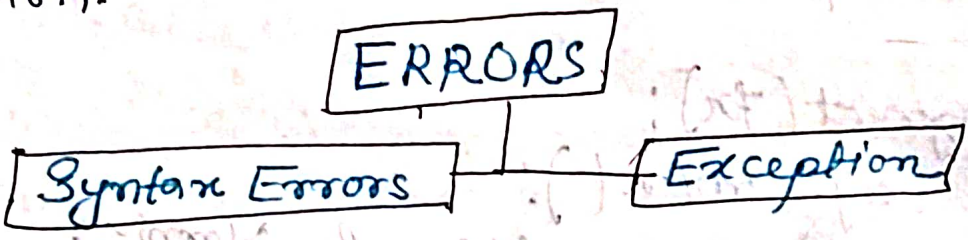
Exception Handling

Error: Error are the problems in a program due to which the program will stop the execution.



Syntax Errors:—

```
amount = 10000
if (amount > 2999)
print ("You are eligible to purchase
                Dsa Self Paced")
```

What are exceptions?

⊙ Exceptions are unexpected or erroneous events that occur during the ~~excep~~ execution of a program, disrupting the normal flow of execution.

⊙ They can arise due to various reasons such as invalid input, file not found, division by zero, etc.

Syntax Errors:-

```
marks = 10000
a = marks / 0
print(a)
```

Why use exception handling?

⊙ Exception handling allows programmers to anticipate and gracefully handle these unexpected events, preventing program crashes and improving code reliability.

⊙ It enables the program to recover from errors and continue executing or provide meaningful error messages to users.

Common Types of Exceptions

SyntaxError:
Occurs when the syntax of the code is incorrect, such as missing parentheses or invalid indentation.

IndentationError:
Arises when there are issues with the indentation of code blocks, which is crucial in Python's syntax.

NameError:
ValueError:
FileNotFoundError
ZeroDivisionError
IndexError
KeyError
AttributeError.

Try and Except Statement – Catching Exceptions

```
a = [1,2,3]
try:
    print("second element=",(a[1]))
    print("fourth element=",(a[3]))
except:
    print("An error occurred")
```

Output

second element = 2
An error occurred.

Catching Specific Exception

```
x=5
y="hello"
try:
    z=x+y
except TypeError:
    print("Error: Cannot add an int
                and a str")
```

```
def fun(a):
    if a<4:
        b = a/(a-3)
    print("value of b =",b)

try:
    fun(3)
    fun(5)
except ZeroDivisionError:
    print("ZeroDivisionError Occurred
                and Handled")
```

```
except NameError:
    print ("NameError Occurred and Handled")
```

Try with Else Clause

In Python, we can also use the else clause on the try-except block which must be present after all the except clauses.

```
def Abc(a, b)
    try:
        c = ((a+b) / (a-b))
    except ZeroDivisionError:
        print ("a/b result in 0")
    else:
        print (c)
```

call

```
Abc(2.0, 3.0)
Abc (3.0, 3.0)
```

Finally Keyword

Python provides a keyword finally which is always executed after the try and except blocks.

```
try:
    k = 5 // 0
    print(k)
except ZeroDivisionError:
    print ("Can't divide by zero")
finally:
    print ("This is always executed")
```

Raising Exception

The raise statement allows the programmer to force a specific exception to occur. The sole argument in raise indicates the exception to be raised.

```
try:
    raise NameError("Hi there")
except NameError:
    print("An exception")
    raise
```

```
def validate_input(value):
    if value<0:
        raise ValueError("Input value must be positive")

try:
    validate_input(-5)
except ValueError as e:
    print("Error:", e)
```

Output

Error: Input value must be positive.

Practice

1)
```
def divide(x,y):
    try:
        result = x//y
        print("Yeah! Your answer is:", result)
    except ZeroDivisionError:
        print("Sorry! You are dividing by zero").
```

Call

```
divide(3,2)
```

2) def divide(x,y):
    try:
        result = x//y
        print("Yeah! Your answer is:", result)
    except Exception as e:
        print("The error is:", e)

divide(3, "GFG") divide(3,0)

3) def divide_number(x,y):
    try:
        result = x/y
        print("Result:", result)
    except (ZeroDivisionError, TypeError):
        print("Error: Division by zero or invalid value encountered!")

4) Factorial Calculator with Exception Handling.

def factorial(n):
    if n==0:
        return 1
    else:
        return n * factorial(n-1)

try:
    num = int(input("Enter a non-negative integer to calculate its factorial:"))
    if num < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    print(f"Factorial of {num} is {factorial(num)}")
except ValueError as e:
    print(f"Error: {e}")

Greatest Common Divisor (GCD) Calculator with Exception Handling.

```
def gcd(a, b)
    while b:
        a, b = b, a % b
    return a

try:
    num1 = int(input("Enter first positive integer: "))
    num2 = int(input("Enter second positive integer: "))
    if (num1<=0 or num2<=0)
        raise ValueError("Both number should be pos
    print(f"Gcd of {num1} and {num2} is {gcd(num1, num2)}")
except ValueError as e
    print(f"Error: {e}")
```

Calculator with Exception Handling.

```
def add(a,b):
    return a+b
def sub(a,b):
    return a-b
def mul(a,b):
    return a*b
def div(a,b):
    return a/b
print("Select option\n 1. Addition\n 2. Subst
    if\n 3. Multiplication\n 4. Division")
    if choice == '5':
        print("Stoppeal Calculation")
        break
while True:
    choice = input("Enter choice (1/2/3/4/5
    if choice in ('1','2','3','4','5'):
        try:
            n1 = float(input("Enter first numbe
            n2 = float(input("Enter second numb
```

```
except ValueError:
    print("Invalid input. Please enter
          a number.")
    continue
if choice == '1':
    print(n1, "+", n2, "=", add(n1, n2))
elif choice == '2':
    print(n1, "-", n2, "=", sub(n1, n2))
elif choice == '3':
    print(n1, "*", n2, "=", mul(n1, n2))
elif choice == '4':
    print(n1, "/", n2, "=", div(n1, n2))
else:
    print("Invalid Input")
```

~~Email Validator with Exception Handling~~

Fibonacci Sequence Generator with Exception Handling.

```
def fibonacci_sequence(n):
    if n <= 0:
        raise ValueError("Input must be a positive integer")
    sequence = [0, 1]
    while len(sequence) < n:
        next_num = sequence[-1] + sequence[-2]
        sequence.append(next_num)
    return sequence
#def fibonacci_sequence_generator():
try:
    n = int(input("Enter the number of terms for the Fibonacci sequence: "))
    sequence = fibonacci_sequence(n)
    print(f"Fibonacci sequence up to {n} terms: {sequence}")
except ValueError as e:
    print(f"Error: {e}. Please enter a valid positive integer.")
except Exception as e:
    print(f"An error occurred: {e}")
```

Temperature converter with Exception Handling.

```
def celsius_to_fahrenheit(celsius):
    return (celsius * 9/5) + 32
def fahrenheit_to_celsius(fahrenheit):
    return (fahrenheit - 32) * 5/9
def temperature_converter():
    try:
        choice = int(input("Choose conversion:
            Celsius to Fahrenheit\n2. Fahrenheit
            to Celsius\n Enter your choice:"
        if choice not in [1,2]
            raise ValueError("Invalid choice")
        if choice == 1:
            celsius = float(input("Enter tempera
                in celsius:"))
            fahrenheit = celsius_to_fahrenheit(
                (celsius)
            print(f"{celsius}°C is equa
                to {celsius}°C")
    except ValueError as e:
    elif choice == 2
        fahrenheit = float(input("Ent
            temperature in fahrenhe
        celsius = fahrenheit_to_celsius
            (fahrenheit)
        print(f"{fahrenheit}°F is equ
            to {celsius}°C")
    except ValueError as e:
        print(f"Error: {e}. please ente
            a valid input.")
    except Exception as e:
        print(f"An unexpected error
```

C → 0 - 100
F → 32 - 212 → x

$$\frac{100-x}{100-0} = \frac{212-32}{212-32}$$

$$\frac{C}{5} = \frac{(F-32)}{9}$$

$$C = \frac{5(F-32)}{9}$$

$$F - 32 = \frac{9}{5}C + 32$$