Chapter 2 List

Python provides a type called a list that can store a collection of data of any size. You can store multiple items in a single list variable. Lists allow duplicate items.

1. Creating Lists

Python lists are written with square brackets [ ]. List items can be of any data type (string, int, boolean, etc.):

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

A list can contain different data types:

```
list4 = ["abc", 34, True, 40.5, "male"]
```

Example 01: Create a list.

```
mylist = ["apple", "banana", "cherry"]

print(mylist)
```

Output:

```
['apple', 'banana', 'cherry']
```

Example 02: You can use brackets ([ ]) and multiplication operator to create a list that contains a number of items, and initialize these item with the same values.

```
list1 = ["Hello"] * 5

print(list1)
```

Output:

```
['Hello', 'Hello', 'Hello', 'Hello', 'Hello']
```

Example 03: Using split method to make a list.

```
string = "apple, banana, cherry"

mylist = string.split(", ")

print(mylist)
```

Output:

```
['apple', 'banana', 'cherry']
```

Note: `split()` method returns a list of item(s).

2. Converting List into String

The `join()` function is one of the simplest methods to convert a list to a string in python. The main point to keep in mind while using this function is that the `join()` function can convert only those lists into string that contains only string as its items.

Example 05: Convert a list into a string

```
list1 = ["Programming", "is", "fun"]

str1 = " ".join(list1)

print(str1)
```

Output:

```
Programming is fun
```

3. `in` Operator

Example 05: the **`in`** operator lets you loop through all the members of a list and check if there's a member in the list that is equal to the given item.

```
num = 1
x = [0, 1, 2]

if num in x:
   print(num, "is in the list.")
else:
   print(num, "is not in the list.")
```

Output:

```
1 is in the list.
```

Example 06: the **`in`** operator lets you loop through all the members of a list and check if there's a member in the list that is equal to the given item.

```
num = 1
x = [0, 1, 2]

if num not in x:
   print(num, "is not in the list.")
else:
   print(num, "is in the list.")
```

Output:

```
1 is in the list.
```

4. Displaying Lists Items

Example 07: Display list items.

```
mylist = [1, 2, 3]

print(mylist)
print(*mylist)
print(*mylist, sep = "\t")
```

Output:

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

5. Accessing List Items

Same as strings, you can access list items by referring to the index number. The first item has index `0`. Negative indexing means beginning from the end, `-1` refers to the last item, `-2` refers to the second last item etc.

Example 08: Access list items by referring to the index number.

```
mylist = ["apple", "banana", "cherry"]

print(mylist[0])
print(mylist[1])
print(mylist[-1])
print(mylist[-2])
```

Output:

```
apple
banana
cherry
banana
```

6. Updating List Items

Example 09: Update list items.

```
mylist = ["apple", "banana", "cherry"]

mylist[1] = "watermelon"
mylist[-1] = "orange"

print(mylist)
```

Output:

```
['apple', 'watermelon', 'orange']
```

7. Slicing a List

Slicing a list means extracting a part of a list.

Example 10: Slice a list where its data in positions 1, 2 and 3.

```
mylist = ["apple", "banana", "cherry", "watermelon", "orange"]

newlist = mylist[1:4]

print(newlist)
```

Output

```
['banana', 'cherry', 'watermelon']
```

8. Reversing a List

Same as strings, use can reverse a list using `[::-1]`.

Example 11: Reverse a list

```
mylist = ["apple", "banana", "cherry", "watermelon", "orange"]

newlist = mylist[::-1]

print(newlist)
```

Output

```
['orange', 'watermelon', 'cherry', 'banana', 'apple']
```

9. Iterate Through a List

As strings are sequences, you can loop through them directly.

Example 12: Iterate through a list

```
mylist = ["apple", "banana", "cherry"]

for item in mylist:
    print(item, end = "\t")
```

Output

```
apple    banana    cherry
```

Example 13: Iterate through a list by index.

```
mylist = ["apple", "banana", "cherry"]

for i in range(3):
    print(mylist[i], end = "\t")
```

Output

```
apple    banana    cherry
```

10. List Methods

Python has a set of built-in methods that you can use on lists such as:

Method	*Description*
sort()	Sorts the list
count()	Returns the number of items with the specified value
append()	Adds an item at the end of the list
pop()	Removes the item at the specified position
insert()	Adds an item at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
index()	Returns the index of the first item with the specified value. If the item is not found, a ValueError exception is raised.

To sort an items in a list, you can use a method called **sort()**.

Syntax:

```
list.sort(reverse = True|False)
```

The `sort()` method sorts the list ascending by default. Use `list.sort(reverse = True)` will sort the list descending. This does not work if the list is storing data of different types in the same list.

Example 14: Use **sort()** method to sort the list in ascending order.

```
car_list = ["Ford", "Camry", "Prius"]

car_list.sort()
print(car_list)
```

Output:

```
['Camry', 'Ford', 'Prius']
```

Example 15: Use **sort()** method to sort the list in descending order.

```
car_list = ["Ford", "Camry", "Prius"]

car_list.sort(reverse = True)
print(car_list)
```

Output:

```
['Prius', 'Ford', 'Camry']
```

Example 16: Use `count()` method to count the number of items with the specified value

```
list1 = [2, 3, 4, 3, 10, 3, 5, 6, 3]
item_count = list1.count(3)

print("There are", item_count, "of number 3 in the list.")
```

Output:

```
There are 4 of number 3 in the list.
```

Example 17: Use `append()` method to add an item to the end of the list.

```
list1 = []

list1.append(1)
list1.append(2)

print(list1)
```

Output:

```
[1, 2]
```

Example 18: `insert()` to insert a value at a specific index in the list:

```
list1 = ["apple", "banana", "cherry"]

list1.insert(1, "orange")

print(list1)
```

Output:

```
['apple', 'orange', 'banana', 'cherry']
```

Example 19: use pop() method to remove the last item from a list.

```
list1 = ["apple", "banana", "cherry"]

list1.pop()

print(list1)
```

Output:

```
["apple", "banana"]
```

If you specify the index, the pop() method removes the item at the index.

Example 20: In this example, we will use pop() method to remove the second item:

```
list1 = ["apple", "banana", "cherry"]

list1.pop(1)

print(list1)
```

Output

```
["apple", "cherry"]
```

Example 21: pop() also returns the value it pops:

```
list1 = ["apple", "banana", "cherry"]

x = list1.pop(1)

print(x)
```

Output

```
Banana
```

Example 22: The remove() method to remove the first matching item (which is passed as an argument) from the list.

```
animals = ["cat", "dog", "rabbit", "pig", "rabbit"]

animals.remove("rabbit")

print("Updated animals list: ", animals)
```

Output

```
Updated animals list: ['cat', 'dog', 'pig', 'rabbit']
```

Example 23: User `reverse()` method to reverse a list.

```
list1 = [1, 20, 3, 40, 5]

list1.reverse()

print (list1)
```

Output

```
[5, 40, 3, 20, 1]
```

Example 24: User `index()` method to find the position of an item in a list.

```
list1 = [1, 20, 3, 20, 5]

i = list1.index(20)

print(i)
```

Output:

```
1
```

11. Functions for Lists

Several functions that can be used with lists:

Function	Description
`len()`	returns the number of items in the list.
`max()`	returns the item with the greatest values in the list.
`min()`	returns the item with the lowest values in the list.
`sum()`	returns the sum of all item in the list.

Example 25: Use functions on a list.

```
list1 = [1, 2, 3, 4, 1, 5]

print(len(list1))
print(max(list1))
print(min(list1))
print(sum(list1))
```

Output:

```
6
5
1
16
```

12. `list()` Constructor

The `list()` constructor returns a list. Let's see some examples of how `list()` constructor can be used.

Example 26: Get multiple integers from the user input and store them in a list

```
mylist = list(eval(input("Enter integers separated by commas: ")))

print(mylist)
```

Output:

```
Enter integers separated by commas: 1, 3, 5, 7, 9
[1, 3, 5, 7, 9]
```

Here is how the line 1 in the program above works:

- `input("Enter integers separated by commas: ")` prompts the user to input a string of integers, separated by commas (e.g., `1, 3, 5, 7, 9`).
- `eval()` converts a input string `"1, 3, 5, 7, 9"` to a tuple: `(1, 3, 5, 7, 9)`.
- `list()` converts the evaluated tuple `(1, 3, 5, 7, 9)` into a list: `[1, 3, 5, 7, 9]`.
- The resulting list `[1, 3, 5, 7, 9]` is assigned to the variable `mylist`.

Example 27: Get multiple integers from the user input and store them in a list

```
mylist = list(eval(input("Enter numbers separated by spaces: ").replace(" ", ",")))

print(mylist)
```

Output:

```
Enter numbers separated by spaces: 1 3 5 7 9
[1, 3, 5, 7, 9]
```

Here is how the line 1 in the program above works:

- `input("Enter integers separated by space: ")` prompts the user to input a string of integers, separated by commas (e.g., `1, 3, 5, 7, 9`).
- `replace(" ", ",")` replaces all spaces in the input string with commas. `"1 3 5 7 9"` → `"1,3,5,7,9"`. Now the input is formatted as a comma-separated string, which `eval()` function can interpret.
- `eval()` converts a input string `"1,3,5,7,9"` to a tuple: `(1, 3, 5, 7, 9)`.
- `list()` converts the evaluated tuple `(1, 3, 5, 7, 9)` into a list: `[1, 3, 5, 7, 9]`.
- The resulting list `[1, 3, 5, 7, 9]` is assigned to the variable `mylist`.

Example 28: Using the **list()** constructor and a build-in function called **range()** to create sequential numerical list items.

```
list1 = list(range(5))          # list1 = [0, 1, 2, 3, 4]
list2 = list(range(10, 15))     # list2 = [10, 11, 12, 13, 14]
list3 = list(range(1, 10, 3))   # list3 = [1, 4, 7]

print(list1)
print(list2)
print(list3)
```

Output:

```
[0, 1, 2, 3, 4]
[10, 11, 12, 13, 14]
[1, 4, 7]
```

Example 29: To get a duplicate copy of list1 into list2, you can use: list2 = [] + list1

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

print(list2)
```

Output:

```
[0, 1, 2, 3, 4]
```

Example 30: Combine multiple lists together

```
list1 = [1, 2, 3, 4]
list2 = [10, 11, 12, 13, 14]

list3 = list1 + list2

print(list3)
```

Output:

```
[ 0, 1, 2, 3, 4, 10, 11,12, 13, 14]
```

13. List Comprehension

In Python, you can turn for loops into one-liners by using comprehensions.

The list comprehension goes with the syntax:

```
output_list = [expression for var in input_list if condition]
```

where the if condition part is optional.

For example, let's create a new list of numbers from a list of numbers by leaving out all the negative numbers.

Example 31: Copy items which are positive from a list to another list.

```
old_list = [-2, 4, -6, 8, -10, 12]
new_list = []

for num in old_list:
    if num > 0:
        new_list.append(num)

print(new_list)
```

Output:

```
[ 4, 8, 12]
```

Now, we will make this for loop shorter by using a list comprehension:

Example 32: Copy items which are positive from a list to another list using list comprehension.

```
old_list = [-2, 4, -6, 8, -10, 12]

new_list = [num for num in old_list if num > 0]

print(new_list)
```

Output:

```
[ 4, 8, 12]
```

14. Multi-dimensional Lists

A multi-dimensional list is a list that contains other lists as its items. It is considered as nested list. Data in a table or a matrix can be stored in a two-dimensional list (2D list).

For example, the following table, which provides the distances between cities, can be stored in a list named distances.

Distance Table (in miles)

	Chicago	Boston	New York	Atlanta	Miami
Chicago	0	983	787	714	1,375
Boston	983	0	214	1,102	1,505
New York	787	214	0	888	1,549
Atlanta	714	1,102	888	0	661
Miami	1,375	1,505	1,549	661	0

The data in the table above can be stored in a array named distances as below:

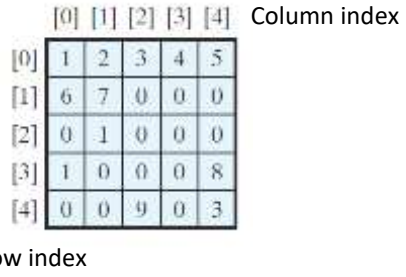
```
distances = [[0, 983, 787, 714, 1375, 967, 1087],
             [983, 0, 214, 1102, 1505, 1723, 1842],
             [787, 214, 0, 888, 1549, 1548, 1627],
             [714, 1102, 888, 0, 661, 781, 810],
             [1375, 1505, 1549, 661, 0, 1426, 1187]]
```

14.1 Accessing Items in 2D Lists

You can think of a two-dimensional list as a table that consists of rows and columns. The values in a two-dimensional list can be accessed through row and column indexes.

Example 33: Create A two-dimensional list named matrix and access its items.

```
matrix = [[1, 2, 3, 4, 5],
          [6, 7, 0, 0, 0],
          [0, 1, 0, 0, 0],
          [1, 0, 0, 0, 8],
          [0, 0, 9, 0, 3]]

print(matrix[0][0])
print(matrix[4][4])
print(matrix[0])
print(matrix[4])
print(matrix)
```



Output:

```
1
3
[1, 2, 3, 4, 5]
[0, 0, 9, 0, 3]
[[1, 2, 3, 4, 5], [6, 7, 0, 0, 0], [0, 1, 0, 0, 0], [1, 0, 0, 0, 8], [0, 0, 9, 0, 3]]
```

14.2 Initializing 2D Lists

Example 34: Initialize a 2D list with input values

```
matrix = [] # Create an empty list

num_rows = eval(input("Enter the number of rows: "))

for row in range(num_rows):
    row_items = list(eval(input("Enter items separated by a comma: ")))
    matrix.append(row_items) # Add a new row

print(matrix)
```

Output:

```
Enter the number of rows: 3
Enter items separated by a comma: 1, 2, 3
Enter items separated by a comma: 4, 5, 6
Enter items separated by a comma: 7, 8, 9
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

13.3 Iterating 2D Lists

In order to iterate through these nested lists, we must use a nested loop. The outer loop iterates through the row number, the inner loop runs through the items inside of a row.

Example 35: Display each item in a 2D list by using a loop.

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

for row in matrix:
    for value in row:
        print(value, end = " ")
    print()
```

Output:

```
1 2 3
4 5 6
7 8 9
```

Example 36: Display each item in a 2D list by using a loop by index.

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

for row in range(len(matrix)):
    for column in range(len(matrix[row])):
        print(matrix[row][column], end = " ")
    print()
```

Output:

```
1 2 3
4 5 6
7 8 9
```

Example 37: Display each item in a 2D list. In this example, the inner list is unpacked into three variables (a, b, c).

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

for a, b, c in matrix:
    print(a, b, c)
```

Output:

```
1 2 3
4 5 6
7 8 9
```

You can apply the `sort`() method to sort a two-dimensional list. It sorts the rows on their first items. For the rows with the same first item, they are sorted on the second item. If the first and second items in the rows are the same, their third items are sorted, and so on.

Example 38: Sort a 2D list.

```
points = [[4, 2], [1, 7], [4, 5], [1, 2], [1, 1], [4, 1]]

points.sort()

print(points)
```

Output:

```
[[1, 1], [1, 2], [1, 7], [4, 1], [4, 2], [4, 5]]
```

14. Three-dimensional Lists

A two-dimensional list consists of a list of one-dimensional lists and a three-dimensional list consists of a list of two-dimensional lists, and so on. You can create n-dimensional lists for any integer n.

Example 39: In this example, we use a three-dimensional list to store exam scores for a class of three students with four exams, and each exam has two parts (multiple-choice and essay).

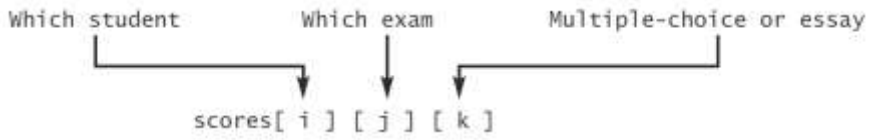
```
scores = [[[11.5, 20.5], [11.0, 22.5], [15, 33.5], [13, 21.5]],
          [[4.5, 21.5], [11.0, 22.5], [15, 34.5], [12, 20.5]],
          [[6.5, 30.5], [11.4, 11.5], [11, 33.5], [11, 23.5]]]

print(scores[0])
print(scores[0][1])
print(scores[0][1][0])
print(scores[0][1][1])
print(len(scores))
print(len(scores[0]))
print(len(scores[0][0]))
```

Output:

```
[[11.5, 20.5], [11.0, 22.5], [15, 33.5], [13, 21.5]]
[11.0, 22.5]
11.0
22.5
3
4
2
```

`scores[0][1][0]` refers to the multiple-choice score for the first student's second exam, which is `11.0`. `scores[0][1][1]` refers to the essay score for the first student's second exam, which is `22.5`.

The following figure depicts the meaning of the values in the list.



Exercises

Write the following programs:

1. Ask the user to enter the names of three people they want to invite to a party and store them in a list. After they have entered all three names, ask them if they want to add another. If they do, allow them to add more names until they answer "no". When they answer "no", display how many people they have invited to the party. Here is a sample run:

```
Enter the names of three people: Sok Rasy, Chim Narith, Lim Nary    Enter
Would you like to invite another person? yes    Enter
Enter the person's name: Ly Narin               Enter
Would you like to invite another person? yes    Enter
Enter the person's name: Bun Sophat             Enter
Would you like to invite another person? no     Enter

You have invited 5 people to the party. They are:
Sok Rasy
Chim Narith
Lim Nary
Ly Narin
Bun Sophat
```

2. (Assign grades) Write a program that asks the user to enter a list of scores separated by a comma in one line, and then assigns grades based on the following scheme:
   The grade is A if score is >= 85.
   The grade is B if 85 > score >= 70.
   The grade is C if 70 > score >= 60.
   The grade is F otherwise.

   Here is the sample run:

```
Enter scores: 70, 65, 92, 45    Enter

Score is 70 and Grade is B
Score is 65 and Grade is C
Score is 92 and Grade is A
Score is 45 and Grade is F
```

3. Write a program that reads some integers between 1 and 100 separated by a comma in one line, and counts the occurrences of each. Here is the sample run:

```
Enter integers between 1 and 100: 2, 5, 6, 5, 4, 3, 23, 43, 2    Enter

2 occurs 2 times
3 occurs 1 time
4 occurs 1 time
..
43 occurs 1 time
```

   Note that if a number occurs more than one time, the plural word "times" is used in the output.

4. Write a program that reads in numbers separated by a comma in one line, and displays distinct numbers (i.e., if a number appears multiple times, it is displayed only once). (Hint: Read all the numbers and store them in list1. Create a new list list2. Add a number in list1 to list2. If the number is already in the list, ignore it.) Here is the sample run:

```
Enter ten numbers:  1, 2, 3, 2, 1, 6, 3, 4, 5, 2    Enter
The distinct numbers are: 1 2 3 6 4 5
```

5. (Game: locker puzzle) There are 50 lockers and 50 players. When the game starts, all the lockers are closed. As the players enter the building, the first player (P1) has to open every locker. Then the second player (P2) begins with the second locker (L2), and has to close every other locker (L2 to L50). Player P3 begins with the third locker (L3) and has to change (closes the locker if it was open, and open it if it was closed) every third locker (L3, L6, …). Player P4 begins with locker L4 and changes every fourth locker (L4, L8, …), and so on, until Player P50 changes L50.

   Write a program to display all open locker numbers and the total number of open lockers when the game ends. Hint: Use a list of 50 boolean elements, each of which indicates whether a locker is open (True) or closed (False).

6. (Guess the capitals) Write a program that stores five countries and their capital cities in a 2D list.

Cambodia	Phnom Penh
China	Beijing
Japan	Tokyo
India	Delhi
Malaysia	Kuala Lumpur

   The program repeatedly asks the user to enter a capital city for a country. Upon receiving the user input, the program reports whether the answer is correct. The user's answer is **not case sensitive**. The program asks the user to answer all the countries' capital cities and displays the total correct count.

   Here is a sample run:

```
What is the capital city of Cambodia?  Phnom Pehn   Enter
Your answer is correct
What is the capital city of China?  Tokyo Enter
The correct answer should be Beijing
What is the capital city of Japan?  …
…
The correct count is 2.
```