A Gentle Introduction to Python

Lecture - 3

Operators, Expression and Data types

Today's Outline

- Previous Session:
  - Introduction to Token.
  - Statements and Expressions.

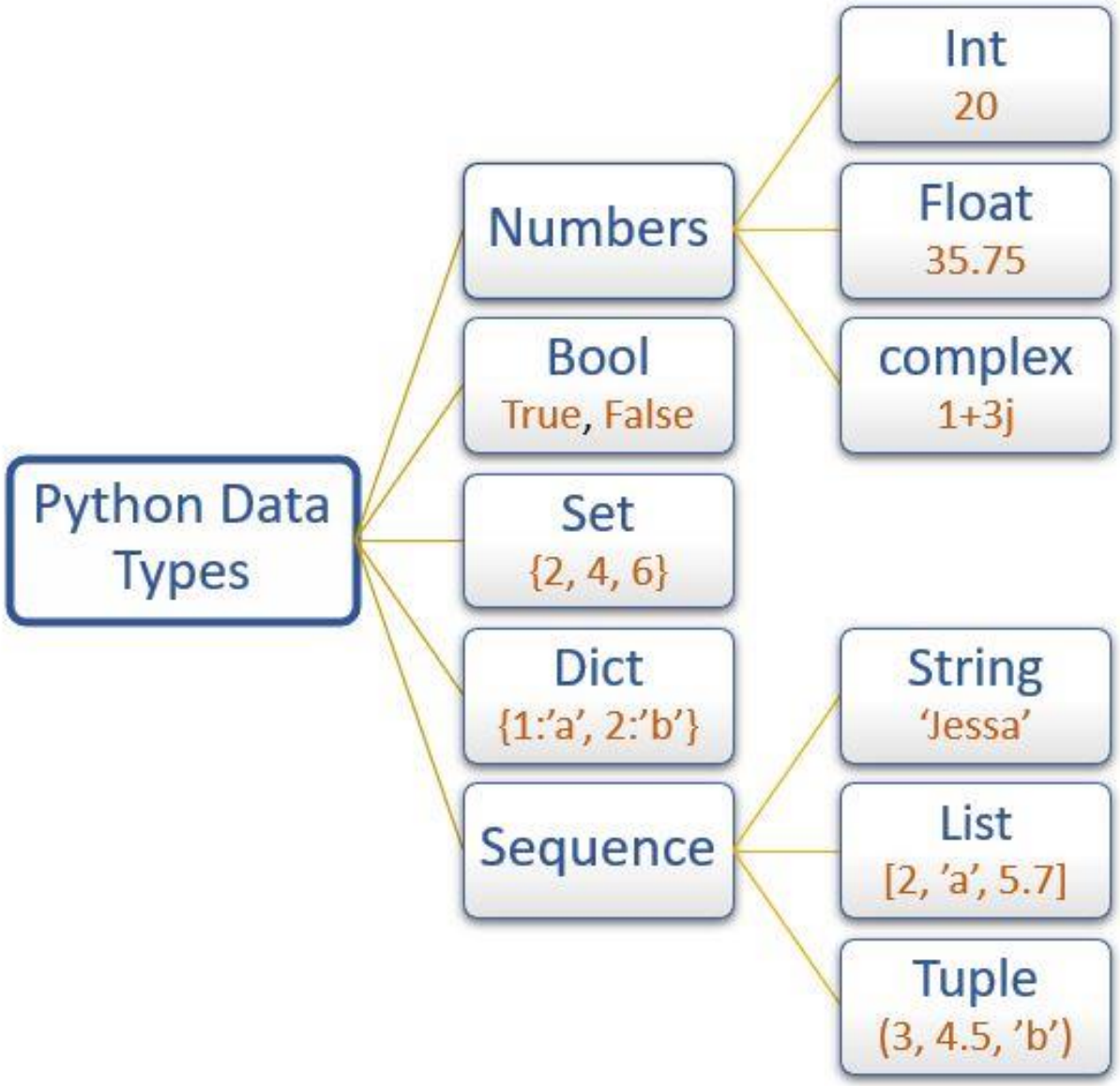
- Today's Session:
  - Data Types, indexing and Operators
    - Mutable and Immutable data types.
    - Data type conversion.
    - Operators.

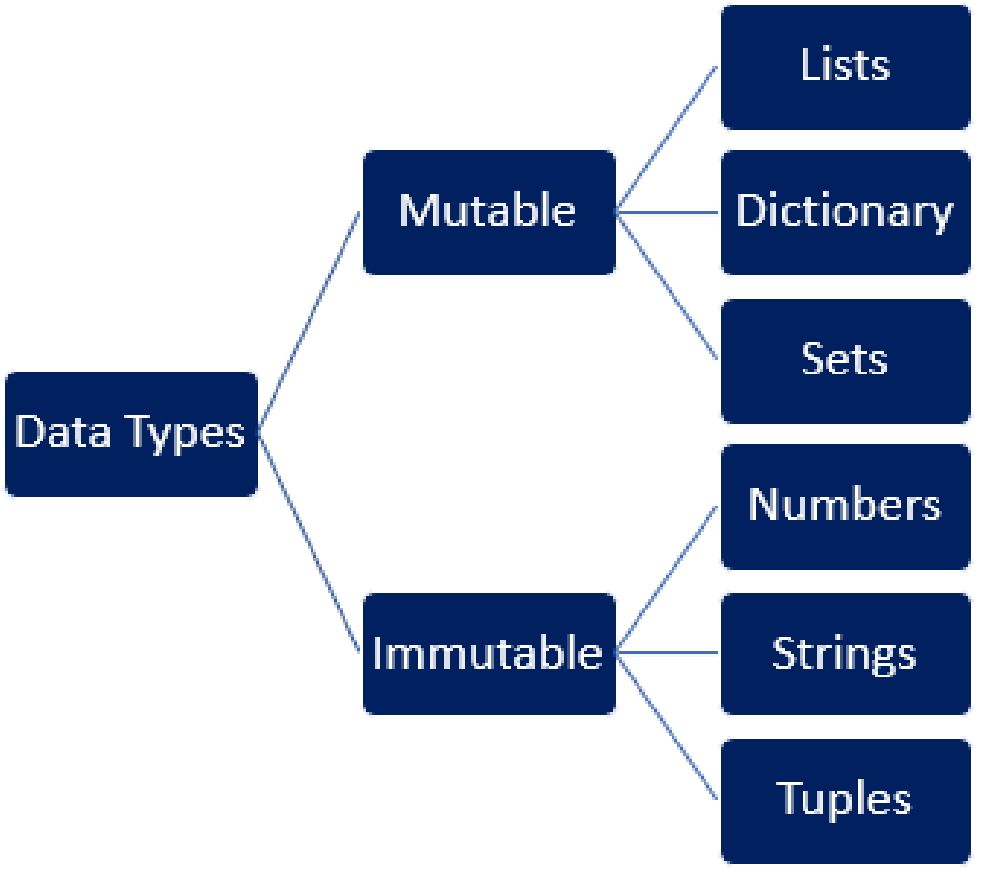
Data Types

- Data type is the classification of the type of values that can be assigned to variables.
- **Dynamically typed languages**, the interpreter itself predicts the data type of the Python Variable based on the type of value assigned to that variable.

  - x is a variable and 2 is its value
  - x can be assigned different values; hence, its type changes accordingly

```
>>> x = 2
>>> type(x)
int
>>> x = 2.3
>>> type(x)
float
```


Python Data Types
Numbers
Int
20
Float
35.75
complex
1+3j
Bool
True, False
Set
{2, 4, 6}
Dict
{1:'a', 2:'b'}
Sequence
String
'Jessa'
List
[2, 'a', 5.7]
Tuple
(3, 4.5, 'b')


Data Types
Mutable
Lists
Dictionary
Sets
Immutable
Numbers
Strings
Tuples

Data Types (Numbers)

The number data type is divided into the following five data types:

- Integer
- Long Integer (removed from py3)
- Floating-point Numbers
- Complex Numbers

```
>>> a = 2
>>> type(a)
int
>>> a = 2.5
>>> type(a)
float
>>> a = 0o11
 >>> type(a)
int
```

```
>>> a = 0x19
>>> type(a)
int
>>> a = 2 + 5j
>>>type(a)
complex
>>> type(a)
float
>>> a = 9999999L
>>> type(a)
long
```

Data Types (String)

- Python string is an ordered collection of characters which is used to represent and store the text-based information.
- Strings are stored as individual characters in a contiguous memory location.
- It can be accessed from both directions: forward and backward.

```
>>> a = “Shiv Nadar”
>>> print(a)
Shiv Nadar
>>> a = ‘University’
>>> print(a)
University
```

Data Types (String)

- Characters of string can be individually accessed using a method called **indexing**.
-
- Forward indexing starts form *0, 1, 2....*
- Backward indexing starts form *−1, −2, −3...*

```
>>> a = “Shiv Nadar University”
>>> print(a[5])
N
>>> print(a[-1])
y
>>> print(a[-5])
r
```

- ***Mutable Data Types:*** Data types in python where the value assigned to a variable can be changed
- ***Immutable Data Types:*** Data types in python where the value assigned to a variable cannot be changed

Data Structure	Ordered	Mutable	Constructor	Example
List	Yes	Yes	`[ ]` or `list()`	`[5.7, 4, 'yes', 5.7]`
Tuple	Yes	No	`( )` or `tuple()`	`(5.7, 4, 'yes', 5.7)`
Set	No	Yes	`{}`* or `set()`	`{5.7, 4, 'yes'}`
Dictionary	No	Yes**	`{ }` or `dict()`	`{'Jun': 75, 'Jul': 89}`

Lists

```
In [1]:
L1=[10,25.5,3+2j,"Hello"]
L1
Out[1]:
[10, 25.5, (3+2j), 'Hello']
```

In above list, each item is of different type.

```
In [5]:
L1[2]=1.22E-5
L1
Out[5]:
[10, 25.5, 1.22e-05, 'Hello']
```

Tuples

```
In [3]:
T1=(10,25.5,3+2j,"Hello")
T1
Out[3]:
(10, 25.5, (3+2j), 'Hello')
In [4]:
T1[1]
Out[4]:
25.5
```

```
In [13]:
L1=[10, 30, 30, 50, 20, 40, 11, 22]
T1=tuple(L1)
print (T1)
(10, 30, 30, 50, 20, 40, 11, 22)
```

```
In [6]:
T1[2]=1.22E-5
T1
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-18a8ef5d3740> in <module>()
----> 1 T1[2]=1.22E-5
      2 T1
TypeError: 'tuple' object does not support item assignment
```

Data Types (Tuples)

- Tuple data type in Python is a collection of various immutable Python objects separated by commas.
- Tuples are generally store different Python Data Types.
- A Python tuple is created using parentheses around the elements in the tuple.

```
>>> a = (1,2,3,4)
>>> print(a)
(1,2,3,4)
>>> a = ('ABC','DEF','XYZ')
>>>print(a)
(ABC,DEF,XYZ)
```

Data Types (Tuple)

- To access an element of a tuple, we simply use the index of that element. We use square brackets.
- **Reverse Indexing** by using indexes as -1, -2, -3, and so on, where -1 represents the last element.
- Slicing: Extract some elements from the tuple.

```
>>> a = (1,2,3,4)
>>> print(a[1])
2
>>> a = ('ABC','DEF','XYZ')
>>>print(a[2])
XYZ
```

```
>>> a = (1,2,3,4)
>>> print(a[-1])
4
>>> a = ('ABC','DEF','XYZ')
>>>print(a[1:])
(DEF, XYZ)
```

Data Types (List)

Unlike strings, **lists can contain any sort of objects; numbers, strings, and even other lists.** Python lists are:

- Ordered collections of arbitrary objects
- Accessed by offset
- Arrays of object references
- Variable length, heterogeneous, and arbitrarily nestable
- Mutable
- Starting index is 0
- Enclosed between square brackets '[ ]'

```
>>> a = [2,3,4,5]
>>> b = ["Shiv",
"Nadar", "University"]
>>> print(a,b)
>>> [2,3,4,5]['Shiv',
'Nadar', 'University']
```

Data Types (List)

- Much like strings, we can use the index number to access items in lists as shown below.

- Accessing a List Using Reverse Indexing
  - To access a list in reverse order, we must use indexing from −1, −2.... Here, −1 represents the last item in the list.

```
>>> a = [“Shiv”, “Nadar” “University”, “Computer”]
>>> print(a[0])
Shiv
>>> print(a[-1])
Computer
>>> print(a[1])
University
```

Data Types (Set)

- It is an **unordered collection** of elements which means elements don't have a specific order.
- A collection that stores elements of different Python Data Types.
- Sets in Python can't have duplicates. Each item is unique.
- The elements of a set in Python are immutable. They can't accept changes once added.

```
>>> myset = {“Shiv Nadar”, “computer”, “science”}
>>>print(myset)
{‘Shiv Nadar', 'computer', 'science’}
>>>myset = set((“Shiv Nadar”, “computer”, “science”))
```

Data Types (Dictionary)

- An unordered collection of elements.
- A dictionary contains keys and values rather than just elements.
- Unlike lists the values in dictionaries are accessed using keys and not by their positions

```
>>>dict1 ={“Branch”: “computer”, “College”: “SNU”, “year”:2011}
>>>print (dict1)
{‘Branch’:’computer’,’College’:’SNU’,’year’:2011}
>>>di = dict({1: ‘abc’,2: ‘xyz’})
```

Data Types (Dictionary)

- The keys are separated from their respective values by a colon (:) between them, and each key–value pair is separated using commas (,).
- All items are enclosed in curly braces.
- While the values in dictionaries may repeat, the keys are always unique.
- The value can be of any data type, but the keys should be of immutable data type, that is
- We can access value of a key using the key inside square brackets.

```
>>>dict1 ={“Branch”:”computer”,”College”:”SNU”,”year”:2011}
>>>print (dict1[year])
2011
```

Datatype Conversion

- We can do various kinds of conversions between strings, integers and floats using the built-in *int*, *float*, and *str* functions

```
>>> x = 10
>>> float(x)
10.0
>>> str(x)
'10'
>>>
```

integer → float
integer → string

```
>>> y = "20"
>>> float(y)
20.0
>>> int(y)
20
>>>
```

string → float
string → integer

```
>>> z = 30.0
>>> int(z)
30
>>> str(z)
'30.0'
>>>
```

float → integer
float → string

Explicit and Implicit Data Type Conversion

- Data conversion can happen in two ways in Python
  1. Explicit Data Conversion (we saw this earlier with the *int*, *float*, and *str* built-in functions)

  2. Implicit Data Conversion
     - Takes place *automatically* during run time between *ONLY* numeric values
       - E.g., Adding a float and an integer will automatically result in a float value
       - E.g., Adding a string and an integer (or a float) will result in an *error* since string is not numeric
     - Applies *type promotion* to avoid loss of information
       - Conversion goes from integer to float (e.g., upon adding a float and an integer) and not vice versa so as the fractional part of the float is not lost

Implicit Data Type Conversion: Examples

- The result of an expression that involves a float number alongside (an) integer number(s) is a float number

```
>>> print(2 + 3.4)
5.4
>>> print( 2 + 3)
5
>>> print(9/5 * 27 + 32)
80.6
>>> print(9//5 * 27 + 32)
59
>>> print(5.9 + 4.2)
10.100000000000001
>>>
```

Implicit Data Type Conversion: Examples

- The result of an expression that involves a float number alongside (an) integer number(s) is a float number
- The result of an expression that involves values of the same data type will not result in any conversion

```
>>> print(2 + 3.4)
5.4
>>> print( 2 + 3)
5
>>> print(9/5 * 27 + 32)
80.6
>>> print(9//5 * 27 + 32)
59
>>> print(5.9 + 4.2)
10.100000000000001
>>>
```

Operators

- Arithmetic Operators (* *, *,/,//,%,+,_)
- Comparison Operators (<, <=, >, >=, ==)
- Python Assignment Operators (=, +=, -=, *=, /=)
- Logical Operators (and, or, not)
- Bitwise Operators (&, |, ~, >>, <<, ^)
- Membership Operators (in, not in)

Operator

- Arithmetic operator

Low

Precedence

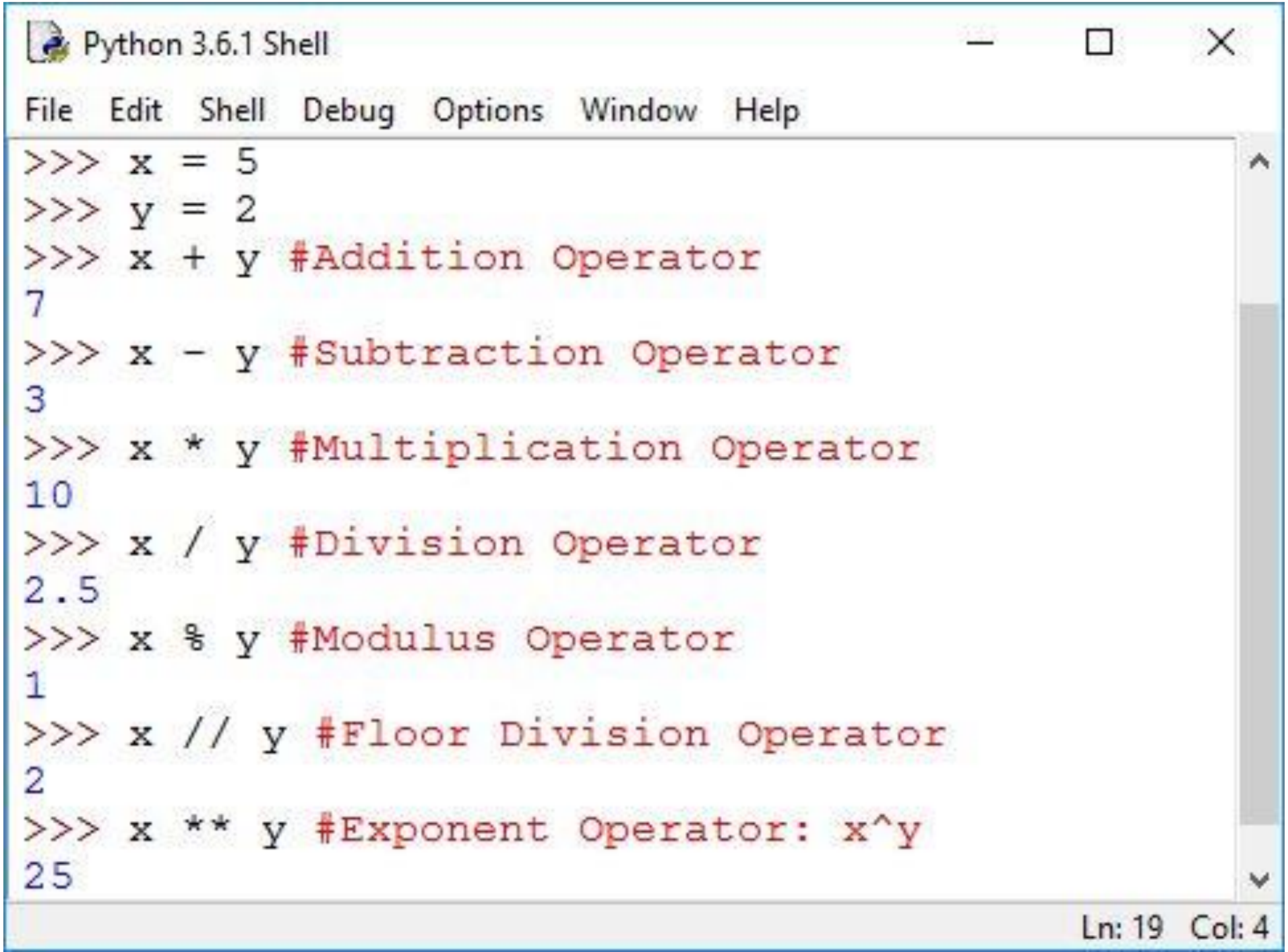
High

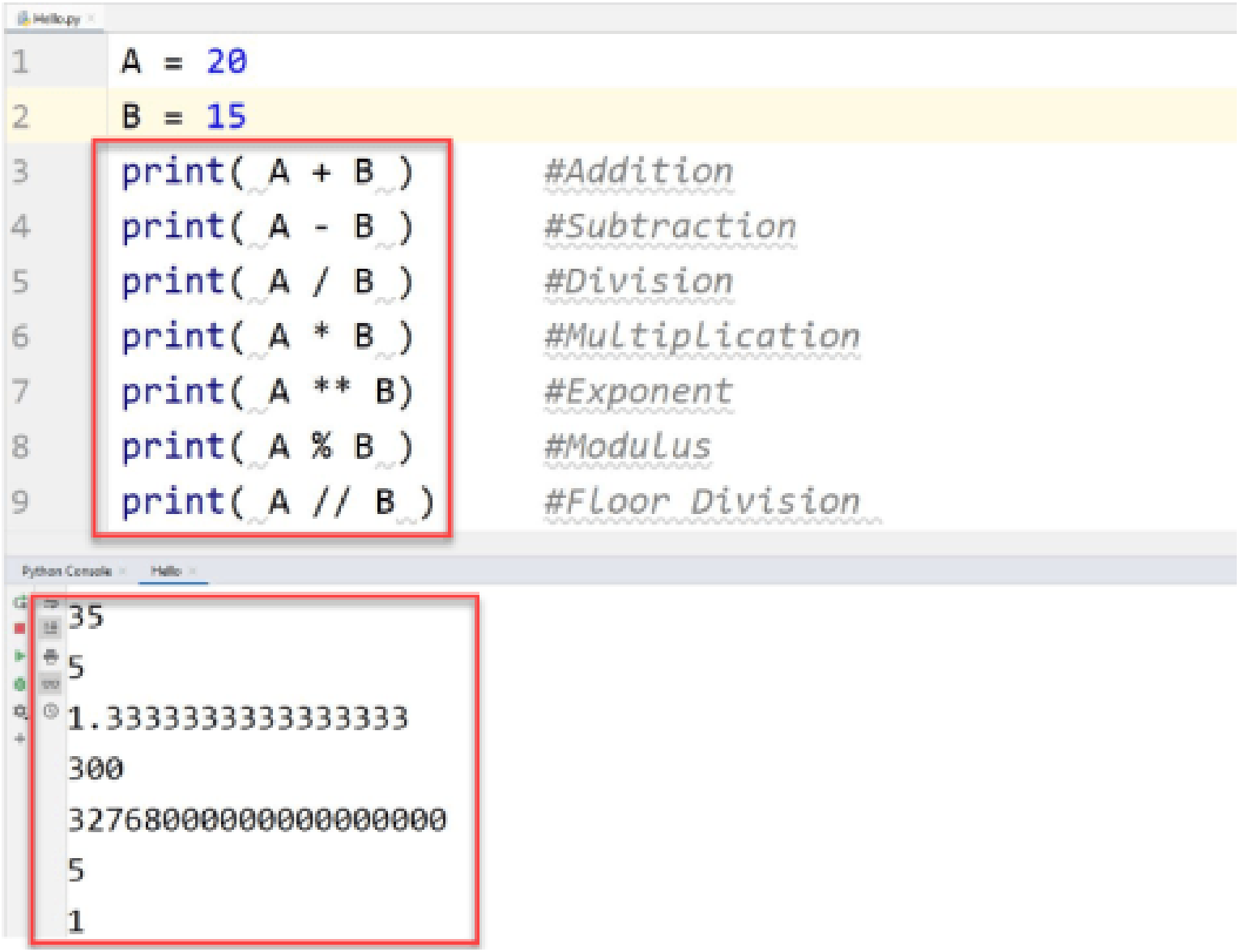
+	-	*	/	//	%	**
Plus	minus	multiplication	Float division	Integer division	Mod (remainder)	power
3	3	2	2	2	2	1

Left to right

Right to left

```
>>> 5.0/3
1.6666666666666667
>>> 45.0/3
15.0
>>> 2/7.0
0.2857142857142857
>>> 2/7.
0.2857142857142857
>>>
```


Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
>>> x = 5
>>> y = 2
>>> x + y #Addition Operator
7
>>> x - y #Subtraction Operator
3
>>> x * y #Multiplication Operator
10
>>> x / y #Division Operator
2.5
>>> x % y #Modulus Operator
1
>>> x // y #Floor Division Operator
2
>>> x ** y #Exponent Operator: x^y
25
Ln: 19 Col: 4


Hello.py
1 A = 20
2 B = 15
3 print(A + B) #Addition
4 print(A - B) #Subtraction
5 print(A / B) #Division
6 print(A * B) #Multiplication
7 print(A ** B) #Exponent
8 print(A % B) #Modulus
9 print(A // B) #Floor Division
Python Console
Hello
35
5
1.3333333333333333
300
32768000000000000000
5
1

Lets solve -

20//9

-20//9

20//-9

-20//-9

a%b = a – (b * (a//b))

20%9

-20%9

20%-9

-20%-9

Lets solve -

$5/10*5+5*2$

$2**3**1$

$5//10*5+5*2$

$1**3**2$

$5\%10*5+5*2$

$2**1**3$

Built-in Functions

abs(x)	# returns absolute value of x
pow(x, y)	# returns value of x raised to y
min(x1, x2,...)	# returns smallest argument
max(x1, x2,...)	# returns largest argument
divmod(x, y)	# returns a pair(x // y, x % y)
round(x [,n])	# returns x rounded to n digits after .
bin(x)	# returns binary equivalent of x
oct(x)	# returns octal equivalent of x
hex(x)	# returns hexadecimal equivalent of x

Assign value in variables; x, y and n.
Check the working in Python

Mathematical Functions

```
pi, e           # values of constants pi and e
sqrt(x)         # square root of x
factorial(x)    # factorial of x
fabs(x)         # absolute value of float x
log(x)          # natural log of x (log to the base e)
log10(x)        # base-10 logarithm of x
exp(x)          # e raised to x
trunc(x)        # truncate to integer
ceil(x)         # smallest integer >= x
floor(x)        # largest integer <= x
modf(x)         # fractional and integer parts of x
```

Assign a value in variable x and check the working in Python

Comparison Operators

Boolean expressions ask a question

- Produce a Yes or No result which we use to control program flow

Boolean expressions using comparison operators evaluate to:

- True / False - Yes / No

Comparison operators look at variables

- But do not change the variables

Operator	Description
==	Equals to
!=	Not equals to
<>	Not equals to
>	Greater than
<	Less Than
>=	Greater Than Equals to
<=	Less Than Equals to

==	**Equals to is used for comparison**
=	**Is used for assignment**

Example

```
x=5
print(x>3)
print(x==5)
print(x>=5)
print(x<=5)
print(x<6)
print(x!=6)
```

```
True
True
True
True
True
True
```

```
a = 5
b = 5
c = 3
print(a = b)
print(a = c)
print(b = c)
```

```
a = 5
b = 3
c = 4
d = 4
e = 5
print(a == b)
print(a == c)
print(a == d)
print(a == e)
```

```
False
False
False
True
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-93bdf5d98946> in <module>()
      2 b = 5
      3 c = 3
----> 4 print(a = b)
      5 print(a = c)
      6 print(b = c)

TypeError: 'a' is an invalid keyword argument for this function
```

Logical operators

- Logical operators are the and, or, not 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

Logical AND/OR

Boolean True/False

AND		
X	**Y**	**Output**
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

```
>>>a=7>7 and 2>-1
>>>print(a)
FALSE
```

```
>>>a=7>7 or 2>-1
>>>print(a)
TRUE
```

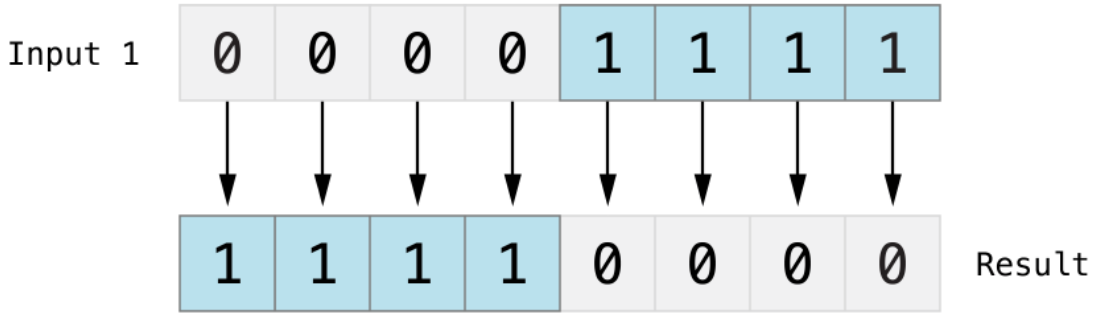
```
>>>print(7 and 0 or 5)
5
```

OR		
X	**Y**	**Output**
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Logical operators NOT

Just the reverse of what is there.

- not(true) → false




```
bool_1 = not (True or True)
bool_2 = True and (False or False)
bool_3 = True or (False or False)
bool_4 = (True or not True) and (True and True)
bool_5 = (3>5) or (5<4 and not 5>=7)
print(bool_1)
print(bool_2)
print(bool_3)
print(bool_4)
print(bool_5)
```

```
False
False
True
True
False
```

- The unary `+` operator in Python refers to the identity operator. This simply returns the integer after it. This is why it is an *identity operation* on the integer
- For example, the value of `+5` is simply `5`, and for `+-5`, it is `-5`. This is a unary operator, which works on real numbers
- The `++a` will be parsed as `+` and `+a`, but the second `+a` is again treated as `(+a)`, which is simply `a`
- Therefore, `+(+(a))` simply evaluates to `a`.

So, even though we wanted to increment the value of `a` by one, we cannot achieve this using the `++` symbols, since this kind of operator, does not exist.

We must, therefore, use the `+=` operator to do this kind of increment.

```
a += 1
a -= 1
```

Bitwise operators

Bitwise operators act on operands as if they were string of binary digits.

It operates bit by bit.

- 2 is 10 in binary and 7 is 111.

In the table below:

- Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

Operator	Description
& Binary AND	Operator copies a bit to the result if it exists in both operands
| Binary OR	It copies a bit if it exists in either operand.
^ Binary XOR	It copies the bit if it is set in one operand but not both.
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.


&
0
*
1
1
0
1
Λ
0
1
a
a
1
0
|
*
1
0
0
1
0
~
0
1
1
0

Bitwise operators
AND

5 & 7

- Same as 101 & 111.
- This results in 101
- Which is binary for 5.

```
print(5&7)
```

```
5
```

Operation	Output
0 & 0	0
0 & 1	0
1 & 0	0
1 & 1	1

Operation	Output
0 | 0	0
0 | 1	1
1 | 0	1
1 | 1	1

4 | 8

- Binary for 4 is 0100, and that for 8 is 1000.
- After I operation, output is 1100
- Which is binary for 12.

```
print(4|8)
```

```
12
```

```
"{0:b}".format(5)
```

```
'101'
```

- Bitwise operators are used in encryption algorithms and applications

Bitwise operators
XOR

XOR (exclusive OR) returns 1

- If one operand is 0 and another is 1.

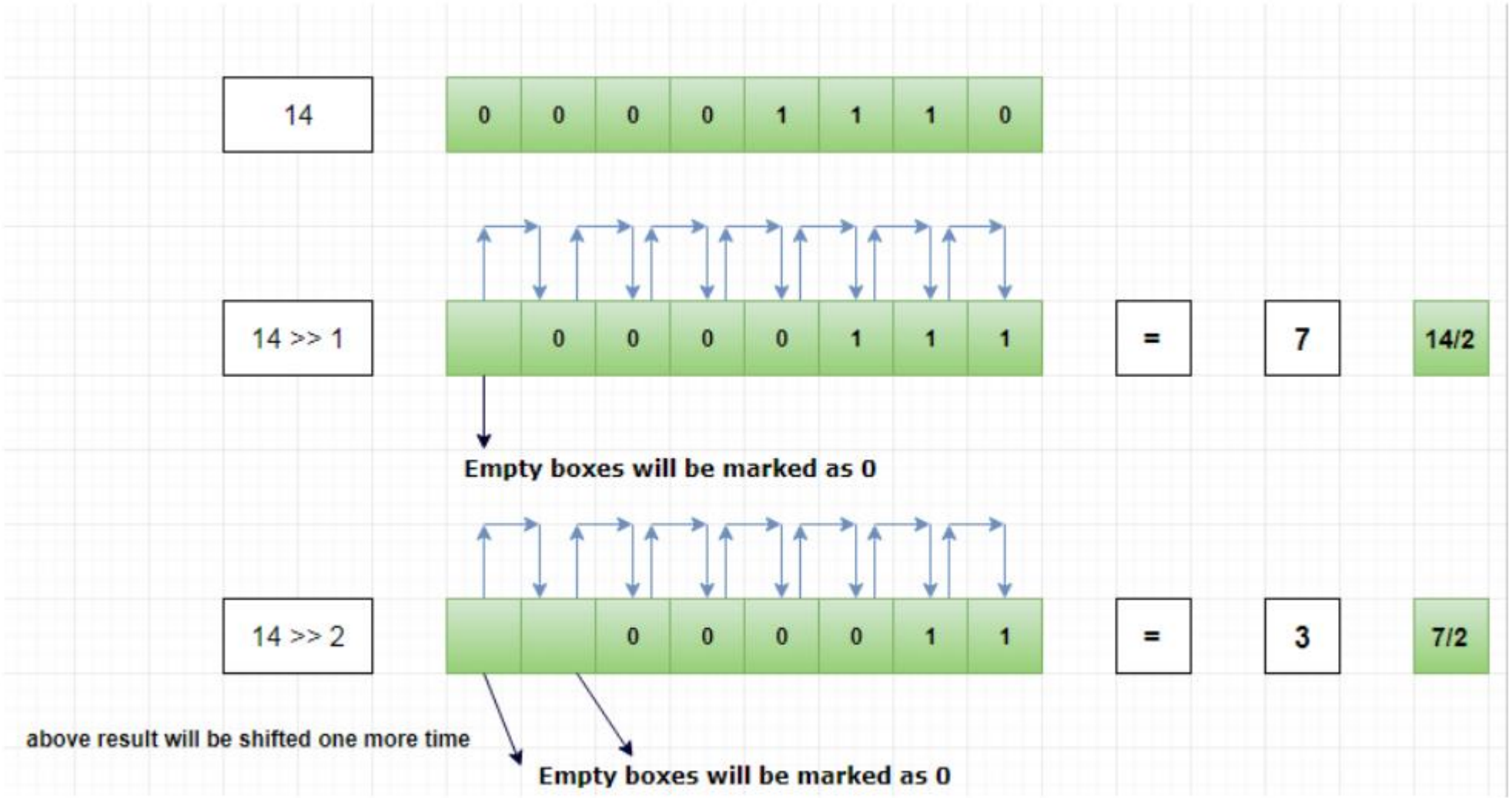
Otherwise, it returns 0.

Operator	Output
0^0	0
0^1	1
1^0	1
1^1	0

```
print(5^3)
```

6

Shift operator



A shift operator **performs bit manipulation on data by shifting the bits of its first operand right or left**

Shift operator

Left Shift (<<)

Right Shift (>>)

```
In [60]: 12<<1
Out[60]: 24

In [61]: 12>>1
Out[61]: 6

In [62]: 12<<2
Out[62]: 48

In [63]: 12>>2
Out[63]: 3
```

Let's Solve

15^13

12 & 10

11|00

5<<2

5>>2

Bitwise operators

A	B	A | B	A & B	A^ B	~A
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

Operator Precedence

Operators	Meaning
()	Parentheses
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==, !=, >, >=, <, <=, in, not in	Comparison, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

Let's Solve

10*4>>2 and 3

```
5|10&12>>2
```

10%(15<10 and 20<30)

```
A=5
A+=10//5
print(A)
```

10/(5-5)

```
A=5
A**=10//5
print(A)
```

2.5%0.15

```
a=7
a%=6.5/7<=7 and 4<=6
a
```

Membership Operators

Membership Operators	Examples	Result
`in`	`10 in (10, 20, 30)`	`True`
	`red in ('red','green','blue')`	`True`
`not in`	`10 not in (10, 20, 30)`	`False`