Python Chapter 6 Lists Activities

Contents

Python Chapter 6 Lists Activities 1
DRY 1
Online Tutorials 1
Tutorial 1: Traversing a List with a Loop 2
Tutorial 2: List and String Slicing 4
Tutorial 3: List Functions 6
in and not in 6
Tutorial 4: List Methods 7
append() 8
insert() 8
sort() 8
reverse() 9
index() 9
remove() 10
pop() 10
extend() 11
Assignment 1: List Slicing and Iteration 12
Assignment Submission 13

Time required: 90 minutes

DRY

Don't Repeat Yourself

Online Tutorials

Go through the following tutorials.

- LearnPython.org Lists

- Python Lists
    - Access List Items
    - Change List Items
    - Add List Items
    - Remove List Items
    - Loop Lists
    - List Comprehensions
    - Sort Lists
    - Copy Lists
    - Join Lists
    - List Methods

Tutorial 1: Traversing a List with a Loop

A List is a sequence of elements. A String is a sequence of elements. A Tuple is a sequence of elements. The major difference is that a List is mutable, it can be changed. Otherwise, each of these sequences behave in similar ways.

The most common way to iterate or traverse the elements of a list is with a for loop. The syntax is the same as for strings.

```
cheeses = ["cheddar", "colby", "swiss"]
for cheese in cheeses:
    print(cheese)
cheddar
colby
swiss
```

If you want to write or update the elements, you need the indices. A common way to do that is to combine the functions **range** and **len**:

```
# 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]}")
```

This loop traverses the list and updates each element. **len** returns the number of elements in the list. **range** returns a list of indices from 0 to n − 1, where **n** is the length of the list. Each time through the loop, **i** gets the index of the next element. The assignment statement in the body uses **i** to read the old value of the element and to assign the new value.

Create a Python program named **list_loops.py**

```
"""
    Name: list_loops.py
    Author:
    Created:
    Purpose:
"""
# Define a list of numbers: 1, 3, 5
numbers = [1, 3, 5]

# Iterate through each number in the list and print it
for number in numbers:
    print(f"{number}")

# Iterate through the indices of the list using a for loop
for i in range(len(numbers)):
    # Multiply each number in the list by 2
    numbers[i] = numbers[i] * 2

    # Print the modified numbers along with their respective indices
    print(f"{i + 1}. {numbers[i]}")
```

Example run:

```
1
3
5
1. 2
2. 6
3. 10
```

Tutorial 2: List and String Slicing

The Python slice operator, represented by the colon **:**, allows you to extract a portion of a sequence like a string, list, or tuple by specifying a start, stop, and step size within square brackets [ ].

The general syntax for slicing is: **sequence[start:stop:step]**

```
new_list = original_list[start:stop:step]
new_string = original_string[start:stop:step]
```

- **start (optional)**: The index where the slice begins (inclusive). If omitted, it starts from the beginning of the sequence (index 0).
- **stop (optional)**: The index where the slice ends (exclusive). If omitted, it goes until the end of the sequence.
- **step (optional)**: The increment between indices while slicing. If omitted, it defaults to 1.

```
t = ["a", "b", "c", "d", "e", "f"]
print(t[1:3])
# Output: ['b', 'c']

print(t[:4])
# Output: ['a', 'b', 'c', 'd']

print(t[3:])
# Output: ['d', 'e', 'f']
```

- `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.

If you omit the first index, the slice starts at the beginning. If you omit the second, the slice goes to the end. If you omit both, the slice is a copy of the whole list.

```
print(t[:])
# Output: ['a', 'b', 'c', 'd', 'e', 'f']
```

Lists and strings can be sliced in similar ways. Create a Python program named **list_slice.py**

```
"""
    Name: list_slice.py
    Author:
    Created:
    Purpose: Demonstrate Python slicing
"""

# Example 1: Slicing a range from a list
numbers = [1, 2, 3, 4, 5]
subset = numbers[1:4]
print(subset)

# Example 1: Slicing a range from a string
text = "PythonProgramming"
subset = text[0:6]
print(subset)
```

Example run:

```
[2, 3, 4]
Python
```

```
# Example 2: Slicing with step
even_numbers = numbers[::2]
print(even_numbers)

# Example 2: Slicing with step
every_second_char = text[::2]
print(every_second_char)  # Output: PtoPormig

# Example 3: Slicing from the end
last_three = numbers[-3:]
print(last_three)

# Example 4: Modifying elements
fruits = ["apple", "banana", "cherry", "date"]
fruits[1:3] = ["pear", "grape"]
print(fruits)
```

Example run:

```
[1, 3, 5]
PtoPormig
[3, 4, 5]
['apple', 'pear', 'grape', 'date']
```

Tutorial 3: List Functions

There are many built-in functions that operate on lists without writing your own loops.

Function	Description
len()	Returns the number of items in the list
sum()	Returns the sum of the items in the list
min()	Returns the minimum of the items in the list
max()	Returns the maximum of the items in the list
sorted()	Returns a sorted list

in and not in

The in operator tells you if a list or string contains something. The not in operator tells you if the list of string doesn't contain something.

Create a Python file named **list_functions.py**

```python
nums = [3, 41, 12, 9, 74, 15]

print(len(nums))

print(max(nums))

print(min(nums))

print(sum(nums))

print(sum(nums) / len(nums))

print(sorted(nums))

# The in operator tells you if your list or string contains an element
if 3 in nums:
    print("Your list contains the number 3. ")

# not in tells you if your list or string does not contain an element
if 0 not in nums:
    print("Your list has no zeroes.")
```

Example run:

```
6
74
3
154
25.666666666666668
[3, 9, 12, 15, 41, 74]
Your list contains the number 3.
Your list has no zeroes.
```

Tutorial 4: List Methods

Python provides methods that operate on lists. Methods use the **.** (dot) operator.

Method	Description
append(x)	add **x** to the end of the list
sort(x)	sort the list in place
count(x)	return the number of times **x** occurs in the list
index(x)	return the location of the first occurrence of **x**

reverse()	reverse the list in place
clear()	remove all the elements from the list
remove(x)	remove the first occurrence of **x** from the list
pop(p)	remove the item at index **p** and return its value
insert(p, x)	insert **x** at index **p** of the list

Create a Python file named **list_methods.py** Add each of these code snippets to the program.

append()

Append adds a new element to the end of a list:

```
1   my_list = ["a", "b", "c"]
2   print(my_list)
3   my_list.append("d")
4   print(f"Append: {my_list}")
```

Example run:

```
Original: ['a', 'b', 'c']
Append: ['a', 'b', 'c', 'd']
```

insert()

Insert places a new element at the designate index and moves the others aside.

```
6   my_list.insert(1, "d")
7   print(f"Insert: {my_list}")
```

Example run:

```
Original: ['a', 'b', 'c']
Append: ['a', 'b', 'c', 'd']
Insert: ['a', 'd', 'b', 'c', 'd']
```

sort()

Arranges the elements of the list from low to high:

```
9   # Sorts list in place
10  my_list.sort()
11  print(f"Sort: {my_list}")
```

Example run:

```
Original: ['a', 'b', 'c']
Append: ['a', 'b', 'c', 'd']
Insert: ['a', 'd', 'b', 'c', 'd']
Sort: ['a', 'b', 'c', 'd', 'd']
```

reverse()

The reverse() reverses the original order of the list. It doesn't sort backward alphabetically, it reverses the current order of the list,

```
12    my_list.reverse()
13    print(f"Reverse: {my_list}")
```

Example run:

```
Original: ['a', 'b', 'c']
Append: ['a', 'b', 'c', 'd']
Insert: ['a', 'd', 'b', 'c', 'd']
Sort: ['a', 'b', 'c', 'd', 'd']
Reverse: ['d', 'd', 'c', 'b', 'a']
```

index()

Returns the index of the first occurrence on the list of the element that is given to index() as argument. A runtime error is generated if the element is not found on the list.

```
15    print(f"Index: {my_list.index("b")}")
```

Example run:

```
Original: ['a', 'b', 'c']
Append: ['a', 'b', 'c', 'd']
Insert: ['a', 'd', 'b', 'c', 'd']
Sort: ['a', 'b', 'c', 'd', 'd']
Reverse: ['d', 'd', 'c', 'b', 'a']
Index: 3
```

count()

Returns an integer that indicates how often the element that is passed to it as an argument occurs in the list.

```
17    print(f"Count: {my_list.count("b")}")
```

Example run:

```
Original: ['a', 'b', 'c']
Append: ['a', 'b', 'c', 'd']
Insert: ['a', 'd', 'b', 'c', 'd']
Sort: ['a', 'b', 'c', 'd', 'd']
Reverse: ['d', 'd', 'c', 'b', 'a']
Index: 3
Count: 1
```

remove()

Removes an item with a certain value.

```
19    my_list.remove("b")
20    print(f"Remove b: {my_list}")
```

Example run:

```
Original: ['a', 'b', 'c']
Append: ['a', 'b', 'c', 'd']
Insert: ['a', 'd', 'b', 'c', 'd']
Sort: ['a', 'b', 'c', 'd', 'd']
Reverse: ['d', 'd', 'c', 'b', 'a']
Index b: 3
Count b: 1
Remove b: ['d', 'd', 'c', 'a']
```

pop()

pop removes an item from a list at a designated index and returns a value. You can assign that value to a variable. If you don't provide an index, it deletes and returns the last element.

```
24    popped = my_list.pop(1)
25    print(f"Pop {popped}: {my_list}")
```

Example run:

```
Original: ['a', 'b', 'c']
Append: ['a', 'b', 'c', 'd']
Insert: ['a', 'd', 'b', 'c', 'd']
Sort: ['a', 'b', 'c', 'd', 'd']
Reverse: ['d', 'd', 'c', 'b', 'a']
Index b: 3
Count b: 1
Remove b: ['d', 'd', 'c', 'a']
Pop d: ['d', 'c', 'a']
```

extend()

Takes a list as an argument and appends all the elements. This example leaves t2 unmodified.

```
27    my_list.extend(my_list)
28    print(f"Extend: {my_list}")
```

Example run:

```
Original: ['a', 'b', 'c']
Append: ['a', 'b', 'c', 'd']
Insert: ['a', 'd', 'b', 'c', 'd']
Sort: ['a', 'b', 'c', 'd', 'd']
Reverse: ['d', 'd', 'c', 'b', 'a']
Index b: 3
Count b: 1
Remove b: ['d', 'd', 'c', 'a']
Pop d: ['d', 'c', 'a']
Extend: ['d', 'c', 'a', 'd', 'c', 'a']
```

join()

.join() is not a list method, it is a string method. It does work very well with lists. **.join()** iterates through the list joining each string with the separator.

```
30    separator = " | "
31    my_list = separator.join(my_list)
32    print(f"Separator: {my_list}")
```

Example run:

```
Original: ['a', 'b', 'c']
Append: ['a', 'b', 'c', 'd']
Insert: ['a', 'd', 'b', 'c', 'd']
Sort: ['a', 'b', 'c', 'd', 'd']
Reverse: ['d', 'd', 'c', 'b', 'a']
Index b: 3
Count b: 1
Remove b: ['d', 'd', 'c', 'a']
Pop d: ['d', 'c', 'a']
Extend: ['d', 'c', 'a', 'd', 'c', 'a']
Separator: d | c | a | d | c | a
```

Assignment 1: List Slicing and Iteration

Objective: Understand list slicing and iteration.

1. Create a list named numbers with the numbers from 1 to 10.
2. Print the first five numbers using list slicing.
3. Print the last three numbers using list slicing.
4. Use a loop to print each number in the list.
5. Use a loop to print each number in the list that is greater than 5.

Example run:

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

Assignment Submission

1. Attach the program files.
2. Attach screenshots showing the successful operation of the programs.
3. Submit in Blackboard.