1. Implement basic Python programs for reading input from console:

```
# Reading input from the console
name = input("Enter your name: ")
print("Hello, " + name + "!")
age = int(input("Enter your age: "))  # Convert input to an integer
print("You will be " + str(age + 1) + " years old next year.")
```

2. Perform Creation, indexing, slicing, concatenation, and repetition operations on Python built-in data types:

Strings:

```
# Creation
my_string = "Hello, World!"
# Indexing
print(my_string[0])  # Output: 'H'
# Slicing
print(my_string[7:12])  # Output: 'World'
# Concatenation
new_string = my_string + " How are you?"
print(new_string)
# Repetition
repeated_string = my_string * 3
print(repeated_string)
```

Lists:

```
# Creation
my_list = [1, 2, 3, 4, 5]
# Indexing
print(my_list[2])  # Output: 3
# Slicing
print(my_list[1:4])  # Output: [2, 3, 4]
# Concatenation
new_list = my_list + [6, 7]
print(new_list)
# Repetition
repeated_list = my_list * 2
print(repeated_list)
```

Tuples, Dictionaries, and Sets:

```
# Creation
my_tuple = (1, 2, 3)
my_dict = {'name': 'John', 'age': 30}
my_set = {1, 2, 3, 4, 5}
# Indexing (Tuples don't support item assignment)
print(my_tuple[0])  # Output: 1
# Accessing values in a dictionary
print(my_dict['name'])  # Output: 'John'
# Set operations
my_set.add(6)  # Adding an element to a set
print(my_set)
# Note: Tuples are immutable, and dictionaries and sets do not support slicing.
```

3. Solve problems using decision and looping statements:

```
# Problem: Print all even numbers from 1 to 10
for num in range(1, 11):
    if num % 2 == 0:
        print(num)
```

4. Apply Python built-in data types and methods to solve problems:

```
# Problem: Count the number of occurrences of a specific word in a sentence.
sentence = "This is a simple sentence. This sentence is for testing."
word_to_count = "sentence"
# Using the count() method for strings
count = sentence.count(word_to_count)
print(f"The word '{word_to_count}' appears {count} times in the sentence.")
```

5. Handle numerical operations using math and random number functions:

```
import math
import random
# Mathematical operations
print(math.sqrt(16))  # Square root
print(math.pi)      # Value of pi

# Random number generation
random_number = random.randint(1, 100)  # Generates a random integer between 1 and 100
print(random_number)
```

6. Create user-defined functions with different types of function arguments:

```
# Function with positional arguments
def add(x, y):
    return x + y
result = add(3, 5)
print(result)  # Output: 8
# Function with default arguments
def greet(name, greeting="Hello"):
    return f"{greeting}, {name}!"
greet_msg = greet("Alice")
print(greet_msg)  # Output: "Hello, Alice!"
# Function with variable-length arguments (*args)
def sum_all(*args):
    return sum(args)
total = sum_all(1, 2, 3, 4, 5)
print(total)  # Output: 15
```

---

1. Create packages and import modules from packages:

Assuming you have the following package structure:

```
my_package/
    __init__.py
    module1.py
    module2.py
```

In module1.py:

```
def function1():
    print("Function 1 from module1")
def function2():
    print("Function 2 from module1")
```

In module2.py:

```
def function3():
    print("Function 3 from module2")
def function4():
    print("Function 4 from module2")
```

```
from my_package import module1, module2
module1.function1()
module2.function3()
```

2. Perform File manipulations:

Open and Read a File:

```
# Open and read a file
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

Write and Append to a File:

```
# Write to a file
with open("example.txt", "w") as file:
    file.write("This is a new line.")
# Append to a file
with open("example.txt", "a") as file:
    file.write("\nThis is an appended line.")
```

Copy from One File to Another:

```
# Copy from one file to another
with open("source.txt", "r") as source_file, open("destination.txt", "w") as destination_file:
    for line in source_file:
        destination_file.write(line)
```

```
import os
# Define a directory path
directory_path = 'my_directory'
# Create a new directory (if it doesn't exist)
if not os.path.exists(directory_path):
    os.mkdir(directory_path)
    print(f"Directory '{directory_path}' created.")
# Check if a file exists
file_path = 'my_directory/my_file.txt'
if os.path.exists(file_path):
    print(f"File '{file_path}' exists.")
else:
    print(f"File '{file_path}' does not exist.")
# Create and write to a file
with open(file_path, 'w') as file:
    file.write("Hello, world!")
# Read from a file
with open(file_path, 'r') as file:
    file_content = file.read()
    print(f"File content: {file_content}")
# Append to a file
with open(file_path, 'a') as file:
    file.write("\nThis is an appended line.")
# Copy from one file to another
copy_file_path = 'my_directory/copied_file.txt'
shutil.copy(file_path, copy_file_path)
print(f"File copied to '{copy_file_path}'.")
# Rename a file
new_file_path = 'my_directory/new_file.txt'
os.rename(file_path, new_file_path)
print(f"File renamed to '{new_file_path}'.")
# List files in a directory
file_list = os.listdir(directory_path)
print(f"Files in '{directory_path}': {file_list}")
# Remove a file
os.remove(new_file_path)
print(f"File '{new_file_path}' removed.")
# Remove a directory (if it's empty)
os.rmdir(directory_path)
print(f"Directory '{directory_path}' removed.")
```

---

3. Handle Exceptions using Python Built-in Exceptions:

```
try:
    # Code that might raise an exception
    result = 10 / 0  # This will raise a ZeroDivisionError
except ZeroDivisionError as e:
    # Handle the exception
    print(f"An error occurred: {e}")
else:
    # Optional: Code to execute if no exception occurs
    print("No exception occurred")
finally:
    # Optional: Code that always executes, whether an exception occurred or not
    print("This always runs")
```

4. Solve problems using Class declaration and Object creation:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")
# Object creation
person1 = Person("Alice", 30)
person1.greet()
```

5. Implement OOP concepts like Data hiding and Data Abstraction:

```
class BankAccount:
    def __init__(self, account_number, balance):
        self._account_number = account_number  # Private attribute
        self._balance = balance
    def get_balance(self):
        return self._balance  # Abstraction, hiding internal details
    def deposit(self, amount):
        if amount > 0:
            self._balance += amount
    def withdraw(self, amount):
        if amount > 0 and amount <= self._balance:
            self._balance -= amount
# Usage
account = BankAccount("12345", 1000)
balance = account.get_balance()
print("Balance:", balance)
```

6. Solve any real-time problem using inheritance concept:

```
class Animal:
    def __init__(self, name):
        self.name = name
    def speak(self):
        pass
class Dog(Animal):
    def speak(self):
        return f"{self.name} says Woof!"
class Cat(Animal):
    def speak(self):
        return f"{self.name} says Meow!"
# Usage
dog = Dog("Buddy")
cat = Cat("Whiskers")
print(dog.speak())  # Output: "Buddy says Woof!"
print(cat.speak())  # Output: "Whiskers says Meow!"
```

1. Create Pandas Series and DataFrame from various inputs:

```
import pandas as pd
# Creating a Pandas Series from a list
my_list = [1, 2, 3, 4, 5]
my_series = pd.Series(my_list)
print(my_series)
# Creating a Pandas DataFrame from a dictionary
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35]}
df = pd.DataFrame(data)
print(df)
```

---

1. Create NumPy arrays from Python Data Structures, Intrinsic NumPy objects, and Random Functions:

```
import numpy as np
# From Python list
python_list = [1, 2, 3, 4, 5]
numpy_array_from_list = np.array(python_list)
# Intrinsic NumPy objects
numpy_zeros = np.zeros((3, 3))  # Creates a 3x3 array of zeros
numpy_ones = np.ones((2, 2))    # Creates a 2x2 array of ones
numpy_empty = np.empty((2, 2))  # Creates an empty 2x2 array
# Random NumPy array
numpy_random = np.random.rand(3, 3)  # Creates a 3x3 array with random values between 0 and 1
print(numpy_array_from_list)
print(numpy_zeros)
print(numpy_ones)
print(numpy_empty)
print(numpy_random)
```

2. Manipulation of NumPy arrays - Indexing, Slicing, Reshaping, Joining, and Splitting:

```
import numpy as np
# Create a sample array
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
# Indexing and slicing
print(arr[0])      # Access the first element
print(arr[1:4])    # Slice from index 1 to 3
print(arr[-1])     # Access the last element
print(arr[::-1])   # Reverse the array
# Reshaping
reshaped_arr = arr.reshape(3, 3)  # Reshape to a 3x3 array
print(reshaped_arr)
# Joining arrays
array1 = np.array([[1, 2], [3, 4]])
array2 = np.array([[5, 6]])
concatenated = np.concatenate((array1, array2), axis=0)  # Concatenate vertically
print(concatenated)
# Splitting arrays
split_arrays = np.split(concatenated, 3, axis=0)  # Split into 3 equal parts along rows
print(split_arrays)
```

3. Computation on NumPy arrays using Universal Functions and Mathematical methods:

```
import numpy as np
# Create sample arrays
array_a = np.array([1, 2, 3, 4, 5])
array_b = np.array([6, 7, 8, 9, 10])
# Universal Functions (element-wise operations)
sum_array = np.add(array_a, array_b)       # Element-wise addition
product_array = np.multiply(array_a, array_b)  # Element-wise multiplication
sqrt_array = np.sqrt(array_a)              # Element-wise square root
# Mathematical methods
mean_value = array_a.mean()            # Mean of the array
max_value = array_b.max()              # Maximum value in the array
print(sum_array)
print(product_array)
print(sqrt_array)
print(mean_value)
print(max_value)
```

4. Import a CSV file and perform various Statistical and Comparison operations on rows/columns:

```
import numpy as np
import pandas as pd
# Load CSV file using Pandas
df = pd.read_csv('data.csv')
# Convert DataFrame to NumPy array
numpy_array = df.to_numpy()
# Statistical operations
mean_column1 = np.mean(numpy_array[:, 0])  # Mean of the first column
sum_row2 = np.sum(numpy_array[1, :])       # Sum of the second row
# Comparison operations
greater_than_5 = numpy_array > 5           # Boolean array where elements > 5
print(mean_column1)
print(sum_row2)
print(greater_than_5)
```

---

1. Create Pandas Series and DataFrame from various inputs:

Creating a Pandas Series from a List:

```
import pandas as pd
data_list = [1, 2, 3, 4, 5]
series_from_list = pd.Series(data_list)
print(series_from_list)
```

Creating a Pandas Series from a Dictionary:

```
import pandas as pd
data_dict = {'A': 1, 'B': 2, 'C': 3}
series_from_dict = pd.Series(data_dict)
print(series_from_dict)
```

Creating a Pandas Series from a NumPy Array:

```
import pandas as pd
import numpy as np
data_array = np.array([10, 20, 30, 40, 50])
series_from_array = pd.Series(data_array)
print(series_from_array)
```

Creating a Pandas DataFrame from a Dictionary of Lists:

```
import pandas as pd
data_dict = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['New York', 'San Francisco', 'Los Angeles']
}
df_from_dict = pd.DataFrame(data_dict)
print(df_from_dict)
```

Creating a Pandas DataFrame from a List of Dictionaries:

```
import pandas as pd
data_list_of_dicts = [
    {'Name': 'Alice', 'Age': 25, 'City': 'New York'},
    {'Name': 'Bob', 'Age': 30, 'City': 'San Francisco'},
    {'Name': 'Charlie', 'Age': 35, 'City': 'Los Angeles'}
]
df_from_list_of_dicts = pd.DataFrame(data_list_of_dicts)
print(df_from_list_of_dicts)
```

Creating a Pandas DataFrame from a NumPy Array:

```
import pandas as pd
import numpy as np
data_array = np.array([
    ['Alice', 25, 'New York'],
    ['Bob', 30, 'San Francisco'],
    ['Charlie', 35, 'Los Angeles']
])
df_from_array = pd.DataFrame(data_array, columns=['Name', 'Age', 'City'])
print(df_from_array)
```

2. Import any CSV file to Pandas DataFrame:

```
Import pandas as pd
# Import a CSV file into a DataFrame
df = pd.read_csv('data.csv')
# (a) Visualize the first and last 10 records
print(df.head(10))  # First 10 records
print(df.tail(10))  # Last 10 records
# (b) Get the shape, index, and column details
print("Shape:", df.shape)
print("Index:", df.index)
print("Columns:", df.columns)
# (c) Select/Delete the records(rows)/columns based on conditions.
# Select rows where 'Age' is greater than 30
selected_rows = df[df['Age'] > 30]
print(selected_rows)
# Delete rows where 'Age' is less than 25
df = df[df['Age'] >= 25]
# Delete a column
df = df.drop('Salary', axis=1)
# (d) Perform ranking and sorting operations
df['Rank'] = df['Salary'].rank(ascending=False)  # Ranking based on 'Salary'
sorted_df = df.sort_values(by='Salary', ascending=False)
# (e) Do required statistical operations on the given columns
mean_salary = df['Salary'].mean()
max_age = df['Age'].max()
# (f) Find the count and uniqueness of the given categorical values
gender_count = df['Gender'].value_counts()
unique_cities = df['City'].unique()
# (g) Rename single/multiple columns
df = df.rename(columns={'Name': 'Full Name', 'Age': 'Years'})
print(df)
```

---

1. Import any CSV file to Pandas DataFrame:

(a) Handle missing data:

```
# Drop rows with missing values
df = df.dropna()
# Fill missing values with a specific value (e.g., mean)
mean_age = df['Age'].mean()
df['Age'].fillna(mean_age, inplace=True)
```

(b) Transform data using apply() and map() method:

```
# Using apply to square the 'Age' column
df['Age_squared'] = df['Age'].apply(lambda x: x**2)
# Using map to map gender to numeric values
gender_mapping = {'Male': 0, 'Female': 1}
df['Gender_numeric'] = df['Gender'].map(gender_mapping)
```

(c) Detect and filter outliers:

```
from scipy import stats
z_scores = np.abs(stats.zscore(df['Salary']))
threshold = 3
outliers = df[z_scores > threshold]
# You can then choose to remove the outliers using df.drop()
```

(d) Perform Vectorized String operations on Pandas Series:

```
import pandas as pd
# Create a sample Pandas Series
data = {'Names': ['Alice', 'Bob', 'Charlie', 'David', 'Eve']}
series = pd.Series(data['Names'])
# Convert all names to uppercase
series_upper = series.str.upper()
# Check if each name contains 'A'
contains_a = series.str.contains('A')
# Extract the first two characters from each name
first_two_chars = series.str[:2]
# Replace 'A' with 'X' in each name
replace_a_with_x = series.str.replace('A', 'X')
# Display the results
print("Original Series:")
print(series)
print("\nUppercase Series:")
print(series_upper)
print("\nContains 'A' Series:")
print(contains_a)
print("\nFirst Two Characters Series:")
print(first_two_chars)
print("\nReplace 'A' with 'X' Series:")
print(replace_a_with_x)
```

(e) Visualize data:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
# Line plot:
data = {'Year': [2010, 2011, 2012, 2013, 2014],
        'Sales': [100, 120, 90, 110, 130]}
df = pd.DataFrame(data)
df.plot(x='Year', y='Sales', kind='line')
plt.xlabel('Year')
plt.ylabel('Sales')
plt.title('Line Plot')
plt.show()
# Bar plot:
data = {'Category': ['A', 'B', 'C', 'D'], 'Value': [50, 75, 60, 90]}
df = pd.DataFrame(data)
df.plot(x='Category', y='Value', kind='bar')
plt.xlabel('Category')
plt.ylabel('Value')
plt.title('Bar Plot')
plt.show()
# Histogram:
data = {'Values': [30, 45, 60, 75, 90, 100, 120, 135, 150, 180]}
df = pd.DataFrame(data)
df.plot(kind='hist', bins=5, rwidth=0.8, title='Distribution of Values')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
# Density plot:
np.random.seed(0)
sample_data = np.random.normal(loc=0, scale=1, size=1000)
sns.histplot(sample_data, kde=True, color="green")
plt.xlabel("Value")
plt.ylabel("Density")
plt.title("Density Plot of Sample Data")
plt.show()
# Scatter plot:
np.random.seed(0)
x = np.random.rand(50)
y = 2 * x + 1 + np.random.randn(50)
plt.scatter(x, y, color='blue', label='Data Points')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot of Sample Data')
plt.legend()
plt.grid(True)
plt.show()
```