Applied Statistical Methods

Introduction to Python - Part II

Xuemao Zhang
East Stroudsburg University

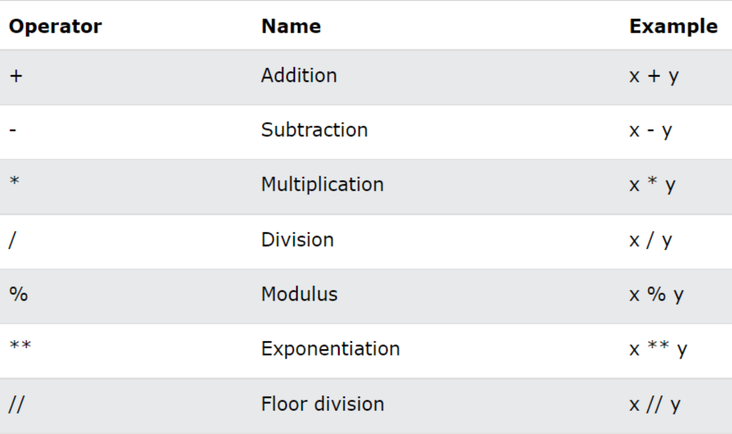
January 25, 2023

Outline

- Operators are used to perform operations on variables and values.
- Python operators
  - Arithmetic Operators
  - Bitwise Operators
  - Assignment Operators
  - Comparison Operators
  - Logical Operators
  - Membership Operators
  - Identity Operators
- Numpy array operators

Arithmetic Operators

- Arithmetic operators are used with numeric values to perform common mathematical operations



Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Arithmetic Operators

```
x = 5
y = 3
print(x + y)
```

```
## 8
```

```
print(x - y)
```

```
## 2
```

```
print(x * y)
```

```
## 15
```

```
print(x / y)
```

```
## 1.6666666666666667
```

```
print(x // y)
```

```
## 1
```

Arithmetic Operators

```
print(x**2)
```

```
## 25
```

```
print(x % y)
```

```
## 2
```

```
print(6 % y)
```

```
## 0
```

- We consider Arithmetic operations for arrays in numpy later

Arithmetic Operators

- String Concatenation

```
a = "Hello, "
b = "World!"
print(a+b)
```

```
## Hello, World!
```

Arithmetic Operators

- List Concatenation by append() or extend()

```
list1 = [100, 50, 65, 82, 23]
list2= [1,2,3,4,5]
list1.extend(list2)
print(list1)
```

```
## [100, 50, 65, 82, 23, 1, 2, 3, 4, 5]
```

- List Concatenation by function sum()

```
list1 = [100, 50, 65, 82, 23]
list2= [1,2,3,4,5]
sum((list1, list2),[])
```

```
## [100, 50, 65, 82, 23, 1, 2, 3, 4, 5]
```

Arithmetic Operators

- List Concatenation by + operator

```
list1 = [100, 50, 65, 82, 23]
list2= [1,2,3,4,5]
print(list1+list2)
```

```
## [100, 50, 65, 82, 23, 1, 2, 3, 4, 5]
```

- List Concatenation by ∗ operator

```
list1 = [100, 50, 65, 82, 23]
list2= [1,2,3,4,5]
list3 = [*list1, *list2]
print(list3)
```

```
## [100, 50, 65, 82, 23, 1, 2, 3, 4, 5]
```

Arithmetic Operators

- Tuple Concatenation by + operator

```
x=(1,2)
y=(3,4,5)
print(x+y)
```

```
## (1, 2, 3, 4, 5)
```

- Tuple Concatenation by function sum

```
x=(1,2)
y=(3,4,5)
print(sum((x,y),()))
```

```
## (1, 2, 3, 4, 5)
```

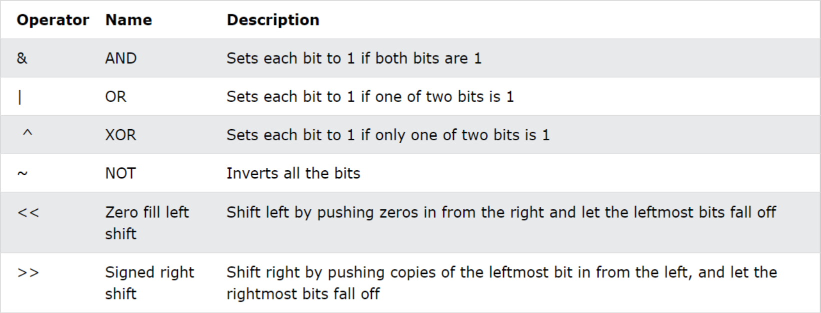
- Or we can convert a tuple to a list and convert it back to a tuple after the operations

```
x=list((1,2))
y=list((3,4,5))
x.extend(y)
print( tuple(x) )
```

```
## (1, 2, 3, 4, 5)
```

Bitwise Operators

- Bitwise operators are used to compare (binary) numbers
- Python bitwise operators work only on integers. see https://www.geeksforgeeks.org/python-bitwise-operators/



Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Bitwise Operators

```
a = 10
b = 4
print(a & b) # Print bitwise AND operation
```

```
## 0
```

```
print(a | b) # Print bitwise OR operation
```

```
## 14
```

```
print(~a) # Print bitwise NOT operation
```

```
## -11
```

```
print(a ^ b)# print bitwise XOR operation
```

```
## 14
```

```
print(a >> 2) # print bitwise right shift operation
```

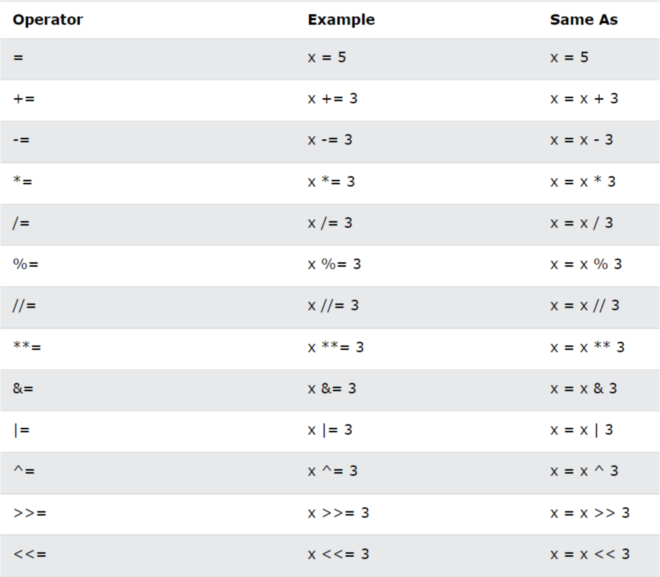
```
## 2
```

```
print(a << 2) # print bitwise left shift operation
```

```
## 40
```

Assignment Operators

- Assignment operators are used to assign values to variables



Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Assignment Operators

```
x = 5
x += 3
print(x)
```

```
## 8
```

```
x -= 3
print(x)
```

```
## 5
```

```
x *= 2
print(x)
```

```
## 10
```

```
x /= 3
print(x)
```

```
## 3.3333333333333335
```

Assignment Operators

```
x=10
x //= 3
print(x)
```

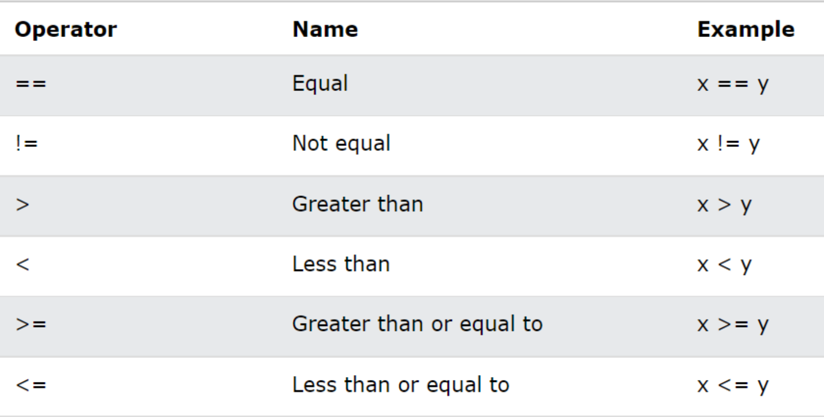
```
## 3
```

```
x **= 4
print(x)
```

```
## 81
```

Comparison Operators

- Comparison operators are used to compare two values



Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Comparison Operators

```
a = 13
b = 33
print(a > b)
```

```
## False
```

```
print(a >= b)
```

```
## False
```

```
print(a < b)
```

```
## True
```

```
print(a <= b)
```

```
## True
```

```
print(a == b)
```

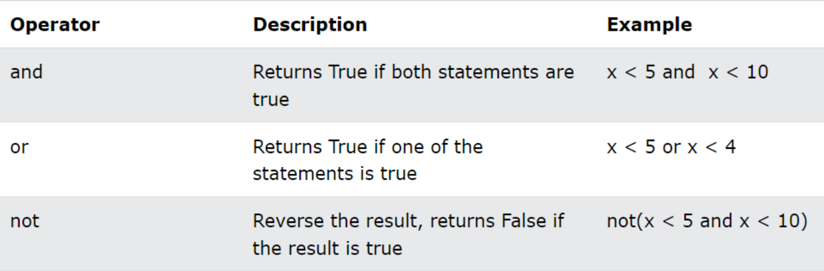
```
## False
```

```
print(a != b)
```

```
## True
```

Logical Operators

- Logical operators are used to **combine conditional statements**



Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Logical Operators

```
a = True
b = False

print(a and b)
```

```
## False
```

```
print(a or b)
```

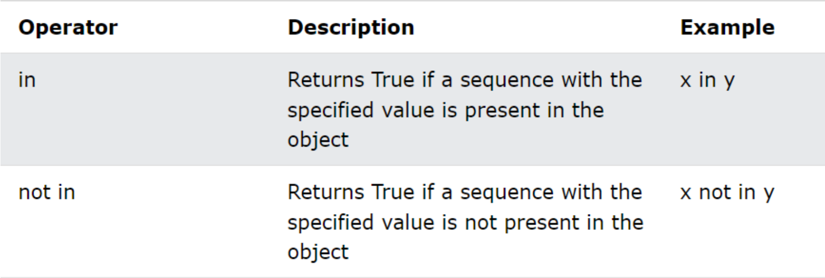
```
## True
```

```
print(not a)
```

```
## False
```

Membership Operators

- Membership operators are used to test if a sequence is presented in an object



Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Membership Operators

```python
x = 24
y = 20
list = [10, 20, 30, 40, 50]

print(x in list)
```

```
## False
```

```python
print(x not in list)
```

```
## True
```

```python
print(y in list)
```

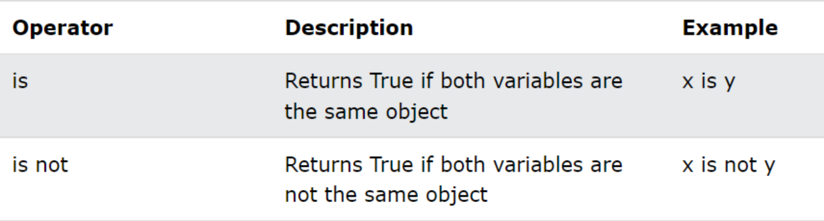
```
## True
```

```python
print(y not in list)
```

```
## False
```

Identity Operators

- Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location



Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Identity Operators

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is z)
# returns True because z is the same object as x
```

```
## True
```

```
print(x is y)
# returns False because x is not the same object as y,
# even if they have the same content
```

```
## False
```

```
print(x == y)
# to demonstrate the difference betweeen "is" and "==":
# this comparison returns True because x is equal to y
```

```
## True
```

Identity Operators

```python
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is not z)
# returns False because z is the same object as x
```

```
## False
```

```python
print(x is not y)
# returns True because x is not the same object as y,
# even if they have the same content
```

```
## True
```

```python
print(x != y)
# to demonstrate the difference betweeen "is not" and "!=":
# this comparison returns False because x is equal to y
```

```
## False
```

Numpy array operations

- In python matrix can be implemented as 2D list or 2D Array. Forming matrix using **2D Arrary**, gives the additional functionalities for performing various operations in matrix.
  - https://www.geeksforgeeks.org/numpy-tutorial/?ref=lbp
- Operation on Matrix:
  - 1. **add()**:- This function is used to perform element wise matrix addition.
  - 2. **subtract()**:- This function is used to perform element wise matrix subtraction.
  - 3. **divide()**:- This function is used to perform element wise matrix division.
  - 4. **multiply()**:- This function is used to perform element wise matrix multiplication.
  - 5. **dot()**:- This function is used to compute the matrix multiplication, rather than element wise multiplication.
  - 6. **sqrt()**:- This function is used to compute the square root of each element of matrix.
  - 7. **sum(x,axis)**:- This function is used to add all the elements in matrix. Optional "axis" argument computes the row sum if axis is 0 and column sum if axis is 1.
  - 8. `T`:- This argument is used to transpose the specified matrix.

Numpy array operators

```
import numpy as np
x = np.array([[1, 2], [4, 5]])
y = np.array([[7, 8], [9, 10]])
print(x)
```

```
## [[1 2]
##  [4 5]]
```

```
print(y)
```

```
## [[ 7  8]
##  [ 9 10]]
```

```
print(np.shape(x))  #dimensions of the matrix
```

```
## (2, 2)
```

Numpy array operators

```
print(np.add(x,y))
```

```
## [[ 8 10]
##  [13 15]]
```

```
print(x+y)
```

```
## [[ 8 10]
##  [13 15]]
```

```
print(np.subtract(x,y))
```

```
## [[-6 -6]
##  [-5 -5]]
```

```
print(x-y)
```

```
## [[-6 -6]
##  [-5 -5]]
```

Numpy array operators

```
print(np.divide(x,y)) #elementwise
```

```
## [[0.14285714 0.25      ]
##  [0.44444444 0.5       ]]
```

```
print(x/y)
```

```
## [[0.14285714 0.25      ]
##  [0.44444444 0.5       ]]
```

Numpy array operators

```
print(np.multiply(x,y)) #elementwise
```

```
## [[ 7 16]
##  [36 50]]
```

```
print(x*y)
```

```
## [[ 7 16]
##  [36 50]]
```

```
print(np.dot(x,y))
```

```
## [[25 28]
##  [73 82]]
```

Numpy array operators

```
print(np.sqrt(x))
```

```
## [[1.         1.41421356]
##  [2.         2.23606798]]
```

```
print(np.sum(x))
```

```
## 12
```

```
print(np.sum(x, 1)) # column sum
```

```
## [3 9]
```

```
print(np.sum(x, 0)) # row sum
```

```
## [5 7]
```

Numpy array operators

- Combining Arrays

```
np.concatenate((x,y),axis=0)
```

```
## array([[ 1,  2],
##        [ 4,  5],
##        [ 7,  8],
##        [ 9, 10]])
```

```
np.concatenate((x,y),axis=1)
```

```
## array([[ 1,  2,  7,  8],
##        [ 4,  5,  9, 10]])
```

Numpy array operators

- Combining Arrays

```
np.vstack((x,y))
```

```
## array([[ 1,  2],
##        [ 4,  5],
##        [ 7,  8],
##        [ 9, 10]])
```

```
np.hstack((x,y))
```

```
## array([[ 1,  2,  7,  8],
##        [ 4,  5,  9, 10]])
```

```
np.column_stack((x,y))
```

```
## array([[ 1,  2,  7,  8],
##        [ 4,  5,  9, 10]])
```

Numpy array operators

- Transpose

```
print( np.transpose(x) )
```

```
## [[1 4]
##  [2 5]]
```

```
print(x.T)
```

```
## [[1 4]
##  [2 5]]
```

Numpy array operators

- Reshape without changing data

```
A = np.array([[1, 2, 3], [4, 5, 6]])
print(A)
```

```
## [[1 2 3]
##  [4 5 6]]
```

```
print(A.reshape(3,2))
```

```
## [[1 2]
##  [3 4]
##  [5 6]]
```

Numpy array operators

```
print(A.reshape(-1,1))
```

```
## [[1]
##  [2]
##  [3]
##  [4]
##  [5]
##  [6]]
```

```
print(A.reshape(1,-1))
```

```
## [[1 2 3 4 5 6]]
```

Numpy array operators

- It is no longer recommended to use `numpy.matrix` based on https://numpy.org/doc/stable/reference/generated/numpy.matrix.html. So we skip the `numpy.matrix` operators.

License