Python - Type Casting

Python Type Casting

From a programming point of view, a type casting refers to converting an object of one type into another. Here, we shall learn about type casting in Python Programming.

> *Python Type Casting is a process in which we convert a literal of one data type to another data type. Python supports two types of casting – **implicit** and **explicit**.*

In Python there are different data types, such as numbers, sequences, mappings etc. There may be a situation where, you have the available data of one type but you want to use it in another form. For example, the user has input a string but you want to use it as a number. Python's type casting mechanism let you do that.

Python Implicit Casting

When any language compiler/interpreter automatically converts object of one type into other, it is called automatic or **implicit casting**. Python is a strongly typed language. It doesn't allow automatic type conversion between unrelated data types. For example, a string cannot be converted to any number type. However, an integer can be cast into a float. Other languages such as JavaScript is a weakly typed language, where an integer is coerced into a string for concatenation.

Note that memory requirement of each data type is different. For example, an **integer** object in Python occupies 4 bytes of memory, while a **float** object needs 8 bytes because of its fractional part. Hence, Python interpreter doesn't automatically convert a **float** to **int**, because it will result in loss of data. On the other hand, **int** can be easily converted into **float** by setting its fractional part to 0.

Implicit **int** to **float** casting takes place when any arithmetic operation on **int** and **float** operands is done.

Consider we have an ,**int** and one **float** variable

```
<<< a=10   # int object
<<< b=10.5 # float object
```

To perform their addition, 10 – the integer object is upgraded to 10.0. It is a float, but equivalent to its earlier numeric value. Now we can perform addition of two floats.

```
<<< c=a+b
<<< print (c)
20.5
```

In implicit type casting, a Python object with lesser byte size is upgraded to match the bigger byte size of other object in the operation. For example, a Boolean object is first upgraded to int and then to float, before the addition with a floating point object. In the following example, we try to add a Boolean object in a float, pleae note that True is equal to 1, and False is equal to 0.

</> Open Compiler

```
a=True;
b=10.5;
c=a+b;

print (c);
```

This will produce the following result:

```
11.5
```

Learn **Python** in-depth with real-world projects through our **Python certification course**. Enroll and become a certified expert to boost your career.

Python Explicit Casting

Although automatic or implicit casting is limited to **int** to **float** conversion, you can use Python's built-in functions int(), float() and str() to perform the explicit conversions such as string to integer.

Python int() Function

Python's built-in **int()** function converts an integer literal to an integer object, a float to integer, and a string to integer if the string itself has a valid integer literal representation.

Using **int()** with an int object as argument is equivalent to declaring an **int** object directly.

```
<<< a = int(10)
<<< a
```

```
10
```

is same as –

```
<<< a = 10
<<< a
10
<<< type(a)
<class 'int>
```

If the argument to **int()** function is a float object or floating point expression, it returns an int object. For example –

```
<<< a = int(10.5) #converts a float object to int
<<< a
10
<<< a = int(2*3.14) #expression results float, is converted to int
<<< a
6
<<< type(a)
<class 'int'>
```

The **int()** function also returns integer 1 if a Boolean object is given as argument.

```
<<< a=int(True)
<<< a
1
<<< type(a)
<class 'int'>
```

String to Integer

The **int()** function returns an integer from a string object, only if it contains a valid integer representation.

```
<<< a = int("100")
<<< a
100
<<< type(a)
<class 'int'>
```

```
<<< a = ("10"+"01")
<<< a = int("10"+"01")
<<< a
1001
<<< type(a)
<class 'int'>
```

However, if the string contains a non-integer representation, Python raises ValueError.

```
<<< a = int("10.5")
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '10.5'
<<< a = int("Hello World")
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'Hello World'
```

The **int()** function also returns integer from binary, octal and hexa-decimal string. For this, the function needs a base parameter which must be 2, 8 or 16 respectively. The string should have a valid binary/octal/Hexa-decimal representation.

Binary String to Integer

The string should be made up of 1 and 0 only, and the base should be 2.

```
<<< a = int("110011", 2)
<<< a
51
```

The Decimal equivalent of binary number 110011 is 51.

Octal String to Integer

The string should only contain 0 to 7 digits, and the base should be 8.

```
<<< a = int("20", 8)
<<< a
16
```

The Decimal equivalent of octal 20 is 16.

Hexa-Decimal String to Integer

The string should contain only the Hexadecimal symbols i.e., 0-9 and A, B, C, D, E or F. Base should be 16.

```
<<< a = int("2A9", 16)
<<< a
681
```

Decimal equivalent of Hexadecimal 2A9 is 681. You can easily verify these conversions with calculator app in Windows, Ubuntu or Smartphones.

Following is an example to convert number, float and string into integer data type:

</> Open Compiler

```
a = int(1)     # a will be 1
b = int(2.2)   # b will be 2
c = int("3")   # c will be 3

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

This produce the following result –

```
1
2
3
```

Python float() Function

The **float()** is a built-in function in Python. It returns a float object if the argument is a float literal, integer or a string with valid floating point representation.

Using float() with an float object as argument is equivalent to declaring a float object directly

```
<<< a = float(9.99)
<<< a
9.99
```

```
<<< type(a)
<class 'float'>
```

is same as –

```
<<< a = 9.99
<<< a
9.99
<<< type(a)
<class 'float'>
```

If the argument to **float()** function is an integer, the returned value is a floating point with fractional part set to 0.

```
<<< a = float(100)
<<< a
100.0
<<< type(a)
<class 'float'>
```

The **float()** function returns float object from a string, if the string contains a valid floating point number, otherwise ValueError is raised.

```
<<< a = float("9.99")
<<< a
9.99
<<< type(a)
<class 'float'>
<<< a = float("1,234.50")
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: '1,234.50'
```

The reason of ValueError here is the presence of comma in the string.

For the purpose of string to float conversion, the sceientific notation of floating point is also considered valid.

```
<<< a = float("1.00E4")
<<< a
10000.0
```

```
<<< type(a)
<class 'float'>
<<< a = float("1.00E-4")
<<< a
0.0001
<<< type(a)
<class 'float'>
```

Following is an example to convert number, float and string into float data type:

```
</>                                                    Open Compiler

a = float(1)     # a will be 1.0
b = float(2.2)   # b will be 2.2
c = float("3.3") # c will be 3.3

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

This produce the following result –

```
1.0
2.2
3.3
```

Python str() Function

We saw how a Python obtains integer or float number from corresponding string representation. The **str()** function works the opposite. It surrounds an integer or a float object with quotes (') to return a str object. The **str()** function returns the string representation of any Python object. In this section, we shall see different examples of **str()** function in Python.

The str() function has three parameters. First required parameter (or argument) is the object whose string representation we want. Other two operators, encoding and errors, are optional.

We shall execute str() function in Python console to easily verify that the returned object is a string, with the enclosing quotation marks (').

Integer to string

You can convert any integer number into a string as follows:

```
<<< a = str(10)
<<< a
'10'
<<< type(a)
<class 'str'>
```

Float to String

str() function converts floating point objects with both the notations of floating point, standard notation with a decimal point separating integer and fractional part, and the scientific notation to string object.

```
<<< a=str(11.10)
<<< a
'11.1'
<<< type(a)
<class 'str'>
<<< a = str(2/5)
<<< a
'0.4'
<<< type(a)
<class 'str'>
```

In the second case, a division expression is given as argument to str() function. Note that the expression is evaluated first and then result is converted to string.

Floating points in scientific notations using E or e and with positive or negative power are converted to string with str() function.

```
<<< a=str(10E4)
<<< a
'100000.0'
<<< type(a)
<class 'str'>
<<< a=str(1.23e-4)
<<< a
'0.000123'
```

```
<<< type(a)
<class 'str'>
```

When Boolean constant is entered as argument, it is surrounded by (') so that True becomes 'True'. List and Tuple objects can also be given argument to str() function. The resultant string is the list/tuple surrounded by (').

```
<<< a=str('True')
<<< a
'True'
<<< a=str([1,2,3])
<<< a
'[1, 2, 3]'
<<< a=str((1,2,3))
<<< a
'(1, 2, 3)'
<<< a=str({1:100, 2:200, 3:300})
<<< a
'{1: 100, 2: 200, 3: 300}'
```

Following is an example to convert number, float and string into string data type:

</> Open Compiler

```
a = str(1)     # a will be "1"
b = str(2.2)   # b will be "2.2"
c = str("3.3") # c will be "3.3"

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

This produce the following result –

```
1
2.2
3.3
```

Conversion of Sequence Types

List, Tuple and String are Python's sequence types. They are ordered or indexed collection of items.

A string and tuple can be converted into a list object by using the **list()** function. Similarly, the **tuple()** function converts a string or list to a tuple.

We shall take an object each of these three sequence types and study their inter-conversion.

```
<<< a=[1,2,3,4,5]   # List Object
<<< b=(1,2,3,4,5)   # Tupple Object
<<< c="Hello"       # String Object

### list() separates each character in the string and builds the list
<<< obj=list(c)
<<< obj
['H', 'e', 'l', 'l', 'o']

### The parentheses of tuple are replaced by square brackets
<<< obj=list(b)
<<< obj
[1, 2, 3, 4, 5]

### tuple() separates each character from string and builds a tuple of
characters
<<< obj=tuple(c)
<<< obj
('H', 'e', 'l', 'l', 'o')

### square brackets of list are replaced by parentheses.
<<< obj=tuple(a)
<<< obj
(1, 2, 3, 4, 5)

### str() function puts the list and tuple inside the quote symbols.
<<< obj=str(a)
<<< obj
'[1, 2, 3, 4, 5]'

<<< obj=str(b)
<<< obj
'(1, 2, 3, 4, 5)'
```

Thus Python's explicit type casting feature allows conversion of one data type to other with the help of its built-in functions.

Data Type Conversion Functions

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	**Python int() function** Converts x to an integer. base specifies the base if x is a string.
2	**Python long() function** Converts x to a long integer. base specifies the base if x is a string.
3	**Python float() function** Converts x to a floating-point number.
4	**Python complex() function** Creates a complex number.
5	**Python str() function** Converts object x to a string representation.
6	**Python repr() function** Converts object x to an expression string.
7	**Python eval() function** Evaluates a string and returns an object.
8	**Python tuple() function** Converts s to a tuple.
9	**Python list() function** Converts s to a list.
10	**Python set() function** Converts s to a set.
11	**Python dict() function** Creates a dictionary. d must be a sequence of (key,value) tuples.
12	**Python frozenset() function** Converts s to a frozen set.
13	**Python chr() function** Converts an integer to a character.

14	**Python unichr() function** Converts an integer to a Unicode character.
15	**Python ord() function** Converts a single character to its integer value.
16	**Python hex() function** Converts an integer to a hexadecimal string.
17	**Python oct() function** Converts an integer to an octal string.