Marwadi University

Marwadi University

01CE1705-Programming with Python

Unit-2 Program Flow Control and Functions

Prof. Chetan Chudasama
Computer Engineering Department

Program flow control

- In programming languages, most of the time in large projects we have to control the flow of execution of our program, and we want to execute some set of statements only if the given condition is satisfied, and a different set of statements when it's not satisfied.
- Conditional statements are also known as decision-making statements.
- We need to use these conditional statements to execute the specific block of code if the given condition is true or false.
- In Python we can achieve decision making by using the following statements:
    - if statements
    - if-else statements
    - elif statements
    - Nested if and if-else statements

If statements:-

- Python if statement is one of the most commonly used conditional statements in programming languages.

- It decides whether certain statements need to be executed or not.
- It checks for a given condition, if the condition is true, then the set of code present inside the ” if ” block will be executed otherwise not.

Syntax:

```
if test expression:
    statement(s)
```

- Here, the program evaluates the test expression and will execute statement(s) only if the test expression is True.
- 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:-

```
num=int(input("Enter Number "))
if num>0:
    print(num, "is positive number")
print("This is always printed.")
```

- The print() statement falls outside of the if block (unindented) is executed regardless of the test expression.

if-else statements:-

- The statement itself says if a given condition is true then execute the statements present inside the “if block” and if the condition is false then execute the “else” block.
- The “else” block will execute only when the condition becomes false. It is the block where you will perform some actions when the condition is not true.

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:

```
num=int(input("Enter Number "))
if num>=0:
    print(num,"is positive number")
else:
    print(num,"is negative number")
```

elif statements:-

- The elif is short for else if.
- The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

•It is used to check multiple conditions.

Syntax:

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

- 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:

```
import math
height=float(input("Enter Your Height "))
weight=float(input("Enter Your Weight "))
bmi=math.floor(weight/height**2)
if bmi<=18:
    print("Your BMI is",bmi,"You are underweight")
elif bmi>18 and bmi<=25:
    print("Your BMI is",bmi,"Your weight is normal")
elif bmi>25 and bmi<=30:
    print("Your BMI is",bmi,"You are overweight")
```

```
elif bmi>30 and bmi<=35:
    print("Your BMI is",bmi,"You are
obese") elif bmi>35:
    print("Your BMI is",bmi,"You are clinically obese")
```

Nested If:-

- It means that If statement is present inside another if block.
- This means that inner if condition will be checked only if outer if condition is true.

•Indentation is the only way to figure out the level of nesting.

Syntax:-

```
if(condition):
        #Statements to execute if condition is true
        if(condition):
                #Statements to execute if
        condition is true #end of nested if
#end of if
```

Example:

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

- **Short Hand If Statement:-**
- If we have only one statement to execute, we can put it on same line as the if statement.

  Example:

```
num=7
if(num>0):print("Number is greater than zero")
```

Short Hand If ... Else

- If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:
- **Example**

```
a = 2
b = 330
print("A") if a > b else print("B")
```

- This technique is known as Ternary Operators, or Conditional Expressions.

FOR LOOP

Marwadi University

- It has the ability to iterate over the items of any sequence, such as a list or a string.

Syntax:-

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

- First Item in the Sequence is assigned to the iterating variable iterating_var.Next,the statements block is executed.
- Each item in the list is assigned to iterating_var,and the statement block is executed until the entire sequence is empty.

Example:

```
for letter in "India":
    print(letter,end=" ")
```

Output:-

I n d i a

Example:

```
playerName=["Virat","Rohit","Bumrah","Hardik"]
for nm in playerName:
    print(nm,end=" ")
```

Output:

Virat Rohit Bumrah Hardik

for loop using range()

- To loop through a set of code a specified number of times, we can use the range() function.
- The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example:

```
for x in range(6):
    print(x,end=" ")
```

Output:

0 1 2 3 4 5

- As you can see in the output, the variable x is not getting the values 0 1 2 3 4 and 5 simultaneously.
- In the first iteration of the loop value of x is the start number of a range.
- Next, In every subsequent iteration of for loop, the value of x is incremented by one.
- So, it means range() produces numbers one by one as the loop moves to the next iteration.
- The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6)

Example:

```
for x in range(2, 6):
    print(x, end=" ")
```

Output:-

2 3 4 5

- The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3):

Example:

```
for x in range(2,15,3):
        print(x,end=" ")
```

Output:-

2 5 8 11 14

- In every subsequent iteration of for loop, the value of x is incremented by the step value. The value of x is determined by the formula x = x + step.

Iterate a List using range() and for loop:-

- We can iterate a list and string using range() and for loop.
- When we iterate a list using for loop, we can access only elements, but when we use range() function along with the loop,

- We can access index number of each element.
- Using index numbers, we can access as well as modify list elements if required.

Example:

```
playerName=["Virat",18,"Rohit",45,True]
for value in range(len(playerName)):
    print(playerName[value],end=" ")
```

Output:

Virat 18 Rohit 45 True

Nested For Loop

- A nested loop is a loop inside a loop.
- The outer loop can contain more than one inner loop. There is no limitation on the chaining of loops.
- For each iteration of an outer loop the inner loop re-start and completes its execution before the outer loop can continue to its next iteration.

Example: Printing a multiplication table of first ten numbers.

```
for i in range(1,11):
    for j in range(1,11):                #Executes 10 times for each outer number
        print(i,"*",j,"=",i*j)
    print()
```

Example: print pattern

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

Output:-

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

For loop with else:

- Python allows the else keyword to be used with the for and while loops too.
- The else block appears after the body of the loop.
- The statements in the else block will be executed after all iterations are completed.

•The program exits the loop only after the else block is executed.

Example:

```
for x in range(6):
    print(x,end=" ")
else:
    print("\nFinally Finished")
```

Example:

```
for x in range(6):
    if x==4:
        break
    print(x,end=" ")
else:
    print("\nFinally Finished")
```

- The else block just after for/while is executed only when the loop is NOT terminated by a break statement.

For loop in one line:-

•If the loop consist of one statement, simply write this statement into the same line.

Example:

```
for x in range(10):print(x)
```

one line for loop with list comprehension

- List comprehensions are used to create new lists from other iterables like tuples, strings, arrays, lists, etc.
- We want to create a list of squared numbers. The traditional way would be to write something along these lines:

Example:

```
squares=[]
for number in range(1,10):
        squares.append(number**2)
print(squares)
```

Output:

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81].

List comprehension condenses this into a single line of code.

- **print([number**2 for number in range(10)])**

List comprehension is a compact way of creating lists. The simple formula is
[expression + context]

Expression: What to do with each list element? In the above code number**2 is expression.

Context: It consist of for and if statements.

WHILE LOOP

- With the while loop we can execute a set of statements as long as a condition is true.
- We generally use this loop when we don't know the number of times to iterate beforehand.

Syntax:

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

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

Example:

```
i = 1
while i < 6:
    print(i)
    i += 1
```

Output:

```
1 2 3 4 5
```

- We need to increase the value of the i variable in body of loop. This is very important. Failing to do so will result in an infinite loop(never-ending loop).
- The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

while loop with else

- Same as with for loops, while loops can also have an optional else block.
- 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. In such cases, the else part is ignored.
- Hence, a while loop's else part runs if no break occurs and the condition is false.

Example

```
i = 1
while i < 6:
    print(i,end=" ")
    i += 1
else:
    print("\ni is no longer less than 6")
```

Output:

```
1 2 3 4 5
i is no longer less than 6
```

while loop in single line:

- There are two methods of writing a one-liner while loop:

Method 1:

If the loop body consist of one statement, write this statement into same line.

Example

```
while(True):print('Hi')
```

This prints the string 'hi' to the shell for as long as you don’t interfere.

Method 2:

If the loop body consists of multiple statements, use the semicolon to separate them.

```
count=5
while(count>0):print(count, end=" ");count-=1
```

Output:

```
5 4 3 2 1
```

Break Statement

Marwadi University

- It is used to terminate the execution of the loop.
- **Syntax:-**

```
Loop #LOOP START
        condition:
            break
#END OF LOOP
```

- break statement in Python is used to bring the control out of the loop when some external condition is triggered.
- break statement is put inside the loop body(generally after if condition).
- It terminates the current loop, i.e., the loop in which it appears, and resumes execution at the next statement immediately after the end of that loop.
- If the break statement is inside a nested loop, the break will terminate the innermost loop.
- The break statement can be used in both while and for loops.

Example

```
for i in range(10):
    if i==4:
        break
    print(i,end=" ")
```

Output:

0 1 2 3

CONTINUES STATEMENT

- It 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.
- It can be used in both while and for loops.

Example:

```
for i in range(10):
        if i==2 or i==7:
                continue
        print(i,end=" ")
print("\nThe End")
```

- If value of i is 2 or 7,python interpreter not executing the print statement.Hence,we see in our output that all number except 2 and 7 gets printed.

PASS STATEMENT

- It is a null statement.
- When the user does not know what code to write, So user simply places pass at that line.
- Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future.
- They cannot have an empty body. The interpreter would give an error. So, we use the pass statement to construct a body that does nothing.

Example:

```
sequence = {'p', 'a', 's', 's'}
for val in sequence:
    pass
```

- We can do the same thing in an empty function or class as well.

```
def function(args):
    pass

class Example:
    pass
```

- But the difference between pass and comment is that comment is ignored by the interpreter whereas pass is not ignored.

Module

- Modules refer to a file containing Python statements and definitions.
- We use modules to break down large programs into small manageable and organized files.
- Modules provide reusability of code.
- We can define our most used functions in a module and import it, instead of copying their definitions into different programs.

Example:

Let us create a module. Type the following and save it as marwadi.py

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

- Here, we have defined a function add() inside a module named marwadi.The function takes in two numbers and returns their sum.

How to import modules in Python?

- We can import the functions, classes defined in a module to another file using the import keyword.
- To import our previously defined module marwadi,we type following in text editor.

import marwadi

- This does not import the names of the functions defined in module marwadi.py directly in the current file. It only imports the module name marwadi.

•Using the module name, we can access the function using the dot . operator

Example:

marwadi.add(4,5)

- Python has large number of standard module. These modules are in Lib directory inside the location where we installed python.
- Standard modules can be imported the same way as we import our user-defined modules.

- There are various ways to import modules.

Python import statement

We can import a module using the import statement and access the definitions inside it using the dot operator.

Example:

```
import math
print(math.pi)
```

Import with renaming

We can import a module by renaming it.

```
import math as m

print(m.pi)
```

- We have renamed the math module as m.This can save us typing time in some cases.

Marwadi University

Python from...import statement

We can import specific names from a module without importing the module as a whole.

```
from math import sqrt, factorial
print(sqrt(16))
print(factorial(6))
```

- Here, we imported only the sqrt and factorial attribute from the math module.
- In such cases, we don't use the dot operator.

Import all names

- It is also possible to import all names from a module into the current file by using the following import statement.

```
from modulename import *
```

- The use of * has its advantages and disadvantages.
- If you know exactly what you will be needing from the module, it is not recommended to use *, else do so.

Example:

```
from math import *
print(sqrt(16))
print(factorial(6))
```

dir() built in function:-

- We can use the dir() function to find out names that are defined inside a module.
- For example, we have defined a function add() in the module marwadi.
- We can use dir() function in following way:

```
print(dir(marwadi))
```

output:

['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '___spec___', 'add', 'mul']

- Here, we can see add and mul. All other names that begin with an underscore are default Python attributes.

FUNCTION

Marwadi University

- In Python, a function is a group of related statements that performs a specific task.
- Functions help break our program into smaller and modular chunks.
- Functions avoids repetition and makes code reusable.
- Python functions, once defined, can be called many times and from anywhere in a program.

Syntax:-

```
def name_of_function(parameters):
        #code block
```

- The beginning of a function header is indicated by a keyword called def.
- name_of_function is the function's name that we can use to separate it from others. We will use this name to call the function later in the program. The same criteria apply to naming functions as to naming variables in Python.
- We pass arguments to the defined function using parameters. They are optional.

- The function header is terminated by a **colon (:).**
- We can use a documentation string called docstring in the short form to explain the purpose of the function.
- The body of the function is made up of several valid Python statements.
- The indentation depth of the whole code block must be the same **(usually 4 spaces).**
- We can use return statement to return a value from the function.

Calling a Function:-

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

Example:-

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

collegeName()  #Function Call
```

Arguments

- Information can be passed into functions as arguments.
- Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.
- The following example has a function with one argument (cityName).

```
def my_function(cityName):
    print(cityName)
my_function("Rajkot")
my_function("Ahmedabad")
my_function("Baroda")
```

Number of Arguments

- By default, a function must be called with the correct number of arguments.
- Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

Example:

```
def my_function(fname, lname):
    print(fname + " " + lname)
my_function("Rajesh", "Patel")        #Function Call
```

- If you try to call the function with 1 or 3 arguments, you will get an error.

Arbitrary Arguments, *args

- If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.
- This way the function will receive a tuple of arguments, and can access the items accordingly

Example:

```
def playerName(*nm):
    print("Cricketer Name is "+nm)
playerName("Virat","Rohit","Bumrah")      #Function Call
```

Keyword Arguments

- We can also send arguments with the key = value syntax.
- This way the order of the arguments does not matter. Example:

```
def playerName(first, second, third):
        print(first+" will open the innings with "+second+" and "+third+" will play at number 3")

playerName(third="Virat",first="Rohit",second="Dhawan")        #Function Call
```

Arbitrary Keyword Arguments,args**

- If you do not know how many keyword arguments that will be passed into your function,
  add two asterisk: ** before the parameter name in the function definition.
- This way the function will receive a dictionary of arguments, and can access the items accordingly:

Example:

```
def playerName(**nm):
        print(nm)

playerName(third="virat",first="rohit",second="dhawan")
playerName(first="rohit")
playerName()
```

Default Parameter Value

- If we call the function without argument, it uses the default value:

Example:

```
def playerName(nm="virat"):
    print("Cricketer Name is "+nm)
playerName("rohit")
playerName()
```

Passing a List as an Argument

- We can send any data types of argument to a function(integer,float,boolean) and it will be treated as the same data type inside the function.
- In below example we send list as an argument

```
def playerName(a):
        for value in a:
            print(value)
nm=["virat",18,"rohit",45,True,22.5]
playerName(nm)
```

return statement

- It is used to end the execution of the function and give the calculated value(result) to the caller.
- A return statement includes the return keyword and the value that will be returned after that.
- The statements after the return statements are not executed.
- If we do not write a return statement, then None object is returned by a defined function.
- It cannot be used outside of the Python function.
- Example: Defining a function with return statement

```
def square(num):
    return num**2
print(square(4))
```

Anonymous Function

- An anonymous function is a function that is defined without a name.
- Normal functions are defined using the def keyword in Python, anonymous functions are defined using **the lambda keyword.** Hence Anonymous function are also called as lambda functions.

Syntax:-

lambda arguments:expression

- Lambda functions can have any number of arguments but only one expression.
- The expression is evaluated and returned.
- Lambda functions can be used wherever function objects are required.

Example:-

```
double = lambda x: x * 2
print(double(5))
```

- In the above program, lambda x: x * 2 is the lambda function. Here x is the argument and x * 2 is the expression that gets evaluated and returned.
- This function has no name. It returns a function object which is assigned to the identifier double.
  The statement double=lambda x:x*2 is same as

```
def double(x):
    return x * 2
```

Use of Lamba function in Python:-

- We use lambda functions when we require a nameless function for a short period of time.
- Lambda functions are used along with built-in functions like filter(), map() etc.

map():

- The map() function executes a specified function for each item in a list.
- The item is sent to the function as a parameter.
- **syntax:-**

map(function, list)

- function:-Required. **The function to execute for each item**
- list:-Required. A sequence or collection. We can send as many list as we like, just make sure that the function has one parameter for each list.
- It returns map object then can be passed to functions like list(),set().
- Example:

```
lst=[1,2,3,4,5]
def  squareIt(n):
    return n**2
print(list(map(squareIt,lst)))
```

output:- [1,4,9,16,25]

- We can also use lambda function with map() to achieve above result. **Example:-**

```
lst=[1,2,3,4,5]
print(list(map(lambda n:n**2,lst)))
```

Add two lists using map and lambda

Example:-

```
numbers1 = [1, 2, 3]
numbers2 = [4, 5, 6]
result = map(lambda x, y: x + y, numbers1, numbers2)
print(list(result))
```

filter():-

- The filter() method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.

  Syntax: **filter**(function,sequence)

  - function: function that tests if each element of a sequence true or not.
  - sequence: sequence which needs to be filtered, it can be sets, lists, tuples

- The filter() function **returns an iterator**.
- We can easily convert iterators to sequences like lists, tuples, strings etc.
- **Example 1:-** A function that returns True if letter is vowel

```
def filter_vowels(variable):
    letters=['a','e','i','o', 'u']
    if variable in letters:
        return True
    else:
        return False
sequence=['g','e','e','a','c','d','l']
print(list(filter(filter_vowels, sequence)))
```

Output:

['e', 'e', 'a', 'i’]

- Here, the filter() function extracts only the vowel letters from the sequence list.
- Each element of sequence list is passed to filter_vowels() function.
- If filter_vowels() returns True, that element is extracted otherwise it's filtered out.
- **Example 2:**
- Using Lambda Function Inside filter()

```
numbers = [1, 2, 3, 4, 5, 6, 7]
print(list(filter(lambda x:x%2==0,numbers)))
```

- Output:

[2, 4, 6]

- **Example 3:** Using None as a Function Inside filter()

```
l1=[True,18,'virat',45,'rohit',False,0]
print(list(filter(None,l1)))
```

Output:-

```
[True, 18, 'virat', 45, 'rohit']
```

- When None is used as the first argument to the filter() function, all elements that are True values are extracted.

reduce():

It allows us to reduce a list in a shorter way.

Syntax:- **reduce**(function, iterable)

- function: A valid, pre-defined function. This is a lambda function in most cases.
- iterable: This is an iterable object (e.g., list, tuple, dictionary).

- A single value gained as a result of applying the function to the iterable's element.
- Unlike the map() and filter() functions, the reduce() isn’t a built-in function in Python. In fact, the reduce() function belongs to the functools module.
- To use the reduce() function, you need to import it from the functools module using the following statement at the top of the file:

from functools import reduce

Example:

reduce() function to calculate sum of elements of the score list:

```
from functools import reduce

def sum(a,b):

    return a+b

scores = [75, 65, 80, 95, 50]

print(reduce(sum,scores))
```

Example:

To make the code shorter, We can use lambda function instead of defining sum() function

```
from functools import reduce
scores = [75, 65, 80, 95, 50]
print(reduce(lambda a,b:a+b,scores))
```

Marwadi University

Thank You