Chapter 1

1.The Zen of python Reflection:

Principle 1: “Beautiful is better than ugly.”

This principle emphasizes writing code that is clean, readable, and aesthetically pleasing. In advanced Python projects, this belief encourages consistent formatting, meaningful variable names, and clear logic flow. Beautiful code is easier for collaborators to understand, maintain, and extend. It also reduces the cognitive load required to debug and review code, making the development process more efficient.

In large-scale or long-term applications, following this principle prevents technical debt. Readable code ages better, and this principle often guides decisions such as choosing clarity over cleverness or selecting design patterns that prioritize structure and elegance.

Principle 2: “Simple is better than complex.”

This principle encourages avoiding unnecessary abstraction, over-engineering, or deeply nested logic. In advanced Python development, simplicity supports scalability: when each component has a clear purpose, you avoid tangled dependencies and unpredictable behavior. It also guides architectural decisions—modular design, small functions, and minimal moving parts.

In practice, this principle helps maintain focus on solving the problem rather than adding premature optimizations. Simple solutions are easier to test, refactor, and reason about, especially in large or collaborative codebases.

Principle 3: “There should be one—and preferably only one—obvious way to do it.”

This idea shapes Python's philosophy of reducing ambiguity. In advanced projects, it motivates choosing a single clear approach rather than offering multiple slightly-different solutions that confuse collaborators. For example, following consistent patterns for error handling, configuration, or application structure prevents inconsistent “styles” across the codebase.

This principle also promotes best practices. When there is a widely accepted method (like using list comprehensions or context managers), choosing it increases readability and standardization. It keeps code uniform and predictable across teams.

Bytecode Inspection:

```
def square(x):
    return x * x

def multiply(a, b):
    return a * b

def add(a, b):
    return a + b

print(f"square(5) = {square(5)}")
print(f"multiply(3, 4) = {multiply(3, 4)}")
print(f"add(2, 3) = {add(2, 3)}")
```

Dynamic typing in action:

```
data = 42
print(f"First, data = {data} (it's an {type(data)})")

data = [1, 2, 3]
print(f"Now, data = {data} (it's a {type(data)})")

def my_func():
```

```
    pass

data = my_func
print(f"Finally, data = {data} (it's a {type(data)})")
```

Comparing Python Implementations:

brief research:

PyPy is an alternative Python interpreter known for its Just-In-Time (JIT) compilation, which makes it significantly faster for long-running programs. It aims to be highly compatible with CPython but focuses heavily on speed optimization. PyPy manages memory differently and can run computational code much faster.
Jython is a Python implementation that runs on the Java Virtual Machine (JVM). It allows Python code to import and interact directly with Java libraries. Jython integrates seamlessly with the Java ecosystem, making it ideal for projects requiring JVM compatibility.

How PyPy and Jython differ from CPython:

PyPy is an alternative Python interpreter that uses a Just-In-Time (JIT) compiler, which allows it to execute long-running or loop-heavy programs significantly faster than CPython. While CPython interprets code line by line, PyPy can compile parts of the program into machine code at runtime, improving performance. PyPy aims for high compatibility with CPython, though some C-extension modules may not work perfectly. It is ideal for CPU-intensive applications where execution speed matters.

Jython is a Python implementation designed to run on the Java Virtual Machine (JVM) instead of the standard CPython runtime. Unlike CPython, Jython allows direct integration with Java libraries and lets Python code interact seamlessly with Java classes. However, it doesn't support CPython's C-extension modules, which limits compatibility with some packages. Jython is most useful in Java-based enterprise systems or when Python needs to interoperate closely with Java.

When PyPy is More Advantageous Than CPython:

•CPU-intensive or algorithm-heavy programs, such as simulations, data processing loops, or mathematical computations, run significantly faster due to PyPy's JIT compiler.
•Long-running services or backend applications benefit because PyPy optimizes repeated execution over time.
•Pure Python codebases (with minimal C-extensions) are ideal for PyPy since it performs best when executing native Python logic.
•Performance-critical prototypes where experimenting with faster execution is important.
When Jython is More Advantageous Than CPython
•Projects that need direct access to Java libraries, such as Java-based networking, GUI tools, or enterprise frameworks.
•Enterprise environments already built on JVM, where integrating Python scripts directly into Java applications is easier and avoids cross-language barriers.
•Systems requiring JVM features, such as Java's garbage collection, security model, or multithreading advantages.
•Organizations standardizing on JVM infrastructure, making Jython useful for embedding Python inside Java systems.

Abstract syntax Tree (AST) exploration:

```
Import ast
code_snippet = "y = (4 * 5) - 3"
tree = ast.parse(code_snippet)

print("AST for 'y = (4 * 5) - 3':")
print(ast.dump(tree, indent=4))
print("\n" + "="*50 + "\n")
```

Mutability and object identity:

```
my_list = [10, 20, 30]
print(f"my list now: {my_list}")

first_address = id(my_list)
print(f"Memory address: {first_address}")

my_list.append(40)
print(f"Adding 40: {my_list}")

second_address = id(my_list)
print(f"Memory address now: {second_address}")

if first_address == second_address:
    print("✓ Same address! Lists keep their home when you add items.")
else:
    print("✗ Different address!")
print()
```

Chapter 2

Vector3D Class with Operator Overloading:

```
class Vector3D:
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

    def __add__(self, other):
        return Vector3D(self.x + other.x, self.y + other.y, self.z + other.z)

    def __sub__(self, other):
        return Vector3D(self.x - other.x, self.y - other.y, self.z - other.z)

    def __mul__(self, other):
        return self.x * other.x + self.y * other.y + self.z * other.z

    def __repr__(self):
        return f"Vector3D({self.x}, {self.y}, {self.z})"

print("Creating two vectors:")
v1 = Vector3D(1, 2, 3)
v2 = Vector3D(4, 5, 6)
print(f"v1 = {v1}")
print(f"v2 = {v2}")

print(f"v1 + v2 = {v1 + v2}")
print(f"v1 - v2 = {v1 - v2}")
print(f"v1 * v2 = {v1 * v2}")
```

Positive Number Descriptor:

```
class BankAccount:
    def __init__(self, balance=0):
        self._balance = balance

    @property
    def balance(self):
        return self._balance

    @balance.setter
    def balance(self, value):
        if value < 0:
            print("ERROR: Balance can't be negative!")
        else:
            self._balance = value

    def withdraw(self, amount):
        if self._balance - amount < 0:
            print(f"ERROR: Can't withdraw ${amount}. Only ${self._balance} available.")
        else:
            self._balance -= amount
            print(f"Withdrew ${amount}. New balance: ${self._balance}")

account = BankAccount(100)
print(f"Starting balance: ${account.balance}")
account.withdraw(50)
```

```
account.withdraw(60)
account.balance = -20
```

Point Class with __slots__:

```
class Point:
    __slots__ = ('x', 'y')

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return f"Point({self.x}, {self.y})"

print("Creating a point at (3, 4):")
p = Point(3, 4)
print(f"Point: {p}")
print(f"p.x = {p.x}")
print(f"p.y = {p.y}")

try:
    p.z = 5
    print("Successfully added z attribute!")
except AttributeError as e:
    print(f"Failed to add z: {e}")
    print("Why? Because _slots_ restricts us to only x and y!")
```

Disassembling a simple function:

```
def calculate_sum(a, b):
    return a + b

print(f"sum of 2 and 3 is {calculate_sum(2, 3)}")
print("Function: calculate_sum(a, b)")
print("What it does: returns a + b")

import dis

print("\n(bytecode):")
dis.dis(calculate_sum)
print(f"calculate_sum(5, 3) = {calculate_sum(5, 3)}")

print("\nWhat the bytecode means:")
print("- LOAD_FAST: Loads a variable onto the stack (like getting a value ready)")
print("- BINARY_ADD: Takes two values from stack, adds them, puts result back")
print("- RETURN_VALUE: Returns the result from the function")
```

Chapter 3

- Multiple Choice Questions (MCQs)

9.Which of the following is a characteristic of a pure function?

a) Depends on global variables

b) Produces side effects

c)Always returns the same output for the same input

d) Modifies input arguments

10.Which functional programming concept ensures that once a variable is assigned, it cannot be changed?

a) Recursion

b)Immutability

c) Closures

d) Memoization

11.Which built-in function applies a function to all elements of an iterable and returns an iterator?

a)filter()

b) reduce()

c) map()

d) zip()

12.Which module provides the reducefunction in Python?

a)operator

b) itertools

c) functools

d) collections

13.The filter()function in Python returns:

a) A list of all elements

b)An iterator containing elements that satisfy the condition

c) A tuple of matching elements

d) None

-True/False Questions

14.Functional programming in Python encourages immutability and pure functions. True

15.The map() function modifies the original iterable in place. False

16.Closures allow inner functions to access variables from their enclosing function even after the outer function has finished execution. True

17.The reduce() function is a built-in function in Python and does not require an import. False

18.itertools provide memory-efficient tools for working with iterators, including infinite sequences. True

Chapter 4

Multiple Choice Questions (MCQs)

1. Which function in Python checks only at the beginning of a string?

a) re.match()

b) re.search()

c) re.findall()

d) re.sub()

2. What does the regex pattern \d+ match?

a) One or more letters

b) One or more digits

c) Exactly one digit

d) Zero or more digits

3. Which regex will match any string ending with "ing"?

a) ^ing b) ing$

b)ing$

c) .*ing

d) (ing)?

4. The output of re.findall(r"[aeiou]", "Python Programming") is:

a) ['a', 'o', 'o', 'a']

b) ['o', 'o', 'a', 'i']

c) ['y', 'a', 'i']

d) []

5. Which regex matches a valid variable name in Python (letters, numbers, underscores, not starting with digit)?

a) ^\d\w*$

b) ^[A-Za-z_]\w*$

c) ^\w+$

d) ^[A-Z]\d*$

6. The metacharacter ^ inside brackets [^...] means:

a) Start of string

b) End of string

c) Negation (not these characters)

d) Match newline

7. What will be the result of: re.split(r"\s+", "Python is easy")

a) ['Python is easy']

b) ['Python', 'is', 'easy']

c) ['Python', ' is easy']

d) ['', 'Python', 'is', 'easy']

8. Which function is best for replacing substrings using regex?

a) re.match()

b) re.sub()

c) re.search()

d) re.findall()

True / False Questions

1. re.match() scans the entire string for a pattern. False

2. The regex . matches any character except a newline. True

3. Regex \w+ matches only uppercase letters. False

4. The regex \d{3} matches exactly three digits. True

5. re.sub() can be used for both searching and replacing. True

6. Regex patterns in Python are case-sensitive unless re.IGNORECASE is used. True

7. re.findall() returns only the first match found. False

8. ^Python$ matches the string "I love Python". False

Chapter 5

Rectangle Class:

```
class rectangle:
    def __init__(self,width,height):
        self.width=width
        self.height=height
    def area(self):
        return self.width*self.height
    def perimeter(self):
        return 2*(self.width+self.height)
rect=rectangle(5,10)
print("Area:",rect.area())
print("Perimeter:",rect.perimeter())
```

Employee Class with Alternative Constructor:

```
class employee:
    def __init__(self,name,employee_id,salary):
        self.name=name
        self.salary=salary
        self.employee_id=employee_id
    @classmethod
    def from_string(cls,emp_str):
        name,employee_id,salary=emp_str.split(",")
        return cls(name,employee_id,int(salary))
    def display_employee_info(self):
        print(f"Name:{self.name}, ID:{self.employee_id}, Salary:{self.salary}")
emp1=employee.from_string("johd doe,E123,50000")
emp1.display_employee_info()
class employee:
    def __init__(self,name,employee_id,salary):
        self.name=name
        self.salary=salary
        self.employee_id=employee_id
    @classmethod
    def from_string(cls,emp_str):
        name,employee_id,salary=emp_str.split(",")
        return cls(name,employee_id,int(salary))
    def display_employee_info(self):
        print(f"Name:{self.name}, ID:{self.employee_id}, Salary:{self.salary}")
emp1=employee.from_string("johd doe,E123,50000")
emp1.display_employee_info()
```

Vehicle Hierarchy:

```
class vehicle:
     def move(self):
         print("The vehicle is moving")
class car(vehicle):
     def move(self):
         print("The car is driving")
class bike(vehicle):
     def move(self):
         print("The bike is cycling")
vehicles = [vehicle(), car(), bike()]
for v in vehicles:
     v.move()
```

Vector Class with Operator Overloading:

```
class vector:
    def __init__(self,x,y):
        self.x=x
        self.y=y
```

```
    def __sub__(self,other):
        return vector(self.x-other.x,self.y-other.y)
    def __mul__(self,other):
        return self.x * other.x + self.y * other.y
    def __repr__(self):
        return f"vector({self.x}, {self.y})"
v1=vector(3,4)
v2=vector(1,2)
print("Subtraction:",v1 - v2 )
print("Dot Product:",v1 * v2 )
```

Shape Polymorphism Function:

```
class shape:
    def area(self):
       return 0
class circle(shape):
   def __init__(self,r):
         self.r = r
   def area(self):
        return 3.14 * self.r **2
class rectangle(shape):
    def __init__(self,l,b):
         self.l = l
         self.b = b
    def area(self):
        return self.l * self.b
c = circle(5)
r = rectangle(4,6)
print("Area of circle:",c.area())
print("Area of rectangle:",r.area())
```

Chapter 6

Multiple Choice Questions (MCQs)

1. Which Python module is used for reading and writing CSV files?

a) json

b) csv

c) pandas

d) openpyxl

2. What does csv.DictReader() return when reading a CSV file?

a) List of lists

b) Dictionary for each row with column names as keys

c) Tuple for each row

d)String

3. Which function is used to convert a Python object into a JSON string?

a) json.load()

b) json.loads()

c) json.dumps()

d) json.dump()

4. What will the following code produce?

```
import pandas as pd    df = pd.read_excel("data.xlsx", sheet_name="Sheet1")
```

a) Reads all sheets into a dictionary of DataFrames

b) Reads only the specified sheet into a DataFrame

c) Creates a new Excel file named "data.xlsx"

d) Reads an empty DataFrame

5. Which library must be installed to read/write Excel files with pandas?

a) xlrd

b) openpyxl

c) csv

d) numpy

True / False Questions

1. The csv module automatically converts numbers in a CSV file to integers or floats. False

2. json.dump() writes JSON data directly to a file. True

3. pandas can read both CSV and JSON files into DataFrames. True

4. The default file format supported by pandas.read_excel() is .xlsx. True

5. Excel files can be written using pandas without any external library. True

Short Answer / Conceptual Questions

1. **Differentiate between json.load() and json.loads().**
   Json.load(): reads json from a file but json.loads() reads json from a string

2. **Explain the difference between csv.reader and csv.DictReader.**
   Csv.reader returns each row as a list and csv.dictreader print each row as a disctionary using column names as keys.

3. **Why might we prefer to use pandas for CSV and Excel files instead of the built-in csv module?**
   cuz pandas is faster,easier to use,works well with large data

4. **How can you write data to multiple sheets in an Excel file using pandas?**
   by using excelwriter : with pd.excelwriter("students.xlsx") as writer:
   df1.to_excel(writer,sheet_name="sheet1", index= false) ##repeat this for other sheets.

5. **What are the advantages of JSON over CSV in representing hierarchical data?**
   Supports nested data and can store lists,Booleans,numbers and complex structures.

Programming Problems

CSV Handling

```
import csv
students=[["ID","Name","Grade"],
[1,"ali",85],
[2,"mona",92],
[3,"omar",78]]
with open("students.csv","w",newline="")as f:
   writer=csv.writer(f)
   writer.writerows(students)
with open("students.csv","r")as f:
   reader=csv.DictReader(f)
   for row in reader:
      if int(row["Grade"])>=80:
         print(row["Name"])
```

JSON Handling

```
import json
data = {"course": "Python",
       "duration": "3 months",
       "students": ["Ali", "Sara"]}
with open("course.json","w")as f:
   json.dump(data,f)
with open("course.json","r")as f:
   content=json.load(f)
print(content["students"])
```

Excel Handling

```
 import pandas as pd
df=pd.DataFrame({
   "id":[1,2,3,4,5],
   "name":["Alice","Bob","Charlie","David","Eva"],
   "salary":[50000,60000,55000,70000,65000]})
df.to_excel("employees.xlsx",index=False)
df2=pd.read_excel("employees.xlsx")
print(df2[["name","salary"]])
```

Data Transformation

```
import csv import json
data=[["Name","Age","City"],
      ["Ali",25,"cairo"],
      ["Mona",30,"Alex"]]
with open('input.csv','w',newline='') as f:
    writer=csv.writer(f)
    writer.writerows(data)
def csv_to_json(csv_file, json_file):
    people=[]
    with open(csv_file,'r') as f:
        reader = csv.DictReader(f)
        for row in reader:
            people.append({
                "Name": row["Name"],
                "Age": int(row["Age"]),
                "City": row["City"]})
            output={"people": people}
with open(json_file,'w') as f:
      json.dump(output,f, indent=4)
      csv_to_json('input.csv', 'output.json')
with open('output.json','r') as f:
      content=f.read()
print(content)
```

Chapter 7

Multiple Choice Questions (MCQs)

1. Which Python module is included in the standard library for working with SQLite databases?

a) mysql.connector

b) psycopg2

c) sqlite3

d) sqlalchemy

2. What does conn.commit() do after an INSERT statement?

a) Closes the database connection

b) Saves changes permanently in the database

c) Rolls back the transaction

d) Executes the SQL query again

3. Which placeholder style is used in parameterized queries with sqlite3?

a) %s

b) :param

c) ?

d) $1

4. Which method is used to fetch only the first row of a query result?

a) fetchall()

b) fetchmany()

c) fetchone()

d) next()

5. In SQLAlchemy, which class is commonly used to define ORM models?

a) Base

b) Mapper

c) Session

d) Engine

True / False Questions

1. SQLite databases are stored in memory only and cannot be written to a file. False

2. Using parameterized queries helps prevent SQL injection attacks. True

3. The rollback() method can undo uncommitted changes in a transaction. True

4. SQLAlchemy provides both Core (SQL Expression Language) and ORM interfaces. True

5. cursor.execute() always returns a list of results. False

Short Answer Questions

1. **What is the difference between fetchone(), fetchmany(n), and fetchall() in database cursors?**
   Fetchone():Returns one row from the query result
   Fetchmany(n):Returns the first n rows from the result
   Fetchall():Returns all remaining rows from the result

2. **Why are parameterized queries preferred over string concatenation when inserting user input into SQL statements?**
   parameterized queries prevent SQL injection treat user input as date (not executable SQL) improve security and reduce errors

3. **What is a transaction in databases, and why is it important?**
   A transaction databae is a group of database operations executed as a single unit it is important cuz it ensure (atomicity,consistency,isolation,rollback ability if an error occurs

4. **Write the steps (in order) to connect to an SQLite database and insert a row into a table.**
   1.import sqlite3
   2. use connect() to open the database
   3.create a cursor
   4.execute an INSERT statement
   5.call commit() to save changes

6.close the connection using closer()

5. **Briefly explain how ORM (Object Relational Mapping) improves database handling in Python.**

ORM converts database tables into Python classes and rows into objects. This makes database operations easier, safer, and more readable. You can create, update, delete, and fetch data using simple Python code instead of complex SQL. ORM also reduces errors, improves productivity, and keeps the code more organized and maintainable.

Programming Problems

Basic SQLite CRUD

```
import sqlite3

conn=sqlite3.connect('school.db')
c=conn.cursor()
c.execute('''CREATE TABLE IF NOT EXISTS students
             (id INTEGER PRIMARY KEY,
              name TEXT,
              grade REAL)''')
students = [(1, 'Ali', 85.5),
    (2, 'Sara', 92.0),
    (3, 'Mohamed', 78.3)]
c.executemany('INSERT OR REPLACE INTO students VALUES (?, ?, ?)', students)
c.execute('SELECT * FROM students')
rows = c.fetchall()
for row in rows:
    print(row)
conn.commit()
conn.close()
```

Parameterized Queries

```
##same code as the previous question
name=input("Enter name: ")
grade=float(input("Enter grade: "))
c.execute('INSERT INTO students (name, grade) VALUES (?, ?)', (name, grade))
print("--update records--")
```

Transactions

```
import sqlite3
conn=sqlite3.connect('school.db')
c = conn.cursor()
try:
    print("starting transaction")
    conn.execute("BEGIN")
    c.execute('''CREATE TABLE IF NOT EXISTS students
             (id INTEGER PRIMARY KEY,
              name TEXT,
              grade REAL)''')
    c.execute('INSERT INTO students (name, grade) VALUES (?, ?)', ('Lina', 88.7))
    c.execute('INSERT INTO students (name, grade) VALUES (?, ?)', ('Omar', 91.2))
    error=1/0
except Exception as e:
    print("An error occurred:", e)
    conn.rollback()
    print("Transaction rolled back ")
finally:
    students = [(1, 'Ali', 85.5),
    (2, 'Sara', 92.0),
    (3, 'Mohamed', 78.3)]
c.executemany('INSERT OR IGNORE INTO students VALUES (?, ?, ?)', students)
print('final records:')
c.execute('SELECT * FROM students')
rows = c.fetchall()
for row in rows:
```

```
    print(row)
conn.commit()
conn.close()
```

ORM with SQLAlchemy

```
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import declarative_base,sessionmaker

base=declarative_base()
class book(base):
   __tablename__ = 'books'
   id = Column(Integer, primary_key=True)
   title = Column(String)
   author = Column(String)
   def __repr__(self):
      return f"<Book(id='{self.id}', Title='{self.title}', Author='{self.author}')>"
engine = create_engine('sqlite:///books.db')
base.metadata.create_all(engine)
session=sessionmaker(bind=engine)
session=session()
book1=book(title='Python Basics',author='Guido')
book2=book(title='AI with Python',author='Mohamed')
session.add_all([book1,book2])
session.commit()
books=session.query(book).all()
for b in books:
   print(b)
```

Chapter 8

Multiple Choice Questions (MCQs)

1. Which of the following is NOT a core feature of NumPy?
a) N-dimensional arrays
b) Vectorized operations
c) Web routing and URL mapping
d) Broadcasting

2. In Pandas, which method is used to group data for aggregation?
a) group()
b) aggregate()
c) groupby()
d) merge()

3. Which library provides high-level visualization functions like heatmap and pairplot?
a) Matplotlib
b) Seaborn
c) NumPy
d) SciP

4. Flask is considered a:
a) Micro web framework
b) Full-stack web framework
c) Machine learning library
d) Numerical computing library

5. In Django ORM, a database table is typically represented as:
a) A Python dictionary
b) A Pandas DataFrame
c) A model class
d) A NumPy array

6. Which library uses tensors and is widely used for deep learning?
a) TensorFlow
b) Flask
c) Pandas
d) Matplotlib

7. Which of the following can be achieved with SciPy but not directly with NumPy?
a) Eigenvalues computation
b) Array creation
c) Element-wise multiplication
d) Broadcasting

8. Which statement is true regarding PyTorch?
a) It does not support GPU acceleration.
b) It is mainly used for scientific computing like NumPy.
c) It supports dynamic computation graphs.
d) It cannot be used for deep learning.

True / False Questions

1. NumPy arrays are less efficient than Python lists for numerical computations. False

2. Pandas DataFrame is a two-dimensional labeled data structure. True

3. Seaborn is built on top of Matplotlib. True

4. Flask is heavier and more complex than Django. True

5. TensorFlow and PyTorch both provide tensor operations and automatic differentiation. True

6. Django ORM automatically creates SQL queries for models. True

Short Answer Questions

1. **Explain the difference between NumPy and SciPy.**

   NumPy :A fundamental library for numerical computing.Provides ndarrays, vectorized operations, broadcasting, basic linear algebra, and mathematical functions.Focuses on fast array operations.

   SciPy :Built on top of NumPy.Provides advanced scientific and engineering tools, such as optimization, signal processing, statistics, interpolation, and advanced linear algebra.Used for more complex scientific computations.

2. **What is the purpose of the groupby() function in Pandas? Give an example.**

   The groupby() function is used to split data into groups based on a column, then apply aggregation functions like sum, mean, count

```
result = df.groupby("Class")["Score"].mean()
print(result)
```

   This will calculate the average score for each class.

3. **Compare Flask and Django in terms of complexity and use cases.**

   Flask:A micro web framework.Very simple, lightweight, and flexible.You build everything yourself Best for small projects, APIs, prototypes, and when you want full control.

   Django:A full-stack web framework.Comes with many built-in features: ORM, admin panel, authentication, security tools, templates.More structured and opinionated.Best for large, complex, production-level applications.

4. **Why are tensors important in deep learning frameworks like PyTorch and TensorFlow?**

   They are multi-dimensional arrays optimized for mathematical operations.can run on GPUs, which makes computations extremely fast.support automatic differentiation, which is necessary for training neural networks (backpropagation).They represent data and model parameters efficiently.

5. **What is the difference between Matplotlib and Seaborn in visualization?**

   Matplotlib:A basic, low-level visualization library.Gives full control but requires more code.Looks more “classic” and less visually polished.

   Seaborn:Built on top of Matplotlib.Provides beautiful, high-level, statistical visualizations.Easier to use and produces attractive plots by default (heatmap, pairplot, violinplot, etc.)

Programming Problems

NumPy Operations

```
import numpy as np

num=np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
mean=np.mean(num)
median=np.median(num)
std=np.std(num)
print("Mean:", mean)
print("Median:", median)
print("Standard Deviation:", std)
```

Pandas Filtering

```
import pandas as pd
df=pd.DataFrame({'Name':["Alice", "Bob", "Charlie",'john'],
            "Age":[25, 30, 35,20],
             "Score":[85.5, 90.0, 95.5,70.0]})
filter=df[df['Score']>80]
print(filter)
```

Visualization with Matplotlib

```
import matplotlib.pyplot as plt
x=[1,2,3,4,5]
y=[1,4,9,16,25]
plt.plot(x,y)
```

```
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Plot')
plt.show()
```

Flask Application

```
from flask import Flask

app = Flask(__name__)
@app.route('/hello')
def home():
    return "Hello,Advanced Python!"
if __name__ == '__main__':
    app.run(debug=True)
```

PyTorch Tensor Operations

```
import torch

a=torch.tensor([1,2,3])
b=torch.tensor([4,5,6])
dot=torch.dot(a,b)
elementwise=a*b
print("Dot Product:", dot)
print("Element-wise Multiplication:", elementwise)
```

Chapter 9

Multiple Choice Questions (MCQs)

1. Which Python library is best suited for sending HTTP requests?
a) os
b) requests
c) selenium
d) re

2. Which function in requests is used to fetch a webpage?
a) requests.open()
b) requests.page()
c) requests.get()
d) requests.read()

3. In BeautifulSoup, which method is used to extract all tags?
a) soup.find("a")
b) soup.select("a")
c) soup.find_all("a")
d) soup.get("a")

4. What does the .text property of a BeautifulSoup element return?
a) The HTML tags
b) The attribute values
c) The inner text of the tag
d) None of the above

5. Which library is used to automate interaction with JavaScript-heavy websites?
a) requests
b) BeautifulSoup
c) re
d) Selenium

6. Which of the following is an ethical consideration in web scraping?
a) Ignoring robots.txt
b) Scraping sensitive/private data
c) Respecting rate limits
d) Stealing copyrighted material

7. The compile() function is used in scraping to:
a) Convert Python code into machine language
b) Compile JavaScript on a page
c) Compile HTML into text
d) None of the above

True/False Questions

1. requests.get() returns both the HTML source and status code. True

2. BeautifulSoup can directly fetch web pages from the internet. False

3. Selenium can be used to fill forms and click buttons on web pages. True

4. Scraping a website too frequently can overload the server. True

5. Saving data into JSON format requires the csv module. False

Short Answer Questions

1. **Explain the difference between requests and Selenium in web scraping.**
   Requests: is used to fetch HTML content directly and works well with static websites. It is fast and lightweight. Selenium :automates a real browser, allowing interaction with dynamic websites that use JavaScript. It is slower but more powerful for complex pages.

2. **What is the purpose of the robots.txt file on a website?**
   The robots.txt file tells web scrapers and search engine bots which parts of the website they are allowed or not allowed to access. It defines the website's scraping rules.

3. **Write the difference between .find() and .find_all() methods in BeautifulSoup.**
   find() returns the first matching element only.
   find_all() returns a list of all matching elements.

4. **Why is it important to use headers like "User-Agent": "Mozilla/5.0" in requests.get()?**
   Some websites block bots. Adding a User-Agent makes your request look like it came from a real browser, helping avoid blocks and giving you access to the page.

5. **List three possible formats to store scraped data.**
   CSV- JSON-Excel (XLSX)

Programming Problems

Fetch a Web Page Title

```
import requests
from bs4 import BeautifulSoup
url='http://example.com'
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')
title=soup.title.text
print("page title:",title)
```

Extract All Links

```
import requests
from bs4 import BeautifulSoup
url='http://example.com'
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')
for link in soup.find_all('a'):
    print(link.get('href'))
```

Extract a Table

```
from bs4 import BeautifulSoup
html='''
<table>
<tr><th>Name</th><th>Age</th></tr>
<tr><th>Alice</th><th>25</td></tr>
<tr><th>Bob</th><th>30<td></tr>
</table>
'''
soup=BeautifulSoup(html,'html.parser')
rows=soup.find_all('tr')
for row in rows:
    cells=[cell.text for cell in row.find_all(['th','td'])]
    print(cells)
```

Automate Google Search

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from time import sleep
driver=webdriver.Chrome()
driver.get("http://ww.google.com")
search_box=driver.find_element("name","q")
search_box.send_keys("Python Web Scraping")
search_box.send_keys(Keys.RETURN)
sleep(2)
print(driver.title)
driver.quit()
```

Save Scraped Data to CSV

```
from bs4 import BeautifulSoup
```

```
import csv
html = """
<ul>
<li>Apple</li>
<li>Banana</li>
<li>Cherry</li>
</ul>
"""
soup=BeautifulSoup(html, 'html.parser')
fruits = [li.text for li in soup.find_all('li')]
with open('fruits.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Fruit'])
    for fruit in fruits:
        writer.writerow([fruit])
print('fruits.csv created sucessfully')
```

Chapter 10

Multiple Choice Questions (MCQs)

1. Which method is called when entering a context manager block using with?

a) __init__()

b) __enter__()

c) __exit__()

d) __call__()

2. Which keyword is used in Python generators?

a) return

b) yield

c) await

d) break

3. In the Observer pattern, what is the primary responsibility of the Subject?

a) Execute business logic

b) Maintain state and notify observers

c) Inject dependencies

d) Create objects dynamically

4. Which design pattern ensures only one instance of a class exists?

a) Factory

b) Singleton

c) Observer

d) Proxy

5. Which of the following is NOT a benefit of Dependency Injection?

a) Increased flexibility

b) Easier testing

c) Stronger coupling between classes

d) Better modularity

True / False Questions

1. The __exit__() method in a context manager is always executed, even if an exception occurs inside the with block.

True

2. Generators return all values at once like a list False

3. Coroutines can both produce values and receive input using send(). True

4. The Factory pattern is used to notify multiple observers when the state of an object changes. False

5. Dependency Injection helps reduce coupling between classes. True

Short Answer / Conceptual Questions

1. What is the difference between a generator and a coroutine in Python?

A generator produces values lazily using yield, while a coroutine can also consume values sent into it using send(). Coroutines are often used for event-driven programming and concurrency.

2. Explain why the with statement is preferred over manual resource management.

The with statement ensures resources (like files, sockets, or locks) areautomatically cleaned up via __exit__(), even if an exception occurs, making code safer and cleaner.

3. Give a real-world example where the Observer pattern might be applied.

In a stock trading app, multiple UI components (observers) need to be updated whenever stock prices (subject state) change.

4. What problem does the Factory pattern solve?

It abstracts object creation, allowing clients to create objects without depending on their concrete classes.

5. How does Dependency Injection improve testability of code?

It allows dependencies (like services or databases) to be swapped out with mock objects during testing, making unit tests easier and more isolated.

Programming Problems

Context Manager

```
import time
class timer:
    def __enter__(self):
        self.start = time.time()
        return self
 def __exit__(self, exc_type, exc_value, traceback):
        end = time.time()
        print(f'executed tool {end-self.start:.5f} seconds')
with timer() as t:
    for i in range(1000000):
        pass
```

Generator

```
def even(n):
    for num in range(2,n+1,2):
        yield num
for num in even(10):
    print(num)
```

Coroutine

```
def filter_positive():
    while True:
        num=yield
        if num>0:
            print(f"positive number: {num}")
co=filter_positive()
next(co)
co.send(-3)
co.send(5)
co.send(0)
```

Factory Pattern

```
class circle:
    def draw(self):
        return "drawing a circle"
class square:
    def draw(self):
        return "drawing a square"
def factory(shape_type):
    if shape_type=="circle":
        return circle()
    elif shape_type=="square":
        return square()
    else:
        return None
shape=factory("circle")
print(shape.draw())
```

Observer

```
class subject:
    def __init__(self):
        self.observers = []
    def attach(self, observer):
        self.observers.append(observer)
    def notify(self, message):
        for observer in self.observers:
            observer.update(message)

class observer:
    def update(self,message):
        print(f"Received update {message}")

subject = subject()
obs1,obs2=observer(),observer()
subject.attach(obs1)
subject.attach(obs2)
subject.notify("Update available!")
```