Unit II

Control Flow - if Statement, if-else, if-elif-else. Iterative Statements – while, for, Nested Loops, Loop Control Statements – break, continue, pass; else with loops , **Need for Functions**, Defining & Invoking User-defined Functions, Return Statement, Function Input/Output Cases, **Scope of Variables** - Local, Global, Nested Functions, Function Arguments - Required, Positional, Default, Variable-length, main() Function, Documentation Strings, Recursive Functions, Anonymous Functions (Lambda), Modules - Import, from..import, Creating & Using Modules

---

Control Statements in Python:-

Control statements are used to control the flow of execution of a program.
They decide which statements run, how many times they run, or when execution should stop or skip.

In Python, control statements are mainly divided into three types :-
1) Decision making or conditional statements
2) iterative statements
3) Transfer statements

1) Decision making or conditional statements :-

Decision control statements check a condition and execute a particular block of code only if the condition is satisfied.
Decision Control Statements are used to make decisions in a program.
They allow the program to execute different blocks of code based on conditions (True or False).

Types of Conditional statements :-

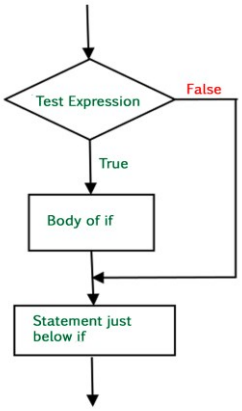
- if statement
- nested if statement
- if-else statement
- if-elif-else statement

if statement :-

The if statement checks a condition.
If the condition evaluates to True, the statements inside the if block are executed; otherwise, they are skipped.

Syntax :-

```
if condition:
    statement(s)
```



Ex :-

```
language=input("Enter the language:")
if(language=="python"):
   print("Python is a Geneal purpose used programming language.")
```

Ex :-

```
x = 10
if x > 5:
   print("x is greater than 5")
```

Ex :-

```
age = int(input("Enter the age:"))
if age >= 18 and age <= 60:
   print("Eligible to work")
```

Nested if statement:- (inner if)

It is also known as inner if statement. inner if statement can be represent as defining or
declaring a if statement inside another if statement while working nested or inner if statement outer if statement must be true.

Syntax :-

```
if condition1:
   if condition2:
      statement(s)
```

Ex :-

```
first_name=input("Enter the first name:")
last_name=input("Enter the last name:")
if(first_name=="Shri"):
   print("my first name is:",first_name)
   if(last_name=="Gnanambica"):
      print("my last name is:",last_name)
```

Ex :-

```
username = "admin"
password = "1234"
if username == "admin":
   if password == "1234":
      print("Login Successful")
```

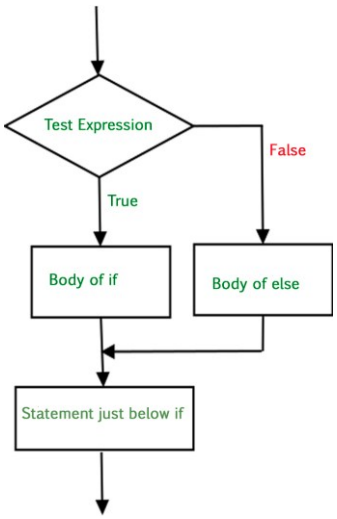
if-else statement :-

An if-else statement checks a condition; if it is true, the if block executes, otherwise the else block executes.
The if-else statement is a decision control statement used to execute one block of code

when a condition is true and another block when the condition is false.

Syntax :-

```
if condition:
   statement(s)
else:
   statement(s)
```



Ex :-

```
p1=input("Enter the password:")
p2=input("Enter the confirm password:")
if(p1==p2):
   print(p1, p2 ,"valid password")
else:
   print(p1,p2, " invalid password")
```

Ex :-

```
number=int(input("Enter the number:"))
if(number % 2==0):
   print(number, "Even number")
else:
   print(number, "odd number")
```

Ex:-

```
age = 16
if age >= 18:
   print("Eligible to vote")
else:
   print("Not eligible to vote")
```

Ex :-

```
marks =int(input("Enter the marks:"))
if marks >= 35:
   print("Pass")
else:
   print("Fail")
```

Ex :-

```
str1=input("Enter the object:")
if(str1==str1[::-1]):
```

```
    print(str1,"Palindrome number")
else:
    print(str1,"Not palindrome number")
```

if-elif-else statement :-

The if-elif-else statement is a decision control statement used to check multiple conditions one by one and execute the block of code whose condition is True.
The if-elif-else statement allows a program to test multiple conditions and execute only one block of code among them.

Syntax :-

```
if condition1:
    statement(s)
elif condition2:
    statement(s)
elif condition3:
    statement(s)
else:
    statement(s)
```

Ex :-

```
language=input("Enter the technology:")
if(language=="Python"):
    print(language, "Python is a general purpose")
elif(language=="Javascript"):
    print(language, "client side validation")
elif(language=="Django"):
    print(language, "to develop web application")
elif(language=="Reactjs"):
    print(language, " single page application")
elif(language=="CSS"):
    print(language, "creating styles")
else:
    print(language,"Invalid please enter proper language")
```

Ex :-

```
marks = int(input("Enter marks: "))
if marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
```

```
elif marks >= 60:
    print("Grade C")
else:
    print("Fail")
```

Iterative Statements :-

Iterative statements are used to repeat a block of code multiple times until a given condition is satisfied. They are also called looping statements.
Iterative statements allow a program to execute a set of instructions repeatedly based on a condition or a sequence.

Types of Iterative Statements in Python :-

Python has two main iterative statements:

1. for loop
2. while loop

1) for loop :-

The for loop is an iterative (looping) statement used to execute a block of code repeatedly for a fixed number of times or for each element in a sequence (like list, string, tuple, or range).

Syntax :-

```
for variable in sequence:
    statement(s)
```

Ex :- (Print numbers from 1 to 5)

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

Ex :- (Loop through a string)

```
name = "Python"
for ch in name:
    print(ch)
```

Ex :-

```
for x1 in range(15):
    print("Python")
```

Ex :- (Loop through list)

```
numbers = [10, 20, 30, 40]
for num in numbers:
```

```
    print(num)
```
Ex:-
```
for i in range(1,11):
    if i % 2==0:
        print(i)
```

while loop :-

A while loop is used to repeat a block of code as long as a condition is true. When the condition becomes false, the loop stops.

Syntax :-
```
while condition:
    statements
```

Ex :- (Print numbers from 1 to 5)
```
i = 1
while i <= 5:
    print(i)
    i = i + 1
```
Explanation :-

i = 1 → start value(initialization)

i <= 5 → condition (The condition decides how long the loop should run.)

print(i) → prints number

i = i + 1 → increases value

Loop stops when i becomes 6

- Initialization is assigning a first value to a variable before using it in a loop or program.

Infinite while loop :-

An infinite while loop is a loop that never stops running because its condition is always True.

Ex :-
```
while True:
    print("Hello")
```
(This loop runs forever.)

How to stop an infinite loop :-

Use break statement inside the loop:

Ex :-
```
while True:
    print("Hello")
    break   # stops the loop after first iteration
```

Output :-
Hello

Stop it manually (in console) :-

Press Ctrl + C in Python terminal or IDE.

Ex:-

```
i=1
while i<=10:
   if i %2==0:
      print(i)
   i=i+1
```

Ex :-

```
i=1
while i<=10:
   if i % 2==0:
      print(i, "Even")
   else:
      print(i,"odd")
   i+=1
```

Ex :-

```
n = int(input("Enter a number: "))
i = 1
while i <= n:
   if i % 2 == 0:
      print(i, "EVEN")
   i += 1
```

Transfer Statements:-

In Python, transfer statements (also known as jump statements or loop control statements) alter the normal flow of program execution.

Transfer statements are used to change the normal flow of execution of a program. they transfer the control of the program from one part of the code to another.

in python, the main transfer statements are:

1. break
2. continue
3. pass

1. break statement :-

The break statement is used to immediately stop a loop when a certain condition becomes true.

When break is executed, the control comes out of the loop and the program continues with the next statement after the loop.

Ex:-

```
for i in range(1,6):
   if i==3:
      break
   print(i)
```

(when I becomes 3, the loop stops and exits).

Ex:-

```
i=1
while i <=5:
   if i==4:
      break
   print(i)
   i=i+1
```

Ex:-

```
color=["red","white","blue", "yellow"]
for i in color:
   if i=="blue":
      break
   print(i)
```

2. continue statement :-

The continue statement is used to skip the current iteration of a loop and move to the next iteration immediately. Unlike break, it does not stop the loop; it just skips the remaining statements for the current iteration.

Ex:-

```
for i in range(1,6):
   if i==3:
      continue
   print(i)
```

Ex:-

```
i=0
while i < 5:
   i=i+1
   if i==3:
      continue
   print(i)
```

Explanation:-

1. i starts from 0.
2. each time i increases by 1.
3. when i==3, the continue statement skips printing 3.
4. the loop moves to the next iteration.
5. so the final output prints 1,2,4,5.
6. when using continue in a while loop make sure the increment statement comes before continue, otherwise it may cause an infinite loop.

3. pass statement :-

The pass statement in Python is a null statement.
It does nothing when executed and is used as a placeholder in code.
Python requires at least one statement in a block. If you don't want to write code yet, you can use pass to avoid errors.

Ex:-

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

Ex:-

```
i=1
while i<=5:
   pass
   i=i+1
```

Difference Between While Loop and For Loop:-

Loops are used in Python to **repeat a block of code multiple times** until a condition is satisfied. Python mainly provides **two types of loops: while loop and for loop**. Both loops are used for repetition, but they are used in different situations depending on the requirement of the program.

While Loop	For Loop
The while loop executes a block of code as long as the condition is true.	The for loop executes a block of code for a fixed number of iterations.
It is mainly used when the number of iterations is not known in advance.	It is used when the number of iterations is already known.
The condition is checked before every iteration.	It automatically takes values from a sequence like list, tuple, or range.
The programmer must manually update the loop variable.	The loop variable is automatically updated.
If the condition never becomes false, it may lead to an infinite loop.	It rarely leads to an infinite loop because the sequence controls the iterations.

Python Program to Display Numbers from 1 to 10 Using For Loop and Function:-

A function is a block of code that performs a specific task. Functions help in organizing programs and allow code reuse. In this program, a function is created to display numbers from 1 to 10 using a for loop.

Program

```
def display_numbers():
    for i in range(1, 11):
        print(i)

display_numbers()
```

Ex:-

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

Functions:-

A function is a block of code that performs a specific task. function improves efficiency and reduces errors because of the reusability of a code. once we create a function, we can call it anywhere and anytime. the benefit of using a function is reusability. --> Reusability means You can write a function once and use it many times without rewriting the same code again and again.

Reusability of code:-

using the same code again and again instead of writing it multiple times. code reusability means write once, use many times.

Ex:-

```
a=10
b=20
print(a+b)
c=5
d=7
print(c+d)
```

-->> same logic written again and again.

Ex:-

```
def add(x,y):
    return x+y
print(add(10,20))
print(add(5,2))
```

-->> same code reused many times.

creating a function:-

* function blocks start with the def keyword. after that the function name and parentheses () are used.
* an argument or parameters should be pass inside the parentheses.
* any function's code block begins with a colon(:) and is indented.

Calling a function:-

To call a function, write its name followed by parentheses. You can call the same function multiple times.

Ex:-

```
def my_function():
    print("Hello world")
my_function()
```

parameters :-

a parameter is the variable listed inside the parentheses in the function definition.

Arguments:-

Information can be passed into functions as arguments.
An argument is the actual value that is sent to the function when it is called.

Ex:-

```
def my_function( name):
    print(name)
my_function("Ramu")
my_function("Ravi")
```

Parameter

arguments

Importance of Functions in Python:-

1. Code Reusability

Functions allow the same piece of code to be used multiple times in a program. Once a function is defined, it can be called whenever required. This reduces repetition of code and saves time in programming.

2. Reduces Program Length

Functions help reduce the size of the program. Instead of writing the same code repeatedly, we can write it once inside a function and call it whenever needed.

3. Easy to Understand

Functions make programs easier to read and understand. By dividing a large program into small functions, the code becomes more organized and clear.

4. Easy Debugging

If an error occurs in a program, it becomes easier to find and fix the error because each function performs a specific task.

5. Improves Program Structure

Functions help in developing well-structured and modular programs. Each function handles a specific task, which improves program organization.

6. Saves Development Time

Using functions saves programming time because programmers do not need to write the same logic again and again. This makes the development process faster.

Return statement :-

The return statement in python is used to send a value from a function back to the place where the function was called. When a function reaches the return statement, it stops executing and returns the specified value. The returned value can be stored in a variable or used in further calculations.

If a function does not contain a return statement, it automatically returns None. The return statement is useful when we want the function to produce a result that can be used later in the program.

Ex:-

```
def add(a,b):
    result=a+b
    return result
sum=add(10,20)
print(sum)
```

Ex:-

```
def square(n):
   return n * n
print(square(5))
```

Ex:-

```
def check_number(num):
   if num % 2==0:
      return "Even Number"
   else:
      return "odd Number"
print( check_number(10))
print( check_number(7))
```

Types Of Functions:-

1. Built-in Functions(Predefined Function):-

Built-in functions are the predefined functions provided by python. These functions are already available in the python standard library and can be used directly them. They perform common operations such as printing output, calculating length, converting data types, and performing mathematical calculations.

Built-in functions make programming easier and reduce the need to write code for common tasks.

Examples:- print(), len(), type(), input(), sum(), max() and min().

Ex:-

```
name="python programming"
print(len(name))
print(type(name))
```

2. User-Defined Functions:-

User-defined functions are the functions created by the programmer according to the requirements of the program. These functions are defined using the def keyword followed by the function name and parameters. User-defined functions help divide large programs into smaller and manageable parts. They improve code reusability, readability, and maintainability. Once a function is defined, it can be called multiple times in the program.

Ex:-

```
def add(a,b):
   print(a+b)
add(10,20)
add(5,15)
```

3. Lambda Functions (Anonymous Functions):-

A lambda function is a small anonymous function that is defined without using the def keyword. It is created using the lambda keyword and usually contains a single expression. Lambda functions are used when a small function is required for a short period of time. They are commonly used with functions like map(), filter() and reduce().
Lambda function make the code shorter and simpler when performing simple operations such as mathematical calculations.

Ex:-
```
square = lambda x: x * 2
print(square(2))
```

Ex:-
```
addition=lambda a,b : a+b
print(addition(2,3))
```

Ex:- (normal function)
```
def squaringNumber(number):
   return number **2
print(squaringNumber(10))
```

Ex:- Lambda function
```
print((lambda number: number**2)(10))
```

4. Recursive Functions:-

A recursive function is a function that calls it self during its execution. Recursion is used to solve problems that can be broken down into smaller subproblems. In recursive functions, a base condition must be defined to stop the recursion, otherwise, the program may run indefinitely. Recursive functions are commonly used in problems like factorial calculation, Fibonacci series, searching algorithms.
A recursive function has two important parts:
1. Base Condition:- the condition that stops the recursion.
2. Recursive call :- the function calling itself with a smaller or simple input.
Ex:-
```
def fact(n):
   if n==0:
      return 1
   return n * fact(n-1)
print(fact(5))
```

Explanation:-
Fact() is a recursive function
If n==0 is the base condition

Fact(n-1) is the recursive call
Each call reduces the value of n until it reaches 0
The final result is 120 (5*4*3*2*1)

Ex:- (sum of natural numbers)

```
def sum_numbers(n):
   if n==1:
      return 1
   return n + sum_numbers(n-1)
print(sum_numbers(5))
```

Scope of Variable in Python :-

The scope of a variable in Python is defined as the specific area or region where the variable is accessible to the user. In python, variables defined in different places of a program have different scopes. The scope determines where a variable is visible and where it can be used. If a variable is accessed outside its scope, python will produce an error.

Python mainly has two types of variable scope:-
1. Local Scope
2. Global Scope

1. Local Scope:-

A local variable is a variable that is defined inside function. It can be accessed only with that function and cannot be used outside the function. Local variables help in protecting data and avoiding conflicts with variables in other parts of the program.

Ex:-

```
def display():
   x=10  # local variable
   print(x)
display()
```

2. Global Scope:-

A global variable is a variable that is defined outside all functions. It can be accessed from any part of the program, including inside functions. It can be used inside and outside functions.

Ex:-

```
x=20
def show():
   print(x)
show()
print(x)
```

using Global keyword:-

sometimes we want to modify a global variable inside a function. For this, python provides the global keyword.

Ex:-

```
x=5
def change():
   global x
   x=15
change()
print(x)
```

Different Types of Function Arguments in Python:-

Function arguments are the values that are passed to a function when the function is called. These arguments provide input to the function so that it can perform operations and produce results. Python supports different types of function arguments to make functions flexible and easy to use.

The main types of function arguments in python are:

1. Positional Arguments
2. Keyword Arguments
3. Default Arguments
4. Variable-length Arguments.

1. Positional Arguments:-

In positional arguments, the values passed to the function are assigned to parameters based on their position. The order of arguments must be the same as the order of parameters defined in the function.

Positional arguments must be in the correct order.

Ex:-

```
def student(name,age):
   print(name)
   print(age)
student("Ravi", 21)
```

(here “Ravi” is assigned to name and 21 is assigned to age according to their position in the function call).

2. Keyword Arguments:-

In keyword arguments, the arguments are passed using the parameter names. The order of arguments does not matter.

You can send Arguments with key=value syntax.

Ex:-

```
def student(name, age):
   print(name)
   print(age)
student(name="Ravi", age=21)
```

(since the arguments are specified with keywords, python assigns the values to the correct parameters even if the order changes).

3. Default Arguments:-

A default argument is a parameter that has a default value. If the user does not provide a value when calling the function, the default value is used

Ex:-

```
def greet(name="Student"):
   print(name)
greet("Ravi")
greet()
```

Ex:-

```
def my_function(name="Friend"):
   print(name)
my_function("Ravi")
my_function("Ram")
my_function()
```

4. Variable-Length Arguments:-

Variable-length arguments are used when the number of arguments passed to a function is not fixed. In some programs, we may not know how many values will be given to the function. Python allows this by using the asterisk (*) symbol in the function parameter.
All the values passed are **stored in a tuple** inside the function.

Ex:-

```
def numbers(*num):
   print(num)
numbers(10,20,30)
```

Ex:-

```
def add(*n):
   print(sum(n))
add(5,10)
add(5,10,15)
add(5,10,15,20)
```

Modules in Python:-

A module in Python is a file that contains Python code such as functions, variables, and classes that can be reused in other programs. Modules help programmers organize large programs into smaller, manageable parts. Instead of writing the same code again and again, we can store the code in a module and import it whenever needed. This improves code reusability, readability, and maintenance.

Why modules are used:-

- To organize large programs into smaller files
- To reuse code in multiple programs
- To make programs easy to understand and maintain
- To reduce code duplication
- To share functions and variables between different programs.

Types of Modules in Python:-

Python mainly has two types of modules.

1. Built-in Modules
2.User-defined Modules

1. Built-in Modules

Built-in modules are the modules that are already available in python. These modules provide many useful functions that help programmers perform different tasks such as mathematical calculations, random number generation, date and time operations, etc.

Examples of built-in modules include

- math
- random
- datetime
- os
- sys

math module:-

the math module is a built-in module in python. it provides mathematical functions like square root, factorial, power, pi.

Ex:-

```
import math
print("Square root of 25:", math.sqrt(25))
print("Factorial of 5:", math.factorial(5))
print("Value of pi:", math.pi)
print("2 power 3:", math.pow(2, 3))
```

Random module:-

the random module is a built-in module used to generate random numbers or select random values.

it is useful in games, OTP generation, password generation etc.

Ex:-

```
import time
import random
for i in range(5):
   time.sleep(1)
```

```
   print(random.randint(1000,5000))
```
Ex:-

```
import random
print(random.choice([10,45,78,52,81]))
```

Date and Time module:-

python provides the datetime module to work with date and time.
it helps us to get current date, get current time, format date and time.

Ex:- (Get current Date and Time)

```
import datetime
now=datetime.datetime.now()
print('current date and time:', now)
```

Ex:- (Get only date)

```
import datetime
today=datetime.date.today()
print("today date is:", today)
```

Ex:- (Get only time)

```
import datetime
current_time=datetime.datetime.now().time()
print('current time is:',current_time)
```

Ex:- (Displaying Calendar)

```
import calendar
print(calendar.calendar(2022))
```

Ex:-

```
import calendar
year=2026
month=4
print(calendar.month(year,month))

# Display the calendar for the specified month and year
print(calendar.month(year, month))
```

OS module:-

the os module is a built-in python module used to interact with the operating system.
it helps us to: work with files and folders, get current directory, create or remove directories, run system-related operations.

Example 1:- (Get current working directory)

```
import os
print(os.getcwd())
```

(getcwd() → Returns the current working directory path)

Example 2:- (list files in a folder)

```
import os
print(os.listdir())
```

(listdir() → Shows all files and folders in the current directory.)

Example 3: Create a New Folder

```
import os
os.mkdir("NewFolder")
```

mkdir() → Creates a new directory.

Example 4: Remove a Folder

```
import os
os.rmdir("NewFolder")
```

rmdir() → Removes a directory.

sys module:-

the sys module is a built-in python module used to interact with the python interpreter and system-specific parameters.
it helps us to: access command-line arguments, exit from a program, check python version, view module search path.

Ex:- (check python version)

```
import sys
print(sys.version)
```

(sys.version → Shows the Python version you are using.)

2). user-Defined Modules:-

A user-defined module is a module that is created by the programmer.
It is simply a Python file (.py file) that contains functions, variables, or classes, and can be reused in other programs.
When we create our own .py file and import it into another file, it is called a user defined module.

Ex:-1

Step 1: Create a Module File

mymodule.py

```
def greet():
   print("Hello, welcome to Python")
```

step-2 : use the module in another file

```
import mymodule
mymodule.greet()
```

Example 2: Addition and Subtraction Module

Step 1: Create module (calc.py)

```
def add(a, b):
   return a + b

def sub(a, b):
   return a – b
```

Step 2: Main program

```
import calc
print(calc.add(10, 5))
print(calc.sub(10, 5))
```

Example-3: Module with Variables

data.py

```
college=”Shri Gnanambica Degree College”
year=2026
```

main program:

```
import data
print(data.college)
print(data.age)
```

Fibonacci series:-

The **Fibonacci series** is a sequence of numbers in which each number is the sum of the two preceding numbers. It usually starts with 0 and 1. This sequence was introduced by the Italian mathematician **Leonardo Fibonacci**. The Fibonacci series appears in many areas of mathematics and computer science. Sequence looks like: 0, 1, 1, 2, 3, 5, 8, 13, ...

Ex:-

```
def fib(n):
   a=0
   b=1
   if n==1:
      print(a)
   else:
      print(a)
      print(b)
      for i in range(2,n):
         c=a+b
         a=b
         b=c
         print(c)
fib(10)
```

Explanation

- The function fib(n) is defined to print n Fibonacci numbers.
- Variables a and b are initialized with values 0 and 1.
- If n == 1, only the first number (0) is printed.
- Otherwise, the first two numbers (0 and 1) are printed.
- A for loop runs from 2 to n-1 to generate remaining terms.
- Inside the loop:
    - c = a + b calculates the next number.
    - Values are updated (a = b, b = c) for the next iteration.
    - The new value c is printed.

Ex:- (for loop)

```
n = 10  # number of terms

a = 0
b = 1

print("Fibonacci Series:")

for i in range(n):
   print(a, end=" ")
   c = a + b
   a = b
   b = c
```

5 Marks

1. Differentiate between while loop and for loop
2. Explain loop control statements
3. Explain the scope of variables in Python
4. What are modules in Python?
5. Discuss the importance of functions
6. Write a program using for loop to display numbers from 1 to 10.

10 Marks

1. Write a detailed note on loops in Python. Explain while, for, nested loops with examples.
2. Explain Control Flow Statements in Python in detail.
3. Write a Python program to print Fibonacci series up to n terms using Recursion.
4. Explain different types of Function Arguments in Python with programs.
5. Explain Iterative Statements in Python.

B.Devendra Msc-CS
Dept of Computers
Shri Gnanambica Degree College(A)