Unit 2 Control Flow Structures

❖ Introduction to Control Structures

✓A control structure (or flow of control) is a block of programming that analyses variables and chooses a direction in which to go based on given parameters.

✓A control structure is just a decision that the computer makes. So, it is the basic **decision-making process** in programming and flow of control determines how a computer program will respond when given certain conditions and parameters.

✓ There are two basic aspects of computer programming: **data** and **instructions** .

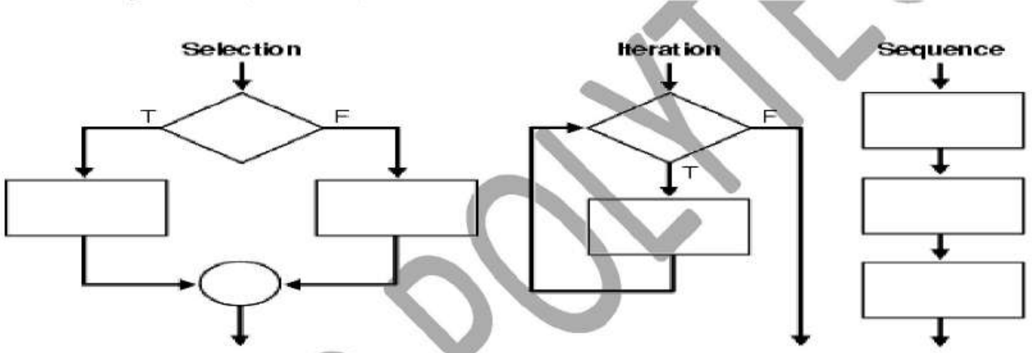
✓ To work with data, you need to understand variables and data types;

✓ to work with instructions, you need to understand control structures and statements.

✓ **Flow of control** through any given program is implemented with three basic types of control structures: Sequential, Selection and Repetition.

✓ There are three types of **Python control structures.**

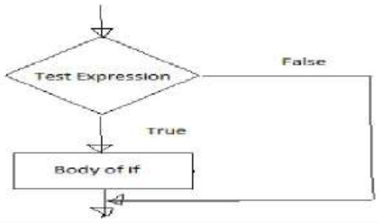
1. **Sequential**
2. **Selection**
3. **Repetition (iteration).**



1. **Python Sequential:** Flow of a program that executes in an order, without skipping, jumping or switching to another block of code. You cannot execute the second instruction before executing the instruction above it.
2. **Python Iteration:** Iteration repeats a body of code as long as condition is true. Python provides while and for loop for iteration.
3. **Python Selection:** Selection allows programmer to ask questions, based on the reslt, perform different action.

❖ if Statement : In this control structure statement execute only if the result of condition is true .

Flowchart	Syntax

	if test expression:
	    statement(s)

- ✓ The program evaluates the test expression. If the test expression is True, then statement(s) will be executed.
- ✓ If the test expression is False, the statement(s) is not executed.
- ✓ In Python, the body of the if statement is indicated by the indentation. The body starts with an indentation and the first unindented line marks the end.
- ✓ Python interprets non-zero values as True. None and 0 are interpreted as False.

Example:

Example: Python if Statement

```
num = 3
if num > 0:
   print(num, "is a positive number.")
print("This is always printed.")
num = -1
if num > 0:
   print(num, "is a positive number.")
print("This is also always printed.")

Output:
3 is a positive number.
This is always printed.
This is also always printed.
```

```
a = 3
if a > 2:
   print(a, "is greater")
print("done")
a = -1
if a < 0:
   print(a, "is smaller")
print("Finish")

Output:
3 is greater
done
-1 is smaller
Finish
```

❖ **if else Statement :**When we have to select only one option from the given two option, then we use if …else.

Flowchart

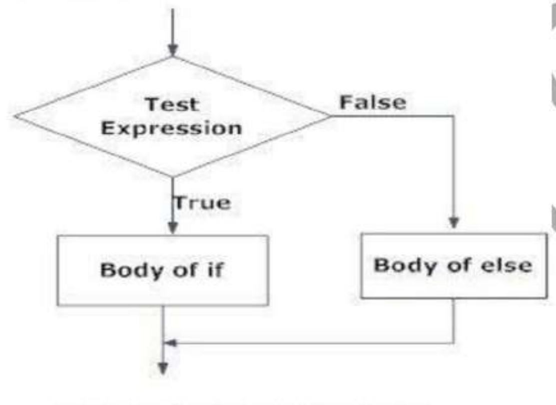


Fig: Operation of if...else statement

Syntax

```
if test expression:
   Body of if
else:
   Body of else
```

- ✓ The if..else statement evaluates test expression and will execute the body of if only when the test condition is True.
- ✓ If the condition is False, the body of else is executed. Indentation is used to separate the blocks.

Example: Python if…else Statement

```
num = 3
if num >= 0:
   print("Positive or Zero")
else:
   print("Negative number")
```

Output:

Positive or Zero

```
a = 7
b = 0
if (a > b):
   print("a is greater than b")
else:
  print("b is greater than a")
```

Output:

a is greater than b

❖ Nested if-else :

Nested "if-else" statements mean that an "if" statement or "if-else" statement is present inside another if or if-else block.

Syntax

```
if ( test condition 1):
        if ( test condition 2):
               Statement1
        else:
                Statement2
else:
      Statement 3
```

- ✓ First test condition1 is checked, if it is true then check condition2.
- ✓ If test condition 2 is True, then statement1 is executed.
- ✓ If test condition 2 is false, then statement2 is executed.
- ✓ If test condition 1 is false, then statement3 is executed

Example: Python Nested if…else Statement

```
num = int(input("Enter a number: "))
if num >= 0:
   if num == 0:
      print("Zero")
   else:
      print("Positive number")
else:
   print("Negative number")
```

Output:

```
Enter a number: 12
Positive number
```

```
a=int(input("Enter A: "))
b=int(input("Enter B: "))
c=int(input("Enter C: "))
if a>b:
   if a>c:
      max=a
   else:
      max=c
else:
   if b>c:
      max=b
   else:
      max=c

print("Maximum  = ",max)
```

Output:

```
 Enter A: 23
Enter B: 2
Enter C: 56
Maximum  =  56
```

❖ if-elif-else statements

Flowchart

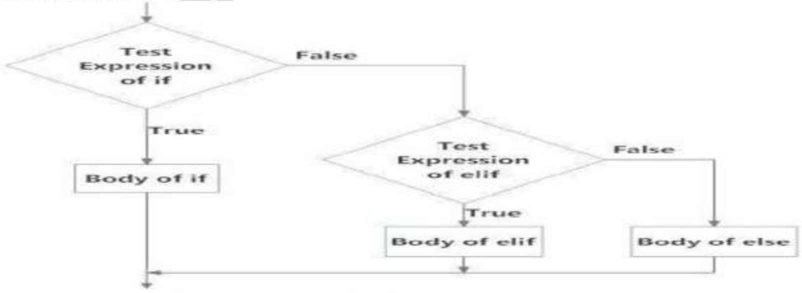


Fig: Operation of if...elif...else statement

Syntax

```
if test expression:
   Body of if
elif test expression:
   Body of elif
else:
   Body of else
```

- The elif is short for else if. It allows us to check for multiple expressions.
- If the condition for if is False, it checks the condition of the next elif block and so on.
- If all the conditions are False, the body of else is executed.
- Only one block among the several if...elif...else blocks is executed according to the condition.
- The if block can have only one else block. But it can have multiple elif blocks.

Example: Python if-elif-else Statement

```
num = -3
if num > 0:
   print("Positive number")
elif num == 0:
   print("Zero")
else:
   print("Negative number")

Output:
Negative number
```

```
a,b,c=12,45,3
if a>b and a>c:
   print("a is max")
elif b>a and b>c:
   print("b is max")
else:
   print("c is max")

Output:
b is max
```

❖ Switch Statement

- Python Switch case is a selection control statement.
- The witch expression evaluated once.The value of the expression is compared with the value of each case.
- If there is a match, the associated block of code is executed.
- Python doesnot have its inbuilt switch case statement.
- We have to implement using different methods.
    - Using Function and Lamda Function
    - Using Dictionary Mapping
    - Using Python classes
    - Using if-elif-else

Using Dictionary Mapping:

```
def monday():
   return "monday"
def tuesday():
   return "tuesday"
def wednesday():
   return "wednesday"
def thursday():
   return "thursday"
def friday():
   return "friday"
def saturday():
   return "saturday"
def sunday():
   return "sunday"
def default():
   return "Incorrect day"

Day = {
   1: monday,
   2: tuesday,
   3: wednesday,
   4: thursday,
   5: friday,
   6: saturday,
   7: sunday
   }

def DayName(dayofWeek):
   return Day.get(dayofWeek,default)( )

print(DayName (3))
print(DayName (8))
Output:
Wednesday
Incorrect day
```

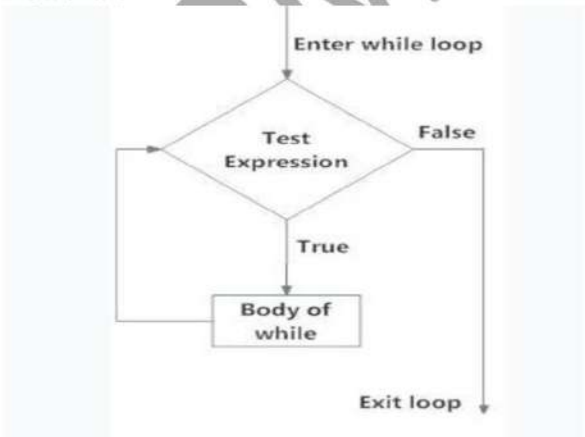
- ✓ First create different functions which return Day name.
- ✓ We create a dictionary in which stored different key-value pair.
- ✓ After that create main function which takes user input for finding day name.
- ✓ At last, called the function to print the desired output

❖ loop

- ✓ A loop statement allows us to execute a statement or group of statements multiple times as long as the condition is true.
- ✓ Repeated execution of a set of statements with the help of loops is called iteration.
- ✓ Loops statements are used when we need to run same code again and again, each time with a different value.
- ✓ In Python Iteration (Loops) statements are of three types:
  1. While Loop
  2. For Loop
  3. Nested For Loops

❖ while loop

Flowchart	Syntax

 Enter while loop
Test Expression
False
True
Body of while
Exit loop
Fig: operation of while loop
	while test_expression:
	        Body of while

- ✓ Loops are either infinite or conditional.
- ✓ In the while loop, test expression is checked first.
- ✓ The body of the loop is entered only if the test_expression evaluates to True.
- ✓ After one iteration, the test expression is checked again. This process continues until the test_expression evaluates to False.
- ✓ The statements that are executed inside while can be a single line of code or a block ofmultiple statements

Example:

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


Output:
1
2
3
4
5
```

```
a = 5
b = 1
while b <= 5:
   print (a,"*",b,"=",a*b)
   b+=1

Output:
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
```

❖ for loop

- ✓ Python for loop is used for repeated execution of a group of statements for the desired number of times.
- ✓ It iterates over the items of lists, tuples, strings, the dictionaries and other iterable objects

Syntax: for var in sequence:

Statement(s)

A sequence of values assigned to var in each iteration

Holds the value of item in sequence in each iteration

Flowchart:

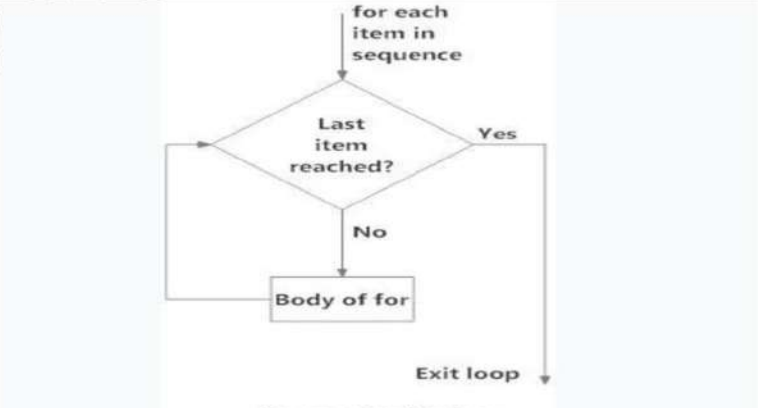


Fig: operation of for loop

Example:

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

Output:
1
2
3
4
5
```

```
a=[10,12,13,14]
for i in a:
        print(i)
Output:
10
12
13
14
```

sum of 1 to 5 no

```
sum=0
for i in range(1,6):
   sum=sum+i
print(sum)

Output:
15
```

Print Multiplication table

```
b=1
a=int(input("enter n:"))
for b in range(1,6):
        print (a,"*",b,"=",a*b)

Output:
enter n:4
4 * 1=4
4 * 2=8
4 * 3=12
4 * 4=16
4 * 5=20
```

```
n=int(input ("enter n:"))
fact=1
for i in range(1,n+1):
   fact=fact*i
print ("factorial=",fact)

Output:
enter n:3
factorial=6
```

```
n=int(input ("enter n"))
n1=0
n2=1
print (n1)
print(n2)
for i in range(3,n+1):
   n3=n1+n2
   print(n3)
   n1=n2
   n2=n3

Output:
0
1
1
2
3
5
```

❖ Nested loops

- ✓ In Python programming language there are two types of loops which are for loop and while loop.
- ✓ Using these loops we can create nested loops in Python.
- ✓ Nested loops mean loops inside a loop. For example, while loop inside the for loop, for loop inside the for loop, etc.

Syntax:	Syntax:
for var in sequence:	while expression:

```
for  var in sequence:
     statements(s)
statements(s)
```

```
while expression:
   statement(s)
statement(s)
```

Example :1

```
rows = 5
for i in range(1, rows + 1):
   for j in range(1, i + 1):
      print('*',end=' ')
   print(‘  ‘)
```

Output:

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

Example :2

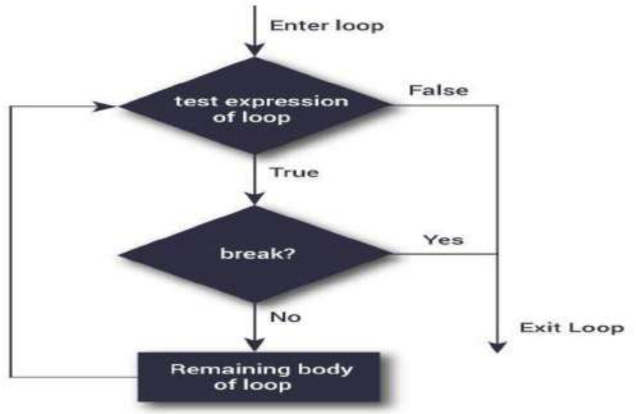
```
rows = 5
for i in range(1, rows + 1):
   for j in range(1, i + 1):
      print(j,end=' ')
   print(‘  ‘)
```

Output:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

❖ break statement

- ✓ The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.
- ✓ If the break statement is inside a nested loop (loop inside another loop), the break statement will terminate the innermost loop.The break statement can be used in both while and for loops.



Syntax:

```
break
```

Example:

```
for i in range(1,11):
   print(i)
   if i == 2:
      break
```

Output:
1

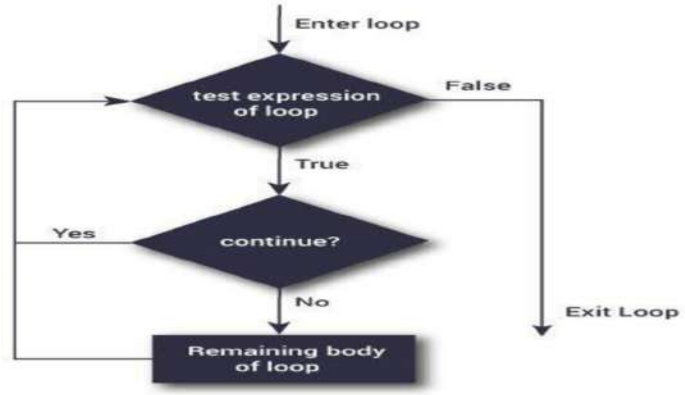
```
i=1
while(i<=10):
   print(i)
   if(i==5):
      break
   i=i+1
```

Output:
1

2	2 3 4 5

❖ continue statement

- ✓ The continue statement in Python returns the control to the beginning of the while loop. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.
- ✓ The continue statement can be used in both while and for loops.

 Enter loop test expression of loop False True Yes continue? No Exit Loop Remaining body of loop	Syntax: continue

Example:

```
n=int(input("enter n"))
for i in range(1,n):
   if(i==5):
      continue
   print("%d" %i)

Output:
enter n10
1 2 3 4 6 7 8 9
```

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

Output:
1  2 4  5
```

❖ Pass Statement

- ✓ In Python programming, the pass statement is a null statement.
- ✓ The difference between a comment and a pass statement in Python is that while the interpreter ignores a comment entirely, pass is not ignored.
- ✓ Nothing happens when the pass is executed. It results in no operation (NOP).
- ✓ When the user **does not know** what code to write, so user simply places **pass** at that line. Sometimes, **pass** is used when the user **doesn't want any code to execute**. So user can simply place **pass** where empty code is not allowed, like in loops, function definitions, class definitions, or in if statements. So using pass statement user avoids this error.
- ✓ Syntax:

```
pass
```

- ✓ Example

```
a= {'h', 'e', 'l', 'l','o'}
for val in a:
   pass
```