**PW SKILLS**

**EXCEPTION HANDLING**

Q1. What is an Exception in Python? Write the difference between Exceptions and Syntax Errors.

- An exception in Python is an event that occurs during the execution of a program that disrupts the normal flow of instructions. When an error occurs, Python generates an exception object to represent the error condition.

- Differences between Exceptions and Syntax Errors:

- Exceptions occur during the execution of the program, while Syntax Errors occur during the parsing of the code.

- Exceptions can be handled using try-except blocks, while Syntax Errors cannot be caught using try-except blocks and must be fixed in the code itself.

Q2. What happens when an exception is not handled? Explain with an example.

- When an exception is not handled, it propagates up the call stack until it reaches the top-level of the program. If it still remains unhandled, the program terminates and prints a traceback indicating the type of exception and the line of code where it occurred.

- Example:

```python

def divide(a, b):

return a / b

result = divide(5, 0)

```

Q3. Which Python statements are used to catch and handle exceptions? Explain with an example.

- Python statements used to catch and handle exceptions are `try`, `except`, `else`, `finally`.

- Example:

```python

try:

result = 10 / 0

except ZeroDivisionError:

print("Cannot divide by zero")

```

Q4. Explain with an example: try, except, else, finally, raise.

- `try`: The try block is used to enclose the code that might raise an exception.

- `except`: The except block is used to handle the exception raised in the try block.

- `else`: The else block is executed if no exception occurs in the try block.

- `finally`: The finally block is always executed regardless of whether an exception occurred or not.

- `raise`: The raise statement is used to manually raise an exception.

- Example:

```python

try:

result = 10 / 2

except ZeroDivisionError:

print("Cannot divide by zero")

else:

print("Division successful")

finally:

print("Execution completed")

```

Q5. What are Custom Exceptions in Python? Why do we need Custom Exceptions? Explain with an example.

- Custom Exceptions are user-defined exceptions that inherit from the built-in Exception class or its subclasses. They allow developers to create specific error types for their applications.

- We need Custom Exceptions to handle application-specific error scenarios and provide meaningful error messages to users.

- Example:

```python

class CustomError(Exception):

pass

def validate\_age(age):

if age < 18:

raise CustomError("Age must be 18 or above")

try:

validate\_age(15)

except CustomError as e:

print(e)

```

Q6. Create a custom exception class. Use this class to handle an exception.

- Example:

```python

class CustomError(Exception):

pass

try:

raise CustomError("This is a custom exception")

except CustomError as e:

print(e)

```