Informatics Practices

Data Handling

By

Rajesh Verma

Rajesh Verma MRA DAV Public School, Solan

Introduction

Most of the computer programming language support data type, variables, operator and expression like fundamentals. Python also support these.

Rajesh Verma MRA DAV Public School, Solan

Data Types

DATA TYPES means to identify the type of data and associated operations of handling it.

Data Types In Python

- Number
- String
- Boolean
- List
- Tuple
- Set
- Dictionary

Numbers In Python

It is used to store numeric values

- *Python has three numeric types:*
    1. *Integers*
    2. *Floating point numbers*
    3. *Complex numbers.*

1. Integers : Integers or int are positive or negative numbers with no decimal point. Integers in Python 3 are of unlimited size.

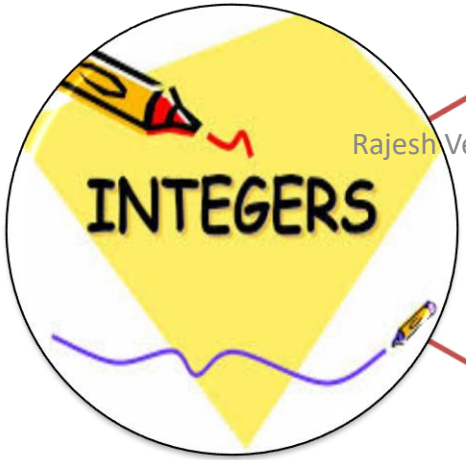
e.g.

```
a= 100
b= -100
c= 10*20
print(a)
print(b)
print(c)
```

Output :-

```
100
-100
200
```

TYPES OF INTEGERS



Integers (Signed).

Booleans.

(i) INTEGERS (SIGNED)

- *Integers can be of any length. Its only limited by the memory available.*
- *It's a signed representation, i.e., the integers can be positive or negative.*

Rajesh Verma MRA DAV Public School, Solan

(ii) BOOLEANS

- *Boolean data types are the truth values, i.e., True form or False form.*
- *Boolean data type is a one kind of integer type they can be represented in the integer form i.e., 0 and 1.*

(ii) BOOLEANS

practically one can execute as

>>>bool(0)

will give result as false

>>>bool(true)

will produce 1 result.

str ( ) Function

str ( ) function converts a value to string type.

>>>str(false)

will give string type result 'false'

>>>str(true)

will produce string type 'true' result.

2. Floating point numbers

It is a positive or negative real numbers with a decimal point.

- *Example :*

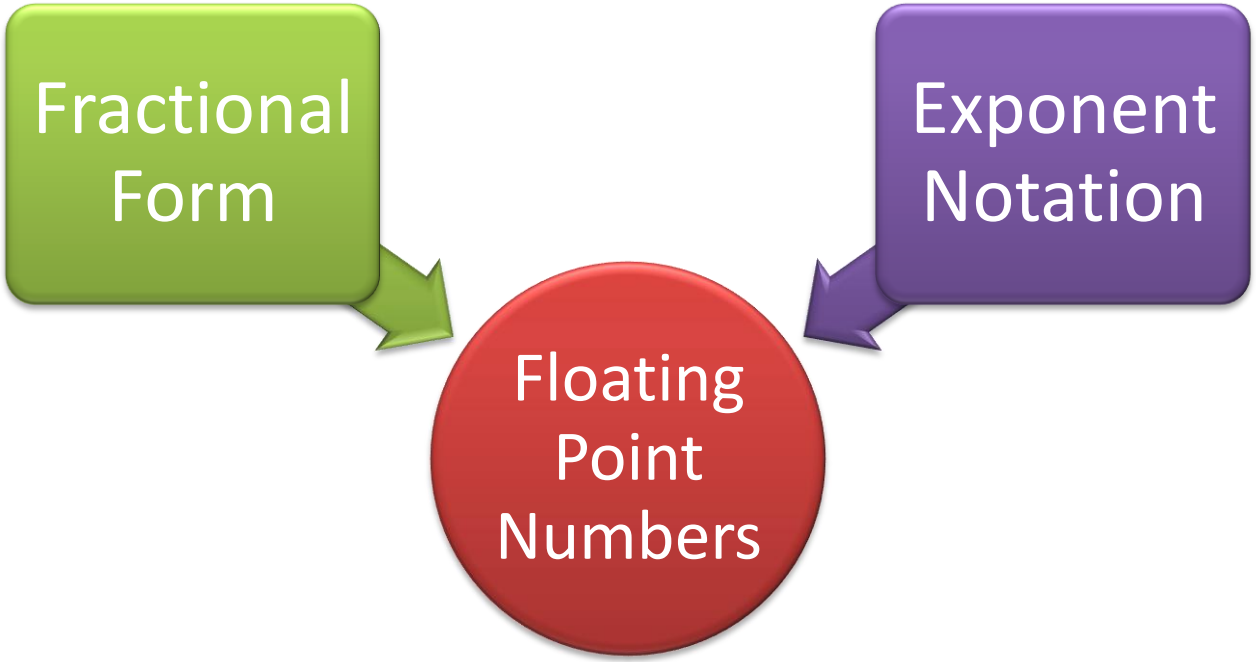
```
a = 101.2
b = -101.4
c = 111.23
d = 2.3*3
print(a)
print(b)
 print(c)
print(d)
```

Run Code

Output :-

```
101.2
-101.4
111.23
6.899999999999995
```

Floating point numbers can be written in two forms.



Note: ***Floating point numbers have precision of 15 Digits(double precision) in python.***

Advantages of Floating Point Numbers

1. They can represent range of values between the integers.

2. They can represent a greater extent/range of values.

Note: ***Floating point numbers have precision of 15 Digits(double precision) in python.***

Disadvantages of Floating Point Numbers

Floating-point operations are usually slower than integer operations.

Complex numbers : Complex numbers are combination of a real and imaginary part. Complex numbers are in the form of X+Yj, where X is a real part and Y is imaginary part.

Example:

```
a = complex(5)      # convert 5 to a real part val and zero imaginary part
print(a)
b=complex(101,23)   #convert 101 with real part and 23 as imaginary part
print(b)
```

Run Code

Output :-

```
(5+0j)
(101+23j)
```

2. String In Python

Strings:

A string can hold any type of known characters it means letters numbers and special characters of any known scripted language. In python 3.x, each character stored in a string is a Unicode character.

Unicode is a system designed to represent every character from every language.

Following are all legal strings in Python:

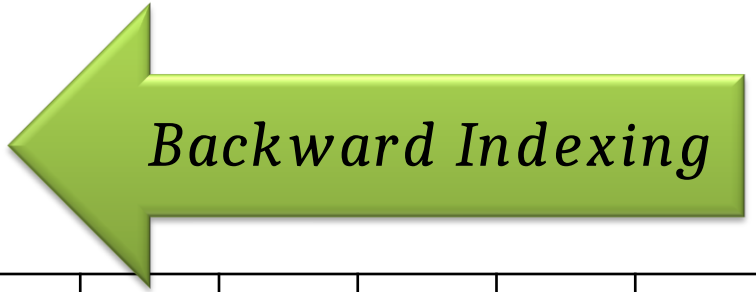
“abcd”, “1234”, “$%^&”, ‘????’

Rajesh Verma MRA DAV Public School,
Solan

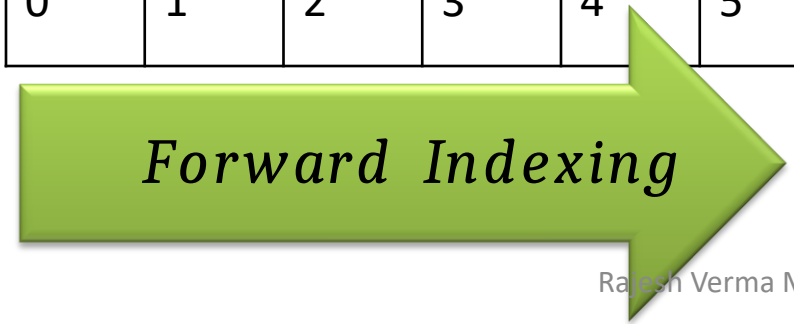
REPRESENTATION OF STRING

>>> s = "Hello Python"

This is how Python would index the string:



-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
H	E	L	L	O		P	Y	T	H	O	N
0	1	2	3	4	5	6	7	8	9	10	11



REPRESENTATION OF STRING

-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
H	E	L	L	O		P	Y	T	H	O	N
0	1	2	3	4	5	6	7	8	9	10	11

To access the first character on the string you just created, type and enter the variable name s and the index 0 within square brackets like this:

>>>s[0]

You'll get this output:

'H'

REPRESENTATION OF STRING

-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
H	E	L	L	O		P	Y	T	H	O	N
0	1	2	3	4	5	6	7	8	9	10	11

- *To access the last character, you can use this expression:*

>>>s[len(s)-1]

You'll get the output:

'n'

Len() function is used to find the length of the string.

REPRESENTATION OF STRING

-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
H	E	L	L	O		P	Y	T	H	O	N
0	1	2	3	4	5	6	7	8	9	10	11

The expression introduces you to the len function. There is actually an easier way to access the last item on the string:

>>>s[-1]

'n'

To access the penultimate character:

>>>s[-2]

'o'

Strings are Arrays

- *Strings can be indexed (subscripted), with the first character having index 0. There is no separate character type; a character is simply a string of size one.*
- *Indices may also be negative numbers, to start counting from the right.*

Note that since -0 is the same as 0, negative indices start from -1.

- *In addition to indexing, slicing is also supported. While indexing is used to obtain individual characters, slicing allows you to obtain substring.*
- *Python strings cannot be changed — they are immutable*

3. Boolean In Python

- *Booleans: - These represent the truth values False and True. The Boolean type is subtype of plain integer, and Boolean values False and True behave like 0 and 1, respectively, to get the Boolean equivalent of 0 and 1, you can type bool(0) or bool(1), python will return False or True respectively .*

List In Python

The lists and tuples are Python's compound data types.

Lists can be changed /modified (it means mutable)

but tuples cannot change or modified (it means immutable).

A list in Python represents a list of comma-separated values of any data type between square brackets. []

LISTS

List is a data type that can be used to store any type and number of variables and information. Lists are mutable in nature.

General format of list is :

my_list = [item_1, item_2, item_3]

Python also allows creation of an empty list:

my_list = []

For Example

colors = ["Orange" , "red" , "Green" , "White“]

Example of List

```
>>>College= ['IIT', 'NIT', 'College of Engg.']
>>>Print(College[0])                    Output  => IIT
>>>Print(College[2])                    Output  => College of Engg.
>>>College[2]='COE'
>>>Print(College[2])                    Output  => COE
>>>Print(College[1:3])                  Output  => NIT , COE
>>>College. append('Graphic Era')
>>>Print(College)                 Output  => IIT,NIT,COE , Graphic Era
>>>College. Insert(2, 'RIE')
>>>Print(College)                 Output  => IIT,NIT,RIE, COE , Graphic Era
```

Tuple In Python

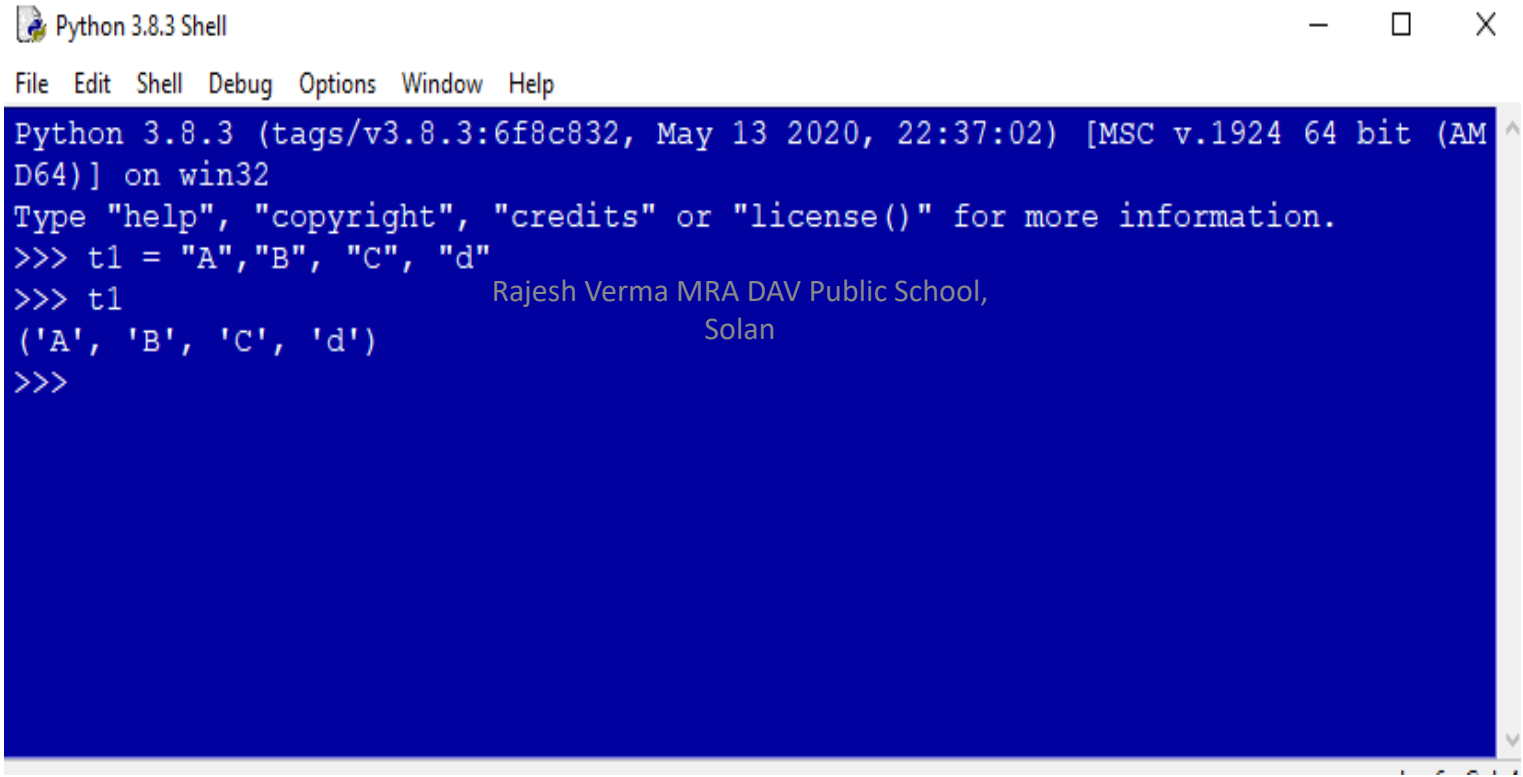
Tuples are those lists which cannot be changed (it means are not modifiable). Tuples are represented as a list of comma-separated values any data type within parentheses.

Rajesh Verma MRA DAV Public School, Solan

For Example

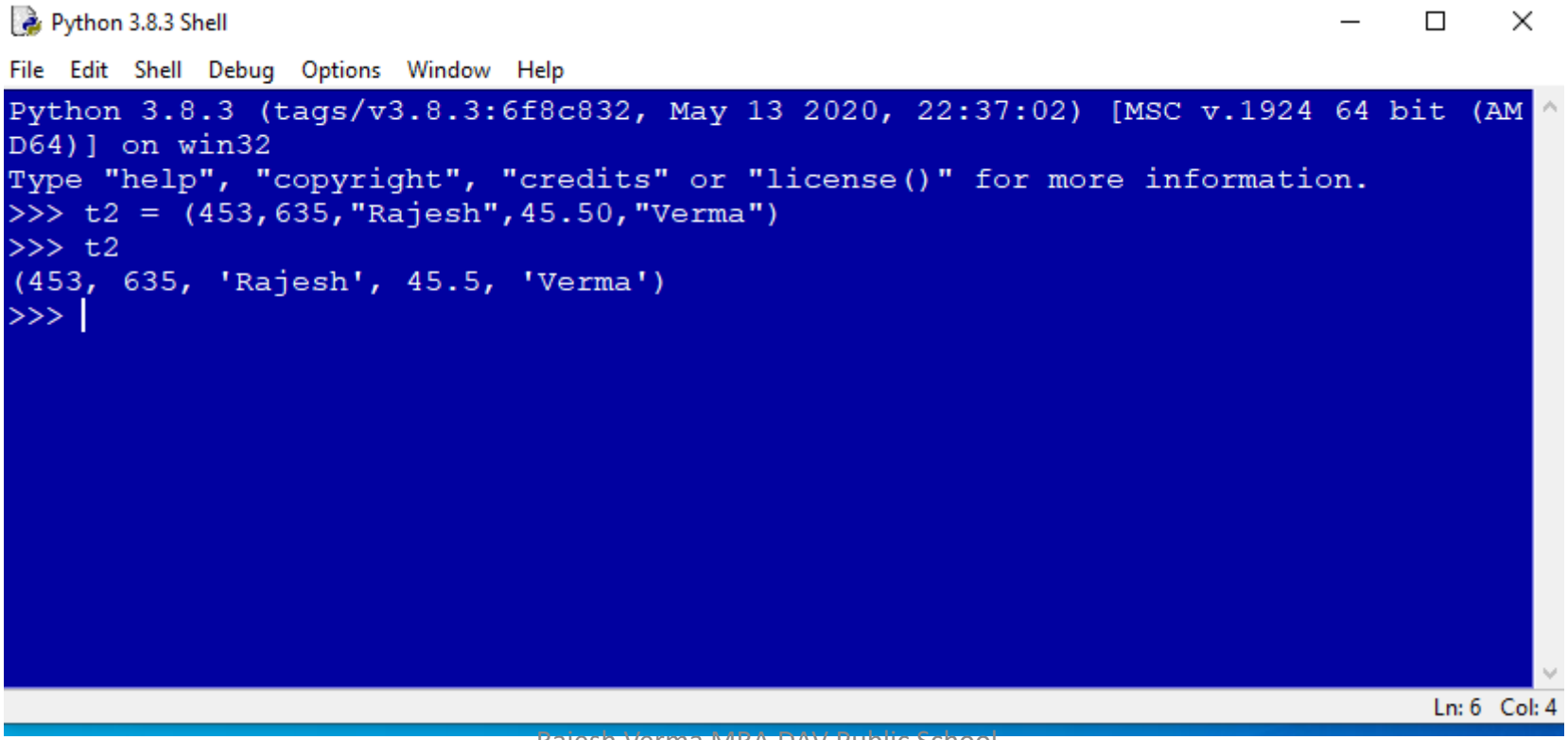
```
tup=(66,99)
Tup[0]=3 # error message will be displayed  print(tup[0])
print(tup[1])
```

Tuples Example
Defining Tuples Without ( )

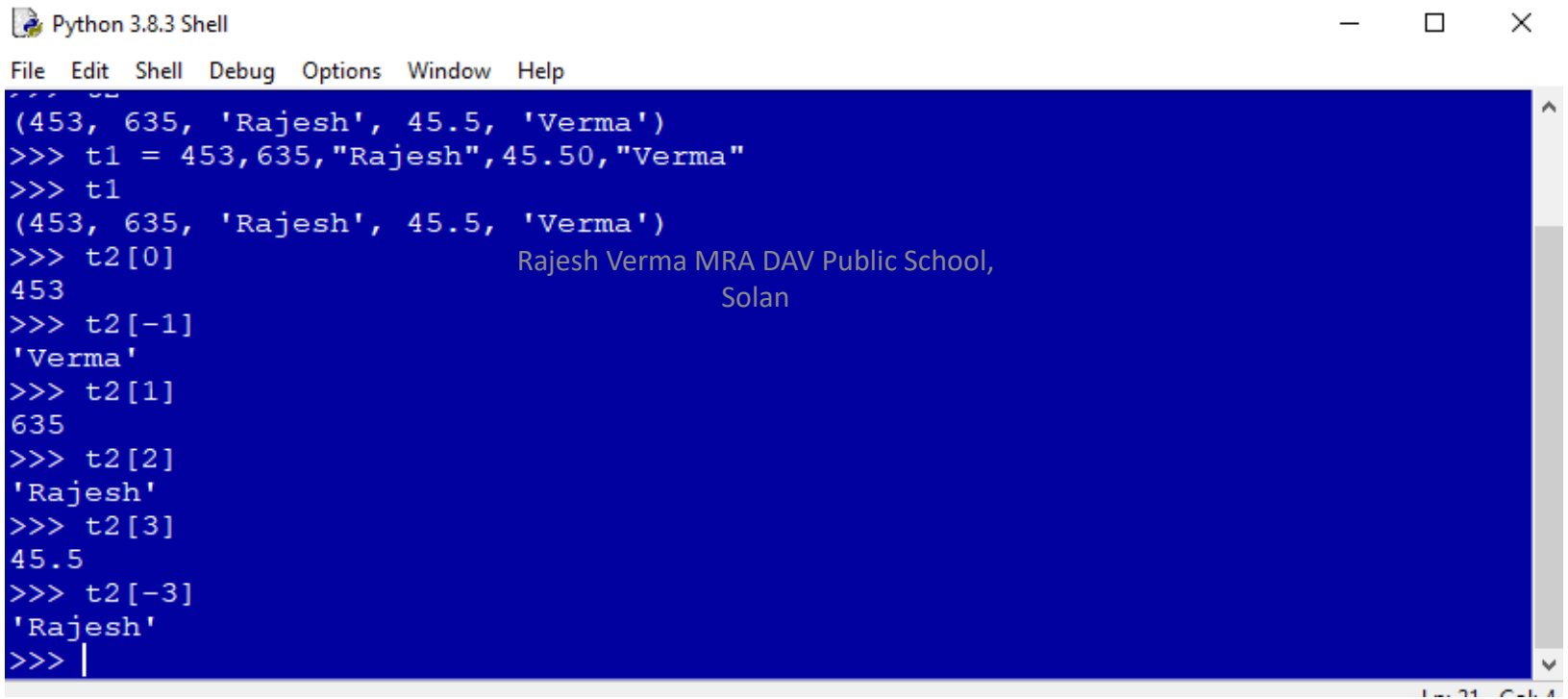


Rajesh Verma MRA DAV Public School,
Solan

Tuples Example
Defining Tuples With ( )



TUPLES EXAMPLE – Accessing individual elements using index number in [ ]



Rajesh Verma MRA DAV Public School, Solan

Dictionary In Python

The dictionary is an unordered set of comma separated key: value pairs, within { } ,with the requirement that within a dictionary no two keys can be the same.

Syntax :

Variable = {key : value, key : value}

Example of Dictionary

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> name = {'Navneet' : 42, 'Gaurav' : 35, 'Sunit' : 38, 'Nitin' : 50}
>>> name
{'Navneet': 42, 'Gaurav': 35, 'Sunit': 38, 'Nitin': 50}
>>> type(name)
<class 'dict'>
>>> print(name)
{'Navneet': 42, 'Gaurav': 35, 'Sunit': 38, 'Nitin': 50}
>>> print('Navneet')
Navneet
>>> print("Navneet")
Navneet
>>> print("Navneet:")
Navneet:
>>> print("Navneet" )
SyntaxError: invalid syntax
>>> print(names.values())
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    print(names.values())
NameError: name 'names' is not defined
>>> print(name.values())
dict_values([42, 35, 38, 50])
>>> print(name.keys())
dict_keys(['Navneet', 'Gaurav', 'Sunit', 'Nitin'])
>>> name['Navneet']
42
>>>
```

Rajesh Verma MRA DAV Public School, Solan

Mutable and Immutable Types

What is Mutable Object?

In object-oriented programming language , an mutable object is an object whose state/ values can be modified after its creation.

In short the an object / variable for which we can change the value is called mutable object or mutable variable.

- *For example : Lists, Dictionaries and sets are mutable in nature.*

Mutable and Immutable Types

What is Immutable Object?

In object-oriented programming language, an immutable object is an object whose state/ values can not be modified after it's creation.

In short an object or a variable , for which we can not change the value is immutable object or immutable variable.

- *For example : Integers, Floating numbers, Booleans, Strings, Tuples are immutable in nature.*

Operator

Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

Arithmetic operators:
used for mathematical operation

Operator	Meaning	Example
+	Add two operands or unary plus	x + y +2
-	Subtract right operand from the left or unary minus	x - y -2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

Arithmatic operator continue

```
Example:
x = 5
y = 4
print('x + y =', x+y)
print('x - y =', x-y)
print('x * y =', x*y)
print('x / y =', x/y)
print('x // y =', x//y)
print('x ** y =', x**y)
```



```
OUTPUT
('x + y =', 9)
('x - y =', 1)
('x * y =', 20)
('x / y =', 1.25)
('x // y =', 1)
('x ** y =', 625)
```

Relational operators :

used to compare values

Operator	Meaning	Example
>	Greater that - True if left operand is greater than the right	x > y
<	Less that - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

Example of Relational Operator

```
x = 101
y = 121
print('x > y is', x>y)
print('x < y is', x<y)
print('x == y is', x==y)
print('x != y is', x!=y)
print('x >= y is', x>=y)
print('x <= y is', x<=y)
```


Output

```
('x > y is', False)
('x < y is', True)
('x == y is', False)
('x != y is', True)
('x >= y is', False)
('x <= y is', True)
```

Logical operators

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

```
Example
x = True
y =  False
print('x and y is',x and y)
print('x or y is',x or y)
print('not x is',not x)
```

```
Output
('x and y is', False)
('x or y is', True)
('not x is', False)
```

Bitwise operators

Used to manipulate bit values

Operator	Meaning	Example
&	Bitwise AND	x& y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x>> 2
<<	Bitwise left shift	x<< 2

Membership Operators

Test for membership in a sequence

Operator	Description
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.

Example:

```
a = 5
b = 10
list = [1, 2, 3, 4, 5 ]
  if (a in list ):
            print ("Line 1 - a is available in the given list")
   else:
            print ("Line 1 - a is not available in the given list")
   if (b not in list ):
            print ("Line 2 - b is not available in the given list")
   else:
            print ("Line 2 - b is available in the given list")
```

output

```
Line 1 - a is available in the given list
Line 2 - b is not available in the given  list
```

Identity Operators

The identity operators are used to check if both the operands reference the same object memory i.e. the identity operators compare the memory location of two objects and returns true and false accordingly.

Operator	*Description*
is	*Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.*
is not	*Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.*

```
Example:   #Identity operators
a = 10
b = 10
print ('Line 1','a=',a,':',id(a), 'b=',b,':',id(b))
   if ( a is b ):
         print ("Line 2 - a and b have same identity")
   else:
         print ("Line 2 - a and b do not have same identity")
OUTPUT
('Line 1', 'a=', 10, ':', 20839436, 'b=', 10, ':', 20839436)
 Line 2 - a and b have same identity
```

Operators Precedence :

highest precedence to lowest precedence table

Operator	Description
()	Parentheses (grouping)
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and-@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Expression

It is a valid combination of operators, literals and variable.

1. *Arithmetic expression :- e.g. c = a+b*
2. *Relational expression :- e.g. x>y*
3. *Logical expression :- a or b*
4. *String expression :- c="comp"+"sc"*

Type conversion

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion.

Python has two types of type conversion.

1. Implicit Type Conversion

2. Explicit Type Conversion

Implicit Type Conversion:

In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.

e.g.

```
num_int = 12
num_flo = 10.23
```

Explicit Type Conversion:

In Explicit Type Conversion, users convert the data type of an object to required data type. We use the predefined functions like int(),float(),str() etc.

e.g.

```
numint = 12
numstr = "45"
print("Data type of num_int:",type(numint))
print("Data type of num_str before Type Casting:",type(numstr))
num_str = int(numstr)
print("Data type of num_str after Type Casting:",type(numstr))
num_sum = numint + numstr
print("Sum of num_int and num_str:",numsum)
print("Data type of the sum:",type(numsum))
```

math module

- *It is a standard module in Python. To use mathematical functions of this module, we have to import the module using import math.*

import math

Rajesh Verma MRA DAV Public School,
Solan

Function	Description	Example
ceil(n)	It returns the smallest integer greater than or equal to n.	math.ceil(4.2) returns 5
factorial(n)	It returns the factorial of value n	math.factorial(4) returns 24
floor(n)	It returns the largest integer less than or equal to n	math.floor(4.2) returns 4
fmod(x, y)	It returns the remainder when n is divided by y	math.fmod(10.5,2) returns 0.5
exp(n)	It returns e**n	math.exp(1) return 2.718281828459045
log2(n)	It returns the base-2 logarithm of n	math.log2(4) return 2.0

Function	*Description*	*Example*
pow(n, y)	It returns n raised to the power y	math.pow(2,3) returns 8.0
sqrt(n)	It returns the square root of n	math.sqrt(100) returns 10.0
cos(n)	It returns the cosine of n	math.cos(100) returns 0.8623188722876839
sin(n)	It returns the sine of n	math.sin(100) returns -0.5063656411097588
tan(n)	It returns the tangent of n	math.tan(100) returns -0.5872139151569291
pi	It is pi value (3.14159...)	It is (3.14159...)

Debugging

Debugging means to remove 'bugs' from a program.

Three types of error in a program

1. *Compile time error*
2. *Run time error*
3. *Logical error*

Compile time error

Error that occur during compile-time, are compile time error. When a program complies, its source code is checked for whether it follows the programming language's rules or not .

Compile time errors of two types:

1. *Syntax errors*
2. *Semantic errors*

Syntax error

Syntax error : Syntax occur when the rules of a programming language are misused i.e. When a grammatical rule of python is violated.

For Example:

*X<-y*z*

*If X = (x*y)*

Semantic error

- *Semantics refers to the set of rules which give the meaning of a statement.*

  For example

  $x*y=z$

Logical error

- *Don't encounter any error during compile-time and run time, your program does not provide the correct result. This is because of the programmer mistaken analysis of the problem they are trying to solve. Such error are known as logical error.*

Run time error

- *Run-time error: Errors that occurs during the execution of a program are run time error.*

Practice Question

- Write a program to repeat the string ‘‘GOOD MORNING” n times. Here n is an integer entered by the user.
- Write a program that asks the user to enter one's name and age. Print out a message addressed to the user that tells the user the year in which he/she will turn 100 years old.

Output: Mohan Das will become 100 years old in the year 2077.

Any Questions Please

Rajesh Verma, MRA DAV Public School,
Solan