except block without exception type or error type is called generic except block. This except block is able to handle any type of error.

Example:

```
try:
    num1=int(input("Enter First Number "))
    num2=int(input("Enter Second Number "))
    num3=num1/num2
    print(f'{num1}/{num2}={num3}')
except:
    print("input must be number or cannot divide number with zero")
```

Output:

```
Enter First Number 5
Enter Second Number 0
input must be number or cannot divide number with zero
```

sys.exc_info()

This function returns the old-style representation of the handled exception. If an exception e is currently handled (so exception() would return e), exc_info() returns the tuple (type(e), e, e.__traceback__).

Example:

```
import sys
try:
    num1=int(input("Enter First Number "))
    num2=int(input("Enter Second Number "))
    num3=num1/num2
    print(f'{num1}/{num2}={num3}')
except:
    a=sys.exc_info()
    print(type(a[1]))
```

Output:

```
Enter First Number 5
Enter Second Number 0
<class 'ZeroDivisionError'>

Enter First Number 6
Enter Second Number abc
```

<class 'ValueError'>

finally

finally is not exception handler.
finally block contain instructions which are executed after execution of try block or except block.
In application development finally block is used to de-allocate resources allocated by try block.

Syntax-1:	Syntax-2:

try:
  statement1
  statement2
except <error-type>:
  statement-3
finally:
  statement-4
	try:
	  statement-1
	  statement-2
	finally:
	  statement-3

Common statements of try and except block are defined inside finally block.

```
try:
   open connection to database
   send SQL statement
except SQLError:
   Error in SQL statement
finally:
   close database connection
```

finally block executed,
1. after execution of try block
2. after execution of except block
3. unhandled exception, after execution of finally block it terminates execution of program

Example:

```
studDict={101:['naresh','python'],
          102:['suresh','java'],
          103:['kishore','c++']}

try:
```

```
    rollno=int(input("Input Rollno "))
    name,course=studDict[rollno]
    print(f'Name {name}')
    print(f'Course {course}')
except KeyError:
    print("Invalid Rollno")
finally:
    print("inside finally block")
```

Output:

```
Input Rollno 101
Name naresh
Course python
inside finally block

Input Rollno 110
Invalid Rollno
inside finally block

Input Rollno abc
inside finally block
Traceback (most recent call last):
  File "E:/student drive/python7amjuly/etest7.py", line 6, in <module>
    rollno=int(input("Input Rollno "))
ValueError: invalid literal for int() with base 10: 'abc'
```

Predefined error types and description

***exception* ValueError**
Raised when an operation or function receives an argument that has the right type but an inappropriate value

***exception* ZeroDivisionError**
Raised when the second argument of a division or modulo operation is zero

***exception* IndexError**

Raised when a sequence subscript is out of range

***exception* KeyError**
Raised when a mapping (dictionary) key is not found in the set of existing keys.

***exception* TypeError**
Raised when an operation or function is applied to an object of inappropriate type

raise keyword
This keyword is used to generate exception.
Generating exception is nothing but creating exception object and giving to PVM.

Syntax:
raise <exception-type>/<error-type>

Example:
```
def multiply(a,b):
    if a==0 or b==0:
        raise ValueError()
    else:
        return a*b



num1=int(input("Enter First Number "))
num2=int(input("Enter Second Number "))
try:
    num3=multiply(num1,num2)
    print(f'{num1}*{num2}={num3}')
except ValueError:
    print("cannot multiply numbers with zero")
```

Output:
Enter First Number 5
Enter Second Number 2
5*2=10

Enter First Number 5
Enter Second Number 0
cannot multiply numbers with zero

Custom Error Types or User defined error types
Every error type is one class. These classes are inherited from Exception class.

Basic steps for creating user defined error type

1. Create class by inheriting Exception class
2. Include constructor within class

Example:

```
class MultiplyError(Exception):
    def __init__(self):
        super().__init__()

def multiply(a,b):
    if a==0 or b==0:
        raise MultiplyError()
    else:
        return a*b



num1=int(input("Enter First Number "))
num2=int(input("Enter Second Number "))
try:
    num3=multiply(num1,num2)
    print(f'{num1}*{num2}={num3}')
except MultiplyError:
    print("cannot multiply numbers with zero")
```

Output:

```
Enter First Number 5
Enter Second Number 0
cannot multiply numbers with zero

Enter First Number 5
```

Enter Second Number 3
5*3=15

Example:

```
usersDict={'nit':'nit123',
           'naresh':'naresh321',
           'kishore':'k456'}

class LoginError(Exception):
    def __init__(self):
        super().__init__()

def login(user,pwd):
    if user in usersDict and pwd==usersDict[user]:
        print(f'{user} welcome')
    else:
        raise LoginError()


uname=input("UserName ")
password=input("Password ")
try:
    login(uname,password)
except LoginError:
    print("Invalid username or password")
```

Output:

UserName nit
Password nit123
nit welcome

UserName ramesh
Password r432
Invalid username or password

Example:

```
class InsuffBalError(Exception):
    def __init__(self):
        super().__init__()
class Account:
```

```
    def __init__(self,a,c,b):
        self.__accno=a
        self.__cname=c
        self.__balance=b
    def deposit(self,a):
        self.__balance=self.__balance+a
    def withdraw(self,a):
        if a>self.__balance:
            raise InsuffBalError()
        else:
            self.__balance=self.__balance-a
    def __str__(self):
        return f'{self.__accno},{self.__cname},{self.__balance}'


acc1=Account(101,"naresh",6000)
print(acc1)
try:
    acc1.deposit(7000)
    print(acc1)
    acc1.withdraw(3000)
    print(acc1)
    acc1.withdraw(20000)
    print(acc1)
except InsuffBalError:
    print("insuff balance")
```

Output

```
101,naresh,6000
101,naresh,13000
101,naresh,10000
insuff balance
```

Nested try blocks