Errors and Exceptions in Python----->>>>

Errors are the problems in a program due to which the program will stop the execution. On the other hand, exceptions are raised when some internal events occur which changes the normal flow of the program. Two types of Error occurs in python.

1. Syntax errors
2. Logical errors (Exceptions)

Syntax errors--->>

When the proper syntax of the language is not followed then a syntax error is thrown.

logical errors(Exception)----->>

When in the runtime an error that occurs after passing the syntax test is called exception or logical type. For example, when we divide any number by zero then the ZeroDivisionError exception is raised, or when we import a module that does not exist then ImportError is raised.

IndexError---->>> When the wrong index of a list is retrieved.

AssertionError ---->

It occurs when the assert statement fails

AttributeError----->>>

It occurs when an attribute assignment is failed.

ImportError---->>>

It occurs when an imported module is not found.

KeyError ---->>>

It occurs when the key of the dictionary is not found.

NameError ------>>>

It occurs when the variable is not defined.

MemoryError ----->>>

It occurs when a program runs out of memory.

TypeError ----->>>

It occurs when a function and operation are applied in an incorrect type.

ATM Withdrawal: Write a function that simulates an ATM withdrawal. The function should check if the account balance is sufficient for the withdrawal amount and raise an exception if not. Handle scenarios where the input withdrawal amount is not a number or is negative.

```python
def bank(amount, withdraw):
    try:
        withdraw = int(withdraw)
        if withdraw <= 0:
            raise ValueError("Not a valid amount")
        if withdraw > amount:
            raise ValueError("Insufficient Balance")
        print(f"The amount {withdraw} has been deducted")
    except ValueError as e:
        print(e)

# Example usage
bank(1000, -10)  # Not a valid amount
bank(1000, 1500) # Insufficient Balance
bank(1000, 500)
bank(1000,"abc")  # The amount 500 has been deducted
```

```
Not a valid amount
Insufficient Balance
The amount 500 has been deducted
invalid literal for int() with base 10: 'abc'
```

User Login System: Create a function that simulates a user login system. It should raise an exception if the username or password is incorrect and handle cases where the input values are empty strings.

```python
def check(username, password):
  try:
    user=input("Enter the username: ")
    password1=int(input("Enter the password: "))
    if username == "" or password1 == "":
        raise ValueError("Username or password cannot be empty")
    if(username==user and password1==password):
      print("Correct password")
    else:
      raise ValueError("Incorrect username or password")
  except ValueError as e:
    print(e)
check("mehak", 123)
```

```
Enter the username: mehak
Enter the password: 123
Correct password
```

Online Shopping Cart: Write a function that adds items to an online shopping cart. Handle scenarios where the item is out of stock, the item ID is invalid, or the quantity requested is more than the available stock.

```
# prompt: Online Shopping Cart: Write a function that adds items to an
online shopping cart. Handle scenarios where the item is out of stock,
the item ID is invalid, or the quantity requested is more than the
available stock.

def add_to_cart(item_id, quantity):
  try:
    # Check if item is in stock
    if item_id not in stock:
      raise ValueError("Item out of stock")

    # Check if quantity is valid
    if quantity <= 0:
      raise ValueError("Invalid quantity")

    # Check if enough stock is available
    if quantity > stock[item_id]:
      raise ValueError("Not enough stock available")

    # Add item to cart
    cart[item_id] = quantity
    print(f"{quantity} of item {item_id} added to cart")
  except ValueError as e:
    print(e)

# Example usage
stock = {"apple": 10, "banana": 5, "orange": 3}
cart = {}

add_to_cart("apple", 2)  # 2 apples added to cart
add_to_cart("banana", 6)  # Not enough stock available
add_to_cart("kiwi", 1)  # Item out of stock
add_to_cart("orange", 0)  # Invalid quantity
for key,value in cart.items():
  print(f"{key}:{value}")

2 of item apple added to cart
Not enough stock available
Item out of stock
Invalid quantity
apple:2
```

Temperature Conversion: Implement a function that converts temperatures between Fahrenheit and Celsius. Raise an exception if the input is not a number and handle this error gracefully.

```
def convert(temp,unit):
  try:
    if(unit=="F"):
      temp=int(temp)*(9/5)+32
      print(f"The temperature in Fahrenheit is {temp}")
    elif(unit=="C"):
      temp=(temp-32)*(5/9)
      print(f"The temperature in Celsius is {temp}")
    if(temp<0):
      raise ValueError("Temperature cannot be negative")
  except ValueError as e:
    print(e)
convert(23,"F")
```

```
The temperature in Fahrenheit is 73.4
```

Student Grades: Write a function that calculates the average grade of a list of student grades. Handle scenarios where the list might be empty, the grades might not be valid numbers, or some grades might be missing.

```
def student(grades):
  try:
    grade_asked=float(input("Enter the grade: "))
    if grade_asked not in grades:
      raise ValueError("Grade not found")
    if(len(grades)==0):
      raise ValueError("List is empty")
    if(grade_asked in grades):
      print(f"The grade is {grade_asked}")
  except ValueError as e:
    print(e)
grades1=[1,2,3,4,5,6]
grades2=[8,9,5,6]
student(grades1)
student(grades2)
```

```
Enter the grade: abc
could not convert string to float: 'abc'
Enter the grade: ss
could not convert string to float: 'ss'
```

Email Sending Service: Simulate an email sending service function that raises an exception if the email address is invalid, the server is unreachable, or the sending fails. Handle these exceptions and provide meaningful error messages.

```
def send_email(email):
  try:
   if not isinstance(email, str):
       raise ValueError("Email must be a string")
   if(email==" "):
    raise ValueError("Email cannot be empty")
   email_check=email.split('@')[-1]
   if(email_check=="gmail.com" or email_check=="yahoo.com" or
email_check=="hotmail.com"):
      print("Email sent successfully")
    else:
      raise ValueError("Email is invalid")

  except ValueError as e:
    print(e)

send_email("mehak.co.in")
```

```
Email is invalid
```

Online Reservation System: Create a function that makes an online reservation for a hotel room. Handle cases where the room is already booked, the reservation date is in the past, or the input date format is incorrect.

```
from datetime import date

def hotel(room, date_required):
    try:
        booked_rooms = ["101", "102", "103"]

        # Check if room is already booked
        if room in booked_rooms:
            raise ValueError("Room already booked")

        # Check if reservation date is in the past
        if date_required < date.today():
            raise ValueError("Reservation date must be today or in the
future")

        # Check if date_required is a valid datetime.date object
        if not isinstance(date_required, date):
            raise ValueError("Invalid date format")

        # If all checks pass, book the room
        print("Room is booked")

    except ValueError as e:
        print(e)

# Example usage
hotel("105", date(2024, 7, 1))  # Correct input using date object
```

```
Room is booked
```

Bank Account Transfer: Write a function that transfers money between two bank accounts. Raise an exception if the source account has insufficient funds or if the transfer amount is invalid. Handle these exceptions appropriately.

```
def transfer(source, destination, amount,required_amount):
  try:
    if not isinstance(source,str):
       raise ValueError("Invalid input")
    if not isinstance(destination,str) :
     raise ValueError("Invalid input")
    if not isinstance(amount,(int,float)):
     raise ValueError("Invalid input")
    if(source==destination):
      raise ValueError("Source and destination accounts cannot be the
same")
    if required_amount<=0:
      raise ValueError("Amount cannot be negative")
    if(amount<required_amount):
      raise ValueError("Insufficient balance")
    else:
      print("Value has been tranfered")
      amount=amount-required_amount
      print(f"The amount {amount} is left in account")
  except ValueError as e:
     print(e)
transfer("abc", "def", 1000,10)
```

```
Value has been tranfered
The amount 990 is left in account
```

File Parsing: Write a function that parses a configuration file. Handle scenarios where the file is missing, the file format is incorrect, or required configuration keys are missing.

```
import json

def parse_config(file_path, required_keys=None):
    if required_keys is None:
        required_keys = []

    try:
        # Try to open and load the JSON configuration file
        with open(file_path, 'r') as f:
            config = json.load(f)
    except FileNotFoundError:
        raise ValueError("Config file not found")
    except json.JSONDecodeError:
        raise ValueError("Invalid JSON format")

    # Check for required keys
    for key in required_keys:
        if key not in config:
            raise ValueError(f"Missing required configuration key:
{key}")
```

```
    return config

# Example usage
required_keys = ['host', 'port', 'username', 'password']
try:
    name = parse_config('config.json', required_keys)
    # Use the parsed configuration
    print("Configuration successfully parsed:", name)
except ValueError as e:
    print(f"Error parsing config file: {e}")
```

```
Configuration successfully parsed: {'host': 'localhost', 'port': 8080,
'username': 'admin', 'password': 'secret'}
```

Currency Converter: Create a function that converts an amount from one currency to another using a predefined exchange rate. Handle scenarios where the input amount is not a number, the currency code is invalid, or the exchange rate is missing.

```
def convert_currency(amount, from_currency, to_currency,
exchange_rates):
    try:
        # Check if amount is a number
        if not isinstance(amount, (int, float)):
            raise ValueError("Amount must be a number")

        # Check if from_currency and to_currency are valid
        if from_currency not in exchange_rates:
            raise ValueError(f"Invalid from_currency code:
{from_currency}")
        if to_currency not in exchange_rates:
            raise ValueError(f"Invalid to_currency code:
{to_currency}")

        # Check if exchange rate exists
        if to_currency not in exchange_rates[from_currency]:
            raise ValueError(f"Exchange rate from {from_currency} to
{to_currency} is missing")

        # Perform the conversion
        rate = exchange_rates[from_currency][to_currency]
        converted_amount = amount * rate
        return converted_amount

    except ValueError as e:
        return str(e)

# Predefined exchange rates
exchange_rates = {
    'USD': {'EUR': 0.85, 'JPY': 110.0},
    'EUR': {'USD': 1.18, 'JPY': 130.0},
    'JPY': {'USD': 0.0091, 'EUR': 0.0077}
}

# Example usage
amount_in_usd = 100
converted_amount = convert_currency(amount_in_usd, 'USD', 'EUR',
exchange_rates)
if isinstance(converted_amount, (int, float)):
    print(f"{amount_in_usd} USD is {converted_amount:.2f} EUR")
else:
    print(f"Error: {converted_amount}")

100 USD is 85.00 EUR
```

Product Price Checker: Write a function that checks the price of a product from an online store's API. Handle cases where the product ID is invalid, the API is unreachable, or the response data is malformed.

```
# prompt: Product Price Checker: Write a function that checks the
price of a product from an online store's API. Handle cases where the
product ID is invalid, the API is unreachable, or the response data is
malformed.

import requests

def check_price(product_id):
    try:
        # Make a request to the API
        response =
requests.get(f"https://api.example.com/products/{product_id}")

        # Check if the request was successful
        if response.status_code != 200:
            raise ValueError(f"Invalid product ID: {product_id}")

        # Parse the JSON response
        data = response.json()

        # Check if the data is valid
        if "price" not in data:
            raise ValueError("Malformed response data")

        # Return the product price
        return data["price"]

    except requests.exceptions.RequestException as e:
        raise ValueError(f"API unreachable: {e}")

# Example usage
try:
    product_id = "12345"
    price = check_price(product_id)
    print(f"The price of product {product_id} is {price}")
except ValueError as e:
    print(f"Error: {e}")
```

```
Error: API unreachable: HTTPSConnectionPool(host='api.example.com',
port=443): Max retries exceeded with url: /products/12345 (Caused by
NameResolutionError("<urllib3.connection.HTTPSConnection object at
0x7ca7403de590>: Failed to resolve 'api.example.com' ([Errno -2] Name
or service not known)"))
```

Flight Booking System: Implement a function that books a flight ticket. Handle scenarios where the flight is fully booked, the passenger details are incomplete, or the payment processing fails.

```
import random

def book_flight(flight_number, passenger_name, seat_class,
payment_info):
    # Simulate random success or failure of booking (for
demonstration)
    booking_successful = random.choice([True, False])

    try:
        # Check if passenger details are complete
        if not passenger_name:
            raise ValueError("Passenger name is required")

        # Check if payment info is complete
        if not payment_info:
            raise ValueError("Payment information is required")

        # Simulate fully booked scenario (30% chance)
        if random.random() < 0.3:
            raise ValueError("Flight is fully booked")

        # Simulate payment processing failure
        if not booking_successful:
            raise ValueError("Payment processing failed")

        # Return booking confirmation
        booking_reference = f"{flight_number}-{random.randint(1000,
9999)}"
        return f"Booking successful! Your booking reference is
{booking_reference}"

    except ValueError as e:
        return str(e)

# Example usage
```

```
try:
    flight_number = "ABC123"
    passenger_name = "John Doe"
    seat_class = "Economy"
    payment_info = {"card_number": "1234 5678 9012 3456",
"expiry_date": "12/23", "cvv": "123"}

    booking_result = book_flight(flight_number, passenger_name,
seat_class, payment_info)
    print(booking_result)
except ValueError as e:
    print(f"Booking failed: {e}")
```

```
Payment processing failed
```

Quiz Application: Write a function that grades a multiple-choice quiz. Handle cases where the answer key is missing, the student's answers contain invalid choices, or the quiz data is corrupted.

```
def grade_quiz(answer_key, student_answers):
    try:
        # Check if answer key is provided and is a dictionary
        if not isinstance(answer_key, dict):
            raise ValueError("Answer key is missing or invalid")

        # Check if student answers are provided and is a dictionary
        if not isinstance(student_answers, dict):
            raise ValueError("Student answers are missing or invalid")
        for question, answer in answer_key.items():
            if student_answers[question] not in answer:
              raise ValueError("Invalid answer")
        return "Quiz graded successfully"
    except ValueError as e:
        return str(e)

answer_key = {
        "Q1": "B",
        "Q2": "A",
        "Q3": "C"
    }

student_answers = {
        "Q1": "B",
        "Q2": "A",
        "Q3": "C"
    }
result = grade_quiz(answer_key, student_answers)
print(result)
```

```
Quiz graded successfully
```

Library Management System: Create a function that issues a book to a library member. Handle scenarios where the book is not available, the member's subscription is expired, or the member ID is invalid.

```python
def issue_book(member_id, book_title, available_books, 
member_records):
    try:
        # Check if member ID is valid
        if member_id not in member_records:
            raise ValueError("Invalid member ID")

        # Check if member's subscription is active
        if not member_records[member_id]["subscription_active"]:
            raise ValueError("Member's subscription is expired")

        # Check if the book is available
        if book_title not in available_books or 
available_books[book_title] <= 0:
            raise ValueError("Book is not available")

        # Issue the book to the member
        available_books[book_title] -= 1
        return f"Book '{book_title}' issued to member ID: {member_id}"

    except ValueError as e:
        return f"Issue book failed: {e}"

# Example usage
try:
    # Simulated data
    available_books = {
        "Python Programming": 3,
        "Introduction to Machine Learning": 1,
        "Data Structures and Algorithms": 0
    }

    member_records = {
        "001": {"name": "Alice", "subscription_active": True},
        "002": {"name": "Bob", "subscription_active": False}
    }

    member_id = "001"
    book_title = "Python Programming"

    result = issue_book(member_id, book_title, available_books, 
member_records)
    print(result)
except ValueError as e:
    print(f"Error: {e}")
```

```
Book 'Python Programming' issued to member ID: 001
```

Inventory Management: Write a function that updates the inventory levels of products in a warehouse. Handle cases where the product ID is invalid, the update quantity is negative, or the database connection fails.

```python
import random  # For simulating database connection failure

def update_inventory(product_id, update_quantity):
    try:
        # Simulate random database connection failure (for 
demonstration)
        if random.random() < 0.1:  # 10% chance of database connection 
failure
            raise ConnectionError("Database connection failed")

        # Check if product ID is valid (simulated validation)
        valid_product_ids = ["P001", "P002", "P003"]
        if product_id not in valid_product_ids:
            raise ValueError("Invalid product ID")

        # Check if update quantity is negative
        if update_quantity < 0:
            raise ValueError("Update quantity cannot be negative")

        # Update inventory (simulate database update)
        # Normally here you would perform database operations to 
update inventory levels
        # This is simulated by printing the update
        print(f"Updating inventory for product ID {product_id} by 
{update_quantity}")

        return "Inventory updated successfully"

    except ConnectionError as e:
        return f"Failed to update inventory: {e}"

    except ValueError as e:
        return f"Failed to update inventory: {e}"

# Example usage
try:
    product_id = "P002"
    update_quantity = 10

    result = update_inventory(product_id, update_quantity)
    print(result)
except ValueError as e:
    print(f"Error: {e}")
```

```
except ConnectionError as e:
    print(f"Error: {e}")

Updating inventory for product ID P002 by 10
Inventory updated successfully
```

File Handling---->>>>

Exception handling in Python is a mechanism to respond to runtime errors, preventing the program from crashing and allowing the program to handle errors gracefully. It helps in debugging, maintaining clean code, and providing user-friendly error messages.

Key Concepts

1. Exception: An exception is an error that occurs during the execution of a program. When an exception is raised, the normal flow of the program is interrupted.
2. Try Block: The code that might raise an exception is placed inside a try block.
3. Except Block: The code that handles the exception is placed inside an except block.
4. Else Block: The code inside the else block is executed if no exceptions are raised.
5. Finally Block: The code inside the finally block is executed regardless of whether an exception is raised or not.
6. Raise: Used to raise an exception manually.

Common Built-in Exceptions

IndexError

KeyError

ValueError

TypeError

ZeroDivisionError

FileNotFoundError

IOError

ImportError

AttributeError

RuntimeError

Your program needs to load user settings from a file named settings.txt. Write a function to read the entire content of this file.

```python
def read_settings_file(file_path):
    try:
        with open(file_path, 'r') as file:
            content = file.read()
            return content
    except FileNotFoundError:
        return f"Error: The file '{file_path}' was not found."
    except IOError as e:
        return f"Error: Failed to read the file '{file_path}'. {e}"

# Example usage
file_path = 'settings.txt'
try:
    settings_content = read_settings_file(file_path)
    print("Settings file content:")
    print(settings_content)
except Exception as e:
    print(f"Error: {e}")
```

```
Settings file content:
Hello world
```

Your application generates a report. Write a function to save the string "Monthly Report: June" to a file named report.txt.

```python
def save_report():
    try:
        # Open the file named report.txt in write mode ('w')
        with open('report.txt', 'w') as file:
            file.write("Monthly Report: June")
        return "Report saved successfully."
    except IOError as e:
        return f"Error: Failed to save report. {e}"

# Example usage
try:
    result = save_report()
    print(result)
except Exception as e:
    print(f"Error: {e}")
```

```
Report saved successfully.
```

A web server needs to log user access times. Write a script to append the string "User accessed at 2023-06-15 10:00 AM" to a file named access.log.

```python
from datetime import datetime

def log_user_access():
    try:
        # Get current timestamp in the specified format
        current_time = datetime.now().strftime("%Y-%m-%d %I:%M %p")

        # Prepare the log message
        log_message = f"User accessed at {current_time}\n"

        # Append to access.log file
        with open('access.log', 'a') as file:
            file.write(log_message)

        return "Access logged successfully."

    except IOError as e:
        return f"Error: Failed to log access. {e}"

# Example usage
try:
    result = log_user_access()
    print(result)
except Exception as e:
    print(f"Error: {e}")
```

```
Access logged successfully.
```

Your script needs to process a data file named data.csv. Write a function that attempts to read this file and handles the FileNotFoundError by creating an empty file with the same name.

```
def process_data_file(file_path):
    try:
        # Attempt to read the file
        with open(file_path, 'r') as file:
            data = file.read()
            return f"File content:\n{data}"

    except FileNotFoundError:
        # Handle FileNotFoundError by creating an empty file
        with open(file_path, 'w') as file:
            file.write("")
        return f"File '{file_path}' not found. Created an empty file."

    except IOError as e:
        return f"Error: Failed to read or create the file
'{file_path}'. {e}"

# Example usage
file_path = 'report.txt'
try:
    result = process_data_file(file_path)
    print(result)
```

```
except Exception as e:
    print(f"Error: {e}")

File content:
Monthly Report: June
```

Your image processing software needs to read raw binary image data from a file named image.bin. Write a function to read and print the first 10 bytes.

```
def read_first_10_bytes(file_path):
    try:
        # Open the file in binary mode ('rb')
        with open(file_path, 'rb') as file:
            # Read the first 10 bytes
            first_10_bytes = file.read(10)
            # Print the first 10 bytes as hexadecimal representation
            print(f"First 10 bytes of {file_path}:
{first_10_bytes.upper()}")

    except FileNotFoundError:
        print(f"Error: The file '{file_path}' was not found.")
    except IOError as e:
        print(f"Error: Failed to read the file '{file_path}'. {e}")

# Example usage
file_path = 'first_10_bytes.bin'
read_first_10_bytes(file_path)

First 10 bytes of first_10_bytes.bin: b'HELLO WORL'
```

Before processing log data, your script needs to ensure the log file logfile.log exists. Write a function to check for the file's existence and create it if it doesn't exist

```
import os

def ensure_log_file_exists(file_path):
    try:
        # Check if the file exists
        if not os.path.exists(file_path):
            # If the file does not exist, create it
            with open(file_path, 'w') as file:
                pass
            print(f"File '{file_path}' created.")
        else:
            print(f"File '{file_path}' already exists.")

    except IOError as e:
        print(f"Error: Unable to create the file '{file_path}'. {e}")
```

```
# Example usage
log_file_path = 'logfile.log'
ensure_log_file_exists(log_file_path)
```

```
File 'logfile.log' created.
```

Your monitoring tool needs to count the number of entries in a log file named server.log. Implement a function to count and return the number of lines.

```
import os

def count_log_entries(file_path):
    try:
        if not os.path.exists(file_path):
            # If the file does not exist, create it
            with open(file_path, 'w') as file:
                pass
            print(f"File '{file_path}' created.")

        # Open the file in read mode and count the lines
        with open(file_path, 'r') as file:
            count = 0
            for line in file:
                count += 1
        return count
    except IOError as e:
        print(f"Error: Unable to read the file '{file_path}'. {e}")
        return 0

# Example usage
log_file_path = 'server.log'
num_entries = count_log_entries(log_file_path)
print(f"Number of entries in '{log_file_path}': {num_entries}")
```

```
Number of entries in 'server.log': 2
```

Activity on basis of random module---->>>>

Assignment using Random module: Simulate Rock-Paper-Scissors game using the knowledge that you have acquired till now..

```
# prompt: Assignment using Random module: Simulate Rock-Paper-Scissors
game using the knowledge that you have acquired till now.

import random

def play_rock_paper_scissors():
    choices = ["Rock", "Paper", "Scissors"]

    # Get player's choice
    player_choice = input("Enter your choice (Rock, Paper, or
Scissors): ").capitalize()
    while player_choice not in choices:
        player_choice = input("Invalid choice. Please enter Rock,
Paper, or Scissors: ").capitalize()

    # Get computer's choice
    computer_choice = random.choice(choices)

    # Determine the winner
    if player_choice == computer_choice:
        print("Tie!")
    elif( (player_choice == "Rock" and computer_choice == "Scissors")
or
         (player_choice == "Paper" and computer_choice == "Rock") or
         (player_choice == "Scissors" and computer_choice ==
"Paper")):
         print("You win!")
    else:
        print("You lose!")

# Play the game
play_rock_paper_scissors()
```

```
Enter your choice (Rock, Paper, or Scissors): paper
You win!
```