Exception handling :- Any illegal operation in program can cause exceptions or error to handle an error or except in program we use the concept of exception handling.

There are many built-in exception in python that are raised when corresponding errors occur.

→ we can view all the built-in exceptions using the built in local() function.

→ Exception Base Exception :- This is base class for all built-in exceptions. but it is not inherit by user-defined class directly.

→ Exception Arithematic Error :-
This class is the base class for those built-in exceptions that are raised for various Arithematic such as :- Overflow Error
- ZeroDivision Error
- Floating point Error.

Ex:-
```
try:
    a = 10/0
    print(a)
except ArithmeticError:
    print("This statement is raising an Arithemetic Error exception:")
else:
    print("Success")
```

Exception BufferError: This exception is raised when buffer related operations can't be perfo

Exception LookupError :- This is the base class for those exceptions that are raised when a key or index is invalid or not found. The exception are

1. KeyError
2. IndexError.

Ex:-

```
try:
    a = [1, 2, 3]
    print(a[3])
except LookupError:
    print("Index out of range")
else:
    print("Sucess")
```

→ built in exception class.

Exception NameError :- This error is raised when a local or global name is not found.
for ex:- an unqualified variable name.

Ex :-

```
def func():
    print ans

func()
```

O/P:-

Missing parenthesis in print did u mean print(...)

Finally Keyword in Python :-

'Finally' keyword in python means the code in 'finally' block is always executed after 'try' and 'except' block.

The finally block always execute after normal termination or after 'try' block termination due to an exception

Syn:

```
try:
    ____
    ____
except:
    ____
    ____
finally:
    ____
    ____
```

Raise Keyword:- The raise keyword is uses to raise an exception.

⇒ we can define what kind of error to raise and the text to print the user.

Ex:-

```
x = "hello"

if not type(x) is int:
    raise TypeError("only integers is allow")
```

Exception :- An Exception is an event, which occurs during the execution of the program it is also called run-time error when that error occurs, python generates an exception during the execution and that can be handled to avoid interruption in program execution.

Try :- This block having the code in which error have to occur.

Except :- In this block we have to handle the exception which will occur either by built-in exception or user define exception.

Else :- If their will be no exception or error than this block will be executed.

Finally :- This block code is always executed either exception is generated or not.

Syn :-

```
try:
    ____
    ____
except:
    ____ optional block
    ____ Handling of exception
else:
    ____ (execute if no exception)
finally:
    ____ (execute in any way)
```

→ First try clause is executed that is the code between try and except clause

→ If no except then only 'try' clause will execute 'except' clause not executed.

→ If any exception occurs the try clause will be skipped and except clause will run.

→ If any exception occurs, but the except clause within the code doesn't handle it is passed to the outer try statement. if the exception is left unhandled then execution stop.

→ The 'try' statement can have more than one 'except' clause.

Ex:-

```
def divide(x, y):
    try:
        result = x//y
        print("answer is:", result)
    except ZeroDivision error:
        print("Not divisible by zero")
```

divide (3, 2)

" (3, 0)

→ O/P :- 1

→ O/P — not divisible by zero

Else clause → The else block is only executed if 'try' block did'nt raise an exception or error.

Ex:-

```
try:
    result = x//y
except: ZeroDivisionError:
    print (" Dividing by 0 is not possible")
else:
    print (" Answer is:" result)

divide (3,2)
"      (3,0)
```

O/P :

Answer is : 1 (This is in else clause it is only executed when it not gives exception or error)

O/P :-

Dividing by 0 is not possible

Finally keyword:- Python provide keyword "finally" it is always executed after try and except blocks. It is executed after normal termination of 'try' block or 'try' block executed is stop due to exception or error.

Ex:-

```
def div1 (x, y):
    try:
        result = x//y
    except : ZeroDivisionError:
        print (" not divided by 0")
```

finally:
    print ("This is always executed")

div(3,2)
div(3,0)

→ O/P:
    res = 1
    This is always executed.
    not divide by 0
    This is always executed.

both finally block is executed

→ Raising Exception:- The 'raise' statement allows programmer to force a specific exception to oc

→ The argument with 'raise' statement indicates exception to be raised, this must be either a exception instance or an exception class (i.e. c derives from exception class).

Syn:-

```
try:
    raise NameError ("Hi there")
except NameError:
    print ("An Exception")
    raise
```

Ex:-

```
def validate_input (value):
    if value < 0:
        raise ValueError ("i/p value must be +ve")

try:
    validate_input (-5) # This will raise a value error
except ValueError as e:
    print ("Error:", e)
```

User-defined Exception in Python :- We may name our own exception by creating a new exception class

→ User-defined exception must be derived from exception class, either directly or indirectly

read it) - Exception handling

⇒ The other way of writing except statement is as below

```
def divide(x, y):
    try:
        result = x // y
        print("Answer is:", result)
    except Exception as e:
        print("The error is:", e)
```

→ In this we take the value of error.

```
divide(3, "GFG")
divide(3, 0)
```

→ O/P :- the error is unsupported oper and type(s) for //: 'int' and 'str'

→ O/P :- The error is integer division or modulo by zero.

Ex:- Calculate factorial with Exception Handling:

```
def factorial(num):
    if num == 0
        return 1
    else:
        return num * factorial(num-1)
try:
    num = int(input("enter the +ve number to find factorial"))
```

```
if num < 0:
    raise ValueError("factorial of -ve number is not define")
print("factorial of", num, "is",
      factorial(num))

except Exception as e:
    print("error as", e)
```

factorial (6)
" (-6)
" (xy)

o/p:

factorial of 6 is 720
error as factorial of -ve number is not defined

→ error as invalid literal for int() with base 10: 'x'