Q-1: Find the nearest element in the array to a given integer.

```
Ex:-
a=23 and array - [10 17 24 31 38 45 52 59].
Nearest element is 24
```

Hint: Read about this function `argmin()`

In [2]:
```
# code here
import numpy as np
a=np.array([1,3,56,78,23,67])
x=int(input('input value'))
index=np.abs(a-x).argmin()
near=a[index]
print(near)
```

```
input value15
23
```

Q-2: Replace multiples of 3 or 5 as 0 in the given array.

```
arr=[1 2 3 4 5 6 7 9]

result-> [1 2 0 4 0 0 7 0]
```

In [3]:
```
import numpy as np
a=np.array([1,2,3,4,5,6,7,9])
a[(a % 3== 0)]=[0]
a[(a% 5==0)]=0
print(a)
```

```
[1 2 0 4 0 0 7 0]
```

In [4]:
```
# code here
import numpy as np
a=np.random.randint(1,100,25).reshape(5,5)
a[(a % 3== 0)]=[0]
a[(a% 5==0)]=0
print(a)
```

```
[[ 0 32 67 89 37]
 [ 0  0  0  0  0]
 [ 0 31  0 56 74]
 [ 0  0  0 34 34]
 [ 0 16 83  0 44]]
```

Q-3: Use Fancy Indexing.

1. Double the array elements at given indexes

```
arr = np.arrange(10)
indexes = [0,3,4,9]
```

Result -> `[ 0  1  2  6  8  5  6  7  8 18]`

2. Using a given array make a different array as in below example

```
array = [1,2,3]
result array -> [1 1 1 2 2 2 3 3 3]
```

- Internal-repetion should be as length of the array.

Hint:

```
if a is an array
a = [2,4]
a[[1,1,0,1]] will result in-> [4 4 2 4]
```

In [5]:
```
# code here
import numpy as np
a=np.arange(10).reshape(1,10)
a[:,[0,3,4,9]]*=2
print(a)
```

```
[[ 0  1  2  6  8  5  6  7  8 18]]
```

In [6]:
```
a=np.array([1,2,3,4])

np.repeat(a,len(a))
```

Out[6]:
```
array([1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4])
```

Q-4: Your are given an array which is havig some nan value. You job is to fill those nan values with most common element in the array.

```
arr=np.array([[1,2,np.nan],[4,2,6],[np.nan,np.nan,5]])
```

In [7]:
```
# code here
from statistics import mode
a=np.array([[1,2,np.nan],[4,2,6],[np.nan,np.nan,5]])
w=mode(a[~np.isnan(a)])
a[(np.isnan(a))]=w
print(a)
```

```
[[1. 2. 2.]
 [4. 2. 6.]
 [2. 2. 5.]]
```

Q-5: Write a NumPy program

- to find the missing data in a given array. Return a boolean matrix.
- also try to fill those missing values with 0. For that, you can use `np.nan_to_num(a)`

```
import numpy as np

np.array([[3, 2, np.nan, 1],
          [10, 12, 10, 9],
          [5, np.nan, 1, np.nan]])
```

In [8]:
```
# code here
a=np.array([[3,2,np.nan,1],[10,12,10,9],[5,np.nan,1,np.nan]])
w=np.isnan(a)
print(w)
a[(np.isnan(a))]=0
print(a)
```

```
[[False False  True False]
 [False False False False]
 [False  True False  True]]
[[ 3.  2.  0.  1.]
 [10. 12. 10.  9.]
 [ 5.  0.  1.  0.]]
```

Q-6: Given two arrays, X and Y, construct the Cauchy matrix C.

`Cij =1/(xi - yj)`

http://en.wikipedia.org/wiki/Cauchy_matrix

```
x = numpy.array([1,2,3,4]).reshape((-1, 1)
y = numpy.array([5,6,7])
```

In [9]:
```
#with loop
x=np.array([1,2,3,4])
y=np.array([5,6,7])
print(x)
print(y)
n=len(x)
m=len(y)
c=np.zeros((n,m))
for i in range(n):
 for j in range(m):
  c[i,j]=1/x[i]-y[j]
print(c)
```

```
[1 2 3 4]
[5 6 7]
[[-4.         -5.         -6.        ]
 [-4.5        -5.5        -6.5       ]
 [-4.66666667 -5.66666667 -6.66666667]
 [-4.75       -5.75       -6.75      ]]
```

In [10]:
```
# code here
#without loop
import numpy as np
x=np.array([1,2,3,4]).reshape(-1,1)
y=np.array([5,6,7])
print(x)
print(y)
z=x-y
print(z)
```

```
c=1/x-y
print(c)
```

```
[[1]
 [2]
 [3]
 [4]]
[5 6 7]
[[-4 -5 -6]
 [-3 -4 -5]
 [-2 -3 -4]
 [-1 -2 -3]]
[[-4.         -5.         -6.        ]
 [-4.5        -5.5        -6.5       ]
 [-4.66666667 -5.66666667 -6.66666667]
 [-4.75       -5.75       -6.75      ]]
```

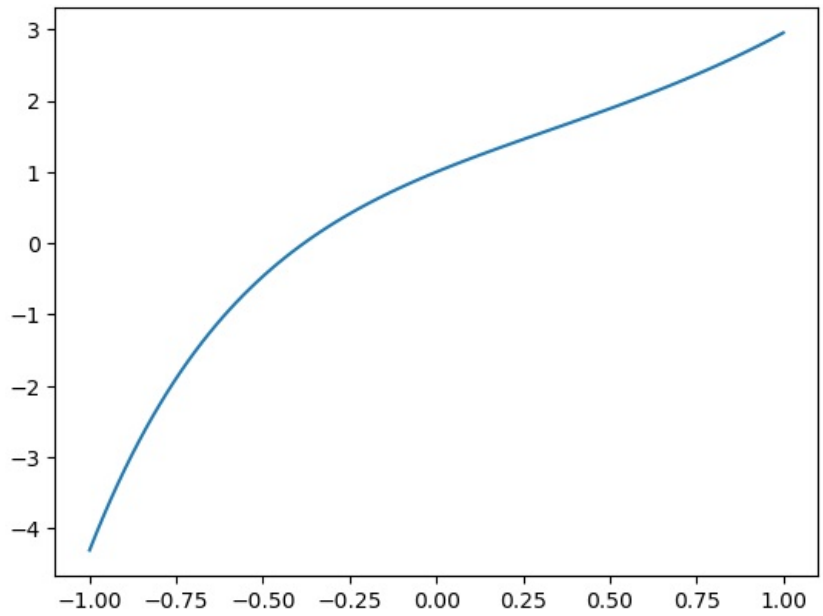
Q-7: Plot the following equation.

$$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Note: This equation is called tanh activation function. In deep learning, many times this function is used. If you find some difference between the sigmoid function and this tanh function, note that to your notebook.

In [11]:
```
# code here
import matplotlib.pyplot as plt
x=np.linspace(-1,1,100)
y=np.exp(x)-np.exp(-x)/np.exp(x)+np.exp(-x)
plt.plot(x,y)
```

Out[11]:
```
[<matplotlib.lines.Line2D at 0x16ec12d8310>]
```



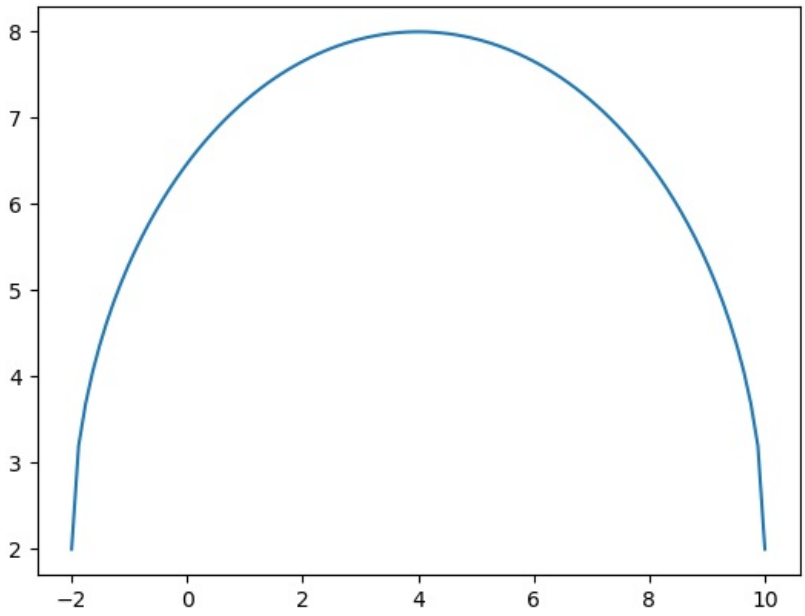
Q-8: Plot the below equation.

$$y = \sqrt{36 - (x-4)^2} + 2$$

The range of x should be between -2 to 10. $x \in [-2, 10]$

In [12]:
```
# code here
x=np.linspace(-2,10,100)
y=np.sqrt(36-(x-4)**2)+2
plt.plot(x,y)
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x16ec12fa490>]
```



Q-9: Write a program implement Broadcasting Rule to check if two array can be added or not.

Given tuples of shapes.

```
shape of a- (3,2,2)
shape of b- (2,2)

check_broadcast(a, b) -> return Boolean (True if can broadcasted, False other wise.)
```

In [13]:
```
# code here
```

Q-10: Create a random 3x4 matrix with value between 0-100. And perform below tasks

```
i. Sort this matrix. np.sort()
ii. Sort this matrix based on values in 2nd column.
iii. Sort this matrix based on max value in each row.
iv. Sort based on elements value.
```

See examples:

```
arr =
    [[92 90 74]
    [ 6 63 93]
    [15 93 96]
    [70 60 48]]

i. np.sort
[[74 90 92]
 [ 6 63 93]
 [15 93 96]
 [48 60 70]]

ii. based on 2nd column
[[70 60 48]
 [ 6 63 93]
 [92 90 74]
 [15 93 96]]

iii. based on row max- ascending
[[15 93 96]
 [ 6 63 93]
```

```
 [92 90 74]
 [70 60 48]]

iv. based on elements value
[[ 6 15 48]
 [60 63 70]
 [74 90 92]
 [93 93 96]]
```

In [14]:
```
# code here
a=np.random.randint(0,101, size=(3,4))
print(a)
np.sort(a)
```

Out[14]:
```
[[ 98  88  19 100]
 [ 21  57  39  19]
 [ 89  52  58  24]]
array([[ 19,  88,  98, 100],
       [ 19,  21,  39,  57],
       [ 24,  52,  58,  89]])
```

In [15]:
```
a[np.lexsort((a[:,1],))]
```

Out[15]:
```
array([[ 89,  52,  58,  24],
       [ 21,  57,  39,  19],
       [ 98,  88,  19, 100]])
```

In [16]:
```
b=np.max(a,axis=1)
a[np.lexsort((b,))]
```

Out[16]:
```
array([[ 21,  57,  39,  19],
       [ 89,  52,  58,  24],
       [ 98,  88,  19, 100]])
```

In [17]:
```
b=a.flatten()
print(b)
np.sort(b).reshape(3,4)
```

Out[17]:
```
[ 98  88  19 100  21  57  39  19  89  52  58  24]
array([[ 19,  19,  21,  24],
       [ 39,  52,  57,  58],
       [ 88,  89,  98, 100]])
```

`Q-11:` There is an array of marks of 5 students in 4 subjects. Further you are asked to perform below task.

```
i. Add marks every student of an extra subject in the same array.
ii. Add two new students marks in respective 5 subjects.(one subject added in above task)
iii. Add extra column with sum of all subjects(5-subjects) marks
iv. Sort the array(non-ascending order) on total marks column--one added in above task. Show
top 2 rows.
```

Note: Change dimension of arrays during concatenation or appending if required.

Given Array-

```
marks = [[13, 10,  9, 33],
       [63, 46, 90, 42],
       [39, 76, 13, 29],
       [82,  9, 29, 78],
       [67, 61, 59, 36]]

extra_subject = [41, 87, 72, 36, 92]
#Two extra students record-
rec1 = [77, 83, 98, 95, 89]
rec2 = [92, 71, 52, 61, 53]
```

In [18]:
```
# code here

a= np.array([[80, 75, 85, 90],
                 [65, 70, 75, 80],
                 [90, 85, 95, 100],
                 [70, 60, 65, 75],
                 [85, 90, 95, 85]])

print(a)
b = np.array([85, 90, 75, 80,95])
c= np.column_stack((a,b))
print(c)
```

```
[[ 80  75  85  90]
 [ 65  70  75  80]
 [ 90  85  95 100]
 [ 70  60  65  75]
 [ 85  90  95  85]]
[[ 80  75  85  90  85]
 [ 65  70  75  80  90]
 [ 90  85  95 100  75]
 [ 70  60  65  75  80]
 [ 85  90  95  85  95]]
```

In [19]:
```
d = np.array([85, 90, 75, 80,95])
e = np.array([45,34,26,90,70])
arra=np.row_stack((c,d,e))
print(arra)
```

```
[[ 80  75  85  90  85]
 [ 65  70  75  80  90]
 [ 90  85  95 100  75]
 [ 70  60  65  75  80]
 [ 85  90  95  85  95]
 [ 85  90  75  80  95]
 [ 45  34  26  90  70]]
```

In [20]:
```
sum_arra=np.sum(arra,axis=1)
print(sum_arra)
final=np.column_stack((arra,sum_arra))
print(final)
```

```
[415 380 445 350 450 425 265]
[[ 80  75  85  90  85 415]
 [ 65  70  75  80  90 380]
 [ 90  85  95 100  75 445]
 [ 70  60  65  75  80 350]
 [ 85  90  95  85  95 450]
 [ 85  90  75  80  95 425]
 [ 45  34  26  90  70 265]]
```

In [21]:
```
sorting=final[np.lexsort((final[:,-1],))]
print(sorting)
sorting[:2,:]
```

```
[[ 45  34  26  90  70 265]
 [ 70  60  65  75  80 350]
 [ 65  70  75  80  90 380]
 [ 80  75  85  90  85 415]
 [ 85  90  75  80  95 425]
 [ 90  85  95 100  75 445]
 [ 85  90  95  85  95 450]]
```

Out[21]:
```
array([[ 45,  34,  26,  90,  70, 265],
       [ 70,  60,  65,  75,  80, 350]])
```

Q-12: Find unique arrays from a 2D array column wise and row wise.

```
arr = np.array([[1,2,3,3,1,1],
                [0,9,1,2,8,8],
                [1,2,3,8,8,8],
                [1,2,3,3,1,1]])
```

Expected Result-

```
Row Wise
[[0 9 1 2 8 8]
 [1 2 3 3 1 1]
 [1 2 3 8 8 8]]

Col Wise
[[1 1 2 3 3]
 [0 8 9 1 2]
 [1 8 2 3 8]
 [1 1 2 3 3]]
```

In [22]:
```
# code here
a=np.array([[1,2,3,3,1,1],
               [0,9,1,2,8,8],
                [1,2,3,8,8,8],
                [1,2,3,3,1,1]])
np.unique(a,axis=0)
```

Out[22]:
```
array([[0, 9, 1, 2, 8, 8],
       [1, 2, 3, 3, 1, 1],
       [1, 2, 3, 8, 8, 8]])
```

In [23]:
```
np.unique(a,axis=1)
```

Out[23]:
```
array([[1, 1, 2, 3, 3],
       [0, 8, 9, 1, 2],
       [1, 8, 2, 3, 8],
       [1, 1, 2, 3, 3]])
```

Q-13: Flip given 2-D array along both axes at the same time.

In [24]:
```
# code here
np.flip(a,axis=(0,1))
```

Out[24]:
```
array([[1, 1, 3, 3, 2, 1],
       [8, 8, 8, 3, 2, 1],
       [8, 8, 2, 1, 9, 0],
       [1, 1, 3, 3, 2, 1]])
```

Q-14: Get row numbers of NumPy array having element larger than X.

```
arr = [[1,2,3,4,5],
       [10,-3,30,4,5],
       [3,2,5,-4,5],
       [9,7,3,6,5]]

X = 6
```

In [25]:
```
# code here
import numpy as np
arr = np.array([[1, 2, 3,4,5],
                [10,-3,30,4,5],
                [3,2,5,-4,5],
               [9,7,3,6,5]])

X = 6
row_numbers = np.where(np.any(arr > X, axis=1))

print(row_numbers)
```

```
(array([1, 3], dtype=int64),)
```

Q-15: How to convert an array of arrays into a flat 1d array?

In [26]:
```
# These arrays are given.
arr1 = np.arange(3)
arr2 = np.arange(3,7)
arr3 = np.arange(7,10)
```

In [27]:
```
# code here
arr1.flatten()
```

Out[27]:
```
array([0, 1, 2])
```

In [28]:
```
arr2.flatten()
```

Out[28]:
```
array([3, 4, 5, 6])
```

In [29]:
```
arr3.flatten()
```

Out[29]:
```
array([7, 8, 9])
```

Q-16: You are given a array. You have to find the minimum and maximum array element and remove that from the array.

```python import numpy as np

np.random.seed(400) arr = np.random.randint(100, 1000, 200).reshape((1, 200))

In [30]:
```
code here

np.random.seed(400)
a= np.random.randint(100, 1000, 200).reshape((1, 200))
min_val = np.min(a)
max_val = np.max(a)
b = np.delete(a, np.where((a == min_val) | (a == max_val)))
print(b)
```
```

```
[563 418 240 507 362 345 236 719 291 298 639 458 387 262 613 267 882 181
 425 790 635 889 818 872 967 277 470 336 920 917 295 557 830 506 385 353
 975 592 997 137 340 222 215 472 459 617 649 935 956 914 932 645 952 921
 490 527 972 278 307 840 958 246 449 251 957 627 920 824 356 825 173 323
 372 960 710 464 244 782 763 635 436 774 171 469 178 458 624 211 771 270
 308 231 952 514 699 702 433 900 373 318 265 503 320 230 324 922 967 620
 743 527 117 566 804 123 946 587 227 853 757 944 328 855 930 325 729 426
 514 296 879 575 936 705 209 191 743 510 513 628 559 658 528 395 525 922
 136 496 225 895 975 263 908 420 711 800 976 786 235 930 859 618 226 695
 460 218 483 490 803 621 453 193 607 677 637 728 724 534 748 291 194 761
 875 687 569 228 482 781 554 654 739 885 197 266 228 892 207 883 588]
```

`Q-17:` You are given an arrays. You have to limit this array's elements between 100 to 700. $arr \in [100, 700]$. So replace those values accordingly with the minimum and maximum value. Then sort the array and perform the cumulative sum of that array.

In [31]:
```
# code here
arr = np.array([50, 200, 800, 300, 600, 900, 100])
arr_clipped = np.clip(arr, 100, 700)
arr_clipped[arr < 100] = 100
arr_clipped[arr > 700] = 700
arr_sorted = np.sort(arr_clipped)
cumulative_sum = np.cumsum(arr_sorted)
print(arr_clipped)
print(arr_sorted)
print(cumulative_sum)
```

```
[100 200 700 300 600 700 100]
[100 100 200 300 600 700 700]
[ 100  200  400  700 1300 2000 2700]
```

`Q-18:` You are given a array ($arr \in [0, 1]$). First you have round off the elements upto 3 decimal places and compare that

- 0th percentile == minimum value of the array
- 100th percentile == maximum value of the array
- also find the difference betwen 51th percenile and 50th percentile values

In [32]:
```
# code here
arr = np.random.rand(1000)
arr_rounded = np.round(arr,3)

percentile_0 = np.percentile(arr_rounded, 0)
percentile_100 = np.percentile(arr_rounded, 100)
percentile_50 = np.percentile(arr_rounded, 50)
percentile_51 = np.percentile(arr_rounded, 51)
difference_51_50 = percentile_51 - percentile_50

print(percentile_0)
print( percentile_100)
print( difference_51_50)
```

```
0.0
0.998
0.006470000000000031
```

In [ ]:
```

```