SNS COLLEGE OF ENGINEERING
Kurumbapalayam (Po), Coimbatore – 641 107
An Autonomous Institution
Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF CSE

19IT103 – COMPUTATIONAL THINKING AND PYTHON PROGRAMMING

- **A readable, dynamic, pleasant, flexible, fast and powerful language**

Recap

- A Boolean expression is an expression that is evaluated as either true or false.
- Two boolean operators are and and or.
- If statement executes its body only when it is true.
- To execute alternative statements when a condition fails, if-else is useful
- If-elif-else is used to check multiple conditions
- Conditionals inside conditional is said to be nested conditional

3.2 Iterations

- Iterations execute a set of instructions repeatedly until some limiting criteria is met.
- Iterations are **performed through 'for' and 'while' loops**.

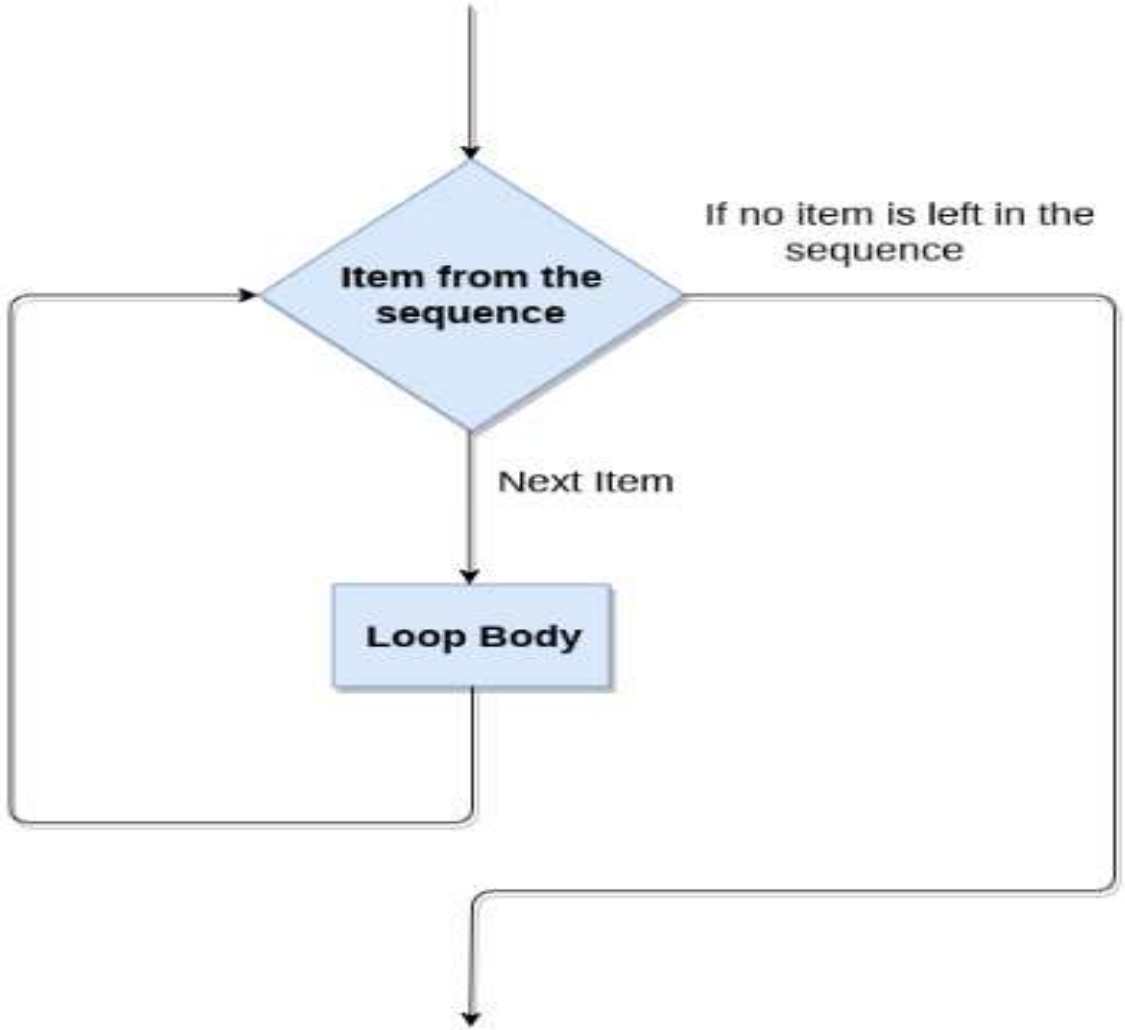
3.2 Iterations

3.2.1 'for' LOOP

- The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects.
- Iterating over a sequence is called traversal.
- Iteration control variable that takes the value of the item inside the sequence on each iteration.
- Loop continues until we reach the last item in the sequence.
- The body of for loop is separated from the rest of the code using indentation.

3.2 Iterations

The for loop flowchart



3.2 Iterations

3.2.1 'for' LOOP

```
[*]: from time import sleep
     from random import randint

     for _ in range(0,5):
         print('Blah')
         sleep(randint(1,4))

     Blah
```

3.2 Iterations

3.2.1 'for' LOOP

Syntax:

```
for val in sequence:
        Body of for
```

Example 1:

```
#number list
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
sum = 0
for val in numbers:
    sum = sum+val
print("The sum is", sum)
```

Output:

```
The sum is 48
>>> |
```

3.2 Iterations

3.2.1 'for' LOOP

Example 2:

```
str = "Python"
for i in str:
    print(i)
```

Output:

```
P
y
t
h
o
n
>>> |
```

3.2 Iterations

3.2.1 'for' LOOP using range() function

- The **range()** function is used to generate the sequence of the numbers.
- If we pass the range(10), it will generate the numbers from 0 to 9.

Syntax

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

- The start represents the beginning of the iteration.
- The stop represents that the loop will iterate till stop-1.
  The **range(1,5)** will generate numbers 1 to 4 iterations. It is optional.
- The step size is used to skip the specific numbers from the iteration. It is optional to use. By default, the step size is 1. It is optional.

3.2 Iterations

3.2.1 'for' LOOP using range() function

Example 3:

```
for i in range(10):
    print(i,end = ' ')
```

Output:

```
0 1 2 3 4 5 6 7 8 9
>>>
```

3.2 Iterations

3.2.1 'for' LOOP using range() function

Example 4:

```
n = int(input("Enter the number "))
for i in range(1,11):
    c = n*i
    print(n,"*",i,"=",c)
```

Output:

```
Enter the number 5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
>>>
```

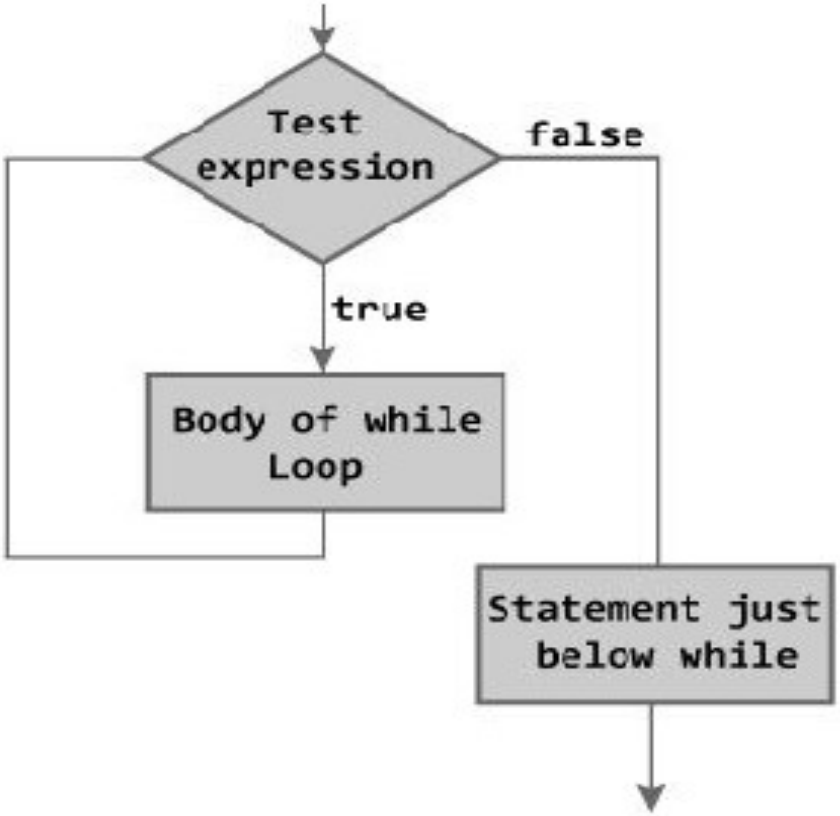
3.2 Iterations

3.2.2 'while' LOOP

- In 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.
- In Python, the body of the while loop is determined through indentation.
- Body starts with indentation and the first unintended line marks the end.
- Python interprets any non-zero value as True. None and 0 are interpreted as False.

3.2 Iterations

3.2.2 'while' LOOP



3.2 Iterations

3.2.2 'while' LOOP

code

```
a = 1
while a < 7 :
    if(a % 2 == 0):
        print(a, "is even")
    else:
        print(a, "is odd")
    a += 1
```

output

variables

www.penjee.com

3.2 Iterations

- **3.2.2 'while' LOOP**

```
1  numbers = [12, 37, 5, 42, 8, 3]
2  even = []
3  odd = []
4  while len(numbers) > 0 :
5      number = numbers.pop()
6      if(number % 2 == 0):
7          even.append(number)
8      else:
9          odd.append(number)
```

3.2 Iterations

3.2.2 'while' LOOP

Syntax

```
while test_expression:
        Body of while
```

Example:

```
n = int(input("Enter a number: "))
sum = 0
i= 1
while i <= n:
    sum = sum + i
    i = i+1
print("The sum is", sum)
```

Output:

```
Enter a number: 10
The sum is 55
>>>
```

3.2 Iterations

3.2.2 'while' LOOP with else

- An optional else block with while loop can also be used.
- The else part is executed if the condition in the while loop evaluates to False.
- The while loop can be terminated with a break statement.

3.2 Iterations

3.2.2 'while' LOOP with else

Example:

```
counter = 0
while counter < 3:
        print("Inside loop")
        counter = counter + 1
else:
        print("Inside else")
```

Output:

```
Inside loop
Inside loop
Inside loop
Inside else
>>>
```

3.2 Iterations

Difference between while and for loop:

while loop	for loop
Indefinite loop	Definite loop
The exit condition will be evaluated again and execution resumes from the top(repeatedly executes a set of code)	The for is to iterate over a sequence (List, Tuple and dictionary etc)

3.2 Iterations

3.2.3 State

- State is a behavioral design pattern that allows an object to change the behavior when its internal state changes.
- The pattern extracts state-related behaviors into separate state classes and forces the original object to delegate the work to an instance of these classes, instead of acting on its own.

Summary

- Iterative statements are used for repeated execution
- ‘for’ and ‘while’ are two looping statements used in python
- ‘for loop’ is definite loop whereas ‘while loop’ is indefinite loop
- State is the change in the behaviour of the objects

3.2 Iterations

- Sometimes there may be a need to exit the loop completely when an external condition is triggered or there may be a situation to skip a part of the code and start the next execution.
- Python provide the following statements
  i. Break
  ii. Continue
  iii. Pass
- In Python, break and continue statements can alter the flow of a normal loop.
- Loops iterate over a block of code until test expression is false, to terminate the current iteration or even the whole loop without checking test expression.
- The break and continue statements are used in these cases.

3.2 Iterations

3.2.4. break Statement

- The break statement terminates the loop containing it.
- Control of the program is transferred to the statement which is present immediately after the body of the loop.
- If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

3.2 Iterations

3.2.4. break Statement

Syntax:

```
break
```

Example 1:

```
for val in "string":
    if val == "i":
        break
    print(val)
print("The end")
```

Output:

```
s
t
r
The end
>>>
```

3.2 Iterations

3.2.4. break Statement

Example 2:

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

Output:

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

3.2 Iterations

3.2.5. continue Statement

- The continue statement is used to skip the rest of the code inside a loop for the current iteration only.
- Loop does not terminate but continues on with the next iteration.

Syntax:

```
continue
```

3.2 Iterations

3.2.5. continue Statement

Example 1:

```
for val in "string":
        if val == "i":
                continue
        print(val)
print("The end")
```

Output:

```
s
t
r
n
g
The end
```

3.2 Iterations

3.2.5. pass Statement

- pass is used when a statement is required syntactically but you do not want any command or code to execute.
- The pass statement is a null operation; nothing happens when it executes.
- The pass is also useful in places where your code will eventually go, but has not been written yet.

3.2 Iterations

3.2.5. pass Statement

Syntax

```
pass
```

Example 1:

```
for letter in "Python":
        if letter == 'h':
                pass
                print("This is pass block")
                continue
        print("Current Letter :",letter)
print("Good bye!")
```

Output:

```
Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : o
Current Letter : n
Good bye!
```

3.2 Iterations

3.2.5. pass Statement

Example:

```
for num in [20, 11, 9, 66, 4, 89, 44]:
    if num%2 == 0:
        pass
    else:
        print(num)
```

```
11
9
89
```

3.4 Functions

- Python Functions is a block of related statements designed to perform a computational, logical, or evaluative task.
- Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or docstring.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

3.4 Functions

Syntax:

```
def function_name(parameters):
    """docstring"""
    statement(s)
    return expression
```

3.4 Functions

Function Definition and Use

- In Python a function is defined using the def keyword:

```
def my_function():
  print("Hello from a function")
```

- To call a function, use the function name followed by parenthesis:

```
def my_function():
  print("Hello from a function")

my_function()
```

3.4 Functions

Function Definition and Use

```
function one() {
  console.log('A');
  two();
  console.log('C');
}
function two() {
  console.log('B');
}
one();
```

Call Stack

Console

3.4 Functions

Function Definition and Use

Example

- https://replit.com/@ErAmbikaM/functionexample#main.py

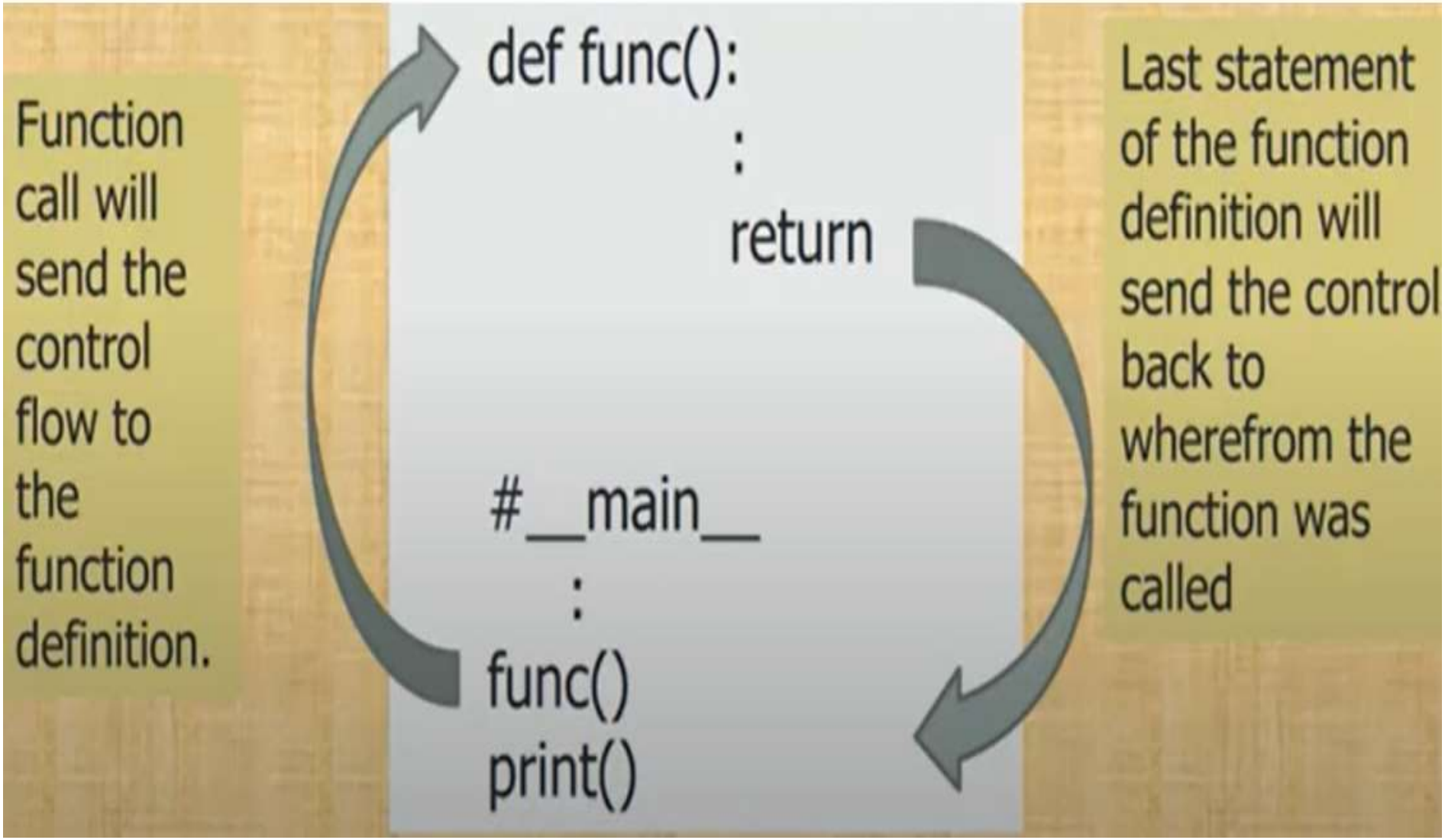
3.4 Functions

Flow of Execution

- Flow of execution - the order in which statements are executed
- Execution always starts at the first statement of the program
- Statements execute one at a time from top to bottom
- Functions definitions do not alter the flow of execution
- When a function is called, the flow control will jump to the first line of the called function
- Then, it will execute all the statements there. After that, it will come back to pick up where it left off.

3.4 Functions

Flow of Execution



3.4 Functions

Flow of Execution

```
1.  # Program to add two numbers
2.  def sum(a,b):
3.              c=a+b                          #statement 1
4.              return c                       #statement 2

5.  num1=int(input("Enter value"))             #statement 4
6.  num2= int(input("Enter value"))            #statement 5
7.  res=sum(num1,num2)                         #statement 6
8.  print("Sum=",res)                          #statement 7
```

Flow of execution according to line numbe is

2->5->6->7->2->3->4->7->8

Function called and executed

Summary

- “break” statement is used terminate the loop in between the iterations
- “continue” statement is used to skip an iteration
- “pass” statement acts as a placeholders for future code
- Python Functions is a block of related statements designed to perform a computational, logical, or evaluative task.
- Flow of execution is the order in which statements are executed


THANK YOU