1. Python

Python is a high-level, interpreted programming language known for its simplicity and readability. It allows developer to write clear programs for both small and large-scale projects.

* Python is a widely used general purpose, high level programming language.
* It was initially designed by Guido van Rossum in 1991 and developed by python software Foundation.
* Python is a programming language that lets you work quickly and integrate systems more efficiently.

There are two major Python versions - Python 2 and Python 3.

- On 16 October 2000, Python 2.0 was released with many new features.
- On 3rd December 2008, Python 3.0 was released with more testing & includes new features.

Why is Python ?

1. Python is Object-Oriented
2. Indentation
3. It's free (Open source)
4. It's powerful
   - Dynamic typing
   - Built-in types & tools
   - library utilities

5. It's portable / platform
6. It's easy to use and learn
7. Interpreted language
8. Interactive programming language
9. Straight forward syntax / simple syntax

Application of Python :-

1. Web development (e.g., Django, Flask)
2. Data Science and Machine learning (e.g., Pandas, Tensorfl
3. Automation (e.g., scripting tasks)
4. Software development and more.
5. Desktop development
6. Gaming App development
7. Mobile App development
8. Graphical user interface
9. IOT
10. AI

Python Benifits

1. Easier debugging : Errors are reported line by li
2. Faster development:

COMMENTS

COMMENTS are part of code but wont execute at runtime.

There are 2 types of comments

1. Single line comment :-
   '#' single line comment represent by hash

2. multi line comment :-

"'' multi line comment "'' or """ multi line comment """

* Comments are ignored by the python interpreter and used to explain the code or leave notes for yourself or others. They do not affect the execution of the program.

KEYWORDS:-

Python keywords are reserved words that have specific meanings and purposes in the language. They define the syntax and structure of python programs, and you cannot use them as variable names, function names, or any other identifiers.

Here's a list of commonly used python keywords (as of python 3.10+):

* Control flow: if, else, elif, for, while, break, continue, pass
* Logical Operators: and, or, not
* Data types: True, False, None
* Functions and classes: def, return, lambda, class

* Exception Handling:
  try, except, finally, raise, assert

* Importing Modules:
  import, from, as

* Variable scope:
  global, nonlocal

* Other keywords:
  with, yield, del, is, in

Python also introduced soft keywords like match and case in Python 3.10 for pattern matching.
To see all keywords in your python version,

Input/Output Function:

Input Function

In Python, we use the input() function to take input from the user. The data entered by the user is always received as a string, so if you want to use it as a different data type (e.g., integer or float), you'll need to convert it using type conversion functions like int() or float().

```
name = input("Enter your name: ")
age = int(input("Enter your age: ")) # Convert input to integer.
```

Output to the Console:

The print() function is used to display output to the console. You can use it to display text, variables, or results of expression.

* print ("Hello," + name + "! You are" + str(age) + "years old

* You can also use f-strings (formatted string literals) for more readable code.

* Print (f"Hello, {name}! You are {age} years old.")

```
# my name is Niveditha, i am from karnataka
n= input ("Enter your name")
p= input(" Enter your place")
# general printing stament
Print ('my name is', n, " ,i am from", p)
```

```
enter your name niveditha
enter your place karnataka.
my name is niveditha, i am from karnataka.
```

Methods of string formatting:

• Format method (Dot Format method)

The .format() method in Python is a powerful and flexible way to format strings. It allows you to insert variables or values into placeholders within a string, making your output dynamic and readable.

#. format method

print("my name is {}, iam from {}".format(n,p))

O/P : my name is nivedithia, i am from Karnataka.

Advantages :-

* More readable and versatile than the older formatting.
* Supports advanced formatting like alignment, padding, and number formatting.

For more complex formatting needs, consider using f-strings (introduced in Python 3.6) as a modern alternative.

④ f Strings

Python f-strings (formatted string literals) allow you to embed variables and expressions directly into strings using curly braces {}. Introduced in python 3.6, they are concise, readable, and faster than older formatting methods.

Basic Usage:

```
name = "Alice"
age = 30
print(f"My name is {name} and I am {age} years old.")
```

Output:

My name is Alice and I am 30 years old.

Embedding Expressions:

```
x = 5
y = 10
print (f"The sum of {x} and {y} is {x+y}.")
```

output:-

The sum of 5 and 10 is 15

Formatting numbers:-

```
Pi = 3.14159
print (f"Pi rounded to two decimal place: {Pi:.2f}")
```

output:-

Pi rounded to two decimal place: 3.14

Multiline f-strings

note: For long text, you can use triple quotes:

```
name: "Bob"
role: "Developer"
print (f"""Name: {name}
        Role: {Role}
        Welcome aboard!""")
```

output:-

Name: Bob
Role: Developer
Welcome aboard!

note:- Statements only should be inside the double or single quote, remaining should be in outside quote.

Variables :-

* Variable is a place to store the values with specific variable name.
* Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
* Variables are created when you assign a value to them, and you don't need to declare their type (Python is dynamically typed).
* Variable are should not be present in the quote.

Rules :-

Valid variable declaration :

1. Uppercase :→ NUM
2. lowercase → num
3. num
4. ~~stu-name~~ Underscore → stu_name
5. Characters + number → num1, num2

Invalid variable declaration:

1. Should not start with numbers = 1num
2. Should not use space = stu id
3. not use special characters = @ # * ! etc--,

Rules

→ A variable name must start with a letter or the underscore character.

→ A variable name cannot start with a number

→ A variable name can only contain alpha-numeric characters & underscores (A-Z, 0-9, and _)

→ Variable names are case-sensitive (age, Age and AGE are three different variables).

Assigning values to variables:

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

Ex:-

```
a = 100      ——→ # An integer assignment
b = 1000.    ——→ # A floating point
c = "John"   ——→ # A string

Print (a)
print (b)
print (c)
```

Data types :-

* Data types in Python are a way to classify data items.
* They represent the kind of value, which determines what operations can be performed on that data.
* Python has various built-in data types.
* To find the data type of variable
  => Use `type()` function.

a. Integer (Int)

Int, or Integer, is a whole number, positive or negative, without decimals of unlimited length.

```
Print(12345678910)
12345678910
```

b. Float

Float or "Floating point number" is a number, positive or negative, containing one or more decimals.

```
Y = 2.8
2.8
```

c. Boolean:-

Objects of boolean type may have one of two values, True or False:

Type (True)

Type (False)

d. String:

* String in python are identified as a contiguous set of characters represented in the quotation marks.
* Python allows for either pairs of single or double quotes.
* 'hello' is the same as: "hello"
* Strings can be output to screen using the print function.
  Ex:- print("hello").

Single value datatypes:-

*Datatype is classification that tells a computer what kind of value a variable holds and what operations can be performed on what data.

→ Int,
float,
complex
boolean

Multi valued datatypes:-

str
list
tuple
set
dictionary
data structure.

Swap numbers

- Swapping two variables in python is simple and can be done in multiple ways.

methods

1. Using Tuple Unpacking (Most Pythonic way)

```
x = 10
y = 50
x, y = y, x
print("x =", x)    # output: x = 50
print("y =", y)    # output: y = 10
```

2. Using a temporary variable

```
x = 10
y = 50

temp = x
x = y
y = temp
print("x =", x)    # output: x = 50
print("y =", y)    # output: y = 10
```

Rules and formulas for triangles, circles, and rectangles

1. Triangle :-

Area = 1/2 x base x height

2. Circle :-

Area = $\pi \times r^2$

3. Rectangle:-

Area = length x width

length = float(input("enter length of rectangle"))

breadth = float(input("enter breadth of rectangle"))

area_rectangle = length x breadth

print("area of rectangle =", area_rectangle)

Operators:-

* It is a special symbol used for particular operation between two operands.

a + b = c

+, = → Operators

a, b, c → Operands

Types of operators

1. Arithmatic Operators:-

Python supports basic arithmatic operations like addition, subtraction, multiplication, division, and more.

Operators:-

* + (Addition)
* - (subtraction)
* * (multiplication)
* / (Division)
* // (floor division)
* % (modular)
* ** (exponentiation)

a = int(input("enter a number: "))

b = int(input("enter a number: "))

print(f"the sum of {a} & {b} is {a+b}\n the sub of {a} & {b} is {a-b}\n the sub of {a} & {b} is {a*b}\n the sub of {a} & {b} is {a/b}\n the sub of {a} & {b} is {a**b}")

note:- print statement should be in one line and in Jupyter notebook can we write n number things in one line.

2. Assignment operators:-

Assignment operators are used to assign value to variables. The simplest one is = which assigns the value on the right to the variable on the left. There are also compound assignment operators that combine arithmetic operations with assignment.

common assignment operators:

* = :→ Assigns value on the right to the variable on the left.
* += :→ Adds right operand to the left operand and assigns the result to the left operand.
* -= :→ Subtracts the right operand from the left operand and assigns the result to the left operand.
* *= :→ Multiplies the left operand by the right operand and assigns the result to the left operand.
* /= :→ Divides the left operand by the right operand and assigns the result to the left operand.
* %= :→ Takes modulus of left operand by right operand and assigns the result to the

Example:-

```
x = 5  # Assigns 5 to x
x += 3 # Equivalent to x = x + 3, now x is 8
x -= 2 # Equivalent to x = x - 2, now x is 6
x *= 4 # Equivalent to x = x * 4, now x is 24
x /= 6 # Equivalent to x = x / 6, now x is 4.0
x //= #
x %= #
x **= #
```

3. Relational Operators / Comparison Operators :-

In python, comparison operators (also called relational operators) are used to compare two values. They return a Boolean value: True or False.

These operators are commonly used in conditional statements and loops.

List of relational / comparison operators:

Operator	Description	Example	Result
==	Equal to	5 == 5	True
!=	Not equal to	5 != 3	True
>	Greater than	4 > 3	True
<	Less than	3 < 7	True
>=	Greater than or equal to	5 >= 5	True
<=	Less than or equal to	4 <= 5	True

4. Logical Operators:

Logical operators are used to combine conditional statements. They evaluate expressions and return either True or False.

Common Logical operators:-

* and : Returns True if both conditions are true.
* or : Returns True if at least one condition is true.
* not : Reverses the logical state of its operand (True becomes False, and vice versa).

Ex:-

and

age = 24

Country = "india"

age >= 18 and country == "india"

O/P = True

Or

age = 24

country = "india"

age >= 18 or country == "india"

O/P = True

not

a = False

not(a)

O/P => True

5. Membership Operators

Membership operators test for membership within a sequence, such as a list, string, or tuple. They return True or False based on whether the value is found in the sequence.

membership operators:

* in : Returns True if the specified value is found in the sequence.
* not in : Returns True if the specified value is not found in the sequence.

Examples

in

grp = [1, 2, 3, 5, 8, 7, 3]

2 in grp

o/p → True

grp = [1, 2, 5, 8, 7, 3]

9 in grp

o/p → False

not in

grp = [1, 2, 5, 8, 7, 3]

2 not in grp

o/p → False

grp = [1, 2, 5, 8, 7, 3]

9 not in grp

o/p → True

6) Identity Operator:

Identity Operators are used to compare the memory locations of two objects. They help determine whether two variables point to the same object in memory, rather than just having the same value.

Operator:-

* IS operator:-

This is operator returns True if both variables point to the same object in memory. It is used to check the identity of objects.

ex

a = 5

b = 6

a is b

o/p → False.

a = 5

b = 5

a is b

o/p → True

* Is NOT Operator:-

* This is not operator returns True if both variables do not point to the same project in memory.

* It is used to check if two objects are not identical

* Eg

```
# Define two lists with the same content
a = [1, 2, 3]
b = [1, 2, 3]
c = a

# Compare using 'is not' operator
print(a is not c)
```

o/p False,

```
a = 5
b = 6
a is not b
```

True

Bitwise Operators:-

Bitwise operators perform operations on binary representation of integers. These operations are useful for low-level programming tasks like working with bits & bytes.

Binary

2^n

$2^2 = 4 \rightarrow 0, 1, 2, 3$

$2^3 = 8 \rightarrow 0, 1, 2, 3, 4, 5, 6, 7$

$2^4 = 16 \rightarrow 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$

$\frac{8}{2^3}$	$\frac{4}{2^2}$	$\frac{2}{2^1}$	$\frac{1}{2^0}$	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

$12 - 9 = 3$

$12 \mid 9 = 13$

Right shift

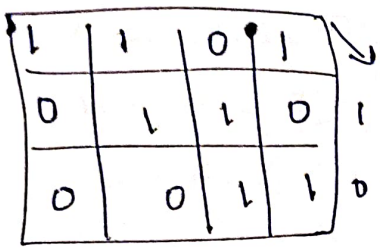
1 2 9 6 → 1 2 9 6

Unit
Ten
Hundred
Thousand

1 2 9 % = 129

1 2 % = 12

13 >> 2



1 = 5

1 = 3

Left shift

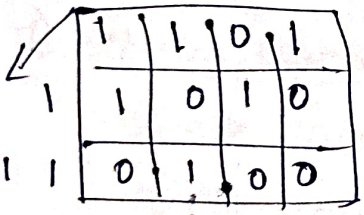
1296 →

1 2 9 6

1 2 9 6 0 = 12960

1 2 9 6 0 0 = 129600

13 << 2



= 26

Common Bitwise Operators

- & = Bitwise AND (sets each bit to 1 if both bits are 1)
- | = Bitwise OR (sets each bit to 1 if one of the bits is 1)
- ^ = Bitwise XOR (sets each bit to 1 if only one of the bits is 1).
- ~ = Bitwise NOT (inverts all the bits).

* << = left shift (shifts bits to the left by a specified number of positions).
* >> = Right shift (shifts bits to the right by a specified number of positions).

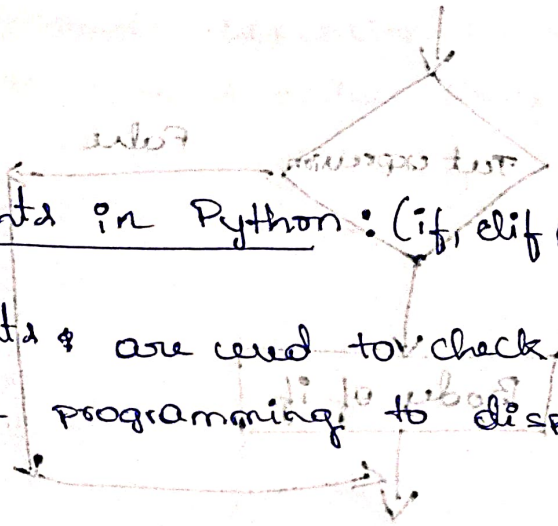
Ex:-

12 & 9 — 13

13 << 1 — 26

13 << 2 — 52

Conditional Statements in Python: (if, elif and else)

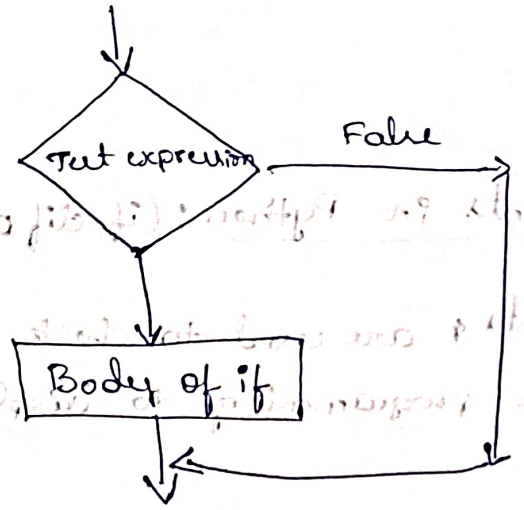


* Conditional statements are used to check the conditions in the programming to display the result.
* In programming, conditional statements are used to perform different actions based on different conditions.
* Python uses if, elif, and else statements to allow your program to make decisions.
* if
  if else
  if else if ladder / continuous if else
  nested if else
* One tab space is used to create a relationship, if used ':' automatically created tab space.

1. IF (if) statement: -

The if statement is used to test a condition. If the condition is True, the block of code under the if statement is executed.

if statement flowchart



Operation of if statement

Syntax of if: -

```
if (condition):
    statement
```

Example: -

Eligibility criteria to vote in India

```
age = int(input())
country = input()

if ( age >= 18 and country == "india"):
        print ("you are eligible for voting.")
```

note: - 1. Python is case sensitive language

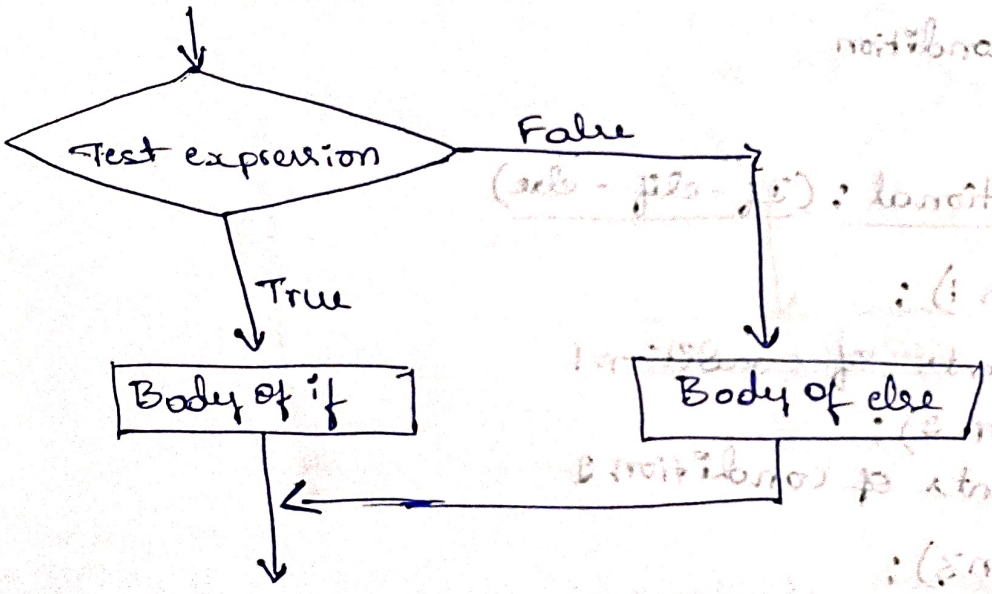
2. In python string is the default data type, no need to mention the str.

o/p= 23
india
you are eligible for voting.

2. if - else statement

An else statement can be combined with an if statement contains the block of code (false block) that executes if the conditional expression in the if statement resolves to 0 or a False value.

If else flowchart



Operation of if - else statement

Note:- In else we can not write conditions.

Ex:- age = int(input())
Country = input()
PC = country.lower()
if (age >= 18 and country == 'india'):
    Print ("you are eligible for voting")
else:
    print ("you are not eligible for voting")

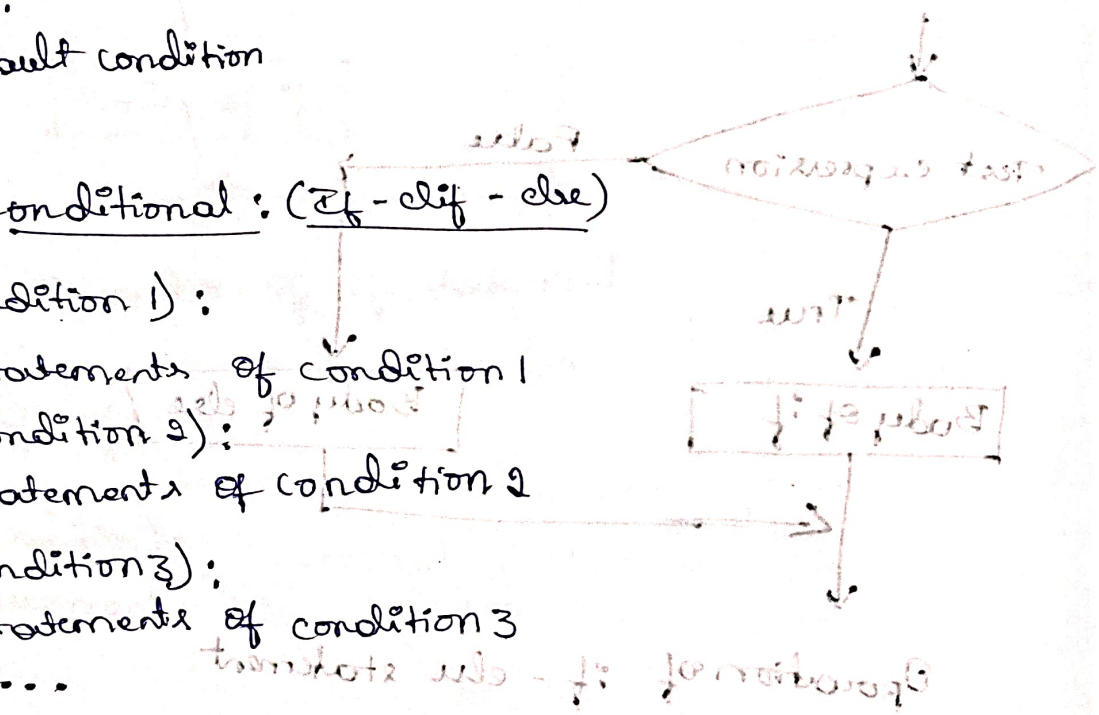
o/p= 17
india
You are not eligible for voting.

3. if else if ladder

Syntax:-

```
if (condition1):
    Statement of condition1
else:
    if (condition2):
        Statements of condition2
    else:
        if(condition3):
            statements of condition3
else:
Default condition
```

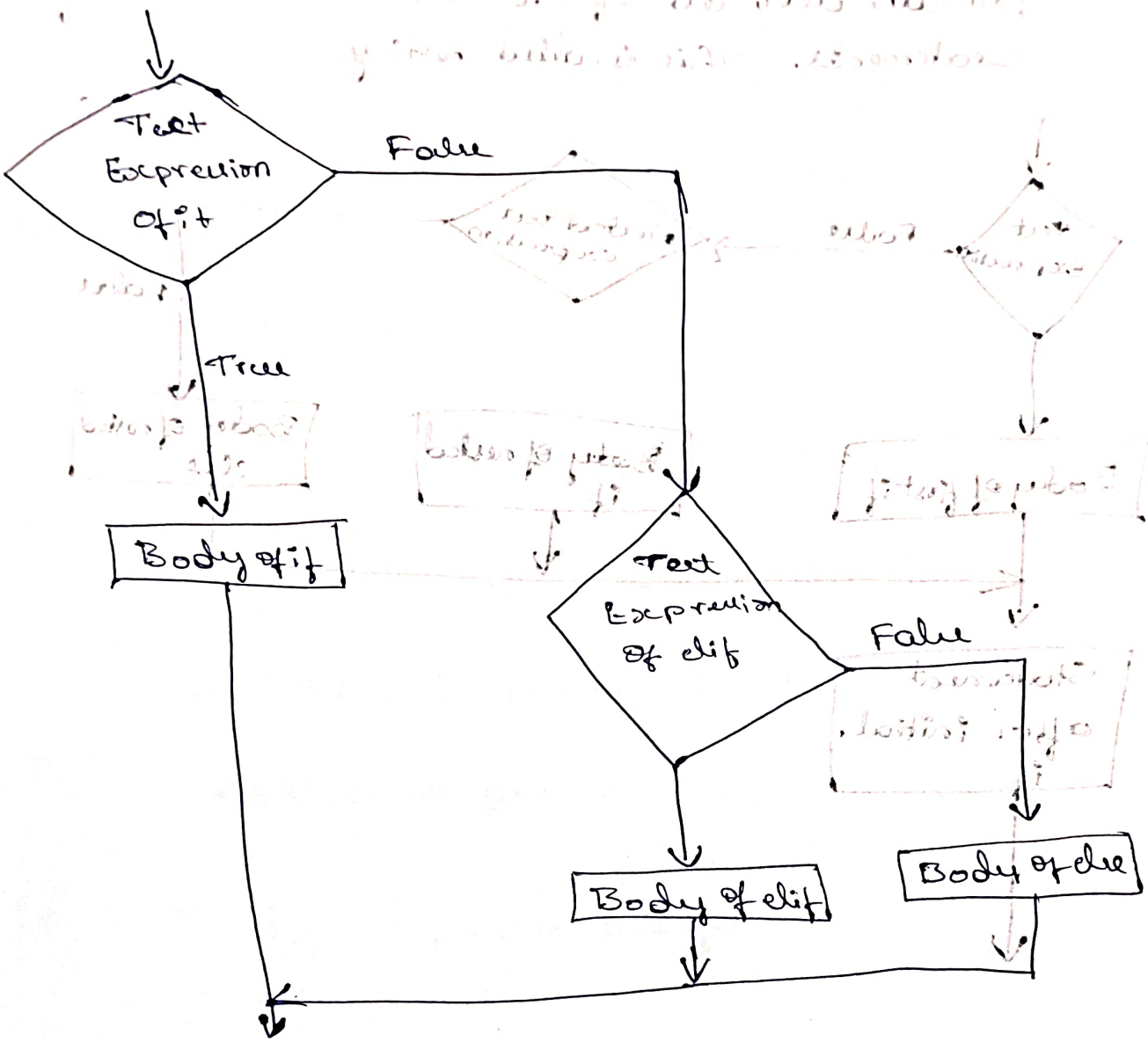
4. Chained Conditional : (If - elif - else)



```
if (condition 1):
    Statements of condition 1
elif (condition 2):
    Statements of condition 2
elif (condition 3):
    Statements of condition 3
    ....
else:
default condition
```

The elif statement allows us to check multiple expression for True & execute a block of code as soon as one of the conditions evaluate to true. Similar to the else, the elif statement is optional.

Flowchart of if - elif - else



Ex:- write a program to check the given number is positive, negative or zero

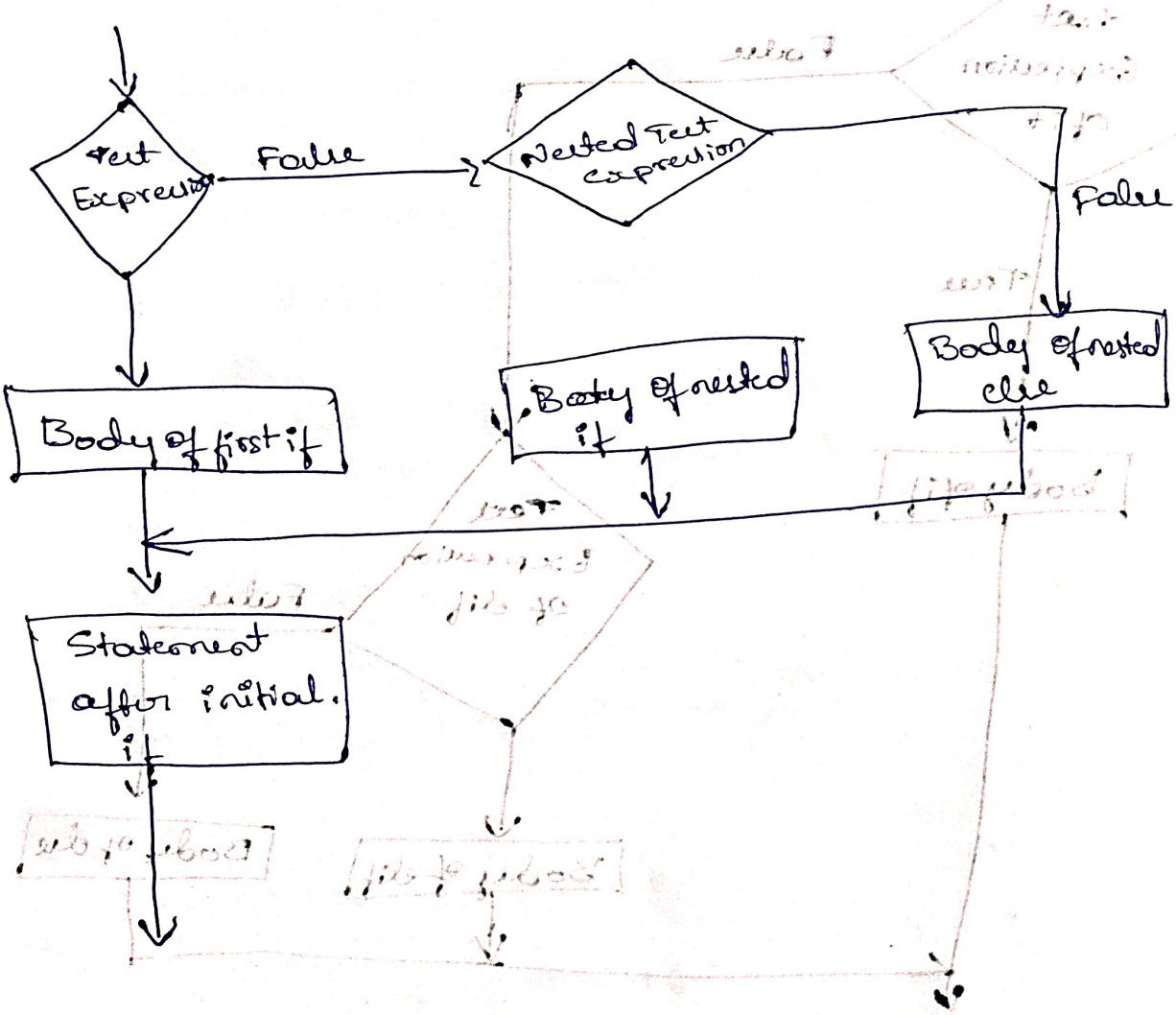
```
num = int(input())

if (num > 0):
    print ("positive number")
elif (num < 0):
    print ("nagetive number")
else:
    print ("zero")
```

O/P -24

nagetive number

5. Nested if statements:

You can also use if statements inside other if statements. This is called nesting.



Syntax:-

```
if(Condition1): # outer if
    if(Condition2): # inner if
        Statement of inner if
    else: # inner else
        Statements of inner else
else:
    statement of outer else
```

Ex:- Triangle type checker take three sides of a triangle as input. And check it is equilateral, issocelus, scalene or not a triangle.

```
a = 11
b = 1
c = 30

if (a+b>c) and (a+c>b) and (b+c>a)):
    if(a==b or b==c or a==c):
        if(a==b==c):
            print("It is an equilateral triangle.")
        else:
            print("It is an Isosceles triangle.")
    else:
        print("It is a scalene triangle.")
else:
    print("This do no form a triangle.")
```

O/P => This do not form a triangle.