In [2]:
```python
import os
import pandas as pd

# Specify the drive or folder to scan
drive_path = "D:\\"  # Change this to the drive or folder you want to scan
output_csv = "D:\\file_details.csv"  # Output CSV file name

# List to hold file information
file_info_list = []

# Walk through the directory
for root, dirs, files in os.walk(drive_path):
    for file in files:
        file_path = os.path.join(root, file)
        try:
            file_size = os.path.getsize(file_path)  # Get file size in bytes
            file_info_list.append({
                "FileName": file,
                "FilePath": file_path,
                "FileSize": file_size
            })
        except (OSError, FileNotFoundError) as e:
            print(f"Error accessing {file_path}: {e}")

# Create a DataFrame and save to CSV
file_info_df = pd.DataFrame(file_info_list)
file_info_df.to_csv(output_csv, index=False)
print(f"File details saved to {output_csv}")
```

```
File details saved to D:\file_details.csv
```

In [56]:
```python
def sort_names(x):
    return x[0]

names = [('John', 'Kelly'), ('Chris', 'Rock'), ('Will', 'Smith')]
sorted_names = sorted(names, key=sort_names)
print(sorted_names)
print(names)
```

```
[('Chris', 'Rock'), ('John', 'Kelly'), ('Will', 'Smith')]
[('John', 'Kelly'), ('Chris', 'Rock'), ('Will', 'Smith')]
```

In [18]:
```python
list = [1, 3, 6, 7]
print(list[0])
```

```
1
```

In [9]:
```python
import this
print(this)
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
<module 'this' from 'C:\\anaconda\\Lib\\this.py'>
```

In [12]:
```
for i in range(0,5):
 print(i)
```

```
0
1
2
3
4
```

In [26]:
```
# Generate and print the 3rd element for each iteration
for i in range(0, 5):
    generated_lst = [j for j in range(5)]  # Renamed variable to 'generated_lst'
    print(generated_lst[2])  # This will print the 3rd element, which is '2'
```

```
2
2
2
2
2
```

In [72]:
```
import pandas as pd

# Data setup
data = {
    'c1': ['ABCDEF', 'GHIJKL', 'MNOPQR', 'STUVWX'],
    'c2': [1, 2, 3, 1],
    'c3': [100, 200, 300, 400],
    'c4': [5, 15, 10, 20],
    'c5': [30, 25, 35, 20],
    'date_column': ['2024-11-04 12:00:00', '2024-11-04 15:00:00', '2024-11-03 08
}
df = pd.DataFrame(data)
df['date_column'] = pd.to_datetime(df['date_column'])
print()
```

In [94]:
```
# Extract first 2 characters of 'c1' column
df['c1_first_2'] = df['c1'].str[0:2]
```

```
df['c1_first_2_otherway'] = df['c1'].str[:2]
df['c1_last_2'] = df['c1'].str[-2:]
df['c1_last_3'] = df['c1'].str[-3:]

# Display the modified DataFrame
#print(df['c1_first_two'])
print(df[['c1', 'c1_first_2','c1_last_2','c1_last_3','c1_first_2_otherway']])
#print(df)
```

```
       c1 c1_first_2 c1_last_2 c1_last_3 c1_first_2_otherway
0  ABCDEF         AB        EF       DEF                  AB
1  GHIJKL         GH        KL       JKL                  GH
2  MNOPQR         MN        QR       PQR                  MN
3  STUVWX         ST        WX       VWX                  ST
```

In [62]:
```
# Extract the date part as a string
df['date_only'] = df['date_column'].dt.strftime('%Y-%m-%d')
print(df[['date_column', 'date_only']])
```

```
          date_column   date_only
0 2024-11-04 12:00:00  2024-11-04
1 2024-11-04 15:00:00  2024-11-04
2 2024-11-03 08:00:00  2024-11-03
3 2024-11-03 20:00:00  2024-11-03
```

In [47]:
```
# Extract the time part
df['time_only'] = df['date_column'].dt.strftime('%H:%M:%S')
print(df[['date_column', 'time_only']])
```

```
          date_column time_only
0 2024-11-04 12:00:00  12:00:00
1 2024-11-04 15:00:00  15:00:00
2 2024-11-03 08:00:00  08:00:00
3 2024-11-03 20:00:00  20:00:00
```

In [49]:
```
# Get only the date part
df['only_date'] = df['date_column'].dt.date
print(df[['date_column', 'only_date']])
```

```
          date_column   only_date
0 2024-11-04 12:00:00  2024-11-04
1 2024-11-04 15:00:00  2024-11-04
2 2024-11-03 08:00:00  2024-11-03
3 2024-11-03 20:00:00  2024-11-03
```

In [50]:
```
# Extract year, month, and day
df['year'] = df['date_column'].dt.year
df['month'] = df['date_column'].dt.month
df['day'] = df['date_column'].dt.day
print(df[['date_column', 'year', 'month', 'day']])
```

```
          date_column  year  month  day
0 2024-11-04 12:00:00  2024     11    4
1 2024-11-04 15:00:00  2024     11    4
2 2024-11-03 08:00:00  2024     11    3
3 2024-11-03 20:00:00  2024     11    3
```

In [51]:
```
# Get current date and time
from datetime import datetime, timedelta

current_date = datetime.now().date()
current_time = datetime.now()
```

```
print(f"Today's Date: {current_date}")
print(f"Current Time: {current_time}")
```

```
Today's Date: 2024-11-04
Current Time: 2024-11-04 23:38:16.277807
```

In [52]:
```
# Adding and subtracting days
df['add_10_days'] = df['date_column'] + pd.Timedelta(days=10)
df['remove_10_days'] = df['date_column'] - pd.Timedelta(days=10)
print(df[['date_column', 'add_10_days', 'remove_10_days']])
```

```
          date_column         add_10_days      remove_10_days
0 2024-11-04 12:00:00 2024-11-14 12:00:00 2024-10-25 12:00:00
1 2024-11-04 15:00:00 2024-11-14 15:00:00 2024-10-25 15:00:00
2 2024-11-03 08:00:00 2024-11-13 08:00:00 2024-10-24 08:00:00
3 2024-11-03 20:00:00 2024-11-13 20:00:00 2024-10-24 20:00:00
```

In [54]:
```
# Group by 'c1' and calculate count, sum, max, and min
agg_df = df.groupby('c1').agg(
    cnt_c2=('c2', 'count'),
    total_c3=('c3', 'sum'),
    mx_c4=('c4', 'max'),
    mn_c5=('c5', 'min')
).reset_index()
print(agg_df)
```

```
   c1  cnt_c2  total_c3  mx_c4  mn_c5
0  ABC       3       800     20     20
1  DEF       1       200     15     25
```

In [48]:
```
# Use slicing for substring (example string)
example_string = "My Name is Raj"
substring = example_string[11:14]  # 'Raj' (11 is starting index)
print(substring)
```

```
Raj
```

In [113…
```
from datetime import datetime
# Cast string to Date and Date Time
date_string = "2024-01-05 13:42:25"
casted_datetime  = datetime.strptime(date_string, '%Y-%m-%d %H:%M:%S')
casted_2_date  = casted_datetime.date()

month_number = casted_datetime.strftime('%m')          # e.g., '11'
month_number_singledigit = casted_datetime.strftime('%#m') #single digit 1 in ca
month_name = casted_datetime.strftime('%B')            # e.g., 'November'
month_short_name = casted_datetime.strftime('%b')      # e.g., 'Nov'

# Print the results
print("Month Number:", month_number)
print("Month Number Single Digit:", month_number_singledigit)
print("Month Name:", month_name)
print("Month Short Name:", month_short_name)
print("date:", casted_2_date)
print("datetime:", casted_datetime)
```

```
Month Number: 01
Month Number Single Digit: 1
Month Name: January
Month Short Name: Jan
date: 2024-01-05
datetime: 2024-01-05 13:42:25
```

In [111…
```
# Single line statement
p1 = 10 + 20
p1
```

Out[111… 30

In [115…
```
# Single line statement
p2 = ['a' , 'b' , 'c' , 'd']

p2
```

Out[115… ['a', 'b', 'c', 'd']

In [116…
```
# Multiple line statement
p1 = 20 + 30 \
 + 40 + 50 +\
 + 70 + 80
p1
```

Out[116… 290

In [117…
```
# Multiple line statement
p2 = ['a' ,
 'b' ,
 'c' ,
 'd'
 ]
p2
```

Out[117… ['a', 'b', 'c', 'd']

In [118…
```
p = 10
if p == 10:
 print ('P is equal to 10') # correct indentation
```

```
P is equal to 10
```

In [119…
```
for i in range(0,5):
 print(i)
```

```
0
1
2
3
4
```

In [120…
```
j=20
for i in range(0,5):
 print(i) # inside the for loop
print(j) # outside the for loop
```

```
0
1
2
3
4
20
```

In [122…
```
p=1
hex(id(p)) # Memory address of the variable
```

Out[122…
```
'0x7ffafa7c59b8'
```

In [132…
```
# Iteration
mystr1 = "Hello Everyone"
for i in mystr1:
    print(i)
```

```
H
e
l
l
o

E
v
e
r
y
o
n
e
```

In [133…
```
for i in enumerate(mystr1):
 print(i)
```

```
(0, 'H')
(1, 'e')
(2, 'l')
(3, 'l')
(4, 'o')
(5, ' ')
(6, 'E')
(7, 'v')
(8, 'e')
(9, 'r')
(10, 'y')
(11, 'o')
(12, 'n')
(13, 'e')
```

In [166…
```
A={1,2,3}
B={3,4,5}
C={4,5,6}
#A.union(B) #A union B ---->same as A|B --->{1, 2, 3, 4, 5}
#A.union(B,C) #A Union B Union C --> {1, 2, 3, 4, 5, 6}
#A-B # {1, 2} # same A.difference(B)  # {1, 2}
#A.difference(B)  # {1, 2}
#A&B # intersection -->{3}
#A.union(B,C)
#A.update(B,C)  #A.union(B,C)  print(A) A union b union c ===> A.update(B,C) pri
#A.intersection(B)  #{3} ===>A.intersection_update(B) print(A)
# sum(A)  # == 1+2+3=6
# max(A) # 3
# min(A) #1
count_A = len(A)
print("Count of A:", count_A)
```

```
Count of A: 3
```

In [172…

```
a = 10
b = 4
x = 'Vijendar'
y = 'Reddy'
# Addition
c = a + b
print('Addition of {} and {} will give : {}\n'.format(a,b,c))
#Concatenate string using plus operator
z = x+y
print ('Concatenate string \'x\' and \'y\' using \'+\' operaotr : {}\n'.format(z

# Subtraction
c = a - b
print('Subtracting {} from {} will give : {}\n'.format(b,a,c))

# Multiplication
c = a * b
print('Multiplying {} and {} will give : {}\n'.format(a,b,c))

# Division
c = a / b
print('Dividing {} by {} will give : {}\n'.format(a,b,c))

# Modulo of both number
c = a % b
print('Modulo of {} , {} will give : {}\n'.format(a,b,c))

# Power
c = a ** b
print('{} raised to the power {} will give : {}\n'.format(a,b,c))
# Division(floor)
c = a // b
print('Floor division of {} by {} will give : {}\n'.format(a,b,c))
```

```
Addition of 10 and 4 will give : 14

Concatenate string 'x' and 'y' using '+' operaotr : VijendarReddy

Subtracting 4 from 10 will give : 6

Multiplying 10 and 4 will give : 40

Dividing 10 by 4 will give : 2.5

Modulo of 10 , 4 will give : 2

10 raised to the power 4 will give : 10000

Floor division of 10 by 4 will give : 2
```