Variables

- In Python, a variable is a named location that we can store data. Variables are used to store data that can be used by a program.
- A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:
- 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 and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the Python keywords.

Example

- Legal variable names, Only you can create variable like below.
- myname = 26
- my_name = 30.354
- _my_name = "Ali"
- myName = "Ali"
- MYNAME = "Ali"
- myname2 = "Ali"

Example

- Illegal variable names ,We can not create variable like below
- 2myname = 6
- my-name = 789.23
- my Name = "Ali"

Camel Case

- Each word, except the first, starts with a capital letter:
- myNameAli = "Ali"

Pascal Case

- Each word starts with a capital letter:
- MyNameAli = "Ali"

Snake Case

- Each word is separated by an underscore character:
- my_name_is = "Ali"

In [18]:
```
a=10    #int data type
b=20.30  # floating data type
print(a+b)
```
```
30.3
```

In [17]:
```
a = "Ali"   #str data type
print(a)
```
```
Ali
```

In [19]:
```
# Many Values to Multiple Variables

x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```
```
Orange
Banana
Cherry
```

In [22]:
```
# Many Values to Multiple Variables

x = y = z = "Mango"
print(x)
print(y)
print(z)
```
```
Mango
Mango
Mango
```

In [26]:
```
x, y, z = "Orange", "Banana", "Cherry"
print(x,y,z)
```
```
Orange Banana Cherry
```

Output Variables

- The Python print() function is often used to output variables

In [23]:
```
a = "Python is awesome"
print(a)
```
```
Python is awesome
```

In [24]:
```
# in the print() function, you output multiple variables, separated by a comma:

x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```
```
Python is awesome
```

In [25]:
```
# You can also use the + operator to output multiple variables:

x = "Python "
y = "is "
z = "awesome"
print(x + y + z)
```
```
Python is awesome
```

Data Types in python

Data types in Python define the type of data that can be stored in a variable. There are 7 main data types in Python:

- Integer: An integer is a whole number, positive or negative. Examples of integers are 1, 2, -1, 0
- Float: A float is a number with a decimal point. Examples of floats are 1.2, -3.4, 0.0.
- String: A string is a sequence of characters. Examples of strings are "Hello, world!", "This is a string", "".
- Boolean: A Boolean is a value that can be either True or False. Examples of Booleans are True, False.
- List: A list is an ordered sequence of objects. Examples of lists are [1, 2, 3], ["Hello", "world"], [].
- Dictionary: A dictionary is an unordered mapping from keys to values. Examples of dictionaries are {"name": "Bard", "age": 30}, {"key1": "value1", "key2": "value2"}.
- Set: A set is an unordered collection of unique objects. Examples of sets are {1, 2, 3}, {"Hello", "world"}, set()

Integer Datatyoe (int)

There are three numeric types in Python:

- int
- float
- complex

- Variables of numeric types are created when you assign a value to them:

In [27]:
```
a =23
b =3546773
c =3

print(a)
print(b)
print(c)
```
```
23
3546773
3
```

Checking datatype by using Typy() function that which data is store in the variable

In [28]:
```
# Integer data type

a= 1
b = 35656222554887711
c = -3255522

print(type(a))
print(type(b))
print(type(c))
```
```
<class 'int'>
<class 'int'>
<class 'int'>
```

In [30]:
```
# Float

a = 1.10
b = 1.0
c = -35.59

print(a,type(a))
print(b,type(b))
print(c,type(c))
```
```
1.1 <class 'float'>
1.0 <class 'float'>
-35.59 <class 'float'>
```

Float can also be scientific numbers with an "e" to indicate the power of 10.

In [34]:
```
a = 35e3
b = 12E4
c = -87.7e100

print(a,type(a))
print(b,type(b))
print(c,type(c))
```
```
35000.0 <class 'float'>
120000.0 <class 'float'>
-8.77e+101 <class 'float'>
```

Complex

- Complex numbers are written with a "j" as the imaginary part:

In [36]:
```
a = 3+5j
b = 5j
c = -5j

print(a,type(a))
print(b,type(b))
print(c,type(c))
```
```
(3+5j) <class 'complex'>
5j <class 'complex'>
(-0-5j) <class 'complex'>
```

Type Conversion

- You can convert from one type to another with the int(), float(), and complex() methods:

In [37]:
```
a = 1    # int
b = 2.8  # float
c = 1j   # complex

#convert from int to float:
x = float(a)

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

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

print(a)
print(b)
print(c)

print(type(x))
print(type(y))
print(type(z))
```
```
1
2.8
1j
<class 'float'>
<class 'int'>
<class 'complex'>
```

In [43]:
```
a= 20.0894    #float
b =int(a)
print((b),id(b))
print(type(b))
```
```
20 140706801554824
<class 'int'>
```

In [44]:
```
x= 1    #int
y = float(1)
print(y)
```
```
1.0
```

Strings

- Strings in python are surrounded by either single quotation marks, or double quotation marks.
- 'hello' is the same as "hello".

In [38]:
```
print("Hello")
print('Hello')
```
```
Hello
Hello
```

In [39]:
```
a="Hello Ali"
b="Hello Hassan"

print(a)
print(b)
```
```
Hello Ali
Hello Hassan
```

In [45]:
```
#You can use three double quotes

a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```
```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

In [46]:
```
# Or three single quotes:

a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```
```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

Boolean Data Type

- In programming you often need to know if an expression is True or False.

In [47]:
```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```
```
True
False
False
```

In [48]:
```
# Print a message based on whether the condition is True or False:

a = 200
b = 33

if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```
```
b is not greater than a
```

In [50]:
```
print(bool("Hello"))
print(bool(15))
print(bool())
```
```
True
True
False
```

List Lists

- Lists are used to store multiple items in a single variable.

In [ ]:
```
list1 = ["apple", "banana", "cherry", "apple", "cherry"]
print(list1)
```

Tuples Data Type

In [51]:
```
Tuple1 = ("apple", "banana", "cherry")
print(Tuple1)
```
```
('apple', 'banana', 'cherry')
```

Sets Data type

In [55]:
```
set1 = {"apple", "banana", "cherry"}
print(set1)
```
```
{'apple', 'cherry', 'banana'}
```

Dictionaries data type

- Dictionaries are used to store data values in key:value pairs.

In [56]:
```
#Create and print a dictionary:

country_capitals = {
  "United States": "Washington D.C.",
  "Italy": "Rome",
  "England": "London"
}

# printing the dictionary
print(country_capitals)
```
```
{'United States': 'Washington D.C.', 'Italy': 'Rome', 'England': 'London'}
```

Python Operators

- Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

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

Arithmetic Operator

In [57]:
```
a = 7
b = 2

# addition
print ('Sum: ', a + b)

# subtraction
print ('Subtraction: ', a - b)

# multiplication
print ('Multiplication: ', a * b)

# division
print ('Division: ', a / b)

# floor division
print ('Floor Division: ', a // b)

# modulo
print ('Modulo: ', a % b)

# a to the power b
print ('Power: ', a ** b)
```
```
Sum:  9
Subtraction:  5
Multiplication:  14
Division:  3.5
Floor Division:  3
Modulo:  1
Power:  49
```

Assignment Operators

1. Assign(=)
2. Add and Assign(+=)
3. Subtract and Assign(-=)
4. Multiply and Assign(*=)
5. Divide and Assign(/=)
6. Modules and Assign(%=)
7. Divide(floor) and Assign(//=)
8. Exponent and Assign(**=)
9. Bitwise and Assign(&=)
10. Bitwise or Assign(|=)
11. Bitwise XOR Assign(^=)
12. Bitwise Right shit and Assign(>>=)
13. Bitwise Right shit and Assign(<<=)

In [1]:
```
a = 21
b = 10
c = 0

c = a + b
print("Line 1 - Value of c is ", c)

c += a
print("Line 2 - Value of c is ", c)

c *= a
print ("Line 3 - Value of c is ", c )

c /= a
print( "Line 4 - Value of c is ", c)

c=2
c %= a
print( "Line 5 - Value of c is ", c)

c **= a
print( "Line 6 - Value of c is ", c)

c //= a
print("Line 7 - Value of c is ", c)
```
```
Line 1 - Value of c is  31
Line 2 - Value of c is  52
Line 3 - Value of c is  1092
Line 4 - Value of c is  52.0
Line 5 - Value of c is  2
Line 6 - Value of c is  2097152
Line 7 - Value of c is  99864
Line 7 - Value of c is  113
```

Comparison Operators

In [8]:
```
a = 10
b = 20

print(a < b)  # True
print(a <= b)  # True
print(a > b)  # False
print(a >= b)  # False
print(a == b)  # False
print(a != b)  # True
```
```
True
True
False
False
False
True
```

Logical Operators

In [9]:
```
x = 5

#In And both Conditions should be True


print(x > 3 and x < 10)
```
```
True
```

In [10]:
```
# In OR at least 1 Condition Should be Truth

x = 5

print(x > 3 or x < 4)
```
```
True
```

In [ ]:
```
# returns False because not is used to reverse the result

x = 5

print(not(x > 3 and x < 10))
```

Identity Operators

In [11]:
```
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)
```
```
True
False
True
```

Membership Operators

In [13]:
```
# returns True because a sequence with the value "banana" is in the list

x = ["apple", "banana"]

print("banana" in x)



x = ["apple", "banana"]

print("Orange" in x)
```
```
True
False
```

In [15]:
```
# returns True because a sequence with the value "pineapple" is not in the list

x = ["apple", "banana"]

print("pineapple" not in x)



x = ["apple", "banana"]

print("apple" not in x)
```
```
True
False
```

Bitwise Operators

- Bitwise operators are used to compare (binary) numbers:

In [17]:
```
# Bitwise XOR Assign(^=)
a=10
b=20


a^=b
print("Line 7 - Value of c is ", a)
```
```
Line 7 - Value of c is  30
```

In [18]:
```
#Bitwise Right shit and Assign(>>=)

a =55
b= 10

a &=b
print(a)
```
```
2
```

In [19]:
```
# Bitwise Right shit and Assign(<<=)

a =55
b= 10

a |=b
print(a)
```
```
63
```

In [16]:
```
# Zero fill left shift Shift left by pushing zeros in from the right and let the leftmost bits fall off

print(3 << 2)
```
```
12
```

In [21]:
```
# Signed right shift    Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

print(3 >>2)
```
```
0
```

In [ ]: