



Jawaharlal Nehru Technological University Hyderabad

Kukatpally, Hyderabad - 500 085, Telangana, India

PYTHON PROGRAMMING

Conditions and Loops

Session 2 , 4 May 2023

Dr N V Ganapathi Raju
Professor and HOD of IT
GRIET

Introduction to Branching and Looping

- Generally, Programs contain set of Statements.
- Set of statements, gets executed sequentially in the order in which they are written and appear.
- But, situations may arise where we may have to change the order of execution of statements depending on specific conditions.
- This involves a kind of decision making from a set of logical conditions/tests.
- Decision structures is to evaluate one or multiple expressions/logical conditions, which return TRUE or FALSE outcomes.
- We can then determine what actions the program should take by defining the statements to execute when the outcome is TRUE or FALSE.

Branching Statements

- Python has three major decision making instructions—

if statement

if-else statement

if...elif...else statement

Branching: if statement

- The **if** statement is used to carry out a logical expression, returns Boolean.



- The expression in parentheses must produce a Boolean result
- At run time, the computer evaluates the expression as True/False
- If True, the computer executes Statement(s).
- If False, goes out of if condition
- **if** keyword and ends with a colon (:)
- In Python, the *if* block statements are determined through indentation and the first un-indented statement marks the end.

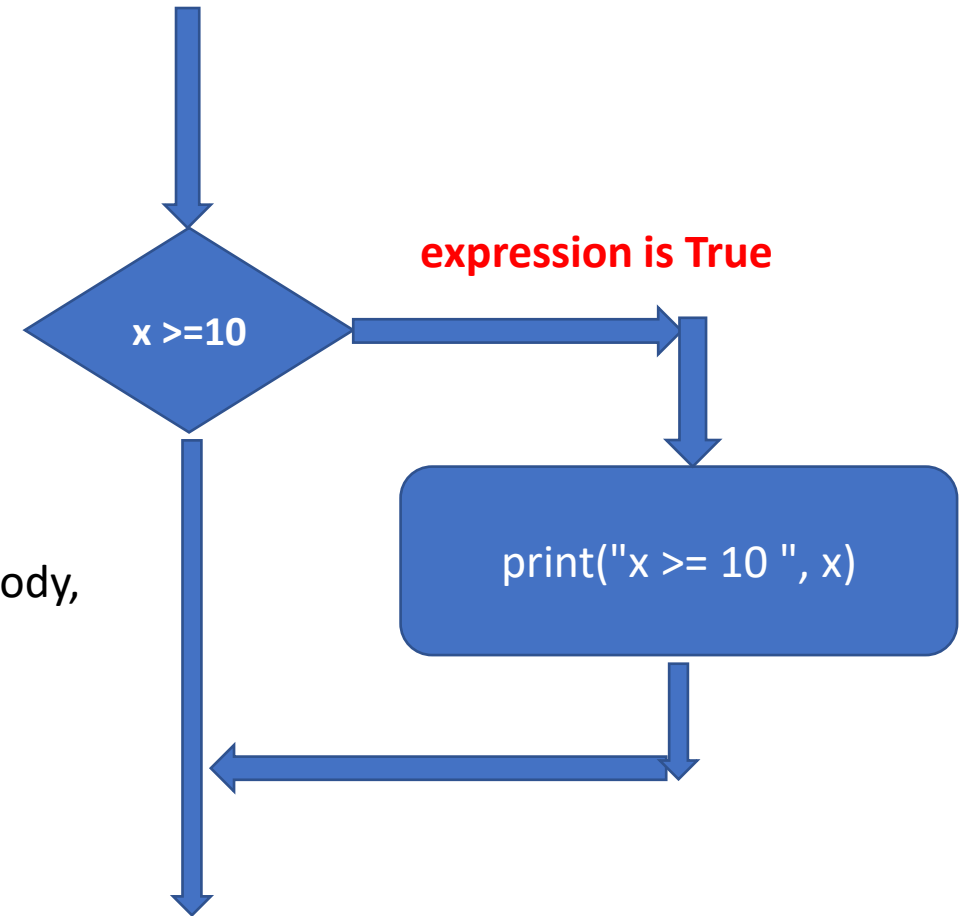
Example on if statement

- If the expression returns True, prints statements.

```
x = 10
if (x >= 10 ):
    print("x >= 10 ",x)
```

Note:

- There is no limit on the number of statements that can appear in the body, but there must be at least one.
- It is useful to have a body with no statements
- Use the **pass** statement, which does nothing.



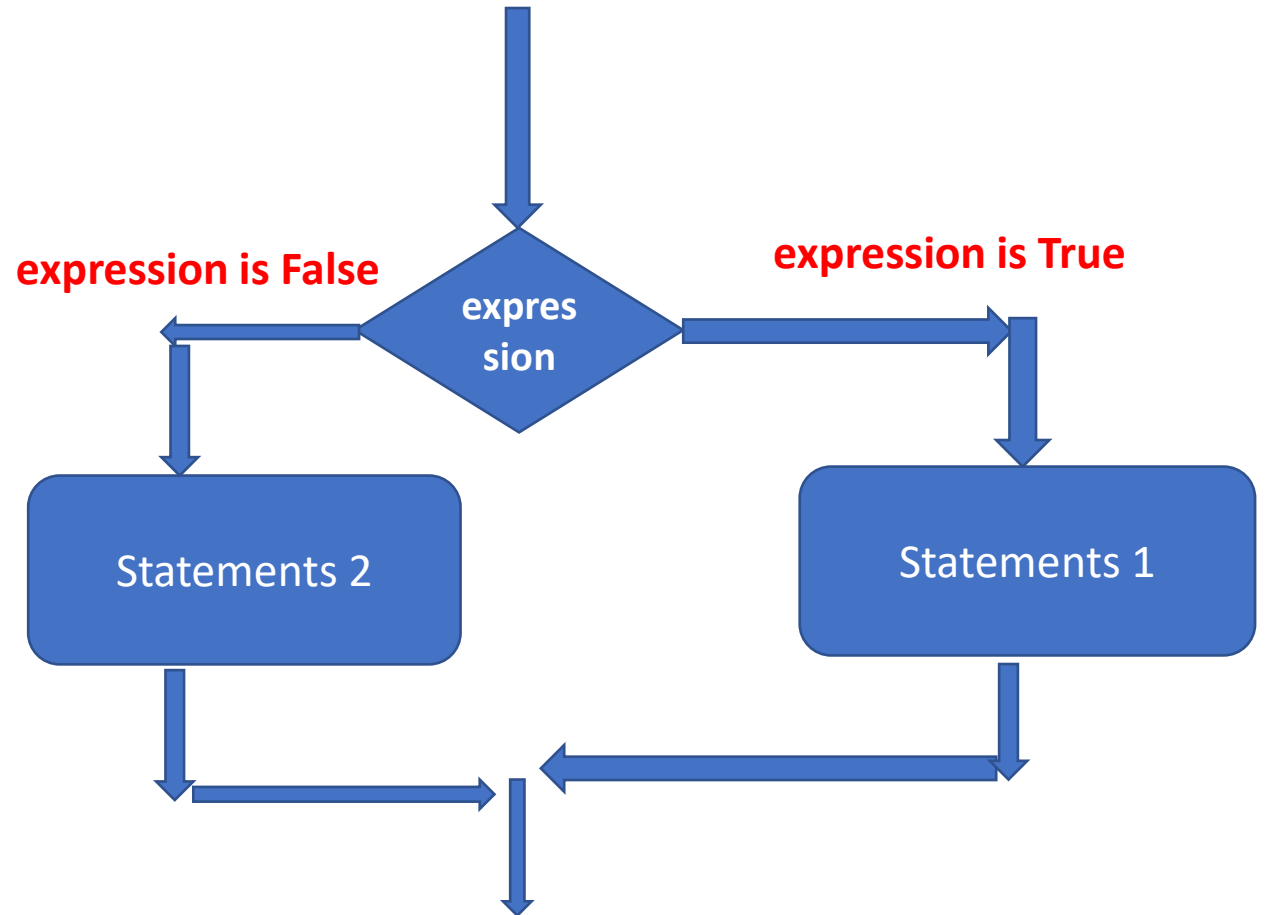
if...else... statement

- A second form of the if statement in which there are two possibilities and the condition determines which one gets executed.

```
if ( expression ) : column  
    statement(s) 1 ; True  
else: column  
    statement(s) 2 ; False
```

indentation →

indentation →



Programs on if...else...

- **Program to Find If a Given Number Is Odd or Even**

```
x = 10
if x%2 == 0 :
    print('x is even', x)
else :
    print('x is odd', x)
```

- **Program to Find the Greater of Two Numbers**

```
x, y=10,20
if (x > y):
    print("x is big",x)
else:
    print("y is big",y)
```

if...elif...else statement (multy-way control)

- **When you need to choose from several possible alternatives, then an *elif* statement is used along with an *if* statement.**

```
If expression 1:  
    statement 1  
elif expression 2:  
    statement 2  
    :  
    :  
    :  
else:  
    statement last
```

If expression 1 is *True*, then statement_1 is executed.

If expression 1 is *False* and expression 2 is *True*, then statement 2 is executed.

If none of the expression is *True*, then statement last is executed.

Program using if...elif...else statement

- Program to find whether a given number is greater than / less than / equal to another.

```
x = int(input("enter first number:"))
y = int(input("enter second number:"))

if (x > y):
    print("x > y")
elif (x < y):
    print("x < y")
else:
    print("x == y")
```

```
1 x = int(input("enter first number:"))
2 y = int(input("enter second number:"))
3
4 if (x > y):
5     print("x > y")
6 elif (x < y):
7     print("x < y")
8 else:
9     print("x == y")
```

```
enter first number:20
enter second number:10
x > y
```

for loop

- Probably the most popular looping instruction.

```
for iteration_variable in sequence:  
    statement(s)
```

- The *for* loop starts with *for* keyword and ends with a colon.
- The first item in the sequence gets assigned to the iteration variable *iteration_variable*. *iteration_variable* can be any valid variable name. Then the statement block is executed.
- This process of assigning items from the sequence to the *iteration_variable* and then executing the statement continues until all the items in the sequence are completed.
- Actually, the *for* loop is designed to do more complicated tasks - it can "browse" large collections of data item by item.

range()

- The range() type returns an immutable sequence of numbers between the given start integer to the stop integer.
- range() constructor has two forms of definition:
 - range(stop)
 - range(start, stop[, step])
 - start - integer starting from which the sequence of integers is to be returned
 - stop - integer before which the sequence of integers is to be returned. The range of integers end at stop - 1.
 - step (Optional) - integer value which determines the increment between each integer in the sequence
- range() returns an immutable sequence object of numbers depending upon the definitions used:

Example for loop with range()

```
for a in range (5):  
    print (a)
```

```
1 for a in range (5):  
2     print (a)
```

```
0  
1  
2  
3  
4
```

```
for a in range (6,10):  
    print (a)
```

```
1 for a in range (6,10):  
2     print (a)
```

```
6  
7  
8  
9
```

```
for a in range (11,20,2):  
    print (a)
```

```
1 for a in range (11,20,2):  
2     print (a)
```

```
11  
13  
15  
17  
19
```

for loop with string

- Program to Iterate through Each Character in the String

```
for ch in "Python":  
    print(ch)
```

Example for loop

- Program : Sum of all first 10 Natural numbers

```
sum=0
for n in range(1,11):
    sum+=n
print (sum)
```

```
: 1 sum=0
   2 for n in range(1,11):
   3     sum+=n
   4 print (sum)
```

55

Example for loop

- Program to Find the Sum of All Odd and Even Numbers

```
num = int(input("Enter a number: "))  
even = 0  
odd = 0  
for i in range(num):  
    if i % 2 == 0:  
        even = even + i  
    else:  
        odd = odd + i  
print(f"Sum of Even are {even} odd are {odd}")
```

```
1 num = int(input("Enter a number: "))  
2 even = 0  
3 odd = 0  
4 for i in range(num):  
5     if i % 2 == 0:  
6         even = even + i  
7     else:  
8         odd = odd + i  
9 print(f"Sum of Even are {even} odd are {odd}")
```

Enter a number: 10
Sum of Even are 20 odd are 25

while loop

- The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.
- Condition: We generally use this loop when we don't know the number of times to iterate beforehand.

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

```
# Program to add natural # numbers up to
```

```
n = 10  
sum = 0  
i = 1
```

```
while i <= n:  
    sum = sum + i  
    i = i+1
```

```
print("The sum is", sum)
```


break , continue statements

- In Python, break and continue statements can alter the flow of a normal loop.
- Loops iterate over a block of code until the test expression is false, but sometimes we wish to terminate the current iteration or even the whole loop without checking test expression.
- The **break** statement terminates the loop containing it.
- If the **break** statement is inside a nested loop (loop inside another loop), the break statement will terminate the innermost loop.
- 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 with the next iteration.

Example with break , continue statements

```
for i in range(1, 6):  
    if i == 3:  
        break  
    print("Inside the loop.", i)  
print("Outside the loop.")
```

```
1 for i in range(1, 6):  
2     if i == 3:  
3         break  
4     print("Inside the loop.", i)  
5 print("Outside the loop.")
```

Inside the loop. 1
Inside the loop. 2
Outside the loop.

```
for i in range(1, 6):  
    if i == 3:  
        continue  
    print("Inside the loop.", i)  
print("Outside the loop.")
```

```
1 for i in range(1, 6):  
2     if i == 3:  
3         continue  
4     print("Inside the loop.", i)  
5 print("Outside the loop.")
```

Inside the loop. 1
Inside the loop. 2
Inside the loop. 4
Inside the loop. 5
Outside the loop.

else statement

- With the **else** statement we can run a block of code once when the condition no longer is true:

```
i = 1
```

```
while i < 6:
```

```
    print(i)
```

```
    i += 1
```

```
else:
```

```
    print("i is no longer less than 6")
```

```
for i in range(0, 10,2):
```

```
    print(i, end=" ")
```

```
else:
```

```
    print("\n No Break")
```

In Python, a nested loop is a loop inside another loop

Syntax:

Outer loop:

Inner loop:

statements of inner loop

statements of outer loop

```
for loop:  
    for loop:  
        statements of for loop  
statements of for loop
```

```
while loop:  
    while loop:  
        statements of while loop  
statements of while loop
```

```
for loop:  
    while loop:  
        statements of while loop  
statements of for loop
```

```
while loop:  
    for loop:  
        statements of for loop  
statements of while loop
```

Single line if else statements

```
if condition:  
    value_when_true  
else:  
    value_when_false
```

```
value_when_true if condition else value_when_false
```

- If condition returns True then value_when_true is returned
- If condition returns False then value_when_false is returned

```
b = int(input("Enter value for b: "))  
  
a = "positive" if b >= 0 else "negative"  
  
print(a)
```

```
b = int(input("Enter value for b: "))  
  
if b >= 0:  
    a = "positive"  
else:  
    a = "negative"  
  
print(a)
```