

python :

Data Handling

Data Types

Data can be of many type e.g., character, integer, real, string etc. Anything enclosed in quotes represents string data in Python. Numbers without fractions represent integer data. Number with fractions represent real data and True and False represent Boolean data.

Built in core Data Types

. Numbers (Int, float, complex) . String . List . Tuple . Dictionary

Numbers

As it is clear by the name the number data types are used to store numeric values in Python. The numbers in Python have following **[cbsepythonedu.com]** core data types :

(i) Integers

* Integers (signed)

* Booleans

(ii) Floating-point Numbers

(iii) Complex Numbers

Integers

Integers are whole numbers such as 5, 39, 1917, 0 etc.

There are two types of integers in Python :

(i) Integers (signed). It is the normal integer representation of whole numbers. Integers Python 3.x can be of any length, it is only limited by the memory available.

(ii) Booleans. These represent the truth values False and True. The boolean type is subtype of plain integers, and Boolean values False and True behave like the values 0 and 1, respectively. To get the Boolean equivalent of 0 or 1, you can type bool (0) or bool (1), Python will return False or True respectively.

Floating Point Numbers

A number having fractional part is floating-point number.

Fractional numbers can be written in two forms :

(i) Fractional Form (Normal Decimal Notation) e.g., 3500.75, 0.00005, 147.9101 etc.

(ii) Exponent Notation e.g., 3.50075E03, 0.5E-04, 1.479101E02 etc.

Complex Numbers

Python is a versatile language that offers you a numeric type to represent complex Numbers.

Mathematically, a complex number is a number of the form $A + Bi$ where i is the imaginary number, equal to the square root **[cbsepythonedu.com]** of -1 i.e., $\sqrt{-1}$.

A complex number is made up of both real and imaginary components. In complex number $A + Bi$, A and B are real numbers and i is imaginary. If we have a complex number z , where $z = a + bi$ then a would be the real component and b would represent the imaginary

python:

component of z , e.g., real component of $z = 4 + 3i$ is 4 and the imaginary component would be 3.

Unlike Python's other numeric types, complex numbers are a composite quantity made of two parts : the real part and the imaginary part, both of which are represented internally as float values (floating point numbers)

You can retrieve the two component using attribute references. For a complex number z :

- * $z.real$ gives the real part.

- * $z.imag$ gives the imaginary part as a float, not as a complex value.

for example

```
>>> z = (1 + 2.56j) + (-4 - 3.56j)
```

```
>>> a
```

```
(-3 - 1j)
```

```
>>> z.real
```

```
-3.0
```

It will display real part [[cbsepythonedu.com](http://www.cbsepythonedu.com)] of complex number z

```
>>> z.imag
```

```
-1.0
```

It will display imaginary part of complex number z

Strings

A string data type lets you hold string data, i.e., any number of valid characters into a set of quotation marks. A string can hold any type letters, numbers, and special characters, of any known scripted [www.cbsepythonedu.com] language.

String as a sequence of characters

A Python string is a sequence of characters and each character can be individually accessed using its index.

Since length of string variable can be determined using function `len(<string>)`, we can say that :

- * first character of the string is at index 0 or $-\text{length}$

- * second character of the string is at index 1 or $-(\text{length}-1)$

- * second last character of the string is at index $(\text{length}-2)$ or -2

- * last character of the string is at index [[cbsepythonedu.com](http://www.cbsepythonedu.com)] $(\text{length}-1)$ or -1

Lists and Tuples

The lists and tuples are Python's compound data types. we have taken them together in one section because they are basically the same types with one difference. Lists can be changed/modified (i.e., mutable) but tuples cannot be changed or modified (i.e., immutable)

List

A list in Python represents a list of comma-separated values of any datatype between square brackets e.g., [www.cbsepythonedu.com] following are some lists

python :

```
[1,2,3,4,5]
```

```
['a','e','i','o','u']
```

```
['Neha',102,79.5]
```

Tuples

You can think of tuples (pronounced as tu-pp-le, rhyming with couple) as those lists which cannot be changed i.e., are not modifiable. Tuples are represented as list comma-separated values of any data type within parentheses, e.g., following are some tuples :

```
p = (1,2,3,4,5)
```

```
q = (2,4,6,8)
```

```
r = ('a','e','i','o','u')
```

```
h = (7,8,9,'A','B','C')
```

Dictionary

Dictionary data type is another feature in Python's hat. The dictionary is an unordered set of comma-separated Key : value pairs, within {}, with the requirement that within a dictionary, no two **[www.cbsepythonedu.com]** keys can be the same (i.e., there are unique keys within a dictionary). For instance, following are some dictionaries :

```
{'a':1,'e':2,'i':3,'o':4,'u':5}
```

Mutable and Immutable Types

The python data objects can be broadly categorized into two - **mutable and immutable** types, in simple words changeable or modifiable and non-modifiable types.

1. Immutable type

The immutables types are those that can never change their value in place. In Python, the following types are immutable : **integers, floating point numbers, strings, tuples**. You already know that in Python, variable-names are just the references to value-objects i.e., **[www.cbsepythonedu.com]** data values. The variable-names do not store values themselves i.e., they are not storage containers. Variable names are stored as references to a value-object. Each time you change the value, the variable's reference memory address changes.

Variables (of certain types) are NOT LIKE storage containers i.e., with fixed memory address where value changes every time. hence they are IMMUTABLE

* Integer * Floating point number * Booleans * Strings * Tuples

Mutability means that in the same memory address, new value can be stored as and when you want.

The types that do not support this property are **immutable type**

Mutable types

The mutable types are those whose values can be changed in place. Only three types are mutable in Python. these are : lists, dictionaries and sets.

To change a member of a list, you may write :

python:

Chk = [2,4,6]

Chk[1] = 40

Variable Internals

Python is an object oriented language. Python calls every entity that stores any values or any type of data as an object. [www.cbsepythonedu.com]

An object is an entity that has certain properties and that exhibit a certain type of behavior, e.g., integer values are objects -they hold whole numbers only and they have infinite precision (Proper ties); they support all arithmetic operations (behavior).

Every Python object has three key attributes associated to it :

(i) The type of an object [www.cbsepythonedu.com]

The type of an object determines the operations that can be performed on the object. Built-in function type () returns the type of an object.

(ii) The value of an object

It is the data-item contained in the object. For a literal, the value is the literal itself and for a variable the value is the data-item it (the variable) is currently referencing, Using print statement you can display value of an object.

(iii) The id of an object [www.cbsepythonedu.com]

The id of an object is generally the memory location of the object. Although id is implementation dependent but in most implementations it returns the memory location of the object. Built-in function id () returns the id of an object,

Operators

The operations being carried out on data, are represented by operators. The symbols that trigger the operation / action on data, are called operators.

Python's rich set of operators comprises of these types of operators :

- | | | |
|--------------------------|---------------------------|---------------------------|
| (i) Arithmetic operators | (ii) Relational operators | (iii) Identity operators |
| (iv) Logical operators | (v) Bitwise operators | (vi) Membership operators |

Arithmetic Operators

To do arithmetic, Python uses arithmetic operators. Python provides operators for basic calculations, as given below : [www.cbsepythonedu.com]

+	addition	//	floor division
-	subtraction	%	remainder
*	multiplication	**	exponentiation
/	division		

The complement Operator ~

The complement operator inverts the value of each bit of the operand : if the operand bit is 1 the

python :

result is 0 and if the operand bit is 0 the result is 1.

Operator Precedence

When an expression or statement involves multiple operators, Python resolves the order of execution through [www.cbsepythonedu.com] operator Precedence.

Operator	Description
()	Parentheses (grouping)
**	Exponentiation
~x	Bitwise
+ x, -x	Positive, negative (unary +,-)
*, /, //, %	Multiplication, division, floor division, remainder
+, -	Addition, subtraction
&	Bitwise and
^	Bitwise XOR
	Bitwise OR
<, <=, >, >=, <>, !=, ==,	Comparisons (Relational operators), identity operators
is, is not	
not x	Boolean NOT
and	Boolean AND
or	Boolean OR

Operator Associativity

Python allows multiple operators in a single expression. If the operators used in an expression have different precedence, there is not any problem as Python will evaluate the operator with higher precedence first. But what if the expression contains two operators that have the same precedence? In that case, associativity helps determine the order of operations. Associativity is the order in which an expression (having multiple operators of same precedence) is evaluated. Almost all the operators have left-to-right associativity except exponentiation (**), which has right-to-left associativity. That means, in case of multiple operators with same precedence, other than **, in same expression the operator on the left is [www.cbsepythonedu.com] evaluated first and then the operator on its right

Expressions

An expression in Python is any valid combination of operators, literals and variables. An expression is composed of one or more operations, with operators, literals, and variables as the [www.cbsepythonedu.com] constituents of expressions.

Python puts it in this way : a valid combination of atoms and operators forms a Python expression. In simplest words, an atom is something that has a value. So all of these are atoms in Python : identifiers, literals and values-in-enclosures such as quotes (''), parentheses, brackets, etc. i.e., strings, tuples, lists, dictionaries, sets etc.

* The type of operators and operands used in an expression determine the expression type .

* An expression can be compound expression to if it involves multiple types of operators, Implicit type conversion (coercion). An implicit type conversion is a conversion performed by

python :

the compiler without programmer's intervention. An implicit conversion is applied generally whenever differing data types are intermixed in an expression (mixed mode expression), so as not to lose information. [www.cbsepythonedu.com]

In a mixed arithmetic expression, Python converts all operands up to the type of the largest operand (type promotion). In simplest form, an expression is like op1 operator op2 (e.g., x/y or p**a). Here, if both arguments are standard numeric types, the following coercions are applied :

- * If either argument is a complex number, the other is converted to complex
- * Otherwise, if either argument is a floating point number, the other is converted to floating point ;
- * No conversion if both operands are integers.

Evaluating Relational Expressions (comparisons) [www.cbsepythonedu.com]

All comparison operation in Python have the same priority, which is lower than that of any arithmetic operations. All relational expressions (comparisons) yield Boolean values only i.e., True or False

Further, chained expressions like $a < b < c$ have the interpretation that is conventional in mathematics i.e., comparisons in Python are chained arbitrarily, e.g., $a < b < c$ is internally treated as $a < b$ and $b < c$. [www.cbsepythonedu.com]

For chained comparisons like $x < y \leq z$ (which is internally equivalent to $x < y$ and $y \leq z$), the common expression (the middle one, y here) is evaluated only once and the third expression (z here) is not evaluated at all when first comparison ($x < y$ here) is found to be False.

Type casting

You have learnt in earlier section that in expression with mixed types, Python internally changes the data type of some operands so that all operands have same data type. This type of conversion is automatic, i.e., implicit and hence known as implicit type conversion. [www.cbsepythonedu.com] Python, however, also supports explicit type conversion.

Explicit Type conversion

An explicit type conversion is user-defined conversion that forces an expression to be of specific type. The explicit type conversion is also known as Type casting.

Python Data Conversion Functions

Conversion Function: `int()` `float()` `complex()`

Type Casting Issues

Assigning a value to a type with a greater range (e.g., from short to long) poses no problem, however, assigning a value of larger data type to a smaller data type (e.g., from floating-point to integer) may result in losing some precision. [www.cbsepythonedu.com]

Then you can use math library's functions as `Math.<function-name>`. conventionally (not a syntactical requirement), you should give import statements at the top of the program code.

python :

Some mathematical function in math module: (www.cbsepythonedu.com)

S. No.	Function	Prototype (General form)	Description	Example
1.	ceil	math.ceil (num)	The ceil () function returns the smallest integer not less than num	math.ceil (1.03) gives 2.0 math.ceil(-1.03) gives -1.0
2.	sqrt	math.sqrt (num)	The sqrt () function returns the square root of num. If num <0, domain error occurs.	math.sqrt (81.0) gives 9.0
3.	exp	math.exp (arg)	The exp () function returns the natural logarithm e raised to the arg power	Math.exp(2.0) gives the value of e^2
4.	fabs	math.fabs (num)	The fabs () function returns the absolute value of num	math.fabs (1.0) gives 1.0 math.fabs(-1.0) gives 1.0
5.	floor	math.floor (num)	The floor () function returns the largest integer not greater than num	math.floor (1.03) gives 1.0 math.floor (-1.03) gives -2.0
6.	log	math.log (num,[base])	The log () function returns the natural logarithm for num. A domain error occurs if num is negative and a range error occurs if the argument num is zero	math.log(1.0) gives the natural logarithm for 1.0 math.log (1024, 2) will give logarithm of 1024 to the base 2.
7.	log10	math.log10 (num)	The log10 () function returns the base 10 logarithm for num. A domain error occurs if num is negative and a range error occurs if the argument is zero.	math.log10(1.0) gives base 10 logarithm for 1.0
8.	pow	math.pow (base, exp)	The pow () function returns base raised to exp power i.e., base exp. A domain error occurs if base =0 and exp <=0 ; also if base <0 and exp is not integer.	math.pow (3.0,0) gives value of 3^0 math.pw(4.0, 2.0) gives value of 4^2
9.	sin	math.sin (arg)	The sin () function returns the sine of arg. The value of arg must be in radians.	math.sin (val) (val is a number)
10.	cos	math.cos (arg)	The cos () function returns the cosine of arg. The value of arg must be in radians.	math.cos (val) (val is a number)
11.	tan	math.tan (arg)	The tan () function returns the tangent of arg. The value of arg must be in radians.	math.tan (val) (val is a number)
12.	degrees	math.degrees (x)	The degree () converts angle x from radians to degrees	math.degrees (3.14) would give 179.91
13.	radians	math.radians (x)	The radians () converts angle from degrees to radians	math.radians (179.91) would give 3.14