

## Chapter 6: Python Lists Oh My!

### Contents

Chapter 6: Python Lists Oh My!.....	1
DRY.....	2
Overview.....	2
Why Use Lists? .....	2
List Basics .....	2
Tutorial 6.1: A Simple List.....	3
Creating Lists .....	3
Tutorial 6.2: Traversing a List with a Loop.....	3
List Mutability and Indexing.....	4
Assignment 6.1: Lists and Loops .....	6
List Functions, Methods and Membership.....	7
Tutorial 6.3: Working with Lists .....	11
Tutorial 6.4: The Slice Operator ([:]) .....	12
Assignment 6.2: List Slicing.....	13
Tutorial 6.5: Input a Contact List .....	13
Tutorial 6.6: Irene’s Interactive Shopping List.....	15
Tutorial 6.7: Parallel Lists.....	20
Lists and the Random Module .....	21
Tutorial 6.8: Random Events .....	22
Assignment 6.3: Square Root Calculator .....	23
Glossary.....	23
Assignment Submission.....	24



**Red light: No AI**

Time required: 120 minutes

# DRY

Don't Repeat Yourself

## Overview

Lists and tuples are sequence types in Python. They store ordered collections of items. The key difference:

- **Lists** are **mutable** (you can change them).
- **Tuples** are **immutable** (they cannot be changed after creation).

## Why Use Lists?

When working with large datasets—like 30 student test scores—managing each value individually is inefficient.

Instead, group values in a single data structure like a list.

## List Basics

### Definition

A **list** is an ordered collection of elements.

Each element can be of any data type—integers, strings, floats, or even other lists.

### Syntax

```
# Creating a list of characters
a_list = ["p", "y", "t", "h", "o", "n"]
```

### Indexing

Python lists use **zero-based indexing**:

Index	0	1	2	3	4	5
Element	"p"	"y"	"t"	"h"	"o"	"n"
Negative Index	-6	-5	-4	-3	-2	-1

## Tutorial 6.1: A Simple List

The program below shows how to create a list and access items in a list.

```
1  # simple_list.py
2  # Create a simple list with three integers
3  my_list = [1, 2, 3]
4
5  # Print the list
6  print(my_list)
7
8  # Access the first element of the list
9  print(my_list[0])
10
11 # Access the second element of the list
12 print(my_list[1])
13
14 # Access the last element of the list
15 print(my_list[-1])
```

## Creating Lists

There are several ways to create a new list; the simplest is to enclose the elements in square brackets.

```
empty_list = []
integer_list = [1, 2, 3, 10]
string_list = ["crunchy frog", "ram bladder", "lark vomit"]
```

The elements of a list don't have to be the same type. The following list contains a string, a float, an integer, and another list:

```
monty_list = ["spam", 2.0, 5, [10, 20]]
```

## Tutorial 6.2: Traversing a List with a Loop

A for loop allows you to process each item in a list automatically. This is helpful for displaying data in a clean format or performing calculations.

```

1  """
2      Name: access_a_list.py
3      Author:
4      Created:
5      Purpose: Print list with for loop, access by user input
6  """
7  # access_a_list.py
8  # Create a list of strings
9  cheeses = ["1. Cheddar", "2. Swiss", "3. Feta"]
10
11 # Print the list using a for loop
12 # This iterates through the list one element at a time
13 for cheese in cheeses:
14     print(cheese)
15
16 # Prompt the user for a choice
17 choice = int(input("Enter the number of the cheese you wish: "))
18
19 # Print the user choice
20 # Subtract one to align the user choice with the list index
21 print(cheeses[choice - 1])

```

Example

run:

```

1. Cheddar
2. Swiss
3. Feta
Enter the number of the cheese you wish: 3
3. Feta

```

## List Mutability and Indexing

In Python, **lists are mutable**, meaning their contents can be changed after creation.

### You Can:

- Reassign individual elements
- Change the order of items
- Use indices to read or update values

### Example: Reassigning a List Element

```
numbers = [17, 123]
numbers[1] = 5
print(numbers)
Output: [17, 5]
```

The number 1 element of the numbers list, which used to be 123, is now 5.

## Lists as Mappings

Think of a list as a **mapping** from an index to a value.

Each index points to an item in the list.

### Key Points:

- Indexing starts at 0
- Negative indices count from the end: -1 is the last item
- Accessing an index that doesn't exist causes an `IndexError`
- You can use **any integer expression** as an index

Values from a list can be used just like a variable.

```
# print_message.py
dogs = ["Affenpinscher", "Afghan Hound", "Akita"]
message = f"My dog is an {dogs[0]}"
print(message)
```

Example run:

```
My dog is an Affenpinscher
```

You can think of a list as a relationship between indices and elements. This relationship is called a mapping; each index “maps to” one of the elements.

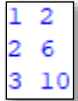
- Any integer expression can be used as an index.
- If you try to read or write an element that does not exist, you get an `IndexError`.
- If an index has a negative value, it counts backward from the end of the list.

## Updating All Elements in a List

You can loop through the list using **range()** and **len()** to update values by index.

```
# range_len.py
numbers = [1, 3, 5]
for i in range(len(numbers)):
    # Update element in list
    numbers[i] = numbers[i] * 2
    print(f"{i + 1} {numbers[i]}")
```

Example run:



```
1 2
2 6
3 10
```

Explanation:

- `len(numbers)` returns 3
- `range(3)` gives [0, 1, 2]
- Each element is updated using its index

## Assignment 6.1: Lists and Loops

### Learning Objectives

- Understand and apply list indexing and mutability
- Use for and while loops to process and manipulate lists
- Identify and work with nested lists

### Instructions

Complete both tasks below. Submit your **list\_loops.py** file with clear comments for each section.

#### Task 1: List Creation and Indexing

##### Instructions:

1. Create a list named `tools` with at least **5 string values** (e.g., "hammer", "screwdriver", etc.).
2. Print the **first**, **third**, and **last** items using indexing.
3. Ask the user to enter an index. Print the item at that index.

## Task 2: Modify and Traverse with for Loop

### Instructions:

1. Create a list named numbers with values [2, 4, 6, 8, 10].
2. Use a for loop and the range(len(...)) method to multiply each item by **3**.
3. Print the **index and new value** on each line.

Example run:

```
First tool: hammer
Third tool: screwdriver
Last tool: saw
Enter an index (0 to 4): 3
You selected: pliers

1: 6
2: 12
3: 18
4: 24
5: 30
```

## List Functions, Methods and Membership

### 1. Built-in List Functions

These are general-purpose functions that work on lists (and other sequences):

Function	Description
len()	Returns the number of items in the list
sum()	Returns the sum of numeric items
min()	Returns the smallest item
max()	Returns the largest item
sorted()	Returns a new sorted list (does not modify original)

Example:

```

nums = [3, 41, 12, 9, 74, 15]
print(len(nums))          # 6
print(max(nums))          # 74
print(min(nums))          # 3
print(sum(nums))          # 154
print(sum(nums)/len(nums)) # 25.67
print(sorted(nums))       # [3, 9, 12, 15, 41, 74]

```

## 2. Building a List from User Input

Use a while loop to collect input and store values in a list.

```

num_list = []
while True:
    num = input("Enter a number (Enter to quit): ")
    if num == "":
        break
    num_list.append(float(num))

average = sum(num_list) / len(num_list)
print(f"Average: {average}")

```

## 3. Membership Operators: in and not in

Operator	Description
in	Checks if a value exists in a list
not in	Checks if a value does not exist

Example:

```

# Using 'in'
letter = input("Enter a character: ")
if letter in "Norwegian Wood":
    print(f"{letter} is in the string")
else:
    print("I don't need that letter")

# Using 'not in'
activity = input("What game would you like to play? ")
if "golf" not in activity.lower() and "baseball" not in activity.lower():
    print("But I want to play golf or baseball")

```



#### 4. List Methods

These are specific to list objects and use the dot (.) notation.

Method	Description
<code>append(x)</code>	Adds x to the end of the list
<code>insert(p, x)</code>	Inserts x at index p
<code>sort()</code>	Sorts the list in-place
<code>reverse()</code>	Reverses the list in-place
<code>count(x)</code>	Counts occurrences of x
<code>index(x)</code>	Returns the index of the first occurrence of x
<code>remove(x)</code>	Removes the first occurrence of x
<code>pop(p)</code>	Removes and returns item at index p (default: last)
<code>clear()</code>	Removes all elements
<code>extend(list2)</code>	Adds elements from list2 to the current list

## Examples:

```
# append()
t = ["a", "b", "c"]
t.append("d")
print(t)  # ['a', 'b', 'c', 'd']

# insert()
t.insert(1, "x")
print(t)  # ['a', 'x', 'b', 'c', 'd']

# sort()
t.sort()
print(t)  # ['a', 'b', 'c', 'd', 'x']

# reverse()
t.reverse()
print(t)  # ['x', 'd', 'c', 'b', 'a']

# count() and index()
fruitlist = ["apple", "banana", "cherry", "banana", "durian"]
print(fruitlist.count("banana"))  # 2
print(fruitlist.index("banana"))  # 1

# remove()
fruitlist.remove("banana")
print(fruitlist)  # ['apple', 'cherry', 'banana', 'durian']

# pop()
popped = fruitlist.pop(2)
print(popped)      # 'banana'
print(fruitlist)  # ['apple', 'cherry', 'durian']

# extend()
t1 = ["a", "b"]
t2 = ["c", "d"]
t1.extend(t2)
print(t1)  # ['a', 'b', 'c', 'd']
```

## Tutorial 6.3: Working with Lists

The following program manipulates a list using various methods and functions.

Name the file: **list\_manipulation.py**

```
1  """
2      Name: list_manipulation.py
3      Author:
4      Created:
5      Purpose: Demonstrate various list methods and functions
6  """
7
8
9  def main():
10     # Create and populate the list
11     list = [4, -4, 7, 9, 10, 100]
12     print(f"Original list: {list}")
13
14     # Get the length of the list
15     print(f"Length of list: {len(list)}")
16
17     # Sort the list
18     list.sort()
19     print(f"Sorted List: {list}")
20
21     # Reverse the list
22     list.reverse()
23     print(f"Reversed list: {list}")
24
25     # Remove the element value 9 from the list
26     list.remove(9)
27     print(f"Removed 9 from list: {list}")
28
29     # Add 50 to the end of the list
30     list.append(50)
31     print(f"Appended 50 to end of list: {list}")
32
33     # Sort the list
34     list.sort()
35     print(f"Sorted List: {list}")
36
37
38  """
39      If a standalone program, call the main function
40      Else, use as a module
41  """
42  if __name__ == "__main__":
43      main()
```

Example run:

```
Original list: [4, -4, 7, 9, 10, 100]
Length of list: 6
Sorted List: [-4, 4, 7, 9, 10, 100]
Reversed list: [100, 10, 9, 7, 4, -4]
Removed 9 from list: [100, 10, 7, 4, -4]
Appended 50 to end of list: [100, 10, 7, 4, -4, 50]
Sorted List: [-4, 4, 7, 10, 50, 100]
```

## Tutorial 6.4: The Slice Operator ([:])

Slicing allows you to extract a **subsection of a list** without using a loop.

The syntax is:

```
list_name[start : end]
```

- start is the index to **begin the slice (inclusive)**
- end is the index to **end the slice (exclusive)**

Create a Python program named **slice\_lists.py**

```
1  my_list = ["a", "b", "c", "d", "e", "f"]
2
3  new_list = my_list[1:3]
4  print(new_list)
5
6  new_list = my_list[:4]
7  print(new_list)
8
9  new_list = my_list[3:]
10 print(new_list)
11
12 new_list = my_list[:]
13 print(new_list)
```

`t[1:3]` slices the index 1 and 2 items.

`t[:4]` slices everything from the beginning of the list to index 3.

`t[3:]` slices everything after and including index 3.

`t[:]` slices the entire list and makes a copy

Example run:

```
['b', 'c']  
['a', 'b', 'c', 'd']  
['d', 'e', 'f']  
['a', 'b', 'c', 'd', 'e', 'f']
```

## Assignment 6.2: List Slicing

Create a Python program named `list_slicing.py`

Given this list:

```
letters = ["a", "b", "c", "d", "e", "f", "g"]
```

### Tasks:

1. Print the first 4 letters using slicing.
2. Print the last 3 letters using slicing.
3. Print the middle 3 letters (c, d, e).
4. Print a copy of the full list using slicing.

Example run:

```
First 4 letters: ['a', 'b', 'c', 'd']  
Last 3 letters: ['e', 'f', 'g']  
Middle 3 letters: ['c', 'd', 'e']  
Full copy of the list: ['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

## Tutorial 6.5: Input a Contact List

Let's use the `append` and `sort` functions to add items to a list. Create a program named **`list_append.py`**

```

1  """
2      Name: list_append.py
3      Author:
4      Created:
5      Purpose: Create a list from user input
6  """
7
8
9  def main():
10     # Create an empty list.
11     friends = []
12
13     # Add names to the list.
14     while True:
15         # Get a name from the user.
16         name = input("Enter a name (Press Enter to quit): ")
17
18         # Exit the loop if the user presses enter
19         if name == "":
20             break
21
22         # Append the name to the list.
23         friends.append(name)
24
25         # Print a blank line
26         print()
27
28         # Sort the names in the list
29         friends.sort()
30
31         # Inform the user you will print the list
32         print("Here is your sorted contact list:")
33
34         # Iterate through and print your list
35         for name in friends:
36             print(name)
37
38
39 # If a standalone program, call the main function
40 # Else, use as a module
41 if __name__ == "__main__":
42     main()

```

Example run:

```
Enter a name (Press Enter to quit): Alvin
Enter a name (Press Enter to quit): Theadore
Enter a name (Press Enter to quit): Brian
Enter a name (Press Enter to quit):

Here is your sorted contact list:
Alvin
Brian
Theadore
```

## Tutorial 6.6: Irene's Interactive Shopping List

The following program demonstrates how to create and manipulate a list with user input. This list also demonstrates how to Write Python Beautiful Code with PEP8.

Create a Python program called **shopping\_list.py**.

```

1  """
2      Name: shopping_list.py
3      Author:
4      Created:
5      Purpose: An interactive shopping list in Python
6  """
7
8
9  # ----- MAIN -----#
10 def main():
11     """Create an interactive shopping list.
12
13     Create a blank list. Create an interactive menu for user input.
14     The user can choose from different list methods.
15     """
16     # Boolean variable to determine if we should continue or leave the loop
17     running = True
18     print(" Irene's Interactive Shopping List")
19     # Create blank list outside the loop
20     shoplist = []
21     while running:
22         print()
23         print(" (A)dd item")
24         print(" (R)emove item")
25         print(" (S)ort list")
26         print(" (P)rint list")
27         print(" (Enter) Exit")
28
29         # Convert input to lower case for easier comparison
30         menu_choice = input(">> ").lower()
31
32         # Determine user user choice
33         if menu_choice == "":
34             running = False
35         elif menu_choice == "a":
36             add_item(shoplist)
37             print_list(shoplist)
38         elif menu_choice == "s":
39             sort_list(shoplist)
40             print_list(shoplist)
41         elif menu_choice == "p":
42             print_list(shoplist)
43         elif menu_choice == "r":
44             print_list(shoplist)
45             remove_item(shoplist)
46             print_list(shoplist)

```



```

49 # ----- ADD ITEM -----#
50 def add_item(shoplist):
51     """Append user input item to list."""
52     item = input(f" Enter an item: ")
53     # .append() adds the item to the end of the list in place
54     shoplist.append(item)
55
56
57 # ----- PRINT LIST -----#
58 def print_list(shoplist):
59     """Iterate through and print the shopping list.
60
61     Iterate through list one at a time to print each item.
62     Number each item to more easily select item
63     to delete or modify.
64     """
65     print(" Current Shopping List")
66     count = 1
67     # Iterate through each item in the list
68     # Add 1 to count each time around
69     for item in shoplist:
70         print(f"{count} {item}")
71         count += 1
72
73
74 # ----- SORT LIST -----#
75 def sort_list(shoplist):
76     """Sort the list."""
77     # sort() method sorts the list in place
78     shoplist.sort()
79     print(f" Sorted shopping list")
80

```

```

74 # ----- SORT LIST -----#
75 def sort_list(shoplist):
76     """Sort the list."""
77     # sort() method sorts the list in place
78     shoplist.sort()
79     print(f" Sorted shopping list")
80
81
82 # ----- REMOVE ITEM -----#
83 def remove_item(shoplist):
84     """Remove item from list."""
85     # User inputs item number of the list to select the item to delete
86     item_num = int(input(" Remove which item number: "))
87
88     # item_num is one off from the list index
89     # Subtract 1 from item_num to align with the list index
90     delete_item = shoplist[item_num - 1]
91
92     # Let the user know which item is being deleted
93     print(f" Removing {delete_item}")
94
95     # Remove the item from the list in place
96     del shoplist[item_num - 1]
97
98
99 # If a standalone program, call the main function
100 # Else, use as a module
101 if __name__ == "__main__":
102     main()

```

Example run:

### Irene's Interactive Shopping List

```
(A)dd item
(R)emove item
(S)ort list
(P)rint list
(Enter) Exit
>> a
Enter an item: Apple
Current Shopping List
1 Apple

(A)dd item
(R)emove item
(S)ort list
(P)rint list
(Enter) Exit
>> a
Enter an item: Bread
Current Shopping List
1 Apple
2 Bread

(A)dd item
(R)emove item
(S)ort list
(P)rint list
(Enter) Exit
>> a
Enter an item: Asparagus
Current Shopping List
1 Apple
2 Bread
3 Asparagus

(A)dd item
(R)emove item
(S)ort list
(P)rint list
(Enter) Exit
>> s
Sorted shopping list
Current Shopping List
1 Apple
2 Asparagus
3 Bread
```

```

(A)dd item
(R)emove item
(S)ort list
(P)rint list
(Enter) Exit
>> r
Current Shopping List
1 Apple
2 Asparagus
3 Bread
Remove which item: 2
Removing Asparagus
Current Shopping List
1 Apple
2 Bread

(A)dd item
(R)emove item
(S)ort list
(P)rint list
(Enter) Exit
>>

```

## Tutorial 6.7: Parallel Lists

Parallel lists are multiple lists where the elements at the **same index** across each list are related.

This approach helps **reduce the number of separate variables** and keeps related data organized.

### Why Use Parallel Lists?

- Instead of creating separate variables for related data (e.g., name1, score1, name2, score2),
- Store related information in **multiple lists** indexed consistently.

```

1  """
2      Name: parallel_lists.py
3      Author:
4      Created:
5      Purpose: Access items in parallel lists by index
6  """
7  # Parallel lists of names and ages
8  names = ["James", "Elli", "MrHamsho", "Adem", "Aida"]
9  ages = [24, 25, 21, 48, 17]
10
11 print("Access a single element from each parallel list")
12 print(f"{names[4]} is {ages[4]}")
13
14 print("Loop through all elements in each parallel list")
15 # Counter to iterate through the ages list
16 i = 0
17 for name in names:
18     print(f"{name} is {ages[i]}")
19     # Increment the counter for the next item in ages
20     i += 1

```

Example run:

```

Access a single element from each parallel list
Aida is 17
Loop through all elements in each parallel list
James is 24
Elli is 25
MrHamsho is 21
Adem is 48
Aida is 17

```

## Lists and the Random Module

The Python **random** module provides useful functions for working with lists.

Function	Description
<code>choice(L)</code>	picks a random item from L
<code>sample(L,n)</code>	picks a group of n random items from L
<code>shuffle(L)</code>	Shuffles the items of L

### Choice Function

- Selects a **random element** from a list or sequence.
- Works with **lists**, **strings**, or any **sequence**.

```
from random import choice
names = ["Joe", "Bob", "Sue", "Sally"]
current_player = choice(names)
print(current_player)
```

This function also works with strings, picking a random character from a string. Here is an example that uses choice to fill the screen with a bunch of random characters.

```
from random import choice
s='abcdefghijklmnopqrstuvwxyz1234567890!@#$%^&*()'
for i in range(10000):
    print(choice(s), end='')
```

Choice can also pick random numbers out of a list.

```
# Import the choice function from random
from random import choice
numbers = [1, 2, 3, 4]
random_choice = choice(numbers)
print(random_choice)
```

### Use Cases

- Games (random player selection)
- Simulations (random choices)
- Random testing

## Tutorial 6.8: Random Events

This tutorial chooses a random event from a list. The advantage of using a list is that it is easy to add, remove, or modify an event without changing the code.

```
1  from random import choice
2
3  event = [
4      "A monsoon wiped out your store",
5      "You received a cash gift from Uncle Carl",
6      "The lemons were spoiled"
7  ]
8
9  # Gets random items from the list
10 random_event = choice(event)
11
12 print(random_event)
```

Example run:

```
You received a cash gift from Uncle Carl
```

## Assignment 6.3: Square Root Calculator

Write a program that generates a list of the first 10 square numbers and prints them.

### Tasks:

- Create an empty list to store square numbers.
- Use a loop to calculate and store the square of numbers from 1 to 10.
- Print the list of square numbers.

Example run:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

---

## Glossary

**aliasing** A circumstance where two or more variables refer to the same object.

**delimiter** A character or string used to indicate where a string should be split.

**element** One of the values in a list (or other sequence); also called items.

**equivalent** Having the same value.

**index** An integer value that indicates an element in a list.

**identical** Being the same object (which implies equivalence).

**list** A sequence of values.

**list traversal** Sequential accessing of each element in a list.

**nested list** A list that is an element of another list.

**object** Something a variable can refer to. An object has a type and a value.

**reference** The association between a variable and its value.

---

## Assignment Submission

1. Attach the pseudocode or create a TODO.
2. Attach all tutorials and assignments.
3. Attach screenshots showing the successful operation of each tutorial program.
4. Submit in Blackboard.