raise keyword

This keyword is used for generating error or exception.
Creating object of error class and giving to PVM is called generating error. These errors are generated by functions or methods.

Error handling provides clear separation of business logic and error logic.

Syntax: raise error-class()

Example

```
def multiply(n1,n2):
    if n1==0 or n2==0:
        raise ValueError()
    else:
        return n1*n2




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

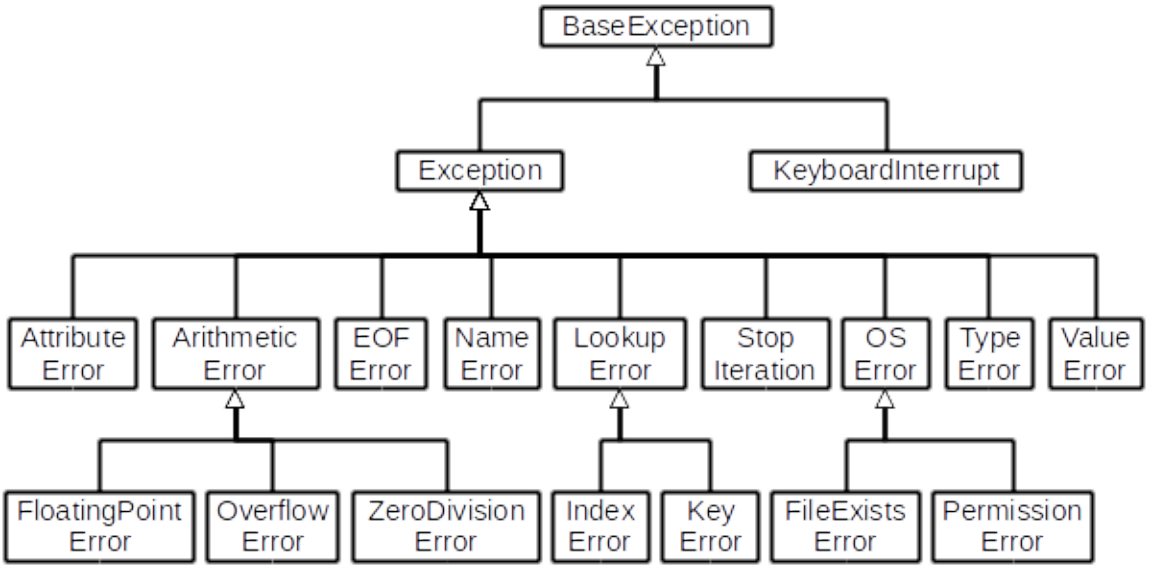
Output

```
Enter First Number 5
Enter Second Number 2
```

product of 5*2=10

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

Customer Error types or customer Exceptions or User defined Exception



Every error is one type or class. All these error classes are inherited from Exception class.

Syntax:

```
class <error-type>(Exception):
        attributes
        methods
```

Example:

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

```
def multiply(n1,n2):
    if n1==0 or n2==0:
        raise MultiplyError()
    else:
        return n1*n2



try:
    num1=int(input("Enter First Number "))
    num2=int(input("Enter Second Number "))
    num3=multiply(num1,num2)
    print(f'product of {num1}*{num2}={num3}')
except ValueError:
    print("input only integer values")
except MultiplyError:
    print("cannot multiply number with zero")
```

Output

```
Enter First Number 8
Enter Second Number 0
cannot multiply number with zero

Enter First Number 5
Enter Second Number abc
input only integer values
```

Example:

```
users={'naresh':'nit123','suresh':'s456','ramesh':'r678'}
class LoginError(Exception):
    def __init__(self,msg):
```

```
        super().__init__()
        self.__msg=msg
    def __str__(self):
        return self.__msg

def login(user,pwd):
    if user in users and users[user]==pwd:
        print(f'Welcome')
    else:
        raise LoginError("Invalid username or password")



# Login application
user=input("UserName :")
pwd=input("Password :")
try:
    login(user,pwd)
except LoginError as a:
    print(a)
```

Output

```
UserName :naresh
Password :nit123
Welcome

UserName :abc
Password :xyz
Invalid username or password
```

Example:

```
class InsuffBalError(Exception):
```

```
    def __init__(self):
        super().__init__()

class Account:
    def __init__(self,a,cn,b):
        self.__accno=a
        self.__cname=cn
        self.__balance=b
    def deposit(self,t):
        self.__balance=self.__balance+t
    def withdraw(self,t):
        if self.__balance<t:
            raise InsuffBalError()
        else:
            self.__balance=self.__balance-t
    def getBalance(self):
        return self.__balance


# Banking Application
acc1=Account(101,"naresh",60000)
while True:
    print("1. Deposit ")
    print("2. Withdraw ")
    print("3. Balance Enq")
    print("4. Exit")
    opt=int(input("Enter your option "))
    if opt==1:
        amt=int(input("Enter Amount "))
        acc1.deposit(amt)
        print("Amount Deposit...")
    elif opt==2:
```

```
        amt=int(input("Enter Amount "))
        try:
            acc1.withdraw(amt)
            print("Amount Withdraw...")
        except InsuffBalError:
            print("balance not avl")
    elif opt==3:
        bal=acc1.getBalance()
        print(f'Balance is {bal}')
    elif opt==4:
        break
    else:
        print("invalid option")
```

Output

```
1. Deposit
2. Withdraw
3. Balance Enq
4. Exit
Enter your option 1
Enter Amount 5000
Amount Deposit...
1. Deposit
2. Withdraw
3. Balance Enq
4. Exit
Enter your option 3
Balance is 65000
1. Deposit
2. Withdraw
3. Balance Enq
4. Exit
```

Enter your option 2
Enter Amount 10000
Amount Withdraw...

nested try blocks

nested try blocks are used to handle nested exceptions or errors.
Try block within try block is called nested try block.

Syntax:

```
try:  □ Outer Try block
  statement-1
  statement-2
  try: □ Inner Try block/Nested Try Block
     statement-3
     statement-4
  except <error-type>:
     statement-5
except <error-type>:
   statement-6
```

if there is an error inside inner try block, it is handled by inner except block. If inner except block not handle error, it is given to outer except block.

Example:

```
try: # Outer try block
   num1=int(input("Enter first number "))
   num2=int(input("Enter second number "))
   try: # Nested try block/inner try block
      num3=num1/num2
      print(f'division of {num1}/{num2}={num3}')
   except ZeroDivisionError:
```

```
        print("cannot divide number with zero")

except ValueError:
    print("input must be integer type")
```

Output

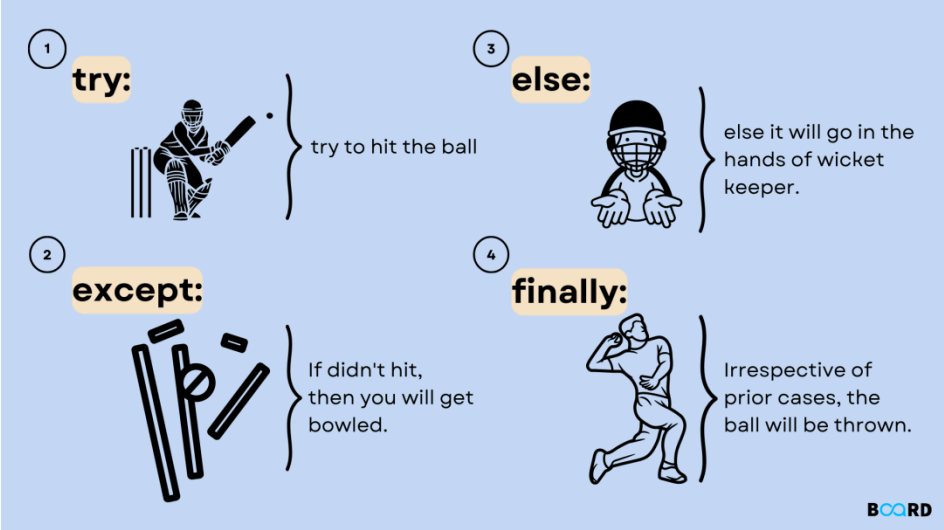
```
Enter first number 5
Enter second number 2
division of 5/2=2.5

Enter first number 4
Enter second number abc
input must be integer type

Enter first number 4
Enter second number 0
cannot divide number with zero
```

try..except..else..finally

else block is executed when there is no error within try block.

Syntax:

```
try:
   statements which generate errors
except <error-type>:
   print user defined error message
else:
   statement which is executed if no error inside try block
finally:
   statement which is executed after try and except block
```



Example:

```
try:
    print("inside try block")
except:
    print("inside except block")
else:
    print("inside else block")
finally:
    print("inside finally block")
```

Output

```
inside try block
inside else block
inside finally block
```