10.TypeCasting_Operators

April 30, 2020

- Numbers

[1]:
```
x = 5 #
y = 34.5
z = 12+5j
x + y * z
```

[1]:
```
(419+172.5j)
```

- strings

[2]:
```
s = "Hello World"
```

[3]:
```
s[0]
```

[3]:
```
'H'
```

[4]:
```
s[:5]
```

[4]:
```
'Hello'
```

[5]:
```
s[::-1]
```

[5]:
```
'dlroW olleH'
```

[6]:
```
s1 = s.lower()
print(s1)
```

```
hello world
```

[7]:
```
s1 = s.upper()
print(s1)
```

```
HELLO WORLD
```

[8]:
```
s = s.split()
print(s)
```

```
['Hello', 'World']
```

[11]:
```
name = 'sachin'
s = " %s name inside string."%(name)
print(s)
s = " {} name inside string.".format(name)
print(s)
s = f" {name} name inside string."
print(s)
```

```
 sachin name inside string.
 sachin name inside string.
 sachin name inside string.
```

[12]:
```
l = [ 'java', 'c']
```

[13]:
```
l.insert(0, 'python')
```

[14]:
```
l.extend([ 'c++', 'ruby', 'perl'])
```

[15]:
```
print(l)
```

```
['python', 'java', 'c', 'c++', 'ruby', 'perl']
```

[16]:
```
l.append('lang')
```

[17]:
```
l
```

[17]:
```
['python', 'java', 'c', 'c++', 'ruby', 'perl', 'lang']
```

[18]:
```
l.sort()
```

[19]:
```
l
```

[19]:
```
['c', 'c++', 'java', 'lang', 'perl', 'python', 'ruby']
```

[20]:
```
l.reverse()
```

[21]:
```
l
```

[21]:
```
['ruby', 'python', 'perl', 'lang', 'java', 'c++', 'c']
```

[22]:
```
c = l.copy()
```

[23]:
```
c
```

[23]:
```
['ruby', 'python', 'perl', 'lang', 'java', 'c++', 'c']
```

[24]:
```
t = ( 'java' , 'c', ['java', 'c'], {'name':'sachin', 'age': 24})
```

[25]:
```
t
```

[25]:
```
('java', 'c', ['java', 'c'], {'name': 'sachin', 'age': 24})
```

[26]:
```
t[:2]
```

[26]:
```
('java', 'c')
```

[27]:
```
t[::-1]
```

[27]:
```
({'name': 'sachin', 'age': 24}, ['java', 'c'], 'c', 'java')
```

[28]:
```
print(t)
```

```
('java', 'c', ['java', 'c'], {'name': 'sachin', 'age': 24})
```

[29]:
```
d = { 'name':'sachin', 'lang': [ 'java', 'c', 'c++']}
```

[30]:
```
d['name']
```

[30]:
```
'sachin'
```

[31]:
```
d['color'] = 'fair'
```

[32]:
```
print(d)
```

```
{'name': 'sachin', 'lang': ['java', 'c', 'c++'], 'color': 'fair'}
```

[33]:
```
from pprint import pprint
```

[34]:
```
pprint(d)
```

```
{'color': 'fair', 'lang': ['java', 'c', 'c++'], 'name': 'sachin'}
```

[35]:
```
d.pop('color')
```

[35]:
```
'fair'
```

[36]:
```
d
```

[36]:
```
{'name': 'sachin', 'lang': ['java', 'c', 'c++']}
```

[37]:
```
d.update([ ('age', 24), ('some', 'good')])
```

[38]:
```
d
```

[38]:
```
{'name': 'sachin', 'lang': ['java', 'c', 'c++'], 'age': 24, 'some': 'good'}
```

[39]:
```
print(d)
```

```
{'name': 'sachin', 'lang': ['java', 'c', 'c++'], 'age': 24, 'some': 'good'}
```

[41]:
```
rajat = { 'java', 'python', 'bash', 'html'}
sachin = { 'java', 'c', 'c++', 'ruby', 'java', 'java'}
```

[42]:
```
print(type(sachin))
```

```
<class 'set'>
```

[43]:
```
common = rajat.intersection(sachin)
print(common)
```

```
{'java'}
```

[44]:
```
diff = rajat.difference(sachin)
```

[45]:
```
diff
```

[45]: {'bash', 'html', 'python'}

[46]:
```
diff = sachin.difference(rajat)
```

[47]:
```
diff
```

[47]: {'c', 'c++', 'ruby'}

Boolean

True --> yes

False --> no

what things are false in python -->

0, False, None, any empty object ('', [], {}, ...)

[48]:
```
True
```

[48]: True

[49]:
```
False
```

[49]: False

[50]:
```
4 > 5
```

[50]: False

[51]:
```
5 > 4
```

[51]: True

None

represents a value which is nothing, also known as null value

[52]:
```
None
```

[53]:
```
x = print()
```

[54]:
```
print(x)
```

None

[55]:
```
l = [ 1, 2, 3, 4, 5]

l = l.sort()

print(l)
```

None

[56]:
```
l
```

[57]:
```
l = [ 1, 2, 3, 4, 5]

item = l.pop()
```

[58]:
```
print(item)
```

5

[59]:
```
l
```

[59]: [1, 2, 3, 4]

marrige function --> dulhan

birthday function --> party

[60]:
```
x = input() #'value'
print(x)
```

```
value
value
```

[61]:
```
print("Hello World")
```

```
Hello World
```

0.0.1 Type Casting or Type Conversion

changing type of a data to another type

int --> used to convert data into decimal

float --> used to convert data into float values

complex --> used to convert data into complex values

bin --> so on

oct

hex

str

list

tuple

dict

set

frozenset

bool

[62]:
```
x = input("x: ")
y = input("y: ")

r = x + y
print(r)
```

```
x: 5
y: 10
510
```

input() --> value --> str type

[64]:
```
x = input("x: ")
print(type(x), repr(x))
```

```
x: 5
<class 'str'> '5'
```

[65]:
```
"Hello World " + "It's easy"
```

[65]: "Hello World It's easy"

[66]:
```
"5" + "10"
```

[66]: '510'

[67]:
```
5 + 10
```

[67]: 15

[68]:
```
x = input("input : ")
```

```
input : 10
```

[69]:
```
type(x)
```

[69]: str

[70]:
```
x
```

[70]: '10'

[71]:
```
x = int(x)

print(type(x))
```

```
<class 'int'>
```

[72]:
```
s = "1234"

i = int(s)

print(type(s), repr(s))
print(type(i), repr(i))
```

```
<class 'str'> '1234'
<class 'int'> 1234
```

[74]:
```
s = "abcdef"
```

```
# is it possible to convert this string into integer value ?
```

binary numbers

octal numbers

hexa-decimal

[75]:
```
s
```

[75]:
```
'abcdef'
```

[76]:
```
int(s)
```

```
␣
↪---------------------------------------------------------------------------

       ValueError                                Traceback (most recent call␣
↪last)

       <ipython-input-76-2c4720ab420a> in <module>
   ----> 1 int(s)


       ValueError: invalid literal for int() with base 10: 'abcdef'
```

[77]:
```
print(int.__doc__) # __doc__ --> doc string / documentation string
```

```
int([x]) -> integer
int(x, base=10) -> integer

Convert a number or string to an integer, or return 0 if no arguments
are given.  If x is a number, return x.__int__().  For floating point
numbers, this truncates towards zero.

If x is not a number or if base is given, then x must be a string,
bytes, or bytearray instance representing an integer literal in the
given base.  The literal can be preceded by '+' or '-' and be surrounded
by whitespace.  The base defaults to 10.  Valid bases are 0 and 2-36.
Base 0 means to interpret the base from the string as an integer literal.
>>> int('0b100', base=0)
4
```

[79]:
```
s = "abcdef" # hexa-decimal --> 16
```

[80]:
```
int(s, 16)
```

[80]:
```
11259375
```

[81]:
```
s = "1010"
int(s, 2)
```

[81]:
```
10
```

[84]:
```
s = '77'
int(s)
```

[84]:
```
77
```

[85]:
```
int(s, 8)
```

[85]:
```
63
```

[86]:
```
int('alkdjfksjdlkfjd')
```

```
␣
↪---------------------------------------------------------------------------

       ValueError                                Traceback (most recent call␣
↪last)

       <ipython-input-86-1d770410d3c7> in <module>
   ----> 1 int('alkdjfksjdlkfjd')


       ValueError: invalid literal for int() with base 10: 'alkdjfksjdlkfjd'
```

[87]:
```
d = float("10.56")
d
```

[87]:
```
10.56
```

[88]:
```
type(d)
```

[88]:
```
float
```

[89]:
```
c  = complex('123+3.5j')
```

[90]:
```
type(c)
```

[90]:
```
complex
```

[91]:
```
int(125.3323)
```

[91]:
```
125
```

[92]:
```
float(23)
```

[92]:
```
23.0
```

[93]:
```
float('24')
```

[93]:
```
24.0
```

[94]:
```
int('34.34')
```

```
                                                                                ␣
↪---------------------------------------------------------------------------

ValueError                                Traceback (most recent call␣
↪last)

<ipython-input-94-b5b46645f57c> in <module>
----> 1 int('34.34')


ValueError: invalid literal for int() with base 10: '34.34'
```

[96]:
```
x = input("X: ")
y = input("Y: ")
r = x + y
print(r)
```

```
X: 5
Y: 10
510
```

[98]:
```
x = input("X: ") # str
y = input("Y: ") # str
print(type(x), type(y))
print("_"*60)
x = int(x)
y  = int(y)
print(type(x), type(y))
r = x + y
print(r)
```

```
X: 5
Y: 10
<class 'str'> <class 'str'>

------------------------------------------------------------
<class 'int'> <class 'int'>
15
```

[99]:
```
x = int(input("x: "))
y = int(input("y: "))
r = x + y
print(r)
```

```
x: 5
y: 10
15
```

[101]:
```
bin(19)
```

[101]: '0b10011'

[102]:
```
hex(19)
```

[102]: '0x13'

[103]:
```
oct(19)
```

[103]: '0o23'

str

[105]:
```
str(34)
```

[105]: '34'

[106]:
```
str(45.6)
```

[106]: '45.6'

[107]:
```
str(45+66j)
```

[107]: '(45+66j)'

[108]:
```
str([1, 2,3 ,4 ])
```

[108]: '[1, 2, 3, 4]'

[109]:
```
str((1, 2,3 , 4))
```

[109]:
```
'(1, 2, 3, 4)'
```

[110]:
```
d = { 'name':'sachin', 'age': 20}
str(d)
```

[110]:
```
"{'name': 'sachin', 'age': 20}"
```

[111]:
```
s = { 1, 2, 3, 4, 5}
str(s)
```

[111]:
```
'{1, 2, 3, 4, 5}'
```

List

any iterable can be converted into list

[112]:
```
x = list('Python')
```

[113]:
```
print(x)
```

```
['P', 'y', 't', 'h', 'o', 'n']
```

[114]:
```
x = list([1, 2, 3, 4])
x
```

[114]:
```
[1, 2, 3, 4]
```

[115]:
```
x = list( ('hi', 'hello')  )
x
```

[115]:
```
['hi', 'hello']
```

[116]:
```
s = { 1, 2, 3,4 , 5}
x = list(s)
print(x)
```

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

[117]:
```
d = { 'name': 'sachin', 'age': 24, 'country': 'india', 'language':[
    'hindi', 'english'
]}
```

[118]:
```
print(d)
```

```
{'name': 'sachin', 'age': 24, 'country': 'india', 'language': ['hindi',
'english']}
```

[119]:
```
l = list(d)
```

[120]:
```
print(l)
```

```
['name', 'age', 'country', 'language']
```

[121]:
```
l = list(d.values())
l
```

[121]:
```
['sachin', 24, 'india', ['hindi', 'english']]
```

[123]:
```
l = list(d.items())
print(l)
```

```
[('name', 'sachin'), ('age', 24), ('country', 'india'), ('language', ['hindi',
'english'])]
```

0.0.2 dict

[124]:
```
l =  [ ('hello', 'hello world'), ['name', 'sachin'], ['language', ('hindi',␣
↪'english')]  ]
```

[125]:
```
d = dict(l)
print(d)
```

```
{'hello': 'hello world', 'name': 'sachin', 'language': ('hindi', 'english')}
```

[129]:
```
l = [ ('name', 'sachin', ), ('age', 24), 'py', {'hello': 'bye', 'world': 'hi'}]
d = dict(l)
print(d)
```

```
{'name': 'sachin', 'age': 24, 'p': 'y', 'hello': 'world'}
```

set

any iterable can be converted into set

[132]:
```
s = 'python is python and python is awesome.'
```

[133]:
```
s = set(s)
```

[134]:
```
print(s)
```

```
{'m', 'o', 'a', '.', 's', 'h', ' ', 'w', 't', 'y', 'e', 'd', 'n', 'i', 'p'}
```

[135]:
```
s = set( ['hello', 1, 2, 1, 2, 3, 1, 'hello', 'hello', 1, 2, 3, 1,2 ,3 ] )
```

[136]:
```
print(s)
```

```
{1, 2, 3, 'hello'}
```

[137]:
```
set([ 'hello', 'hi', 'bye', 'good', 'bye', 'hello', 'hello', 'good'])
```

[137]:
```
{'bye', 'good', 'hello', 'hi'}
```

[138]:
```
d = { 'name': 'sachin', 'age': 24, 'you': 'me'}
```

[139]:
```
set(d)
```

[139]:
```
{'age', 'name', 'you'}
```

[142]:
```
set(d.keys())
```

[142]:
```
{'age', 'name', 'you'}
```

[143]:
```
set(d.values())
```

[143]:
```
{24, 'me', 'sachin'}
```

[147]:
```
frozenset(d.items())
```

[147]:
```
frozenset({('age', 24), ('name', 'sachin'), ('you', 'me')})
```

Boolean

[148]:
```
bool
```

[148]:
```
bool
```

[149]:
```
bool(False)
```

[149]:
```
False
```

[150]:
```
bool(True)
```

[150]:
```
True
```

[151]:
```
bool(0)
```

[151]:
```
False
```

[152]:
```
bool(None)
```

[152]:
```
False
```

[153]:
```
bool('')
```

[153]:
```
False
```

[154]:
```
bool([])
```

[154]: False

[155]:
```
bool(())
```

[155]: False

[157]:
```
bool({})
```

[157]: False

[158]:
```
bool(' ')
```

[158]: True

[159]:
```
bool([ '' ])
```

[159]: True

0.0.3 Operators in Python

which operats on data

tools to work with raw_data to shape it into meaningfull data

how to store data in memory ?

    Data Types

how to process data in memory ?

    using built-in functions of python and data classes of python

    operators

operation theaters

operends (patients)

operator (doctor)

Unary Operators

Binary Operators

Ternery Operators

[161]:
```
x = 5

y = 10
```

[162]:
```
x + y


# x is left side operand
# y is right side operand

# + is a binary operator
```

[162]: 15

Type of Operators in Python

Airthmatic Operators

+, -, *, /, %, //, **

Comparision Operators

>, <, <=, >=, ==, !=

Logical Operators

or, and , not

Membership Operators

in, not in

Identity Operators

is , is not

Bit-wise Operators

&, |, ^, ~, <<, >>

Asignment Operators

+=, -=, *=, /=, %=, **=, //=

Airthmatic Operators

Works well with number not with others

[163]:
```
5 + 5 # plus
```

[163]: 10

[164]:
```
"hello " + "world" # concatination
```

[164]: 'hello world'

[165]:
```
[ 'java', 'c', 'c++'] + [ 'python', 'ruby']
```

[165]: ['java', 'c', 'c++', 'python', 'ruby']

[166]:
```
('java', 'c') + ('python', 'c++')
```

[166]: ('java', 'c', 'python', 'c++')

[168]:
```
l1= [ 1, 2, 3, 4]
l2 = [ 4, 5, 6, 7]

print(id(l1))
print(id(l2))

l1 = l1 + l2
print(id(l3))
print(l1)
```

```
2855286192008
2855286193032
2855285893448
[1, 2, 3, 4, 4, 5, 6, 7]
```

[169]:
```
l1= [ 1, 2, 3, 4]
l2 = [ 4, 5, 6, 7]
print(id(l1))
print(id(l2))
l1.extend(l2)
print(id(l1))
print(l)
```

```
2855293652936
2855287555336
2855293652936
[('name', 'sachin'), ('age', 24), 'py', {'hello': 'bye', 'world': 'hi'}]
```

[170]:
```
{ 'name': 'sachin'} + { 'age': 23}
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-170-0cb7fcf94b1b> in <module>
----> 1 { 'name': 'sachin'} + { 'age': 23}


TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
```

[171]:
```
56 -70
```

[171]:
```
-14
```

[172]:
```
'hello ' - 'world'
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-172-fbecfce7a0ff> in <module>
----> 1 'hello ' - 'world'


TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

[173]:
```
5 * 5
```

[173]:
```
25
```

[176]:
```
22 / 7  # true division, absolute division
```

[176]:
```
3.142857142857143
```

[178]:
```
22 // 7 # int division, floor division
```

[178]:
```
3
```

[183]:
```
int(22 / 7 )
```

[183]: 3

[180]:
```
from math import floor, ceil
```

[181]:
```
floor(10.9)
```

[181]: 10

[182]:
```
ceil(10.1)
```

[182]: 11

[184]:
```
round(10.4)
```

[184]: 10

[188]:
```
round(11.5)
```

[188]: 12

[189]:
```
round(10.6)
```

[189]: 11

[191]:
```
round(6.63986218372189379281739 2)
```

[191]: 7

[192]:
```
round(6.639862183721893792817392, 2)
```

[192]: 6.64

[193]:
```
round(5, 2)
```

[193]: 5

[194]:
```
round(5.4, 3)
```

[194]: 5.4

[195]:
```
round(5.5555555, 2)
```

[195]: 5.56

[197]:
```
x  = 5
s = f"{x:.2f}"
print(s)
```

```
5.00
```

[198]:
```
5 / 3
```

[198]:
```
1.6666666666666667
```

[199]:
```
5 // 3
```

[199]:
```
1
```

[201]:
```
10 % 3
```

[201]:
```
1
```

[202]:
```
13  % 7
```

[202]:
```
6
```

[203]:
```
5 % 25
```

[203]:
```
5
```

[204]:
```
5 ** 2
```

[204]:
```
25
```

[205]:
```
2 ** 3
```

[205]:
```
8
```

[206]:
```
"Hello " * 5 # replication
```

[206]:
```
'Hello Hello Hello Hello Hello '
```

[207]:
```
[ 'java'] * 5
```

[207]:
```
['java', 'java', 'java', 'java', 'java']
```

[209]:
```
print("_"*60)
```

```
------------------------------------------------------------
```

[210]:
```
"hello " * "world"
```

```
␣
↪------------------------------------------------------------------------
```

```
TypeError                                 Traceback (most recent call last)

<ipython-input-210-e6904ae5aa91> in <module>
----> 1 "hello " * "world"


TypeError: can't multiply sequence by non-int of type 'str'
```

Comparision

[211]:
```
5 > 6
```

[211]: False

[212]:
```
6 < 6
```

[212]: False

[213]:
```
6 >= 6
```

[213]: True

[214]:
```
5 != 6
```

[214]: True

[215]:
```
5 == 7
```

[215]: False

>, <, >=, <=, ==, !=

Logical Operators

or

[216]:
```
True or True
```

[216]: True

[217]:
```
True or False
```

[217]: True

[218]:
```
False or True
```

[218]: True

[219]:
```
False or False
```

[219]: False

[220]:
```
None or 0
```

[220]: 0

[221]:
```
5 or 6
```

[221]: 5

[223]:
```
print(print("Hello World") or print("Bye World") )
```

```
Hello World
Bye World
None
```

[225]:
```
6 or 0
```

[225]: 6

[226]:
```
0 or 6
```

[226]: 6

[227]:
```
True and True
```

[227]: True

[228]:
```
True and False
```

[228]: False

[229]:
```
False and False
```

[229]: False

[230]:
```
output = print("Hello World") and print("Bye World")

# ouput =  None and print("Bye World")

print(output)

print(bool(output))
```

```
Hello World
None
False
```

unary operator

[231]:
```
not 0
```

[231]: True

[232]:
```
not 1
```

[232]: False

[233]:
```
not False
```

[233]: True

[234]:
```
not False
```

[234]: True

[235]:
```
not 5 > 6
```

[235]: True

&&, ||, !

and, or, not

Membership

in or not int

[236]:
```
'hello' in 'hello world'
```

[236]: True

[237]:
```
'java' in [ 'c', 'c++', 'ruby', 'perl', 'python']
```

[237]: False

[238]:
```
s = { 'name': 'sachin', 'age': 24, 'color':'fair'}
```

[239]:
```
'country' in s # keys
```

[239]: False

[240]:
```
'language' not in s # keys
```

[240]: True

Identity Operator

[241]:
```
x = 5

y = x
```

[242]:
```
x is y
```

[242]: True

[243]:
```
x is not y
```

[243]: False

[244]:
```
l = [ 1, 2, 3, 4, 5]

l1 = l
```

[245]:
```
l is l1
```

[245]: True

[246]:
```
s = "hello world"
s1 = 'hello world'
```

[247]:
```
s == s1
```

[247]: True

[248]:
```
s is s1
```

[248]: False

[249]:
```
id(s) != id(s1)
```

[249]: True

Bit-wise Operator

[250]:
```
x = 17
```

[251]:
```
print(bin(x))
```

0b10001

16 8 4 2 1

1 0 0 0 1

[252]:
```
x = 5
y = 3
```

[253]:
```
# 1 0 1 --> 5
# 0 1 1 --> 3
```

[254]:
```
5 & 3
```

[254]: 1

[255]:
```
# 1 0 1 --> 5
# 0 1 1 --> 3
# 0 0 1 --> 1
```

[256]:
```
5 | 3
```

[256]: 7

[257]:
```
# 1 0 1 --> 5
# 0 1 1 --> 3
# 1 1 1 --> 7
```

[258]:
```
5 ^ 3
```

[258]: 6

[259]:
```
# 1 0 1 --> 5
# 0 1 1 --> 3
# 1 1 0 --> 6
```

[260]:
```
~ 5
```

[260]: -6

[262]:
```
~ -6
```

[262]: 5

`<<` left shift

`>>` right shift

[265]:
```
5 << 3
```

[265]: 40

1 0 1

1 0 1 0

1 0 1 0 0 1 0 1 0 0 0

32 16 8 4 2 1

[266]:
```
5 >> 3
```

[266]:
```
0
```

10 real data store

Operator Precedence

[268]:
```
5+6*3/2-4**3//2*6
```

[268]:
```
-178.0
```

[ ]:
```

```