Python Programming

by Narendra Allam

Chapter 3

Control Structures

Topics Covering

Control structures:

- if statement
- if - else statement
- if - elif statement
- Nest if-else
- Multiple if
- Which control structure to choose?
- Looping statements
  - while loop
  - for loop
    - range()
    - xrange()
    - Iterator and generator Introduction
  - for - else
  - When to use for-else ?
- Interview Questions
- Exercise Programs
- Summary

Control Structures

Control structures are used to control the flow of execution. Some, times we may require to skip the execution of some statements. Some times we may want to execute two blocks of statements exclusively. Some times may need to execute a block of statements repeatedly. Below are the control statments we are going to discuss now.

1. if
2. if-else
3. if-elif ladder
4. multiple if
5. nested if-else
6. while loop
7. for loop

if statement:

Syntax:

```
    if condition:
        statements
```

Program: Read a number from keyboard, and if the number is even, print 'EVEN'.

In [3]:

```
x = int(input("Enter x:"))

if x%2 == 0:
    print ('----')
    print ("Even")
    print ('----')

print('The End')
```

```
Enter x:12
----
Even
----
The End
```

If the given number is not even, we do not see any ouput.

Identation: Python identifies code-blocks using indentation. In the above program, we can see that, all the statatements which needs to be excuted when condition is True, should be placed after 'if condition:' preceeded by one tab(generally 4 spaces). All the statments after if statment with one tab indentation, are considered as if block. lines 4, 5, 6 belong to if block, but not line 8

if-else statement:

When we want to execute two blocks of code exclusively we use if-else statement. Only one block of code is executed all the time.

Syntax:

```
    if condition:
        # statements...
    else:
        #statements...
```

Program: Read a number from keyboard, and if the number is even, print 'EVEN' else print 'ODD'.

In [4]:

```
x = int(input("Enter x:"))

if x%2 == 0:
    print("Even")
else:
    print("Odd")

print('The End')
```

```
Enter x:15
Odd
The End
```

if-elif ladder - more exclusive cases

When we have mulitple exclusive cases, we use if-elif ladder.

Syntax:

```
if condition:
    # statements...
elif condition:
    # statments...
elif condition:
    # statments...

else:
    # statments
```

Program: Read a character from keyboard. If the given input is,

'a' - print 'Apple'
'b' - print 'Ball'
'c' - print 'Cat'
'd' - print 'Dog'
Any other character, print 'Unknown'

In [5]:

```
x = int(input("Enter x:"))

if x < 0:
    print("-ve")
elif x == 0:
    print("Neutral")
else:
    print("+ve")

print('The End')
```

```
Enter x:0
Neutral
The End
```

In [6]:

```
char = input("Enter character:")

if char == 'a':
    print ("Apple")
elif char == 'b':
    print ("Ball")
elif char == 'c':
    print ("Cat")
elif char == 'd':
    print ("Dog")
else: # if non of the ifs are true, else gets executed
    print ("Unknown")

print ('The End') # This is just to know that program ended.
```

```
Enter character:k
Unknown
The End
```

Mutiple if: a special case

Eventhough it is not a seperate control statement, there is a special use case for this. When the given input falls under multiple categories we use mutiple ifs.

Program: Which of the following numbers [3, 7, 11] are the factors for a given number

In [7]:

```
x = int(input("Enter x:"))

if x%3 == 0 :
    print("3 is a factor")

if x%7 == 0 :
    print("7 is a factor")

if x%11 == 0 :
    print("11 is a factor")
```

```
Enter x:33
3 is a factor
11 is a factor
```

Nested if-else: one more special case

Like multiple if, nested if-else is also a basic control structure it has a special use-case. When we have multiple sub-categories under a condition we use nesed if-else.

Program: Read a number from keyboard,
if the number is even square it
if it is odd multiple of 5 cube it
if it is odd non-multiple of 5 mutiply with 10

In [ ]:

```
x = int(input("Enter x:"))

if x%2 == 0:
    print (x**2)
else:
    if x%5 == 0:
        print (x**3)
    else:
        print (x*10)
```

Looping statements

When a block of statements is required to be executed, repeatedly, we use looping staments. There are three looping statements in python.

1. while and while-else
2. for and for-else

While loop:

As log as condition is true, block of staments under while are executed repeatedly. Once condition is false, loop stops exection and control goes immediate next stament after while block.

Syntax:

```
while condition:
    statements
```

Program: Print first 10 natural numbers.

In [12]:

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

```
1
2
3
4
5
6
7
8
9
10
```

Observe the behaviour of above while loop carefully. 1 is the start value of 'i', and then checking the condition, 'i <= 10'. I if it is True, conteol is going inside the loop. Inside while, 'i' is getting incremented by 1(step) in each iteration of the loop. Actually, 'i' is moving towards the end value 10. start, end and step are the three factors on which loop execution depends. Now, we can understand how string slicing is wroking in python.

Program: Print all the numbers which end with 3 below 100.

In [13]:

```
i = 1
while i <= 100:
    if i%10 == 4 and i%7 == 0:
        print(i)
    i += 1
```

```
14
84
```

Exercises for non-programmers

Program: Print all the numbers which end with 3 and multiples of 3 below 100.

In [16]:

```
i = 1
while i <= 100:
    if i%10 == 3 and i%3 == 0:
        print (i)
    i += 1
```

```
3
33
63
93
```

Program: Print the count of the numbers which end with 3 and multiples of 3 below 100.

In [17]:

```
i = 1
c = 0
while i <= 100:
    if i%10 == 3 and i%3 == 0:
        c += 1
    i += 1

print ("Count: ", c)
```

```
Count:  4
```

In [18]:

```
i = 1
s = 0
while i <= 100:
    if i%10 == 3 and i%3 == 0:
        s += i
    i += 1

print ("Sum: ", s)
```

```
Sum:  192
```

Program: Count number of multiples of 7 from 100 to 135.

In [19]:

```
i = 100
c = 0
while i <= 135:
    if i%7 == 0:
        c += 1
    i += 1
print ("Number of multiples of 7 between 100 and 135: ", c)
```

```
Number of multiples of 7 between 100 and 135:  5
```

Program: Sum of the multiples of 3 below 100

In [20]:

```
i = 1
s = 0
while i <= 100:
    if i%3 == 0:
        s += i
    i += 1
print ("sum of the multiples of 3 below 100 : ", s)
```

```
sum of the multiples of 3 below 100 :  1683
```

Program: Multiples of n below 100.

In [21]:

```
n = int(input("Enter n value:"))
i = 1
while i <= 100:
    if i%n == 0:
        print (i)
    i+=1
```

```
Enter n value:17
17
34
51
68
85
```

Program: Printing factors of a given number.

In [22]:

```
n = int(input("Enter n:"))
i = 1
while i <= n:
    if n%i ==0:
        print(i)
    i+=1
```

```
Enter n:12
1
2
3
4
6
12
```

Program: check the given number is prime or not (Version 1)

In [43]:

```
n = int(input("Enter n:"))
i = 2
c = 0
rt = n**0.5
while i <= rt:
    if n%i ==0:
        c += 1
    i+=1

if c == 0:
    print ('Prime')
else:
    print ('Not Prime')
```

```
Enter n:17
Prime
```

Using **break:**

In the above program, if condition at line 6, is true, that means we can tell the number is not prime, and we do not need to continue remaining iterations of the loop. this is where we can use **break** which breaks the current loop and control jumps to the immedeate next statement after while.

In [1]:

```
n = int(input("Enter n:"))
i = 2

rt = n**0.5
while i <= rt:
    if n%i ==0:
        print ('Not Prime')
        break
    i+=1

if i > rt:
    print ('Prime')
```

```
Enter n:17
Prime
```

Program: check the given number is prime or not (Version 2) - using while-else

In [40]:

```
n = int(input("Enter n:"))
i = 2

rt = n**0.5
while i <= rt:
    if n%i ==0:
        print ('Not Prime')
    i+=1
else:
    print ('Prime')
```

```
Enter n:17
Prime
```

Note: *else:* will be executed after successfull completion of while loop. While loop should complete all of its iterations to reach else. If break is used , else will not be executed.

Program: Print prime numbers between 501 to 600.

In [27]:

```
n = 501
while n <= 600:
    i = 2

    rt = n**0.5
    while i <= rt:
        if n%i ==0:
            break
        i+=1
    else:
        print (n)

    n += 1
```

```
503
509
521
523
541
547
557
563
569
571
577
587
593
599
```

Program: Print the digits in the given number in reverse order.

In [28]:

```
n = int(input("Enter x:"))

while n > 0:
    r = n%10
    print (r, end='')
    n //= 10
```

```
Enter x:1234
4321
```

Program: Print count of the digits in the given number.

In [31]:

```
n = int(input("Enter x:"))
c = 0
b = n
while n > 0:
    n //= 10
    c += 1

print ('Number of digits in {} is {}'.format(b, c))
```

```
Enter x:45321
Number of digits in 45321 is 5
```

Program: Sum of the digits in the number

In [32]:

```
n = int(input("Enter x:"))
b = n
s = 0

while n > 0:
    r = n%10
    n //= 10
    s += r

print ('Sum of the digits in {} is {}'.format(b, s))
```

```
Enter x:31425
Sum of the digits in 31425 is 15
```

Program: Magic number of the given number. Sum all the digits, until you get single digit sum.

In [34]:

```
n = int(input("Enter x:"))
b = n
s = n

while s > 9:
    n = s
    s = 0
    while n > 0:
        r = n%10
        s += r
        n //= 10

print ('Magic number of {} is {}'.format(b, s))
```

```
Enter x:51234
Magic number of 51234 is 6
```

Program: Reverse the given number.

In [35]:

```
n = int(input("Enter x:"))
b = n
rev = 0

while n > 0:
    r = n%10
    rev = rev*10 + r
    n //= 10

print ('reverse of {} is {}'.format(b, rev))
```

```
Enter x:1234
reverse of 1234 is 4321
```

Program: Printing number of factors of a given number

In [36]:

```
n = int(input("Enter n:"))
i = 1
c = 0
while i <= n:
    if n%i ==0:
        c += 1
    i+=1
print ("Number of factors of {} is {}".format(n, c))
```

```
Enter n:12
Number of factors of 12 is 6
```

Program: Check the given number is perfect or not/ A Perfect number is a number whose sum of the factors(exclusing iteslef) should be equal to itself.

In [37]:

```
n = int(input("Enter n:"))
i = 1
s = 0
while i <= n//2:
    if n%i ==0:
        s += i
    i+=1

if s == n:
    print ("Perfect")
else:
    print ("Not perfect")
```

```
Enter n:6
Perfect
```

Exercise: Print all perfect numbers below 100

for loop:

Unlike other programming languages, for loop in python completely works on iteration protocol. iterable in the below syntax should provide a next value in each iteration to iter_var.

Syntax:

```
for iter_var in iterable:
    statements
```

In [2]:

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

```
1
2
3
4
5
6
7
8
9
10
```

In [3]:

```
range(10)
```

Out[3]:

```
range(0, 10)
```

Using range() function in for loops

range() is a function which produces a value in each iteration in the range given.

Syntax:

```
range(start, end, step)
```

Some examples of using range() function:

In [7]:

```
for x in range(5): # if we pass one value to range, it treats it as 'end'. default
    print(x)
```

```
0
1
2
3
4
```

In [8]:

```
for x in range(1, 6): # if we pass two values to range, asumes and 'start' and 'end
    print(x)
```

```
1
2
3
4
5
```

In [9]:

```
for x in range(1, 11, 2):
    print(x)
```

```
1
3
5
7
9
```

Program: Printing first 10 natural numbers using for

In [10]:

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

```
1
2
3
4
5
6
7
8
9
10
```

Note: for loop is also used to iterate on any sequence type like string etc

In [15]:

```
s = 'Apple'
for x in s:
    print(x)
```

```
A
p
p
l
e
```

In [16]:

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

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

This is equivalent to below while loop.

In [13]:

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

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

In [17]:

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

```
1
2
3
4
5
```

Below loop will not print anything as end=1, which is exclusive.

In [18]:

```
for i in range(1, 1):
    print('*')
```

If we want to iterate the loop one time, 'end' must be 2 for the below loop.

In [19]:

```
for i in range(1, 2):
    print('*')
```

```
*
```

In [24]:

```
for i in range(4):
    for j in range(3):
        print("Apple")
```

```
Apple
Apple
Apple
Apple
Apple
Apple
Apple
Apple
Apple
Apple
Apple
Apple
```

Printing in the same line

Each print stament takes the control to the net line. If you to keep the control in the same line, use end='' as the last argument of the print statement line.

In [20]:

```
print('Apple')
print('Orange')
```

```
Apple
Orange
```

In [21]:

```
print('Apple', end='')
print('Orange')
```

```
AppleOrange
```

Program: Print the below trianlge using for loop.

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

In [26]:

```
for i in range(1, 6):
    for j in range(1, i+1):
        print('*', end='')
    print()
```

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

In [25]:

```
for i in range(1, 6):
    for j in range(1, i+1):
        print(i, end=' ')
    print()
```

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

In [33]:

```
for i in range(5, 0, -1):
    for j in range(1, i+1):
        print(i, end=' ')
    print()
```

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

Exercises for non-programmers

Program: Print the below trianlge using for loop.

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

In [32]:

```
for i in range(1,6):
    for j in range(1, i+1):
        print(i, end=' ')
    print()
```

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

Program: Print the below trianlge using for loop.

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

In [31]:

```
for i in range(1,6):
    for j in range(1, i+1):
        print(j, end=' ')
    print()
```

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

Program: Print the below trianlge using for loop.

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

In [30]:

```
for i in range(5, 0, -1):
    for j in range(5, i-1, -1):
        print(i, end=' ')
    print()
```

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

Program: Print the below trianlge using for loop.

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

In [29]:

```
for i in range(5, 0, -1):
    for j in range(5, i-1, -1):
        print(j, end=' ')
    print()
```

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

Program: Print the below trianlge using for loop.

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

In [28]:

```
c = 1
for i in range(1, 6):
    for j in range(1, i+1):
        print(c, end=' ')
        c += 1
    print()
```

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

break:

break statement breaks the execution of while or for loops, control trnasfered to immediate next statement of the loop. Refer the above example.

In [ ]:

```
# Using break
n = int(raw_input("Enter Value:"))
rt = int(n ** 0.5)
i = 2
while i <= rt:
    if n%i == 0:
        print("Not Prime")
        break
    i += 1
else:
    print ("Prime")
```

In [ ]:

```
# Using break
n = int(raw_input("Enter Value:"))
rt = int(n ** 0.5)

for i in range(2, rt+1):
    if n%i == 0:
        print "Not Prime"
        break

if i == rt:
    print "Prime"
```

for-else

**IQ. 'else' block gets executed on the successfull completion of for loop

In [ ]:

```

n = int(input("Enter Value:"))

rt = int(n**0.5)

for i in range(2, rt+1):
    if n%i == 0:
        print ("Not Prime")
        break
else:
    print ("Prime")
```

Program: Print the prime numbers between 50 to 100.

In [2]:

```
for n in range(50, 100+1):
    rt = int(n**0.5)
    for i in range(2, rt+1):
        if n%i == 0:
            break
    else:
        print (n)
```

```
53
59
61
67
71
73
79
83
89
97
```

continue:

continue skips the execution to next iteration of the loop.

Program:
Sum all the elements in the range 1 to 10, which are non-multiples of 3

In [3]:

```
s = 0
for x in range(1, 11):
    if x%3 == 0:
        continue
    s += x

print ("sum of non-multiples of 3: ", s)
```

```
sum of non-multiples of 3:  37
```

Interview Questions

1. How do you swap two values?
2. range() vs xrange() ?
3. How for-else construct works?

Practice programs

Program 1: A Company insures its drivers in the following cases

1. If the driver is married
2. If the driver is unmarried, male and above 30 years of age

3. If the driver is unmarried, female and above 25 years of age
   in all other cases, the driver is not insured. If the marital status, gender, age of the drivers are the inputs, write a program to determine whether the driver is insured or not.(use 'nested - if')

Solution:

In [35]:

```
marrital_status = input('Enter marrital status[married/unmarried]:')
gender = input('Enter gender[male/female]:')
age    = int(input('Enter age:'))

if marrital_status == 'marreid':
    print ('Inssured')
else:
    if (gender == 'male' and age >= 30) or (gender == 'female' and age >= 25):
        print ('Inssured')
    else:
        print ('Not Inssured')
```

```
Enter marrital status[married/unmarried]:married
Enter gender[male/female]:male
Enter age:23
Not Inssured
```

Program 2: Indian income tax calcuation:
0% tax - upto 3.0 lac 5% tax - till 5.0 lac
20% tax - till 10 lac
30% tax - for above 10 lac

In [36]:

```
actual_salary = float(input('Enter salary:'))

tax = 0.0
salary = actual_salary

if salary > 1000000:
    slab = salary - 1000000
    tax = slab * 0.3
    salary -= slab

if salary > 500000:
    slab = salary - 500000
    tax += slab * 0.2
    salary -= slab

if salary > 300000:
    slab = salary - 300000
    tax += slab * 0.05
    salary -= slab


print('tax for {} is {} if gender is {}'.format(actual_salary, tax, gender))
```

```
Enter salary:1350000
tax for 1350000.0 is 215000.0 if gender is male
```

Program 3: Write a programe to read the characters continuosly until '$' is given and display the number of

characters entered.

Program 4: Write a programe to read a character and find out whether it is uppercase or lowercare

Solution:

In [37]:

```
ch = input('Enetr a charcater:')

if ch.isdigit():
    print('Digit Character')
elif ch.islower():
    print('Lower case Character')
elif ch.isupper():
    print('Upper case Character')
else:
    print('Special Character')

print('The End')
```

```
Enetr a charcater:*
Special Character
The End
```

Programe 5: Write a program to print the uppercase letter of a given lowercase

Program 6: Write a program to check whether the given input is digit or lowercase character or uppercase character or a special character(use 'if-else-if' lasser)

Program 7: Write a program to print all prime numbers between 100 to 200 using only for and for-else looping statements.