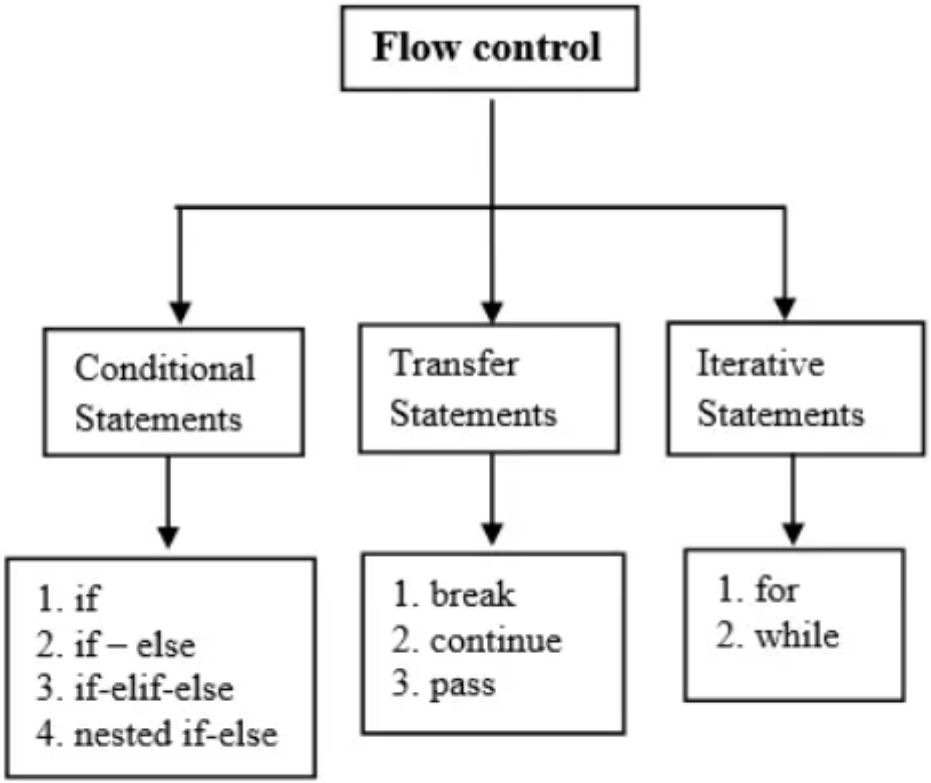
Control Flow Statements

The flow control statements are divided into three categories

1. Conditional statements
2. Iterative statements.
3. Transfer statements



Python control flow statements

Conditional statements

In Python, condition statements act depending on whether a given condition is true or false. You can execute different blocks of codes depending on the outcome of a condition. Condition statements always evaluate to either True or False.

There are three types of conditional statements.

1. if statement
2. if-else
3. if-elif-else
4. nested if-else

Iterative statements

In Python, iterative statements allow us to execute a block of code repeatedly as long as the condition is True. We also call it a loop statements.

Python provides us the following two loop statement to perform some actions repeatedly

[1].for loop [2].while loop

Transfer statements

In Python, transfer statements are used to alter the program's way of execution in a certain manner. For this purpose, we use three types of transfer statements.

1. break statement
2. continue statement
3. pass statements

If statement in Python

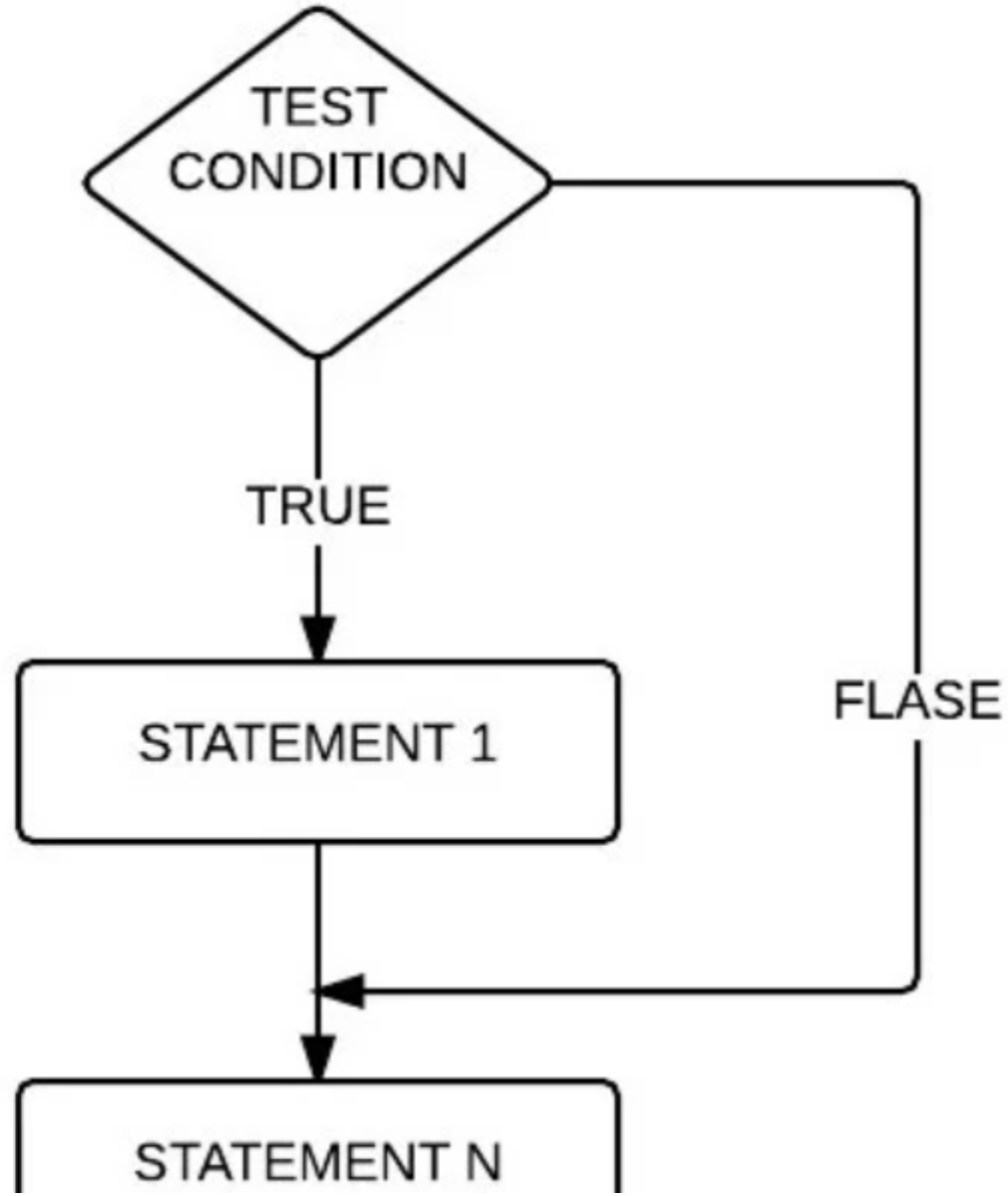
In control statements, The if statement is the simplest form. It takes a condition and evaluates to either True or False.

If the condition is True, then the True block of code will be executed, and if the condition is False, then the block of code is skipped, and The controller moves to the next line

Syntax of the if statement:

```
if condition:
     statement 1
     statement 2
     statement n
```

Python If Statement Flow Chart



we will calculate the square of a number if it greater than 5

In [1]:
```
# Example :
number = 6
if number > 5:
    # Calculate square
    print(number * number)
print('Next lines of code')
```

```
36
Next lines of code
```

In [2]:
```
# Example :

# If the number is positive, we print an appropriate message

num = 3
if num > 0:
    print(num, "is a positive number.")
print("This is always printed.")

num = -1
if num > 0:
    print(num, "is a positive number.")
print("This is also always printed.")
```

```
3 is a positive number.
This is always printed.
This is also always printed.
```

- In the above example, num > 0 is the test expression.
- The body of if is executed only if this evaluates to True.
- When the variable num is equal to 3, test expression is true and statements inside the body of if are executed.
- If the variable num is equal to -1, test expression is false and statements inside the body of if are skipped.

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

In [3]:
```
# Example : if condition
price = 50
if price < 100:
    print("price is less than 100")
```

```
price is less than 100
```

In the above example, the expression price < 100 evaluates to True, so it will execute the block. The if block starts from the new line after : and all the statements under the if condition starts with an increased indentation, either space or tab. Above, the if block contains only one statement. The following example has multiple statements in the if condition.

In [4]:
```
# Example : multiple statments in if block

price = 50
quantity = 5
if price*quantity < 600:
    print("price * quantity is less than 600")
    print("price", price)
    print("quantity",quantity)
```

```
price * quantity is less than 600
price 50
quantity 5
```

Above, the if condition contains multiple statements with the same indentation. If all the statements are not in the same indentation, either space or a tab then it will raise an IdentationError.

In [5]:
```
# Example: Invalid Indentation in the Block
price = 50
quantity = 5
if price*quantity < 600:
    print("price * quantity is less than 600")
    print("price", price)
     print("quantity",quantity)
```

```
  File "<ipython-input-5-795827ea4981>", line 7
    print("quantity",quantity)
    ^
IndentationError: unexpected indent
```

Note : The statements with the same indentation level as if condition will not consider in the if block. They will consider out of the if condition.

In [ ]:
```
# Example: Out of Block Statements
price = 50
quantity = 5
if price*quantity < 100:
    print("price * quantity is less than 600")
    print("price", price)
    print("quantity",quantity)
print("No if block is executed")
```

The following example demonstrates multiple if conditions.

In [ ]:
```
price = 100
if price > 100:
    print("price is greater than 100")
if price ==100:
    print("price is 100")
if price < 100:
    print("price is less than 100")
```

Note : Notice that each if block contains a statement in a different indentation, and that's valid because they are different from each other.

If – else statement

The if-else statement checks the condition and executes the ‘if’ block of code when the condition is True, and if the condition is False, it will execute the else block of code.

```
if condition:
    statement 1
else:
    statement 2
# If the condition is True, then statement-1 will be executed If the condition is False, statement-2 will be executed. See the following flowchart for more detail.
```

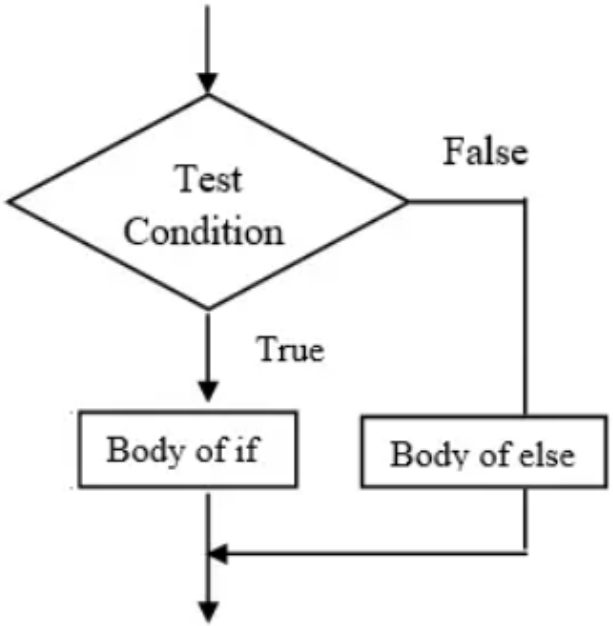


Fig. Flowchart of if-else

Python if else statements

In [ ]:
```
 # Take values of length and breadth of a rectangle from user and check if it is square or not.
print ("Enter length")
length = input()
print ("Enter breadth")
breadth = input()
if length == breadth:
    print ("Yes, it is square")
else:
    print ("No, it is only Rectangle")
```

In [ ]:
```
i = 10
if (i < 20):
    print("i is smaller than 10")
else:
    print(" i is greater than 10")
```

In [ ]:
```
# A company decided to give bonus of 5% to employee if his/her year of service is more than 5 years.
# Ask user for their salary and year of service and print the net bonus amount.

print("Enter salary")
salary = input()
print("Enter year of service")
yos = int(input())

if (yos> 5):
    print(" Bonus is",.05*salary)
else:
    print(" No Bonus")
```

In [ ]:
```

```

Chain multiple if statement in Python

In Python, the if-elif-else condition statement has an elif keyword used to chain multiple conditions one after another. The if-elif-else is useful when you need to check multiple conditions.

With the help of if-elif-else we can make a tricky decision. The elif statement checks multiple conditions one by one and if the condition fulfills, then executes that code.

Syntax of the if-elif-else statement:

```
if condition1:
    statment1
elif condition2:
    statment2
elif condition3:
```

```
    statment3
    ....
else:
    statment

```

In [6]:

```
# Take two int values from user and print greatest among them.

print("Enter first number")
first = input()
print(" Enter second number")
second = input()

if first > second:
    print("Greater is ", first)
elif second > first :
    print("Greater is ", second)
else:
    print("Both are equal")
```

```
Enter first number
4000
 Enter second number
700000
Greater is  700000
```

In [7]:

```
price = 50

if price > 100:
    print("price is greater than 100")
elif price == 100:
    print("price is 100")
else:
    print("price is less than 100")
```

```
price is less than 100
```

Example A school has following rules for grading system:

- a. Below 25 - F
- b. 25 to 45 - E
- c. 45 to 50 - D
- d. 50 to 60 - C
- e. 60 to 80 - B
- f. Above 80 - A
- Ask user to enter marks and print the corresponding grade.

In [14]:

```
print(" Enter marks")

marks = int(input())
if marks < 25:
    print ("F")
elif marks >= 25 and marks < 45:
    print ("E")
elif marks >= 50 and marks < 60:
    print ("D")
elif marks >= 60 and marks < 80:
    print ("B")
else:
    print("A")
```

```
 Enter marks
30
E
```

In [15]:
```
print("Enter first age")
first = int(input())
print(" Enter second age")
second = int(input())
print(" Enter third age")
third = int(input())

if first>=second and second>=third:
    print("Oldest is ", first)
elif second>=first and second>=third:
    print("Oldest is ", second)
elif third>=second and first >= third:
    print("Oldest is ", third)
else:
    print("All are equal")
```

```
Enter first age
45
 Enter second age
30
 Enter third age
24
Oldest is  45
```

```
A student will not be allowed to sit in exam if his/her attendence is less than 75%.
Take following input from user
Number of classes held
Number of classes attended.
And print
percentage of class attended
Is student is allowed to sit in exam or not.
```

In [18]:
```
print("Enter No. of class held")
class_held = int(input())
print("Enter No. of class attended")
class_attended = int(input())
atten = (class_attended/float(class_held))*100
print(" Attendence is",atten)
if atten >= 75:
    print("u are allowed to sit in exam")
else:
    print("Sorry!,Not allowed to sit in exam")
```

```
Enter No. of class held
30
Enter No. of class attended
12
 Attendence is 40.0
Sorry!,Not allowed to sit in exam
```

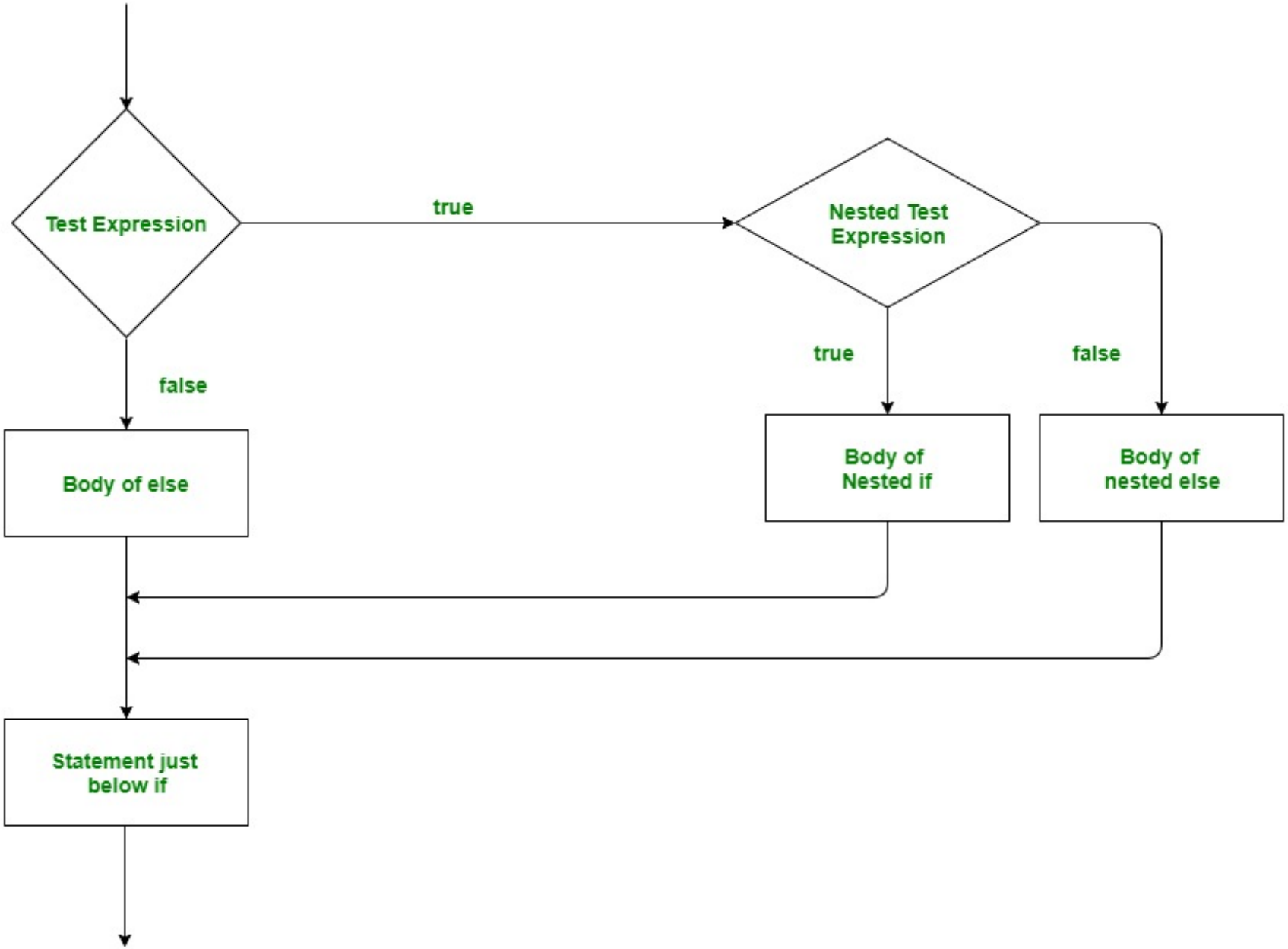
Nested if-else statement

In Python, Nested-if-else statement is an if statement inside another if-else statement. It is allowed in Python to put any number of if statements in another if statement.

Indentation is the only way to differentiate the level of nesting. The nested-if else is useful when we want to make a series of decisions.

Syntax of the nested-if-else:

```
if (condition1):
   # Executes when condition1 is true
   if (condition2):
      # Executes when condition2 is true
   # if Block is end here
# if Block is end here
```


Test Expression
true
Nested Test Expression
false
true
false
Body of else
Body of Nested if
Body of nested else
Statement just below if

In [11]:
```
# example 1

x = 41

if x > 40 :
    print("Above ten")
    if x>20:
        print("also above ten")
    else:
        print("not above 10")
```

```
Above ten
also above ten
```

In [19]:
```
i = 13

if (i == 13):
    #  First if statement
    if (i < 15):
        print ("i is smaller than 15")

    # Nested - if statement
    # Will only be executed if statement above
    # it is true
    if (i < 12):
        print ("i is smaller than 12 too")
    else:
        print ("i is greater than 12 and smaller than 15")
```

```
i is smaller than 15
i is greater than 12 and smaller than 15
```

In [ ]:
```

```