UNIT II: Functions and Strings

Subtopic 6: Lists as Arrays and Illustrative Programs

Using Lists as Arrays in Python

In Python, lists are versatile and can be used to represent arrays. Lists are mutable, meaning their elements can be changed, and they can store items of different data types, though typically used for homogeneous data when mimicking arrays.

Example of Lists:

```
# Creating a list
numbers = [1, 2, 3, 4, 5]

# Accessing elements
print(numbers[0])  # Output: 1

# Modifying elements
numbers[1] = 10
print(numbers)  # Output: [1, 10, 3, 4, 5]

# Adding elements
numbers.append(6)
print(numbers)  # Output: [1, 10, 3, 4, 5, 6]

# Removing elements
numbers.remove(10)
print(numbers)  # Output: [1, 3, 4, 5, 6]
```

Illustrative Programs

6.1 Calculating the Square Root

Using lists to store and manipulate numbers, we can write a function to calculate the square root of each number in a list.

Example:

```
import math

def calculate_square_roots(numbers):
    return [math.sqrt(num) for num in numbers]

numbers = [1, 4, 9, 16, 25]
square_roots = calculate_square_roots(numbers)
print(square_roots)  # Output: [1.0, 2.0, 3.0, 4.0, 5.0]
```

6.2 Finding the Greatest Common Divisor (GCD)

The GCD of two numbers is the largest number that divides both of them without leaving a remainder.

Example:

```
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

numbers = [48, 64, 16]
result = gcd(numbers[0], numbers[1])
for num in numbers[2:]:
    result = gcd(result, num)

print(f"GCD of the list is: {result}")  # Output: GCD of the list is:
16
```

6.3 Exponentiation

Exponentiation of numbers in a list can be performed using a simple function.

Example:

```
def exponentiate(numbers, exponent):
    return [num ** exponent for num in numbers]

numbers = [2, 3, 4]
exponent = 3
results = exponentiate(numbers, exponent)
print(results)  # Output: [8, 27, 64]
```

6.4 Summing an Array of Numbers

Summing all elements in a list can be done using a loop or Python's built-in `sum` function.

Example:

```
def sum_array(numbers):
    return sum(numbers)

numbers = [1, 2, 3, 4, 5]
total = sum_array(numbers)
print(f"Sum of the array: {total}")  # Output: Sum of the array: 15
```

6.5 Linear Search

Linear search involves checking each element in the list to find the target value.

Example:

```
def linear_search(numbers, target):
    for index, num in enumerate(numbers):
        if num == target:
            return index
    return -1

numbers = [1, 2, 3, 4, 5]
target = 3
index = linear_search(numbers, target)
print(f"Element found at index: {index}")  # Output: Element found at
index: 2
```

6.6 Binary Search

Binary search is a more efficient way to search for an element in a sorted list by repeatedly dividing the search interval in half.

Example:

```
def binary_search(numbers, target):
    low, high = 0, len(numbers) - 1
    while low <= high:
        mid = (low + high) // 2
        if numbers[mid] == target:
            return mid
        elif numbers[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1

numbers = [1, 2, 3, 4, 5]
target = 4
index = binary_search(numbers, target)
print(f"Element found at index: {index}")  # Output: Element found at
index: 3
```

Summary

Using lists as arrays in Python allows for flexible and powerful data manipulation. By understanding and applying various list operations, you can perform complex computations and solve a wide range of practical problems. Through illustrative programs like calculating the square root, finding the GCD, exponentiation, summing arrays, and searching, you can see the practical applications of lists in real-world scenarios.

```
# Example of Lists
# Creating a list
numbers = [1, 2, 3, 4, 5]
```

```python
# Accessing elements
print(numbers[0])  # Output: 1

# Modifying elements
numbers[1] = 10
print(numbers)  # Output: [1, 10, 3, 4, 5]

# Adding elements
numbers.append(6)
print(numbers)  # Output: [1, 10, 3, 4, 5, 6]

# Removing elements
numbers.remove(10)
print(numbers)  # Output: [1, 3, 4, 5, 6]



# Calculating the Square Root
import math

def calculate_square_roots(numbers):
    return [math.sqrt(num) for num in numbers]

numbers = [1, 4, 9, 16, 25]
square_roots = calculate_square_roots(numbers)
print(square_roots)  # Output: [1.0, 2.0, 3.0, 4.0, 5.0]



# Finding the Greatest Common Divisor (GCD)
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

numbers = [48, 64, 16]
result = gcd(numbers[0], numbers[1])
for num in numbers[2:]:
    result = gcd(result, num)

print(f"GCD of the list is: {result}")  # Output: GCD of the list is:
16



# Exponentiation
def exponentiate(numbers, exponent):
    return [num ** exponent for num in numbers]

numbers = [2, 3, 4]
```

```
exponent = 3
results = exponentiate(numbers, exponent)
print(results)  # Output: [8, 27, 64]



# Summing an Array of Numbers
def sum_array(numbers):
    return sum(numbers)

numbers = [1, 2, 3, 4, 5]
total = sum_array(numbers)
print(f"Sum of the array: {total}")  # Output: Sum of the array: 15



# Linear Search
def linear_search(numbers, target):
    for index, num in enumerate(numbers):
        if num == target:
            return index
    return -1

numbers = [1, 2, 3, 4, 5]
target = 3
index = linear_search(numbers, target)
print(f"Element found at index: {index}")  # Output: Element found at
index: 2



# Binary Search
def binary_search(numbers, target):
    low, high = 0, len(numbers) - 1
    while low <= high:
        mid = (low + high) // 2
        if numbers[mid] == target:
            return mid
        elif numbers[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1

numbers = [1, 2, 3, 4, 5]
target = 4
index = binary_search(numbers, target)
print(f"Element found at index: {index}")  # Output: Element found at
index: 3
```