UNIT –III

5.1 CONTROL STATEMENTS

- ***The for loop***
- ***while statement***
- **if statement**

5.1.1. The for loop

The python for loop is iterator-based for loop.

It goes through the elements in any ordered sequence list, i.e, string, lists, tuples, the keys of dictionary and other iterables.

In each iteration statement, a loop variable is set to a value.

The for loop in python is a bit different from the for loop in any other programming language.

```
>>>for x in y:
        Block1
    else:                          #optional
        Block2
```

The for loop is used to iterate over a sequence.

Hence, x is used to iterate over y and when the loop exits normally then the else part of the for loop execute otherwise not.

Example:1

```
>>> for letter in "python":
        print ("current letter", letter)
```

Output

```
current letter p
current letter y
current letter t
current letter h
current letter o
current letter n
```

Example:2

```
>>>sub=["maths","physics","chemistry","computer"]
        for index in sub:
                print ("current subject:",index)
```

Output

current subject: maths

current subject: physics

current subject: chemistry

current subject: computer

Example:3

```
>>> for x in range(7):
                print(x)
        else:
              print("else part")
```

Output

```
0
1
2
3
4
5
6
else part
```

range() function

The range() function is a built-in function in python that helps us to iterate over the sequence of numbers.

It produces an iterator that follows arithmetic progression.

Example 1:

>>>range(8)

range(8) provides a sequence of numbers 0-7.

ie, range(n) generates a sequence of numbers that starts with 0 and ends with (n-1)

range() function can also be passed with two arguments: begin and end

Example 2:

>>>range(3,9)

The begin index with 3 and end index with 9.

Hence, the range function generates a sequence iterator of numbers that starts from 3 and ends at 9.

Example 3:

```
>>>range(3,40,2)
```

This range( ) function gives us a sequence that starts from 3 and ends at 39; every number in the list has a different of 2.

Example 4:

```
>>> sub=["maths","physics","chemistry","computer"]
>>>  for index in range(len(sub)):
                print ("current subject:",sub[index])
```

Output

```
current subject: maths
current subject: physics
current subject: cheistry
current subject: computer
```

5.1.2. while statement

The while statement is used when have a piece of code and want to repeat it ‘n’ number of times or forever.

With while loop, we have to give conditional statement that tells the interpreter when the loop will halt.

Syntax:

```
>>>while condition:
        Block
    else:                    #optional
        statement
```

Here, condition is a statement by which interpreter decides when to halt the loop and block is the piece of code that we want to repeat.

Example

Write a while statement that prints integers from 0 to 5.

```
>>>count=0
>>>while count<=6:
        print (count)
        count+=1
```

Output

0
1
2
3
4
5
6

5.2. break and continue statements

The break and continue statements are often useful in a while loop as well as in for loop.

The break statement

exits from the loop and

transfers the execution from the loop to the statement that is immediately following the loop.

The continue statement

causes execution to immediately continue at the start of the loop,

it skips the execution of the remaining body part of the loop.

Example:

```
#print first five even numbers
>>> for val in "string":
                if val == "i":
                     break
            else:
                    print(val)

       print("The end")
```

Output

```
s
t
r
The end
```

Example:

```
#print first five even numbers
>>> for val in "string":
        if val == "i":
            continue
        print(val)

    print("The end")
```

Output

s
t
r
n
g
The end

Note:

- If the break statement in a for loop is executed then the else part of that for loop is skipped.
- The break and continue statements are often useful in a while statement.
- The break statement exits from the loop.
- The continue statement causes execution to immediately continue at the start of the loop.

5.3. if statement

5.3.1. if statement

The if statement is known as the decision-making statement in programming languages.

With an if clause, a condition is provided.

if the condition is True then the block of statement written in the if clause will be executed, otherwise not.

Example:

```
>>>var=100
>>>if (var==100):
        print ( “value of expression is 100”)
```

Output

value of expression is 100

5.3.2. if..else statements

An else statement can be combined with an if statement.

It contains the block of code that executes if the conditional expression in the if statement resolves to 0 or False value.

It is an optional statement.

Syntax

```
>>> if  expression:
            Statement1
      else:
            Statement 2
```

Example:

```
n=int(input("ENTER ANY NUMERIC VALUE"))
if n >= 0:
    print("Positive or Zero")
else:
    print("Negative number")
```

Output

```
ENTER ANY NUMERIC VALUE23
Positive or Zero
ENTER ANY NUMERIC VALUE-12
Negative number
```

5.3.3. elif statement

We can check multiple expressions for TRUE with the help of elif statement and execute a block of code written just below the elif statement whose condition is TRUE.

Similar to else, the elif statement is optional.

However, there can be more than one elif statement following an if.

Syntax

```
>>>if expression1:
            Statement1
    elif expression2:
            Statement2
    elif expression3:
            Statement3
    else expression4:
            Statement4
```

Example:

```
n=int(input("ENTER ANY NUMERIC VALUE"))
if n>0:
    print("Positive number")
elif n == 0:
    print("Zero")
else:
    print("Negative number")
```

Output

```
ENTER ANY NUMERIC VALUE20
Positive number
ENTER ANY NUMERIC VALUE0
Zero
ENTER ANY NUMERIC VALUE-23
Negative number
```

5.4. The Alternative Executions

The alternative execution provides two possibilities.

The condition determines which possibility is executed if the condition is TRUE, the first block is executed but if the condition is FALSE, another block of code is executed.

The condition can be either true or false. So, only one alternative will be executed. The alternatives are called branches, because they are branches in the flow of execution.

Syntax

```
>>> x=int(input('Enter the value of x'))
        if  x % 2==0: print ('x is even')
        else:
                print ('x is odd')
```

Output

```
Enter the value of x35
x is odd
Enter the value of x24
x is even
```

5.5. iteration – while statement

A while statement is the command that repeats a piece of code up to several times.

The number of times the piece of code is executed depends on the condition expression written with the while statement.

Syntax

```
>>>while expression:
        Statement(s)
    else:
        Statement
```

- Here, statement(s) may be a single statement or a block of statements.
- The condition may be any expression,
- and whenever the expression resembles a non-zero value, it will be treated as TRUE,
- otherwise it will be treated as FALSE.
- The loop iterates while the condition is TRUE.
- The while loop makes use of an optional else clause which is executed when the condition of while statement fails.

Example:

```
>>>count=0
      while (count<9):
            print("the count is :",count)
            count=count+1
      else:
   print("End of while")
```

Output

```
the count is : 0
the count is : 1
the count is : 2
the count is : 3
the count is : 4
the count is : 5
the count is : 6
the count is : 7
the count is : 8
End of while
```

5.6. input from keyword

The input from keyboard by user plays the most important role in executing a program.

Python language also provides the facility to user to provide input from keyboard.

This can be done in two ways.

5.6.1. input() function

The first function for prompting the input from user is through input() function.

input() function has an optional parameter, which is the prompt string.

when the input() function is called, in order to take input from the user then the execution of program halts and waits for the user to provide an input.

The input is given by the user through keyboard and it is ended by the return key.

input() function interprets the input provided by the user,

ie, if user provides an integer value as input then the input function will return this integer value.

On the other hand, if the user has input a string, then the function will return a string.

Example:

```
>>>name=input(“What is your Name”)
>>>print(“hello”+name)
>>>age=input(“Enter your age”)
>>>print (“age:”+age)
```

Output

```
What is your Nameanu
helloanu
Enter your age25
age:25
```

5.6.2. raw_input () function

The raw_input() is somewhat different from the input() function, the raw_input() function also takes the input from the user but it does not interpret the input and also returns the input of the user without doing any changes.

ie, raw.

Afterwards, we can change this raw input into any data type which is needed for our program.

In order to take input from the user in desired data types.

We can use the casting function with raw_input().

Example:

```
# no casting
>>>age = raw_intput(“What is your age”)
>>>type(age)
<type ‘str’>
#using casitng
>>>age=int(raw_input(“What is your Age”))
>>>type(age)
<type ‘int’>
```

5.7. pass statement

- The pass statement is used as a placeholder for future code.
- When the pass statement is executed, nothing happens, but you avoid getting an error when empty code is not allowed.
- Empty code is not allowed in loops, function definitions, class definitions, or in if statements.

Example 1

Using the pass keyword in a function definition:

```
def myfunction:
        pass
```

Example 2

Using the pass keyword in an if statement:

```
a = 33
b = 200
if  b > a:
        pass
```

5.8 The return statement

- The return statement is used to return value of a function.
- A function may or may not return a value.
- If a function returns a value, it is passed back by the return statement as argument to the caller.
- If it does not return a value, we simply write return with no arguments.

Syntax

return [expression]

Example

#function definition

```
>>>def div(arg1,arg2):
        division = arg1/arg2
        return division
>>>arg3=div(30,5)
>>> print “division:”,arg3
```

Output

6

5.9 Composition

A function can call another is called function composition.

As an example, we'll write a function that takes two points, the center of the circle and a point on the perimeter, and computes the area of the circle.

Assume that the center point is stored in the variables xc and yc, and the perimeter point is in xp and yp.

The first step is to find the radius of the circle, which is the distance between the two points.

radius = distance(xc, yc, xp, yp)

The next step is to find the area of a circle with that radius;

result = area(radius)

Encapsulating these steps in a function,

```
def circle_area(xc, yc, xp, yp):
radius = distance(xc, yc, xp, yp)
result = area(radius)
return result
```

The temporary variables radius and result are useful for development and debugging

composing the function calls:

```
def circle_area(xc, yc, xp, yp):
return area(distance(xc, yc, xp, yp))
```

5.10 . Python Recursive Function

If a function, procedure or method calls itself, it is called recursive
.
Example:

```
>>>def fact_rec(x):
        if x==0:
            return 1
        else:
            return (x* fact_rec(x-1))

 >>>fact_rec(4)
24
>>>fact_rec(10)
3628800
```

5.11 The Anonymous Functions

The anonymous function is the functions created using lambda keyword.

They are not defined by using def keyword.

For this reason, they are called anonymous function.

We can pass any number of arguments to a lambda form function.

But still they return only one value in the form of expression.

An anonymous function cannot directly call print command as the lambda needs an expression.

It cannot access the parameters that are not defined in its own namespace.

An anonymous function is a single line statement function.

Syntax

lambda [arg1,[arg2,arg3,…..]]:expression

The Anonymous Functions

Example:

#function definition

>>>mult=lambda val1,val2: val1* val2

#function call

>>>print “Value:”, mult(20,40)

Output

Value:800

In function call, we can directly call the mult function with two valid values as arguments and produce the output.

5.12 Variables and parameters are local

When you create a variable inside a function, it is **local**, which means that it only exists inside the function.

For example:

```
def cat_twice(part1, part2):
        cat = part1 + part2
print_twice(cat)
```

This function takes two arguments, concatenates them, and prints the result twice.

References

1. Allen B Downey. “Think Python: How to Think like a Computer Scientist”, Green Tea Press,version 2.0.17.
2. E. Balagurusamy , “Introduction to Computing and Problem Solving Using Python”,Mc Graw hill education.
3. Guido van Rossum and Fred L. Drake Jr, “An Introduction to Python – Revised and updated for Python 3.2”, Network Theory Ltd., 2011.
4. John V Guttag, “Introduction to Computation and Programming Using Python”, Revised and expanded Edition, MIT Press , 2013
5. www.w3school.com
6. https://www.programiz.com/python-programming.
7. https://www.tutorialspoint.com