```
import numpy as np
```

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()`

```
# code here
arr = np.array([10, 17, 24, 31, 38, 45, 52, 59])
a=39
arr[abs(arr - a).argmin()]
```

```
38
```

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]
```

```
# code here
arr=np.array([1, 2, 3, 4, 5, 6, 7, 9])
arr[(arr%3==0) | (arr%5==0)] = 0
arr
```

```
array([1, 2, 0, 4, 0, 0, 7, 0])
```

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]
```

```
# code here
#1
arr = np.arange(10)
indexes = [0,3,4,9]
arr[indexes] *=2
arr
```

```
array([ 0,  1,  2,  6,  8,  5,  6,  7,  8, 18])
```

```
array = np.array([1,2,3])
indexes = []
for index in range(len(array)):
    indexes.extend([index] * len(array))
array[indexes]
```

```
array([1, 1, 1, 2, 2, 2, 3, 3, 3])
```

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]])
```

```
# code here
arr=np.array([[1,2,np.nan],[4,2,6],[np.nan,np.nan,5]])
d = {}
for i in arr.flatten():
    if i in d:
        d[i] += 1
    else:
        d[i] = 1

arr[np.isnan(arr)] = sorted(d.items(), key = lambda x: x[1])[-1][0]
arr
```

```
array([[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]])
```

```
# code here
#1
arr = np.array([[3, 2, np.nan, 1],
          [10, 12, 10, 9],
          [5, np.nan, 1, np.nan]])
np.isnan(arr)
```

```
array([[False, False,  True, False],
       [False, False, False, False],
       [False,  True, False,  True]])
```

```
#2
np.nan_to_num(arr)
```

```
array([[ 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])
```

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

1/(x-y)
```

```
array([[-0.25      , -0.2       , -0.16666667],
       [-0.33333333, -0.25      , -0.2       ],
       [-0.5       , -0.33333333, -0.25      ],
       [-1.        , -0.5       , -0.33333333]])
```

Q-7: Plot this below equation.

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

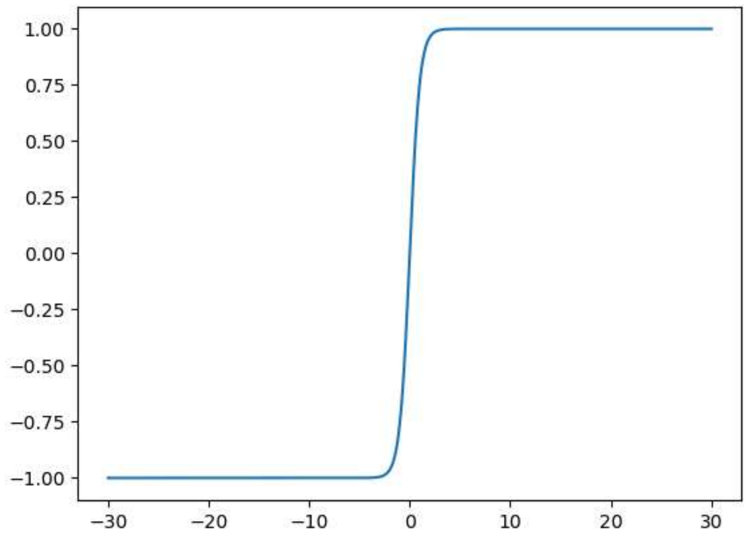
Note: This equation is called tanh activation function. In deep learning, many times this function is used. If you find some difference between the sigmoid function and this tanh function, note that to your notebook.

```
# code here
import matplotlib.pyplot as plt

x = np.linspace(-30, 30, 1000)
y = (np.exp(x) - np.exp(-x)) / (np.exp(x) + np.exp(-x))

plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x2b9364fcd30>]
```



Q-8: Plot the below equation.

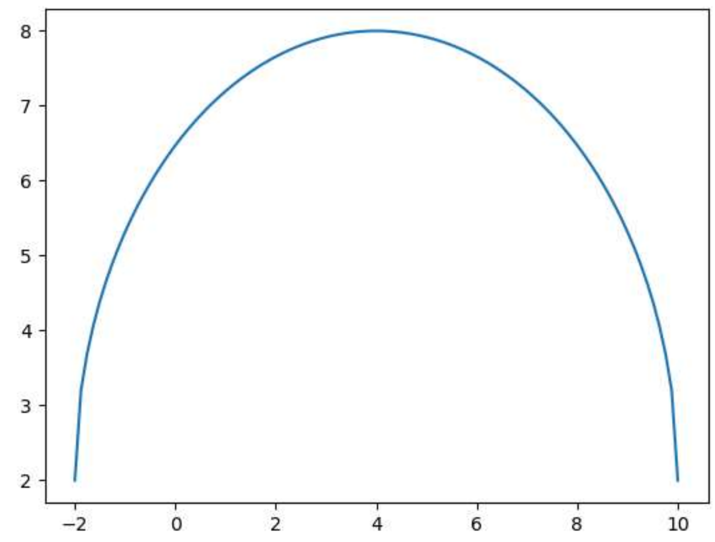
$$y = \sqrt{36 - (x-4)^2} + 2$$

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

```
# code here
x = np.linspace(-2, 10, 100)
y = np.sqrt(36 - (x-4)**2) + 2

plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x2b936bfd700>]
```



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.)
```

```
# code here

# 4 3 3 5
#     2 5

# Align both shapes
# Loop for both shapes from behind
#     If two numbers are not equal and none of the numbers are 1, return false
# return true

def check_broadcast(a,b):
    a = a[::-1]
    b = b[::-1]
    for ai, bi in zip(a,b):
        if ai != bi and a1 != 1 and b1 != 1:
            return False
    return
```