```python
# 1. a.    Write a python program to read 2 numbers from the keyboard
and perform the basic arithmetic operations based on the choice. (1-
Add, 2-Subtract, 3-Multiply, 4-Divide)

num1= int(input("ENTER FIRST NUM: "))
num2= int(input("ENTER SECOND NUM: "))

key = int(input("ENTER CHOICE: \n 1. ADD \n 2. SUBTRACT \n 3. MULTIPLY
\n 4. DIVIDE \n"))

if(key==1):
    print("SUM: ", num1+num2)
elif(key==2):
    print("DIFFERENCE: ", num1-num2)
elif(key==3):
    print("PRODUCT: ", num1*num2)
elif(key==4):
    print("QUOTIENT: ", num1/num2)
else:
    print("WRONG OPERAND")
```

```
SUM:  3
```

```python
# 1. b.    Write a python program to create a list of tuples having
first element as the strings and the second element as the length of
the string. Output the list of tuples sorted based on the length of
the string.

strings = ["apple", "banana", "cherry", "date"]
tuples = [(string, len(string)) for string in strings]

tuples.sort(key = lambda x:x[1])
print(tuples)
```

```
[('date', 4), ('apple', 5), ('banana', 6), ('cherry', 6)]
```

```python
# 2. a.    Write a python program to display all the prime numbers in
the given range

def prime_check(n):
  if n>1:
    for i in range(2, n//2+1):
      if n%i==0:
        return False
    return True
  else:
    return False


n1 = int(input("Enter the start of range: "))
n2 = int(input("Enter the end of range: "))
```

```python
print(f"Prime No's in range {n1} to {n2} are: ")
for num in range(n1,n2+1):
  prime = prime_check(num)
  if prime:
    print(num, end=", ")
```

```
Enter the start of range: 2
Enter the end of range: 3
Prime No's in range 2 to 3 are:
2,
```

```python
# 2. b.    Write a python program to create a list with all the subject names of the 4th semester and perform the following operations.
# •   Display the list using for loop.
# •   Display 2nd and 5th element of the list.
# •   Display first 4 elements of the list using the range of indexes.
# •   Display last 4 elements of the list using the range of negative indexes.
# •   Display if "Python Programming Lab" is available in the List or not.
# •   Demonstrate the working of append () and insert () function.
# •   Demonstrate the working of remove() and pop() function.

subjs = ["DAA", "DCN", "MCIoT", "Maths", "FAFL", "PyLab", "DCNLab", "DAALab"]

# •   Display the list using for loop.
for subs in subjs:
  print(subs)

# Display 2nd and 5th element of the list.
print(subjs[1])
print(subjs[4])

# Display first 4 elements of the list using the range of indexes.
print(subjs[:4])

# Display last 4 elements of the list using the range of negative indexes.
print(subjs[-4:])

# Display if "Python Programming Lab" is available in the List or not.
if "PyLab" in subjs:
  print("Yes")

# Demonstrate the working of append () and insert () function.
subjs.append("IntraInternship")
print(subjs)
```

```
subjs.insert(3, "WrongSubject")
print(subjs)

# Demonstrate the working of remove() and pop() function.
subjs.pop()
print(subjs)
subjs.remove("WrongSubject")
print(subjs)
```

```
DAA
DCN
MCIoT
Maths
FAFL
PyLab
DCNLab
DAALab
DCN
Maths
['DAA', 'DCN', 'MCIoT', 'Maths']
['FAFL', 'PyLab', 'DCNLab', 'DAALab']
Yes
['DAA', 'DCN', 'MCIoT', 'Maths', 'FAFL', 'PyLab', 'DCNLab', 'DAALab',
'IntraInternship']
['DAA', 'DCN', 'MCIoT', 'WrongSubject', 'Maths', 'FAFL', 'PyLab',
'DCNLab', 'DAALab', 'IntraInternship']
['DAA', 'DCN', 'MCIoT', 'WrongSubject', 'Maths', 'FAFL', 'PyLab',
'DCNLab', 'DAALab']
['DAA', 'DCN', 'MCIoT', 'Maths', 'FAFL', 'PyLab', 'DCNLab', 'DAALab']
```

```
# 3. a.    Create a dictionary for words and their meanings. Write
functions to add a new entry (word: meaning), search for a particular
word and retrieve meaning, given meaning find words with same meaning,
remove an entry, display all words sorted alphabetically. [Program
must be menu driven].

word_meaning_dict = {}

def add_entry(word, meaning):
    if word in word_meaning_dict:
        print("Word already exists in the dictionary.")
    else:
        word_meaning_dict[word] = meaning
        print("Entry added successfully.")

def search_word(word):
    if word in word_meaning_dict:
        print("Meaning of", word + ":", word_meaning_dict[word])
    else:
        print("Word not found in the dictionary.")
```

```
def find_word_by_meaning(meaning):
    words = [word for word, mean in word_meaning_dict.items() if mean
== meaning]
    if words:
        print("Words with the meaning", meaning + ":", ",
".join(words))
    else:
        print("No words found with the specified meaning.")

def remove_entry(word):
    if word in word_meaning_dict:
        del word_meaning_dict[word]
        print("Entry removed successfully.")
    else:
        print("Word not found in the dictionary.")

def display_sorted_words():
    sorted_words = sorted(word_meaning_dict.keys())
    print("Words in the dictionary (sorted alphabetically):")
    for word in sorted_words:
        print(word + ":", word_meaning_dict[word])

def display_menu():
    print("\nMenu:")
    print("1) Add a new word as entry")
    print("2) Search a word")
    print("3) Find words with same meaning")
    print("4) Remove an entry")
    print("5) Display")
#     print("6) Exit")

conti = "y"

while conti=="y":
    display_menu()
    choice = input("Enter your choice: ")

    if choice == '1':
        word = input("Enter the word: ")
        meaning = input("Enter the meaning: ")
        add_entry(word, meaning)
    elif choice == '2':
        word = input("Enter the word to search: ")
        search_word(word)
    elif choice == '3':
        meaning = input("Enter the meaning to search: ")
        find_word_by_meaning(meaning)
    elif choice == '4':
        word = input("Enter the word to remove: ")
        remove_entry(word)
    elif choice == '5':
        display_sorted_words()
    else:
        print("Invalid choice. Please enter a valid option.")

    conti = input("Continue?  (y)/(n): ")

    if conti == "n":
        print("Exiting...")
```

```
Menu:
1) Add a new word as entry
2) Search a word
3) Find words with same meaning
4) Remove an entry
5) Display
Enter your choice: 6
Invalid choice. Please enter a valid option.
Continue?  (y)/(n): n
Exiting...
```

```
# 3. b.    Write a python program to perform the following operations
using user defined functions
# •   Display the maximum and minimum number from the array.
# •   Display the second largest number from the array without sorting

def inputArray():
  n = int(input("Enter the number of elements: "))
  arr = []
  for i in range(n):
    print(f"Enter element {i+1}: ", end=' ')
    ele = int(input())
    arr.append(ele)
  return arr

arr = inputArray()
print(arr)

#    Display the maximum and minimum number from the array.
print(f"Maxi element in arr is : {max(arr)}")
print(f"Mini element in arr is : {min(arr)}")

#    Display the second largest number from the array without sorting

second_max = max([i for i in arr if i!=max(arr)])

num1 = max(arr)
arr.remove(num1)
num2 = max(arr)
```

```
print(f"Second largest element in arr is : {num2}")


# or

def second_largest(arr):
  larg = -float('inf')
  second_larg = -float('inf')
  for num in arr:
    if num > larg:
      second_larg = larg
      larg = num
    elif num > second_larg and num != larg:
      second_larg = num

  return second_larg

second_largest_num = second_largest(arr)


Enter the number of elements: 5
Enter element 1:  5
Enter element 2:  7
Enter element 3:  9
Enter element 4:  1
Enter element 5:  3
[5, 7, 9, 1, 3]
Maxi element in arr is : 9
Mini element in arr is : 1
Second largest element in arr is : 7
5

# 4. a.    Write a python program to initialize a dictionary of
usernames and passwords
# associated with it.passwd={‘rahul’: ‘genius’, ‘kumar’: ‘smart’,
‘ankita’: ‘intelligent’} perform the following functions:
# •   To print all the items in the dictionary.
# •   To print all the keys in the dictionary.
# •   To print all the values in the dictionary.
# •   To get the passwords of users. For example, passwd[‘rahul’]=
genius
# •   Change the password of a particular user. For example,
passwd[‘ankita’]=‘brilliant’

users = {
    "rahul": "xyz",
    "sanchit": "hawyeah",
    "saurabh": "nikhil",
}
```

```python
# To print all the items in the dictionary.
print("All items in the dictionary:")
for key, value in users.items():
    print(key, ":", value)

# To print all the keys in the dictionary.
print("\nAll keys in the dictionary:")
for key in users.keys():
    print(key)

# To print all the values in the dictionary.
print("\nAll values in the dictionary:")
for value in users.values():
    print(value)

print(users["sanchit"])

users["sanchit"] = "I'm wet"

print("\n",users["sanchit"])
```

```
All items in the dictionary:
rahul : xyz
sanchit : hawyeah
saurabh : nikhil

All keys in the dictionary:
rahul
sanchit
saurabh

All values in the dictionary:
xyz
hawyeah
nikhil
hawyeah

 I'm wet
```

```python
# 4 b.Develop a python program to count all the occurrences of vowels,
consonants and digits from the given text using Regular expressions.

import re

strr = input("Enter the text: ")

vow = re.findall('[aeiouAEIOU]', strr)
digits = re.findall("[0-9]", strr)
cons = re.findall("[bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ]",
strr)
```

```
print(f"Vowels : {vow}")
print(f"Consonants : {cons}")
print(f"Digits : {digits}")
```

```
Vowels : ['a', 'e', 'i']
Consonants : ['b', 'c', 'd', 'f', 'g', 'h', 'j', 'k']
Digits : ['1', '2', '3', '4']
```

```
# 5. a.    Write a python program to create a tuple and perform the
following operations
# •   Adding an items
# •   Displaying the length of the tuple
# •   Checking for an item in the tuple
# •   Accessing an items

tup = (1, 2, 3, 4, 5)

#     Adding an items
tup = tup + (6,)
print(tup)

print("Length of tuple: ",len(tup))
print(tup)

if 5 in tup:
  print("5 is in the tuple")
else:
  print("5 is not in the tuple")

print("3rd element is", tup[3])
```

```
(1, 2, 3, 4, 5, 6)
Length of tuple:  6
(1, 2, 3, 4, 5, 6)
5 is in the tuple
3rd element is 4
```

```
# 5. b.    Write a python program to create a text file and ask the
user to enter 5-6 lines of text. Display the longest and the shortest
word from the file. Display the length of these words.

myfile =  open("new.txt", "w")

for i in range(5):
    line = input(f"Enter line: {i+1}: ")
    myfile.write(line + "\n")

myfile.close()

myfile = open("new.txt", "r")
```

```
longest_word = ""
shortest_word = None
# Read the contents of the file
contents = myfile.read()
# Split the contents of the file into words
words = contents.split()

for word in words:
    if len(word) > len(longest_word):
        longest_word = word

    if shortest_word is None or len(word) < len(shortest_word):
        shortest_word = word

myfile.close()

print("\nLongest word in the file:")
print(f"{longest_word} (Length: {len(longest_word)})")
print("\nShortest word in the file:")
print(f"{shortest_word} (Length: {len(shortest_word)})")
```

```
Enter line: 1: a
Enter line: 2: asas
Enter line: 3: asasasa
Enter line: 4:
Enter line: 5: asas

Longest word in the file:
asasasa (Length: 7)

Shortest word in the file:
a (Length: 1)
```

```
# 6. a.    Write a python function binary Search () to read a sorted
array and search for the key element. Display the appropriate
messages.

def binsearch(s, e, key, arr):
    if(s>e):
        print("Element not found")
        return
    mid = s + (e-s)//2
    if(arr[mid] == key):
        print(f"{key} found at position {mid}")
        return
    elif(arr[mid] > key):
        binsearch(s, mid-1, key, arr)
    else:
        binsearch(mid+1, e, key, arr)
```

```
arr =[1,10,3,4,5,6,6,7,8,9]
arr.sort()
print(arr)
binsearch(0, len(arr), 10, arr)
```

```
[1, 3, 4, 5, 6, 6, 7, 8, 9, 10]
10 found at position 9
```

```
# 6. b.    Write a python program to simulate saving account 
processing in a bank using constructors. Create Deposit and Withdraw 
with other member function and Check for Validation while withdrawing 
the amount. Raise the appropriate exceptions when depositing and 
withdrawing an incorrect amount. Display appropriate messages.

class Bank:
    def __init__(self, account_number, account_holder_name, 
initial_balance=0):
        self.account_number = account_number
        self.account_holder_name = account_holder_name
        self.balance = initial_balance

    def deposit(self, amount):
        if amount <= 0:
            raise ValueError("Deposit amount should be greater than 
zero")
        self.balance += amount
        print(f"Deposit of {amount} successful. New balance is 
{self.balance}")

    def withdraw(self, amount):
        if amount <= 0:
            raise ValueError("Withdrawal amount should be greater than 
zero")
        if amount > self.balance:
            raise ValueError("Insufficient balance")
        self.balance -= amount
        print(f"Withdrawal of {amount} successful. New balance is 
{self.balance}")

    def check_balance(self):
        print(f"Current balance: {self.balance}")


account = Bank(12345, "John Doe", 1000)

account.check_balance()
account.deposit(500)
account.check_balance()
account.withdraw(200)
account.check_balance()
account.withdraw(2000)

Current balance: 1000
Deposit of 500 successful. New balance is 1500
Current balance: 1500
Withdrawal of 200 successful. New balance is 1300
Current balance: 1300

-----------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
<ipython-input-49-6a41286c38e0> in <cell line: 34>()
     32 account.withdraw(200)
     33 account.check_balance()
---> 34 account.withdraw(2000)

<ipython-input-49-6a41286c38e0> in withdraw(self, amount)
     17             raise ValueError("Withdrawal amount should be
greater than zero")
     18         if amount > self.balance:
---> 19             raise ValueError("Insufficient balance")
     20         self.balance -= amount
     21         print(f"Withdrawal of {amount} successful. New balance
is {self.balance}")

ValueError: Insufficient balance

# 7. a.    Develop a python program to create two classes called as
Stack and Queue. Provide the necessary data members and methods to
display the operations that can be performed on stacks and queues.
Test for all type of conditions

class Stack:
    def __init__(self):
        self.stack = []

    def push(self, item):
        self.stack.append(item)
        print(f"Pushed {item} to stack")

    def pop(self):
        if self.stack:
            print("Stack is empty, cannot pop")
            return None
        return self.stack.pop()

    def peek(self):
        if self.is_empty():
            print("Stack is empty, nothing to peek")
```

```python
            return None
        return self.stack[-1]

    def is_empty(self):
        return len(self.stack) == 0

    def display(self):
        if self.is_empty():
            print("Stack is empty")
        else:
            print("Stack contents:", self.stack)

class Queue:
    def __init__(self):
        self.queue = []

    def enqueue(self, item):
        self.queue.append(item)
        print(f"Enqueued {item} to queue")

    def dequeue(self):
        if self.is_empty():
            print("Queue is empty, cannot dequeue")
            return None
        return self.queue.pop(0)

    def front(self):
        if self.is_empty():
            print("Queue is empty, nothing at front")
            return None
        return self.queue[0]

    def is_empty(self):
        return len(self.queue) == 0

    def display(self):
        if self.is_empty():
            print("Queue is empty")
        else:
            print("Queue contents:", self.queue)

# Test cases
if __name__ == "__main__":
    # Testing Stack
    print("Testing Stack Operations:")
    stack = Stack()
    stack.push(1)
    stack.push(2)
    stack.push(3)
    stack.display()
    print("Popped item:", stack.pop())
    print("Peek item:", stack.peek())
    stack.display()
    print("Popped item:", stack.pop())
    print("Popped item:", stack.pop())
    print("Popped item:", stack.pop())  # Stack is empty
    stack.display()

    # Testing Queue
    print("\nTesting Queue Operations:")
    queue = Queue()
    queue.enqueue(1)
    queue.enqueue(2)
    queue.enqueue(3)
    queue.display()
    print("Dequeued item:", queue.dequeue())
    print("Front item:", queue.front())
    queue.display()
    print("Dequeued item:", queue.dequeue())
    print("Dequeued item:", queue.dequeue())
    print("Dequeued item:", queue.dequeue())  # Queue is empty
    queue.display()
```

```
Testing Stack Operations:
Pushed 1 to stack
Pushed 2 to stack
Pushed 3 to stack
Stack contents: [1, 2, 3]
Stack is empty, cannot pop
Popped item: None
Peek item: 3
Stack contents: [1, 2, 3]
Stack is empty, cannot pop
Popped item: None
Stack is empty, cannot pop
Popped item: None
Stack is empty, cannot pop
Popped item: None
Stack contents: [1, 2, 3]

Testing Queue Operations:
Enqueued 1 to queue
Enqueued 2 to queue
Enqueued 3 to queue
Queue contents: [1, 2, 3]
Dequeued item: 1
Front item: 2
Queue contents: [2, 3]
Dequeued item: 2
Dequeued item: 3
```

```
Queue is empty, cannot dequeue
Dequeued item: None
Queue is empty

# 7. b.    Write a python program to utilize NumPy and perform the
following operations.
# •   Read and display a 2D Array.
# •   Display the array elements in the reverse order.
# •   Display all the elements of principal diagonal elements.
# •   Sort the 2D array in ascending and descending order

import numpy as np

#    Read and display a 2D Array.
arr2d = []
rows = int(input("Enter no of rows: "))
cols = int(input("Enter no of cols: "))

for i in range(rows):
  arr = []
  print(f"Enter {i+1} row elements: ")
  # numbers = list(map(int, input("Enter ", cols, " integers separated
by spaces: ").split()))
  # arr.append(numbers)
  for j in range(cols):
    num = int(input())
    arr.append(num)
  arr2d.append(arr)

arr2d = np.array(arr2d)
print("\n", arr2d , "\n")

# Display the array elements in the reverse order.
print("2D Array in Reverse Order (without loop):")
print(arr2d[::-1])
print()

print("2D Array in Reverse Order (with loop):")
for row in reversed(arr2d):
    print(row)
print()

#    Display all the elements of principal diagonal elements.
diagonal_elements = np.diag(arr2d)
print("Principal diagonal elements:", diagonal_elements)
print()

# Sort the 2D array in ascending and descending order
arr_sorted_asc = np.sort(arr2d, axis=None)
arr_sorted_desc = np.sort(arr2d, axis=None)[::-1]
```

```
print("Sorted array in ascending order:")
print(arr_sorted_asc)
print("Sorted array in descending order:")
print(arr_sorted_desc)
```

```
Enter 1 row elements:
Enter 2 row elements:
Enter 3 row elements:

 [[3 4 5]
 [2 1 5]
 [6 7 8]]

2D Array in Reverse Order (without loop):
[[6 7 8]
 [2 1 5]
 [3 4 5]]

2D Array in Reverse Order (with loop):
[6 7 8]
[2 1 5]
[3 4 5]

Principal diagonal elements: [3 1 8]

Sorted array in ascending order:
[1 2 3 4 5 5 6 7 8]
Sorted array in descending order:
[8 7 6 5 5 4 3 2 1]
```

```
# 8. a.    Develop a python program to read 20 random numbers. Display all the odd numbers from this list which are of length 2 and 4.

import random

arr = [random.randint(0, 100) for i in range(20)]
for num in arr:
  if num%2 == 1:
    num_str = str(num)  # Convert the integer to a string to check the number of digits
    if len(num_str) == 2:
      print("Len 2: ", num)
    elif len(num_str) == 4:
      print("Len 4: ", num)
```

```
Len 2:  67
Len 2:  77
Len 2:  57
Len 2:  45
Len 2:  99
Len 2:  91
Len 2:  73
Len 2:  87
Len 2:  63
Len 2:  65

# 8. b.    Develop a python program to create a text file and ask the
user to enter 5-6 lines of text. Count the number of upper case, lower
case and digits in the file. Display the details of the file.

myfile =  open("file2.txt", "w")

for i in range(5):
    line = input(f"Enter line: {i+1}: ")
    myfile.write(line + "\n")

myfile.close()

myfile = open("new.txt", "r")

upper_count = 0
lower_count = 0
digit_count = 0

for line in myfile:
    for char in line:
        if char.isupper():
            upper_count += 1
        elif char.islower():
            lower_count += 1
        elif char.isdigit():
            digit_count += 1

myfile.close()

print(f"Uppercase Count: {upper_count}, Lowercase Count:
{lower_count}, Digit Count: {digit_count}")

Enter line: 1: asasas
Enter line: 2: d
Enter line: 3: ghdfhdhd
Enter line: 4: ""
Enter line: 5: "
Uppercase Count: 0, Lowercase Count: 16, Digit Count: 0

# 9. a.    Develop a python program read a dataset and perform the
following using Pandas
# •   Visualize the dataset using plot ().
# •   Draw the Scatter plot for the dataset on any column.
# •   Display the scatter plot with different colors.
# •   Draw the Histogram for the dataset on any column.

import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv('iris.csv')

# Line plot of the dataset
plt.figure(figsize=(10, 6))
df.plot(kind='line', subplots=True, layout=(5, 5), figsize=(15, 10),
title="Dataset Visualization")
plt.tight_layout()
plt.show()

# Scatter plot for sepal length vs. sepal width with colors
df.plot(kind='scatter', x='sepal.length', y='sepal.width',
c='sepal.width', colormap='viridis', title='Scatter Plot with Color
Variation')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.show()

# Histogram for sepal length

df['sepal.length'].plot(kind='hist', bins=20, color='green',
edgecolor='black')
plt.xlabel('Sepal Length')
plt.title('Histogram of Sepal Length')
plt.show()

# # Histogram for sepal length
# plt.scatter(df.index, df['sepal.length'], c='black')
# plt.xlabel('Index')

# df.plot(kind='hist')
# plt.show()
```

```
----------------------------------------------------------------------
-----
FileNotFoundError                         Traceback (most recent call
last)
Cell In[6], line 10
      7 import matplotlib.pyplot as plt
      8 import pandas as pd
---> 10 df = pd.read_csv('iris.csv')
     12 # Line plot of the dataset
     13 plt.figure(figsize=(10, 6))

File c:\Python312\Lib\site-packages\pandas\io\parsers\readers.py:1026,
in read_csv(filepath_or_buffer, sep, delimiter, header, names,
index_col, usecols, dtype, engine, converters, true_values,
false_values, skipinitialspace, skiprows, skipfooter, nrows,
na_values, keep_default_na, na_filter, verbose, skip_blank_lines,
parse_dates, infer_datetime_format, keep_date_col, date_parser,
date_format, dayfirst, cache_dates, iterator, chunksize, compression,
thousands, decimal, lineterminator, quotechar, quoting, doublequote,
escapechar, comment, encoding, encoding_errors, dialect, on_bad_lines,
delim_whitespace, low_memory, memory_map, float_precision,
storage_options, dtype_backend)
   1013 kwds_defaults = _refine_defaults_read(
   1014     dialect,
   1015     delimiter,
   (...)
   1022     dtype_backend=dtype_backend,
   1023 )
   1024 kwds.update(kwds_defaults)
-> 1026 return _read(filepath_or_buffer, kwds)

File c:\Python312\Lib\site-packages\pandas\io\parsers\readers.py:620,
in _read(filepath_or_buffer, kwds)
    617 _validate_names(kwds.get("names", None))
    619 # Create the parser.
--> 620 parser = TextFileReader(filepath_or_buffer, **kwds)
    622 if chunksize or iterator:
    623     return parser

File c:\Python312\Lib\site-packages\pandas\io\parsers\readers.py:1620,
in TextFileReader.__init__(self, f, engine, **kwds)
   1617     self.options["has_index_names"] = kwds["has_index_names"]
   1619 self.handles: IOHandles | None = None
-> 1620 self._engine = self._make_engine(f, self.engine)

File c:\Python312\Lib\site-packages\pandas\io\parsers\readers.py:1880,
in TextFileReader._make_engine(self, f, engine)
   1878     if "b" not in mode:
   1879         mode += "b"
-> 1880 self.handles = get_handle(
   1881     f,
   1882     mode,
   1883     encoding=self.options.get("encoding", None),
   1884     compression=self.options.get("compression", None),
   1885     memory_map=self.options.get("memory_map", False),
   1886     is_text=is_text,
   1887     errors=self.options.get("encoding_errors", "strict"),
   1888     storage_options=self.options.get("storage_options", None),
   1889 )
   1890 assert self.handles is not None
   1891 f = self.handles.handle
```

```
File c:\Python312\Lib\site-packages\pandas\io\common.py:873, in
get_handle(path_or_buf, mode, encoding, compression, memory_map,
is_text, errors, storage_options)
    868 elif isinstance(handle, str):
    869     # Check whether the filename is to be opened in binary
mode.
    870     # Binary mode does not support 'encoding' and 'newline'.
    871     if ioargs.encoding and "b" not in ioargs.mode:
    872         # Encoding
--> 873         handle = open(
    874             handle,
    875             ioargs.mode,
    876             encoding=ioargs.encoding,
    877             errors=errors,
    878             newline="",
    879         )
    880     else:
    881         # Binary mode
    882         handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: 'iris.csv'

# 9. b.    Develop a python program to demonstrate handling multiple
exceptions using try, except , else and finally block statements

def handle_multiple_exceptions():
    try:
        # Code that may raise multiple exceptions
        value = int(input("Enter a number: "))  # May raise ValueError
        result = 10 / value  # May raise ZeroDivisionError
        my_list = [1, 2, 3]
        print(my_list[value])  # May raise IndexError
        my_dict = {'a': 1, 'b': 2}
        print(my_dict[value])  # May raise KeyError if value is not an
alphabet

    except ValueError as e:
        print(f"ValueError caught: {e}")

    except ZeroDivisionError as e:
        print(f"ZeroDivisionError caught: {e}")

    except IndexError as e:
        print(f"IndexError caught: {e}")

    except KeyError as e:
        print(f"KeyError caught: {e}")

    else:
        # This block will run if no exceptions were raised
        print("All operations were successful!")

    finally:
        # This block will always run
        print("Execution completed, whether an exception was raised or
not.")

# Call the function to demonstrate exception handling
handle_multiple_exceptions()
```

```
Enter a number: 0
ZeroDivisionError caught: division by zero
Execution completed, whether an exception was raised or not.
```

```
# 10. a. Write a python program to demonstrate handling of the
following exceptions using try and except.
# •   Name Error
# •   Index Error
# •   Key Error
# •   Zero Division Error

def handle_exceptions():
    # Handling NameError
    try:
        print(non_existent_variable)
    except NameError as e:
        print(f"NameError caught: {e}")

    # Handling IndexError
    try:
        my_list = [1, 2, 3]
        print(my_list[5])
    except IndexError as e:
        print(f"IndexError caught: {e}")

    # Handling KeyError
    try:
        my_dict = {'a': 1, 'b': 2}
        print(my_dict['c'])
    except KeyError as e:
        print(f"KeyError caught: {e}")

    # Handling ZeroDivisionError
    try:
        result = 10 / 0
    except ZeroDivisionError as e:
        print(f"ZeroDivisionError caught: {e}")

# Call the function to see exception handling in action
handle_exceptions()
```

```
NameError caught: name 'non_existent_variable' is not defined
IndexError caught: list index out of range
KeyError caught: 'c'
ZeroDivisionError caught: division by zero

# 10. b. Write a python program to read the Iris dataset and perform
the following operations using Pandas
# •   Display first 5 rows of the dataset.
# •   Display last 5 rows of the dataset.
# •   Display the information about the dataset.
# •   Display the overview of the values of each column.
# •   Visualize the dataset using plot ().

df=pd.read_csv('iris.csv')

print("First 5 rows of the dataset:")
print(df.head())

print("\nLast 5 rows of the dataset:")
print(df.tail())

print("\nInformation about the dataset:")
print(df.info())

print("\nOverview of the values of each column:")
print(df.describe())

plt.figure(figsize=(10, 6))
df.plot(kind='line', subplots=True, layout=(5, 5), figsize=(15, 10),
title="Dataset Visualization")
plt.tight_layout()
plt.show()
```