lab 9

October 6, 2023

[1]:

```
class py_solution:
    def int_to_Roman(self, num):
        val = [
            1000, 900, 500, 400,
            100, 90, 50, 40,
            10, 9, 5, 4,
            1
            ]
        syb = [
            "M", "CM", "D", "CD",
            "C", "XC", "L", "XL",
            "X", "IX", "V", "IV",
            "I"
            ]
        roman_num = ''
        i = 0
        while  num > 0:
            for _ in range(num // val[i]):
                roman_num += syb[i]
                num -= val[i]
            i += 1
        return roman_num


print(py_solution().int_to_Roman(1))
print(py_solution().int_to_Roman(4000))
```

```
I
MMMM
```

[2]:

```
# Python program to get all possible unique subsets from a set of distinct␣
 ↪integers
class py_solution:
    def sub_sets(self, sset):
        return self.subsetsRecur([], sorted(sset))

    def subsetsRecur(self, current, sset):
        if sset:
            return self.subsetsRecur(current, sset[1:]) + self.
 ↪subsetsRecur(current + [sset[0]], sset[1:])
        return [current]

print(py_solution().sub_sets([4,5,6]))
```

```
[[], [6], [5], [5, 6], [4], [4, 6], [4, 5], [4, 5, 6]]
```

[3]:

```
#Python class to find a pair of elements (indices of the two numbers) from a␣
 ↪given array whose sum equals a specific target number.
class py_solution:
  def twoSum(self, nums, target):
       lookup = {}
       for i, num in enumerate(nums):
           if target - num in lookup:
               return (lookup[target - num], i )
           lookup[num] = i
print("index1=%d, index2=%d" % py_solution().twoSum((10,20,10,40,50,60,70),50))
```

```
index1=2, index2=3
```

[4]:

```
#Python class to reverse a string
class py_solution:
    def reverse_words(self, s):
        return ' '.join(reversed(s.split()))


print(py_solution().reverse_words('hello .py'))
```

```
.py hello
```

[5]:

```
#Python program to create a Balanced Binary Search Tree (BST) using an array of␣
 ↪elements where array elements are sorted in ascending order
class TreeNode(object):
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

def sorted_array_to_bst(nums):

    if not nums:
        return None
    mid_val = len(nums)//2
    node = TreeNode(nums[mid_val])
    node.left = sorted_array_to_bst(nums[:mid_val])
    node.right = sorted_array_to_bst(nums[mid_val+1:])
    return node
```

```
def preOrder(node):
    if not node:
        return
    print(node.val)
    preOrder(node.left)
    preOrder(node.right)

result = sorted_array_to_bst([1, 2, 3, 4, 5, 6, 7])
preOrder(result)
```

```
4
2
1
3
6
5
7
```

[6]:

```
#Python program to find the kth smallest element in a given binary search tree
class TreeNode(object):
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None

def kth_smallest(root, k):
    stack = []
    while root or stack:
        while root:
            stack.append(root)
            root = root.left
        root = stack.pop()
        k -= 1
        if k == 0:
            break
        root = root.right
    return root.val

root = TreeNode(8)
root.left = TreeNode(5)
root.right = TreeNode(14)
root.left.left = TreeNode(4)
root.left.right = TreeNode(6)
root.left.right.left = TreeNode(8)
root.left.right.right = TreeNode(7)
root.right.right = TreeNode(24)
root.right.right.left = TreeNode(22)

print(kth_smallest(root, 2))
print(kth_smallest(root, 3))
```

```
5
8
```

[7]:

```
#Python program to locate the right insertion point for a specified value in sorted order.
import bisect
def index(a, x):
    i = bisect.bisect_right(a, x)
    return i

a = [1,2,4,7]
print(index(a, 6))
print(index(a, 3))
```

```
3
2
```

[8]:

```
#Python program to find the index position of the last occurrence of a given number in a sorted list using Binary Search (bisect).
from bisect import bisect_right
def BinarySearch(a, x):
    i = bisect_right(a, x)
    if i != len(a)+1 and a[i-1] == x:
        return (i-1)
    else:
        return -1
nums = [1, 2, 3, 4, 8, 8, 10, 12]
x = 8
num_position   = BinarySearch(nums, x)
if num_position == -1:
    print("not presetn!")
else:
    print("Last occurrence of", x, "is present at", num_position)
```

```
Last occurrence of 8 is present at 5
```

[9]:

```
#NumPy program to convert a list of numeric values into a one-dimensional NumPy array
import numpy as np
l = [12.23, 13.32, 100, 36.32]
print("Original List:",l)
a = np.array(l)
print("One-dimensional NumPy array: ",a)
```

```
Original List: [12.23, 13.32, 100, 36.32]
One-dimensional NumPy array:  [ 12.23  13.32 100.    36.32]
```

[10]:

```python
#NumPy program to convert an array to a floating type
import numpy as np
import numpy as np
a = [1, 2, 3, 4]
print("Original array")
print(a)
x = np.asfarray(a)
print("Array converted to a float type:")
print(x)
```

```
Original array
[1, 2, 3, 4]
Array converted to a float type:
[1. 2. 3. 4.]
```

[11]:

```python
#NumPy program to display all the dates for the month of March, 2017
import numpy as np
print("March, 2017")
print(np.arange('2017-03', '2017-04', dtype='datetime64[D]'))
```

```
March, 2017
['2017-03-01' '2017-03-02' '2017-03-03' '2017-03-04' '2017-03-05'
 '2017-03-06' '2017-03-07' '2017-03-08' '2017-03-09' '2017-03-10'
 '2017-03-11' '2017-03-12' '2017-03-13' '2017-03-14' '2017-03-15'
 '2017-03-16' '2017-03-17' '2017-03-18' '2017-03-19' '2017-03-20'
 '2017-03-21' '2017-03-22' '2017-03-23' '2017-03-24' '2017-03-25'
 '2017-03-26' '2017-03-27' '2017-03-28' '2017-03-29' '2017-03-30'
 '2017-03-31']
```

[12]:

```python
#NumPy program to count the number of days of specific month
import numpy as np
print("Number of days, February, 2016: ")
print(np.datetime64('2016-03-01') - np.datetime64('2016-02-01'))
print("Number of days, February, 2017: ")
print(np.datetime64('2017-03-01') - np.datetime64('2017-02-01'))
print("Number of days, February, 2018: ")
print(np.datetime64('2018-03-01') - np.datetime64('2018-02-01'))
```

```
Number of days, February, 2016:
29 days
Number of days, February, 2017:
28 days
Number of days, February, 2018:
28 days
```

[13]:

```
#NumPy program to find the number of weekdays in March 2017.

#Note: "busday" default of Monday through Friday being valid days.
import numpy as np
print("Number of weekdays in March 2017:")
print(np.busday_count('2017-03', '2017-04'))
```

```
Number of weekdays in March 2017:
23
```

[14]:

```
#NumPy program to compute the cross product of two given vectors
import numpy as np
p = [[1, 0], [0, 1]]
q = [[1, 2], [3, 4]]
print("original matrix:")
print(p)
print(q)
result1 = np.cross(p, q)
result2 = np.cross(q, p)
print("cross product of the said two vectors(p, q):")
print(result1)
print("cross product of the said two vectors(q, p):")
print(result2)
```

```
original matrix:
[[1, 0], [0, 1]]
[[1, 2], [3, 4]]
cross product of the said two vectors(p, q):
[ 2 -3]
cross product of the said two vectors(q, p):
[-2  3]
```

[15]:

```
#NumPy program to compute the eigenvalues and right eigenvectors of a given␣
  ↪square array
import numpy as np
m = np.mat("3 -2;1 0")
print("Original matrix:")
print("a\n", m)
w, v = np.linalg.eig(m)
print( "Eigenvalues of the said matrix",w)
print( "Eigenvectors of the said matrix",v)
```

```
Original matrix:
a
 [[ 3 -2]
 [ 1  0]]
Eigenvalues of the said matrix [2. 1.]
Eigenvectors of the said matrix [[0.89442719 0.70710678]
 [0.4472136  0.70710678]]
```

[16]:

```
#NumPy program to compute the determinant of an array.
import numpy as np
a = np.array([[1,2],[3,4]])
print("Original array:")
print(a)
result =  np.linalg.det(a)
print("Determinant of the said array:")
print(result)
```

```
Original array:
[[1 2]
 [3 4]]
Determinant of the said array:
-2.0000000000000004
```

[17]:

```
#Pandas program to import given excel data (coalpublic2013.xlsx ) into a Pandas␣
 ↪dataframe
import pandas as pd
import numpy as np
df = pd.read_excel('excel file path')
print(df.head)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Input In [17], in <cell line: 4>()
      2 import pandas as pd
      3 import numpy as np
----> 4 df = pd.read_excel('E:\coalpublic2013.xlsx')
      5 print(df.head)

File ~\anaconda3\lib\site-packages\pandas\util\_decorators.py:311, in␣
 ↪deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*args,␣
 ↪**kwargs)
    305 if len(args) > num_allow_args:
    306     warnings.warn(
    307         msg.format(arguments=arguments),
    308         FutureWarning,
    309         stacklevel=stacklevel,
    310     )
--> 311 return func(*args, **kwargs)

File ~\anaconda3\lib\site-packages\pandas\io\excel\_base.py:457, in␣
 ↪read_excel(io, sheet_name, header, names, index_col, usecols, squeeze, dtype,␣
 ↪engine, converters, true_values, false_values, skiprows, nrows, na_values,␣
 ↪keep_default_na, na_filter, verbose, parse_dates, date_parser, thousands,␣
 ↪decimal, comment, skipfooter, convert_float, mangle_dupe_cols, storage_options)
    455 if not isinstance(io, ExcelFile):
    456     should_close = True
--> 457     io = ExcelFile(io, storage_options=storage_options, engine=engine)
```

```
    458 elif engine and engine != io.engine:
    459     raise ValueError(
    460         "Engine should not be specified when passing "
    461         "an ExcelFile - ExcelFile already has the engine set"
    462     )

File ~\anaconda3\lib\site-packages\pandas\io\excel\_base.py:1376, in ExcelFile.
 ↪__init__(self, path_or_buffer, engine, storage_options)
   1374     ext = "xls"
   1375 else:
-> 1376     ext = inspect_excel_format(
   1377         content_or_path=path_or_buffer, storage_options=storage_options
   1378     )
   1379     if ext is None:
   1380         raise ValueError(
   1381             "Excel file format cannot be determined, you must specify "
   1382             "an engine manually."
   1383         )

File ~\anaconda3\lib\site-packages\pandas\io\excel\_base.py:1250, in␣
 ↪inspect_excel_format(content_or_path, storage_options)
   1247 if isinstance(content_or_path, bytes):
   1248     content_or_path = BytesIO(content_or_path)
-> 1250 with get_handle(
   1251     content_or_path, "rb", storage_options=storage_options, is_text=False
   1252 ) as handle:
   1253     stream = handle.handle
   1254     stream.seek(0)

File ~\anaconda3\lib\site-packages\pandas\io\common.py:798, in␣
 ↪get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text,␣
 ↪errors, storage_options)
    789         handle = open(
    790             handle,
    791             ioargs.mode,
   (…)
    794             newline="",
    795         )
    796     else:
    797         # Binary mode
--> 798         handle = open(handle, ioargs.mode)
    799     handles.append(handle)
    801 # Convert BytesIO or file objects passed with an encoding

FileNotFoundError: [Errno 2] No such file or directory: 'E:\\coalpublic2013.xlsx'
```

```
[ ]: #Pandas program to import excel data (coalpublic2013.xlsx ) into a Pandas␣
      ↪dataframe and display the last ten rows
     import pandas as pd
     import numpy as np
     df = pd.read_excel('E:\coalpublic2013.xlsx')
     df.tail(n=10)
```

```
[ ]: #Pandas program to read specific columns from a given excel file
     import pandas as pd
     import numpy as np
     cols = [1, 2, 4]
     df = pd.read_excel('E:\coalpublic2013.xlsx', usecols=cols)
     df
```

```
[ ]: import pandas as pd
     import matplotlib.pyplot as plt
     df = pd.read_csv("alphabet_stock_data.csv")
     start_date = pd.to_datetime('2020-4-1')
     end_date = pd.to_datetime('2020-09-30')
     df['Date'] = pd.to_datetime(df['Date'])
     new_df = (df['Date']>= start_date) & (df['Date']<= end_date)
     df2 = df.loc[new_df]
     plt.figure(figsize=(10,10))
     df2.plot(x='Date', y=['Open', 'Close']);
     plt.suptitle('Opening/Closing stock prices of Alphabet Inc.,\n 01-04-2020 to␣
      ↪30-09-2020', fontsize=12, color='black')
     plt.xlabel("Date",fontsize=12, color='black')
     plt.ylabel("$ price", fontsize=12, color='black')
     plt.show()
```

```
[ ]: import pandas as pd
     import matplotlib.pyplot as plt
     df = pd.read_csv("alphabet_stock_data.csv")
     start_date = pd.to_datetime('2020-4-1')
     end_date = pd.to_datetime('2020-9-30')
     df['Date'] = pd.to_datetime(df['Date'])
     new_df = (df['Date']>= start_date) & (df['Date']<= end_date)
     df1 = df.loc[new_df]
     df2 = df1[['Open','Close','High','Low']]
     #df3 = df2.set_index('Date')
     plt.figure(figsize=(25,25))
     df2.plot.hist(alpha=0.5)
     plt.suptitle('Opening/Closing/High/Low stock prices of Alphabet Inc.,\n From␣
      ↪01-04-2020 to 30-09-2020', fontsize=12, color='blue')
     plt.show()
```

[ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv("alphabet_stock_data.csv")
start_date = pd.to_datetime('2020-4-1')
end_date = pd.to_datetime('2020-4-30')
df['Date'] = pd.to_datetime(df['Date'])
new_df = (df['Date']>= start_date) & (df['Date']<= end_date)
df1 = df.loc[new_df]
df2 = df1[['Open']]
plt.figure(figsize=(15,15))
df2.plot.hist(orientation='horizontal', cumulative=True)
plt.suptitle('Opening stock prices of Alphabet Inc.,\n From 01-04-2020 to␣
 ↪30-04-2020', fontsize=12, color='black')
plt.show()
```

[ ]:

```
import pandas as pd

student_data1 = pd.DataFrame({
        'student_id': ['S1', 'S2', 'S3', 'S4', 'S5'],
         'name': ['Danniella Fenton', 'Ryder Storey', 'Bryce Jensen', 'Ed␣
 ↪Bernal', 'Kwame Morin'],
        'marks': [200, 210, 190, 222, 199]})

student_data2 = pd.DataFrame({
        'student_id': ['S4', 'S5', 'S6', 'S7', 'S8'],
        'name': ['Scarlette Fisher', 'Carla Williamson', 'Dante Morse', 'Kaiser␣
 ↪William', 'Madeeha Preston'],
        'marks': [201, 200, 198, 219, 201]})

print("Original DataFrames:")
print(student_data1)
print("-------------------------------------")
print(student_data2)
print("\nJoin the said two dataframes along rows:")
result_data = pd.concat([student_data1, student_data2])
print(result_data)
```

[ ]:

```
import pandas as pd
student_data1  = pd.DataFrame({
        'student_id': ['S1', 'S2', 'S3', 'S4', 'S5'],
         'name': ['Danniella Fenton', 'Ryder Storey', 'Bryce Jensen', 'Ed␣
 ↪Bernal', 'Kwame Morin'],
        'marks': [200, 210, 190, 222, 199]})

s6 = pd.Series(['S6', 'Scarlette Fisher', 205], index=['student_id', 'name',␣
 ↪'marks'])
```

```
dicts = [{'student_id': 'S6', 'name': 'Scarlette Fisher', 'marks': 203},
         {'student_id': 'S7', 'name': 'Bryce Jensen', 'marks': 207}]

print("Original DataFrames:")
print(student_data1)
print("\nDictionary:")
print(s6)
combined_data =  student_data1.append(dicts, ignore_index=True, sort=False)
print("\nCombined Data:")
print(combined_data)
```

[ ]:

```
import pandas as pd
s1 = pd.Series([0, 1, 2, 3], name='col1')
s2 = pd.Series([0, 1, 2, 3])
s3 = pd.Series([0, 1, 4, 5], name='col3')
df = pd.concat([s1, s2, s3], axis=1, keys=['column1', 'column2', 'column3'])
print(df)
```