Values & Data types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types −

- Numbers
- String
- List
- Tuple
- Dictionary

Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them. For example −

var1 = 1

var2 = 10

Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

Python Dictionary

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

In Python there are some distinct numeric types. These are integers type numbers, floating point numbers, complex numbers. In the complex numbers, there are two parts the real and imag. The complex numbers are denoted like (a + bj).

There is another function called fraction. The fraction holds rational numbers and decimal holds floating point numbers.

Some functions like int(), float(), complex(), these are used to convert numbers to integers, floats or complex numbers.

Some of the operations and functions of these numeric types are as follows −

Sr.No.	Operation/Functions & Description
1	x + y Sum of x and y
2	x - y Subtract y from x
3	x * y Multiply x and y
4	x / y Divide x by y

5

x // y

Quotient of x after dividing by y

6

x % y

Remainder of x after dividing by y

7

x ** y

X to the power y

8

-x

The negated value of x

9

+x

Unchanged x value

10

abs(x)

Absolute (Magnitude) value of x

11

int(x)

Convert x to integer

12

float(x)

Convert x to floating point data

13

complex(re, im)

Convert from real and imaginary data to complex number

14

x.conjugate()

Find conjugate of a complex x

15

divmod(x,y)

Find Quotient and Remainder as a tuple

16

pow(x,y)

Find x to the power y

Example Code

Live Demo

```
from fractions import Fraction
x = 100
y = 3.256

print(x + y)
print(x - y)
print(x * y)
print(x / y)
print(x // y)
print(x % 7)
print(12 ** 3)

myComplex1 = complex('7+5j')
myComplex2 = complex('26+8j')
res = myComplex1 + myComplex2
print(res)
print(res.conjugate())
print(divmod(x, 3))
print(Fraction(0.125))
```

Output

```
103.256
96.744
325.59999999999997
30.712530712530715
30.0
2
1728
(33+13j)
(33-13j)
(33, 1)
1/8
```

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable. For example –

Live Demo

```
#!/usr/bin/python

counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string

print counter
print miles
print name
```

Here, 100, 1000.0 and "John" are the values assigned to counter, miles, and name variables, respectively. This produces the following result –

```
100
1000.0
```

John

Multiple Assignment

Python allows you to assign a single value to several variables simultaneously. For example –

```
a = b = c = 1
```

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example –

```
a,b,c = 1,2,"john"
```

Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

Standard Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types –

Numbers

String

List

Tuple

Dictionary

Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them. For example –

```
var1 = 1
```

var2 = 10

You can also delete the reference to a number object by using the del statement. The syntax of the del statement is −

del var1[,var2[,var3[....,varN]]]]

You can delete a single object or multiple objects by using the del statement. For example −

del var

del var_a, var_b

Python supports four different numerical types −

int (signed integers)

long (long integers, they can also be represented in octal and hexadecimal)

float (floating point real values)

complex (complex numbers)

Examples

Here are some examples of numbers −

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

Python allows you to use a lowercase l with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.

A complex number consists of an ordered pair of real floating-point numbers denoted by

x + yj, where x and y are the real numbers and j is the imaginary unit.

Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator. For example –

Live Demo

```
#!/usr/bin/python

str = 'Hello World!'

print str          # Prints complete string
print str[0]       # Prints first character of the string
print str[2:5]     # Prints characters starting from 3rd to 5th
print str[2:]      # Prints string starting from 3rd character
print str * 2      # Prints string two times
print str + "TEST" # Prints concatenated string
```

This will produce the following result –

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

Python Lists

Lists are the most versatile of Python's compound data types. A list contains items

separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator. For example –

```
#!/usr/bin/python

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print list          # Prints complete list
print list[0]       # Prints first element of the list
print list[1:3]     # Prints elements starting from 2nd till 3rd
print list[2:]      # Prints elements starting from 3rd element
print tinylist * 2  # Prints list two times
print list + tinylist # Prints concatenated lists
```

This produce the following result –

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as read-only lists. For example −

Live Demo

```
#!/usr/bin/python

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')

print tuple              # Prints the complete tuple
print tuple[0]           # Prints first element of the tuple
print tuple[1:3]         # Prints elements of the tuple starting from 2nd till 3rd
print tuple[2:]          # Prints elements of the tuple starting from 3rd element
print tinytuple * 2      # Prints the contents of the tuple twice
print tuple + tinytuple  # Prints concatenated tuples
```

This produce the following result −

```
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
```

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists −

```
#!/usr/bin/python

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
list = [ 'abcd', 786 , 2.23, 'john', 70.2  ]
```

tuple[2] = 1000 # Invalid syntax with tuple

list[2] = 1000 # Valid syntax with list

Python Dictionary

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]). For example –

Live Demo

```
#!/usr/bin/python

dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}


print dict['one']       # Prints value for 'one' key
print dict[2]           # Prints value for 2 key
print tinydict          # Prints complete dictionary
print tinydict.keys()   # Prints all the keys
print tinydict.values() # Prints all the values
```

This produce the following result –

```
This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
['dept', 'code', 'name']
```

['sales', 6734, 'john']

Dictionaries have no concept of order among elements. It is incorrect to say that the elements are "out of order"; they are simply unordered.

Data Type Conversion

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

Sr.No.	Function & Description
1	int(x [,base]) Converts x to an integer. base specifies the base if x is a string.
2	long(x [,base]) Converts x to a long integer. base specifies the base if x is a string.
3	float(x) Converts x to a floating-point number.
4	complex(real [,imag]) Creates a complex number.

5

str(x)

Converts object x to a string representation.

6

repr(x)

Converts object x to an expression string.

7

eval(str)

Evaluates a string and returns an object.

8

tuple(s)

Converts s to a tuple.

9

list(s)

Converts s to a list.

10

set(s)

Converts s to a set.

11

dict(d)

Creates a dictionary. d must be a sequence of (key,value) tuples.

12

frozenset(s)

Converts s to a frozen set.

13

chr(x)

Converts an integer to a character.

14

unichr(x)

Converts an integer to a Unicode character.

15

ord(x)

Converts a single character to its integer value.

16

hex(x)

Converts an integer to a hexadecimal string.

17

oct(x)

Converts an integer to an octal string.