Unit II: Types, Operators and Expressions

Types - Integers, Strings, Booleans; Operators- Arithmetic, Comparison (Relational), Assignment, Logical, Bitwise, Membership, Identity, Precedence, Control Flow- if-elif-else, for, while, break, continue, loops.

Types - Integers, Strings, Booleans

Python Data Types

Every value has a datatype, and variables can hold values.

a = 5

- We did not specify the type of the variable a, which has the value five from an integer. The Python interpreter **will automatically interpret the variable as an integer**.
- The **type() function** in Python returns the type of the passed variable.

Example

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

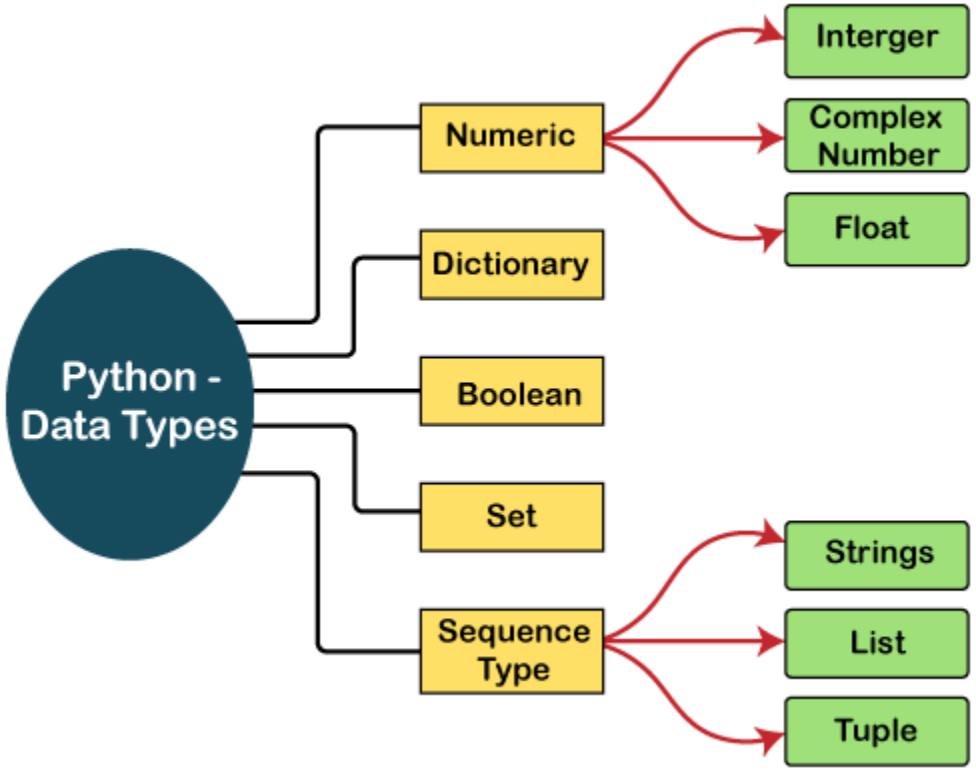
Output:

```
<type 'int'>
<type 'str'>
<type 'float'>
```

Standard data types

The storage method for each of the standard data types that Python provides is specified by Python. The following is a list of the Python-defined data types.

- Numbers
- Sequence Type
- Boolean
- Set
- Dictionary



Python Data Types

1. Numbers

Numeric values are stored in numbers. The whole number float, and complex qualities have a place with a Python Numbers datatype. Python offers **the type() function to determine a variable's data type**. The **instance () capability is utilized to check whether an item has a place with a specific class**.

When a number is assigned to a variable, Python generates **Number objects**. For instance,

```
a = 5
print("The type of a", type(a))
b = 40.5
print("The type of b", type(b))
c = 1+3j
print("The type of c", type(c))
print(" c is a complex number", isinstance(1+3j,complex))
```

In [3]:

```
1 a = 5
2 print("The type of a", type(a))
3 b = 40.5
4 print("The type of b", type(b))
5 c = 1+3j
6 print("The type of c", type(c))
7 print(" c is a complex number", isinstance(1+3,complex))
```

```
The type of a <class 'int'>
The type of b <class 'float'>
The type of c <class 'complex'>
 c is a complex number False
```

In [13]:

```
1 a = 5
2 print("The type of a", type(a))
3 b = 40.5
4 print("The type of b", type(b))
5 c = 1+3j
6 print("The type of c", type(c))
7 print(" c is a complex number", isinstance(1+3j,complex))
```

```
The type of a <class 'int'>
The type of b <class 'float'>
The type of c <class 'complex'>
 c is a complex number True
```

Python supports three kinds of numerical data.

Int: Whole number worth can be any length, like numbers 10, 2, 29, - 20, - 150, and so on. An integer can be **any length** you want in Python. Its worth has a place with int.

Float: Float stores drifting point numbers like 1.9, 9.902, 15.2, etc. It can be accurate to within 15 decimal places.

Complex: An intricate number contains an arranged pair, i.e., x + iy, where x and y signify the genuine and non-existent parts separately. The complex numbers like 2.14j, 2.0 + 2.3j, etc.

String

The sequence of **characters in the quotation marks** can be used to describe the string. A string can be defined in Python using **single, double, or triple quotes**.

When dealing with strings, the operation "hello"+" python" returns "hello python," and the operator + is used to combine two strings.

.

```
In [5]:  1 print("hello"+"Python")
         2 print("hello","Python")

helloPython
hello Python
```

The Python string is demonstrated in the following example.

Example - 1

```
str = "string using double quotes"
print(str)
s = '''A multiline
string'''
print(s)
```

In [6]:

```
1 str = "string using double quotes"
2 print(str)
3 s = '''''A multiline
4 string'''
5 print(s)
```

```
string using double quotes
''A multiline
string
```

Example - 2

```
str1 = 'hello Python'
str2 = ' how are you'
print (str1[0:2])
print (str1[4])
print (str1*2)
print (str1 + str2)
```

In [14]:

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

```
he
o
how are youhow are you
hello Pythonhow are you
```

Program for Explicit Type Conversion

In [15]:

```
#convert from int to float:
x = float(1)

#convert from float to int:
y = int(2.8)

#convert from int to complex:
z = complex(1)

print(x)
print(y)
print(z)

print(type(x))
print(type(y))
print(type(z))
```

```
1.0
2
(1+0j)
<class 'float'>
<class 'int'>
<class 'complex'>
```

Python - Slicing Strings

In [1]:

```
#slicing
b = "Hello, World!"
print(b[2:5])
#slice from start
b = "Hello, World!"
print(b[:5])
#slice to the end
b = "Hello, World!"
print(b[2:])
#Negative Indexing
b = "Hello, World!"
print(b[-5:-2])
```

```
llo
Hello
llo, World!
orl
```

Boolean

True and False are the two default values for the Boolean type. These qualities are utilized to decide the given assertion valid or misleading. The class book indicates this. **False can be represented by the 0,** while true can be represented **by any value that is not zero.**

Look at the following example.

Python program to check the boolean type

In [4]:

```
print(type(True))
print(type(False))
x=(1==True)#Gives value True because 1 is considered as True
y=(2==False)
z=(0==False)#Gives value True because 0 is considered as False
c=(4==True)
print(x,y,z,c)
if(2):
    print("hello")
if(0):
    print("Python")
```

```
<class 'bool'>
<class 'bool'>
True False True False
hello
```

Python - String Methods

In [5]:

```python
#Upper Case
a = "Hello, World!"
print("Upper Case")
print(a.upper())

#Lower Case
a = "Hello, World!"
print("\nLower Case")
print(a.lower())

#Remove Whitespace
a = " Hello, World! "
print("\nRemove Whitespace")
print(a.strip()) # returns "Hello, World!"

#Replace String
a = "Hello, World!"
print("\nReplace String")
print(a.replace("H", "J"))

#Split String
a = "Hello, World!"
print("\nSplit String")
print(a.split(",")) # returns ['Hello', ' World!']

```

```
Upper Case
HELLO, WORLD!

Lower Case
hello, world!

Remove Whitespace
Hello, World!

Replace String
Jello, World!

Split String
['Hello', ' World!']
```

```python
#Capitalize String
txt = "python is FUN!"
x = txt.capitalize()
print("\nCapitalize String")
print (x)

#Casefold String
txt = "Hello, And WELCOME To My World!"
x = txt.casefold()
print("\nCasefold String")
print(x)

#Center String
txt = "banana"
x = txt.center(20,'*')
print("\nCenter String")
print(x)

#SwapCase String
txt = "Hello My Name Is PETER"
x = txt.swapcase()
print("\nSwapCase String")
print(x)

#Strip String
txt = "     banana     "
x = txt.strip()
print("\nStrip String")
print("of all fruits", x, "is my favorite")
```

```
Capitalize String
Python is fun!

Casefold String
hello, and welcome to my world!

Center String
*******banana*******

SwapCase String
hELLO mY nAME iS peter

Strip String
of all fruits banana is my favorite
```

```python
#StartWith function String
txt = "Hello, welcome to my world."
x = txt.startswith("life")
print("\nStartWith function String")
print(x)

#zfill method strings
txt = "50"
x = txt.zfill(10)
print("\nzfill method")
print(x)

#Split function strings
txt = "welcome to the jungle"
x = txt.split()
print("\nSplit function strings")
print(x)

#Count function strings
txt = "I love apples, apple are my favorite fruit"
x = txt.count("apple")
print("\nCount")
print(x)

```

```
StartWith function String
False

zfill method
0000000050

Split function strings
['welcome', 'to', 'the', 'jungle']

Count
2
```

```python
#endswith()→Returns true if the string ends with the specified value
txt = "Hello, welcome to my world."
x = txt.endswith(".")
print("\nendswith()")
print(x)

#expandtabs()→Sets the tab size of the string
txt = "H\te\tl\tl\to"
x =  txt.expandtabs(2)
print("\nexpandtabs()")
print(x)

#find()->Searches the string for a specified value and returns the position of where it was found
txt = "Hello, welcome to my world."
x = txt.find("welcome")
print("\nfind()")
print(x)

#isalnum()→Returns True if all characters in the string are alphanumeric
txt = "Company12"
x = txt.isalnum()
print("\nisalnum()")
print(x)

#isalpha()→Returns True if all characters in the string are in the alphabet
txt = "CompanyX"
x = txt.isalpha()
print("\nisalpha()")
print(x)
```

```
endswith()
True

expandtabs()
H e l l o

find()
7

isalnum()
True

isalpha()
True
```

```
#isdecimal()→Returns True if all characters in the string are decimals
txt = "1234"
x = txt.isdecimal()
print("\nisdecimal()")
print(x)

#islower()→Returns True if all characters in the string are lower case
txt = "hello world!"
x = txt.islower()
print("\nislower()")
print(x)

#isnumeric()→Returns True if all characters in the string are numeric
txt = "565543"
x = txt.isnumeric()
print("\nisnumeric()")
print(x)

# isprintable()→Returns True if all characters in the string are printable
txt = "Hello!\nAre you #1?"
x = txt.isprintable()
print("\nisprintable()")
print(x)

#isspace()→Returns True if all characters in the string are whitespaces
txt = "   "
x = txt.isspace()
print("\nisspace()")
print(x)
```

```
isdecimal()
True

islower()
True

isnumeric()
True

isprintable()
False

isspace()
True
```

```python
#istitle()→Returns True if the string follows the rules of a title
txt = "Hello, And Welcome To My World!"
x = txt.istitle()
print("\nistitle()")
print(x)

#isupper()→Returns True if all characters in the string are upper case
txt = "THIS IS NOW!"
x = txt.isupper()
print("\nisupper()")
print(x)

#partition()——→Returns a tuple where the string is parted into three parts
txt = "I could eat bananas all day"
x = txt.partition("bananas")
print("\npartition()")
print(x)

#join()→Joins the elements of an iterable to the end of the string
myTuple = ("John", "Peter", "Vicky")
x = "#".join(myTuple)
print("\njoin()")
print(x)

```

```
istitle()
True

isupper()
True

partition()
('I could eat ', 'bananas', ' all day')

join()
John#Peter#Vicky
```

```python
#String Format
#1
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))

#2
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))

#3
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))

#Python Escape characters
txt = "We are the so-called \"Vikings\" from the north."
print(txt)

```

```
My name is John, and I am 36
I want 3 pieces of item 567 for 49.95 dollars.
I want to pay 49.95 dollars for 3 pieces of item 567.
We are the so-called "Vikings" from the north.
```

Operators- Arithmetic, Comparison (Relational), Assignment, Logical, Bitwise, Membership, Identity, Precedence

Arithmetic Operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

```python
#Arithmetic and Assignment Operators

# //	Floor division	x // y
x = 15
y = 2
print(x // y)
# Output: 7
# the floor division // rounds the result down to the nearest whole number

# &=
x = 5
x &= 3
print(x)
# Output: 1

# <<=
x = 5
x <<= 3
print(x)
# Output: 1

# >>=
x = 5
x >>= 3
print(x)
# Output: 0

# ^=
x = 5
x ^= 3
print(x)
# Output: 6

# |=
x = 5
x |= 3
print(x)
# Output: 7
```

Python Bitwise Operators

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

```python
# Bitwise Operators

# >>(Right Shift)
print(8 >> 2)
# Output: 2

# <<(Left Shift)
print(3 << 2)
# Output: 12

# ~(NOT)
print(~3)
# Output: -4
"""
The ~ operator inverts each bit (0 becomes 1 and 1 becomes 0).
Inverted 3 becomes -4:
 3 = 0000000000000011
-4 = 1111111111111100
"""

# ^(XOR)
print(6 ^ 3)
# Output: 5

# |(Bitwise OR)
print(6 | 3)
# Output: 7
```

```python
# &(Bitwise AND)
print(6 & 3)
# Output: 2
```

Python Comparison Operators

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

```python
#COMPARISON OPERATORS

# ==
x = 5
y = 3
print(x == y)
# returns False because 5 is not equal to 3

# !=
x = 5
y = 3
print(x != y)
# returns True because 5 is not equal to 3

# >
x = 5
y = 3
print(x > y)
# returns True because 5 is greater than 3

# <
x = 5
y = 3
print(x < y)
# returns False because 5 is not less than 3

# >=
x = 5
y = 3
print(x >= y)
# returns True because five is greater, or equal, to 3

# <=
x = 5
y = 3
print(x <= y)
# returns False because 5 is neither less than or equal to 3

```

Python Logical Operators

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Identity Operators

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Python Membership Operators

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

```python
# Membership, Identity and Logical operators

# in operator
x = ["apple", "banana"]
print("banana" in x)
# returns True because a sequence with the value "banana" is in the list

# not in
x = ["apple", "banana"]
print("pineapple" not in x)
# returns True because a sequence with the value "pineapple" is not in the list

# is
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is z)
# returns True because z is the same object as x
print(x is y)
# returns False because x is not the same object as y, even if they have the same content
print(x == y)
# to demonstrate the difference betweeen "is" and "==": this comparison returns True because x is equal to y
```

```python
#is not
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x
print(x is not z)
# returns False because z is the same object as x
print(x is not y)
# returns True because x is not the same object as y, even if they have the same content
print(x != y)
# to demonstrate the difference betweeen "is not" and "!=": this comparison returns False because x is equal to y

# not
x = 5
print(not(x > 3 and x < 10))
# returns False because not is used to reverse the result

# or
x = 5
print(x > 3 or x < 4)
# returns True because one of the conditions are true (5 is greater than 3, but 5 is not less than 4)

# and
x = 5
print(x > 3 and x < 10)
# returns True because 5 is greater than 3 AND 5 is less than 10
```

Operator Precedence

Operator	Description
()	Parentheses
**	Exponentiation
+x -x ~x	Unary plus, unary minus, and bitwise NOT
* / // %	Multiplication, division, floor division, and modulus
+ -	Addition and subtraction
<< >>	Bitwise left and right shifts
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
== != > >= < <= is is not in not in	Comparisons, identity, and membership operators
not	Logical NOT
and	AND
or	OR

If two operators have the same precedence, the expression is evaluated from left to right.

```python
#Operator Precedence Table
#()	Parentheses
print((6 + 3) - (6 + 3))
"""
Parenthesis have the highest precedence, and need to be evaluated first.
The calculation above reads 9 - 9 = 0
"""

#**	Exponentiation
print(100 - 3 ** 3)
"""
Exponentiation has higher precedence than subtraction, and needs to be evaluated first.
The calculation above reads 100 - 27 = 73
"""

#+x  -x  ~x	Unary plus, unary minus, and bitwise NOT
print(100 + ~3)

"""
Bitwise NOT has higher precedence than addition, and needs to be evaluated first.
The calculation above reads 100 + -4 = 96
"""
```

```python
#*  /  //  %    Multiplication, division, floor division, and modulus
print(100 + 5 * 3)

"""
Multiplication has higher precedence than addition, and needs to be evaluated first.
The calculation above reads 100 + 15 = 115
"""

#+  -    Addition and subtraction
print(100 - 5 * 3)

"""
Subtraction has a lower precedence than multiplication, and we need to calculate the multiplication first.
The calculation above reads 100 - 15 = 85
"""
# <<  >>    Bitwise left and right shifts
print(8 >> 3 - 2)
"""
Bitwise right shift has a lower precedence than subtraction, and we need to calculate the subtraction first.
The calculation above reads 8 >> 2 = 2
More explanation:
The >> operator moves each bit the specified number of times to the right. Empty holes at the left are filled with 0's.
If you move each bit 2 times to the right, 8 becomes 2:
 8 = 0000000000001000
becomes
 2 = 0000000000000010
 """
```

```python
# &Bitwise AND
print(6 & 2 + 1)
"""
Bitwise AND has a lower precedence than addition, and we need to calculate the addition first.
The calculation above reads 6 & 3 = 2
More explanation:
The & operator compares each bit and set it to 1 if both are 1, otherwise it is set to 0:
6 = 0000000000000110
3 = 0000000000000011
--------------------
2 = 0000000000000010
====================
"""

# ^Bitwise XOR
print(6 ^ 2 + 1)
"""
Bitwise XOR has a lower precedence than addition, and we need to calculate the addition first.
The calculation above reads 6 ^ 3 = 5
More explanation:
The ^ operator compares each bit and set it to 1 if only one is 1, otherwise (if both are 1 or both are 0) it is set to 0:
6 = 0000000000000110
3 = 0000000000000011
--------------------
5 = 0000000000000101
====================
"""
```

```python
# |→Bitwise OR
print(6 | 2 + 1)
"""
Bitwise OR has a lower precedence than addition, and we need to calculate the addition first.
The calculation above reads 6 | 3 = 7
More explanation:
The | operator compares each bit and set it to 1 if one or both is 1, otherwise it is set to 0:
6 = 0000000000000110
3 = 0000000000000011
--------------------
7 = 0000000000000111
====================
"""

# ==  !=  >  >=  <  <=  is  is not in  not in →Comparisons, identity, and membership operators
print(5 == 4 + 1)
"""
The "like" comparison has a lower precedence than addition, and we need to calculate the addition first.
The calculation above reads 5 == 5 = True
"""
# not→Logical NOT
print(not 5 == 5)
"""
The logical NOT operator has a lower precedence than "like" comparison, and we need to calculate the comparison first.
The calculation above reads: not True = False
"""
```

```
# and ⟶ AND
print(1 or 2 and 3)

"""
The and operator has a higher precedence than or, and we need to calculate the and expression first.
The calculation above reads: 1 or 3 = 1
"""
# or ⟶ OR
print(4 or 5 + 10 or 8)

#Output: 4
```

More Examples on Operator Precedence

```
188 print(5 == 4 + 1)
189 print(not 5 == 5)
190 print(1 or 2 and 3)
191 print(4 or 5 + 10 or 8)
192 print(6 | 2 + 1)
193 print(6 ^ 2 + 1)
194 print(6 & 2 + 1)
195 print(8 >> 4 - 2)
196 print(100 - 5 * 3)
197 print(100 + ~3)
198 print(100 - 3 ** 3)
199 print((6 + 3) - (6 + 3))
```

Output:

```
True
False
1
4
7
5
2
2
85
96
73
0
```

Control Flow- if-elif-else, for, while, break, continue, loops

Python If-else statements

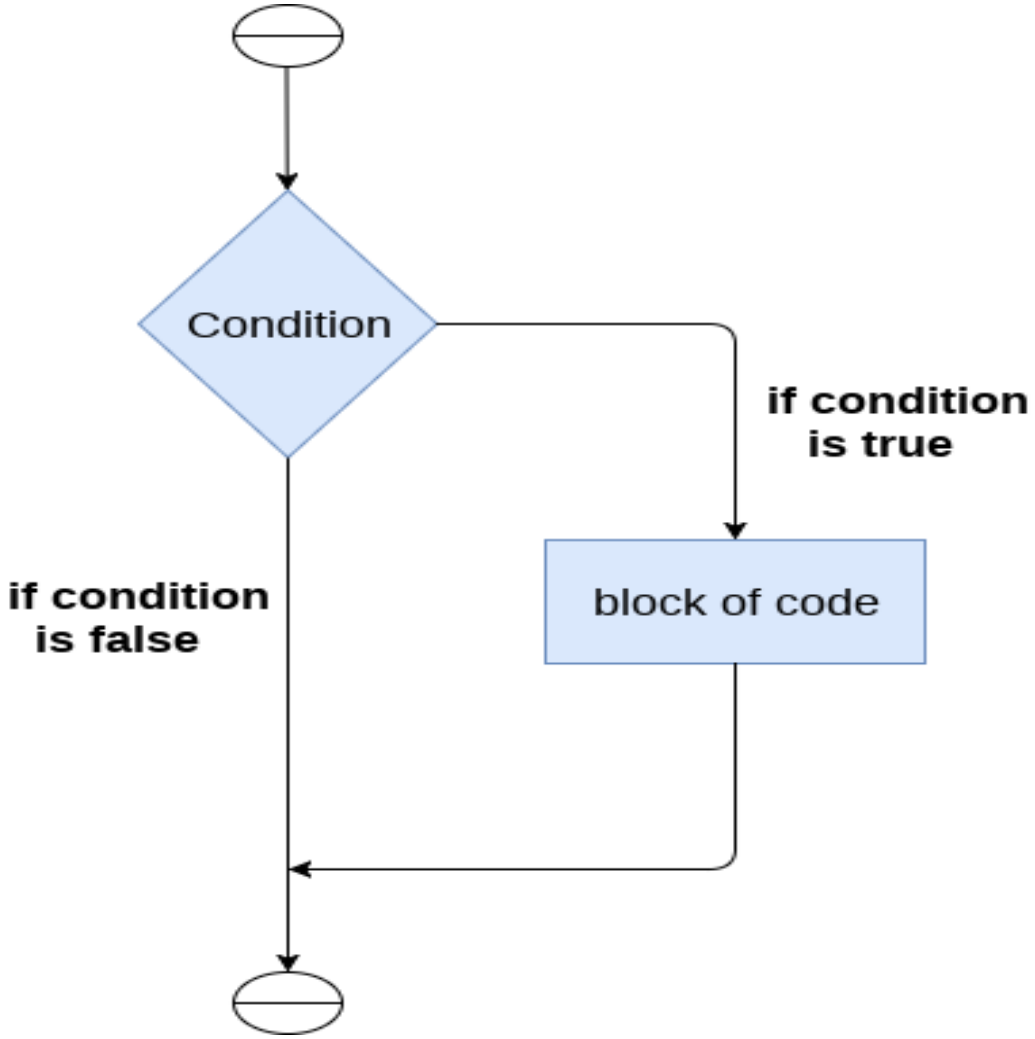
Decision making is the most important aspect of almost all the programming languages. As the name implies, decision making allows **us to run a particular block of code for a particular decision**. Here, the decisions are made on **the validity of the particular conditions**.

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.

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 **syntax** of the if-statement is given below.

```
if expression:
    statement
```

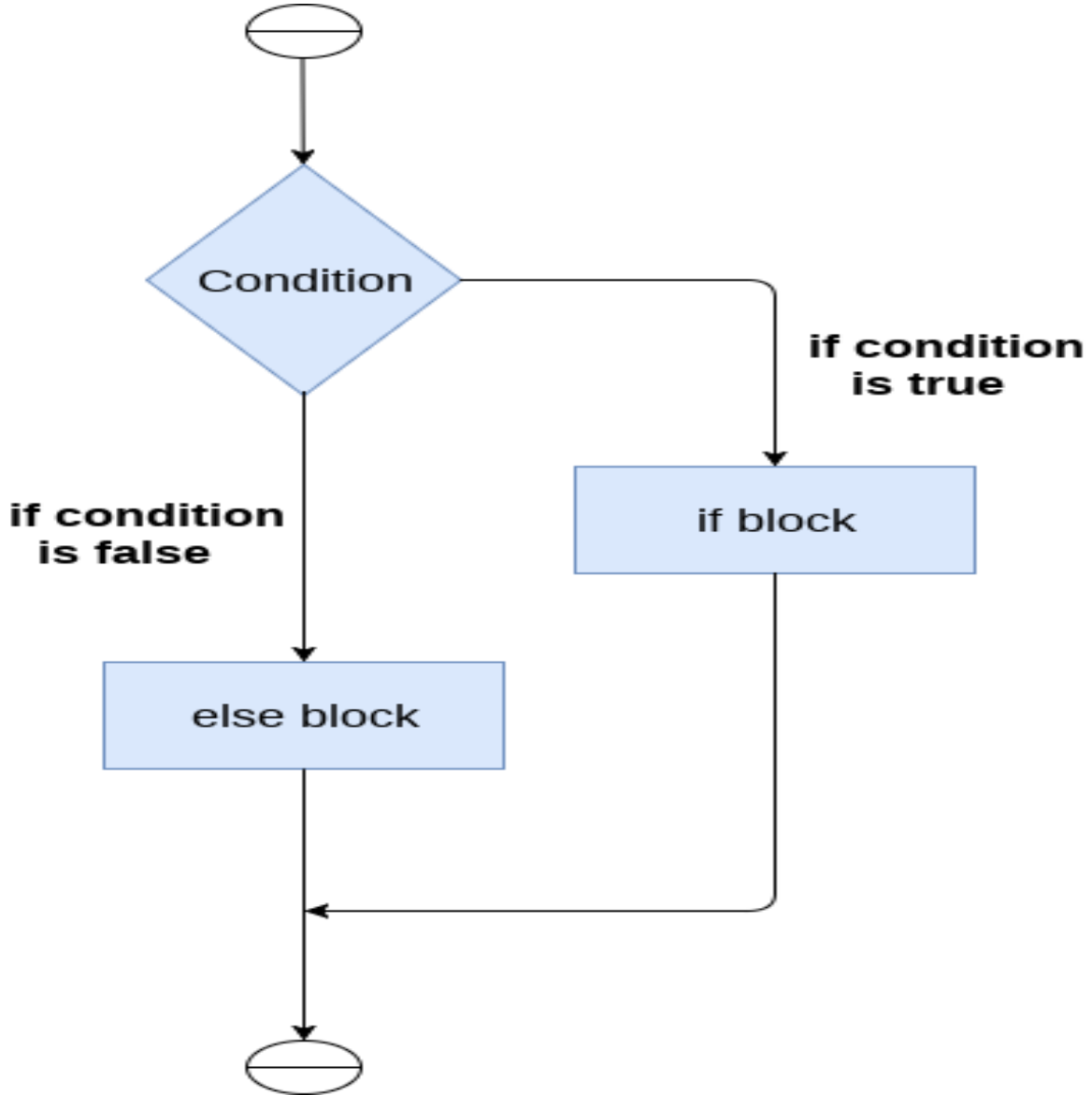
The If Statement

In [3]:

```
# Simple Python program to understand the if statement

# To check if the no is even
num = int(input("enter the number:"))
if num%2 == 0:
    print("The Given number is an even number")

#To check the greatest of Three numbers
a = int (input("Enter a: "));
b = int (input("Enter b: "));
c = int (input("Enter c: "));
if a>b and a>c:

    print ("From the above three numbers given a is largest");
if b>a and b>c:

    print ("From the above three numbers given b is largest");
if c>a and c>b:

    print ("From the above three numbers given 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.**



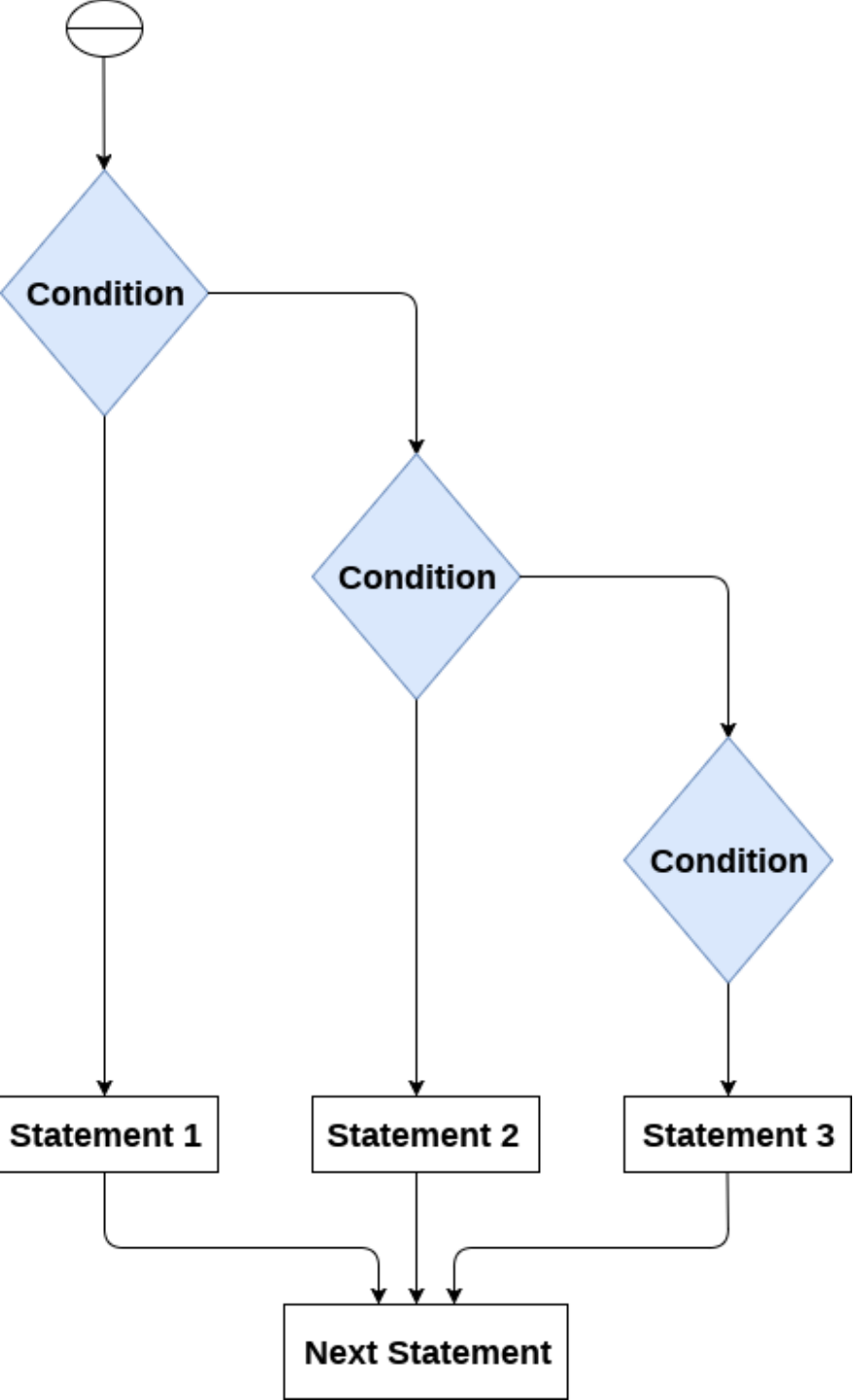
The syntax of the **if-else statement** is given below.

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

The if-else statement

In [6]:

```
#Example 1 : Program to check whether a person is eligible to vote or not.

age = int (input("Enter your age: "))

if age>=18:

    print("ELIGIBLE!!");
else:
    print("INELIGLIBLE");

Example 2: Program to check whether a number is even or not.
# Simple Python Program to check whether a number is even or not.
num = int(input("enter the number:"))
# Here, we are taking an integer num and taking input dynamically
if num%2 == 0:
# Here, we are checking the condition. If the condition is true, we will enter the block
    print("The Given number is an even number")
else:
    print("The Given Number is an odd number")
```



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.

The elif statement works like **an if-else-if ladder statement in C.**

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
```

The elif statement

In [7]:

```
#Example 1

number = int(input("Enter the number?"))

if number==10:

    print("The given number is equals to 10")
elif number==50:

    print("The given number is equal to 50");
elif number==100:

    print("The given number is equal to 100");
else:
    print("The given number is not equal to 10, 50 or 100");

```

```
#Example 2

marks = int(input("Enter the marks? "))

if marks > 85 and marks <= 100:

   print("Your grade is 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("FAIL")
```

Nested if Statement

In [9]:

```
#Example: Python Nested if Statement
number = 5

if (number >= 0):

    if number == 0:
        print('Number is 0')

    else:
        print('Number is positive')

else:
    print('Number is negative')
```

Python for loop

Python frequently uses the Loop to iterate over iterable objects **like lists, tuples, and strings**. Crossing is the most common way of emphasizing across a series, for loops are used when a section of code needs to be repeated a certain number of times. **In Python, the for Statement runs the code block each time it traverses a series of elements. On the other hand, the "while" Loop is used when a condition needs to be verified after each repetition or when a piece of code needs to be repeated indefinitely**.

Syntax of for Loop

```
for value in sequence:
   {loop body}
```

The value is the parameter **that determines the element's value within the iterable sequence on each iteration**. Using indentation, the contents of the Loop are distinguished from the remainder of the program.

The range() Function

It is a built-in Python method that fulfills the requirement of providing a series for the for expression **to run over by following a particular pattern (typically serial integers).** Mainly, they can act straight on sequences, so counting is unnecessary.

Loops

In [2]:
```
# Code to find the sum of squares of each element of the list using for loop

numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]
sum_ = 0
for num in numbers:

    sum_= sum_ + num ** 2

print("The sum of squares is: ", sum_)
```

```
The sum of squares is:  774
```

With range() function

In [6]:
```
# Code to find the sum of squares of each element of the list using for loop
numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]

sum_ = 0

for num in range( len(numbers) ):

    sum_ = sum_ + numbers[num] ** 2

print("The sum of squares is: ", sum_)
```

```
The sum of squares is:  774
```

Using else Statement with for Loop

In [9]:

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

```
0
1
2
3
4
5
Finally finished!
```

Nested For Loops

In [11]:
```
adj = ["red", "big", "tasty"]
fruits = ["apple", "kiwi", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

```
red apple
red kiwi
red cherry
big apple
big kiwi
big cherry
tasty apple
tasty kiwi
tasty cherry
```

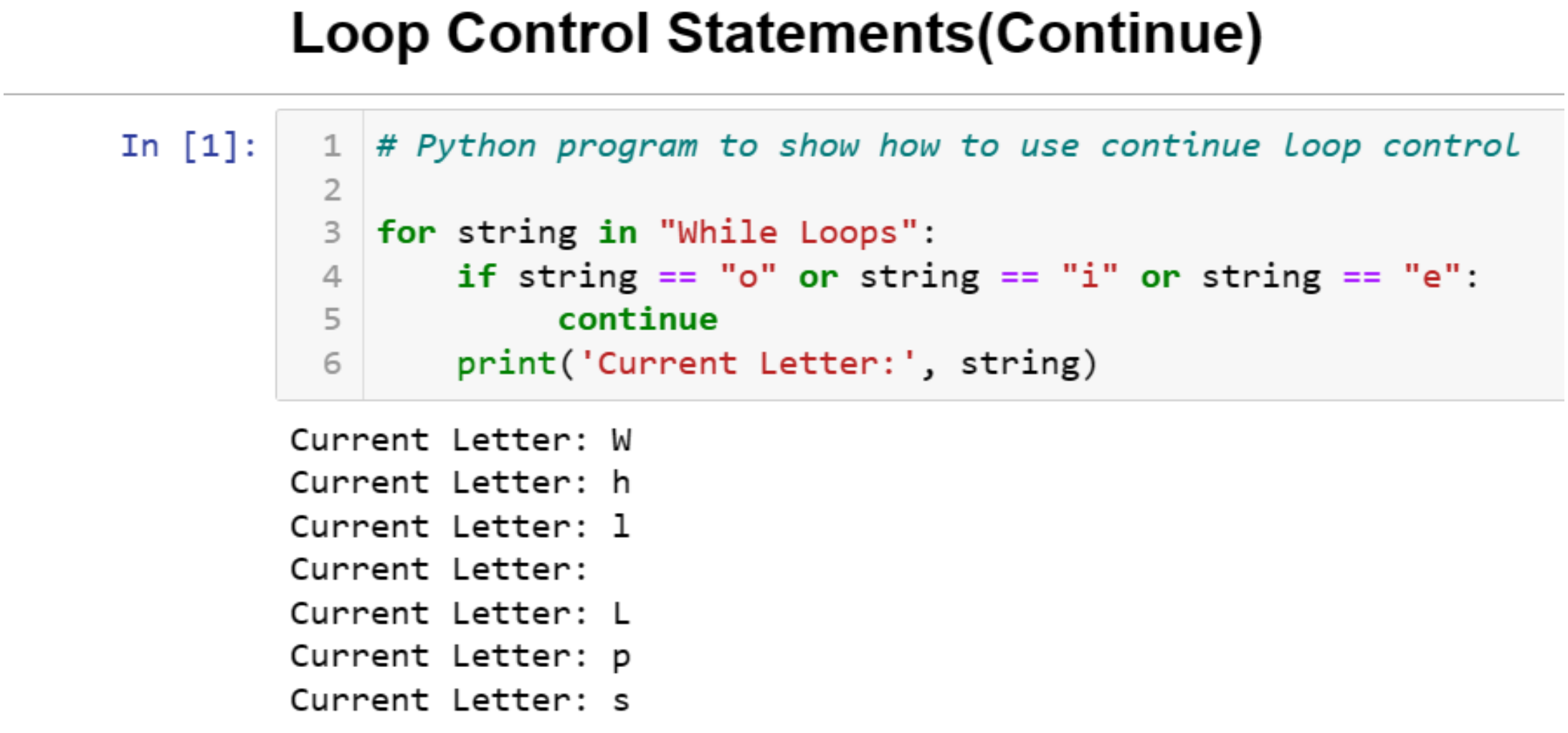
In [12]:
```
import random
numbers = [ ]
for val in range(0, 11):
    numbers.append( random.randint( 0, 11 ) )
for num in range( 0, 11 ):
    for i in numbers:
        if num == i:
            print( num, end = " " )
```

```
1 3 4 4 6 8 8 8 10
```

Loop Control Statements

1. Continue Statement

It **returns the control** of the Python interpreter to **the beginning of the loop**.



2. Break Statement

It **stops the execution of the loop** when the break statement is reached.

In [2]:

```
# BREAK STATEMENT
# Python program to show how to use the break statement

for string in "Python Loops":
    if string == 'n':
        break
    print('Current Letter: ', string)
```

```
Current Letter:  P
Current Letter:  y
Current Letter:  t
Current Letter:  h
Current Letter:  o
```

3. Pass Statement

In Python programming, the pass statement is a null statement that does nothing. The pass statement is used to create loops, if…else statement, functions & classes with an empty body.

Using pass With Conditional Statement

In [9]:
```
1  n = 10
2  # use pass inside if statement
3  if n > 10:
4      pass
5  print('Hello')
```

```
Hello
```

In [10]:
```
1  n = 10
2  # use pass inside if statement
3  if n > 10:
4
5  print('Hello')
```

```
  Input In [10]
    print('Hello')
    ^
IndentationError: expected an indented block
```

In [11]:

```
# Python program to show how to create an empty function and an empty class

# Empty function:
def empty():
    pass

# Empty class
class Empty:
    pass
```

In [12]:

```
# Python program to show how to create an empty function and an empty class

# Empty function:
def empty():


# Empty class
class Empty:
    pass
```

```
  Input In [12]
    class Empty:
    ^
IndentationError: expected an indented block
```

Python Continue vs. Pass

Headings	continue	pass
Definition	The continue statement is utilized to skip the current loop's remaining statements, go to the following iteration, and return control to the beginning.	The pass keyword is used when a phrase is necessary syntactically to be placed but not to be executed.
Action	It takes the control back to the start of the loop.	Nothing happens if the Python interpreter encounters the pass statement.
Application	It works with both the Python while and Python for loops.	It performs nothing; hence it is a null operation.
Syntax	It has the following syntax: -: continue	Its syntax is as follows:- pass
Interpretation	It's mostly utilized within a loop's condition.	During the byte-compile stage, the pass keyword is removed.

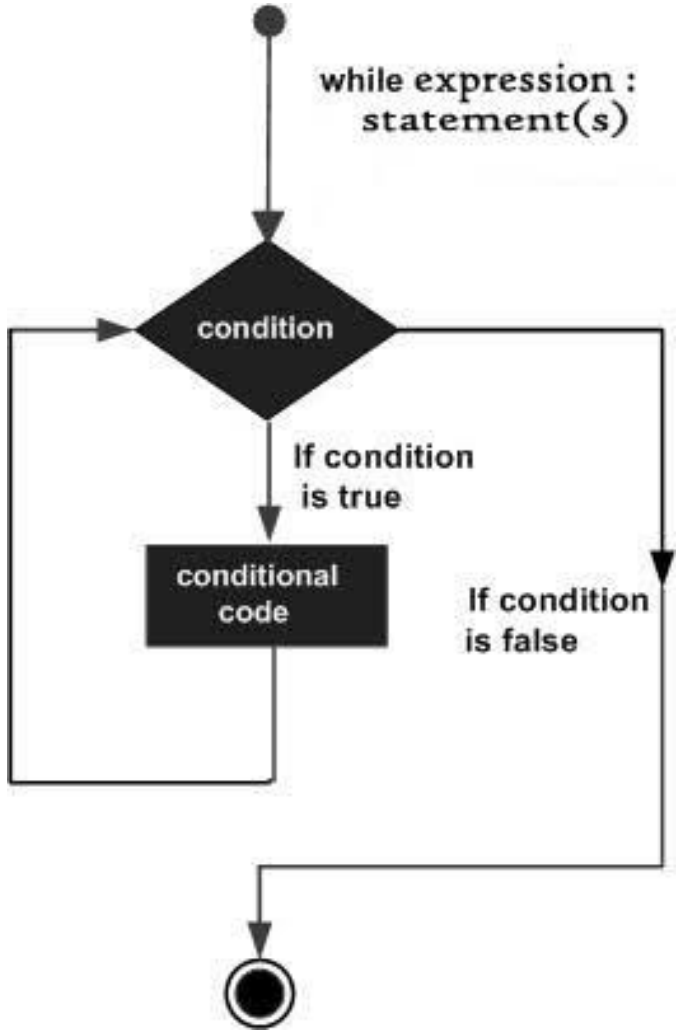
Python While Loop

Python While Loop is used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.

Syntax:

```
while expression:
    statement(s)
```

Flowchart of While Loop :



In [1]:

```
# while loop
count = 0
while (count < 3):
    count = count + 1
    print("Hello Python")
```

```
Hello Python
Hello Python
Hello Python
```

In [7]:

```
# checks if list still contains any element
a = [1, 2, 3, 4]

while a:
    print(a.pop())
if a==[]:
    print("Now List is empty")
```

```
4
3
2
1
Now List is empty
```

In [8]:

```
# Python program to illustrate
# Single statement while block
count = 0
while (count < 5): count += 1; print("Hello Python")
```

```
Hello Python
Hello Python
Hello Python
Hello Python
Hello Python
```

In [9]:

```python
# Python program to demonstrate while-else loop

i = 0
while i < 4:
    i += 1
    print(i)
else:
    print("No Break\n")

i = 0
while i < 4:
    i += 1
    print(i)
    break
else:
    print("No Break")
```

```
1
2
3
4
No Break

1
```

Sentinel Controlled Statement

A sentinel value is a value that is used to terminate a loop whenever a user enters it, generally, the sentinel value is -1.

In [10]:

```
a = int(input('Enter a number (-1 to quit): '))

while a != -1:
    a = int(input('Enter a number (-1 to quit): '))
```

```
Enter a number (-1 to quit): 7
Enter a number (-1 to quit): 2
Enter a number (-1 to quit): 1
Enter a number (-1 to quit): -1
```

While loop with range() function

In [29]:

```
# range() with stop parameter
print("1st")
i=0
while i in range(10):
    print(i,end=" ")
    i=i+1

# range() with start and stop parameters
print("\n2nd")
i=0
while i in range(2,10):
    print(i,end=" ")
    i=i+1

# range() with all parameters
print("\n3rd")
i=0
while i in range(0,10,3):
    print(i,end=" ")
    i=i+3
```

```
1st
0 1 2 3 4 5 6 7 8 9
2nd

3rd
0 3 6 9
```

Print all the letters except p, a and m

Print all the letters except p, a and m

In [14]:

```
# Prints all letters except 'p', 'a' and 'm'
i = 0
a = 'Python Programming Language'

while i < len(a):
    if a[i] == 'p' or a[i] == 'a' or a[i] == 'A' or a[i] == 'P' or a[i] == 'm' or a[i] == 'M':
        i += 1
        continue

    print('Current Letter :', a[i])
    i += 1
```

```
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current Letter :
Current Letter : r
Current Letter : o
Current Letter : g
Current Letter : r
Current Letter : i
Current Letter : n
Current Letter : g
Current Letter :
Current Letter : L
Current Letter : n
Current Letter : g
Current Letter : u
Current Letter : g
Current Letter : e
```

WAP to break the loop as soon it sees 'a'

In [15]:

```python
# Break the loop as soon it sees 'a'

i = 0
a = 'Python Programming Language'

while i < len(a):
    if a[i] == 'a' or a[i] == 'A':
        i += 1
        break

    print('Current Letter :', a[i])
    i += 1
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current Letter :
Current Letter : P
Current Letter : r
Current Letter : o
Current Letter : g
Current Letter : r
```