List

Collection in Python for sequences of diverse data types.

List vs Array

- **Array**: Homogeneous; Contiguous memory; Fast; Numerical/Scientific use.
- **List**: Heterogeneous; Non-contiguous memory; Programmer-friendly; General-purpose.

1. Create
2. Access
3. Edit
4. Add
5. Delete
6. Operations
7. Functions

Arrays vs. Lists in Memory:

- **Arrays:**
  - `int arr[50];` ---> 50 contiguous memory blocks.
  - Elements stored in binary form at consecutive addresses.
- **Lists:**
  - `list_example = [1, 2, 3]` ---> Elements at different locations.
  - Stores references/pointers to elements, not the values.

In [1]:
```
# proof --->
L = [1, 2, 3]
print(id(L))
print(id(L[0]))
print(id(L[1]))
print(id(L[2]))
print(id(1))
print(id(2))
print(id(3))
```

```
1931975005504
1931889344752
1931889344784
1931889344816
1931889344752
1931889344784
1931889344816
```

List Characteristics

- *Ordered*
- *Mutable*
- *Heterogeneous*
- *Duplicates allowed*
- *Dynamic size*
- *Nesting supported*
- *Indexable*
- *Any object type*

1. Create

In [1]:
```
# Empty
L = []
L
```

Out[1]:
```
[]
```

In [2]:
```
# 1D ---> Homo
L1 = [1, 2, 3, 4, 5]
L1
```

Out[2]:
```
[1, 2, 3, 4, 5]
```

In [3]:
```
# Hetrogenous
L2 = ["Hello", 4.5, 5, 6, True, 5+6j]
L2
```

Out[3]:
```
['Hello', 4.5, 5, 6, True, (5+6j)]
```

In [9]:
```
# Multidimentional list:
# 2D
L3 = [1,2,3,[4,5]]
L3
```

Out[9]:
```
[1, 2, 3, [4, 5]]
```

In [10]:
```
# 3D
L4 = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
L4
```

Out[10]:
```
[[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
```

In [11]:
```
# Using Type Conversion
L5 = list("Manish")
L5
```

Out[11]:
```
['M', 'a', 'n', 'i', 's', 'h']
```

In [6]:
```
L6 = list()
L6
```

Out[6]:
```
[]
```

2. Access

In [6]:
```
L1
```

Out[6]:
```
[1, 2, 3, 4, 5]
```

In [7]:
```
L1[3]
```

Out[7]:
```
4
```

In [14]:
```
L1[-2]
```

Out[14]:
```
4
```

In [15]:
```
# Slicing
L1[0:3]
```

Out[15]:
```
[1, 2, 3]
```

In [16]:
```
L1[::-1]
```

Out[16]:
```
[5, 4, 3, 2, 1]
```

In [12]:
```
L3
```

Out[12]:
```
[1, 2, 3, [4, 5]]
```

In [19]:
```
L3[3]
```

Out[19]:
```
[4, 5]
```

In [23]:
```
L3[-1]
```

Out[23]:
```
[4, 5]
```

In [21]:
```
x = L3[-1]
x
```

Out[21]:
```
[4, 5]
```

In [27]:
```
x
```

Out[27]:
```
[4, 5]
```

In [69]:
```
L3[-1][0]
```

Out[69]:
```
4
```

In [70]:
```
L3[-1][-1]
```

Out[70]:
```
5
```

In [27]:
```
L4
```

Out[27]:
```
[[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
```

In [36]:
```
L4[0][0][-1]
```

Out[36]:
```
2
```

In [32]:
```
L4[1][1]
```

Out[32]:
```
[7, 8]
```

In [37]:
```
L4[0][0][0]
```

Out[37]:
```
1
```

3. Edit

In [37]:
```
L1
```

Out[37]:
```
[1, 2, 3, 4, 5]
```

In [38]:
```
# Editing With Indexing
L1[0] = 100
print(L1)
# List in Python are Mutable
```

```
[100, 2, 3, 4, 5]
```

In [39]:
```
L1[-1] = 500
L1
```

Out[39]:
```
[100, 2, 3, 4, 500]
```

In [40]:
```
# Editing With Slicing
L1[1:4] = [200, 300, 400]
L1
```

Out[40]:
```
[100, 200, 300, 400, 500]
```

4. Add

- append()
- extend()

- insert()

In [8]:
```
L1
```

Out[8]:
```
[100, 200, 300, 400, 500]
```

In [9]:
```
L1.append(1000)
L1
```

Out[9]:
```
[100, 200, 300, 400, 500, 1000]
```

In [10]:
```
L1.append("hello")
L1
```

Out[10]:
```
[100, 200, 300, 400, 500, 1000, 'hello']
```

In [11]:
```
L1.extend([5000, 6000, 7000])
L1
```

Out[11]:
```
[100, 200, 300, 400, 500, 1000, 'hello', 5000, 6000, 7000]
```

In [12]:
```
L1.append([5, 6])
L1
```

Out[12]:
```
[100, 200, 300, 400, 500, 1000, 'hello', 5000, 6000, 7000, [5, 6]]
```

In [13]:
```
L1.extend("goa")
L1
```

Out[13]:
```
[100,
 200,
 300,
 400,
 500,
 1000,
 'hello',
 5000,
 6000,
 7000,
 [5, 6],
 'g',
 'o',
 'a']
```

In [14]:
```
L1.insert(3, "world")
L1
```

Out[14]:
```
[100,
 200,
 300,
 'world',
 400,
 500,
 1000,
 'hello',
 5000,
 6000,
 7000,
 [5, 6],
 'g',
 'o',
 'a']
```

5. Delete

- del
- .remove()
- .pop()
- .clear()

In [19]:
```
L2
```

Out[19]:
```
['Hello', 4, 5, 6, True, (5+6j)]
```

In [20]:
```
del L2
L2
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[20], line 2
      1 del L2
----> 2 L2

NameError: name 'L2' is not defined
```

In [21]:
```
del L1[2]
L1
```

Out[21]:
```
[100,
 200,
 'world',
 400,
 500,
 1000,
 'hello',
 5000,
 6000,
 7000,
 [5, 6],
 'g',
 'o',
 'a']
```

In [22]:
```
del L1[-4]
L1
```

Out[22]:
```
[100, 200, 'world', 400, 500, 1000, 'hello', 5000, 6000, 7000, 'g', 'o', 'a']
```

In [23]:
```
del L1[-3:]
L1
```

Out[23]:
```
[100, 200, 'world', 400, 500, 1000, 'hello', 5000, 6000, 7000]
```

In [24]:
```
L1.remove("hello")
L1
```

Out[24]:
```
[100, 200, 'world', 400, 500, 1000, 5000, 6000, 7000]
```

In [25]:
```
L1.pop()
L1
```

Out[25]:
```
[100, 200, 'world', 400, 500, 1000, 5000, 6000]
```

In [26]:
```
L1.pop()
L1
```

Out[26]: [100, 200, 'world', 400, 500, 1000, 5000]

In [27]:
```
L1.clear()
L1
```

Out[27]: []

6. Operations

- Arithmetic
- Membership
- Loop

In [37]:
```
L = [1, 2, 3, 4]
L1 = [5, 6, 7, 8]
```

In [38]:
```
# Concatenation/Merge
L + L1
```

Out[38]: [1, 2, 3, 4, 5, 6, 7, 8]

In [39]:
```
L
```

Out[39]: [1, 2, 3, 4]

In [31]:
```
L1
```

Out[31]: [5, 6, 7, 8]

In [56]:
```
L * 2
```

Out[56]: [1, 2, 3, 4, 1, 2, 3, 4]

In [32]:
```
# Loops
for i in L:
    print(i)
```

```
1
2
3
4
```

In [58]:
```
L3
```

Out[58]:
```
[1, 2, 3, [4, 5]]
```

In [33]:
```
for i in L3:
    print(i)
```

```
1
2
3
[4, 5]
```

In [34]:
```
4 in L3
```

Out[34]:
```
False
```

In [35]:
```
[4, 5] in L3
```

Out[35]:
```
True
```

7. Functions

- len()
- min()
- max()
- sorted()

In [66]:
```
L = [1,2,3,4]
```

In [67]:
```
len(L)
```

Out[67]: 4

In [68]:
```
min(L)
```

Out[68]: 1

In [69]:
```
max(L)
```

Out[69]: 4

In [70]:
```
sorted(L)
```

Out[70]: [1, 2, 3, 4]

In [71]:
```
sorted(L, reverse = True)
```

Out[71]: [4, 3, 2, 1]

In [72]:
```
L
```

Out[72]: [1, 2, 3, 4]

In [73]:
```
L.sort(reverse = True)
```

In [74]:
```
L
```

Out[74]: [4, 3, 2, 1]

In [75]:
```
L.sort()
L
```

Out[75]: [1, 2, 3, 4]

In [43]:
```
# count
L = [1, 2, 1, 3, 4, 1, 5]
L.count(3)
```

Out[43]:
```
1
```

In [41]:
```
L
```

Out[41]:
```
[1, 2, 1, 3, 4, 1, 5]
```

In [48]:
```
# index
L.index(1)
```

Out[48]:
```
0
```

In [49]:
```
# reverse
L = [2, 1, 5, 7, 0]
# permanently reverses the list
L.reverse()
print(L)
```

```
[0, 7, 5, 1, 2]
```

In [74]:
```
# sort (vs sorted)
L = [2, 1, 5, 7, 0]
print(L)
print(sorted(L)) # New sorted list
print(L)         # Original list (unchanged)
L.sort()
print(L)         # Original list (sorted)
```

```
[2, 1, 5, 7, 0]
[0, 1, 2, 5, 7]
[2, 1, 5, 7, 0]
[0, 1, 2, 5, 7]
```

In [50]:
```
# copy ---> shallow
L = [2, 1, 5, 7, 0]
print(L)
print(id(L))
L1 = L.copy()
print(L1)
print(id(L1))
```

```
[2, 1, 5, 7, 0]
2304631558272
[2, 1, 5, 7, 0]
2304631509504
```

In [120]:
```
"hello how are you".title()
```

Out[120]:
```
'Hello How Are You'
```

In [92]:
```
# Title Case a String Without title()
sample = "how are you?"
sample.split()
L = []
for i in sample.split():
    L.append(i.capitalize())
print(L)
print(" ".join(L))
```

```
['How', 'Are', 'You?']
How Are You?
```

In [72]:
```
sample = "saurabh@gmail.com"
print(sample[:sample.find("@")])
```

```
saurabh
```

In [121]:
```
L1 = [1, 1, 2, 2, 3, 3, 4, 4]
# Output: L2 = [1, 2, 3, 4]
L = []
for i in L1:
    if i not in L:
        L.append(i)
print(L)
```

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

List Comprehension

Compact list creation.

```
newlist = [expresion for item in iterable if condition == True]
```

Example:

`squares = [x**2 for x in range(10)]` ---> squares 0-9.

Advantages

- *Efficient*: Time & space.
- *Concise*: Fewer lines.
- *Formulaic*: Iteration ---> expression.

In [51]:
```
L = [1, 2, 3, 4, 5, 6, 7]
L
```

Out[51]:
```
[1, 2, 3, 4, 5, 6, 7]
```

In [52]:
```
L1 = [item * 2 for item in L]
L1
```

Out[52]:
```
[2, 4, 6, 8, 10, 12, 14]
```

In [54]:
```
L2 = [ i**2 for i in range(10)]
L2
```

Out[54]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [55]:
```
L3 = [i**2 for i in range(10) if i%2 != 0]
L3
```

Out[55]:
```
[1, 9, 25, 49, 81]
```

In [57]:
```
fruits = ['Apple', 'Orange', 'Mango', 'Guava']
fruits
```

Out[57]:
```
['Apple', 'Orange', 'Mango', 'Guava']
```

In [58]:
```
L4 = [i for i in fruits if i[0]=="O"]
L4
```

Out[58]:
```
['Orange']
```

In [21]:
```
# Add 1-10 to list
L = []
for i in range(1, 11):
  L.append(i)
print(L)
```

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

In [22]:
```
L = [i for i in range(1, 11)]
print(L)
```

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

In [27]:
```
# Scalar Multiplication
v = [2, 3, 4] # Vector
s = -3        # Scalar
# [-6, -9, -12]
[s*i for i in v]
```

Out[27]:
```
[-6, -9, -12]
```

In [24]:
```
# Add squares
L = [1, 2, 3, 4, 5]
[i**2 for i in L]
```

Out[24]:
```
[1, 4, 9, 16, 25]
```

In [16]:
```
# Print nums divisible by 5 from 1 to 50
[i for i in range(1, 51) if i%5 == 0]
```

Out[16]:
```
[5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
```

In [60]:
```
# Languages starting with 'p'
languages = ['java', 'python', 'php', 'c', 'javascript']
[language for language in languages if language.startswith('p')]
```

Out[60]:
```
['python', 'php']
```

In [62]:
```
# Nested If with List Comprehension
basket = ['apple', 'guava', 'cherry', 'banana']
my_fruits = ['apple', 'kiwi', 'grapes', 'banana']
# Add fruits from `my_fruits` that are in `basket` and start with 'a'
[i for i in my_fruits if i in basket if i.startswith('a')]
```

Out[62]:
```
['apple']
```

In [56]:
```
# 3x3 Matrix via Nested List Comprehension
[[i*j for i in range(1, 4)]for j in range(1,4)]
```

Out[56]:
```
[[1, 2, 3], [2, 4, 6], [3, 6, 9]]
```

In [58]:
```
# Cartesian Products ---> List Comprehension on 2 lists together
L1 = [1, 2, 3, 4,5]
L2 = [5, 6, 7, 8]
[i*j for i in L1 for j in L2]
```

Out[58]:
```
[5, 6, 7, 8, 10, 12, 14, 16, 15, 18, 21, 24, 20, 24, 28, 32, 25, 30, 35, 40]
```

List Traversal

1. **Itemwise**
2. **Indexwise**

In [32]:
```
# Itemwise
L = [1, 2, 3, 4]
for i in L:
  print(i)
```

```
1
2
3
4
```

In [33]:
```
# Indexwise
L = [1, 2, 3, 4]
for i in range(0, len(L)):
  print(L[i])
```

```
1
2
3
4
```

Zip

`zip()` Function:

- Yields zip obj: iterator of tuples.
- Combines 1st items of each iterator, then 2nd, etc.
- Example:

```
list1 = [1, 2, 3]
list2 = ['a', 'b', 'c']
zipped = zip(list1, list2)
print(list(zipped))  # Output: [(1, 'a'), (2, 'b'), (3, 'c')]
```

Length Mismatch:

- Shortest iterator determines output length.

In [65]:
```
# Add items of 2 lists indexwise
L1 = [1, 2, 3, 4]
L2 = [-1, -2, -3, -4]
list(zip(L1,L2))
[i+j for i, j in zip(L1, L2)]
```

Out[65]:
```
[0, 0, 0, 0]
```

In [66]:
```
L = [1, 2, print, type, input]
print(L)
```

```
[1, 2, <built-in function print>, <class 'type'>, <bound method Kernel.raw_input of <ipykernel.ipkernel.IPythonKernel o
bject at 0x0000021894B51DE0>>]
```

Disadvantages of Python Lists

- Slow
- Risky usage
- High memory usage

In [67]:
```
a = [1, 2, 3]
b = a.copy()
print(a)
print(b)
a.append(4)
print(a)
print(b)
# lists are mutable
```

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

List Programs

In [1]:
```
# Split list into odd and even
L = [1, 2, 3, 4, 5, 6]

# Odd numbers
odd_numbers = [x for x in L if x % 2 != 0]

# Even numbers
even_numbers = [x for x in L if x % 2 == 0]

print("Odd numbers:", odd_numbers)
print("Even numbers:", even_numbers)
```

```
Odd numbers: [1, 3, 5]
Even numbers: [2, 4, 6]
```

In [49]:
```python
# List Input from User

# 1. Prompt input
input_string = input("Enter the list elements separated by spaces: ")

# 2. Split string
string_list = input_string.split()

# 3. Convert to integers
integer_list = [int(item) for item in string_list]

# 4. Output
print("The input list is:", integer_list)
```

```
Enter the list elements separated by spaces: 1 2 3 4 5
The input list is: [1, 2, 3, 4, 5]
```

In [42]:
```python
# Merge 2 Lists Without + Operator
L1 = [1, 2, 3, 4]
L2 = [5, 6, 7, 8]

# 1. Using `extend()` Method
L1 = [1, 2, 3, 4]
L2 = [5, 6, 7, 8]
# Merge L2 into L1
L1.extend(L2)
print("Merged list:", L1)

# 2. Using for Loop
L1 = [1, 2, 3, 4]
L2 = [5, 6, 7, 8]
# Merge L2 into L1 using a loop
for element in L2:
    L1.append(element)
print("Merged list:", L1)
```

```
Merged list: [1, 2, 3, 4, 5, 6, 7, 8]
Merged list: [1, 2, 3, 4, 5, 6, 7, 8]
```

In [6]:
```
# Replace item in list
L = [1, 2, 3, 4, 5, 3]
# replace 3 with 300

L = [1, 2, 3, 4, 5, 3]
old_item = 3
new_item = 300
# Iterate through the list and replace old_item with new_item
for i in range(len(L)):
    if L[i] == old_item:
        L[i] = new_item
print("Updated list:", L)
```

```
Updated list: [1, 2, 300, 4, 5, 300]
```

In [7]:
```
# Convert 2D to 1D List

# Define 2D list
L2D = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# Initialize 1D list
L1D = []

# Flatten 2D to 1D
for sublist in L2D:
    for item in sublist:
        L1D.append(item)
print("1D list:", L1D)
```

```
1D list: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [8]:
```python
# Remove Duplicates from List
L = [1, 2, 1, 2, 3, 4, 5, 3, 4]

# Convert to set ---> Removes duplicates; Convert back to list.
L_unique = list(set(L))

print("List with duplicates removed:", L_unique)
```

```
List with duplicates removed: [1, 2, 3, 4, 5]
```

In [7]:
```python
# Check if list is ascending
def is_ascending(L):
    for i in range(len(L) - 1):
        if L[i] > L[i + 1]:
            return False
    return True

# Test
L1 = [1, 2, 3, 4, 5]
L2 = [1, 3, 2, 4, 5]

print("L1 is in ascending order:", is_ascending(L1))
print("L2 is in ascending order:", is_ascending(L2))
```

```
L1 is in ascending order: True
L2 is in ascending order: False
```