Seed Programming

Lists, Functions, and Searching

Lecture 06: Complete Notes

Topic: Lists, Functions, Searching, and Problem Solving

Learning Objectives

After completing this lecture, you should be able to:

- Understand why lists are needed.
- Create and access lists using indexing.
- Traverse lists using loops.
- Use the `len()` function.
- Write reusable functions.
- Solve problems using lists.
- Understand universal and existential search patterns.
- Differentiate between `print` and `return`.
- Use the `match-case` statement.
- Solve a palindrome problem using lists.

1 The Problem

In programming, we often need to work with **large amounts of data**, such as:

- Student names
- Marks
- Salaries
- Temperatures
- Product prices

Suppose we want to check whether the following words are palindromes:

- `maham`
- `ada`
- `racecar`

A **palindrome** is a word that reads the same forward and backward.

Using ordinary variables is not a practical solution because we do **not know how many characters or values** the user will enter.

This leads us to an important Python data structure called a **List**.

2 Lists

A **list** stores multiple values in a single variable.

Syntax

```
numbers = [10, 20, 30]
names = ["Ali", "Ahmed", "Sara"]
mixed = ["Ali", 20, True]
```

Lists are created using **square brackets** `[]`.

3 Indexing

Each element in a list has an **index**.

Indexing always starts from **0**.

Example:

```
names = ["Ali", "Ahmed", "Sara", "Zain"]
```

Index	Value
0	Ali
1	Ahmed
2	Sara
3	Zain

Understanding Indexing

Imagine a street with houses.

- Index **0** means **skip zero houses** and enter the first house.
- Index **1** means skip one house.
- Index **2** means skip two houses.

The number of positions skipped is called the **offset**.

4 Printing a List

```
S = [-1, -2, 20, 10, 14, 25]

for i in range(0, 6):
    print(S[i])
```

Output

```
-1
-2
20
10
14
25
```

5 Why `len()` is Better

Instead of

```
range(0, 6)
```

use

```
range(0, len(S))
```

Why?

Hardcoding the size is poor programming practice.

If the list changes, the code may:

- miss elements
- produce an IndexError

`len()` automatically adjusts to the list size.

6 Sum of Array

```
def SumArray(S):

    Sum = 0

    for i in range(0, len(S)):
        Sum += S[i]

    return Sum
```

Example:

```
S = [-1, -2, 20, 10, 14, 25]

print(SumArray(S))
```

Output

```
66
```

7 Why Functions are Useful

Functions provide:

- Code Reusability
- Easy Maintenance
- Cleaner Programs
- Flexibility

The function only cares about its **parameter**, not the original variable name.

Example:

```
SumArray(A)

SumArray(B)
```

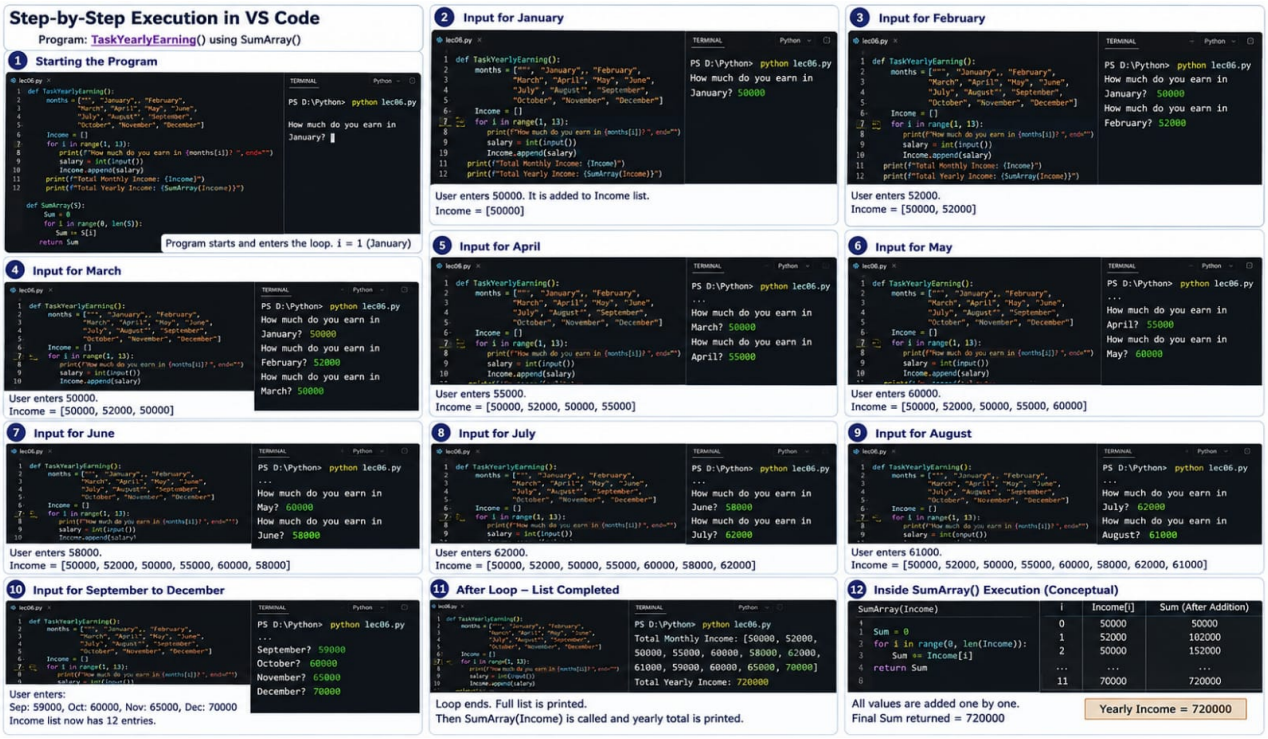
Inside the function, both are simply called `S`.

8 Practical Example — Yearly Income

We can calculate yearly income by storing monthly salaries inside a list.

```
Income = []
```

```
def TaskYearlyEarning():

    months = [
        "",
        "January","February","March","April",
        "May","June","July","August",
        "September","October","November","December"
    ]

    Income = []

    for i in range(1,13):

        print(f"How much do you earn in {months[i]}? ", end="")

        salary = int(input())

        Income.append(salary)

    print(Income)

    print(SumArray(Income))
```



Real-World Application

The total yearly income can be used for:

- Yearly tax calculation
- Budget planning
- Salary analysis
- Financial reports

9 Match-Case Statement

Introduced in **Python 3.10**.

Syntax:

```
match expression:

    case value1:
        ...

    case value2:
        ...

    case _:
        ...
```

`case _` acts like the **else** block.

Match-Case vs If-Else

Similarities Both perform decision making.

Differences

if-else	match-case
Good for complex conditions	Good for many fixed choices
Uses Boolean expressions	Matches values
Uses if, elif, else	Uses match and case

10 Palindrome

```
def isPalindrome(L):

    for i in range(0, len(L)//2):

        if L[i] != L[len(L)-i-1]:
            return False

    return True
```

Notice that we check for **inequality**.

Finding **one unequal pair** proves that the word is **not** a palindrome.

Programming often looks for **counterexamples** rather than proving equality directly.

11 Prime Number

```
def isPrime(N):

    for d in range(2, N):

        if N % d == 0:
            return False

    return True
```

Incorrect logic:

```
if N % d != 0:
    return True
```

Example:

```
N = 9
```

9 is not divisible by 2.

The incorrect code immediately returns True without checking 3.

12 Composite Number

```
def isComposite(N):

    for d in range(2, N):

        if N % d == 0:
            return True
```

```
7
8       return False
```

13 Universal vs Existential Thinking

Universal Condition

Question:

> "Is every comparison correct?"

Examples:

- Palindrome
- Prime number

Programming idea:

Search for **one failure**.

If found:

```
    return False
```

Otherwise:

```
    return True
```

Existential Condition

Question:

> "Does at least one matching case exist?"

Examples:

- Composite number
- Searching
- Finding an element

Programming idea:

Search for **one success**.

If found:

```
    return True
```

Otherwise:

```
    return False
```

14 Searching in a List

```
1 def Search(L, V):
```

```
    for i in range(0, len(L)):

        if L[i] == V:
            return True

    return False
```

Incorrect version:

```
if L[i] == V:
    return True
else:
    return False
```

Why?

Because the value may exist later in the list.

The function would stop after checking only the first element.

Key Insight

Searching follows the **existential pattern**.

We only need **one successful match**.

15 Return vs Print

This is one of the most fundamental programming concepts.

print()

Displays information on the screen.

```
print(25)
```

Output

```
25
```

return

Returns a value to the place where the function was called.

Example:

```
x = Square(5)
```

If

```
return 25
```

then Python treats it as

```
x = 25
```

The function call is replaced by its returned value.

Difference

print	return
Displays output	Sends value back
Used for displaying	Used for computation
Cannot be reused later	Returned value can be stored and reused

16 Homework

Solve the following problem on your coding platform.

Problem: Palindrome Array

Determine whether an array is a palindrome.

Example:

Example

```
Input:
[1,2,3,2,1]
Output:
True
```

Example:

Example

```
Input:
[10,20,30,40]
Output:
False
```

Hint:

- Compare first with last.
- Compare second with second-last.
- Continue until the middle.

Key Takeaways

- Lists store multiple values in one variable.
- Indexing starts from **0**.
- Use `len()` instead of hardcoding list sizes.
- Functions make code reusable.
- `append()` adds elements to a list.
- Use `match-case` for multiple fixed choices.
- Universal problems search for **one failure**.
- Existential problems search for **one success**.
- `return` sends a value back; `print` only displays it.
- Practice is essential—solve problems regularly on coding platforms.