# Python Operators –

### 1. Arithmetic Operators

Used for basic math.

| Operator | Name | Example |
|---|---|---|
| + | Addition | a + b |
| - | Subtraction | a - b |
| * | Multiplication | a * b |
| / | Division | a / b |
| // | Floor Division | a // b |
| % | Modulus (remainder) | a % b |
| ** | Exponentiation | a ** b |

### 2. Assignment Operators

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 |

### 3. Comparison Operators

Used to compare two values.

| Operator | Meaning | Example |
|---|---|---|
| == | Equal to | a == b |
| != | Not equal to | a != b |
| > | Greater than | a > b |
| < | Less than | a < b |
| >= | Greater than or equal to | a >= b |
| <= | Less than or equal to | a <= b |

### 4. Logical Operators

Used to combine conditional statements.

| Operator | Description | Example |
|---|---|---|
| **and** | True if both are True | a > 2 and b < 5 |
| **or** | True if at least one is True | a > 2 or b < 5 |
| **not** | Inverts the result | not(a > 2) |

### 5. Bitwise Operators

Operate on binary numbers.

| Operator | Name | Example |
|---|---|---|
| **&** | AND | a & b |
| **\|** | OR | A\|b |
| ^ | XOR | a ^ b |
| ~ | NOT | ~a |
| << | Left Shift | a << 2 |
| >> | Right Shift | a >> 2 |

### 6. Membership Operators

Test for membership in a sequence (like list, string, etc.)

| Operator | Description | Example |
|---|---|---|
| **in** | True if found | 'a' in 'apple' |
| **not in** | True if not found | 'x' not in 'apple' |

### 7. Identity Operators

Compare memory locations.

| Operator | Description | Example |
|---|---|---|
| **Is** | True if same object | a is b |
| **is not** | True if not same object | a is not b |

## ➢ Arithmetic Operators –

A = 24

B = 6

1. C = A+B

   print(C)            output = 30

2. C = A-B

   print(C)            output = 18

3. C = A*B
   Print(C)            output = 144
4. C = A / B
   Print(C)            output = 4
5. C = A // B
   Print(C)            output = 0
6. C = A ** B (it means 24 ^ 6)
   Print(C)            output = 191102976
7. C = A % B (It returns the remainder after division of two numbers)
   Print(C)            output  = 1.44

## ➢ Assignment Operators

1. x = 10
   x += 5
   print(x)            output = 15      (Because x = x+5, x = 10+5)
2. x = 12
   x -= 6
   print(x)            output = 6
3. x = 28
   x *= 5
   print(x)            output = 140
4. x = 27
   x /= 9
   print(x)            output = 3
5. x = 25
   x %= 5
   print(x)            output = 0
6. x = 15
   x //= 3
   print(x)            output = 5
7. x = 5
   x **= 3
   print(x)            output = 125

## ➢ Logical Operators

**AND**

```
a = 5
b = 10

print(a > 2 and b < 15)
print(a > 2 and b < 8)
```

**OR -**

```
print(a > 2 or b < 15)
print(a > 2 or b < 8)
print(not(a > 2 or b < 8))
```

output -

```
True

False

True
True
False
```

**NOT -**

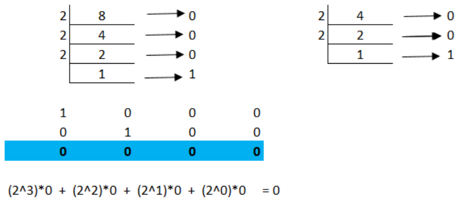
- ‘Not’Reverse the result, returns False if the result is true & returns true if the result is false.
- example - 1

```
a = 5
b = 10

print(not (a > 2))  #False
print(not (a > 6))  #True
```

➢ Bitwise Operators

**AND –**

```
x = 5  #  0101
x &= 3  # 0011; Equivalent to x = x & 3
print(x)  #0001
bin(x)

output – 1
```



```
1    0    0    0
0    1    0    0
0    0    0    0
```

(2^3)*0 + (2^2)*0 + (2^1)*0 + (2^0)*0   = 0

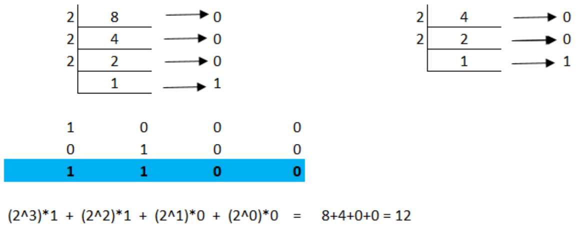
**example – 2**

```
a =8  #1000
b =4  #100

c =a & b

print(c) #0000
bin(c)
output - 0
```

**OR –**

```
a =8  #1000
```

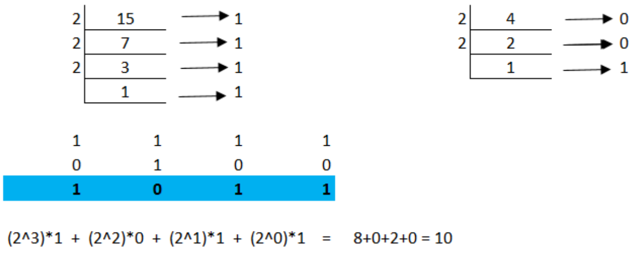
```
b =4  #100

c =a | b

print(c) #1100
bin(c)
output =
```



(2^3)*1 + (2^2)*1 + (2^1)*0 + (2^0)*0 = 8+4+0+0 = 12

### XOR –

```
a =15  #1111
b =4  #100

c =a ^ b

print(c) #1011
bin(c)
output = 10
```



(2^3)*1 + (2^2)*0 + (2^1)*1 + (2^0)*1 = 8+0+2+0 = 10

### Not -

```
a = 5  # 0101
result = ~a  # 1010 (in 2's complement form, this is -6)
print(result)
```

output - -6

### left shift

### right shift

- Membership Operators

### In –

fruits = ["apple", "banana", "cherry"]

print("banana" in fruits)     # Output: True

print("mango" in fruits)      # Output: False

### Not In -

name = "hello world"

print("h" not in name)        # Output: False

print("z" not in name)        # Output: True

## Python Conditional Statements

In Python, **conditional statements** are used to perform different actions based on different conditions.

**If statement -**

```
age = 18
if age >= 18:
    print("You are eligible to vote.")
```

Output - You are eligible to vote.

**If else statement -**

```
age = 16
if age >= 18:
    print("You are eligible to vote.")
else:
    print("You are not eligible to vote.")
```

output - You are not eligible to vote.

**Elif statement -**

```
marks = 85

if marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
elif marks >= 60:
    print("Grade C")
else:
    print("Fail")
```

Output – Grade B

**Another Example –**

```
amount = 750

if amount >= 1000:
    print("You get a 20% discount!")
elif amount >= 750:
    print("You get a 15% discount!")
elif amount >= 500:
    print("You get a 10% discount!")
elif amount >= 250:
    print("You get a 5% discount!")
else:
    print("No discount available. Shop more to save more!")
```

output - You get a 15% discount!

**Nested If statement -**

```
num = 10
if num > 0:
    if num % 2 == 0:
        print("Positive Even Number")
    else:
        print("Positive Odd Number")
else:
    print("Non-positive Number")
```

Output – ‘Positive Even Number’

Python Loop Statements –

In Python, **for** loop is used to iterate over a sequence (like a list, tuple, string, or range) and execute a block of code multiple times.

**Example -1**

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

Output -

```
apple
banana
cherry
```

**Example – 2**

```
for i in range(1, 6):
    print(i)
```

Output –

```
1
```

```
2
3
4
5
```

If you want in horizontal output –

```
for i in range(1, 6):
    print(i, end=”,”)
```

Output -

```
1, 2, 3, 4, 5,
```

While Loops –

In Python, a while loop is used to **repeat a block of code as long as a condition is true**.

```
i = 1
while i <= 5:
    print(i)
    i += 1
```

Output –

```
1
2
3
4
5
```

**Example - 1**

```
count = 0

while count <= 100:  #condition
    print(count)
    count +=2
Output –
0
2
4
.
.
.
.
.

76
78
80
82
84
```

```
86
88
90
92
94
96
98
100   (0 to 100 all even numbers)
```

## Loop Control Statements

1. break The break statement is used to exit the loop prematurely, regardless of the loop's condition.

```
i = 1

while i <= 5:

    if i == 3:

        break  # Exit the loop when i is 3

    print(i)

    i += 1
```

Output –

```
1

2
```

Continue Statement -

```
i = 0

while i < 5:

    i += 1

    if i == 3:

        continue  # Skip printing when i is 3

    print(i)
```

output –

```
1

2

4

5
```

## NumPy :-

**NumPy** (Numerical Python) is a powerful Python library used for numerical computing. It provides support for:

- **Multi-dimensional arrays (ndarrays)**
- **Mathematical functions** to operate on arrays
- **Linear algebra**, **Fourier transforms**, **random number generation**, and more

## Why Use NumPy?

- **Speed:** Much faster than regular Python lists
- **Convenient:** Many built-in functions for mathematical operations
- **Efficient Memory Use**

## NumPy Array:-

```
Example - arr = np.array([[10,20,30,40],[50,60,70,80],[20,30,40,50]])
                print(arr)
```

### Output-

```
[[10 20 30 40]
 [50 60 70 80]
 [20 30 40 50]]
```

### Index In Array-

```
Example - arr = np.array([[10,20,30,40],[50,60,70,80],[20,30,40,50]])
                print(arr[2,0:2])
```

### Output:-

```
[20 30]
```

### Find The Size Of Array-

```
Example -  print(np.size(arr))
```

### Output-

```
12
```

### Find The Shape Of Array-

```
print(np.shape(arr))
```

### Output-

```
(3,4)   #3=Rows, 4= Columns
```

### Find The Datatype Of A Array-

```
print(arr.dtype)
```

### Output-

```
int64
```

## Find Dimension-

```
a = [10,20,30,40]
arr = np.array(a)
print(arr.ndim)
```

**Output = 1**

**NumPy Addition**
**NumPy Substraction**
**NumPy Multiplication**
**NumPy Division**
**NumPy Exponentiation**

**All are in one example -**

```
arr1 = np.array([10,20,50,60,80])

arr2 = np.array([10,20,30,10,20])

print(arr1+arr2)

print(np.add(arr1,arr2))

print(arr1*arr2)

print(np.multiply(arr1,arr2))

print(arr1/arr2)

print(np.divide(arr1,arr2))

print(arr1^arr2)
```

## Ouput-

```
[ 20  40  80  70 100]
[ 20  40  80  70 100]
[ 100  400 1500  600 1600]
[ 100  400 1500  600 1600]
[1.         1.         1.66666667 6.         4.        ]
[1.         1.         1.66666667 6.         4.        ]
[ 0  0 44 54 68]
```

## NumPy Power Function –

```
arr1 = np.array([10,20,50,60,80])
arr2 = np.array([4])
print(np.power(arr1,arr2))
```

## Ouput –

```
[   10000   160000  6250000 12960000 40960000]
```

## Check Equal Or Not (Equal Function Numpy)-

```
arr1 = np.array([10,20,50,60,80])
arr2 = np.array([4])
```

```
print(np.equal(arr1,arr2))
```

### Output -

```
[False False False False False]
```

### Concatenate In NumPy-

```
arr1 = np.array([10,20,30,50])
arr2 = np.array([10,50,60,80])
print(np.concatenate([arr1,arr2]))
```

### Ouput-

```
[10 20 30 50 10 50 60 80]
```

### Sorting In Array-

```
import numpy as np
ar = np.array([[10,20,30,40,60,10,70,50],[10,50,60,20,90,40,50,10]])
print(np.sort(ar))
```

### Ouput –

```
[[10 10 20 30 40 50 60 70]
 [10 10 20 40 50 50 60 90]]
```

### Search & Sort also in one code using NumPy-

```
import numpy as np
a = np.array([10,20,30,40,60,10,70,50])
b = np.searchsorted(a,50)
print(b)
```

### Ouput –

**4**

### Sum-
### Max
### Min
### Mean
### Size
### Cumsum in Array Using NumPy-

```
a = np.array([10,20,30,40,50,60,70,80,90])

print(np.sum(a))
print(np.max(a))
print(np.min(a))
print(np.mean(a))
print(np.size(a))
print(np.cumsum(a))
```

### Ouput-

```
450
```

```
90
10
50.0
9
[ 10  30  60 100 150 210 280 360 450]
```

### How to write a code which gives number randomly-

```
import numpy as np
array = np.random.randint(0,10,(6,5))
print(array)
```

### Ouput –

```
[[2 2 5 6 5]
 [3 0 3 0 0]
 [6 8 5 7 8]
 [2 7 8 6 5]
 [3 3 1 5 8]
 [9 9 9 8 9]]
```

➢

```
import numpy as np
np.full((2,5),3)
```

This code create a array full of 3

### Ouput-

```
array([[3, 3, 3, 3, 3],
       [3, 3, 3, 3, 3]])
```

# Pandas Library:-

# Python- Pandas Library

- Pandas is a powerful, flexible, and easy-to-use open-source data manipulation and analysis library in Python. It is widely used for working with structured data, such as tables, time series, and other data types.

## Pandas provides two main data structures:

1. Series: A one-dimensional labeled array capable of holding any data type.
2. DataFrame: A two-dimensional labeled data structure with columns that can be of different types (like a table or a spreadsheet).

## Key Features of Pandas:

- Data Handling: Pandas allows you to read and write data in various formats, such as CSV, Excel, SQL, and JSON.
- Data Cleaning: You can handle missing data, duplicate entries, and perform transformations efficiently.
- Data Analysis: Pandas offers various functions for aggregating, grouping, and pivoting data.
- Indexing and Selection: You can easily select, filter, and slice data using labels or conditions.

- Time Series: Pandas has powerful tools for handling time series data, including date-time indexing and frequency handling.

# Common Operations in Pandas:

- Creating DataFrames and Series: You can create DataFrames from dictionaries, lists, or NumPy arrays.
- Reading and Writing Data: Functions like pd.read_csv(), pd.read_excel(), and pd.to_sql() are used to read and write data in various formats.
- Handling Missing Data: Functions like fillna(), dropna(), and isna() help handle missing values.
- Grouping and Aggregating: Use groupby() for grouping data and functions like mean(), sum(), and count() for aggregation.

## Creating a Data Series in Pandas –

```
import pandas as pd
data = [1,2,3,4]

series = pd.Series(data)
print(series)
```

## Output-

```
0    1
1    2
2    3
3    4
dtype: int64
```

## Creating A Data Frame In Pandas-

```
import pandas as pd
data = [1,2,3,4]
df = pd.DataFrame(data)
print(df)
```

## Ouput-

```
   0
0  1
1  2
2  3
3  4
```

### Another Example-

```
import pandas as pd
import numpy as np
array = np.array([[5000,6000], ["Suresh","Ramesh"]])
df= pd.DataFrame({"Name":array[1],"Salary":array[0]})
print(df)
```

## Ouput-

```
     Name Salary
0  Suresh   5000
1  Ramesh   6000
```

## Concatenate In Pandas-

```
 import pandas as pd
 df1 = pd.DataFrame({
     'A':[10,20,30,40],
     'B':[100, 200, 300, 400]
 })
 df2 = pd.DataFrame({
     'A':[10,50,60,30],
     'B':[100,180,200,300]
 })
con = pd.concat([df1,df2])
print(con)
```

## Output-

```
    A    B
0  10  100
1  20  200
2  30  300
3  40  400
0  10  100
1  50  180
2  60  200
3  30  300
```

## If you want to write side wise-

```
con = pd.concat([df1,df2], axis=1)
print(con)
```

## Ouput-

```
    A    B   A    B
0  10  100  10  100
1  20  200  50  180
2  30  300  60  200
3  40  400  30  300
```

## If you want to ignore the index number, which is repeat after one dataframe ends. And you want it continue till end the use-

```
import pandas as pd

# Create two DataFrames
df1 = pd.DataFrame({
    'A': ['A0', 'A1', 'A2'],
    'B': ['B0', 'B1', 'B2']
})

df2 = pd.DataFrame({
    'A': ['A3', 'A4', 'A5'],
    'B': ['B3', 'B4', 'B5']
})

# Concatenate DataFrames and reset index
result = pd.concat([df1, df2], ignore_index=True)

print(result)
```

## Ouput-

```
   A   B
0  A0  B0
1  A1  B1
2  A2  B2
3  A3  B3
4  A4  B4
5  A5  B5
```

## What is pandas.merge()?

In **Pandas**, merge() is used to **combine two DataFrames** based on **common columns or indexes**, similar to SQL joins (INNER, LEFT, RIGHT, OUTER).

```
import pandas as pd


# First DataFrame
df1 = pd.DataFrame({
    'ID': [1, 2, 3],
    'Name': ['Alice', 'Bob', 'Charlie']
})
# Second DataFrame
df2 = pd.DataFrame({
    'ID': [1, 2, 4],
    'Score': [85, 90, 75]
})
# Merge on column 'ID'
merged_df = pd.merge(df1, df2, on='ID')
print(merged_df)
```

## Output (INNER JOIN by default):

```
   ID   Name  Score
0   1  Alice     85
1   2    Bob     90
```

## Types of Merge (like SQL joins)-

```
# INNER JOIN (default)
pd.merge(df1, df2, on='ID', how='inner')
```

```
# LEFT JOIN
pd.merge(df1, df2, on='ID', how='left')

# RIGHT JOIN
pd.merge(df1, df2, on='ID', how='right')

# OUTER JOIN
pd.merge(df1, df2, on='ID', how='outer')
```

## Example: LEFT JOIN

```
pd.merge(df1, df2, on='ID', how='left')
```

## Output:

```
   ID   Name  Score
0   1  Alice   85.0
1   2    Bob   90.0
2   3 Charlie   NaN
```

- **Charlie has no matching ID in df2, so Score is NaN.**
  **In this way Right Join & Outer Join also perform.**

### Left Join-

```
df1 = pd.DataFrame({
    'key':['A','B','C','D'],
    'value':[50,60,70,80]
})
df2 = pd.DataFrame({
    'key':['A','G','E','F'],
    'value':[60,70,80,90]
})
result=pd.merge(df1,df2,on='key',how='left')
print(result)
```

## Right Join-

```
df1 = pd.DataFrame({
    'key':['A','B','C','D'],
    'value':[50,60,70,80]
})
df2 = pd.DataFrame({
    'key':['A','G','E','F'],
    'value':[60,70,80,90]
})
result=pd.merge(df1,df2,on='key',how='right')
print(result)
```

## Output-

```
  key  value_x  value_y
0   A     50.0       60
1   G      NaN       70
2   E      NaN       80
3   F      NaN       90
```

## Outer Join-

```
import pandas as pd
df1 = pd.DataFrame({
    'key':['A','B','C','D'],
    'Value':[50,60,80,70]
})
df2 = pd.DataFrame({
    'key':['B','D','E','F'],
    'value':[100,200,500,600]
})
result = pd.merge(df1,df2,on='key',how='outer')
print(result)
```

## Output-

```
  key  Value  value
0   A   50.0    NaN
1   B   60.0  100.0
2   C   80.0    NaN
3   D   70.0  200.0
4   E    NaN  500.0
5   F    NaN  600.0
```

```
# DataFrame 1
df1 = pd.DataFrame({
    'ID': [1, 2, 3, 4],
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [23, 34, 25, 40]
})

# DataFrame 2
df2 = pd.DataFrame({
    'ID': [3, 4, 5, 6],
    'Department': ['HR', 'Finance', 'IT', 'Marketing'],
    'Salary': [50000, 60000, 55000, 70000]
})

# Set ID as the index
df1.set_index('ID', inplace=True)
df2.set_index('ID', inplace=True)

print(df1)

print(df2)
```

Here I use the Row ID as Index.

### Output –

```
      Name  Age
ID
1    Alice   23
2      Bob   34
3  Charlie   25
4    David   40
   Department  Salary
ID
3          HR   50000
4     Finance   60000
5          IT   55000
6   Marketing   70000
```

## Now Import Dataset –

## Importing Excel Dataset Or Import CSV Files –

```
from google.colab import files
uploaded = files.upload()

import pandas as pd
data = pd.read_csv('mtcars2.csv')
print(data.head(10))
```

Then the whole data appears.