**Python Advanced Concepts Learning Document**

**1. Exception Handling**

Exception handling in Python is a mechanism to handle errors during program execution without crashing the program. It uses try, except, else, and finally blocks to manage exceptions effectively.

Example:

def divide\_numbers(a, b):  
 try:  
 result = a / b  
 print("Result:", result)  
 except ZeroDivisionError:  
 print("Error: Cannot divide by zero.")  
 except TypeError:  
 print("Error: Please provide numbers only.")  
 else:  
 print("Division successful.")  
 finally:  
 print("Execution complete.")  
  
divide\_numbers(10, 2)  
divide\_numbers(10, 0)  
divide\_numbers(10, 'a')

• try: Defines the block of code to test.

• except: Defines a block of code to handle errors.

• else: Executes code if no exception occurs.

• finally: Executes code regardless of exception handling.

**2. File Handling**

File handling in Python allows reading, writing, and managing files. It is essential for applications that need to store or retrieve data persistently.

Modes of file handling:

• 'r': Read mode – default mode.

• 'w': Write mode – creates new file or overwrites existing.

• 'a': Append mode – adds content to end of file.

• 'b': Binary mode – read or write binary data.

Example:

with open("sample.txt", "w") as f:  
 f.write("Hello, Python File Handling!")  
  
with open("sample.txt", "r") as f:  
 print(f.read())

**3. Decorators**

Decorators in Python are functions that modify the behavior of other functions or methods. They are widely used in frameworks for logging, authentication, and performance monitoring.

Example:

def decorator\_function(func):  
 def wrapper():  
 print("Before the function call")  
 func()  
 print("After the function call")  
 return wrapper  
  
@decorator\_function  
def say\_hello():  
 print("Hello!")  
  
say\_hello()

**4. Unit Testing**

Unit testing is the process of testing individual components of a program to ensure they work as expected. Python provides the unittest module for writing and executing test cases.

Example:

import unittest  
  
def add(a, b):  
 return a + b  
  
class TestMath(unittest.TestCase):  
 def test\_add(self):  
 self.assertEqual(add(2, 3), 5)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

**5. Python Libraries**

Python has a vast ecosystem of libraries that extend its functionality. Libraries simplify complex tasks and save development time.

• NumPy: For numerical computations and array manipulations.

• Pandas: For data analysis and manipulation.

• Matplotlib: For data visualization.

• Scikit-learn: For machine learning tasks.

• TensorFlow and PyTorch: For deep learning applications.

• Requests: For handling HTTP requests.

• OS and Sys: For interacting with the operating system.

**6. List Comprehensions with Conditions**

List comprehensions provide a concise way to create lists in Python. They can include conditions for filtering elements.

Example:

labels = ["Even" if i%2==0 else "Odd" for i in range(1,6)]  
print(labels)

**7. Importance of Advanced Python Concepts**

Advanced concepts like exception handling, decorators, and unit testing make Python robust and production-ready. These concepts improve error management, allow modification of behavior without changing code, and ensure code reliability through automated testing.

• Exception handling prevents unexpected crashes.

• File handling enables data persistence.

• Decorators promote reusable and modular code.

• Unit testing ensures correctness and reliability of code.

• Libraries extend Python into diverse domains like AI, web development, and data science.