PYTHON INTRODUCTION

VARIABLES

Variable is a name which is used to refer memory location. Variable also known as identifier and used to hold value.

In Python, we don't need to specify the type of variable because Python is a type infer language and smart enough to get variable type.

Variable names can be a group of both letters and digits, but they have to begin with a letter or an underscore.

It is recomended to use lowercase letters for variable name. Rahul and rahul both are two different variables.

IDENTIFIER NAMING

Variables are the example of identifiers. An Identifier is used to identify the literals used in the program. The rules to name an identifier are given below.

- The first character of the variable must be an alphabet or underscore (_).
- All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore or digit (0-9).
- Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).
- Identifier name must not be similar to any keyword defined in the language.

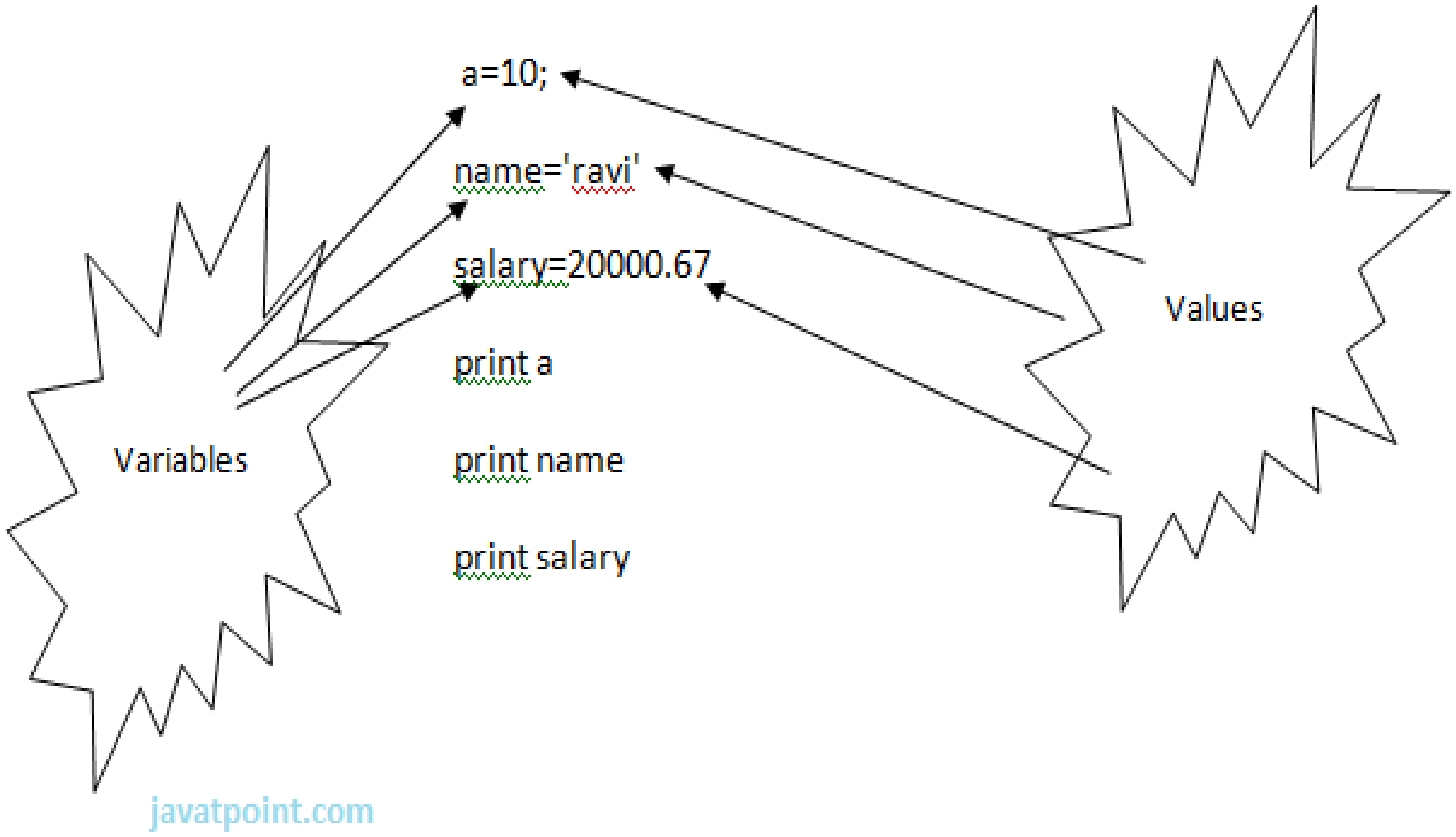
- Identifier names are case sensitive for example myname, and MyName is not the same.
- Examples of valid identifiers : a123, _n, n_9, etc.
- Examples of invalid identifiers: 1a, n%4, n 9, etc.

DECLARING VARIABLE AND ASSIGNING VALUES

Python does not bound us to declare variable before using in the application. It allows us to create variable at required time.

We don't need to declare explicitly variable in Python. When we assign any value to the variable that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.



```
a=10;
name='ravi'
salary=20000.67
print a
print name
print salary
```

MULTIPLE ASSIGNMENT

Python allows us to assign a value to multiple variables in a single statement which is also known as multiple assignment.

We can apply multiple assignments in two ways either by assigning a single value to multiple variables or assigning multiple values to multiple variables. Lets see given examples.

x=y=z=50

print iple

print y

print z

ASSIGNING MULTIPLE VALUES TO MULTIPLE VARIABLES:

```
a,b,c=5,10,15
print a
print b
print c
```

PYTHON DATA TYPES

Variables can hold values of different data types. Python is a dynamically typed language hence we need not define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

Python enables us to check the type of the variable used in the program. Python provides us the **type()** function which returns the type of the variable passed.

Consider the following example to define the values of different data types and checking its type.

```
A=10
b="Hi Python"
c = 10.5
print(type(a));
print(type(b));
print(type(c));
```

STANDARD DATA TYPES

A variable can hold different types of values. For example, a person's name must be stored as a string whereas its id must be stored as an integer.

Python provides various standard data types that define the storage method on each of them. The data types defined in Python are given below.

Numbers

String

List

Tuple

Dictionary

NUMBERS

Number stores numeric values. Python creates Number objects when a number is assigned to a variable. For example;

```
a = 3 , b = 5  #a and b are number objects
```

Python supports 4 types of numeric data.

- int (signed integers like 10, 2, 29, etc.)
- long (long integers used for a higher range of values like 908090800L, -0x1929292L, etc.)
- float (float is used to store floating point numbers like 1.9, 9.902, 15.2, etc.)
- complex (complex numbers like 2.14j, 2.0 + 2.3j, etc.)

Python allows us to use a lower-case L to be used with long integers. However, we must always use an upper-case L to avoid confusion.

A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts respectively).

Python 3 removes this concept. In new version integer automatically converted to long so need to specify suffix L.

STRING

The string can be defined as the sequence of characters represented in the quotation marks. In python, we can use single, double, or triple quotes to define a string.

String handling in python is a straightforward task since there are various inbuilt functions and operators provided.

In the case of string handling, the operator + is used to concatenate two strings as the operation *"hello"+" python"* returns *"hello python"*.

The operator * is known as repetition operator as the operation "Python " *2 returns "Python Python ".

The following example illustrates the string handling in python.

```
str1 = 'hello javatpoint' #string str1
str2 = ' how are you' #string str2
print (str1[0:2]) #printing first two character using slice operator
print (str1[4]) #printing 4th character of the string
print (str1*2) #printing the string twice
print (str1 + str2) #printing the concatenation of str1 and str2
```

LIST

Lists are similar to arrays in C. However; the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

We can use slice [:] operators to access the data of the list. The concatenation operator (+) and repetition operator (*) works with the list in the same way as they were working with the strings.

```
l  = [1, "hi", "python", 2]

print (l[3:]);

print (l[0:2]);

print (l);

print (l + l);

print (l * 3);
```

TUPLE

A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The items of the tuple are separated with a comma (,) and enclosed in parentheses ().

A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

Let's see a simple example of the tuple.

```
t  = ("hi", "python", 2)
print (t[1:]);
print (t[0:1]);
print (t);
print (t + t);
print (t * 3);
print (type(t))
t[2] = "hi";
```

DICTIONARY

Dictionary is an ordered set of a key-value pair of items. It is like an associative array or a hash table where each key stores a specific value. Key can hold any primitive data type whereas value is an arbitrary Python object.

The items in the dictionary are separated with the comma and enclosed in the curly braces {}.

Consider the following example.

```
d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'};
print("1st name is "+d[1]);
print("2nd name is "+ d[4]);
print (d);
print (d.keys());
print (d.values());
```

KEYWORDS

Python Keywords are special reserved words which convey a special meaning to the compiler/interpreter. Each keyword have a special meaning and a specific operation. These keywords can't be used as variable. Following is the List of Python Keywords.

True	False	None	and	as
asset	def	class	continue	break
else	finally	elif	del	except
global	for	if	from	import
raise	try	or	return	pass
nonlocal	in	not	is	lambda

LITERALS

Literals can be defined as a data that is given in a variable or constant.

Python support the following literals:

STRING LITERALS:

String literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes for a String.

Eg:

"Aman" , '12345'

Types of Strings:

There are two types of Strings supported in Python:

Single line String- Strings that are terminated within a single line are known as Single line Strings.

Eg:

```
>>> text1='hello'
```

Multi line String- A piece of text that is spread along multiple lines is known as Multiple line String.

There are two ways to create Multiline Strings:

1). Adding black slash at the end of each line.

Eg:

```
>>> text1='hello\
user'
>>> text1
'hellouser'
>>>
```

Using triple quotation marks:-

```
>>> str2="""welcome
to
SSSIT"""
>>> print str2
welcome
to
SSSIT
>>>
```

NUMERIC LITERALS:

Numeric Literals are immutable. Numeric literals can belong to following four different numerical types.

Int(signed integers)	Long(long integers)	float(floating point)	Complex(complex)
Numbers(can be both positive and negative) with no fractional part.eg: 100	Integers of unlimited size followed by lowercase or uppercase L eg: 87032845L	Real numbers with both integer and fractional part eg: -26.2	In the form of a+bj where a forms the real part and b forms the imaginary part of complex number. eg: 3.14j

BOOLEAN LITERALS:

A Boolean literal can have any of the two values: True or False.

SPECIAL LITERALS.

Python contains one special literal i.e., None.

None is used to specify to that field that is not created. It is also used for end of lists in Python.

Eg:

```
>>> val1=10

>>> val2=None

>>> val1
```

```
10
>>> val2
>>> print val2
None
>>>
```

LITERAL COLLECTIONS.

Collections such as tuples, lists and Dictionary are used in Python.

List:

List contain items of different data types. Lists are mutable i.e., modifiable.

The values stored in List are separated by commas(,) and enclosed within a square brackets([]). We can store different type of data in a List.

Value stored in a List can be retrieved using the slice operator([] and [:]).

The plus sign (+) is the list concatenation and asterisk(*) is the repetition operator.

```
>>> list=['aman',678,20.4,'saurav']
>>> list1=[456,'rahul']
>>> list
['aman', 678, 20.4, 'saurav']
>>> list[1:3]
[678, 20.4]
>>> list+list1
['aman', 678, 20.4, 'saurav', 456, 'rahul']
>>> list1*2
[456, 'rahul', 456, 'rahul']
>>>
```

OPERATORS

The operator can be defined as a symbol which is responsible for a particular operation between two operands. Operators are the pillars of a program on which the logic is built in a particular programming language. Python provides a variety of operators described as follows.

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

ARITHMETIC OPERATORS

Arithmetic operators are used to perform arithmetic operations between two operands. It includes +(addition), - (subtraction), *(multiplication), /(divide), %(reminder), //(floor division), and exponent (**).

Operator	Description
+ (Addition)	It is used to add two operands. For example, if a = 20, b = 10 => a+b = 30
- (Subtraction)	It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value result negative. For example, if a = 20, b = 10 => a ? b = 10
/ (divide)	It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2
*** (Multiplication)**	It is used to multiply one operand with the other. For example, if a = 20, b = 10 => a * b = 200
% (reminder)	It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0
**** (Exponent)**	It is an exponent operator represented as it calculates the first operand power to second operand.
// (Floor division)	It gives the floor value of the quotient produced by dividing the two operands.

COMPARISON OPERATOR

Comparison operators are used to comparing the value of the two operands and returns boolean true or false accordingly. The comparison operators are described in the following table.

Operator	Description
==	If the value of two operands is equal, then the condition becomes true.
!=	If the value of two operands is not equal then the condition becomes true.
<=	If the first operand is less than or equal to the second operand, then the condition becomes true.
>=	If the first operand is greater than or equal to the second operand, then the condition becomes true.
<>	If the value of two operands is not equal, then the condition becomes true.
>	If the first operand is greater than the second operand, then the condition becomes true.
<	If the first operand is less than the second operand, then the condition becomes true.

PYTHON ASSIGNMENT OPERATORS

The assignment operators are used to assign the value of the right expression to the left operand. The assignment operators are described in the following table.

Operator	Description
=	It assigns the the value of the right expression to the left operand.
+=	It increases the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30.
-=	It decreases the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10.
*=	It multiplies the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 10, b = 20 => a* = b will be equal to a = a* b and therefore, a = 200.
%=	It divides the value of the left operand by the value of the right operand and assign the reminder back to left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0.
**=	a**=b will be equal to a=a**b, for example, if a = 4, b =2, a**=b will assign 4**2 = 16 to a.
//=	A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a.

BITWISE OPERATOR

The bitwise operators perform bit by bit operation on the values of the two operands.

if a = 7;

b = 6;

then, binary (a) = 0111

binary (b) = 0011

hence, a & b = 0011

a | b = 0111

a ^ b = 0100

~ a = 1000

Operator	Description
& (binary and)	If both the bits at the same place in two operands are 1, then 1 is copied to the result. Otherwise, 0 is copied.
| (binary or)	The resulting bit will be 0 if both the bits are zero otherwise the resulting bit will be 1.
^ (binary xor)	The resulting bit will be 1 if both the bits are different otherwise the resulting bit will be 0.
~ (negation)	It calculates the negation of each bit of the operand, i.e., if the bit is 0, the resulting bit will be 1 and vice versa.
<< (left shift)	The left operand value is moved left by the number of bits present in the right operand.
>> (right shift)	The left operand is moved right by the number of bits present in the right operand.

LOGICAL OPERATORS

The logical operators are used primarily in the expression evaluation to make a decision. Python supports the following logical operators.

Operator	Description
and	If both the expression are true, then the condition will be true. If a and b are the two expressions, a → true, b → true => a and b → true.
or	If one of the expressions is true, then the condition will be true. If a and b are the two expressions, a → true, b → false => a or b → true.
not	If an expression **a** is true then not (a) will be false and vice versa.

MEMBERSHIP OPERATORS

Python membership operators are used to check the membership of value inside a data structure. If the value is present in the data structure, then the resulting value is true otherwise it returns false.

Operator	Description
in	It is evaluated to be true if the first operand is found in the second operand (list, tuple, or dictionary).
not in	It is evaluated to be true if the first operand is not found in the second operand (list, tuple, or dictionary).

IDENTITY OPERATORS

Operator	Description
is	It is evaluated to be true if the reference present at both sides point to the same object.
is not	It is evaluated to be true if the reference present at both side do not point to the same object.

COMMENTS

Comments in Python can be used to explain any program code. It can also be used to hide the code as well.

Comments are the most helpful stuff of any program. It enables us to understand the way, a program works. In python, any statement written along with # symbol is known as a comment. The interpreter does not interpret the comment.

Comment is not a part of the program, but it enhances the interactivity of the program and makes the program readable.

Python supports two types of comments:

SINGLE LINE COMMENT:

In case user wants to specify a single line comment, then comment must start with ?#?

Eg:

```
# This is single line comment.
print "Hello Python"
```

MULTI LINE COMMENT:

Multi lined comment can be given inside triple quotes.

eg:

```
""" This  Is
    Multipline comment'''
```

eg:

```
#single line comment
print "Hello Python"
"""This is
multiline comment'''
```

DECISION MAKING

PYTHON IF-ELSE STATEMENTS

Decision making is the most important aspect of almost all the programming languages. As the name implies, decision making allows us to run a particular block of code for a particular decision. Here, the decisions are made on the validity of the particular conditions. Condition checking is the backbone of decision making.

In python, decision making is performed by the following statements.

Statement	Description
If Statement	The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.
If - else Statement	The if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked. If the condition provided in the if statement is false, then the else statement will be executed.
Nested if Statement	Nested if statements enable us to use if ? else statement inside an outer if statement.

INDENTATION IN PYTHON

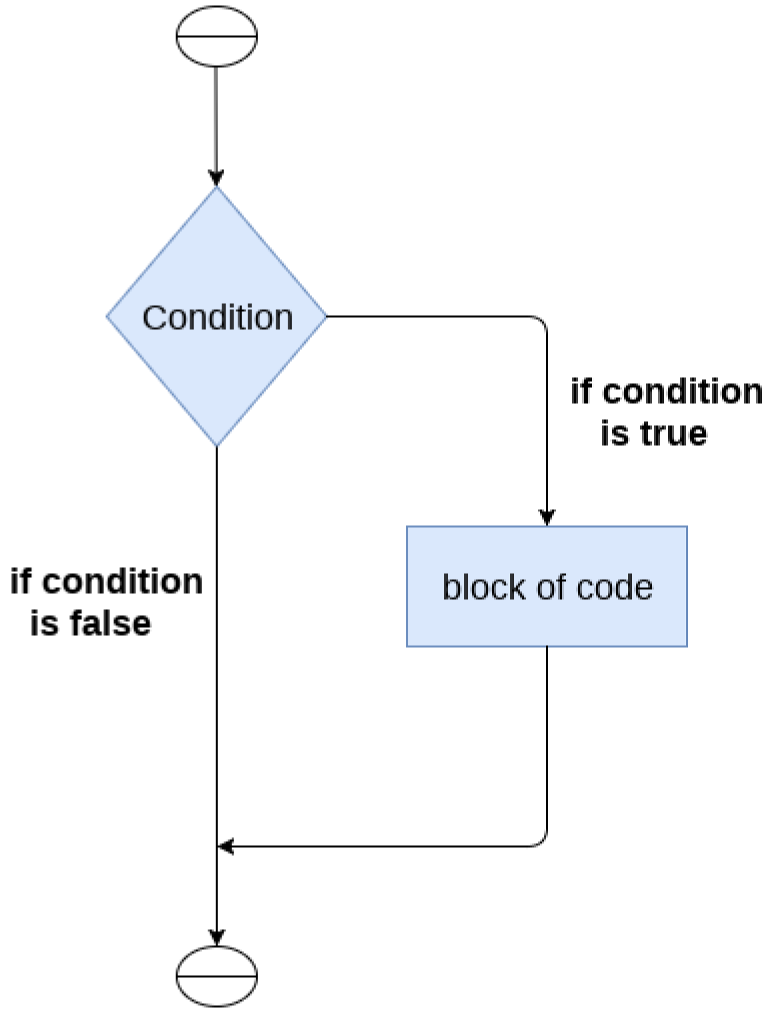
For the ease of programming and to achieve simplicity, python doesn't allow the use of parentheses for the block level code. In Python, indentation is used to declare a block. If two statements are at the same indentation level, then they are the part of the same block.

Generally, four spaces are given to indent the statements which are a typical amount of indentation in python.

Indentation is the most used part of the python language since it declares the block of code. All the statements of one block are intended at the same level indentation. We will see how the actual indentation takes place in decision making and other stuff in python.

THE IF STATEMENT

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.



The syntax of the if-statement is given below.

```
if expression:
    statement
```

```
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even")
```

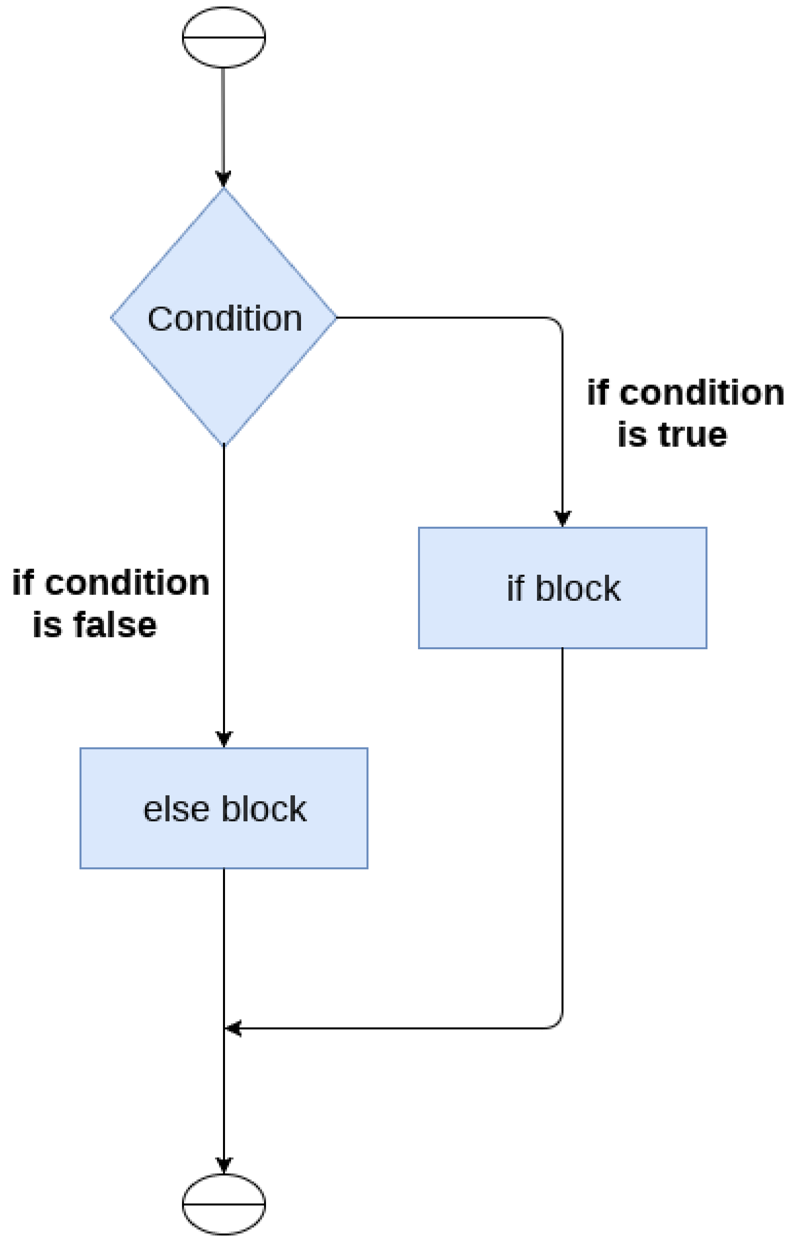
PROGRAM TO PRINT THE LARGEST OF THE THREE NUMBERS.

```
a = int(input("Enter a? "));
b = int(input("Enter b? "));
c = int(input("Enter c? "));
if a>b and a>c:
    print("a is largest");
if b>a and b>c:
    print("b is largest");
if c>a and c>b:
    print("c is largest");
```

THE IF-ELSE STATEMENT

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.

If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.



The syntax of the if-else statement is given below.

```
if condition:
    #block of statements
else:
    #another block of statements (else-block)
```

PROGRAM TO CHECK WHETHER A PERSON IS ELIGIBLE TO VOTE OR NOT.

```
age = int (input("Enter your age? "))
if age>=18:
    print("You are eligible to vote !!");
else:
    print("Sorry! you have to wait !!");
```

PROGRAM TO CHECK WHETHER A NUMBER IS EVEN OR NOT.

```
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even...")
else:
    print("Number is odd...")
```

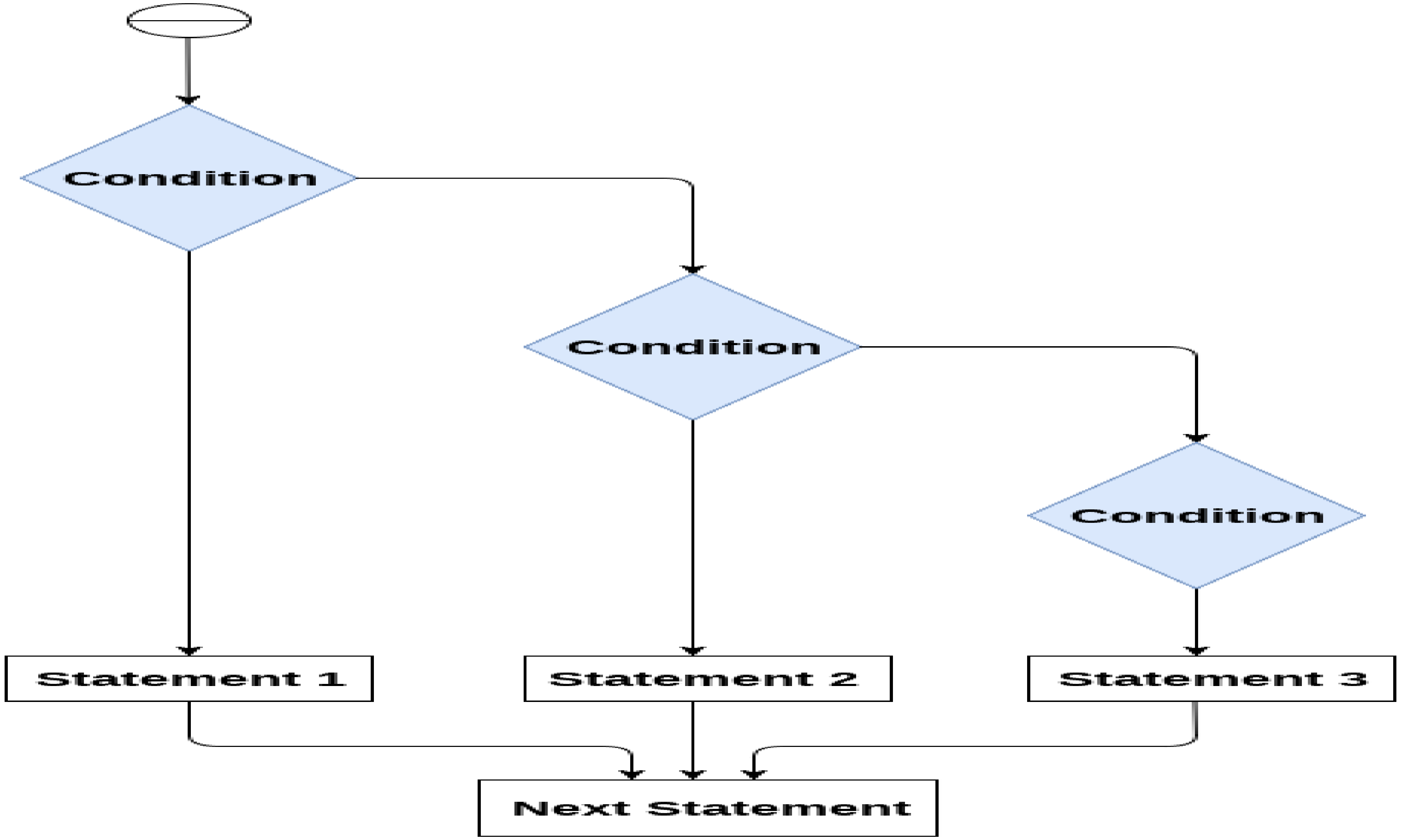
THE ELIF STATEMENT

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.

The syntax of the elif statement is given below.

```
if expression 1:
    # block of statements
elif expression 2:
    # block of statements
elif expression 3:
    # block of statements
else:
    # block of statements
```


Condition
Condition
Condition
Statement 1
Statement 2
Statement 3
Next Statement

EXAMPLE 1

```
number = int(input("Enter the number?"))
if number==10:
    print("number is equals to 10")
elif number==50:
    print("number is equal to 50");
elif number==100:
    print("number is equal to 100");
else:
    print("number is not equal to 10, 50 or 100");
```

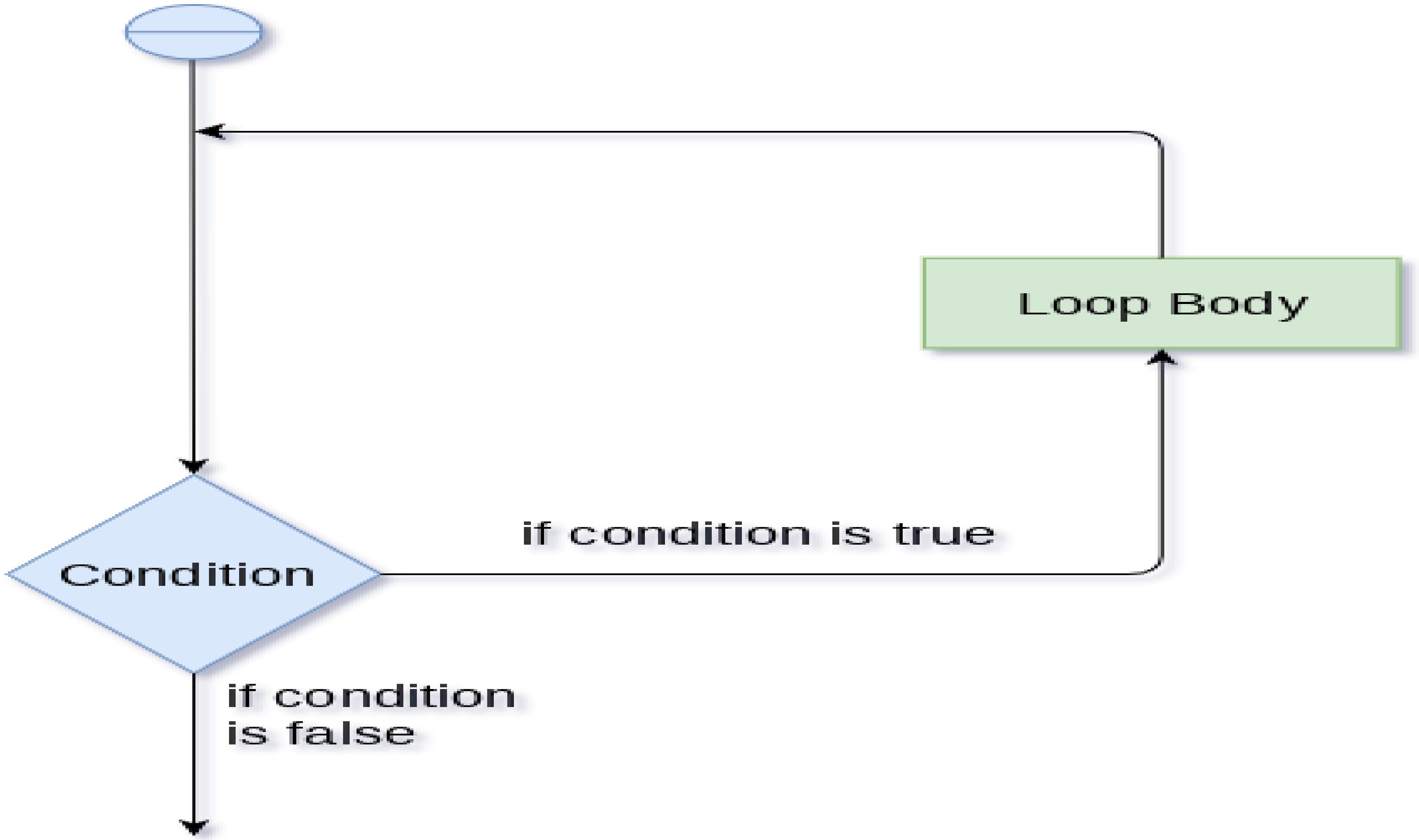
EXAMPLE 2

```
marks = int(input("Enter the marks? "))
if marks > 85 and marks <= 100:
   print("Congrats ! you scored grade A ...")
elif marks > 60 and marks <= 85:
   print("You scored grade B + ...")
elif marks > 40 and marks <= 60:
   print("You scored grade B ...")
elif (marks > 30 and marks <= 40):
   print("You scored grade C ...")
else:
   print("Sorry you are fail ?")
```

LOOPS

The flow of the programs written in any programming language is sequential by default. Sometimes we may need to alter the flow of the program. The execution of a specific code may need to be repeated several numbers of times.

For this purpose, The programming languages provide various types of loops which are capable of repeating some specific code several numbers of times. Consider the following diagram to understand the working of a loop statement.


Loop Body
if condition is true
Condition
if condition
is false

WHY WE USE LOOPS IN PYTHON?

The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the print statement 10 times, we can print inside a loop which runs up to 10 iterations.

ADVANTAGES OF LOOPS

There are the following advantages of loops in Python.

- It provides code re-usability.
- Using loops, we do not need to write the same code again and again.
- Using loops, we can traverse over the elements of data structures (array or linked lists).

THERE ARE THE FOLLOWING LOOP STATEMENTS IN PYTHON.

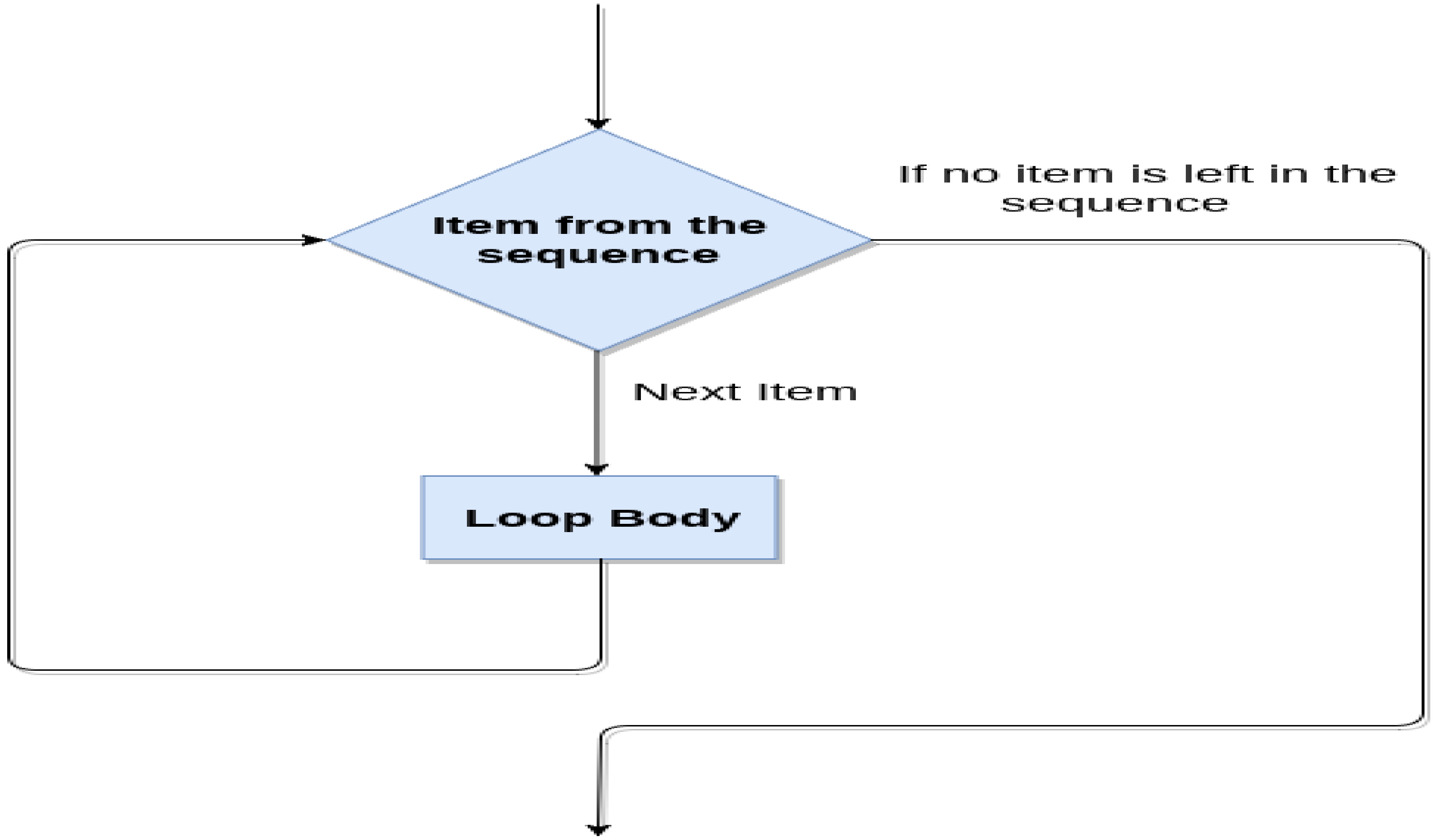
Loop Statement	Description
for loop	The for loop is used in the case where we need to execute some part of the code until the given condition is satisfied. The for loop is also called as a per-tested loop. It is better to use for loop if the number of iteration is known in advance.
while loop	The while loop is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop.
do-while loop	The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once (mostly menu driven programs).

PYTHON FOR LOOP

The for **loop in Python** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.

The syntax of for loop in python is given below.

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


Item from the sequence
If no item is left in the sequence
Next Item
Loop Body

```
i=1
end=int(input("Enter the number up to which you want to print the natural numbers?"))

for i in range(0,10):
    print(i,end = ' ')
```

PRINTING THE TABLE OF THE GIVEN NUMBER

```
i=1;
num = int(input("Enter a number:"));
for i in range(1,11):
    print("%d X %d = %d"%(num,i,num*i));
```

NESTED FOR LOOP IN PYTHON

Python allows us to nest any number of for loops inside a for loop. The inner loop is executed n number of times for every iteration of the outer loop. The syntax of the nested for loop in python is given below.

```
for iterating_var1 in sequence:
    for iterating_var2 in sequence:
        #block of statements
#Other statements
```

EXAMPLE 1

```
n = int(input("Enter the number of rows you want to print?"))
i,j=0,0
for i in range(0,n):
    print()
    for j in range(0,i+1):
        print("*",end="")
```

USING ELSE STATEMENT WITH FOR LOOP

Unlike other languages like C, C++, or Java, python allows us to use the else statement with the for loop which can be executed only when all the iterations are exhausted. Here, we must notice that if the loop contains any of the break statement then the else statement will not be executed.

EXAMPLE 1

```
for i in range(0,5):
    print(i)
else:print("for loop completely exhausted, since there is no break.");
```

In the above example, for loop is executed completely since there is no break statement in the loop. The control comes out of the loop and hence the else block is executed.

EXAMPLE 2

```
for i in range(0,5):
    print(i)
    break;
else:print("for loop is exhausted");
print("The loop is broken due to break statement...came out of loop")
```

In the above example, the loop is broken due to break statement therefore the else statement will not be executed. The statement present immediate next to else block will be executed.

PYTHON WHILE LOOP

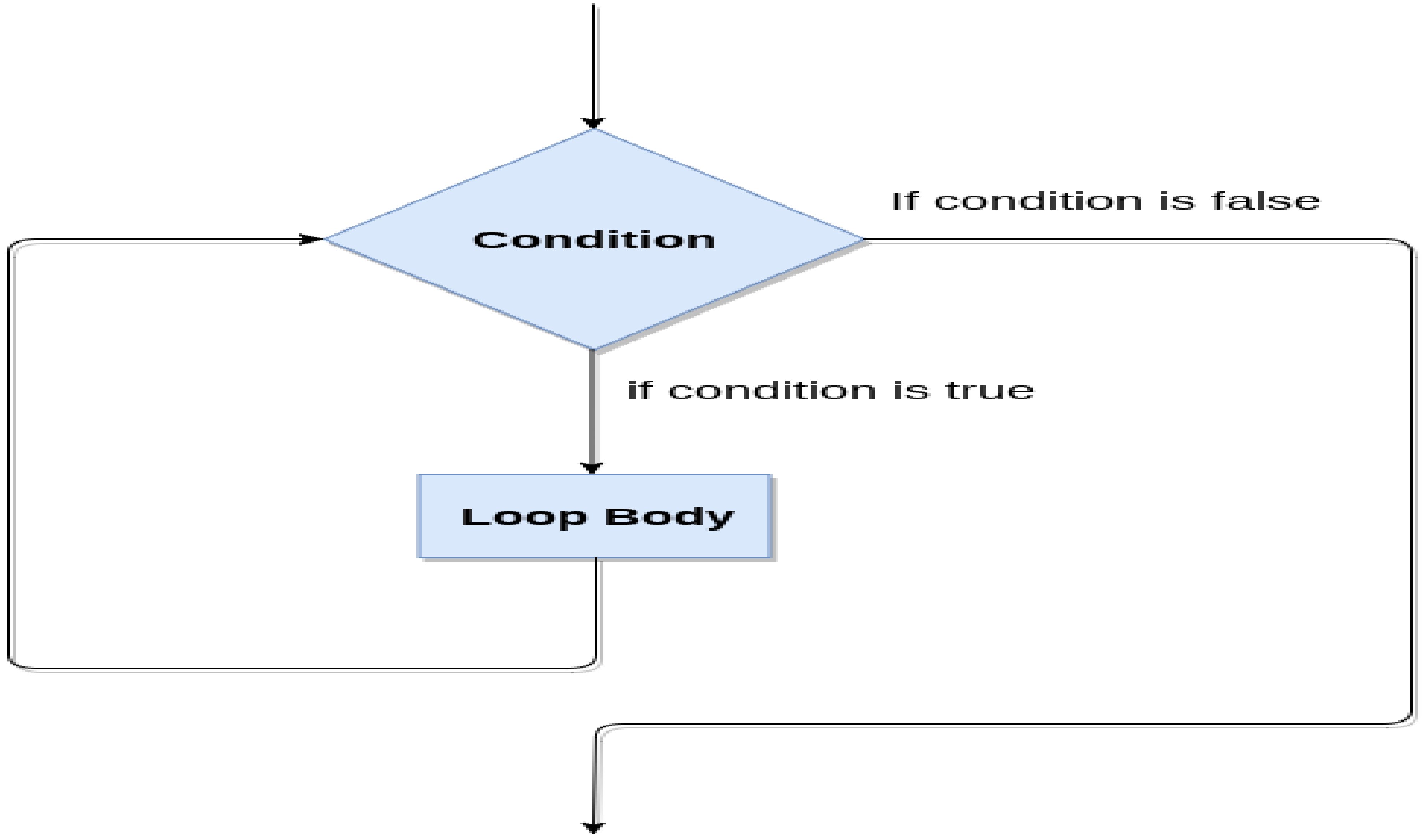
The while loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed as long as the given condition is true.

It can be viewed as a repeating if statement. The while loop is mostly used in the case where the number of iterations is not known in advance.

The syntax is given below.

```
while expression:
    statements
```

Here, the statements can be a single statement or the group of statements. The expression should be any valid python expression resulting into true or false. The true is any non-zero value.


Condition
If condition is false
if condition is true
Loop Body

EXAMPLE 1

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

EXAMPLE 2

```
i=1
number=0
b=9
number = int(input("Enter the number?"))
while i<=10:
    print("%d X %d = %d \n"%(number,i,number*i));
    i = i+1;
```

INFINITE WHILE LOOP

If the condition given in the while loop never becomes false then the while loop will never terminate and result into the infinite while loop.

Any non-zero value in the while loop indicates an always-true condition whereas 0 indicates the always-false condition. This type of approach is useful if we want our program to run continuously in the loop without any disturbance.

EXAMPLE 1

```
while (1):
    print("Hi! we are inside the infinite while loop");
```

EXAMPLE 2

```
var = 1
while var != 2:
    i = int(input("Enter the number?"))
    print ("Entered value is %d"%(i))
```

USING ELSE WITH PYTHON WHILE LOOP

Python enables us to use the while loop with the while loop also. The else block is executed when the condition given in the while statement becomes false. Like for loop, if the while loop is broken using break statement, then the else block will not be executed and the statement present after else block will be executed.

Consider the following example.

```
i=1;
while i<=5:
    print(i)
    i=i+1;
else:print("The while loop exhausted");
```

EXAMPLE 2

```
i=1;
while i<=5:
    print(i)
    i=i+1;
    if(i==3):
        break;
else:print("The while loop exhausted");
```

PYTHON BREAK STATEMENT

The break is a keyword in python which is used to bring the program control out of the loop. The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. In other words, we can say that break is used to abort the current execution of the program and the control goes to the next line after the loop.

The break is commonly used in the cases where we need to break the loop for a given condition.

The syntax of the break is given below.

#loop statements

break;

EXAMPLE 1

```
list =[1,2,3,4]
count = 1;
for i in list:
    if i == 4:
        print("item matched")
        count = count + 1;
        break
print("found at",count,"location");
```

EXAMPLE 2

```
str = "python"
for i in str:
    if i == 'o':
        break
    print(i);
```

EXAMPLE 3: BREAK STATEMENT WITH WHILE LOOP

```
i = 0;
while 1:
    print(i," ",end=""),
    i=i+1;
    if i == 10:
        break;
print("came out of while loop");
```

EXAMPLE 3

```
n=2
while 1:
    i=1;
    while i<=10:
        print("%d X %d = %d\n"%(n,i,n*i));
        i = i+1;
    choice = int(input("Do you want to continue printing the table, press 0 for no?"))
    if choice == 0:
        break;
    n=n+1
```

PYTHON CONTINUE STATEMENT

The continue statement in python is used to bring the program control to the beginning of the loop. The continue statement skips the remaining lines of code inside the loop and start with the next iteration. It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.

The syntax of Python continue statement is given below.

#loop statements

continue;

#the code to be skipped

EXAMPLE 1

```
i = 0;
while i!=10:
    print("%d"%i);
    continue;
    i=i+1;
```

EXAMPLE 2

```
i=1; #initializing a local variable
#starting a loop from 1 to 10
for i in range(1,11):
    if i==5:
        continue;
    print("%d"%i);
```

PASS STATEMENT

The pass statement is a null operation since nothing happens when it is executed. It is used in the cases where a statement is syntactically needed but we don't want to use any executable statement at its place.

For example, it can be used while overriding a parent class method in the subclass but don't want to give its specific implementation in the subclass.

Pass is also used where the code will be written somewhere but not yet written in the program file.

The syntax of the pass statement is given below.

```
list = [1,2,3,4,5]
flag = 0
for i in list:
    print("Current element:",i,end=" ");
    if i==3:
        pass;
        print("\nWe are inside pass block\n");
        flag = 1;
    if flag==1:
        print("\nCame out of pass\n");
        flag=0;
```

PYTHON PASS

In Python, pass keyword is used to execute nothing; it means, when we don't want to execute code, the pass can be used to execute empty. It is same as the name refers to. It just makes the control to pass by without executing any code. If we want to bypass any code pass statement can be used.

Python Pass Syntax

pass

```
for i in [1,2,3,4,5]:
    if i==3:
        pass
        print "Pass when value is",i
    print i,
```

CONTINUE IN NEXT UNIT .....