In [ ]:

```
1)write a function to compute 5/0 and use try/eccept to exceptions.
```

In [27]:

```
def throws():
    return 5/0
```

In [28]:

```
try:
    throws()

except ZeroDivisionError:
    print("division by zero!")

except Exceptionerr:
    print("caught an exception")
finally:
    print("in finally block for clean")
```

```
division by zero!
in finally block for clean
```

In [29]:

```
def divide():
    return 5/0
```

In [30]:

```
try:
    divide()

except ZeroDivisionError as ze:
    print("dividing a number by zero!!",ze)

except:
    print('caught exception')
```

```
dividing a number by zero!! division by zero
```

In [ ]:

```
2)implement a python program to generate all sentences where subject is in  ["Americans","Indians "
] and verbs is in ["plays","watch"] and objects is in ["Baseball","cricket"].


hint:subject,verb,object should be declared in program as shown below.
     subject =["Americans","Indians "]
     verbs = ["plays","watch"]
     objects=["Baseball","cricket"].
```

In [33]:

```
subject=["Americans","Indians"]
verb=["play","watch"]
obj=["Baseball","Cricket"]

sentence_list=[(sub+" " +vb+" " +ob) for sub in subject for vb in verb for ob in obj]
for sentence in sentence_list:
    print(sentence)
```

```
Americans play Baseball
Americans play Cricket
```

```
Americans watch Baseball
Americans watch Cricket
Indians play Baseball
Indians play Cricket
Indians watch Baseball
Indians watch Cricket
```

In [ ]:

```
write a function  so that the columns of the output matrix are powers of the input vector.
 The order of the powers is determined by the increasing boolean argument. specifically,when increa
sing is False, the i-th output column is the input vector rasied element-wise to the power of N-i-1
.

hint:such amatrix with a geometric progression in each row is named for AlexanderTheophile
Vandermonde.
```

In [25]:

```
import numpy as np

def gen_vander_matrix(ipvector,n,increasing=False):

    if not increasing:
        op_matx=np.array([x**(n-1-i) for x in ipvector for i in range(n)]).reshape(ipvector.size,n)
    elif increasing:
        op_matx=np.array([x**i for x in ipvector for i in range(n)]).reshape(ipvector.size,n)
    return op_matx
```

In [26]:

```
print("--------------------output--------------------\n")

inputvector=np.array([1,2,3,4,5])
no_col_opmat=3
op_matx_dec_order=gen_vander_matrix(inputvector,no_col_opmat,False)
op_matx_inc_order=gen_vander_matrix(inputvector,no_col_opmat,True)

print("The input array is:",inputvector,"\n")
print("Number of columns in output matrix should be:",no_col_opmat,"\n")
print("Vander matrix of the input array in decresaing order of powers:\n\n",op_matx_dec_order,"\n"
)
print("Vander matrix of the input array in incresaing order of powers:\n\n",op_matx_inc_order,"\n"
)

inputvector=np.array([1,2,4,6,8,10])
no_col_opmat=5
op_matx_dec_order=gen_vander_matrix(inputvector,no_col_opmat,False)
op_matx_inc_order=gen_vander_matrix(inputvector,no_col_opmat,True)

print("-------------------------------------------------------\n")
print("-------------------------------------------------------\n")
print("The input array is:",inputvector,"\n")
print("Number of columns in output matrix should be:",no_col_opmat,"\n")
print("Vander matrix of the input array in decresaing order of powers:\n\n",op_matx_dec_order,"\n"
)
print("Vander matrix of the input array in incresaing order of powers:\n\n",op_matx_inc_order,"\n"
)
```

```
--------------------output--------------------

The input array is: [1 2 3 4 5]

Number of columns in output matrix should be: 3

Vander matrix of the input array in decresaing order of powers:

 [[ 1  1  1]
 [ 4  2  1]
 [ 9  3  1]
 [16  4  1]
 [25  5  1]]
```

```
Vander matrix of the input array in incresaing order of powers:

 [[ 1  1  1]
 [ 1  2  4]
 [ 1  3  9]
 [ 1  4 16]
 [ 1  5 25]]

------------------------------------------------------------

------------------------------------------------------------

The input array is: [ 1  2  4  6  8 10]

Number of columns in output matrix should be: 5

Vander matrix of the input array in decresaing order of powers:

 [[    1     1     1     1     1]
 [   16     8     4     2     1]
 [  256    64    16     4     1]
 [ 1296   216    36     6     1]
 [ 4096   512    64     8     1]
 [10000  1000   100    10     1]]

Vander matrix of the input array in incresaing order of powers:

 [[    1     1     1     1     1]
 [    1     2     4     8    16]
 [    1     4    16    64   256]
 [    1     6    36   216  1296]
 [    1     8    64   512  4096]
 [    1    10   100  1000 10000]]
```

In [ ]:

```
```

In [ ]:

```
```