1. What is the role of the 'else' block in a try-except statement? Provide an example scenario where it would be useful.

Solution:- If else block is running in the code it means that only try block is executed no except block is executed means there is no error in the code

Example:- **No error**

```
try :
    a = int(input())
    b = int(input())
    print(a/b)
except ZeroDivisionError :
    print("Zero Division error occur")
else :
    print("There is no zero division error occur")
```

[6] ✓ 1.5s

```
2.0

There is no zero division error occur
```

With Error:-

```python
try :
    a = int(input())
    b = int(input())
    print(a/b)
except ZeroDivisionError :
    print("Zero Division error occur")
else :
    print("There is no zero division error occur")
```

[5] ✓ 2.0s

```
Zero Division error occur
```

2. Can a try-except block be nested inside another try-except block? Explain with an example.

Solution :- Yes nested try-except block is possible in Python

Example :- Internal try block is running

```python
try :
    a = int(input())
    b = int(input())
    print(a+b)
    try :
        a = int(input())
        b = int(input())
        print(a/b)
    except ZeroDivisionError:
        print("You entered the zero in the denominator")
except ValueError :
    print("It is an value error you entered the float value instead of integer value")
```

[21] ✓ 5.3s

```
5

You entered the zero in the denominator
```

Example:- Outside try block is running

```python
try :
    a = int(input())
    b = int(input())
    print(a+b)
    try :
        a = int(input())
        b = int(input())
        print(a/b)
    except ZeroDivisionError:
        print("You entered the zero in the denominator")
except ValueError :
    print("It is an value error you entered the float value instead of integer value")
```

[22] ✓ 4.9s

```
It is an value error you entered the float value instead of integer value
```

3. How can you create a custom exception class in Python? Provide an example that demonstrates its usage.

Solution:- Yes we can create with the help of the Exception class in python

No Error:-

```python
# User defined exception in python
class Length_Error(Exception) :
    "Raised when the length is less than 10"
    pass
try :
    n = input()
    if (len(n)<10) :
        raise
    else :
        print("You have entered the length greater than 10")
except :
    print("Please enter the length greater than 10")
```

[25] ✓ 5.2s

```
You have entered the length greater than 10
```

With Error:-

```python
# User defined exception in python
class Length_Error(Exception) :
    "Raised when the length is less than 10"
    pass
try :
    n = input()
    if (len(n)<10) :
        raise
    else :
        print("You have entered the length greater than 10")
except :
    print("Please enter the length greater than 10")
```

[26] ✓ 3.4s

```
Please enter the length greater than 10
```

4. What are some common exceptions that are built-in to Python?

Solution:- Some common exception that are built in python are:-

1:- ZeroDivisionError
2:- ValueError
3:-SyntaxError
4:-Type Error
5:- Index Error
6:- Key Error
7:- File Not found error
8: IO Error
9:- Memory Error

10:- Overflow Error

11:- Import Error

5. What is logging in Python, and why is it important in software development?

Solution:- It is use for tracking the events when the software is run it is important because we can save all the logging details in the separate file for the future debugging

6. Explain the purpose of log levels in Python logging and provide examples of when each log level would be appropriate.

Solution:- Log levels are use that how critical is the error

```python
# Debug Error
import logging
logging.basicConfig(level=logging.DEBUG)
def concatenate(f_name,l_name) :
    full_name = f_name+" "+l_name
    logging.debug("The first name is  %s and the last name is %s",f_name,l_name)
    return full_name
concatenate("Alice","John")
```

[29] ✓ 0.0s

```
DEBUG:root:The first name is  Alice and the last name is John


'Alice John'
```

```python
# Info logging
import logging
logging.basicConfig(logging=logging.INFO)
def information(name) :
    logging.debug("Name of the person is %s",name)
    return name
information("Alice")
```

✓ 0.0s

```
DEBUG:root:Name of the person is Alice


'Alice'
```

```python
import logging
try :
    a = int(input())
    b = int(input())
    result = (a/b)
    logging.basicConfig(logging=logging.INFO)
    logging.debug("Value of the result is %s",result)
except ZeroDivisionError :
    logging.basicConfig(level=logging.ERROR)
    logging.debug("Here denominator is zero")
```

[33] ✓ 1.6s

```
...
DEBUG:root:Here denominator is zero
```

```python
# Import logging
logging.basicConfig(level=logging.CRITICAL)
def LetUsCheckSystem(sys) :
    if (sys!='OK') :
        logging.critical("System failed to boot %s",sys)
LetUsCheckSystem("You need to handle this issue")
```

[35] ✓ 0.0s

```
CRITICAL:root:System failed to boot You need to handle this issue
```

7. What are log formatters in Python logging, and how can you customise the log message format using formatters?

Solution:-In Python's logging module, log formatters are objects that define the format of log messages. They determine how the log records will be formatted before being outputted to the desired logging destination, such as the console or a file.

8. How can you set up logging to capture log messages from multiple modules or classes in a Python application?

Solution:-

```python
import logging

# Configure logging settings
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(name)s - %(levelname)s - %(message)s')

# Create a logger instance for the main application
logger = logging.getLogger('my_application')

# Add handlers to the logger
file_handler = logging.FileHandler('application.log')
console_handler = logging.StreamHandler()

logger.addHandler(file_handler)
logger.addHandler(console_handler)

# Log messages from different modules or classes
class MyClass:
    def __init__(self):
        self.logger = logging.getLogger(__name__)

    def do_something(self):
        self.logger.debug('Debug message')
        self.logger.info('Info message')
        self.logger.warning('Warning message')
        self.logger.error('Error message')
        self.logger.critical('Critical message')

# Create an instance of MyClass and log messages
my_object = MyClass()
my_object.do_something()

# Log messages from the main application
logger.debug('Debug message')
logger.info('Info message')
logger.warning('Warning message')
logger.error('Error message')
logger.critical('Critical message')
```

[74] ✓ 0.0s

9. What is the difference between the logging and print statements in Python? When should you use logging over print statements in a real-world application?

Solution:- Print statement only prints the statement on the screen but with the help of logging we can store the information which was written in the log statement into the separate file which can be helpful in the future to debug the code if any error occurs
The error which are very critical error like ZeroDivision error or system booting error at that time you can use log statement

10. Write a Python program that logs a message to a file named "app.log" with the following requirements:

- The log message should be "Hello, World!"
- The log level should be set to "INFO."
- The log file should append new log entries without overwriting previous ones.

Solution:-

```python
# We want to store the information for the future reference
import os
import logging
print(os.getcwd())
dir_path=r'c:\Users\gupta\OneDrive\Desktop\Assignment'
# File Name
log_file = 'app.txt'
# Join path name and file name
full_path = os.path.join(dir_path,log_file)
# Checking the particular path
os.makedirs(dir_path,exist_ok=True)
logger = logging.getLogger()
# Here you are setting the level of the warning
logger.setLevel(logging.INFO)
# What are the logging event converted to the handler
handler = logging.FileHandler(full_path)
# Here we are setting what information we want to display in the log files
handler.setFormatter(logging.Formatter('%(asctime)s:%(levelname)s:%(message)s'))
logger.addHandler(handler)
def storing_error(a,b) :
    a = int(input())
    b = int(input())
    logging.info("The value of a is %s and the value of the b is %s",a,b)
storing_error(a,b)
```

[50] ✓ 6.4s

```
c:\Users\gupta\OneDrive\Desktop\Assignment

INFO:root:The value of a is 9 and the value of the b is 25
```

app.txt file output:-

app.txt

```
2023-07-03 19:29:04,593:INFO:The value of a is 5 and the value of the b is 11
2023-07-03 19:29:04,593:INFO:The value of a is 5 and the value of the b is 11
2023-07-03 19:29:04,593:INFO:The value of a is 5 and the value of the b is 11
2023-07-03 19:29:04,593:INFO:The value of a is 5 and the value of the b is 11
2023-07-03 19:29:22,949:INFO:The value of a is 9 and the value of the b is 25
2023-07-03 19:29:22,949:INFO:The value of a is 9 and the value of the b is 25
2023-07-03 19:29:22,949:INFO:The value of a is 9 and the value of the b is 25
2023-07-03 19:29:22,949:INFO:The value of a is 9 and the value of the b is 25
2023-07-03 19:29:22,949:INFO:The value of a is 9 and the value of the b is 25
```

11. Create a Python program that logs an error message to the console and a file

named "errors.log" if an exception occurs during the program's execution. The error message should include the exception type and a timestamp.

Solution:-

```python
import os
import logging

dir_path = r'c:\Users\gupta\OneDrive\Desktop\Assignment'
log_file = 'errors.txt'
full_path = os.path.join(dir_path, log_file)
os.makedirs(dir_path, exist_ok=True)

logger = logging.getLogger()
logger.setLevel(logging.ERROR)

handler = logging.FileHandler(full_path)
handler.setFormatter(logging.Formatter('%(asctime)s:%(levelname)s:%(message)s'))
logger.addHandler(handler)

try:
    lst = [4, 5, 7, 8, 9, 10]
    n = int(input("Enter the index: "))
    print("The value at index", n, "is", lst[n])
    logging.info("Value at index %s is %s", n, lst[n])

    try:
        a = int(input("Enter a: "))
        b = int(input("Enter b: "))
        print("The result of a/b is", a / b)
    except ZeroDivisionError as e:
        print("You divided the denominator by zero")
        logging.error("%s occurred", type(e).__name__)
except IndexError as e:
    print("You are accessing an index that is out of range")
    logging.error("%s occurred", type(e).__name__)
```

[73] ✓ 1.7s