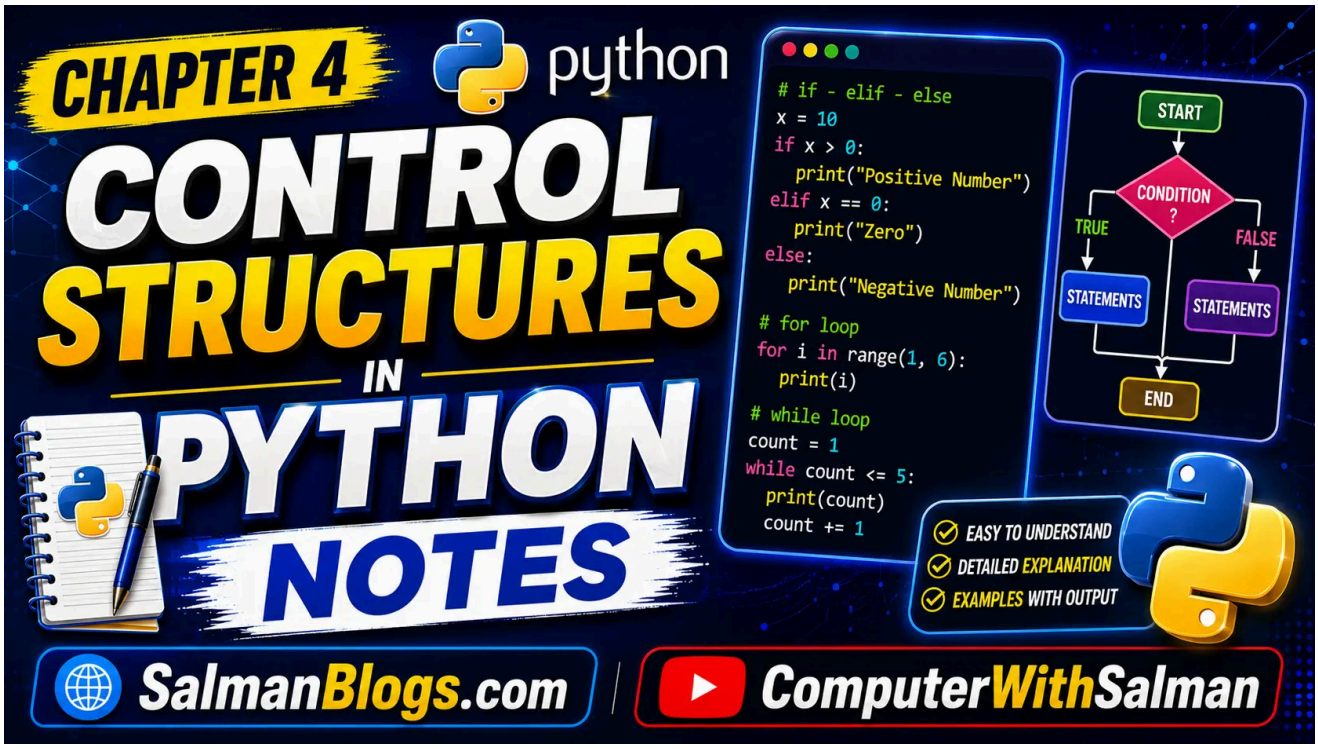
Control Structures in Python

Posted in **10 Computer Science**



These lecture notes are prepared by **Salman Ahmad**, who is both a **teacher** and a **web developer**. The purpose of this material is to make the topic clear and useful for students of Class 10 Computer Science.

Table of Contents

What is a Control Structure in Python?

A **control structure** in Python is a way of **controlling the order in which a program's instructions are executed**.

Main Types of Control Structures

There are mainly **3 types**:

1. Sequence

Code runs **from top to bottom**, one statement after another.

```
print("Step 1")
print("Step 2")
print("Step 3")
```

Output:

```
Step 1
Step 2
Step 3
```

Python executes the statements in order.

Real-life example:

This is **sequence**.

2. Selection / Decision

The program **makes a decision** based on a condition.

Python uses:

- `if`
- `if-else`
- `if-elif-else`

Example

```
age = 20

if age >= 18:
    print("You are an adult")
else:
    print("You are a minor")
```

Output:

```
You are an adult
```

3. Repetition / Iteration

Sometimes we want to **repeat the same code multiple times**.

Python uses:

- `for` loop
- `while` loop

Example using `for` loop

```
for i in range(5):
    print("Hello")
```

Output:

```
Hello
Hello
Hello
Hello
Hello
```

if Statement

An **if statement** is a control structure used to make a decision. It executes a particular block of instructions **only when a given condition is true**.

```
marks = 75

if marks >= 50:
    print("Pass")
```

Output:

```
Pass
```

Explanation: Since `75 >= 50` is **True**, the message `"Pass"` is displayed.

Syntax of `if` Statement

```
if condition:
    statement
```

if-else Statement

An **if-else statement** is a control structure used to make a decision between two alternatives. The `if` block executes when the condition is **True**, while the `else` block executes when the condition is **False**.

Example 1

```
marks = 40

if marks >= 50:
    print("Pass")
else:
    print("Fail")
```

Output:

```
Fail
```

Explanation: Since `40 >= 50` is **False**, the `else` block is executed and `"Fail"` is displayed.

Example 2

```
marks = 75

if marks >= 50:
    print("Pass")
else:
    print("Fail")
```

Output:

```
Pass
```

Explanation: Since `75 >= 50` is **True**, the `if` block is executed and `"Pass"` is displayed.

Syntax of `if-else` Statement

```
if condition:
    statement
else:
    statement
```

Nested Condition

Definition: A **nested condition** is a condition placed **inside another condition**. It is used when we need to check a second condition only after the first condition is true.

Nested `if`

An `if` statement **inside another `if` statement**.

Syntax

```
if condition1:
```

```
    if condition2:
        statement
```

Example

```
age = 20
citizen = True

if age >= 18:
    if citizen:
        print("Eligible")
```

Output:

```
Eligible
```

Nested `if-else`

An `if-else` statement can also be placed inside another `if`.

Syntax

```
if condition1:
    if condition2:
        statement1
    else:
        statement2
else:
    statement3
```

Example

```
age = 20
has_card = False
```

```
if age >= 18:
    if has_card:
        print("Entry allowed")
    else:
        print("Card required")
else:
    print("Under age")
```

Output:

```
Card required
```

> **Note:** Students can do **any one example** from the above examples.

Looping Construct

Definition: A **looping construct** is a control structure used to **repeat a set of instructions again and again** until a specific condition is met or for a specific number of times.
It is also called a **repetitive structure** or **iterative structure**.

While Loop

Definition: A **while loop** is a looping construct that **repeats a set of instructions as long as a given condition is true**.

Syntax

```
while condition:
    statement
```

Example

```
num = 1

while num <= 5:
    print(num)
    num = num + 1
```

Output:

```
1
2
3
4
5
```

How it works

1. `num` starts at `1`.
2. The condition `num <= 5` is checked.
3. If it is true, the statement is executed.
4. `num` increases by `1`.
5. The condition is checked again.
6. When `num` becomes `6`, the condition becomes false and the loop stops.

Important Programs for Exam Preparation

The following programs are **important from an exam point of view**. Practice these programs carefully to understand the working of the `while` loop.

1. Program to Print Numbers from 1 to 10

```
num = 1

while num <= 10:
```

```
    print(num)
    num += 1    # num += 1 means num = num + 1
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

2. Program to Print Even Numbers from 1 to 20

```
i = 2

while i <= 20:
    print(i)
    i += 2    # i += 2 means i = i + 2
```

Output:

```
2
4
6
8
10
12
14
16
```

```
18
20
```

3. Program to Print Odd Numbers from 1 to 20

```
i = 1

while i <= 20:
    print(i)
    i += 2    # i += 2 means i = i + 2
```

Output:

```
1
3
5
7
9
11
13
15
17
19
```

4. Program to Print Table of a Number Stored in a Variable

```
num = 5
i = 1

while i <= 10:
    print(num, "x", i, "=", num * i)
    i += 1    # i += 1 means i = i + 1
```

Output:

```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

5. Program to Print Table of a Number Entered by the User

```
num = int(input("Enter a number: "))
i = 1

while i <= 10:
    print(num, "x", i, "=", num * i)
    i += 1    # i += 1 means i = i + 1
```

Sample Input:

```
Enter a number: 7
```

Output:

```
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
```

```
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

> **Note:** Students must watch these program videos on my YouTube channel **ComputerWithSalman** for better understanding. This will help you understand the programs clearly and perform similar programs easily in your exams.

For Loop

Definition: A **for loop** is a looping construct that **repeats a set of instructions for each item in a sequence or for a specific number of times**.

Syntax

```
for variable in sequence:
    statement
```

Example

```
for num in range(1, 6):
    print(num)
```

Output:

```
1
2
3
```

```
4
5
```

How it works

1. `range(1, 6)` generates numbers from `1` to `5`.
2. The `for` loop takes one number at a time.
3. The value is stored in the variable `num`.
4. The `print()` statement displays the value of `num`.
5. The loop repeats until all numbers in the sequence have been processed.

Important Programs for Exam Preparation

The following programs are **important from an exam point of view**. Practice these programs carefully to understand the working of the `for` loop.

1. Program to Print Numbers from 1 to 10

```
for num in range(1, 11):
    print(num)
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

2. Program to Print Even Numbers from 1 to 20

```
for i in range(2, 21, 2):
    print(i)
```

Output:

```
2
4
6
8
10
12
14
16
18
20
```

3. Program to Print Odd Numbers from 1 to 20

```
for i in range(1, 21, 2):
    print(i)
```

Output:

```
1
3
5
7
9
11
13
15
```

```
17
19
```

4. Program to Print Table of a Number Stored in a Variable

```
num = 5

for i in range(1, 11):
    print(num, "x", i, "=", num * i)
```

Output:

```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

5. Program to Print Table of a Number Entered by the User

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

for i in range(1, 11):
    print(num, "x", i, "=", num * i)
```

Sample Input:

```
Enter a number: 7
```

Output:

```
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

> **Note:** Students must watch these program videos on my YouTube channel **ComputerWithSalman** for better understanding. This will help you understand the programs clearly and perform similar programs easily in your exams.

range() Function

Definition: The `range()` function is used to **generate a sequence of numbers** within a given range. It is commonly used with loops to repeat a task a specific number of times.

Syntax

```
range(start, stop, step)
```

- **start** → The number where the sequence starts.
- **stop** → The number where the sequence stops **(this number is not included)**.
- **step** → The amount by which the number increases or decreases.

Example 1: `range(5)`

```
for i in range(5):
    print(i)
```

Output:

```
0
1
2
3
4
```

Notice that **5 is not included**.

Example 2: `range(1, 6)`

```
for i in range(1, 6):
    print(i)
```

Output:

```
1
2
3
4
5
```

Example 3: Using Step

```
for i in range(2, 11, 2):
    print(i)
```

Output:

```
2
4
6
8
10
```

Here:

- **2** → Starting number
- **11** → Stopping point (not included)
- **2** → Increase by 2

end Statement

Definition: The `end` statement is used with the `print()` function to **stop line breaking and keep the output on the same line**.

Syntax

```
print(value, end="something")
```

Example 1: Default `print()`

```
print("Hello")
print("World")
```

Output:

```
Hello
World
```

Here, `print()` automatically moves to the next line.

Example 2: Using `end=" "`

```
print("Hello", end=" ")
print("World")
```

Output:

```
Hello World
```

`end=" "` tells `print()` to **stay on the same line** and add a space.

Example 3: Using `end=""`

```
print("Hello", end="")
print("World")
```

Output:

```
HelloWorld
```

Example 4: Using `end="-"`

```
print("Hello", end="-")
print("World")
```

Output:

```
Hello-World
```

Nested Loop

Definition: A **nested loop** is a loop placed **inside another loop**.

Nested loops are **mostly used to print patterns**, such as **triangles, squares, rectangles, pyramids, and other shapes**.

Syntax of Nested Loop

Using `for` Loop

```
for variable1 in range():
    for variable2 in range():
        statement
```

Using `while` Loop

```
while condition1:
    while condition2:
        statement
```

Examples of Nested Loop

Example 1: Square Pattern

```
for i in range(1, 5):
    for j in range(1, 5):
        print("*", end=" ")
    print()
```

Output:

```
* * * *
* * * *
* * * *
* * * *
```

Here:

- The **outer loop** controls the rows.
- The **inner loop** controls the columns.
- Together, they create the **square pattern**.

Example 2: Triangle Pattern

```
for i in range(1, 5):
    for j in range(i):
        print("*", end=" ")
    print()
```

Output:

```
*
* *
* * *
* * * *
```

Square Pattern Using `while` Loop

```
i = 1

while i <= 4:
    j = 1

    while j <= 4:
        print("*", end=" ")
        j += 1    # j += 1 means j = j + 1

    print()
    i += 1        # i += 1 means i = i + 1
```

Output:

```
* * * *
* * * *
* * * *
* * * *
```

Triangle Pattern Using `while` Loop

```
i = 1

while i <= 4:
    j = 1

    while j <= i:
        print("*", end=" ")
        j += 1    # j += 1 means j = j + 1
```

```
    print()
    i += 1        # i += 1 means i = i + 1
```

Output:

```
*
* *
* * *
* * * *
```

Libraries

Definition: A **library** is a collection of **pre-written code, functions, and tools** that we can use in a program instead of writing everything from the beginning.

Why do we use libraries?

Libraries save **time and effort** because they provide ready-made functions for different tasks.

1. `random` Library

The `random` **library** is used to generate **random numbers or random values**.

Example:

```
import random

num = random.randint(1, 10)

print(num)
```

Output:

```

```

```
7
```

The output can be any number from **1 to 10**, so it may be different each time.

2. `statistics` Library

The `statistics` **library** is used to perform **statistical calculations**, such as finding the mean, median, and mode.

Example:

```
import statistics

marks = [60, 70, 80, 90, 100]

print(statistics.mean(marks))
```

Output:

```
80
```

Here, `mean()` calculates the **average** of the given numbers.

Another Example:

```
import statistics

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

print(statistics.median(numbers))
print(statistics.mode(numbers))
```

Output:

```
20
20
```

- `mean()` → finds the **average**.
- `median()` → finds the **middle value**.
- `mode()` → finds the **most repeated value**.

> **Note:** Dear Students, the extra examples are provided for **better understanding** and **additional learning**. You can write **any one example** in the exam paper and **skip the extra examples**.

Python List

Definition

A **list** is a collection of multiple values stored in a **single variable**.
A list can store different types of data, such as numbers, strings, or a combination of values.

Creating a List

A list is created using **square brackets** `[ ]`, and values are separated by commas.

Syntax

```
list_name = [value1, value2, value3, ...]
```

Example 1: List of Numbers

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

print(numbers)
```

Output:

```
[10, 20, 30, 40, 50]
```

Example 2: List of Names

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

print(names)
```

Output:

```
['Ali', 'Ahmed', 'Sara', 'John']
```

Example 3: List with Different Data Types

```
student = ["Ali", 20, 85.5]

print(student)
```

Output:

```
['Ali', 20, 85.5]
```

List Elements and Index Numbers

List Elements

The **values stored inside a list** are called **list elements** or **list items**.

Example:

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

Here:

- `Ali` is a list element.
- `Ahmed` is a list element.
- `Sara` is a list element.
- `John` is a list element.

Index Number

An **index number** is the **position number of an element in a list**.

In a list, the index number **starts from 0**.

Element	Ali	Ahmed	Sara	John
Index	0	1	2	3

> **Note: Element** = The value stored in the list. **Index** = The position of the element in the list. The first element always has **index 0**.

Accessing List Items

Definition: Accessing list items means **getting or retrieving a particular value from a list**.

List items are accessed using their **index number**.

Important Point

The index of a list **starts from 0**.

For example:

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

Item	Ali	Ahmed	Sara	John
Index	0	1	2	3

Example

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

print(names[0])
print(names[2])
```

Output:

```
Ali
Sara
```

Here:

- `names[0]` → accesses **Ali**.
- `names[2]` → accesses **Sara**.

Accessing Items from the End

Python also allows **negative indexing**.

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

print(names[-1])
print(names[-2])
```

Output:

```
John
Sara
```

Here:

- `-1` → last item.

- `-2` → second-last item.

> **Note:** We access list items by using their **index number**, starting from **0**.

Modifying a List

Definition: Modifying a list means **changing, adding, or removing elements from an existing list**.
Python provides different operations to modify a list, such as **append, remove, sort, and reverse**.

1. `append()`

The `append()` method is used to **add a new element at the end of a list**.
Example:

```
numbers = [10, 20, 30]

numbers.append(40)

print(numbers)
```

Output:

```
[10, 20, 30, 40]
```

Here, `40` is added at the **end** of the list.

2. `remove()`

The `remove()` method is used to **remove a specific element from a list**.
Example:

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

numbers.remove(20)
```

```
print(numbers)
```

Output:

```
[10, 30, 40]
```

Here, `20` is removed from the list.

3. `sort()`

The `sort()` method is used to **arrange the elements of a list in ascending order**.

Example:

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

numbers.sort()

print(numbers)
```

Output:

```
[10, 20, 30, 40]
```

4. `reverse()`

The `reverse()` method is used to **reverse the order of elements in a list**.

Example:

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

numbers.reverse()
```

```
print(numbers)
```

Output:

```
[40, 30, 20, 10]
```

Quick Summary

Operation	Purpose
`append()`	Adds an element at the end
`remove()`	Removes a specific element
`sort()`	Arranges elements in ascending order
`reverse()`	Reverses the order of elements

Testing and Debugging

1. Testing

Definition: Testing is the process of checking a program to find **errors or problems** and to make sure it works correctly.

Types of Testing

1. Unit Testing

Testing a **small individual part** of a program, such as a function.

Example: Testing a function that calculates the sum of two numbers.

> **Note: Unit = Testing one small part**

2. Integration Testing

Testing whether **different parts of a program work correctly together**.

Example: Checking whether the login system works correctly with the database.

> **Note: Integration = Testing parts together**

3. Functional Testing

Testing whether the program's **features work according to the requirements**.

Example: Testing whether a calculator correctly performs addition, subtraction, multiplication, and division.

> **Note: Functional = Does the feature work correctly?**

4. Regression Testing

Testing the program again after making **changes or fixing errors** to make sure the old features still work correctly.

Example: You fix the login system. After fixing it, you test the login system and other features to make sure the changes did not break anything else.

> **Note: Regression = Check again after changes**

2. Debugging

Definition: Debugging is the process of **finding and fixing errors** in a program.

Common Debugging Techniques

1. Print Statement

We can use `print()` to check the **values of variables** and understand where the program is going wrong.

Example:

```
num = 10
result = num * 2

print("num =", num)
print("result =", result)
```

Output:

```
num = 10
result = 20
```

This helps us check whether the values are correct.

> **Note: Print statement = Check values and program flow**

2. Debugging Tools

Debugging tools help us **find and understand errors step by step**.

They can allow us to:

- Run the program one line at a time.
- Check variable values.
- Find where the error occurs.
- Stop the program at a particular line using a **breakpoint**.

> **Note: Debugging tools = Find errors step by step**

3. Error Messages

When an error occurs, the program provides an **error message** that tells us what went wrong and often shows the **line where the error occurred**.

Example:

```
num = 10
print(numm)
```

Error:

```
NameError: name 'numm' is not defined
```

Here, the error message tells us that `numm` has not been defined. The correct variable is `num`.

> **Note: Error message = Gives information about the problem**

Quick Summary

Topic	Meaning
Testing	Checking whether the program works correctly
Unit Testing	Testing one small part
Integration Testing	Testing different parts together
Functional Testing	Testing whether features work correctly
Regression Testing	Testing again after changes
Debugging	Finding and fixing errors
Print Statement	Checking values and program flow
Debugging Tools	Finding errors step by step
Error Messages	Information about what went wrong

Multiple Choice Questions (MCQs) on Python Programming

1. What is a control structure in Python?

a. A way to store data
b. A way to control the order of program execution
c. A type of library
d. A type of variable

> **Answer: b. A way to control the order of program execution**

2. How many main types of control structures are discussed?

a. 2
b. 3
c. 4
d. 5