Lab Work — Exception Handling in Python

Objective

This lab introduces Python exception handling techniques to manage runtime errors in engineering applications. You will practice using try, except, specific error types, raise, and assert to prevent program crashes and display meaningful messages. Applying these techniques improves program reliability and safety in real-world scenarios.

Task 1: Type Error (Add int and string)

Step 1: Run This Code

In [1]:
```
a = 10
b = "5"
result = a + b
print(result)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[1], line 3
      1 a = 10
      2 b = "5"
----> 3 result = a + b
      4 print(result)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Fixed Code:

In [15]:
```
a = 10
b = "5"
try:
    result = a + b
    print(result)
except TypeError:
    print("Type Error Cannot add an integer and a string")
```

```
Type Error Cannot add an integer and a string
```

Task 2: Division by Zero

Step 1: Run This Code

In [3]:
```
x = 20
y = 0
result = x / y
print(result)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[3], line 3
      1 x = 20
      2 y = 0
----> 3 result = x / y
      4 print(result)

ZeroDivisionError: division by zero
```

Fixed Code:

In [4]:
```
x = 20
y = 0
try:
    result = x / y
    print(result)
except ZeroDivisionError:
    print("Error: Division by zero is not allowed")
```

```
Error: Division by zero is not allowed
```

Task 3: List Index Out of Range

Step 1: Run This Code

In [5]:
```
numbers = [10, 20, 30]
print(numbers[5])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[5], line 2
      1 numbers = [10, 20, 30]
----> 2 print(numbers[5])

IndexError: list index out of range
```

Fixed Code:

In [16]:
```
numbers = [10, 20, 30]
try:
    print(numbers[5])
except IndexError:
    print("Error! Index is out of range")
```

```
Error! Index is out of range
```

Task 4: Logical Error Using raise

Step 1: Run This Code

In [7]:
```
age = -5
print("Age:", age)
```

```
Age: -5
```

Fixed Code:

In [8]:
```
age = -5

if age < 0:
    raise Exception("Invalid age value")

print("Age:", age)
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
Cell In[8], line 4
      1 age = -5
      3 if age < 0:
----> 4     raise Exception("Invalid age value")
      6 print("Age:", age)

Exception: Invalid age value
```

Task 5: Assert Failure

Step 1: Run This Code

In [9]:
```
total_marks = 0
assert total_marks > 0
print("Total marks:", total_marks)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[9], line 2
      1 total_marks = 0
----> 2 assert total_marks > 0
      3 print("Total marks:", total_marks)

AssertionError:
```

Fixed Code:

In [12]:
```
total_marks = 0

try:
    assert total_marks > 0
    print("Total marks:", total_marks)
except AssertionError:
    print("Assertion Error: Total marks must be greater than zero")
```

```
Assertion Error: Total marks must be greater than zero
```

Task 6: Assert Failure

In [13]:
```
try:
```

```
    x = int("2")
except ValueError:
    print("Error")
else:
    print("No error occurred")
finally:
    print("Always executed")
```

```
No error occurred
Always executed
```

In [14]:
```
try:
    x = int("abc")
except ValueError:
    print("Error")
else:
    print("No error occurred")
finally:
    print("Always executed")
```

```
Error
Always executed
```