

LAB INDEX

DATA STRUCTURES USING PYTHON

|  |  |  |  |
| --- | --- | --- | --- |
| Sr.No | Practical Name | Date | Signature |
| 1. | Implement the Various functions of List under the categories:   * Create List * Access List * List Length * Updating a list * Checking operations on List * List Finding Operations * List conversion to Dictionary |  |  |
| 2. | Write programs in python to apply numeric operations on list like   * Average of elements in list * Largest element in the list * Smallest element in the list * Sum of elements in the list * Product of elements in the list * Sort the elements in the list |  |  |
| 3. | Implement Linear search in Python |  |  |
| 4. | Implement Binary search in Python |  |  |
| 5 | Sort the items in the array using Bubble sort |  |  |
| 6 | Sort the items in the array using Selection sort |  |  |
| 7. | Sort the items in the array using shell sort |  |  |
| 8. | Sort the items in the array using merge sort |  |  |
| 9. | WAP to find factorial of a number using recursion |  |  |
| 10. | WAP to print Fibonacci series using recursion |  |  |
| 11. | WAP to find sum of digit of a number using recursion |  |  |
| 12 | WAP to find certain power of a number |  |  |
| 13. | WAP to show implementation of Unordered list (Linked List ) in python |  |  |

PRACTICAL 1

### 1. Create List

You can create a list in Python using square brackets `[]` or the `list()` constructor.

# Using square brackets

my\_list = [1, 2, 3, 4, 5]

### 2. Access List

You can access list elements using indexing.

# Accessing elements

first\_element = my\_list[0] # First element (index 0)

last\_element = my\_list[-1] # Last element (negative index)

print("First Element:", first\_element)

print("Last Element:", last\_element)

### 3. List Length

You can find the length of a list using the `len()` function.

# Finding the length of a list

list\_length = len(my\_list)

print("Length of List:", list\_length)

### 4. Updating a List

You can update elements of a list using indexing.

# Updating an element

my\_list[0] = 10 # Change the first element to 10

print("Updated List:", my\_list)

### 5. Checking Operations on List

You can check for membership and other operations.

# Checking if an element exists in the list

is\_present = 5 in my\_list

is\_not\_present = 20 in my\_list

print("Is 5 in the list?", is\_present)

print("Is 20 in the list?", is\_not\_present)

### 6. List Finding Operations

You can find the index of an element or count occurrences.

# Finding an element's index

try:

index\_of\_3 = my\_list.index(3)

except ValueError:

index\_of\_3 = -1 # If not found

# Counting occurrences of an element

count\_of\_10 = my\_list.count(10)

print("Count of 10 in List:", count\_of\_10)

### 7. List Conversion to Dictionary

You can convert a list to a dictionary using a dictionary comprehension or by pairing list elements.

# Creating a list of tuples

tuple\_list = [('a', 1), ('b', 2), ('c', 3)]

# Converting to dictionary

dict\_from\_list = dict(tuple\_list)

print("Dictionary from List:", dict\_from\_list)

PRACTICAL 2

### Python Program for Numeric Operations on a List

from functools import reduce

# Sample list of numbers

numbers = [10, 20, 30, 40, 50]

# 1. Average of elements in the list

def calculate\_average(lst):

return sum(lst) / len(lst)

# 2. Largest element in the list

def find\_largest(lst):

return max(lst)

# 3. Smallest element in the list

def find\_smallest(lst):

return min(lst)

# 4. Sum of elements in the list

def calculate\_sum(lst):

return sum(lst)

# 5. Product of elements in the list

def calculate\_product(lst):

return reduce(lambda x, y: x \* y, lst)

# 6. Sort the elements in the list

def sort\_elements(lst):

return sorted(lst)

PRACTICAL 3

### Linear Search Implementation

def linear\_search(arr, target):

for index in range(len(arr)):

if arr[index] == target:

return index # Target found, return the index

return -1 # Target not found

# Example usage:

if \_\_name\_\_ == "\_\_main\_\_":

# Sample list

sample\_list = [34, 12, 5, 78, 22, 90, 11, 23]

target\_value = 78

# Performing the linear search

result = linear\_search(sample\_list, target\_value)

### Explanation of the Code:

1. \*\*Function Definition\*\*: The `linear\_search` function takes two parameters: `arr`, which is the list to be searched, and `target`, which is the value we are looking for.

2. \*\*Looping through the List\*\*: We use a `for` loop to iterate through each element in the list.

3. \*\*Comparison\*\*: Inside the loop, we check if the current element is equal to the target value.

4. \*\*Return Index\*\*: If a match is found, the function returns the index of that element.

5. \*\*Return -1\*\*: If the loop finishes and the target is not found, the function returns -1.

6. \*\*Example Usage\*\*: The `if \_\_name\_\_ == "\_\_main\_\_":` block contains a sample list and a target value to demonstrate how to use the `linear\_search` function.

PRACTICAL 4

### Binary Search Implementation

def binary\_search(arr, target):

left, right = 0, len(arr) - 1

while left <= right:

mid = (left + right) // 2 # Find the middle index

# Check if the target is in the middle

if arr[mid] == target:

return mid # Target found

# If target is greater than the middle value, ignore the left half

elif arr[mid] < target:

left = mid + 1

# If target is smaller than the middle value, ignore the right half

else:

right = mid - 1

return -1 # Target not found

# Example usage:

if \_\_name\_\_ == "\_\_main\_\_":

# Sample sorted list

sample\_list = [5, 11, 12, 22, 23, 34, 78, 90]

target\_value = 22

# Performing the binary search

result = binary\_search(sample\_list, target\_value)

# Output the result

if result != -1:

print(f"Element {target\_value} found at index: {result}.")

else:

print(f"Element {target\_value} not found in the list.")

PRACTICAL 5

Here’s a Python implementation of the Bubble Sort algorithm:

n = len(arr)

# Traverse through all array elements

for i in range(n):

# Last i elements are already in place

for j in range(0, n-i-1):

# Traverse the array from 0 to n-i-1

# Swap if the element found is greater than the next element

if arr[j] > arr[j+1]:

arr[j], arr[j+1] = arr[j+1], arr[j]

return arr

# Example usage:

if \_\_name\_\_ == "\_\_main\_\_":

sample\_list = [64, 34, 25, 12, 22, 11, 90]

print("Original list:", sample\_list)

sorted\_list = bubble\_sort(sample\_list)

print("Sorted list:", sorted\_list)

### Explanation of the Code:

1. \*\*Function Definition\*\*: The `bubble\_sort` function takes one argument, `arr`, which is the list to be sorted.

2. \*\*Outer Loop\*\*: The outer loop runs `n` times where `n` is the length of the array. This ensures that we go through the list enough times to fully sort it.

3. \*\*Inner Loop\*\*: The inner loop compares adjacent elements and swaps them if they are in the wrong order (if the first element is greater than the second).

4. \*\*Swapping Elements\*\*: When swapping elements, Python allows for a concise syntax: `arr[j], arr[j+1] = arr[j+1], arr[j]`.

5. \*\*Return the Sorted List\*\*: Once sorting is complete, the sorted list is returned.

### Running the Program

When you run this program, it prints the original unsorted list followed by the sorted list.

Example output:

Original list: [64, 34, 25, 12, 22, 11, 90]

Sorted list: [11, 12, 22, 25, 34, 64, 90]

PRACTICAL 6

### Selection Sort Implementation

Here's a Python implementation of the Selection Sort algorithm:

def selection\_sort(arr):

n = len(arr)

# Traverse through all array elements

for i in range(n):

# Find the minimum element in the unsorted portion of array

min\_index = i

for j in range(i + 1, n):

if arr[j] < arr[min\_index]:

min\_index = j # Update index of the minimum element

# Swap the found minimum element with the first element of the unsorted region

arr[i], arr[min\_index] = arr[min\_index], arr[i]

return arr

# Example usage:

if \_\_name\_\_ == "\_\_main\_\_":

sample\_list = [64, 34, 25, 12, 22, 11, 90]

print("Original list:", sample\_list)

sorted\_list = selection\_sort(sample\_list)

print("Sorted list:", sorted\_list)

PRACTICAL 7

Here’s a Python implementation of the Shell Sort algorithm:

def shell\_sort(arr):

n = len(arr)

gap = n // 2 # Initialize gap size

# Start with a big gap, then reduce the gap

while gap > 0:

# Perform a gapped insertion sort for this gap size

for i in range(gap, n):

# Save the current value to be inserted

current\_value = arr[i]

j = i

# Shift earlier gap-sorted elements up until the correct location for current\_value is found

while j >= gap and arr[j - gap] > current\_value:

arr[j] = arr[j - gap]

j -= gap

# Place current\_value in its correct location

arr[j] = current\_value

gap //= 2 # Reduce the gap size

return arr

# Example usage:

if \_\_name\_\_ == "\_\_main\_\_":

sample\_list = [64, 34, 25, 12, 22, 11, 90]

print("Original list:", sample\_list)

sorted\_list = shell\_sort(sample\_list)

print("Sorted list:", sorted\_list)

PRACTICAL 8

### Merge Sort Implementation

def merge\_sort(arr):

if len(arr) > 1:

mid = len(arr) // 2 # Finding the mid of the array

left\_half = arr[:mid] # Dividing the elements into 2 halves

right\_half = arr[mid:]

# Recursive call on each half

merge\_sort(left\_half)

merge\_sort(right\_half)

# Initial indexes for left, right and merged subarray

i = j = k = 0

# Copy data to temp arrays left\_half and right\_half

while i < len(left\_half) and j < len(right\_half):

if left\_half[i] < right\_half[j]:

arr[k] = left\_half[i]

i += 1

else:

arr[k] = right\_half[j]

j += 1

k += 1

# Checking if any element was left

while i < len(left\_half):

arr[k] = left\_half[i]

i += 1

k += 1

while j < len(right\_half):

arr[k] = right\_half[j]

j += 1

k += 1

return arr

# Example usage:

if \_\_name\_\_ == "\_\_main\_\_":

sample\_list = [64, 34, 25, 12, 22, 11, 90]

print("Original list:", sample\_list)

sorted\_list = merge\_sort(sample\_list)

print("Sorted list:", sorted\_list)

### Explanation of the Code:

1. \*\*Function Definition\*\*: The `merge\_sort` function takes an array `arr` as input.

2. \*\*Base Case\*\*: The function checks if the length of the array is greater than one. If it’s not, the list is already sorted.

3. \*\*Dividing the Array\*\*: The array is split into two halves, `left\_half` and `right\_half`.

4. \*\*Recursive Calls\*\*: The function is called recursively on both halves to sort them.

5. \*\*Merging\*\*: After the two halves are sorted, the function merges them back together:

- It initializes indices for the left half, right half, and the merged array.

- It compares elements from both halves and merges them in sorted order.

6. \*\*Copying Remaining Elements\*\*: Once one half is exhausted, any remaining elements in the other half are copied over to the result.

7. \*\*Returning the Sorted List\*\*: Finally, the function returns the sorted array.

### Running the Program

When you run the above code, it will output the original unsorted list and the sorted list.

Example output:

Original list: [64, 34, 25, 12, 22, 11, 90]

Sorted list: [11, 12, 22, 25, 34, 64, 90]

PRACTICAL 9

### Python Program to Find Factorial Using Recursion

def factorial(n):

if n < 0:

raise ValueError("Factorial is not defined for negative numbers.")

elif n == 0 or n == 1:

return 1

else:

return n \* factorial(n - 1)

# Example usage:

if \_\_name\_\_ == "\_\_main\_\_":

number = int(input("Enter a non-negative integer to find its factorial: "))

try:

result = factorial(number)

print(f"The factorial of {number} is {result}.")

except ValueError as e:

print(e)

### Explanation of the Code:

1. \*\*Function Definition\*\*: The `factorial` function is defined to calculate the factorial of a given integer `n`.

2. \*\*Base Case Check\*\*:

- If `n` is negative, a `ValueError` is raised because factorials for negative numbers are undefined.

- If `n` is either `0` or `1`, the function returns `1` since the factorial of both is defined as `1`.

3. \*\*Recursive Case\*\*: If `n` is greater than `1`, the function calls itself with the argument `n-1` and multiplies the result by `n`.

4. \*\*Input Handling\*\*: In the `\_\_main\_\_` block, the user is prompted to enter a non-negative integer, and the `factorial` function is called with that input.

5. \*\*Error Handling\*\*: A `try-except` block is used to handle potential errors, such as entering a negative integer.

PRACTICAL 10

- The first two Fibonacci numbers are 0 and 1.

- Each subsequent number is the sum of the two preceding ones.

### Python Program to Print Fibonacci Series Using Recursion

def fibonacci(n):

if n < 0:

raise ValueError("Input should be a non-negative integer.")

elif n == 0:

return 0

elif n == 1:

return 1

else:

return fibonacci(n - 1) + fibonacci(n - 2)

def print\_fibonacci\_series(count):

for i in range(count):

print(fibonacci(i), end=' ')

print() # For a new line after the series

# Example usage:

if \_\_name\_\_ == "\_\_main\_\_":

number\_of\_terms = int(input("Enter the number of terms in the Fibonacci series: "))

try:

print\_fibonacci\_series(number\_of\_terms)

except ValueError as e:

print(e)

PRACTICAL 11

### Python Program to Find the Sum of Digits Using Recursion

def sum\_of\_digits(n):

# Base case: when n is 0, the sum of digits is 0

if n == 0:

return 0

else:

# Recursive case: sum the last digit and recurse on the remaining digits

return (n % 10) + sum\_of\_digits(n // 10)

# Example usage:

if \_\_name\_\_ == "\_\_main\_\_":

number = int(input("Enter a number: "))

if number < 0:

print("Please enter a non-negative integer.")

else:

result = sum\_of\_digits(number)

print(f"The sum of the digits of {number} is: {result}")

### Explanation of the Code:

1. \*\*Function `sum\_of\_digits(n)`\*\*:

- This is a recursive function that computes the sum of the digits of the number `n`.

- \*\*Base Case\*\*: If `n` is `0`, it returns `0` because the sum of the digits of `0` is just `0`.

- \*\*Recursive Case\*\*: It adds the last digit (obtained using `n % 10`) to the sum of the digits of the number obtained by removing the last digit (using `n // 10`).

2. \*\*User Input\*\*: The program prompts the user to enter a non-negative integer. It checks if the input is negative and prompts the user to enter a valid number if it is.

3. \*\*Output\*\*: The sum of the digits is calculated and printed.

#### Example Output:

Enter a number: 12345 The sum of the digits of 12345 is: 15

PRACTICAL 12

### Python Program to Find the Power of a Number

result = 1

for \_ in range(exponent):

result \*= base

return result

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

base = float(input("Enter the base number: "))

exponent = int(input("Enter the exponent (non-negative integer): "))

if exponent < 0:

print("Please enter a non-negative integer for the exponent.")

else:

# Using the custom power function

result\_custom = power(base, exponent)

# Using the built-in power operator

result\_builtin = base \*\* exponent

print(f"{base} raised to the power of {exponent} using custom function: {result\_custom}")

print(f"{base} raised to the power of {exponent} using built-in operator: {result\_builtin}")

PRACTICAL 13

### Unordered Linked List Implementation in Python

class Node:

"""A Node class for a single element in a linked list."""

def \_\_init\_\_(self, data):

self.data = data # The data stored in the node

self.next = None # Pointer to the next node in the list

class UnorderedList:

"""An unorder linked list class to hold the nodes."""

def \_\_init\_\_(self):

self.head = None # Initially the list is empty

def is\_empty(self):

"""Check if the list is empty."""

return self.head is None

def add(self, item):

"""Add a new item to the list."""

new\_node = Node(item) # Create a new node with the item

new\_node.next = self.head # Point the next of new node to current head

self.head = new\_node # Update head to new node

def delete(self, item):

"""Delete the first occurrence of an item from the list."""

current = self.head

previous = None

while current is not None:

if current.data == item:

if previous is None: # The item to delete is the head

self.head = current.next

else:

previous.next = current.next # Bypass the current node

return

previous = current

current = current.next

print(f"{item} not found in the list.")

def search(self, item):

"""Search for an item in the list."""

current = self.head

while current is not None:

if current.data == item:

return True

current = current.next

return False

def display(self):

"""Display the entire list."""

current = self.head

if current is None:

print("The list is empty.")

return

print("Unordered List: ", end="")

while current:

print(current.data, end=" ")

current = current.next

print()

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

unordered\_list = UnorderedList()

# Add items to the list

unordered\_list.add(10)

unordered\_list.add(20)

unordered\_list.add(30)

# Display the list

unordered\_list.display()

# Search for an item

item\_to\_search = 20

found = unordered\_list.search(item\_to\_search)

print(f"Item {item\_to\_search} found: {found}")

# Delete an item

item\_to\_delete = 20

print(f"Deleting item: {item\_to\_delete}")

unordered\_list.delete(item\_to\_delete)

unordered\_list.display()

# Attempt to delete an item not in the list

unordered\_list.delete(40)

# Final display

unordered\_list.display()

```

### Explanation of the Code:

1. \*\*Node Class\*\*:

- Represents a single node in the linked list, containing the data and a pointer to the next node.

2. \*\*UnorderedList Class\*\*:

- This class manages the linked list with operations to add, delete, search, and display elements.

3. \*\*Methods\*\*:

- \*\*is\_empty()\*\*: Checks if the list is empty.

- \*\*add(item)\*\*: Adds an item to the beginning of the list.

- \*\*delete(item)\*\*: Deletes the first occurrence of an item from the list.

- \*\*search(item)\*\*: Searches for an item and returns True if found, otherwise returns False.

- \*\*display()\*\*: Prints all elements in the list.

### Example Usage:

- The program demonstrates how to create an unordered linked list, add items, search for an item, delete an item, and display the list.

You can run this program to see how the unordered linked list operates and modify it to test different scenarios!