Rules for Naming Variables

The symbol used for a variable is called an identifier or variable name. A set of rules are used for naming variables. These rules are:

- Valid Identifier is a sequence of one or more letters, **digits, or underscores characters (_).**
- Neither spaces nor punctuation marks or symbols can be part of an identifier. Only leters, digits, and single underscore characters are valid.
- Variable identifiers always have to begin with a letter. They can also begin with an underscore character (_).
- In no case, a variable can begin with a digit.
- A reserved Word of the Python language cannot be used as an identifier.
- Names should be meaningful.
- Name should not be too long.

Python language is a **"Case Sensitive"** language which means that an identifier written in capital letters is not equivalent to the one written in small letters. (age, Age, and AGE are three different variables)

For Example, **myVar** is not equal to **myvar.**

In [1]:
```
name = "Ikram "
b = 20
c = 3.5
print(name, b, c)
```
```
Ikram  20 3.5
```

Operators

Operators are the symbols which perform operation on operands. The operation may be arithmetic or comparison of data.

```
z = x + y;
```

Here, are x, y, and z are operands and ‘+’ and ‘=’ are the operators.
The set of Python operators is divided into two main categories: Unary operator, and Ternary Operator. This division is on the basis of a number of operands upon which the operators operate.
Unary Operators

Unary operators are those operators which need a single operand. There are two arithmetic unary operators: unary plus + and unary -. Consider the following examples:

```
x = -y;
a = +b;
```

The logical ! operators is also a unary operator. Consider the following examples:

```
!x;
!(x > y);
```

Binary Operators

Binary operators are those operators which need two operands. For example (+, -, *, /, %).

The assignment operator = is also a binary operator as it requires two operands.

The logical and relational operators (<, <=, ==, >, >=, !=, &&, ||) are binary operators as well, as they require two operands.

Python divides the operators in the following groups:

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

Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

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

In [2]:

```
# Arithmetic Operators
x = 12
y = 3
```

```
print(x + y)
print(x - y)
print(x * y)
print(x / y)
print(x % y)
print(x ** y)   # same as 12*12*12 = 1728
print(x // y)   # the floor division // rounds the result down to the nearest whole num
```

```
15
9
36
4.0
0
1728
4
```

In [3]:

```
x = 5
y = 2.1

print(x // y)
```

```
2.0
```

Assignment Operators

Assignment operators are used to assign values to variables:

> `(=(x = 5), +=(x = x + 4), -=(x = x - 4), *=(x = x * 4), /=(x = x / 4), %=(x = x % 4), //=(x = x // 4), **=(x = x ** 4))`

In [4]:

```
# Assignment Operators

x = 5
print(x)  # 5

x = 5
# x += 3
x = x + 3 # x = x + 3  = 8
print(x)

x = 5
x -= 3 # x = x - 3 = 2
print(x)

x = 5
x *= 3 # x = x * 3 = 15
print(x)
```

```
5
8
2
15
```

Comparison Operators

Comparison operators are used to compare two values:

```
==, !=, >, <, >=, <=
```

In [5]:

```
# 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
```

```
False
True
True
False
True
False
```

In [6]:

```
# x = 5
# y = 5
# print(x == y)
# print(x != y)

x = 5
y = 5
print(x >= y)
print(x > y)
```

```
True
False
```

Logical Operator

Logical operators are used to combine conditional statements:

```
and, or, not
```

In [7]:

```
# Logical Operators

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

x = 5
print(x > 3 or x < 4)  # returns True because one of the conditions are true (5 is grea

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

```
True
True
False
```

Identity Operatos

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

```
is, is not
```

In [8]:

```
# Identity Operators

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 h

print(x == y)    # to demonstrate the difference betweeen "is" and "==": this compariso
```

```
True
False
True
```

In [9]:

```
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 h

print(x != y) # to demonstrate the difference betweeen "is not" and "!=": this comparis
```

```
False
True
False
```

Membership Operator

Membership operators are used to test if a sequence is presented in an object:

```
in, not in
```

In [10]:

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

```
True
```

In [11]:

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

```
True
```

Bitwise Operators

Bitwise operators are used to compare (binary) numbers:
In python, bitwise operators are used to performing bitwise calculations on integers. The integers are first converted into binary and then operations are performed on bit by bit, hence the name bitwise operators. Then the result is returned in decimal format.

Note: Python bitwise operators work only on integers.

```
&, | ^, <<, >>
```

Bitwise AND(&) Operator

In [12]:

```
a = 10            # = 1010 (Binary)
b = 4             # =  0100 (Binary)

print(a & b)      # = 1010
                  #   &
                  #   0100
                  # = 0000
                  # = 0 (Decimal)
```

```
0
```

Bitwise OR(|) Operator

In [13]:

```
a = 10            # = 1010 (Binary)
b = 4             # =  0100 (Binary)

print(a | b)      # = 1010
                  #   |
                    #   0100
                    # = 1110
                    # = 14 (Decimal)
```

```
14
```

Bitwise XOR(^) Operator

In [14]:

```python
a = 10              # = 1010 (Binary)
b = 4               # = 0100 (Binary)

print(a ^ b)        # = 1010
                    #   ^
                    #   0100
                    # = 1110
                    # = 14 (Decimal)
```

```
14
```