


ABDELGHAFOR'S VIRTUAL INTERNSHIP

# PYTHON PROGRAM

SESSION (5)

PREPARED BY : MARK KOSTANTINE



# AGENDA OVERVIEW

03

TRY EXCEPT

07

TASKS

10

QUESTIONS



# TRY EXCEPT

- The `try` block lets you test a block of code for errors.
- The `except` block lets you handle the error.
- The `else` block lets you execute code when there is no error.
- The `finally` block lets you execute code, regardless of the result of the try- and except blocks.

## EXCEPTION HANDLING

When an error occurs, or exception as we call it, Python will normally stop and generate an error message. These exceptions can be handled using the try statement

```
try:  
    print(x)  
except:  
    print("An exception occurred")
```

# MANY EXCEPTIONS

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error

```
try:
    print(x)
except NameError:
    print("Variable x is not defined")
except:
    print("Something else went wrong")
```

**Note:** There's another types of exceptions like `TypeError`, `DividedByZero` ..etc

# ELSE

You can use the `else` keyword to define a block of code to be executed if no errors were raised

```
try:
    print("Hello")
except:
    print("Something went wrong")
else:
    print("Nothing went wrong")
```

# FINALLY

The `finally` block, if specified, will be executed regardless if the try block raises an error or not.

```
try:
    print(x)
except:
    print("Something went wrong")
finally:
    print("The 'try except' is finished")
```

# RAISE AN EXCEPTION

As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the `raise` keyword.

```
x = -1
```

```
if x < 0:
```

```
    raise Exception("Sorry, no numbers below zero")
```

The `raise` keyword is used to raise an exception.

You can define what kind of error to raise, and the text to print to the user.

```
x = "hello"
```

```
if not type(x) is int:
```

```
    raise TypeError("Only integers are allowed")
```

# TASKS

## Beginner Level :

- **Basic Calculator with Multiple Exceptions:** Create a program that asks the user to input two numbers and an operation (addition, subtraction, etc.). Handle cases where the user enters invalid numbers, chooses an invalid operation, or tries to divide by zero.
- **List Element Retrieval:** Write a function that takes a list and asks the user for an index to retrieve an element. Handle cases where the index is out of range or the input is not an integer. Print an appropriate error message in each case.
- **Dictionary Key Lookup with Basic Error Handling:** Ask the user to input a key to look up in a pre-defined dictionary. Use try-except to handle cases where the key does not exist. Additionally, handle cases where the user input is not valid, such as providing an empty string or a non-string data type (like a number).



# TASKS

## Intermediate Level :

- **Complex Dictionary and List Handling:** Create a program that takes a dictionary where the values are lists. Ask the user for a key and an index to retrieve a specific list element. Handle errors for invalid keys, invalid indexes, and improper data types for the input.
- **Multiple File Operations:** Design a function that asks the user for a filename, tries to open and read the file, and then processes its content. Use try-except blocks to handle the cases where the file does not exist, the file is empty, or the content cannot be processed due to type or formatting issues.
- **Class Method with Error Handling:** Write a class with a method that processes a list of numbers and performs operations like finding the maximum, average, and sum. Use try-except blocks to handle cases where the list is empty, contains non-numeric elements, or other unexpected issues arise.





# TASKS

## Advanced Level :

- **Nested Error Handling in Multiple Operations:** Develop a program that performs multiple tasks like file reading, dictionary lookups, and mathematical operations within a single try block. Use nested try-except blocks to handle different exceptions at various levels, such as handling errors in file reading, invalid dictionary keys, and division by zero in mathematical operations.
- **Custom Exception Handling in a Class with Polymorphism:** Create a base class and several subclasses that perform different calculations. Implement custom exceptions that are raised when certain conditions are not met (e.g., division by zero, invalid input types). Ensure the exceptions are caught at the appropriate level of the class hierarchy and demonstrate polymorphism in handling these exceptions.
- **Error Handling in Complex List and Dictionary Operations:** Write a program that processes a large dictionary where the values are lists of integers. Ask the user for a key to access a list and then for an index to access an element within that list. Use try-except to handle errors such as key not existing, index being out of range, or the list containing non-integer elements. Ensure the program continues to run smoothly despite these errors.



# ANY QUESTIONS ?



PYTHON PROGRAM

# THANK YOU

UPCOMING NEXT WEEK : SESSION (6)