `L = [“physics”, “chemistry”, “maths”, 2021, 2025]`
  - `L[2] = “biology”`
  - `print(L) #[“physics”, “chemistry”, “biology”, 2021, 2025]`
- Can also change a range of list items by selecting a slice and assigning it values (we will ignore it)

Some important operations on Lists

- The **`len()`** function gives the **length** of the list, i.e. the no. of items

  `len([1, 2, 5]) is 3`

- The **`sum()`** function gives the **sum** of items (for int/float - error for str/mixed)

  `sum([1, 2, 5]) is 8`

- Checking if an **item exists** in a list: **in** operator (membership testing). Only for checking of items, not for sub-list

  `l3 = [1, 2.0, "str", 3, 5]`

  4 in l3 is False; "str" in l3 is True; 2.0 in l3 is True

- Checking **absence** using **`not in`**

  `12 not in l3` will return **`True`**; **`5 not in l3`** is **`False`**

- Concatenate : **Join** lists by **+** operation

  `l1+l2` returns a new list by adding l2 to end of l1

- Can **replicate** lists by * operation:

  Given **`l1=[1,2]`**, then **`l1*4 returns [1, 2, 1, 2, 1, 2, 1, 2]`**

Operations on a List – Adding Data

- Multiple operations are provided on a list object
- Syntax of an operation on an object: var_name.opname()
- If the var L is a list, some operations that can be done are:
  - **L.append(item)** # adds an item to the end of L
  - **L.insert(i, item)** # insert item at i th location in L - earlier items from i onwards are pushed back (i.e. their index increases)
  - **L.extend(list)** # append the list to L, (the argument must be a list)
  -

Operations on a List – Removing Data

- Multiple operations provided for removing items from a list L, e.g
  - **L.remove(item)** # removes the item if it exists (first occurrence only), else gives ValueError
  - **L.pop(index)** # removes L[index] item ; if no i specified, removes last item
  - **L.clear()** # clears the list; L becomes an empty list: []
- Items can also be removed from the list using the del function:
  - **`del L[index]`** # removes the item at the specified index

Operations on Lists – others

- **L.index(item)** # returns the lowest index where the searched item appears
- **L.reverse()** # reverses the order of list elements, updates the existing list, does not return any value

Example

Given a list as input, write a program to create a new list that has products of frequency of repeating elements (which come together) and value of elements.

Input :

lst = [2, 2, 2, 2, 5, 5, 5, 8, 8, 8, 8, 8, 6, 6, 6, 6, 4, 4, 7, 7, 7, 5, 7,7]

Output :

res = [8, 15, 40, 24, 8, 21, 5, 14]

```
res = []
count = 1

for i in range(1,len(lst)):
  if lst[i]!=lst[i-1]:
    res.append(count*lst[i-1])
    count = 0
  count = count+1

res.append(count*lst[-1])
print(res)
```

Example : List Operations

Replace 1st occurrence of the given element with new value if found

Input:

l1 = [2, 3, 3, 5, 7, 3, 4, 3]

Element to be replaced: 3

New value = 13

Output:

[2, 11, 3, 5, 7, 3, 4, 3]

```
l1 = [2, 3, 3, 5, 7, 3, 4, 3]

elt = 3
new_value = 13
index = l1.index(elt) #Index :  1
print("Index : ",index)


l1[index] = new_value
print(l1)
# [2, 11, 3, 5, 7, 3, 4, 3]
```

Quiz : Single Correct

What would be the output of the code given on the right?

A. Error
B. [8, 12]
C. [1, 2, 3, 4, 1, 2, 3, 4]
D. [1, 2, 1, 2, 3, 4, 3, 4]

```
list1 = [1, 2]
list2 = [3, 4]

print ((list1 + list2)*2)
```

Quiz : Single Correct

What would be the output of the code given on the right?

A. Error
B. [8, 12]
C. [1, 2, 3, 4, 1, 2, 3, 4]
D. [1, 2, 1, 2, 3, 4, 3, 4]

```
list1 = [1, 2]
list2 = [3, 4]

print ((list1 + list2)*2)
```

Explanation: Recall that + is for concatenation and * is for replication

Copying a List

- Say, L1 is a list of [1, 2, 3]
- L2 = L1 does not create a new list - just creates a new var L2 which also points to the same list
- We can make truly another copy of the list by copy() operation

```
L2=L1.copy()
```

- We can also create a copy of the list using slicing.

```
L2 = L1[:] # Leaving both start and stop as blank
```

- With L1=L2, if you change L1[i] (or L2[i]) , the change is in the list, and as both L1 and L2 point to same list, both will show
- With copy, a new list is created which has same contents as list pointed by L1, and pointer to this is assigned to L2

Copying a list – pythontutor.com

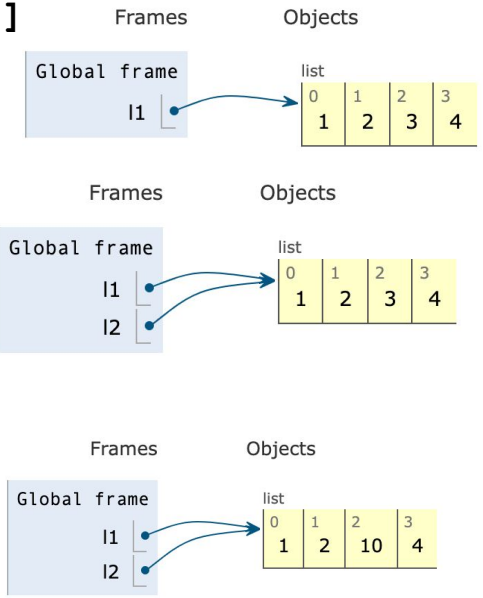
```
l1 = [1,2,3,4]

l2 = l1

l2[2] = 10

print(l1)

print(l2)
```



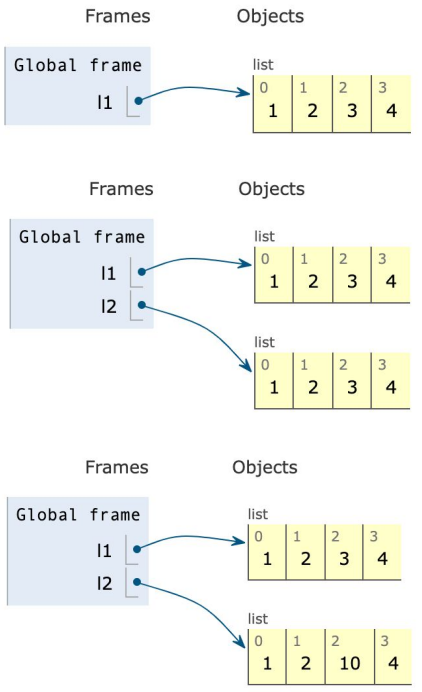

```
l1 = [1,2,3,4]

l2 = l1.copy()

l2[2] = 10

print(l1)

print(l2)
```



Looping over items in a list

- For loop and lists are made for each other - easy to loop

  for item in <list>:

      Loop_body

====

- Looping using index is also easy

  N = len(list)

  for i in range(N):

      use L[i]

- Printing items in a list:

  for item in list1:

      print(item)

====

for i in range(len(list1)):

    print(list1[i])

Eg: Sum of squares of items in a list

```
lst = [1, 3, 2, 5, 9, 4]
total = 0
for item in lst:
    total = total + item*item
print(total)
```

```
lst = [1, 3, 2, 5, 9, 4]
total = 0
for i in range(len(lst)):
    total = total + (lst[i])**2
print(total)
```

Sorting lists

- Lists themselves are any collection of items
- Sorting them is a very common need
- Python provides a powerful operation, with variations, to sort - treats items as strings and does alphanumeric sorting
- **lst.sort()** # lst changed to have items in ascending order
- **lst.sort(reverse=True)** # lst sorted in descending order
- Remember, strings are case sensitive

Special sorting with own key

- Can provide own function as "key" for sorting
- lst.sort(key=myfn)
- Items of list are now arranged based on value of applying myfn on each item
- I.e. sorting of items is done as per the value myfn(item) of each item - items do not change
- E.g a list of strings
- fl = ["orange", "mango", "kiwi", "pineapple", "banana"]
- Sorting using fl.sort(key=len) will sort it using the value provided by the len function for each item, i.e. by length
- E.g. write a function sq to square the value, use it to sort

Using Lists to Assign Multiple vars

- lst = [1,2,3]
- How to assign the list elements to variables?
- Method 1:

```
a = lst[0]
b = lst[1]
c = lst[2]
```

- Method 2 (Better way): Sequence unpacking

```
a,b,c = lst # Now a=1, b=2, c=3
x, y = lst # ValueError: too many values to unpack (expected 2)
```