Day 4 Tutorial : Error Handling.

Task 1: Knowledge and understanding (Exception handling)

1. **What is the difference between a runtime error and a syntax error?**

   *The difference between runtime errors and syntax errors is when they occur. Syntax errors are detected before the program is run, while runtime errors are detected while the program is running.

   *Another difference is that syntax errors are typically caused by errors in the structure of the program's code, while runtime errors can be caused by a variety of factors, including errors in the program 's logic, invalid user input, and hardware failures.

2. **What is an exception and why is it important?**

   An exception is an event that occurs during the exception of a program that disrupts the normal flow of instructions. Exception can be caused by a variety of factors, such as

   - Errors in the program's logic
   - Invalid user input
   - Hardware failures
   - Unforeseen circumstances

   Exception are important because they allow programmers to handle errors gracefully and prevent their programs from crashing.

3. **Explain the purpose of the try and except blocks in exception handling.**

   - The 'try' and 'except' blocks in exception handling are used to handle errors gracefully and prevent programs from crashing.
   - The 'try' block contains the code that is executed first. If an exception occurs while the code in the 'try' block is executing, the program will jump to the appropriate 'except' block.

4.Provide 3 built-in exceptions.

- **Name error:** This exception is raised when a name or variable is not defined.

```
# NameError example:

try:
   print(my_variable)
except NameError:
   print("Variable `my_variable` is not defined.")
```

- **Type error:** This exception is raised when an operation is applied to an object of an inappropriate type

  ```
  # TypeError example:

  try:
     10 + "hello"
  except TypeError:
     print("Unsupported operation: cannot add a string to an integer.")
  ```

- **Zerodivision error:** A zero division error is a type of arithmetic error that occurs when a number is divided by zero.

  ```
  # Zero division error example:
  try:
     10 / 0
  except ZeroDivisionError:
     print("Division by zero error.")
  ```

5. What is the purpose of the finally block in exception handling, and when is it executed?

The finally block in exception handling is used to execute code regardless of whether or not an exception is thrown. It is typically used to clean up resources, such as closing files or database connections.

```
try:
   file = open("myfile.txt", "r")
except IOError:
   print("I/O error.")
finally:
   file.close()
```

6. Describe the use of the else block in a try and except structure.

The else block in a try-except structure is used to execute code if no exception is thrown in the try block. This can be useful for performing clean up tasks or other operations that need to be executed only if the code in the try block completes successfully.

```
 try:

   file = open("myfile.txt", "r")

   contents = file.read()

except IOError:

   print("I/O error.")

else:

 file.close()
```

```
  print(contents)
```

7. Can you catch and handle multiple exceptions in a single except block?

try:

number = float(input(&quot;Enter a floating-point number: &quot;))

print(f&quot;You entered: {number}&quot;)

except ValueError:

print(&quot;Invalid input. Please enter a valid floating-point number.&quot;)

Yes, you can catch and handle multiple exceptions in a single except block. To do this, you specify the exception types in a tuple, separated by commas.

For example:

```
try:
    number = float(input("Enter a floating-point number: "))
    print(f"You entered: {number}")
except (ValueError, TypeError):
    print("Invalid input. Please enter a valid floating-point number.")
```

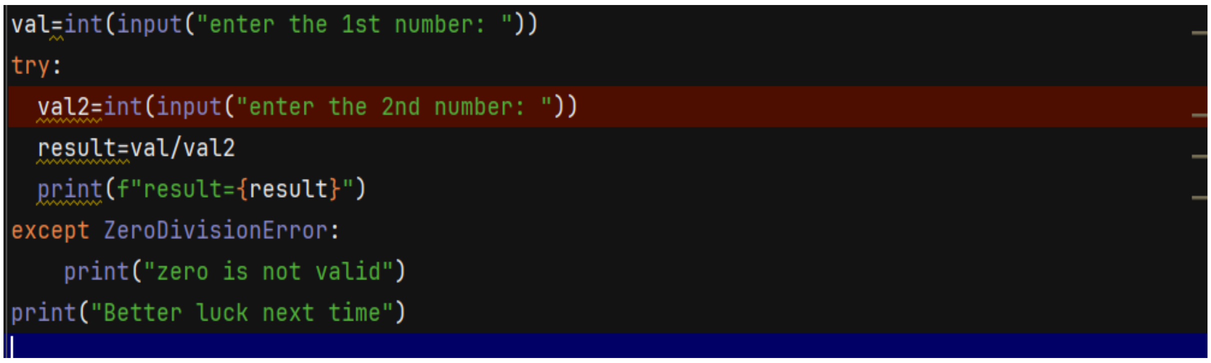
8. Given the python scripts above, explain what the scripts trying to do?

- Read a floating-point number from the user
- Print the floating-point number to the console
- Handle the possibility that the user may enter an invalid input

The script uses a try-except block to handle errors. The try block contains the code that the script wants to execute. If any errors occur while executing the code in the try block, the program will jump to the corresponding except block.

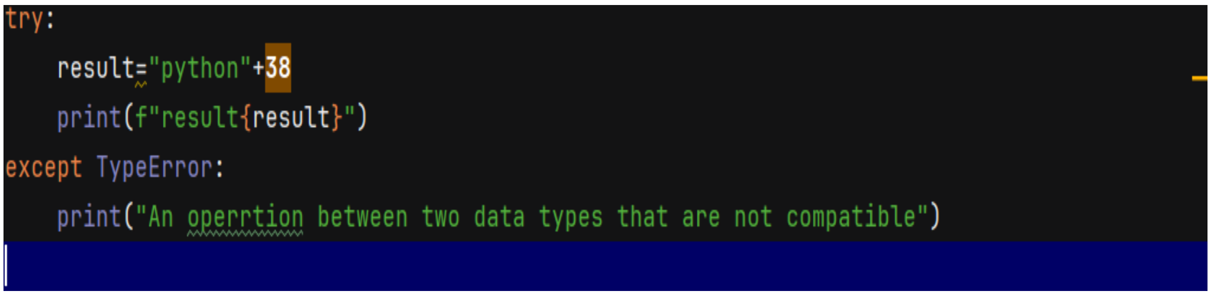
Task 2: Practical (Exception)

1. write program to takes two numbers as input from the user, and then handles the ZeroDivisionError. if the second number is zero. The code should provide a meaningful error message.



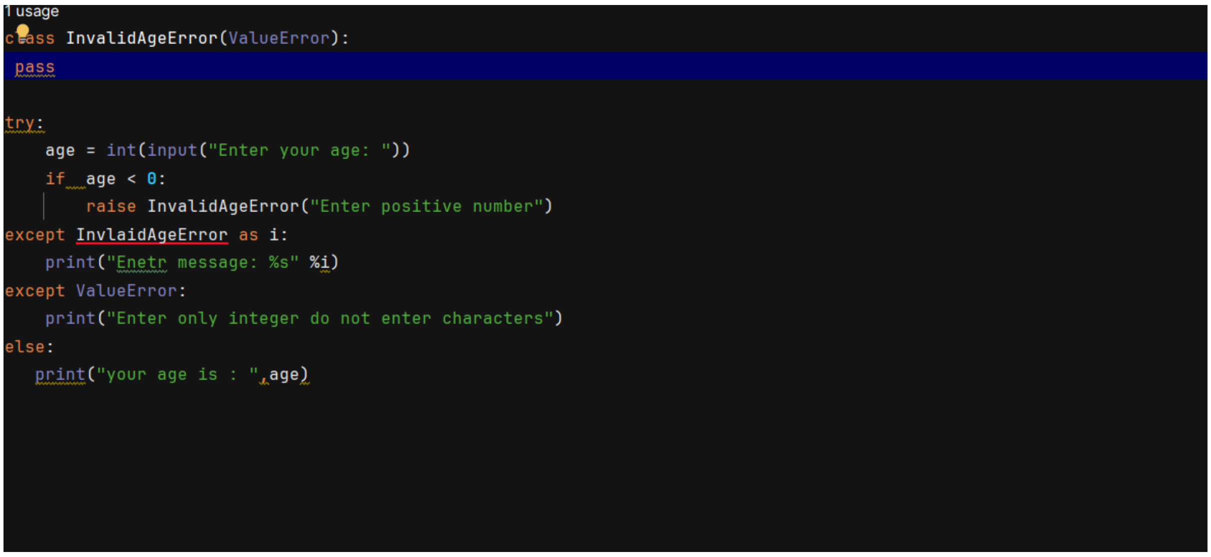
```
val=int(input("enter the 1st number: "))
try:
  val2=int(input("enter the 2nd number: "))
  result=val/val2
  print(f"result={result}")
except ZeroDivisionError:
    print("zero is not valid")
print("Better luck next time")
```

2. Write a program to perform an operation between two data types that are not compatible (e.g., adding a string and an integer). The program need to handle theTypeError` and print a suitable error message.



```
try:
    result="python"+38
    print(f"result{result}")
except TypeError:
    print("An operrtion between two data types that are not compatible")
```

3. Write a program that asks the user to enter their age as an integer. If the user enters a non-integer value, they should handle the `ValueError` and raise a custom `InvalidAgeError` exception. They should provide an informative error message with the exception.



```
1 usage
class InvalidAgeError(ValueError):
 pass

try:
    age = int(input("Enter your age: "))
    if  age < 0:
        raise InvalidAgeError("Enter positive number")
except InvlaidAgeError as i:
    print("Enetr message: %s" %i)
except ValueError:
    print("Enter only integer do not enter characters")
else:
   print("your age is : ",age)
```