errors and exception handling

In [1]:
```
#errors are issues due to which code will stop execution,on the other hand exce
#which changes the flow of program.
print('hello)
```

```
  Input In [1]
    print('hello)
                 ^
SyntaxError: EOL while scanning string literal
```

In [2]:
```
if 4>2:
print("hello")
```

```
  Input In [2]
    print("hello")
    ^
IndentationError: expected an indented block
```

In [3]:
```
print(t)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [3], in <cell line: 1>()
----> 1 print(t)

NameError: name 't' is not defined
```

In [8]:
```
def info(x):
    print (x)
info(2)
```

```
2
```

In [10]:
```
# exception handling
 #TRY AND EXCEPT STATEMENTS.
f=open("testfile", "r")
f.write("hello,file is opened")
f.close()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Input In [10], in <cell line: 3>()
      1 # exception handling
      2  #TRY AND EXCEPT STATEMENTS.
----> 3 f=open("testfile", "r")
      4 f.write("hello,file is opened")
      5 f.close()

FileNotFoundError: [Errno 2] No such file or directory: 'testfile'
```

In [14]:
```
try:
    f=open("testfile","w")
    f.write('hello i am writing this file')
except:
    #this will only check for an IO error
    print("error: could not find file read data")
else:
    print("content written successfully")
    f.close()
```

```
content written successfully
```

In [ ]:
```
try:
    f=open("testfile","w")
    f.write('hello i am writing this file')
except:
    #this will only check for an IO error
    print("error: could not find file read data")
else:
    print("content written successfully")
    f.close()
```

finally

In [16]:
```
try:
    f=open("testfile", "w")
    f.write("test write statement")
    f.close()
finally:
    print("always execute finally blocks")
```

```
always execute finally blocks
```

In [2]:
```
def ask_int_value():
    try:
        val=int(input("please enter an integer:"))
    except:
        print("looks like you did not enter an integer")
        val= int (input("try again-please enter an integer:!"))
    finally:
        print("finally, i executed")
    print (val)
```

In [7]:
```
ask_int_value()
```

```
      4 except:

ValueError: invalid literal for int() with base 10: 'jj'

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
Input In [7], in <cell line: 1>()
----> 1 ask_int_value()

Input In [2], in ask_int_value()
      4 except:
      5     print("looks like you did not enter an integer")
----> 6     val= int (input("try again-please enter an integer:!"))
      7 finally:
      8     print("finally, i executed")

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

In [14]:
```
while True:
    try:
        val=int(input("please enter an integer:"))
    except:
        print("looks like you did not enter an integer")
        continue
        else:
            print("yep thats an integer")
            print(val)
            break
        finally:
            print("finally i executed")
            print(val)
```

```
  Input In [14]
    else:
    ^
SyntaxError: invalid syntax
```

In [16]:
```
ask_int_value()
```

```
please enter an integer:yy
looks like you did not enter an integer
try again-please enter an integer:!55
finally, i executed
55
```

In [ ]:
```
def ask_value()
while True:
    try:
        val=int(input("enter a number between 10 to 15"))
        except:
            print("no.out of range" )
            continue
        else:
            if val in range(10,15):
                print
```

pyhton pip

In [8]:
```
pip--version
```

```
pip 21.2.4 from C:\Users\Devansh Sharma\anaconda3\lib\site-packages\pip (pyth
on 3.9)Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
```
pip install numpy
```

In [4]:
```
pip list
```

```
pyerfa                        2.0.0
pyflakes                      2.3.1
Pygments                      2.11.2
PyHamcrest                    2.0.2
PyJWT                         2.1.0
pylint                        2.9.6
pyls-spyder                   0.4.0
PyNaCl                        1.4.0
pyodbc                        4.0.32
pyOpenSSL                     21.0.0
pyparsing                     3.0.4
pyreadline                    2.1
pyrsistent                    0.18.0
PySocks                       1.7.1
pytest                        7.1.1
python-dateutil               2.8.2
python-lsp-black              1.0.0
python-lsp-jsonrpc            1.0.0
python-lsp-server             1.2.4
```

class 9th NUMPY

In [ ]:
```
#linear algebra library for python, main building block for data science.
```

In [9]:
```
pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\devansh sharma\anaconda3\lib
\site-packages (1.21.5)
Note: you may need to restart the kernel to use updated packages.
```

In [10]:
```
import numpy as np
```

numpy arrays

In [11]:
```
# vectors and matrices
```

In [12]:
```
l=[1,2,3]
l
```

Out[12]:
```
[1, 2, 3]
```

In [14]:
```
np.array(l)
```

Out[14]:
```
array([1, 2, 3])
```

In [16]:
```
matrix=[[1,2,3], [4,5,6], [7,8,9]]
matrix
```

Out[16]:
```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

In [17]:
```
np.array (matrix)
```

Out[17]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

built in methods

In [18]:
```
#return evenly spaced values within a given interval
```

In [36]:
```
np.arange(0,10,3)
```

Out[36]:
```
array([0, 3, 6, 9])
```

In [25]:
```
np.arange(0,11,12)
```

Out[25]:
```
array([0])
```

In [27]:
```
np.zeros(3)
```

Out[27]:
```
array([0., 0., 0.])
```

In [31]:
```
np.zeros((2,5))
```

Out[31]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [32]:
```
np.ones(3)
```

Out[32]:
```
array([1., 1., 1.])
```

In [33]:
```
np. ones((3,3))
```

Out[33]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [34]:
```
np.ones(10)
```

Out[34]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

linspace

In [35]:
```
#return evenly spaced numbrs over specifeid interval
np.linspace(0,10,3) #
```

Out[35]:
```
array([ 0.,  5., 10.])
```

In [37]:
```
np.linspace(0,10,10)
```

Out[37]:
```
array([ 0.        ,  1.11111111,  2.22222222,  3.33333333,  4.44444444,
        5.55555556,  6.66666667,  7.77777778,  8.88888889, 10.        ])
```

In [38]:
```
np.linspace(0,50)
```

Out[38]:
```
array([ 0.        ,  1.02040816,  2.04081633,  3.06122449,  4.08163265,
        5.10204082,  6.12244898,  7.14285714,  8.16326531,  9.18367347,
       10.20408163, 11.2244898 , 12.24489796, 13.26530612, 14.28571429,
       15.30612245, 16.32653061, 17.34693878, 18.36734694, 19.3877551 ,
       20.40816327, 21.42857143, 22.44897959, 23.46938776, 24.48979592,
       25.51020408, 26.53061224, 27.55102041, 28.57142857, 29.59183673,
       30.6122449 , 31.63265306, 32.65306122, 33.67346939, 34.69387755,
       35.71428571, 36.73469388, 37.75510204, 38.7755102 , 39.79591837,
       40.81632653, 41.83673469, 42.85714286, 43.87755102, 44.89795918,
       45.91836735, 46.93877551, 47.95918367, 48.97959184, 50.        ])
```

In [39]:
```
np.linspace(0,10,50)
```

Out[39]:
```
array([ 0.        ,  0.20408163,  0.40816327,  0.6122449 ,  0.81632653,
        1.02040816,  1.2244898 ,  1.42857143,  1.63265306,  1.83673469,
        2.04081633,  2.24489796,  2.44897959,  2.65306122,  2.85714286,
        3.06122449,  3.26530612,  3.46938776,  3.67346939,  3.87755102,
        4.08163265,  4.28571429,  4.48979592,  4.69387755,  4.89795918,
        5.10204082,  5.30612245,  5.51020408,  5.71428571,  5.91836735,
        6.12244898,  6.32653061,  6.53061224,  6.73469388,  6.93877551,
        7.14285714,  7.34693878,  7.55102041,  7.75510204,  7.95918367,
        8.16326531,  8.36734694,  8.57142857,  8.7755102 ,  8.97959184,
        9.18367347,  9.3877551 ,  9.59183673,  9.79591837, 10.        ])
```

Eye

In [40]:
```
np.eye(4) #creates an identity matrix
```

Out[40]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

random

In [41]:
```
#numpy has  lot of ways to create random number arrays
np.random.rand(5,5) # it will give no negative values
```

Out[41]:
```
array([[0.26971167, 0.97019903, 0.17591429, 0.99693251, 0.88608796],
       [0.94934596, 0.86821605, 0.84585134, 0.58331574, 0.88794474],
       [0.14942708, 0.68452935, 0.33181952, 0.26155136, 0.62258731],
       [0.77984443, 0.76539698, 0.60736971, 0.73312084, 0.18180948],
       [0.5368952 , 0.2723795 , 0.57954522, 0.11043745, 0.65293749]])
```

randn

In [42]:
```
#"standard normal distribution", also gives negative values
np.random.randn(5,5)
```

Out[42]:
```
array([[ 2.0584569 , -0.34076043, -1.4698945 , -0.57664664, -1.20333004],
       [-0.73263387, -1.22265394,  2.43677976,  1.46013836, -0.90938338],
       [ 1.16245543, -0.59859169,  2.57492134, -0.10671111,  1.41286299],
       [ 0.90642941,  0.10678263,  1.07514512,  0.66088678, -2.79891867],
       [-0.89914077, -1.08102309, -0.09152619, -0.48204596,  0.77165233]])
```

randint

In [44]:
```
#return random integers from low (inclusive) to high (exclusive)
np.random.randint(1,100)
```

Out[44]:
```
74
```

In [45]:
```
np.random.randint(1,100,10)
```

Out[45]:
```
array([ 5, 38, 13, 49,  4, 88, 46, 66, 64, 54])
```

array attributes and methods

In [48]:
```
arr= np.arange(25)
arr
```

Out[48]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

In [ ]:
```

```

In [50]:
```
ranarr=np.random.randint(0,50,10)
ranarr
```

Out[50]:
```
array([20, 31, 10,  2, 14,  0, 25, 18,  3,  5])
```

max, min, argmax,argmin

In [51]:
```
ranarr
```

Out[51]:
```
array([20, 31, 10,  2, 14,  0, 25, 18,  3,  5])
```

In [54]:
```
ranarr.max()
```

Out[54]:
```
31
```

In [56]:
```
ranarr.min()
```

Out[56]:
```
0
```

In [57]:
```
ranarr.argmin()
```

Out[57]:
```
5
```

shape and reshape

In [58]:
```
#vector
arr.shape
```

Out[58]:
```
(25,)
```

In [60]:
```
arr.reshape(1,25)
```

Out[60]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
        16, 17, 18, 19, 20, 21, 22, 23, 24]])
```

In [62]:
```
a=arr.reshape(1,25)
a
```

Out[62]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
        16, 17, 18, 19, 20, 21, 22, 23, 24]])
```

In [63]:
```
#vector
b=[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
        16, 17, 18, 19, 20, 21, 22, 23, 24]
c=np.array(b)
c.shape
```

Out[63]:
```
(25,)
```

In [64]:
```
# list of integers 1,2,3,4,5,6,7,8,9 reshape into 3,3
l=[1,2,3,4,5,6,7,8,9]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [64], in <cell line: 3>()
      1 # list of integers 1,2,3,4,5,6,7,8,9 reshape into 3,3
      2 l=[1,2,3,4,5,6,7,8,9]
----> 3 c=np.array(3,3)
      4 c.shape

TypeError: Cannot interpret '3' as a data type
```

In [65]:
```
list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

m = np.array(list).reshape(3, 3)
print(m)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [66]:
```
m.shape
```

Out[66]:
```
(3, 3)
```

In [67]:
```

```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [67], in <cell line: 1>()
----> 1 list.shape

AttributeError: 'list' object has no attribute 'shape'
```

dtype

you can also grab the data type of object in the array

In [68]:
```
arr.dtype
```

Out[68]:
```
dtype('int32')
```

In [77]:
```
#create an empty array of 3,4 and full array of 3,3 with value 55
x=(np.empty(3,3),55)
x
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [77], in <cell line: 2>()
      1 #create an empty array of 3,4 and full array of 3,3 with value 55
----> 2 x=(np.empty(3,3),55)
      3 x

TypeError: Cannot interpret '3' as a data type
```

In [74]:
```
y=(np.full(3,3),55)
y
```

Out[74]:
```
(array([3, 3, 3]), 55)
```

bincount-used to get count of occurances of elements in the array

x=np.array([2,2,3,3,3,3,3,2,2,1,5,6,1]) y=np.bincount(x) y

In [80]:
```
y=np.bincount(x).max()
y
```

Out[80]:
```
4
```

In [81]:
```
#max value
y=np.bincount(x).argmax()
y
```

Out[81]:
```
2
```

In [82]:
```
#find the max value from the given list
#x = [4,18,2,8,3,15,14,15,20,12,6,3,15,12,13,19,14,81,23,44]
```

In [85]:
```
x = [4,18,2,8,3,15,14,15,20,12,6,3,15,12,13,19,14,81,23,44]
x=np.(x).max()
x
```

Out[85]:
```
3
```

In [ ]: