In [1]:
```
print("hello")
```

```
hello
```

In [3]:
```
Print("My Name Is Anand")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [3], in <cell line: 1>()
----> 1 Print("My Name Is Anand")

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

In [20]:
```
print("hello",end = '#')
```

```
hello#
```

In [21]:
```
print("hello",end = '$')
```

```
hello$
```

In [22]:
```
print("hello",end = '')
```

```
hello
```

In [24]:
```
print('''A
B
C
D
E
F''')
```

```
A
B
C
D
E
F
```

In [3]:
```
a = 5 ## CREWATING A VARIABLE
```

In [4]:
```
a
```

Out[4]:
```
5
```

In [5]:
```
type(a)
```

Out[5]:
```
int
```

In [6]:
```
b = 6.8
```

In [7]:
```
type(b)
```

Out[7]:
```
float
```

In [44]:
```
c = 'Hello'
```

In [45]:
```
type(c)
```

Out[45]:
```
str
```

In [46]:
```
len(c)
```

Out[46]:
```
5
```

In [47]:
```
print(c*3)
```

```
HelloHelloHello
```

In [12]:
```
d = 'A'
```

In [13]:
```
type(d)
```

Out[13]:
```
str
```

In [14]:
```
TYPE(d)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [14], in <cell line: 1>()
----> 1 TYPE(d)

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

In [16]:
```
a = 83.7
b = 31.9

print(a/b)
print(a*b)
print(a%b)
print(a//b)
print(a**b)
```

```
2.6238244514106586
2670.0299999999997
19.900000000000006
2.0
2.1624302597286839e+61
```

In [19]:
```
c = 20.0
d = 8.0

print(c//d) ## integral part of the division
print(c**d) ### power functions
```

```
2.0
25600000000.0
```

In [25]:
```
### LISTTTTTT
list_demo = [] ## empty list
```

In [26]:
```
type(list_demo)
```

Out[26]:
```
list
```

In [27]:
```
list_demo
```

Out[27]:
```
[]
```

In [28]:
```
list_demo = ["Anand",21.56,4,'d',"HELLO WORLD"] ##list of items
```

In [29]:
```
list_demo
```

Out[29]:
```
['Anand', 21.56, 4, 'd', 'HELLO WORLD']
```

In [30]:
```
len(list_demo)
```

Out[30]:
```
5
```

In [33]:
```
list_demo[0] ## first element
```

Out[33]:
```
'Anand'
```

In [34]:
```
list_demo[1] ## second element
```

Out[34]:
```
21.56
```

In [35]:
```
list_demo[2]
```

Out[35]:
```
4
```

In [36]:
```
list_demo[3]
```

Out[36]:
```
'd'
```

In [37]:
```
list_demo[4]
```

Out[37]:
```
'HELLO WORLD'
```

In [38]:
```
list_demo[-1] ## last element
```

Out[38]:
```
'HELLO WORLD'
```

In [39]:
```
list_demo[-2] ## second last el;ement
```

Out[39]:
```
'd'
```

In [40]:
```
list_demo[-3]
```

Out[40]:
```
4
```

In [41]:
```
list_demo[-4]
```

Out[41]:
```
21.56
```

In [49]:
```
list_demo = list_demo[::-1] ## reverse the elemenmts of the list
```

In [50]:
```
list_demo
```

Out[50]:
```
['HELLO WORLD', 'd', 4, 21.56, 'Anand']
```

In [51]:
```
list_demo[0]
```

Out[51]:
```
'HELLO WORLD'
```

In [52]:
```
 list_demo[2] = "Test"
```

In [53]:
```
list_demo
```

Out[53]:
```
['HELLO WORLD', 'd', 'Test', 21.56, 'Anand']
```

In [54]:
```
del list_demo[1]
```

In [55]:
```
list_demo
```

Out[55]:
```
['HELLO WORLD', 'Test', 21.56, 'Anand']
```

In [60]:
```
list_dup = ["Anand",21.56,4,'d',"Anand",21.56]  ##list of items
```

In [61]:
```
list_dup
```

Out[61]:
```
['Anand', 21.56, 4, 'd', 'Anand', 21.56]
```

In [64]:
```
### setttttt
dict_empty = {}
```

In [65]:
```
type(dict_empty)
```

Out[65]:
```
dict
```

In [66]:
```
set_demo = {2,3,"Anand"}
```

In [67]:
```
type(set_demo)
```

Out[67]:
```
set
```

In [68]:
```
set_dup = {2,3,"Anand",2,3,"Anand","Manisha"}
```

In [69]:
```
set_dup
```

Out[69]:
```
{2, 3, 'Anand', 'Manisha'}
```

In [70]:
```
set_dup[2]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [70], in <cell line: 1>()
----> 1 set_dup[2]

TypeError: 'set' object is not subscriptable
```

In [71]:
```
len(set_dup)
```

Out[71]:
```
4
```

In [75]:
```
set_dup
```

Out[75]:
```
{2, 3, 'Anand', 'Manisha'}
```

In [72]:
```
for i in set_dup:
    print(i)
```

```
Manisha
2
3
Anand
```

In [73]:
```
set_ex = {4,5,6,7,3,4,7,5,4,1,2,0,5656,4564,234,234,2342,"abc","##","anand","chithı
```

In [74]:
```
set_ex ## there is an order but sorted
```

Out[74]:
```
{'##',
 0,
 1,
 2,
 23.56,
 234,
 2342,
 3,
 4,
 4564,
 5,
 5656,
 6,
 7,
 'abc',
 'anand',
 'chithra'}
```

In [77]:
```
for i in set_ex:
    print(i)
```

```
0
1
2
3
4
5
6
7
4564
23.56
5656
chithra
2342
234
abc
##
anand
```

In [78]:
```
##### range function
range(10)
```

Out[78]:
```
range(0, 10)
```

In [79]:
```
print(range(10))
```

```
range(0, 10)
```

In [80]:
```
print(list(range(10)))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [81]:
```
list(range(3,10))
```

Out[81]:
```
[3, 4, 5, 6, 7, 8, 9]
```

In [82]:
```
list(range(10,3))
```

Out[82]:
```
[]
```

In [83]:
```
list(range(3,10,2)) #numbers from strat to end-1 by jumping 2 numbers in between
```

Out[83]:
```
[3, 5, 7, 9]
```

In [84]:
```
list(range(3.5,10))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [84], in <cell line: 1>()
----> 1 list(range(3.5,10))

TypeError: 'float' object cannot be interpreted as an integer
```

In [85]:
```
l = [1,2,3,4,5,6,7,23,25]
```

In [86]:
```
empty_list = []
```

In [87]:
```
len(l)
```

Out[87]:
```
9
```

In [88]:
```
for i in range(len(l)):
    empty_list.append(l[i] + 2)
```

In [89]:
```
empty_list
```

Out[89]:
```
[3, 4, 5, 6, 7, 8, 9, 25, 27]
```

In [90]:
```
even_list =[]
odd_list = []
```

In [91]:
```
for i in range(len(l)):
    if(l[i] % 2 == 0):
        even_list.append(l[i])
    else:
        odd_list.append(l[i])
```

In [92]:
```
even_list
```

Out[92]:
```
[2, 4, 6]
```

In [93]:
```
odd_list
```

Out[93]:
```
[1, 3, 5, 7, 23, 25]
```

In [94]:
```
### input from the user
a = input()
```

```
Hello
```

In [95]:
```
a
```

Out[95]:
```
'Hello'
```

In [96]:
```
type(a)
```

Out[96]:
```
str
```

In [97]:
```
### design a calculator
num1 = input("Enter first number")
num2 = input("Enter Second Number")

res = num1 + num2
print(res)
```

```
Enter first number23
Enter Second Number45
2345
```

In [98]:
```
"Anand" + "Manisha"
```

Out[98]:
```
'AnandManisha'
```

In [2]:
```
### design a calculator
num1 = int(input("Enter first number "))
num2 = int(input("Enter Second Number "))

add = num1 + num2
sub = num1 - num2
mult = num1 * num2
div = num1 / num2
rem = num1 % num2
power = num1 ** num2

print("Addition : ",+add)
print("Subtraction : ",+sub)
print("Multiplication : ",+mult)
print("Division : ",+div)
print("Remainder : ",+rem)
print("Power : ",+power)
```

```
Enter first number 23
Enter Second Number 56
Addition :  79
Subtraction :  -33
Multiplication :  1288
Division :  0.4107142857142857
Remainder :  23
Power :  18061708005752857061620870906539210750802836284237842525970767008798325863361
```

In [3]:
```
var = "India Is My Country"
```

In [7]:
```
## find
var.find('M') # FIRST OCCURENCCE OF AN ALPHABET
```

Out[7]:
```
9
```

In [6]:
```
var.find('Z') ## if search character  doesnt exist in the string it gives -1
```

Out[6]:
```
-1
```

In [8]:
```
## split
var.split('M') ## doesnt include the split character in the output
```

Out[8]:
```
['India Is ', 'y Country']
```

In [9]:
```
## PARTITION
var.partition('M') ## includes the split character in the output
```

Out[9]:
```
('India Is ', 'M', 'y Country')
```

In [10]:
```
## center
var.center(50,'*')
```

Out[10]:
```
'***************India Is My Country***************'
```

In [1]:
```
text = input("Please tell me your name ")
```

```
Please tell me your name Pooja Ganekar
```

In [2]:
```
"My name is {}".format(text) ## to replace the data at run time
```

Out[2]:
```
'My name is Pooja Ganekar'
```

In [4]:
```
b = [4,6,7,8,'my','name']
```

In [5]:
```
b.append('anand')
```

In [6]:
```
b
```

Out[6]:
```
[4, 6, 7, 8, 'my', 'name', 'anand']
```

In [7]:
```
b.insert(2,"Hello") ## insert at 2nd index and shift all further
```

In [8]:
```
b
```

Out[8]:
```
[4, 6, 'Hello', 7, 8, 'my', 'name', 'anand']
```

In [9]:
```
b.insert(-1,"Manisha") ## shift 1 location backwards
```

In [10]:
```
b
```

Out[10]:
```
[4, 6, 'Hello', 7, 8, 'my', 'name', 'Manisha', 'anand']
```

In [11]:
```
b[::-1] ## original data doesnt gets changed
```

Out[11]:
```
['anand', 'Manisha', 'name', 'my', 8, 7, 'Hello', 6, 4]
```

In [12]:
```
b
```

Out[12]:
```
[4, 6, 'Hello', 7, 8, 'my', 'name', 'Manisha', 'anand']
```

In [13]:
```
b.reverse() ## original data gets chnaged too
```

In [14]:
```
b
```

Out[14]:
```
['anand', 'Manisha', 'name', 'my', 8, 7, 'Hello', 6, 4]
```

In [7]:
```
list_sort = [2,3,45,36,97,12,56,89]
```

In [8]:
```
list_asc = list_sort.sort()
```

In [9]:
```
list_asc
```

In [10]:
```
list_desc = list_sort.sort(reverse=True) ## descending
```

In [13]:
```
list_desc
```

In [12]:
```
print("hello")
```

```
hello
```

In [27]:
```
b ={2,3,4,5,6,7,8,8,9}
```

In [28]:
```
type(b)
```

Out[28]:
```
set
```

In [25]:
```
b.pop()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [25], in <cell line: 1>()
----> 1 b.pop()

KeyError: 'pop from an empty set'
```

In [29]:
```
b
```

Out[29]:
```
{2, 3, 4, 5, 6, 7, 8, 9}
```

In [38]:
```
## tuples
t = (2,2,3,4,5,6)
```

In [39]:
```
type(t)
```

Out[39]:
```
tuple
```

In [41]:
```
t
```

Out[41]:
```
(2, 2, 3, 4, 5, 6)
```

In [42]:
```
t.count(2)
```

Out[42]:
```
2
```

In [43]:
```
t.count(3)
```

Out[43]:
```
1
```

In [44]:
```
t.index(3)
```

Out[44]:
```
2
```

In [45]:
```
t.index(2)
```

Out[45]:
```
0
```

In [46]:
```
t.append(8)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [46], in <cell line: 1>()
----> 1 t.append(8)

AttributeError: 'tuple' object has no attribute 'append'
```

In [47]:
```
t.extend(7)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [47], in <cell line: 1>()
----> 1 t.extend(7)

AttributeError: 'tuple' object has no attribute 'extend'
```

In [49]:
```
t1 = (1,2,3,4,5,6)
t2 = (5,6,7,8,9,10)

res = t1 + t2

print(res)
```

```
(1, 2, 3, 4, 5, 6, 5, 6, 7, 8, 9, 10)
```

In [63]:
```
## tuples objects and strings are immutable i.e you cant change it
t[1] = 9
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [63], in <cell line: 2>()
      1 ## tuples objects are immutable
----> 2 t[1] = 9

TypeError: 'tuple' object does not support item assignment
```

In [64]:
```
str_demo = "anand"
str_demo[3] = 't'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [64], in <cell line: 2>()
      1 str_demo = "anand"
----> 2 str_demo[3] = 't'

TypeError: 'str' object does not support item assignment
```

In [83]:
```
s = [1,2,3,4,5,5]
```

In [84]:
```
s.append([22,35,45,67]) ## adds the element as it is at the end of the list
```

In [98]:
```
s
```

Out[98]:
```
[1, 2, 3, 4, 5, 5, [22, 35, 45, 67]]
```

In [99]:
```
s = s[::-1]
```

In [100…
```
s
```

Out[100]:
```
[[22, 35, 45, 67], 5, 5, 4, 3, 2, 1]
```

In [104…
```
n = []
```

In [103…
```
n
```
Out[103]:
```
[[22, 35, 45, 67], 5, 5, 4, 3, 2, 1]
```

In [106…
```
a = [3,4,5,6,7]
b = [56,6,7,89,8]
c = [2,3,4,5,6,7,8,8,9,0]
d =[a,b,c]
```

In [107…
```
a
```
Out[107]:
```
[3, 4, 5, 6, 7]
```

In [108…
```
b
```
Out[108]:
```
[56, 6, 7, 89, 8]
```

In [109…
```
c
```
Out[109]:
```
[2, 3, 4, 5, 6, 7, 8, 8, 9, 0]
```

In [110…
```
d
```
Out[110]:
```
[[3, 4, 5, 6, 7], [56, 6, 7, 89, 8], [2, 3, 4, 5, 6, 7, 8, 8, 9, 0]]
```

In [111…
```
d[-1]
```
Out[111]:
```
[2, 3, 4, 5, 6, 7, 8, 8, 9, 0]
```

In [112…
```
d[-1][::2]
```
Out[112]:
```
[2, 4, 6, 8, 9]
```

In [113…
```
d[::-1]
```
Out[113]:
```
[[2, 3, 4, 5, 6, 7, 8, 8, 9, 0], [56, 6, 7, 89, 8], [3, 4, 5, 6, 7]]
```

In [114…
```
rev_d = d[::-1]
```

In [115…
```
rev_d
```
Out[115]:
```
[[2, 3, 4, 5, 6, 7, 8, 8, 9, 0], [56, 6, 7, 89, 8], [3, 4, 5, 6, 7]]
```

In [116…
```
empty_rev_list = []
```

In [117…
```
for i in rev_d: ### reversal of a list inside a list
    empty_rev_list.append(i[::-1])
```

In [118…
```
empty_rev_list
```
Out[118]:
```
[[0, 9, 8, 8, 7, 6, 5, 4, 3, 2], [8, 89, 7, 6, 56], [7, 6, 5, 4, 3]]
```

In [120…
```
##list comprehension
[i[3] * 2 for i in empty_rev_list] #3picking 2nd index element from every list and
```
Out[120]:
```
[16, 12, 8]
```

In [ ]:

In [ ]:

In [55]:
```
s[4]
```

Out[55]:
```
5
```

In [58]:
```
s[6][1]
```

Out[58]:
```
35
```

In [60]:
```
s[6][1:3]
```

Out[60]:
```
[35, 45]
```

In [62]:
```
s.index(22) # starting index will be 1
```

Out[62]:
```
7
```

In [53]:
```
s.extend([22,35,45,67]) ## wraps up the data and then insert
```

In [54]:
```
s
```

Out[54]:
```
[1, 2, 3, 4, 5, 5, [22, 35, 45, 67], 22, 35, 45, 67]
```

In [121…
```
########################## dictionary ##############################################
## dictionary has key and value pair
```

In [122…
```
dict_test = {"key1":[3,4,5,6],'key2':["Anand","Pankaj","Manisha","Chithra"]}
```

In [123…
```
dict_test
```

Out[123]:
```
{'key1': [3, 4, 5, 6], 'key2': ['Anand', 'Pankaj', 'Manisha', 'Chithra']}
```

In [124…
```
dict_test.keys()
```

Out[124]:
```
dict_keys(['key1', 'key2'])
```

In [125…
```
dict_test.values()
```

Out[125]:
```
dict_values([[3, 4, 5, 6], ['Anand', 'Pankaj', 'Manisha', 'Chithra']])
```

In [126…
```
dict_test['key1']
```

Out[126]:
```
[3, 4, 5, 6]
```

In [127…
```
dict_test['key1'][2]
```

Out[127]:
```
5
```

In [128…
```
dict_test['key2'][2]
```

Out[128]:
```
'Manisha'
```

In [129…
```
dict_test['key2'][2] = "Rakesh"
```

In [130…
```
dict_test['key2']
```

Out[130]:
```
['Anand', 'Pankaj', 'Rakesh', 'Chithra']
```

In [131…
```
for i in dict_test:
    print(i)
```

```
key1
key2
```

In [132…
```
for i in dict_test.values():
    print(i)
```

```
[3, 4, 5, 6]
['Anand', 'Pankaj', 'Rakesh', 'Chithra']
```

In [133…
```
for i in dict_test.keys():
    print(dict_test[i])
```

```
[3, 4, 5, 6]
['Anand', 'Pankaj', 'Rakesh', 'Chithra']
```

In [135…
```
dict_complex = {"key1" : [1,2,3,4,5,[345,45]],
                "key2" : (3,3,4,5,6),
                "key3" : {'x':"Anand",'y':"Rakesh"}
               }
```

In [136…
```
dict_complex
```

Out[136]:
```
{'key1': [1, 2, 3, 4, 5, [345, 45]],
 'key2': (3, 3, 4, 5, 6),
 'key3': {'x': 'Anand', 'y': 'Rakesh'}}
```

In [137…
```
dict_complex.keys()
```

Out[137]:
```
dict_keys(['key1', 'key2', 'key3'])
```

In [138…
```
dict_complex.values()
```

Out[138]:
```
dict_values([[1, 2, 3, 4, 5, [345, 45]], (3, 3, 4, 5, 6), {'x': 'Anand', 'y': 'Rak
esh'}])
```

In [139…
```
dict_complex['key3']['x']
```

Out[139]:
```
'Anand'
```

In [142…
```
dict_complex['key1'][5][1]
```

Out[142]:
```
45
```

In [143…
```
dict_complex['key2'][3]
```

Out[143]:
```
5
```

In [144…
```
###whereevr tuples are there in my  dictionary copy that to an empty list
empty = []

for i in dict_complex.values():
    if type(i) == tuple:
        for j in i:
            empty.append(j)
```

In [145…
```
empty
```

Out[145]:
```
[3, 3, 4, 5, 6]
```

In [146…
```
## dictionary comprehension
{i:i ** 2 for i in range(10)}
```

Out[146]:
```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

In [147…
```
l = ["anand","RAKESH","POOJA","pankaj","chithra"]
```

In [148…
```
d = {}
```

In [149…
```
for i in l:
    d[i] = i.upper()
```

In [150…
```
d
```

Out[150]:
```
{'anand': 'ANAND',
 'RAKESH': 'RAKESH',
 'POOJA': 'POOJA',
 'pankaj': 'PANKAJ',
 'chithra': 'CHITHRA'}
```

In [151…
```
e = {}
```

In [152…
```
for i in l:
    e[i] = i.lower()
```

In [153…
```
e
```

Out[153]:
```
{'anand': 'anand',
 'RAKESH': 'rakesh',
 'POOJA': 'pooja',
 'pankaj': 'pankaj',
 'chithra': 'chithra'}
```

In [2]:
```
import numpy as np
import pandas as pd
```

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

In [164…
```
arr = np.array([[4,5.5,6],[7,8,9]])
```

In [165…
```
arr
```

Out[165]:
```
array([[4. , 5.5, 6. ],
       [7. , 8. , 9. ]])
```

In [158…
```
type(arr) #type of array
```

Out[158]:
```
numpy.ndarray
```

In [159…
```
arr.ndim # how many rows
```

Out[159]:
```
2
```

In [161…
```
arr.shape # number of rows and columns
```

Out[161]:
```
(2, 3)
```

In [162…
```
arr.size ## rows x columns
```

Out[162]:
```
6
```

In [166…
```
arr.dtype
```

Out[166]:
```
dtype('float64')
```

In [167…
```
arr = np.array([[4,5.5,6,"Anand"],[7,8,9]])
```

```
C:\Users\ajha2\AppData\Local\Temp\ipykernel_20960\3406014508.py:1: VisibleDeprecat
ionWarning: Creating an ndarray from ragged nested sequences (which is a list-or-t
uple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecate
d. If you meant to do this, you must specify 'dtype=object' when creating the ndar
ray.
  arr = np.array([[4,5.5,6,"Anand"],[7,8,9]])
```

In [168…
```
arr
```

Out[168]:
```
array([list([4, 5.5, 6, 'Anand']), list([7, 8, 9])], dtype=object)
```

In [169…
```
arr_ex = np.array([[6,5,1],[2,7,4],[9,1,3]])
```

In [170…
```
arr_ex
```

Out[170]:
```
array([[6, 5, 1],
       [2, 7, 4],
       [9, 1, 3]])
```

In [171…
```
arr_ex.max()
```

Out[171]:
```
9
```

In [172…
```
arr_ex.min() ##minm of al the elments
```

Out[172]:
```
1
```

In [173…
```
arr_ex.max(axis=0) ##column wise maxm elements
```

Out[173]:
```
array([9, 7, 4])
```

In [174…
```
arr_ex.max(axis=1) ##rowwise max elemsts
```

Out[174]:
```
array([6, 7, 9])
```

In [175…
```
arr_ex.sum() ##sm of all the elemnts
```

Out[175]:
```
38
```

In [176…
```
arr_ex.sum(axis=0) ##columnwise sum
```

Out[176]:
```
array([17, 13,  8])
```

In [177…
```
arr_ex.sum(axis=1) ##rowwise sum
```

Out[177]:
```
array([12, 13, 13])
```

In [3]:
```
arr1 = np.random.randint(10, 50, size = (5, 8)) ## genearting 2D array with 5 rows
```

In [4]:
```
arr1
```

Out[4]:
```
array([[49, 26, 29, 21, 20, 40, 20, 38],
       [46, 13, 30, 49, 12, 10, 42, 13],
       [27, 37, 23, 48, 43, 31, 24, 48],
       [32, 32, 25, 15, 39, 17, 11, 38],
       [36, 28, 13, 28, 27, 40, 19, 41]])
```

In [5]:
```
arr2 = np.random.randint(1, 20, size = (2, 3, 6)) ## genearting 3D array with 3 rou
```

In [6]:
```
arr2
```

Out[6]:
```
array([[[ 1, 18,  8,  3, 14,  4],
        [14, 12,  5, 18,  9, 15],
        [19, 15,  6, 12, 16, 18]],

       [[19,  9,  9, 14, 14, 16],
        [ 8, 14, 10,  5,  4,  2],
        [15, 16, 17,  2, 18, 17]]])
```

In [12]:
```
a = np.array([0,2,3,0,-1,6,5,-2])
```

In [13]:
```
a
```

Out[13]:
```
array([ 0,  2,  3,  0, -1,  6,  5, -2])
```

In [14]:
```
np.greater(a,0)
```

Out[14]:
```
array([False,  True,  True, False, False,  True,  True, False])
```

In [15]:
```
np.greater(0,a)
```

Out[15]:
```
array([False, False, False, False,  True, False, False,  True])
```

In [16]:
```
np.greater(a,2)
```

Out[16]:
```
array([False, False,  True, False, False,  True,  True, False])
```

In [17]:
```
arr1 = np.random.randint(10, 50, size = (5, 8))
```

In [18]:
```
arr1
```

Out[18]:
```
array([[35, 44, 33, 41, 10, 35, 10, 22],
       [49, 48, 21, 45, 42, 39, 16, 39],
       [11, 27, 20, 38, 42, 32, 18, 35],
       [48, 15, 22, 22, 27, 43, 49, 20],
       [18, 10, 36, 26, 49, 22, 24, 44]])
```

In [19]:
```
np.greater(arr1,10)
```

Out[19]:
```
array([[ True,  True,  True,  True, False,  True, False,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True,  True,  True],
       [ True, False,  True,  True,  True,  True,  True,  True]])
```

In [21]:
```
np.greater_equal(arr1,32)
```

Out[21]:
```
array([[ True,  True,  True,  True, False,  True, False, False],
       [ True,  True, False,  True,  True,  True, False,  True],
       [False, False, False,  True,  True,  True, False,  True],
       [ True, False, False, False, False,  True,  True, False],
       [False, False,  True, False,  True, False, False,  True]])
```

In [22]:
```
np.less(arr1,32)
```

Out[22]:
```
array([[False, False, False, False,  True, False,  True,  True],
       [False, False,  True, False, False, False,  True, False],
       [ True,  True,  True, False, False, False,  True, False],
       [False,  True,  True,  True,  True, False, False,  True],
       [ True,  True, False,  True, False,  True,  True, False]])
```

In [23]:
```
np.less_equal(arr1,32)
```

Out[23]:
```
array([[False, False, False, False,  True, False,  True,  True],
       [False, False,  True, False, False, False,  True, False],
       [ True,  True,  True, False, False,  True,  True, False],
       [False,  True,  True,  True,  True, False, False,  True],
       [ True,  True, False,  True, False,  True,  True, False]])
```

In [24]:
```
## A boolean array is a NumPy array with boolean (True/False) values. By applying c
## operator to another NumPy array, such array can be obtained:
a = np.reshape(np.arange(25),(5,5))
```

In [25]:
```
a
```

Out[25]:
```
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 [28]:
```
greater_values = (a%2 != 0) ## oddd check
greater_values
```

Out[28]:
```
array([[False,  True, False,  True, False],
       [ True, False,  True, False,  True],
       [False,  True, False,  True, False],
       [ True, False,  True, False,  True],
       [False,  True, False,  True, False]])
```

In [29]:
```
greater_values = (a%2 == 0) ## even check
greater_values
```

Out[29]:
```
array([[ True, False,  True, False,  True],
       [False,  True, False,  True, False],
       [ True, False,  True, False,  True],
       [False,  True, False,  True, False],
       [ True, False,  True, False,  True]])
```

In [30]:
```
b = ~(a%3 == 0) # check numbers which are not divisible by 3
b
```

Out[30]:
```
array([[False,  True,  True, False,  True],
       [ True, False,  True,  True, False],
       [ True,  True, False,  True,  True],
       [False,  True,  True, False,  True],
       [ True, False,  True,  True, False]])
```

In [31]:
```
c = (a%2 == 0) | (a%3 == 0)
c
```

Out[31]:
```
array([[ True, False,  True,  True,  True],
       [False,  True, False,  True,  True],
       [ True, False,  True, False,  True],
       [ True,  True, False,  True, False],
       [ True,  True,  True, False,  True]])
```

In [32]:
```
c = (a%2 == 0) & (a%3 == 0)
```

In [33]:
```
c
```

Out[33]:
```
array([[ True, False, False, False, False],
       [False,  True, False, False, False],
       [False, False,  True, False, False],
       [False, False, False,  True, False],
       [False, False, False, False,  True]])
```

In [34]:
```
''' In numpy.ma.mask_rows() function, mask rows of a 2-Dimensional array which hold
masked values. The numpy.ma.mask_rows() function is a shortcut to mask_rowcols
with axis equal to 0 '''
```

Out[34]:
```
' In numpy.ma.mask_rows() function, mask rows of a 2-Dimensional array which hold
\nmasked values. The numpy.ma.mask_rows() function is a shortcut to mask_rowcols\n
with axis equal to 0'
```

In [39]:
```
import numpy.ma as MA
```

In [35]:
```
array = np.zeros((4,4),dtype = int)
```

In [36]:
```
array
```

Out[36]:
```
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
```

In [37]:
```
array[2,2] = 1
```

In [38]:
```
array
```

Out[38]:
```
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 0]])
```

In [40]:
```
array = MA.masked_equal(array,1)
```

In [41]:
```
array
```

Out[41]:
```
masked_array(
  data=[[0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, --, 0],
        [0, 0, 0, 0]],
  mask=[[False, False, False, False],
        [False, False, False, False],
        [False, False,  True, False],
        [False, False, False, False]],
  fill_value=1)
```

In [47]:
```
masking_row = MA.mask_rows(array)
```

In [48]:
```
masking_row
```

Out[48]:
```
masked_array(
  data=[[0, 0, 0, 0],
        [0, 0, 0, 0],
        [--, --, --, --],
        [0, 0, 0, 0]],
  mask=[[False, False, False, False],
        [False, False, False, False],
        [ True,  True,  True,  True],
        [False, False, False, False]],
  fill_value=1)
```

In [49]:
```
masking_col = MA.mask_cols(array)
```

In [50]:
```
masking_col
```

Out[50]:
```
masked_array(
  data=[[0, 0, --, 0],
        [0, 0, --, 0],
        [0, 0, --, 0],
        [0, 0, --, 0]],
  mask=[[False, False,  True, False],
        [False, False,  True, False],
        [False, False,  True, False],
        [False, False,  True, False]],
  fill_value=1)
```

In [55]:
```
##fancy indexing
x = np.random.randint(100,size = 10)
x
```

Out[55]:
```
array([45, 47, 42,  2, 21, 81, 16,  4, 23, 92])
```

In [56]:
```
[x[3],x[6],x[8]]
```

Out[56]:
```
[2, 16, 23]
```

In [57]:
```
ind = [3,6,8]
x[ind]
```

Out[57]:
```
array([ 2, 16, 23])
```

In [58]:
```
# While utilising fancy indexing, the shape of the result reflects the shape of the
a# rrays instead of the shape of the array being indexed:

ind = np.array([[3,7],[4,5]])
x[ind]
```

Out[58]:
```
array([[ 2,  4],
       [21, 81]])
```

In [59]:
```
x = np.arange(12).reshape((3,4))
x
```

Out[59]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [60]:
```
row = np.array([0,1,2])
col = np.array([2,1,3])

x[row,col]
```

Out[60]:
```
array([ 2,  5, 11])
```

In [61]:
```
''' The broadcasting rules are followed by the pairing of indices in fancy indexing
Therefore, for instance, we get a two-dimensional result if we combine a column vec
as well as a row vector within the indices:'''
```

Out[61]:
```
' The broadcasting rules are followed by the pairing of indices in fancy indexin
g.\nTherefore, for instance, we get a two-dimensional result if we combine a colum
n vector\nas well as a row vector within the indices:'
```

In [63]:
```
x[row[:,np.newaxis],col]
```

Out[63]:
```
array([[ 2,  1,  3],
       [ 6,  5,  7],
       [10,  9, 11]])
```

In [64]:
```
x1 = np.array([1, 2, 3, 4, 5])
x2 = np.array([5, 4, 3])
```

In [65]:
```
x1+x2
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [65], in <cell line: 1>()
----> 1 x1+x2

ValueError: operands could not be broadcast together with shapes (5,) (3,)
```

In [66]:
```
x1_new = x1[:, np.newaxis]    # x1[:, None]
```

In [67]:
```
x1 + x1_new
```

Out[67]:
```
array([[ 2,  3,  4,  5,  6],
       [ 3,  4,  5,  6,  7],
       [ 4,  5,  6,  7,  8],
       [ 5,  6,  7,  8,  9],
       [ 6,  7,  8,  9, 10]])
```

In [68]:
```
x1[np.newaxis, :]
```

Out[68]:
```
array([[1, 2, 3, 4, 5]])
```

In [69]:
```
 x1[:, np.newaxis]
```

Out[69]:
```
array([[1],
       [2],
       [3],
       [4],
       [5]])
```

In [70]:
```
row = np.array([0,1,2])
col = np.array([2,1,3])

x[row,col]
```

Out[70]:
```
array([ 2,  5, 11])
```

In [71]:
```
x
```

Out[71]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [72]:
```
x[2,[2,0,1]]
```

Out[72]:
```
array([10,  8,  9])
```

In [73]:
```
## sorting mechnaism
a = np .array(['Mango','PineApple','Banana'])
```

In [74]:
```
b = np.sort(a)
```

In [75]:
```
b
```

Out[75]:
```
array(['Banana', 'Mango', 'PineApple'], dtype='<U9')
```

In [77]:
```
array_sort = np.array([[5,0,1],[9,3,6]])
```

In [78]:
```
array_sort
```

Out[78]:
```
array([[5, 0, 1],
       [9, 3, 6]])
```

In [82]:
```
np.sort(array_sort)
```

Out[82]:
```
array([[0, 1, 5],
       [3, 6, 9]])
```

In [89]:
```
## structured array

a = np.array([('Jack',3,22.0),('Mark',8,28.0),('Steven',24,36.4)],
             dtype = [('name',(np.str_,10)),('age',np.int32),('weight',np.float64)])
```

In [90]:
```
a
```

Out[90]:
```
array([('Jack',  3, 22. ), ('Mark',  8, 28. ), ('Steven', 24, 36.4)],
      dtype=[('name', '<U10'), ('age', '<i4'), ('weight', '<f8')])
```

In [91]:
```
a_sort_age = np.sort(a,order = 'age')
a_sort_age
```

Out[91]:
```
array([('Jack',  3, 22. ), ('Mark',  8, 28. ), ('Steven', 24, 36.4)],
      dtype=[('name', '<U10'), ('age', '<i4'), ('weight', '<f8')])
```

In [92]:
```
a_sort_name = np.sort(a,order = 'name')
a_sort_name
```

Out[92]:
```
array([('Jack',  3, 22. ), ('Mark',  8, 28. ), ('Steven', 24, 36.4)],
      dtype=[('name', '<U10'), ('age', '<i4'), ('weight', '<f8')])
```

In [93]:
```
a_sort_wt = np.sort(a,order = 'weight')
a_sort_wt
```

Out[93]:
```
array([('Jack',  3, 22. ), ('Mark',  8, 28. ), ('Steven', 24, 36.4)],
      dtype=[('name', '<U10'), ('age', '<i4'), ('weight', '<f8')])
```

In [94]:
```
######################################################## PANDAS ######################
```

In [95]:
```
import pandas as pd
```

In [96]:
```
series_data = pd.Series([0.25,0.5,0.75,1.0])
```

In [97]:
```
series_data
```

Out[97]:
```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
```

In [98]:
```
series_data.index
```

Out[98]:
```
RangeIndex(start=0, stop=4, step=1)
```

In [99]:
```
series_data.values
```

Out[99]:
```
array([0.25, 0.5 , 0.75, 1.  ])
```

In [100…
```
series_data[1:3]
```

Out[100]:
```
1    0.50
2    0.75
dtype: float64
```

In [101…
```
series_data_named_index = pd.Series([0.25,0.5,0.75,1.0],index = ['a','b','c','d'])
```

In [102…
```
series_data_named_index
```

Out[102]:
```
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
```

In [103…
```
series_data_named_index['c']
```

Out[103]:
```
0.75
```

In [104…
```
series_data_named_index['a':'c']
```

Out[104]:
```
a    0.25
b    0.50
c    0.75
dtype: float64
```

In [105…
```
area_dict = {'California':423967,
             'Texas' : 695662,
             'New York' : 141297,
             'Florida' : 170312,
             'Illinois' : 149995}

area = pd.Series(area_dict)
area
```

Out[105]:
```
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995
dtype: int64
```

In [129…
```
area.keys()
```

Out[129]:
```
Index(['California', 'Texas', 'New York', 'Florida', 'Illinois'], dtype='object')
```

In [131…
```
list(area.items())
```

Out[131]:
```
[('California', 423967),
 ('Texas', 695662),
 ('New York', 141297),
 ('Florida', 170312),
 ('Illinois', 149995)]
```

In [123…
```
pd.DataFrame(area,columns = ['Population'])
```

Out[123]:

	Population
California	423967
Texas	695662
New York	141297
Florida	170312
Illinois	149995

In [119…
```
states = pd.DataFrame({ 'Population':area.index,
                        'Area': area.values},index=[1,2,3,4,5])
states
```

Out[119]:

	Population	Area
1	California	423967
2	Texas	695662
3	New York	141297
4	Florida	170312
5	Illinois	149995

In [120…
```
states.columns
```

Out[120]:
```
Index(['Population', 'Area'], dtype='object')
```

In [121…
```
states.index
```

Out[121]:
```
Int64Index([1, 2, 3, 4, 5], dtype='int64')
```

In [122…
```
states['Area']
```

Out[122]:
```
1    423967
2    695662
3    141297
4    170312
5    149995
Name: Area, dtype: int64
```

In [124…
```
data = [{'a':i,'b': 2*i} for i in range(3)]
```

In [125…
```
data
```

Out[125]:
```
[{'a': 0, 'b': 0}, {'a': 1, 'b': 2}, {'a': 2, 'b': 4}]
```

In [127…
```
pd.DataFrame(data)
```

Out[127]:

	a	b
0	0	0
1	1	2
2	2	4

In [128…
```
pd.DataFrame([{'a':1,'b':2},{'b':3,'c':4}])
```

Out[128]:

	a	b	c
0	1.0	2	NaN
1	NaN	3	4.0

In [132…
```
states
```

Out[132]:

	Population	Area
1	California	423967
2	Texas	695662
3	New York	141297
4	Florida	170312
5	Illinois	149995

In [134…
```
states.loc[3]
```

Out[134]:
```
Population    New York
Area            141297
Name: 3, dtype: object
```

In [135…
```
states.loc[1:3]
```

Out[135]:

	Population	Area
1	California	423967
2	Texas	695662
3	New York	141297

In [136…
```
states.loc[:3]
```

Out[136]:

	Population	Area
1	California	423967
2	Texas	695662
3	New York	141297

In [137…
```
states.loc[2:]
```

Out[137]:

	Population	Area
2	Texas	695662
3	New York	141297
4	Florida	170312
5	Illinois	149995

In [138…
```
states.iloc[3]
```

Out[138]:
```
Population    Florida
Area           170312
Name: 4, dtype: object
```

In [139…
```
states.loc[3]
```

Out[139]:
```
Population    New York
Area            141297
Name: 3, dtype: object
```

In [140…
```
area_dict = {'California':423967,
             'Texas' : 695662,
             'New York' : 141297,
             'Florida' : 170312,
             'Illinois' : 149995}

area = pd.Series(area_dict)
area
```

Out[140]:
```
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995
dtype: int64
```

In [141…
```
pop_dict = {'California':38332521,
            'Texas' : 763842356,
            'New York' : 4895732,
            'Florida' : 32432345,
            'Illinois' : 1244213}

pop = pd.Series(pop_dict)
pop
```

Out[141]:
```
California     38332521
Texas         763842356
New York        4895732
Florida        32432345
Illinois        1244213
dtype: int64
```

In [142…
```
data_com = pd.DataFrame({'Area':area,"Population":pop})
data_com
```

Out[142]:

	Area	Population
California	423967	38332521
Texas	695662	763842356
New York	141297	4895732
Florida	170312	32432345
Illinois	149995	1244213

In [146…
```
data_com['Area']
```

Out[146]:
```
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995
Name: Area, dtype: int64
```

In [147…
```
data_com.Area
```

Out[147]:
```
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995
Name: Area, dtype: int64
```

In [148…
```
data_com.Area is data_com['Area']
```

Out[148]:
```
True
```

In [149…
```
data_com['Density'] = data_com['Population'] / data_com['Area']
```

In [150…
```
data_com
```

Out[150]:

	Area	Population	Density
California	423967	38332521	90.413926
Texas	695662	763842356	1098.007877
New York	141297	4895732	34.648520
Florida	170312	32432345	190.429007
Illinois	149995	1244213	8.295030

In [151…
```
data_com['Texas':'Florida']
```

Out[151]:

	Area	Population	Density
Texas	695662	763842356	1098.007877
New York	141297	4895732	34.648520
Florida	170312	32432345	190.429007

In [153…
```
data_com[data_com.Density > 1000]
```

Out[153]:

	Area	Population	Density
Texas	695662	763842356	1098.007877

In [155…
```
df = pd.DataFrame(np.random.randint(0,10,(3,4)),columns = ['A','B','C','D'])
```

In [156…
```
df
```

Out[156]:

	A	B	C	D
0	8	5	2	3
1	8	9	9	0
2	3	4	0	5

In [157…
```
np.exp(df)
```

Out[157]:

	A	B	C	D
0	2980.957987	148.413159	7.389056	20.085537
1	2980.957987	8103.083928	8103.083928	1.000000
2	20.085537	54.598150	1.000000	148.413159

In [161…
```
data = pd.read_csv(r'C:\Users\ajha2\OneDrive\Desktop\POWER BI FILES\Datasets\P6-Eur
```

In [164…
```
pd.isnull(data['Amount']).any() ##checks for any null values in the entire column
```

Out[164]:
```
False
```

In [165…
```
pd.isnull(data['Amount']).sum() ##checks for the total null values in that column
```

Out[165]:
```
0
```

In [172…
```
data.head() #by default first 5 records
```

Out[172]:

	Creditor	Debtor	Amount	Risk
0	Japan	United States	796.0	Stable
1	France	Italy	366.0	High Risk
2	Britain	United States	345.0	Stable
3	Spain	Britain	326.0	Stable
4	Germany	United States	324.0	Stable

In [170…
```
data.head(10) ###first 10 records
```

Out[170]:

	Creditor	Debtor	Amount	Risk
0	Japan	United States	796.0	Stable
1	France	Italy	366.0	High Risk
2	Britain	United States	345.0	Stable
3	Spain	Britain	326.0	Stable
4	Germany	United States	324.0	Stable
5	France	United States	322.0	Stable
6	Germany	Britain	321.0	Stable
7	Spain	United States	163.0	Stable
8	France	Spain	118.0	Medium Risk
9	Italy	Germany	111.0	Stable

In [171…
```
data.tail(8)
## last 8 records
```

Out[171]:

	Creditor	Debtor	Amount	Risk
37	Italy	Ireland	2.83	High Risk
38	Japan	Portugal	2.18	High Risk
39	Japan	Greece	1.37	High Risk
40	Italy	Portugal	0.87	High Risk
41	Spain	Greece	0.78	High Risk
42	Britain	Greece	0.55	High Risk
43	Portugal	United States	0.52	Stable
44	Ireland	Greece	0.34	High Risk

In [174…
```
data.replace(to_replace = 'High Risk',value = 'Higher Risk').head()
```

Out[174]:

	Creditor	Debtor	Amount	Risk
0	Japan	United States	796.0	Stable
1	France	Italy	366.0	Higher Risk
2	Britain	United States	345.0	Stable
3	Spain	Britain	326.0	Stable
4	Germany	United States	324.0	Stable

In [175…
```
ipl_data = { 'Team': ['Riders','Riders','Devils','Devils','Kings','kings','Kings',
                      'Royals','Royals','Riders'],
             'Rank': [1,2,2,3,3,4,1,1,2,4,1,2],
             'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
             'Points': [876,789,863,673,741,812,756,788,694,701,804,690]}
```

In [176…
```
df_ipl = pd.DataFrame(ipl_data)
```

```
df_ipl
```

In [177…

Out[177]:

	Team	Rank	Year	Points
0	Riders	1	2014	876
1	Riders	2	2015	789
2	Devils	2	2014	863
3	Devils	3	2015	673
4	Kings	3	2014	741
5	kings	4	2015	812
6	Kings	1	2016	756
7	Kings	1	2017	788
8	Riders	2	2016	694
9	Royals	4	2014	701
10	Royals	1	2015	804
11	Riders	2	2017	690

In [182…
```
df_ipl.groupby('Team')
```

Out[182]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001CA708D7550>
```

In [183…
```
df_ipl.groupby('Team').groups
```

Out[183]:
```
{'Devils': [2, 3], 'Kings': [4, 6, 7], 'Riders': [0, 1, 8, 11], 'Royals': [9, 10], 'kings': [5]}
```

In [184…
```
df_ipl.groupby(['Team','Year']).groups
```

Out[184]:
```
{('Devils', 2014): [2], ('Devils', 2015): [3], ('Kings', 2014): [4], ('Kings', 2016): [6], ('Kings', 2017): [7], ('Riders', 2014): [0], ('Riders', 2015): [1], ('Riders', 2016): [8], ('Riders', 2017): [11], ('Royals', 2014): [9], ('Royals', 2015): [10], ('kings', 2015): [5]}
```

In [185…
```
grouped = df_ipl.groupby('Year')
```

In [186…
```
for name,group in grouped:
    print(name)
    print(group)
```

```
2014
     Team  Rank  Year  Points
0  Riders     1  2014     876
2  Devils     2  2014     863
4   Kings     3  2014     741
9  Royals     4  2014     701
2015
      Team  Rank  Year  Points
1   Riders     2  2015     789
3   Devils     3  2015     673
5    kings     4  2015     812
10  Royals     1  2015     804
2016
     Team  Rank  Year  Points
6   Kings     1  2016     756
8  Riders     2  2016     694
2017
      Team  Rank  Year  Points
7    Kings     1  2017     788
11  Riders     2  2017     690
```

In [187…
```
print(grouped.get_group(2014))
```

```
     Team  Rank  Year  Points
0  Riders     1  2014     876
2  Devils     2  2014     863
4   Kings     3  2014     741
9  Royals     4  2014     701
```

In [188…
```
df_ipl
```

Out[188]:

	Team	Rank	Year	Points
0	Riders	1	2014	876
1	Riders	2	2015	789
2	Devils	2	2014	863
3	Devils	3	2015	673
4	Kings	3	2014	741
5	kings	4	2015	812
6	Kings	1	2016	756
7	Kings	1	2017	788
8	Riders	2	2016	694
9	Royals	4	2014	701
10	Royals	1	2015	804
11	Riders	2	2017	690

In [191…
```
piv_tab = pd.pivot_table(df_ipl)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [191], in <cell line: 1>()
----> 1 piv_tab = pd.pivot_table(df_ipl)

File ~\anaconda3\lib\site-packages\pandas\core\reshape\pivot.py:95, in pivot_table
(data, values, index, columns, aggfunc, fill_value, margins, dropna, margins_name,
observed, sort)
     92     table = concat(pieces, keys=keys, axis=1)
     93     return table.__finalize__(data, method="pivot_table")
---> 95 table = _internal_pivot_table(
     96     data,
     97     values,
     98     index,
     99     columns,
    100     aggfunc,
    101     fill_value,
    102     margins,
    103     dropna,
    104     margins_name,
    105     observed,
    106     sort,
    107 )
    108 return table.__finalize__(data, method="pivot_table")

File ~\anaconda3\lib\site-packages\pandas\core\reshape\pivot.py:164, in __internal
_pivot_table(data, values, index, columns, aggfunc, fill_value, margins, dropna, m
argins_name, observed, sort)
    161             pass
    162     values = list(values)
--> 164 grouped = data.groupby(keys, observed=observed, sort=sort)
    165 agged = grouped.agg(aggfunc)
    166 if dropna and isinstance(agged, ABCDataFrame) and len(agged.columns):

File ~\anaconda3\lib\site-packages\pandas\core\frame.py:7712, in DataFrame.groupby
(self, by, axis, level, as_index, sort, group_keys, squeeze, observed, dropna)
   7707 axis = self._get_axis_number(axis)
   7709 # https://github.com/python/mypy/issues/7642
   7710 # error: Argument "squeeze" to "DataFrameGroupBy" has incompatible type
   7711 # "Union[bool, NoDefault]"; expected "bool"
-> 7712 return DataFrameGroupBy(
   7713     obj=self,
   7714     keys=by,
   7715     axis=axis,
   7716     level=level,
   7717     as_index=as_index,
   7718     sort=sort,
   7719     group_keys=group_keys,
   7720     squeeze=squeeze,  # type: ignore[arg-type]
   7721     observed=observed,
   7722     dropna=dropna,
   7723 )

File ~\anaconda3\lib\site-packages\pandas\core\groupby\groupby.py:882, in GroupBy.
__init__(self, obj, keys, axis, level, grouper, exclusions, selection, as_index, s
ort, group_keys, squeeze, observed, mutated, dropna)
    879 if grouper is None:
    880     from pandas.core.groupby.grouper import get_grouper
--> 882     grouper, exclusions, obj = get_grouper(
    883         obj,
    884         keys,
    885         axis=axis,
    886         level=level,
    887         sort=sort,
```

```
    888         observed=observed,
    889         mutated=self.mutated,
    890         dropna=self.dropna,
    891     )
    893 self.obj = obj
    894 self.axis = obj._get_axis_number(axis)

File ~\anaconda3\lib\site-packages\pandas\core\groupby\grouper.py:910, in get_grou
per(obj, key, axis, level, sort, observed, mutated, validate, dropna)
    907     groupings.append(ping)
    909 if len(groupings) == 0 and len(obj):
--> 910     raise ValueError("No group keys passed!")
    911 elif len(groupings) == 0:
    912     groupings.append(Grouping(Index([], dtype="int"), np.array([], dtype=n
p.intp)))

ValueError: No group keys passed!
```

In [ ]: