Python - Lists

April 17, 2023

1 Lists

- Lists are an ordered collection of elements
- Lists can be homogeneous and heterogeneous in Python
- Means we can store similar type of data or different types of data in a list.
- Lists are mutable object. i.e., the contents of lists can be changed (modified) even after creation
- Python lists are dynamic in nature. i.e., you can add or delete elements in a list and size changes accordingly.
- List elements are enclosed within square brackets []
- [10, 20, 30] –> List of integers
- [10.2, 12.3] –> List of floating point values
- [‘a’, ‘b’, ‘c’] –> List of strings
- [[10, 20], [20, 30], [40, 50]] –> List of lists
- [10, 12.2, ‘hello’, [‘a’, ‘b’, ‘c’]] –> Heterogeneous list
- Lists are indexed, indexes start from 0
- In python lists (any other sequence types also for that matter) can be indexed from backwards also using negative indexes
- Negative indexes start from -1
- So -1 index will alway hold the last (from left) element of the list
- Length of a list = number of elements in the list
- Length of a list can be known by using len() function on the list

[55]:
```
#ind  -4  -3  -2   -1
lst = [10, 20, 30, 40]
# ind  0   1   2   3
print(lst[2])
print(lst[3])
print(lst[-2])
print(lst[-4])
```

```
30
40
30
10
```

[48]:
```
s = 'hello world'
#ind 012345678910
```

```
print(s[8])
```

```
r
```

[52]:
```
r = range(10, 100, 10)
print(r[7])
```

```
80
```

[56]:
```
lst = [10, 20, 'hello', [10, 20]]
# len() --> gives the number elements
# in an iterable
print(len(lst))
```

```
4
```

[57]:
```
s = 'hello world'
print(len(s))
```

```
11
```

1.1 Reading a list of integers from user

1.1.1 If input is in single line

- Like 10 20 30 40 50
- And we have to read them into a list so it looks like [10, 20, 30, 40]

[1]:
```
lst = list(map(int, input().split())) # This takes the data from the user and␣
↪puts it in the list form
print(lst)
```

```
10 20 30 40 50
[10, 20, 30, 40, 50]
```

[60]:
```
x = list(map(int, input().split()))
print(x)
print(type(x))
print(sum(x))
```

```
10 20 30 40 50
[10, 20, 30, 40, 50]
<class 'list'>
150
```

1.1.2 If input is in seperate lines

- In this case we will always get the len() of the list beforehand
- We can run a loop len() times and get each element in each iteration and append it to the list

[61]:
```
n = int(input()) # 5 - no. of element to be inserted into the list
lst = [] # [10, 20, 30, 40, 50]
for i in range(n):
    x = int(input())
    lst.append(x)
print(lst)
print(type(lst))
print(sum(lst))
```

```
5
10
20
30
40
50
[10, 20, 30, 40, 50]
<class 'list'>
150
```

[3]:
```
# Harry Hermoine Wesely Malfoy Dumbledore
# names = ['Harry', 'Hermoine', 'Wesely']
names = list(map(str, input().split())) # reading list of strings that are in␣
 ↪single line separated by space
print(names)
```

```
Harry Hermoine Wesely Malfoy Dumbledore
['Harry', 'Hermoine', 'Wesely', 'Malfoy', 'Dumbledore']
```

[4]:
```
# reading same names given in different lines
names = []
n = int(input()) # length of list
for _ in range(n):
    s = input() # reading each element
    names.append(s) # appending it to the list
print(names)
```

```
5
Harry
Hermoine
Wesely
Malfoy
Dumbledore
['Harry', 'Hermoine', 'Wesely', 'Malfoy', 'Dumbledore']
```

1.2 Traversing a list

- You can loop through a list in the following 2 ways
  - Element based Traversal
  - Index Based Traversal

[5]:
```
# Element based Traversal
x = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
# in 0   1   2   3   4   5   6   7   8   9
for element in x:
    print(element)
```

```
10
20
30
40
50
60
70
80
90
100
```

[6]:
```
# Element based Traversal
# Counting even numbers in the list
x = [11, 22, 36, 47, 56, 66, 79, 83, 94, 100]
# in 0   1   2   3   4   5   6   7   8   9
cnt = 0
for i in x:
    if i % 2 == 0:
        cnt += 1 # cnt = cnt + 1
print(cnt)
```

```
6
```

[8]:
```
# Index based Traversal
x = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100] # 10
# in 0   1   2   3   4   5   6   7   8   9
for i in range(len(x)):
    print(x[i], end = ' ') #  x[i] ->  element at ith index in list x
```

```
10 20 30 40 50 60 70 80 90 100
```

[9]:
```
# Index based Traversal
# Count even numbers that are present at even indices in the list
x = [11, 22, 36, 47, 56, 66, 79, 83, 94, 100]
# in 0   1   2   3   4   5   6   7   8   9
cnt = 0
for i in range(len(x)):
    if i % 2 == 0 and x[i] % 2 == 0:
        cnt += 1
print(cnt)
```

```
3
```

[11]:
```
# In place operations to the list cannot be done using
# element based traversal
x = [10, 20, 30, 40, 50]
# [11, 21, 31, 41, 51]
for i in x:
    i += 1
print(x)
```

```
[10, 20, 30, 40, 50]
```

[12]:
```
# In place operations to the list cannot be done using
# element based traversal
x = [10, 20, 30, 40, 50]
# [11, 21, 31, 41, 51]
for i in range(len(x)):
    x[i] += 1
print(x)
```

```
[11, 21, 31, 41, 51]
```

1.3 List Methods

- Lists are Mutable
- Mutability: Ability to change after creation
- Mutable Objects: Lists, Sets, Dictionaries
- Immutable Objects: Strings, Tuples, Integers, Floating Point Values
- Insertion Methods
  - append()
  - insert()
  - extend()
- Deletion Methods
  - pop()
  - remove()
  - clear()
- Searching Methods
  - index()
  - count()
- In-Place Modifications
  - reverse()
  - sort()
- copy()

[13]:
```
# Mutability
marks = [55, 46, 77] # created a list
print(marks)
marks[1] = 96 # Changing the  list (Modifying)
print(marks)
```

```
[55, 46, 77]
```

```
[55, 96, 77]
```

[14]:
```
# Immutability
s = 'hallo' #
print(s)
s[1] = 'e' # Trying to modify the string (NOT ALLOWED)
print(s)
```

```
hallo
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_14068\4149375899.py in <cell line: 4>()
      2 s = 'hallo' #
      3 print(s)
----> 4 s[1] = 'e'
      5 print(s)

TypeError: 'str' object does not support item assignment
```

1.3.1 list.append()

- Used to append an object at the end of an existing list
- Can append one object at a time

[15]:
```
lst = [] # empty list
lst.append(10) # list  becomes [10]
lst.append('hello') # list becomes [10, 'hello']
lst.append([10, 20, 30]) # list becomes [10, 'hello', [10, 20, 30]]
print(lst)
```

```
[10, 'hello', [10, 20, 30]]
```

[16]:
```
ages = [14, 7, 41, 65, 44, 79, 6, 2, 21, 22, 43, 68, 1, 90]
# child = < 13 [7, 6, 2, 1]
# teen = >= 13 and <= 19 [14]
# adult = >= 20 and < 50 [41, 44, 21, 22, 43]
# old = >= 50 [65, 79, 68, 90]
child = []
teen = []
adult = []
old = []
for i in ages:
    if i < 13:
        child.append(i)
    elif 13 <= i <= 19:
        teen.append(i)
    elif 20 <= i < 50:
```

```
        adult.append(i)
    else:
        old.append(i)
print(child, teen, adult, old, sep = '\n')
```

```
[7, 6, 2, 1]
[14]
[41, 44, 21, 22, 43]
[65, 79, 68, 90]
```

1.3.2 list.extend()

- Used to extend an existing list with given iterable
- Takes every element of the iterable and appends it to the existing list
- We should only pass iterables as arguments to extend() method.

[19]:
```
marks = [45, 65, 87] # created # [45, 65, 87]
new_marks = [47, 96, 25]
marks.extend(new_marks) # extending the list with new_marks list elements
print(marks)
```

```
[45, 65, 87, 47, 96, 25]
```

[20]:
```
characters = ['A', 'B', 'C', 'D', 'E']
characters.append('FGHIJ')
print(characters)
```

```
['A', 'B', 'C', 'D', 'E', 'FGHIJ']
```

[21]:
```
characters = ['A', 'B', 'C', 'D', 'E']
characters.extend('FGHIJ')
print(characters)
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
```

1.3.3 list.insert(index, object)

- Inserts the given object at specified index
- Moves all the existing elements one step towards right to insert the given object at specified index

[22]:
```
x = [10, 20, 30, 40]
#ind 0   1   2   3
print(x)
x.insert(1, 50)
print(x)
```

```
[10, 20, 30, 40]
[10, 50, 20, 30, 40]
```

[23]:
```
names = ['Harry', 'Hermoine', 'Wesely', 'Malfoy', 'Dumbledore']
# output = ['H', 'a', 'r', 'r', 'y', 'H', 'e', 'r', 'm', 'o', 'i']
output = []
for i in names:
    output.extend(i)
print(output)
```

```
['H', 'a', 'r', 'r', 'y', 'H', 'e', 'r', 'm', 'o', 'i', 'n', 'e', 'W', 'e', 's',
'e', 'l', 'y', 'M', 'a', 'l', 'f', 'o', 'y', 'D', 'u', 'm', 'b', 'l', 'e', 'd',
'o', 'r', 'e']
```

1.3.4 list.pop(index=-1)

- Used to remove and return the element at specified index
- Index is defaulted to -1 (Removes last element in the list)
- Returns the element after removing
- Throws an index out of bound error, if specified index is not present in the list

[24]:
```
x = [10, 20, 30, 40, 50]
popped_element = x.pop()
print(popped_element)
print(x)
```

```
50
[10, 20, 30, 40]
```

[25]:
```
x = [10, 20, 30, 40, 50]
popped_element = x.pop(3)
print(popped_element)
print(x)
```

```
40
[10, 20, 30, 50]
```

[26]:
```
x = [10, 20, 30, 40, 50]
popped_element = x.pop(5)
print(popped_element)
print(x)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_14068\2241645272.py in <cell line: 2>()
      1 x = [10, 20, 30, 40, 50]
----> 2 popped_element = x.pop(5)
      3 print(popped_element)
      4 print(x)
```

```
IndexError: pop index out of range
```

[1]:
```
ls=[10,20]
ls.append(30)
print(ls)
```

```
[10, 20, 30]
```

[5]:
```
n = 10 # 1010
print(n.bit_count())
```

```
2
```

[6]:
```
n = 15 # 1111
print(n.bit_count())
```

```
4
```

1.3.5 list.remove()

- Element based deletion
- Throws an error if the element is not present in the list
- Always deletes the first occurence of the element from left

[7]:
```
x = [10, 20, 30, 40]
x.remove(20)
print(x)
```

```
[10, 30, 40]
```

[8]:
```
x = [20, 20, 30, 20, 30, 20]
print(x.remove(20))
print(x)
```

```
None
[20, 30, 20, 30, 20]
```

1.3.6 list.clear()

- Clears the contents of the list, leaving an empty list

[9]:
```
x = [10, 20, 30]
x.clear()
print(x)
```

```
[]
```

1.3.7 list.index()

- Used to find the index of an element if it's a member in the list

- Throws an error if the specified element is not a member of the list
- Gives the first occurence index if multiple occurences are present.

[11]:
```
lst = [10, 20, 30, 40]
print(lst.index(40))
```

```
3
```

[12]:
```
lst = [10, 20, 30, 40]
print(lst.index(50))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_9132\3277452527.py in <cell line: 2>()
      1 lst = [10, 20, 30, 40]
----> 2 print(lst.index(50))

ValueError: 50 is not in list
```

[13]:
```
lst = [10, 20, 10, 20, 10]
print(lst.index(10))
```

```
0
```

1.3.8 list.count()

- Returns the count of a specified element in the list
- Example: lst = [10, 10, 10, 20]
- Applying lst.count(10) gives 3 as 10 is present for 3 times in the list
- Returns 0 if the specified element is not a member of the list.

[14]:
```
lst = [10, 10, 20, 10]
print(lst.count(10))
```

```
3
```

[15]:
```
lst = [10, 20, 10, 20]
print(lst.count(30))
```

```
0
```

1.3.9 list.reverse()

- Reverses the given list in-place
- You'll lose your original list
- The original list itself will modified to store the reversed version.

[16]:
```
lst = [10, 20, 30, 40, 50]
print(f"List before reversal: {lst}")
lst.reverse()
print(f"List after reversal: {lst}")
```

```
List before reversal: [10, 20, 30, 40, 50]
List after reversal: [50, 40, 30, 20, 10]
```

1.3.10 list.sort()

- Sorts the given list (in ascending order) in-place.
- Can only be applied on homogeneous lists
- Takes two optional keyword arugments
  - reverse
  - key
- reverse will be set to False by default
- setting reverse = True will sort the list in descending order
- You can specify a function as key, if the sort has to take place based on a function

[17]:
```
lst = [2, 17, -6, 4, -147, 16, 32, 24]
lst.sort() # Ascending
print(lst)
```

```
[-147, -6, 2, 4, 16, 17, 24, 32]
```

[18]:
```
names = ['Harry', 'Hermoine', 'Wesely', 'Malfoy', 'Dumbledore']
names.sort()
print(names)
```

```
['Dumbledore', 'Harry', 'Hermoine', 'Malfoy', 'Wesely']
```

[19]:
```
lst = [2, 17, -6, 4, -147, 16, 32, 24]
lst.sort(reverse = True) # Descending
print(lst)
```

```
[32, 24, 17, 16, 4, 2, -6, -147]
```

[20]:
```
names = ['Harry', 'Hermoine', 'Wesely', 'Malfoy', 'Dumbledore']
names.sort(reverse = True)
print(names)
```

```
['Wesely', 'Malfoy', 'Hermoine', 'Harry', 'Dumbledore']
```

[21]:
```
names = ['Harry', 'Hermoine', 'Weasely', 'Malfoy', 'Dumbledore']
# lengths 5         8              7        6          10
# 5 6 7 8 10
# ['Harry', 'Malfoy', 'Weasely', 'Hermoine', 'Dumbledore']
names.sort(key=len)
print(names)
```

```
['Harry', 'Malfoy', 'Weasely', 'Hermoine', 'Dumbledore']
```

[ ]:
```
names = ['Harry', 'Hermoine', 'Weasely', 'Malfoy', 'Dumbledore']
# ['Harry', 'Malfoy', 'Weasely', 'Hermoine', 'Dumbledore']
names.sort(key=len)
print(names)
```

[22]:
```
nums = [10, 4, 5, 24, 12]
nums.sort()
print(nums)
```

```
[4, 5, 10, 12, 24]
```

[23]:
```
def factor_count(n):
    fc = 0
    for i in range(1, n + 1):
        if n % i == 0:
            fc += 1
    return fc
# sorting based on number of factors
nums = [10, 4, 5, 24, 12]
#       4   3  2  7   6
# 5 4 10 12 24
nums.sort(key = factor_count)
print(nums)
```

```
[5, 4, 10, 12, 24]
```

[24]:
```
strings = ['zac', 'apiq', 'paeqiz', 'ruuiosq']
strings.sort()
print(strings)
```

```
['apiq', 'paeqiz', 'ruuiosq', 'zac']
```

[29]:
```
# Functions that returns vowel count of a given string
def vowels(s):
    cnt = 0
    for i in s:
        if i in 'aeiou':
            cnt += 1
    return cnt
strings = ['zac','ruuiosq', 'paeqiz', 'apiq']
# no. of vowels in each string
# ['zac', 'apiq', paeqiz, ruuiosq]
strings.sort(key = vowels, reverse = True)
print(strings)
```

```
['ruuiosq', 'paeqiz', 'apiq', 'zac']
```

1.3.11 list.copy()

- Deep Copy
  - If two objects are deep copied, the changes we make to one object will be reflected on the other.
- Shallow Copy
  - If two objects are shallow copied, the changes we make to one object will not be refleced on the other.
- In Python copying 2 lists in the following manner performs a deep copy - lst2 = lst1
- So every change we make to lst2 will be reflected on lst1
- Every change we make to lst1 will be reflected on lst2
- To avoid this we use list.copy() method which performs shallow copy.

[32]:
```
lst1 = [10, 20, 30]
lst2 = lst1 # deep copy
# modifying list1
lst1.append(40) # 40 will be appended to list2 too.
print(lst1)
print(lst2)
```

```
[10, 20, 30, 40]
[10, 20, 30, 40]
```

[33]:
```
lst1 = [10, 20, 30]
lst2 = lst1.copy() # shallow copy
# modifying list1
lst1.append(40)
print(lst1)
print(lst2)
```

```
[10, 20, 30, 40]
[10, 20, 30]
```

1.4 Slicing

- Used to get a sublist from an existing list
- Slicing operation can be done using the slice operator [:] or [::]
- Syntax
- If we use this [::]
  - [start:stop:index_jump]
  - start indicates starting index (Defaulted to 0 if not provided)
  - stop indicates ending index (Excluded generally and defaulted to len(list) if not provided)
  - index_jump indicates no. of indexes to move in one step (Defaulted to 1 if not provided)
  - Let's Say we have a list lst = [10, 20, 30, 40, 50, 60, 70, 80]
  - If you write lst[::], this results in entire list as start=0, stop=8 and index_jump=1
  - If you write lst[3::], this results in [40, 50, 60, 70, 80] as start=3, stop=8, index_jump=1
  - If you write lst[3:7:], this results in [40, 50, 60, 70] only as stop=7 is always excluded.

– If you write lst[::2], this results in [10, 30, 50, 70] as the different between index_jumps=2

[34]:
```
lst = [10, 20, 30, 40, 50]
# ind  0   1   2   3   4
print(lst[1:4]) # index 1 to index 3
```

```
[20, 30, 40]
```

[2]:
```
lst = [10, 20, 30, 40, 50, 60, 70, 80]
print(lst[::])
print(lst[3::])
print(lst[3:7:])
print(lst[::2])
print(lst[1::3])
```

```
[10, 20, 30, 40, 50, 60, 70, 80]
[40, 50, 60, 70, 80]
[40, 50, 60, 70]
[10, 30, 50, 70]
[20, 50, 80]
```

1.5 List Comprehensions

[35]:
```
# Task
lst = [1, 2, 3, 4, 5, 6]
# squares = [1, 4, 9, 16, 25, 36]
squares = []
for i in lst:
    squares.append(i * i)
print(squares)
```

```
[1, 4, 9, 16, 25, 36]
```

[36]:
```
# Task
lst = [1, 2, 3, 4, 5, 6]
squares = [i * i for i in lst]
print(squares)
```

```
[1, 4, 9, 16, 25, 36]
```

[37]:
```
# Task
# Generate a list that contains the
# factors of a given number
# n = 10
# factors = [1, 2, 5, 10]
n = 10
factors = []
for i in range(1, n + 1):
    if n % i == 0:
```

```
        factors.append(i)
print(factors)
```

```
[1, 2, 5, 10]
```

[38]:
```
# Task
# Generate a list that contains the
# factors of a given number
# n = 10
# factors = [1, 2, 5, 10]
n = 10
factors = [i for i in range(1, n + 1) if n % i == 0]
print(factors)
```

```
[1, 2, 5, 10]
```

[40]:
```
names = ['Harry', 'Hermoine', 'Weasely', 'Malfoy', 'Dumbledore']
new_list = [len(i) for i in names]
print(new_list)
```

```
[5, 8, 7, 6, 10]
```

[41]:
```
names = ['Harry', 'Hermoine', 'Weasely', 'Malfoy', 'Dumbledore']
new_list = [max(i) for i in names]
print(new_list)
```

```
['y', 'r', 'y', 'y', 'u']
```

[5]:
```
nums = [16, 25, 64, 121, 289]
# [4.0, 5.0, 8.0, 11.0, 17.0]
new_list = [i ** 0.5 for i in nums]
print(new_list)
```

```
[4.0, 5.0, 8.0, 11.0, 17.0]
```

[6]:
```
import math
nums = [16, 25, 64, 121, 289]
# [4.0, 5.0, 8.0, 11.0, 17.0]
new_list = [math.sqrt(i) for i in nums]
print(new_list)
```

```
[4.0, 5.0, 8.0, 11.0, 17.0]
```

[12]:
```
cities = ['berlin', 'tokyo',
          'palermo','nairobi',
          'denver', 'rio',
          'lisbon', 'stockholm',
          'bogota', 'helsinki']
# Generate a list of cities, whose name length is less
# than or equal to 5
```

```
result = [i for i in cities if len(i) <= 5]
print(result)
```

```
['tokyo', 'rio']
```

[13]:
```
cities = ['berlin', 'tokyo',
          'palermo','nairobi',
          'denver', 'rio',
          'lisbon', 'stockholm',
          'bogota', 'helsinki']
# Genearate a list of cities, whose maximum character is 'r'
result = [city for city in cities if max(city) == 'r']
print(result)
```

```
['berlin', 'palermo', 'nairobi', 'rio']
```

[14]:
```
cities = ['berlin', 'tokyo',
          'palermo','nairobi',
          'denver', 'rio',
          'lisbon', 'stockholm',
          'bogota', 'helsinki']
# Generate a list of cities, who contain 'i' in them as one of
# the characters
result = [city for city in cities if 'i' in city]
print(result)
```

```
['berlin', 'nairobi', 'rio', 'lisbon', 'helsinki']
```

[16]:
```
cities = ['berlin', 'tokyo',
          'palermo','nairobi',
          'denver', 'rio',
          'lisbon', 'stockholm',
          'bogota', 'helsinki']
# Generate a list of cities that start with either
# 't' or 'd' or 'n'
result = [i for i in cities if i[0] in 'tdn']
print(result)
```

```
['tokyo', 'nairobi', 'denver']
```

[18]:
```
cities = ['berlin', 'tokyo',
          'palermo','nairobi',
          'denver', 'rio',
          'lisbon', 'stockholm',
          'bogota', 'helsinki']
# Generate a list of cities that contains 'o' but doesn't end with 'o'
result = [city for city in cities if  'o' in city and city[-1] != 'o']
print(result)
```

```
['nairobi', 'lisbon', 'stockholm', 'bogota']
```

[23]:
```
def is_prime(n: int) ->  bool:
    fc = 0
    for i in range(1, n + 1):
        if n % i == 0:
            fc += 1
    return fc == 2
primes = [i for i in range(1, 50) if is_prime(i)]
print(primes)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

[26]:
```
# 1 2 3
# 1 2 3
# (1, 1), (1, 2), (1, 3)
# (2, 1), (2, 2), (2, 3)
# (3, 1), (3, 2), (3, 3)
res = []
for i in range(1, 4):
    for j in range(1, 4):
        res.append([i, j])
print(res)
```

```
[[1, 1], [1, 2], [1, 3], [2, 1], [2, 2], [2, 3], [3, 1], [3, 2], [3, 3]]
```

[27]:
```
res = [[i, j] for i in range(1, 4) for j in range(1, 4)]
print(res)
```

```
[[1, 1], [1, 2], [1, 3], [2, 1], [2, 2], [2, 3], [3, 1], [3, 2], [3, 3]]
```

[28]:
```
res = [[i, j] for i in range(1, 4) for j in range(1, 4) if i + j != 4]
print(res)
```

```
[[1, 1], [1, 2], [2, 1], [2, 3], [3, 2], [3, 3]]
```

[29]:
```
lst = [0 for i in range(100)]
print(lst)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

[30]:
```
lst = [0]*100
print(lst)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

[33]:
```
x = [1, 2, 3, 4, 5, 6, 7, 8, 9]
# square of a number if it is even
# else cube of a number
# output = [1, 4, 27, 16, 125, 36, 343, 64, 729]
# [f(x) if condition else g(x) for loop]
output = [i * i if i % 2 == 0 else i * i * i for i in x]
print(output)
```

```
[1, 4, 27, 16, 125, 36, 343, 64, 729]
```

1.6 Nested Lists

- Why do have to nest list?
- List inside another list

[6]:
```
# Nested List
marks = [[45, 65, 77, 46, 39], [74, 89, 46, 34, 29],[84, 32, 21, 67, 89]]
#                  0                     1                     2
print(len(marks))
print(marks[1])
print(marks[0])
print(marks[2])
for i in marks:
    print(i, end = ' ')
```

```
3
[74, 89, 46, 34, 29]
[45, 65, 77, 46, 39]
[84, 32, 21, 67, 89]
[45, 65, 77, 46, 39] [74, 89, 46, 34, 29] [84, 32, 21, 67, 89]
```

[7]:
```
# Nested List
#         0   1   2   3   4     0   1   2   3   4    0   1   2   3   4
marks = [[45, 65, 77, 46, 39], [74, 89, 46, 34, 29],[84, 32, 21, 67, 89]]
#                  0                     1                     2
print(marks[1][0])
```

```
74
```

[10]:
```
# Accessing matrix elements
mat = [[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]]
for i in range(3): # i = 0, 1, 2
    for j in range(3): # j = 0, 1, 2
        print(mat[i][j], end  = ' ')
    print()
```

```
10 20 30
40 50 60
```

```
70 80 90
```

[12]:
```
# Accessing matrix elements
mat = [[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]]
for i in mat:
    for j in i:
        print(j, end = ' ')
    print()
```

```
10 20 30
40 50 60
70 80 90
```

[14]:
```
# Find out the sum of elements in the below matrix
mat = [[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]]
s = 0
for i in range(3): # i = 0, 1, 2
    for j in range(3): # j = 0, 1, 2
        s += mat[i][j]
print(s)
```

```
450
```

[17]:
```
# Find out the sum of elements in the below matrix
mat = [[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]]
s = 0
for i in mat:
    s += sum(i)
print(s)
```

```
450
```

[21]:
```
# Find out the sum of elements in the below matrix
mat = [[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]]
s = sum([sum(i) for i in mat])
print(s)
```

```
450
```

[22]:
```
# Matrix Reading from user
r, c = map(int, input().split())
mat = []
```

```
for i in range(r):
    sub_list = list(map(int, input().split()))
    mat.append(sub_list)
print(mat)
```

```
3 4
10 20 30 40
50 60 70 80
90 10 20 30
[[10, 20, 30, 40], [50, 60, 70, 80], [90, 10, 20, 30]]
```

[23]:
```
# Matrix Reading from user
r, c = map(int, input().split())
mat = [list(map(int, input().split())) for i in range(r)]
mat
```

```
3 3
10 20 30
40 50 60
70 80 90
```

[23]:
```
[[10, 20, 30], [40, 50, 60], [70, 80, 90]]
```

2 Strings

[26]:
```
s = 'hello world'
# Functions
print(len(s))
print(min(s))
print(max(s))
```

```
11

w
```

[29]:
```
s = 'hello world'
# Methods
print(s.upper())
print(s.lower())
print(s.title())
```

```
HELLO WORLD
hello world
Hello World
```

[ ]:
```
# Methods on strings
upper()
lower()
```

```
title()
swapcase()
isupper()
islower()
istitle()
isdigit()
isalpha()
isalnum()
find()
index()
count()
split()
join()
ljust()
rjust()
lstrip()
rstrip()
strip()
replace()
```

[30]:
```
# Split splits the given string based on delimiter and returns a list
# Delimiter is defaulted to spaces
s = 'hello all this is Python'
print(s.split())
```

```
['hello', 'all', 'this', 'is', 'Python']
```

[35]:
```
time = "10:45:32"
hours, minutes, seconds = map(time.split(':'))
print(time.split(':'))
print(hours*2, minutes*2, seconds*2)
```

```
['10', '45', '32']
1010 4545 3232
```

[36]:
```
s = 'this is python and python is easy'
print(s.split('i'))
# ['th', 's ', 's python and python ', 's easy']
```

```
['th', 's ', 's python and python ', 's easy']
```

[37]:
```
s = 'this is python and python is easy'
print(s.split('python'))
# ['this is ', ' and ', ' is easy']
```

```
['this is ', ' and ', ' is easy']
```