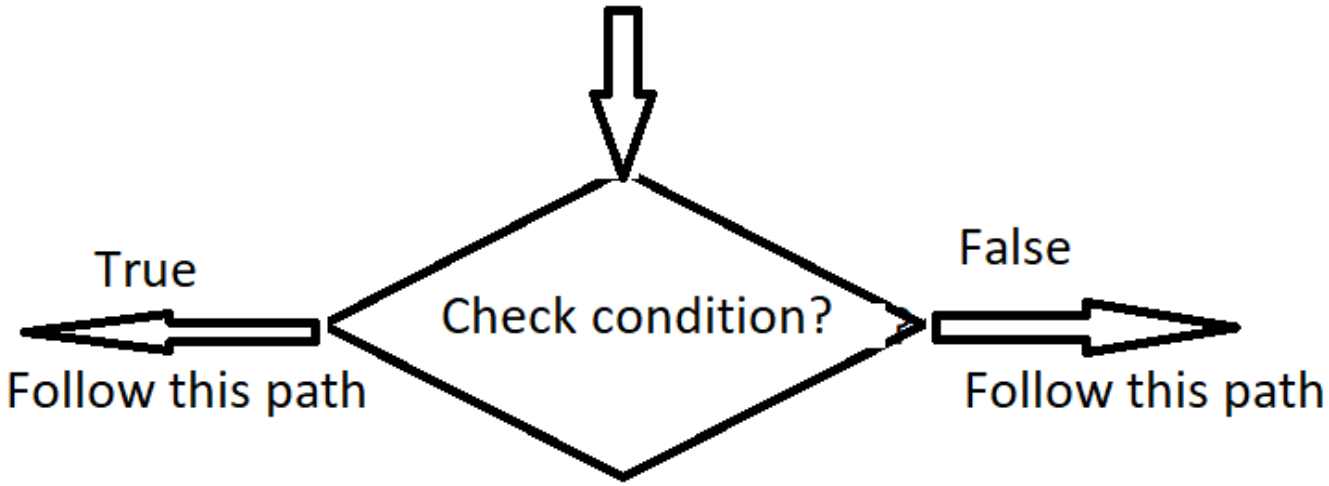
Python - Conditional Executions

Frequently the programs needs to follow a certain flow based on the fullfillment of certain condition. If something is true then follow some path otherwise follow another path.



Simple Example of guessing a number

In [ ]:

```
import random

#generate any random number between 1 to 10
number = random.randint(1,10)

guess = int(input("Enter an integer\n"))
if guess == number :
    print("Your guess is correct!")
else:
    print("Sorry, you failed to guess it!")
```

The syntax of if statement generally has the following form :

```
    if <condition> :
        statements_to_be_executed_when_condition_is_true
    else:
        statements_to_be_executed_when_condition_is_false
```

Carefully look at the colons (:) placed after if and else statements. They are compulsary. Also look at the indentation applied at the if block statements and else block statements. The statements having same indendation will be considered as the part of same block. No curly braces or other symbols are used for the determination of the if and else blocks.

Boolean Expression

The expression that return true or false values as outcome are boolean expressions. == or != or other logical operators can be used withing boolean expression to frame the desired condition.

In [ ]:

```
12 == 12  # returns true
```

In [ ]:

```
12 == 13 # returns false
```

Always keep in mind the difference between = and == operator. = is assignment whereas == is conditional.

In [ ]:

```
x = 5
y = 6

x = y  # this is assignment operator not conditional operator
```

Comparison Operators

In [ ]:

```
x = 5
y = 6
```

In [ ]:

```
x == y   # x is equal to y
```

In [ ]:

```
x != y  # x is not equal to y
```

In [ ]:

```
x > y  # x is greater than y
```

In [ ]:

```
x < y # x is less than y
```

In [ ]:

```
x >= y # x is greater than or equal to y
```

In [ ]:

```
x <= y # x is less than or equal to y
```

In [ ]:

```
x is y # x is same as y
```

In [ ]:

```
x is not y # x is not same as y
```

Logical Operators

'and', 'or' and 'not' are logical operators. They can be used while framing the complex conditions.

In [ ]:

```
x = 5
y = 6
```

In [ ]:

```
x > 4 and y > 5  # two conditions concatenanted , if both are true then only true will be r
```

In [ ]:

```
x > 4 or y < 3 # if any one of the condition is true, then will be returned otherwise false
```

In [ ]:

```
x > 14 or y > 3 # if any one of the condition is true, then will be returned otherwise fals
```

In [ ]:

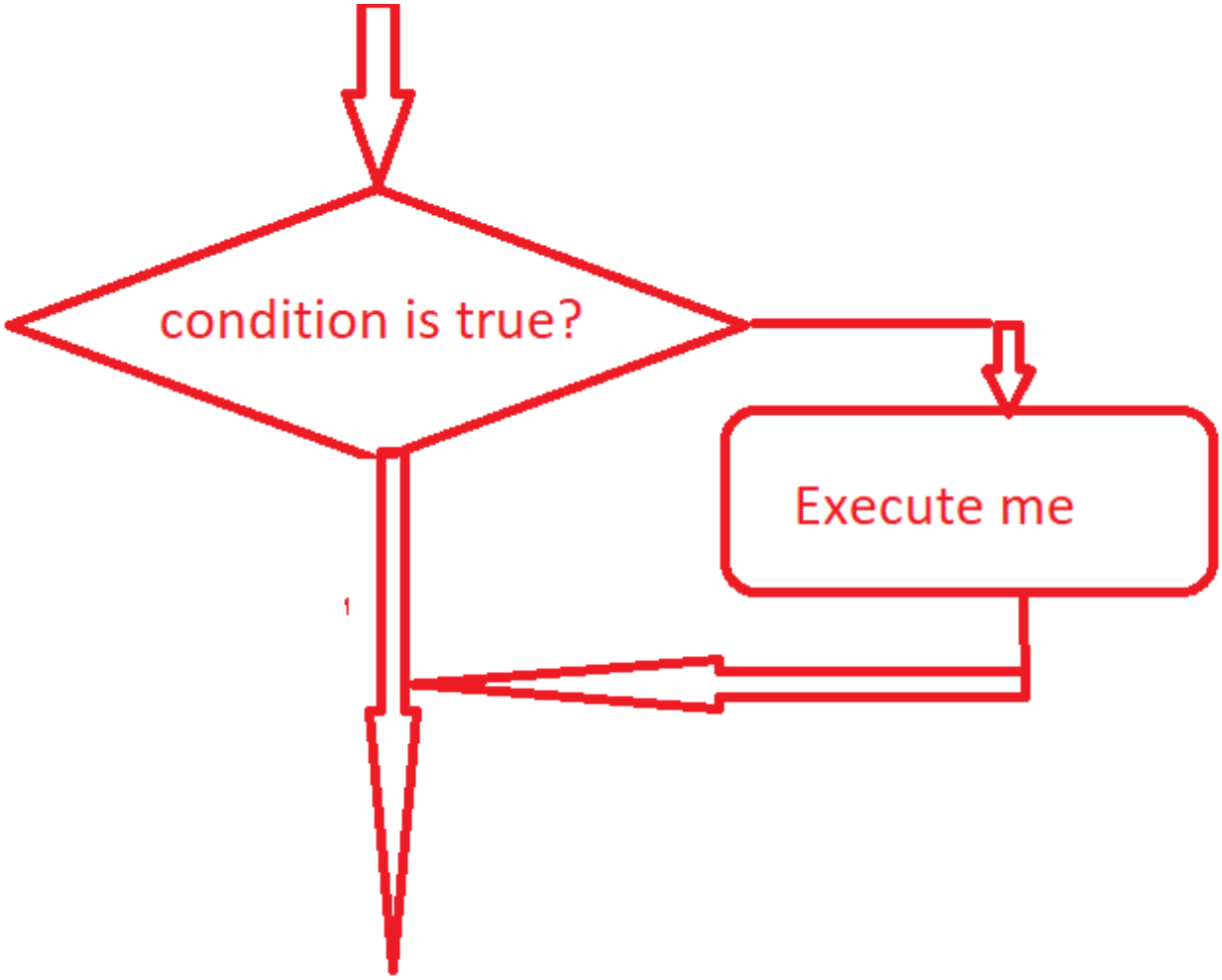
```
x < 4 or y < 3 # if any one of the condition is true, then will be returned otherwise false
```

In [ ]:

```
not(x < y ) # used to negate the value returned by conditional expression
```

Conditional Execution

Conditional statements gives ability to change the flow of the program based on the conidtions specified in the statements.



The simple if statement can have only if block in it. That means , only if the condition is satisfied , then execute the statements in the if block otherwise ignore those statements.

In [ ]:

```
number = int(input("Enter the number\n"))

if number % 2 == 0 :   # if entered number is even then only message is shown to the user
    print(number ," is even number")
```

If block can consists of more than one statemenets. If the condition is satisfied all the statements in that block are executed. The statements having same indentaion constitues the block.

In [ ]:

```
number = int(input("Enter the number\n"))

if number % 2 == 0 :   # if block can consists of more than one statements, here two print
    print("You have entered ", number)
    print(number ," is even number")
```

One more exmaple

In [ ]:

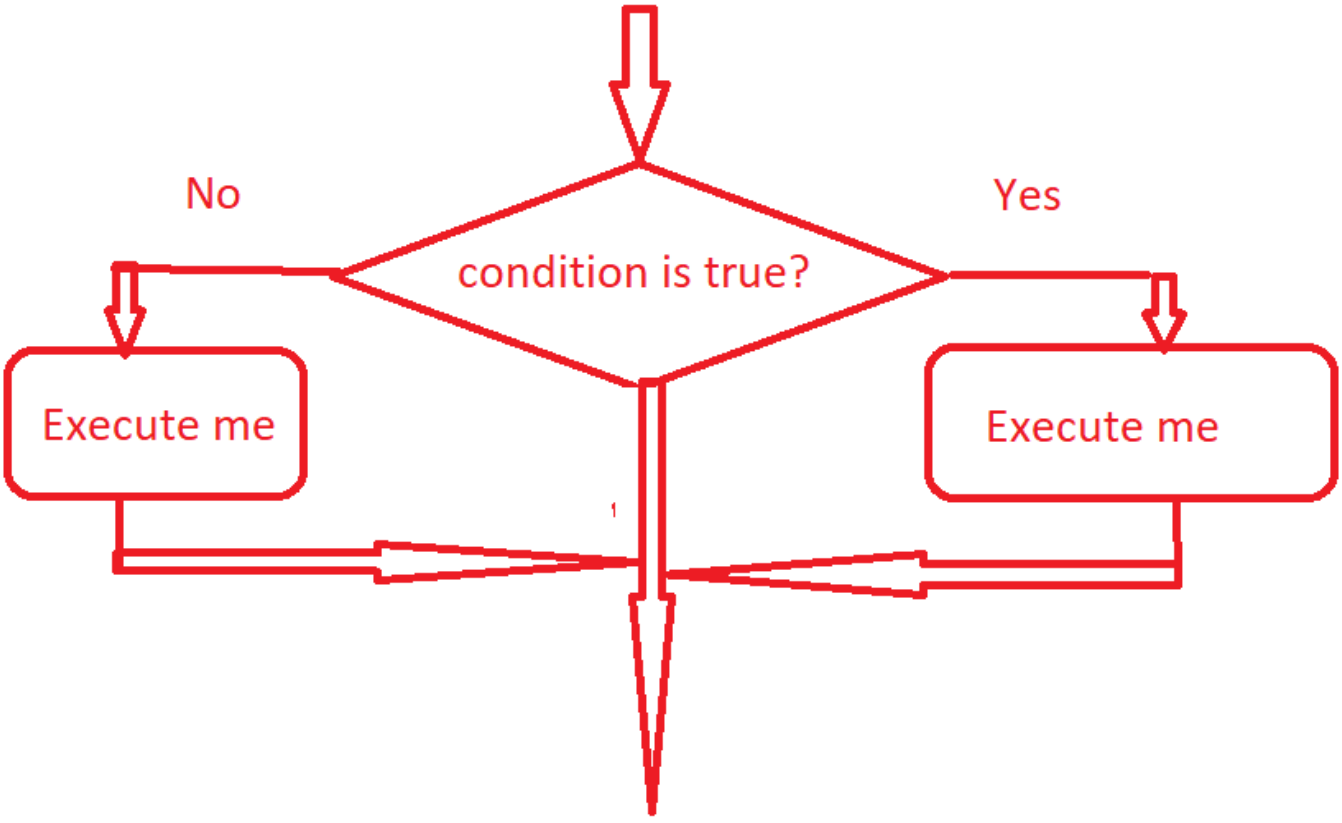
```
number = int(input("Enter the number\n"))

if number % 2 == 0 :   # if block can consists of more than one statements, here two print
    print("You have entered ", number)
    print(number ," is even number")

print("I am always printed!")    # this statement is always printed as its not part of if b
```

Alternative Execution

But there will be situations where we have more than one branches of code that needs to be taken up based on the evaluation of the conditon. In that case, else statement and corresponding else block statements can be added.

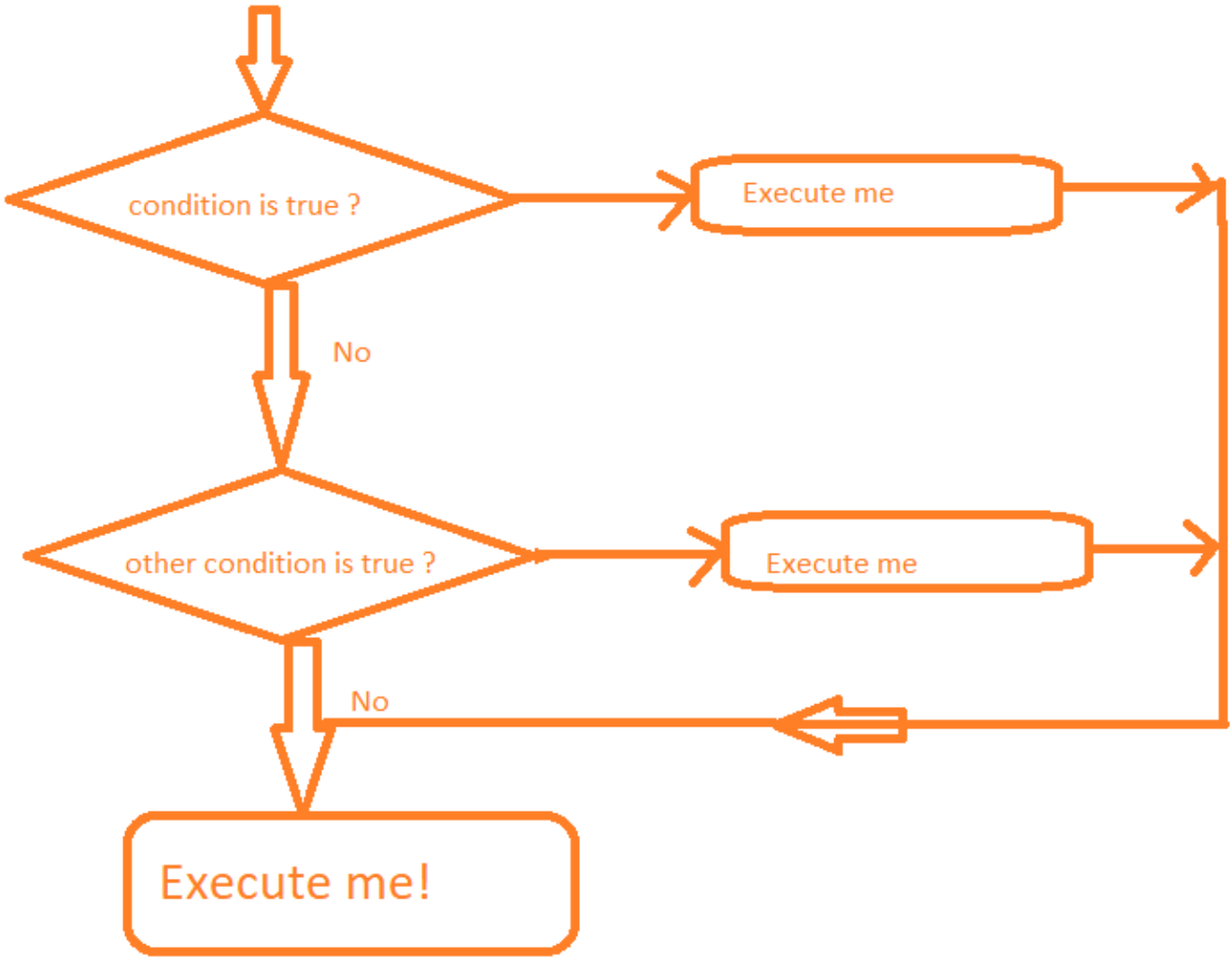


In [ ]:

```
number = int(input("Enter the number\n"))

if number % 2 == 0 :   # if entered number is even then only message is shown to the user
    print(number ," is even number")
else :
    print(number ," is odd number")
```

Chained Conditionals



Another exmaple of multiple branches :

In [ ]:

```
mark = int(input("Enter the mark\n"))

if mark < 0 or mark > 100:
    print("Invalid marks are entered!")
elif mark >= 0 and mark < 35 :
    print("Sorry you failed!")
elif mark >= 35 and mark < 80 :
    print("You passed in first class")
else :
    print("You got distinction!")
```

Nested Conditionals

One conditional can be nested within another conditional. For example :

In [3]:

```
number1 = int(input("Enter first number \n"))
number2 = int(input("Enter second number \n"))

if number1 == number2:
    print("Both numbers are same")
else:
    if number1 > number2:
        print("First number is bigger than second")
    else:
        print("First number is smaller than second")
```

```
Enter first number
5
Enter second number
3
First number is bigger than second
```

Catching Exceptions with try and except

If while executing the code, error occured then execution of code is stopped and error trace is shown to the user. In order to handle that situation gracefully, the code that is expected to raise exception can be put inside a try block and the error can be handled without halting the further execution of the program.

For example, you are asking user to enter a number and he has entered the string. When accession this string value , error will be raised.

In [8]:

```
number = int(input("Enter a number\n")) # if number entered in string then it will throw er

if number % 2 == 0 :
    print("Number is even")
else:
    print("Number is odd")
```

```
Enter a number
4
Number is even
```

This can be handled with try block and user can be made aware about the actual thing that has happened.

In [10]:

```
try :
    number = int(input("Enter a number\n")) # if number entered in string then it will thro

    if number % 2 == 0 :
        print("Number is even")
    else:
        print("Number is odd")
except:
    print("Seems you have entered a string value")
```

```
Enter a number
e
Seems you have entered a string value
```

Exercise

Q1. Write a program that asks user to enter a length in centimeters. If negative value is entered, the program should tell the user that the entry is invalid. Otherwise the program should convert the length to inches and print out the result. There are 2.54 centimeters in inches.

In [ ]:

```
#Try it here
```

Q2. Write a program that accepts sentence from the user as input and then print the unique words present in that sentense along with the number of times they appeared in the sentence. Don't use any of the built in fuction or other data structure.

In [ ]:

```
#Try it here
```

Q3.Write a program that accepts sentence from the user as input and then print the location of each vowel in the sentence.

In [ ]:

```
#Try it here
```

Q4. Write a program that asks user to enter a word and determines whether the word is palindrome or not. A palindrome is a word that read the same backwards as forwards.

In [ ]:

```
#Try it here
```

In [ ]:

```
```