Exception handeling and Logging is a convention that we all are suppose to follow.

- Blocks included in try-Exception
  - Try:
  - except:
  - else:
  - finally:

In [6]:

```
import logging
logging.basicConfig(filename="Resources/logging.log", level=logging.DEBUG,format="%(ascti
try:
    logging.info("this is where the file is opened to be read")
    f= open("Resources/text_try.txt","r")
except Exception as e:
    logging.error("File did not open there is some error")
    print("There is some issue with the code",e)
else:
    logging.info("else block executing as try block executed")
    print("this block will execute once try itself without exception")
```

```
There is some issue with the code [Errno 2] No such file or directory: 'Re
sources/text_try.txt'
```

In [7]:

```
try:
    logging.info("this is where the file is opened to be read")
    f= open("Resources/text_try.txt","w")
except Exception as e:
    logging.error("File did not open there is some error")
    print("There is some issue with the code",e)
else:  #only work if try block executes
    logging.info("else block executing as try block executed")
    print("this block will execute once try itself without exception")
finally:  #will always execute
    logging.info("this is finally block which execute any code which should execute at an
    f.close()
    print("thi9s block will always run")
```

```
this block will execute once try itself without exception
thi9s block will always run
```

In [ ]:

```

```

Custom Exception Handeling

In [9]:

```
#example
a=10
a/0  #division by zero is an exception for the computer.
```

```
---------------------------------------------------------------------------
-
ZeroDivisionError                         Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_17228\1760181836.py in <module>
      1 #example
      2 a=10
----> 3 a/0

ZeroDivisionError: division by zero
```

In [10]:

```
#example: Custom Exception
age =int(input("Enter Your Age:- "))
#in this case -ve age is not an exception for the user but an custom exception for us
```

```
Enter Your Age:- -25
```

In [21]:

```
#Solution
#writing a custom exceptions class- use for printing exception message
class validate_age(Exception):
    def __init__(self,msg):
        self.msg = msg
```

In [22]:

```
#creating a velidate function- for checking validation
def validateage(age):
    if age <0:
        raise validate_age("age should be more them 0") #class object
    elif age>200:
        raise validate_age("age is too high")  #class object
    else:
        print("age")
```

In [25]:

```
try:
    age = int(input("Enter You Age:- "))
    validateage(age)
except validate_age as e:
    print(e)
```

```
Enter You Age:- 225566
age is too high
```

In [ ]:

```
```

List of some General Exceptions

In [26]:

```
try:
    a=10
    a/0
except ZeroDivisionError as e:
    print(e)
```

```
division by zero
```

In [27]:

```
try:
    int("asit")
except(ValueError, TypeError) as e:
    print(e)
```

```
invalid literal for int() with base 10: 'asit'
```

In [30]:

```
try:
    int("asit")
except : #Do not do this we are suppose to mention the type of error specifically
    print("there is an error")
```

```
there is an error
```

In [31]:

```
try:
    import asit
except ImportError as e:
    print(e)
```

```
No module named 'asit'
```

In [33]:

```
try:
    d = {"key1":[1,2,3],"key2":"asit000"}
    d["key10"]
except KeyError as e:
    print(e)
```

```
'key10'
```

In [34]:

```
try:
    "asit".test()
except AttributeError as e:
    print(e)
```

```
'str' object has no attribute 'test'
```

In [36]:

```
try:
    lst=[1,2,3,4]
    lst[10]
except IndexError as e:
    print(e)
```

```
list index out of range
```

In [37]:

```
try:
    123+"asit"
except TypeError as e:
    print(e)
```

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

In [39]:

```
try:
    with open("test_error.txt","r") as f:
        f,read()
except FileNotFoundError as e:print(e)
```

```
[Errno 2] No such file or directory: 'test_error.txt'
```

In [45]:

```
try:
    with open("test_error.txt","r") as f:
        f,read()
except Exception as e:
    print("this is my Exception block",e)
except FileNotFoundError as e:
    print("this is my File not found Error block",e)

#In above case Exception class itself predicts the error type and print the error before
```

```
this is my Exception block [Errno 2] No such file or directory: 'test_erro
r.txt'
```

Never try to write a super class(i.e Exception) in first place always try to write generic/ specific error class

Best Practices for Exception Handeling

1- Always use specific exceptions

In [46]:

```
try:
    a=10
    a/0
except ZeroDivisionError as e:
    print(e)
```

```
division by zero
```

2- always print a valid message

In [47]:

```
try:
    a=10
    a/0
except ZeroDivisionError as e:
    print("this is my error:- ",e)
```

```
this is my error division by zero
```

3- Always log your error

In [50]:

```
import logging
logging.basicConfig(filename="Resources/logging.log", level=logging.ERROR, format="%(asct
try:
    a=10
    a/0
except ZeroDivisionError as e:
    logging.error("this is my error:- {}".format(e))
```

4- Always avoid to write multiple exception handeling

In [51]:

```
try:
    a=10
    a/0
except ZeroDivisionError as e:
    logging.error("this is my error:- {}".format(e))
except TypeError as e:
    print(e)
except AttributeError as e:
    print(e)
```

5- Prepare a proper documentations

avoid incerting or avoid creating anything which may give a 0problem to a future developer whois going to check you code or do a modification. Proper Documentation with proper validatioin is very important.

5- Cleanup all the resources

example a file opened must be closed as it is taking up our memory.

In [53]:

```
try:
    with open ("Resources/cleanup.txt","w")as f:
        f.write("this is written statement")
except FileNotFoundError as e:
    logging.error("This is my error:- {}".format(e))
finally:
    f.close()
```

In [ ]:

```

```