Note: In Data Science googling is a very important skill. If you find some difficulties to solve the problem, google it and try to find some clues to solve.

```
import numpy as np
```

Q–1 Create a null vector of size 10 but the fifth value which is 1.

```
#Code here

# Filling the numpy array with NaN values
a = np.nan * np.empty(10)
a[4] = 1
a
```

```
array([nan, nan, nan, nan,  1., nan, nan, nan, nan, nan])
```

Q–2 Ask user to input two numbers a, b. Write a program to generate a random array of shape (a, b) and print the array and avg of the array.

```
#Code here
a,b = map(int, input("Enter two numbers: ").split())
x = np.random.random((a,b))
print(x)
print(x.mean())
```

```
Enter two numbers:  2 5
[[0.0277054  0.57613354 0.48856158 0.464767   0.007949  ]
 [0.80633051 0.46925329 0.42072919 0.49346575 0.67178371]]
0.44266789729058403
```

Q–3 Write a function to create a 2d array with 1 on the border and 0 inside. Take 2-D array shape as (a,b) as parameter to function.

Eg.-

```
[[1,1,1,1],
[1,0,0,1],
[1,0,0,1],
[1,1,1,1]]
```

```
#Code here
def borderArray(a,b):
    x = np.ones((a,b))
    x[1:-1, 1:-1] = 0    # Except the border, changing all other elements to 0
    return x
borderArray(4,5)
```

```
array([[1., 1., 1., 1., 1.],
       [1., 0., 0., 0., 1.],
       [1., 0., 0., 0., 1.],
       [1., 1., 1., 1., 1.]])
```

Q–4 Create a vector of size 10 with values ranging from 0 to 1, both excluded.

```
#Code here
np.linspace(0, 1, 12)[1:-1]
```

```
array([0.09090909, 0.18181818, 0.27272727, 0.36363636, 0.45454545,
       0.54545455, 0.63636364, 0.72727273, 0.81818182, 0.90909091])
```

Q–5 Can you create a identity mattrix of shape (3,4). If yes write code for it.

```
#Code here
# Identity matricx is a square mattrix, means no of rows and columns will always be same.
# Here shape given 3,4 so identuty matrix not possible.

#Identy matrix of size 3-- 3X3
np.eye(3)

#Identy matrix of size 4-- 4X4
np.identity(4)

# eye and identity are same.
```

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

Q-6: Create a 5x5 matrix with row values ranging from 0 to 4.

```
#Code here
z = np.zeros((5,5))
z+= np.arange(5)
z
```

```
array([[0., 1., 2., 3., 4.],
       [0., 1., 2., 3., 4.],
       [0., 1., 2., 3., 4.],
       [0., 1., 2., 3., 4.],
       [0., 1., 2., 3., 4.]])
```

Q-7: Consider a random integer (in range 1 to 100) vector with shape (10,2) representing coordinates, and coordinates of a point as array is given. Create an array of distance of each point in the random vectros from the given point. Distance array should be interger type.

```
point = np.array([2,3])
```

```
#code here
a = np.random.randint(1, 100, (10, 2))
p = np.array([2,3])
np.sqrt(np.sum((a-p)**2, axis=1)).astype(int)
```

```
array([74, 76, 38, 52, 63, 68, 21, 94, 91, 63])
```

Q-8: Consider a (6,7,8) shape array, what is the index (x,y,z) of the 100th element?

```
#Code here
np.unravel_index(100, (6,7,8))
```

```
(1, 5, 4)
```

Q-9: Arrays

You are given a space separated list of numbers. Your task is to print a reversed NumPy array with the element type float.

Input Format:

A single line of input containing space separated numbers.

Output Format:

Print the reverse NumPy array with type float.

Example 1:

Input:

```
1 2 3 4 -8 -10
```

Output:

```
 [-10.  -8.   4.   3.   2.   1.]
```

```
#Code here
a = input().strip().split()
np.array(a[::-1], dtype=np.float32)
```

```
 2 5 6
array([6., 5., 2.], dtype=float32)
```

Q-10: Elements count

Count the number of elements of a numpy array.

Example 1:

Input:

```
 np.array([])
```

Output:

```
 elements_count :  0
```

Example 2:

Input:

```
 np.array([1, 2])
```

Output:

```
 elements_count :  2
```

```
#Code here
a = np.array([])
a = np.array([1, 2])
a = np.zeros((2,3))
a.size
```

```
6
```

Q-11: Softmax function

Create a Python function to calculate the Softmax of the given numpy 1D array. The function only accepts the numpy 1D array, otherwise raise error.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=i}^{K} e^{z_j}}$$

https://en.wikipedia.org/wiki/Softmax_function

Example 1:

Input:

```
 [86.03331084 37.7285648  48.64908087 87.16563062 38.40852563 37.20006318]
```

Output:

```
 [2.43733249e-01, 2.56112115e-22, 1.41628284e-17, 7.56266751e-01,
        5.05514197e-22, 1.50974911e-22]
```

Example 2:

Input:

```
 [33.17344305 45.61961654 82.05405781 80.9647098  68.82830233 91.52064278]
```

Output:

```
 [4.57181035e-26, 1.16249923e-20, 7.73872596e-05, 2.60358426e-05,
       1.39571531e-10, 9.99896577e-01]
```

```
#Code here
def softmax(arr):
    if type(arr) != np.ndarray:
        raise TypeError("Requires Numpy Array")
    elif arr.ndim > 1:
        raise TypeError("Requires 1D Array")
    s = np.sum(np.exp(arr))
    return np.exp(arr)/s
softmax(np.array([86.03331084, 37.7285648,  48.64908087, 87.16563062, 38.40852563, 37.20006318]))
```

```
 array([2.43733248e-01, 2.56112114e-22, 1.41628283e-17, 7.56266752e-01,
        5.05514197e-22, 1.50974911e-22])
```

Q-12: Vertical stack

Write a python function that accepts infinite number of numpy arrays and do the vertical stack to them. Then return that new array as result. The function only accepts the numpy array, otherwise raise error.

Example 1:

Input:

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

b= [[1 1 1 1 1]
 [1 1 1 1 1]]
```

Output:

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

Example 2:

Input:

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

b= [[1 1 1 1 1]
 [1 1 1 1 1]]

c= [[0.10117373 0.1677244  0.73764059 0.83166097 0.48985695]
 [0.44581567 0.13502419 0.55692335 0.16479622 0.61193593]]
```

Output:

```
[[0.         1.         2.         3.         4.        ]
 [5.         6.         7.         8.         9.        ]
 [1.         1.         1.         1.         1.        ]
 [1.         1.         1.         1.         1.        ]
 [0.10117373 0.1677244  0.73764059 0.83166097 0.48985695]
 [0.44581567 0.13502419 0.55692335 0.16479622 0.61193593]]
```

```
#Coder here
def vertical_stack(*args):
    for i in args:
        if type(i) != np.ndarray:
            raise TypeError("Requires Numpy Array")
    return np.vstack(args)

a = np.arange(10).reshape(2, -1)
print("a=",a)
b = np.repeat(1, 10).reshape(2, -1)
print("b=",b)
print(vertical_stack(a,b))
c = np.random.random((2,5))
print("c=", c)
vertical_stack(a,b,c)
```

```
a= [[0 1 2 3 4]
 [5 6 7 8 9]]
b= [[1 1 1 1 1]
 [1 1 1 1 1]]
[[0 1 2 3 4]
 [5 6 7 8 9]
 [1 1 1 1 1]
 [1 1 1 1 1]]
c= [[0.21639132 0.32821234 0.75890757 0.25967328 0.53357754]
 [0.42222234 0.05588129 0.74444709 0.81171534 0.11995402]]
array([[0.        , 1.        , 2.        , 3.        , 4.        ],
       [5.        , 6.        , 7.        , 8.        , 9.        ],
       [1.        , 1.        , 1.        , 1.        , 1.        ],
       [1.        , 1.        , 1.        , 1.        , 1.        ],
       [0.21639132, 0.32821234, 0.75890757, 0.25967328, 0.53357754],
       [0.42222234, 0.05588129, 0.74444709, 0.81171534, 0.11995402]])
```

Q-13: Dates

Create a python function named **date_array** that accepts two dates as string format and returns a numpy array of dates between those 2 dates. The function only accept 2 strings, otherwise raise error. The date format should be like this only: `2022-12-6` . The end date should be included and for simplicity, choose dates from a same year.

Example 1:

Input:

```
date_array(start = '2020-09-15', end = '2020-09-25')
```

Output:

```
['2020-09-15', '2020-09-16', '2020-09-17', '2020-09-18',
 '2020-09-19', '2020-09-20', '2020-09-21', '2020-09-22',
 '2020-09-23', '2020-09-24', '2020-09-25']
```

Example 2:

Input:

```
date_array(start = '2022-12-01', end = '2022-12-06')
```

Output:

```
['2022-12-01', '2022-12-02', '2022-12-03', '2022-12-04', '2022-12-05', '2022-12-06']
```

Example 3:

Input:

```
date_array(start = '2020-11-25', end = '2020-11-30')
```

Output:

```
['2020-11-25', '2020-11-26', '2020-11-27', '2020-11-28',
 '2020-11-29', '2020-11-30']
```

```
#Code here

def date_array(start: str, end: str):
    if type(start) != str or type(end) != str:
        raise TypeError

    total_days_of_month = {"01": 31, "02": 28, "03": 31, "04":30, "05": 31, "06":30, "07":31, "08":31, "09":30, "10": 31, "1

    end = end.split("-")
    end_last = int(end[-1]) + 1

    # If the next day of end falls in the next month, account for that
    if total_days_of_month[end[-2]] < end_last:
        days_diff = end_last - total_days_of_month[end[-2]]
        end[-1] = f'0{days_diff}' if days_diff< 10 else f'{days_diff}'
        next_month = int(end[-2]) + 1
        end[-2] = f'0{next_month}' if next_month< 10 else f'{next_month}'
    else:
        end[-1] = f'0{end_last}' if end_last < 10 else f'{end_last}'
    end = "-".join(end)
    return np.arange(start, end, dtype="datetime64[D]")    # Use arange() to generate all dates between start and end
date_array(start = '2020-11-25', end = '2020-11-30')
```

```
array(['2020-11-25', '2020-11-26', '2020-11-27', '2020-11-28',
       '2020-11-29', '2020-11-30'], dtype='datetime64[D]')
```

Q-14: Subtract the mean of each row from a matrix.

```
#code here
x = np.random.random((5,4))
x - x.mean(axis = 1, keepdims = True)
```

```
array([[-0.06845162,  0.15698291, -0.07289683, -0.01563446],
       [ 0.01134548,  0.11410879,  0.19929023, -0.32474449],
       [-0.26055217,  0.40783042,  0.07308281, -0.22036106],
       [ 0.10630007, -0.27265075,  0.37913096, -0.21278028],
       [ 0.23016348, -0.03754832,  0.05033272, -0.24294787]])
```

Q-15: Swap column-1 of array with column-2 in the array.

```
#Code here
a = np.arange(9).reshape(3,3)
a[:, [0, 2, 1]]
```

```
array([[0, 2, 1],
       [3, 5, 4],
       [6, 8, 7]])
```

Q-16: Replace odd elements in arrays with -1.

```
#Code here
a = np.arange(10)
a[a%2 == 1] = -1
a
```

```
array([ 0, -1,  2, -1,  4, -1,  6, -1,  8, -1])
```

Q-17: Given two arrays of same shape make an array of max out of two arrays. (Numpy way)

```
a=np.array([6,3,1,5,8])
b=np.array([3,2,1,7,2])
```

```
Result-> [6 3 1 7 8]
```

```
#Code here
a=np.array([6,3,1,5,8])
b=np.array([3,2,1,7,2])

a[b>a] = b[a<b]
a
```

```
array([6, 3, 1, 7, 8])
```

Q-18 Answer below asked questions on given array:

1. Fetch Every alternate column of the array
2. Normalise the given array

https://en.wikipedia.org/wiki/Normalization_(statistics)

There are different form of normalisation for this question use below formula.

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

```
arr1=np.random.randint(low=1, high=10000, size=40).reshape(8,5)
```

```
# Given
arr1=np.random.randint(low=1, high=10000, size=40).reshape(8,5)
arr1
```

```
array([[5004, 6405, 1638, 2845, 3510],
       [6966, 3244, 2711, 4785, 7351],
       [ 689, 8978,  572, 3661,  245],
       [5895, 4114, 4716, 2627, 4822],
       [ 994, 2038, 5624, 5577, 2819],
       [9362, 7754, 7941, 4262, 2240],
       [5707, 5418, 9980,  255, 4026],
       [8057, 3557, 8667, 7068, 8319]])
```

```
#Code here
#1
arr1[:, ::2]
```

```
array([[5004, 1638, 3510],
       [6966, 2711, 7351],
       [ 689,  572,  245],
       [5895, 4716, 4822],
       [ 994, 5624, 2819],
       [9362, 7941, 2240],
       [5707, 9980, 4026],
       [8057, 8667, 8319]])
```

```
(arr1 - arr1.min())/(arr1.max() - arr1.min())
```

```
array([[0.48885465, 0.63276836, 0.14309194, 0.26707756, 0.33538778],
       [0.69039548, 0.30806369, 0.25331279, 0.4663585 , 0.7299435 ],
       [0.04560863, 0.89707242, 0.03359014, 0.35089882, 0.        ],
       [0.58038007, 0.39743195, 0.45927067, 0.24468413, 0.47015922],
       [0.07693888, 0.18418079, 0.55254237, 0.54771443, 0.26440678],
       [0.93651772, 0.77134052, 0.79054956, 0.41263482, 0.20493066],
       [0.56106831, 0.53138161, 1.        , 0.00102722, 0.3883924 ],
       [0.80246533, 0.34021572, 0.86512583, 0.70087314, 0.82937853]])
```

Q-19: Write a function which will accept 2 arguments.

First: A 1D numpy array arr

Second: An integer n {Please make sure n<=len(arr)}

Output: The output should be the nth largest item out of the array

```
# Example1 : arr=(12,34,40,7,1,0) and n=3, the output should be 12
# Example2 : arr=(12,34,40,7,1,0) and n=1, the output should be 40
```

```
#Code here
def nthmax(arr, n):
    if n>len(arr):
        raise IndexError("n is way out of limit")
    arr.sort()
    return arr[-n]
nthmax(np.array([12,34,40,7,1,0]),2)
```

```
34
```

Q-20: Create the following pattern without hardcoding. Use only numpy functions and the below input array a.

```
# Input: a = np.array([1,2,3])
# Output: array([1, 1, 1, 2, 2, 2, 3, 3, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3])
```

```
#code here
a = np.array([1,2,3])
np.hstack([np.repeat(a, 3), np.tile(a, 3)])
```

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