	Marwadi University **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Practical based on NumPy ndarray	
Experiment No: 07	**Date:**	**Enrollment No:92400133189**

Aim: Practical based on NumPy ndarray

IDE:

NumPy is a Python package created in 2005 that performs numerical calculations. It is generally used for working with arrays. NumPy also includes a wide range of mathematical functions, such as linear algebra, Fourier transforms, and random number generation, which can be applied to arrays.

Import NumPy in Python

We can import NumPy in Python using the import statement.

```
import numpy as np
```

The code above imports the numpy package in our program as an alias np. After this import statement, we can use NumPy functions and objects by calling them with np.

NumPy Array Creation

An array allows us to store a collection of multiple values in a single data structure. The NumPy array is similar to a list, but with added benefits such as being faster and more memory efficient. There are multiple techniques to generate arrays in NumPy.

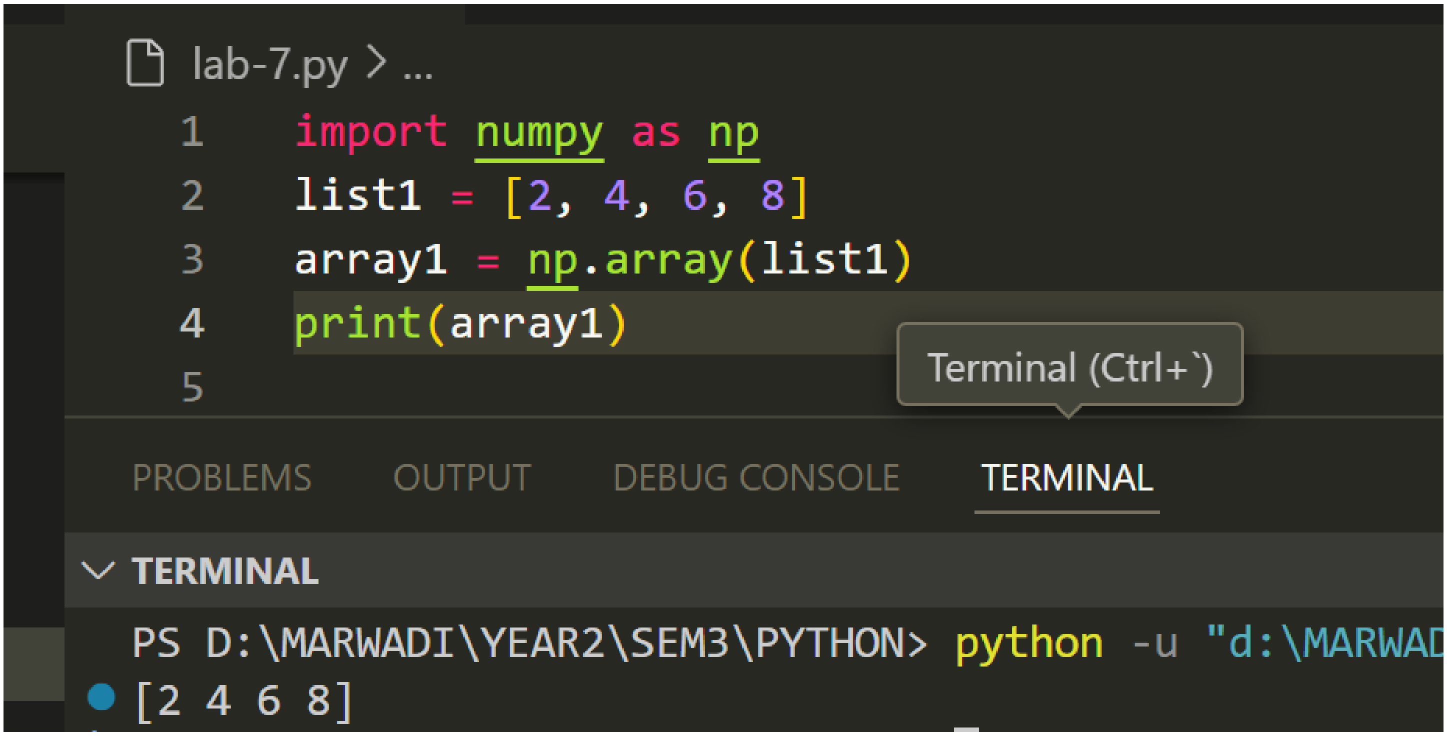
Create Array Using Python List

We can create a NumPy array using a Python List. For example,

Example

```
import numpy as np
list1 = [2, 4, 6, 8]
array1 = np.array(list1)
print(array1)
```

Output



```
import numpy as np
list1 = [2, 4, 6, 8]
array1 = np.array(list1)
print(array1)
```

```
PS D:\MARWADI\YEAR2\SEM3\PYTHON> python -u "d:\MARWAD
[2 4 6 8]
```

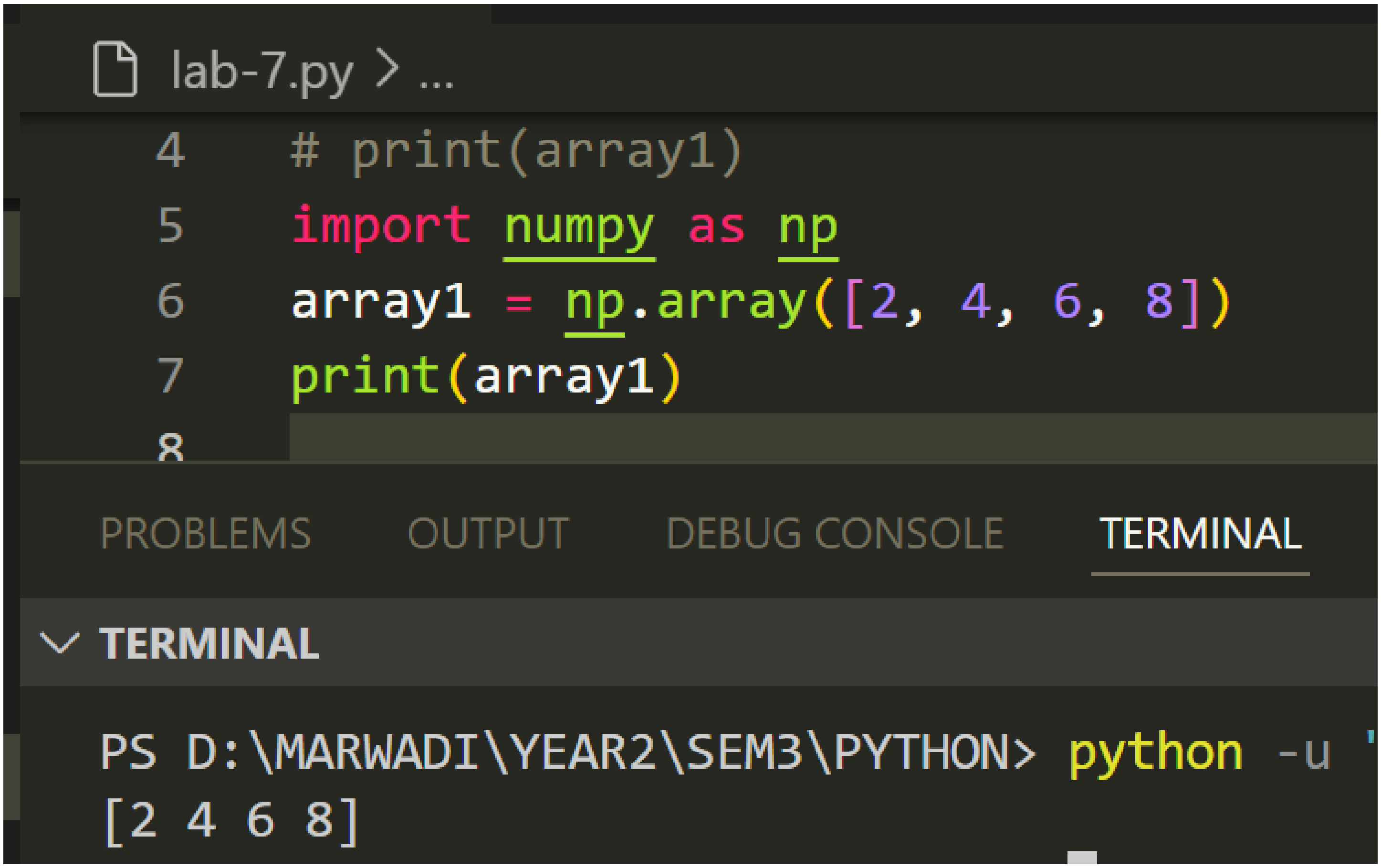
Example

import numpy as np

array1 = np.array([2, 4, 6, 8])

print(array1)

Output



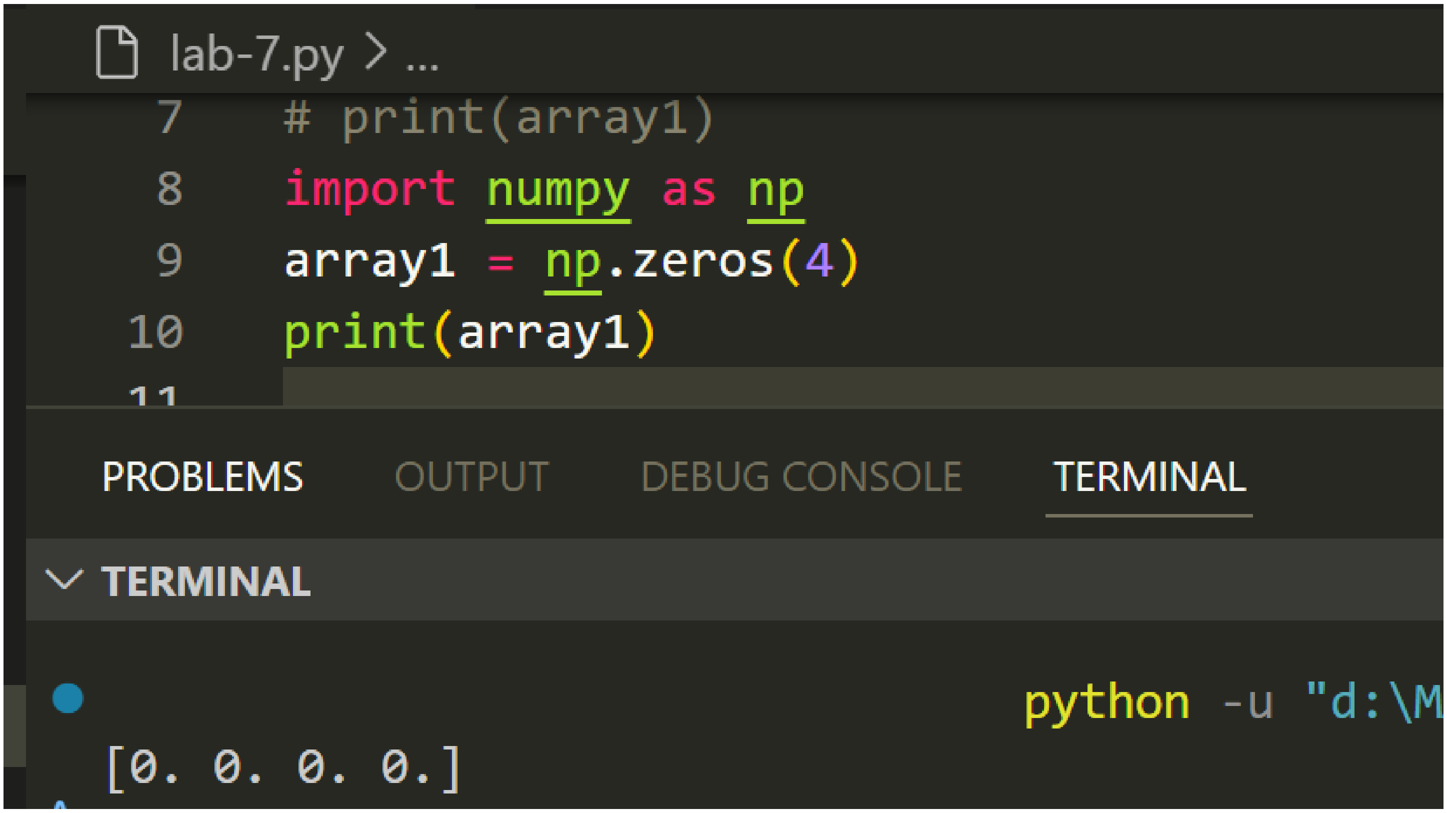
```
# print(array1)
import numpy as np
array1 = np.array([2, 4, 6, 8])
print(array1)
```

```
PS D:\MARWADI\YEAR2\SEM3\PYTHON> python -u '
[2 4 6 8]
```

Create an Array Using np.zeros()

The np.zeros() function allows us to create an array filled with all zeros. For example,

Example
import numpy as np
array1 = np.zeros(4)
print(array1)
Output



```
# print(array1)
import numpy as np
array1 = np.zeros(4)
print(array1)
```

```
python -u "d:\M
[0. 0. 0. 0.]
```

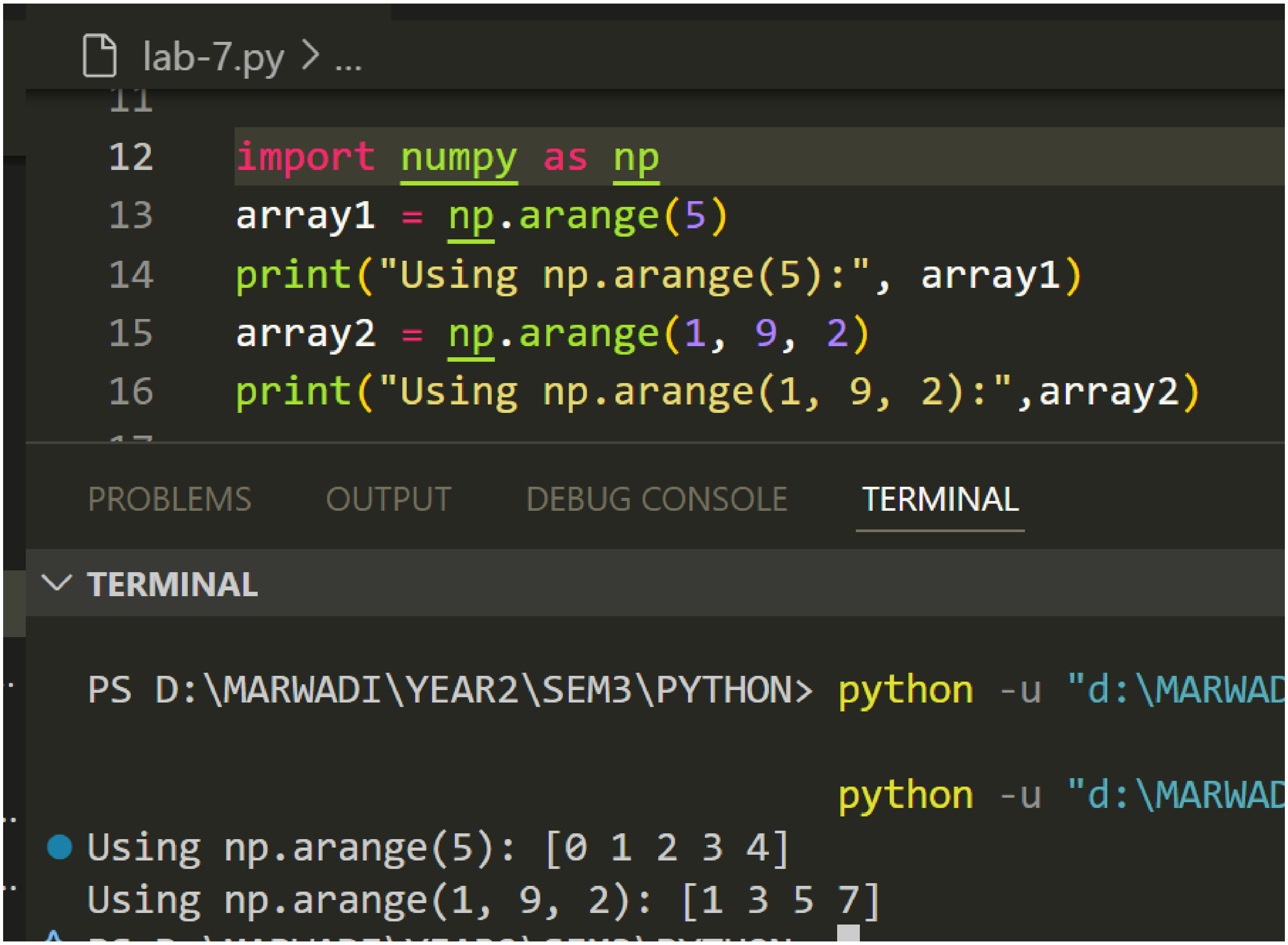
Create an Array With np.arange()

The np.arange() function returns an array with values within a specified interval. For example,

Example

```
import numpy as np
# create an array with values from 0 to 4
array1 = np.arange(5)
print("Using np.arange(5):", array1)
# create an array with values from 1 to 8 with a step of 2
array2 = np.arange(1, 9, 2)
print("Using np.arange(1, 9, 2):",array2)
```

Output



```
import numpy as np
array1 = np.arange(5)
print("Using np.arange(5):", array1)
array2 = np.arange(1, 9, 2)
print("Using np.arange(1, 9, 2):",array2)
```

```
PS D:\MARWADI\YEAR2\SEM3\PYTHON> python -u "d:\MARWAD

                                 python -u "d:\MARWAD
Using np.arange(5): [0 1 2 3 4]
Using np.arange(1, 9, 2): [1 3 5 7]
```

Create an Array With np.random.rand()

The np.random.rand() function is used to create an array of random numbers. Let's see an example to create an array of **5** random numbers,

Example

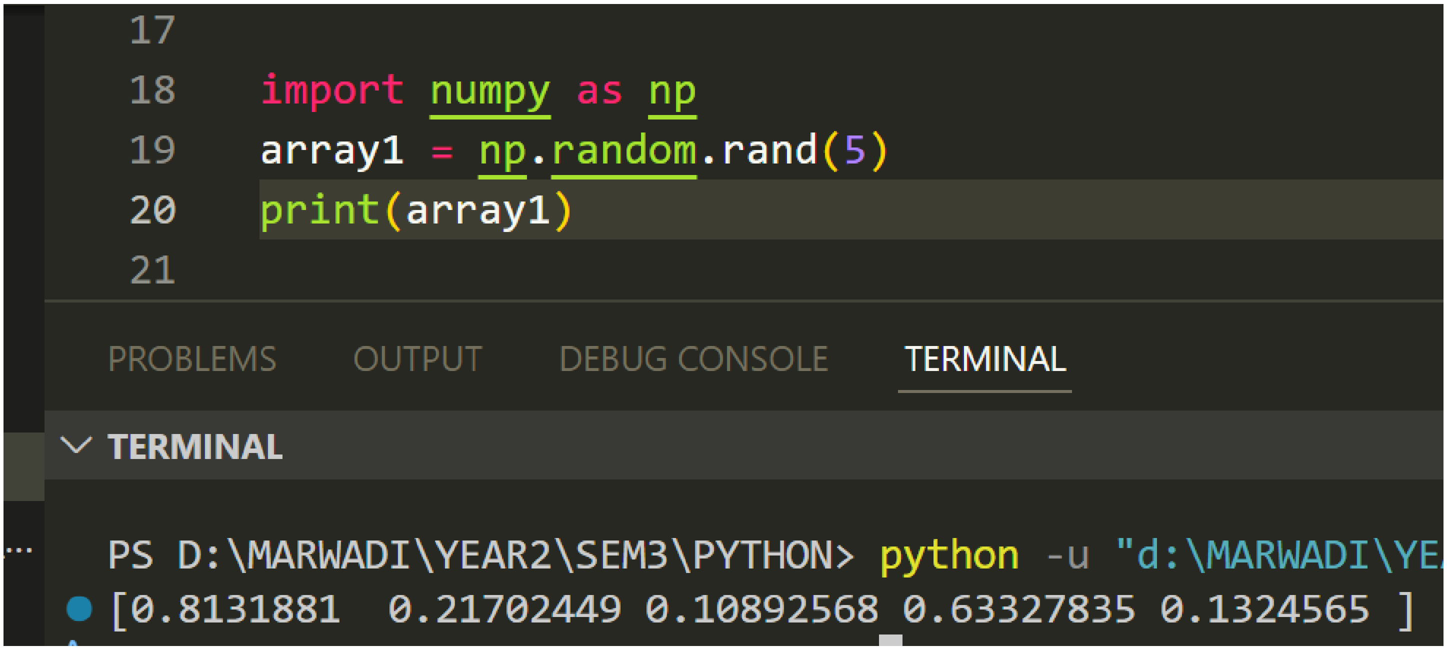
import numpy as np

# generate an array of 5 random numbers

array1 = np.random.rand(5)

print(array1)

Output



```
import numpy as np
array1 = np.random.rand(5)
print(array1)
```

```
PS D:\MARWADI\YEAR2\SEM3\PYTHON> python -u "d:\MARWADI\YE
[0.8131881  0.21702449 0.10892568 0.63327835 0.1324565 ]
```

Taks:

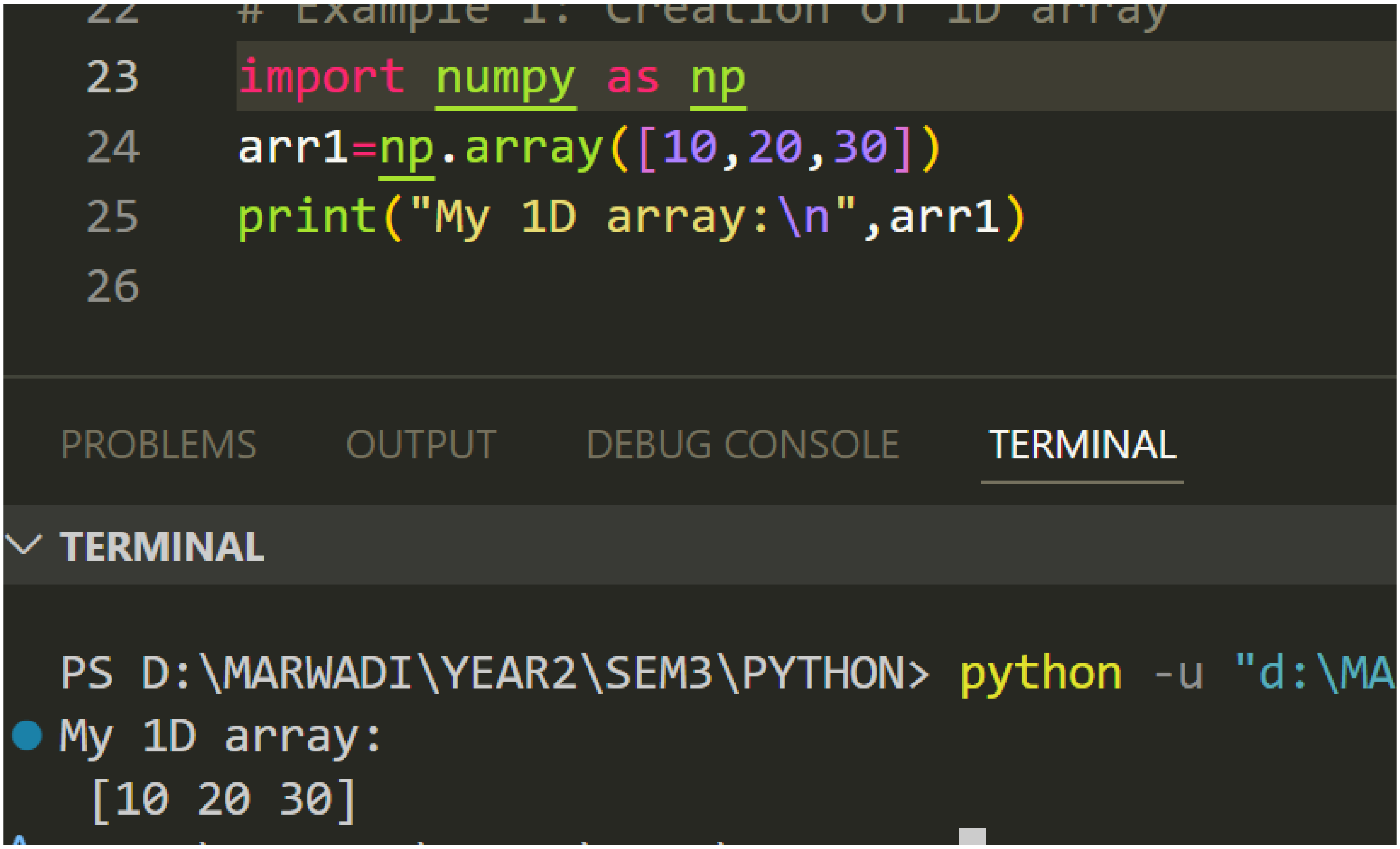
import numpy as np

# Example 1: Creation of 1D array

arr1=np.array([10,20,30])

print("My 1D array:\n",arr1)

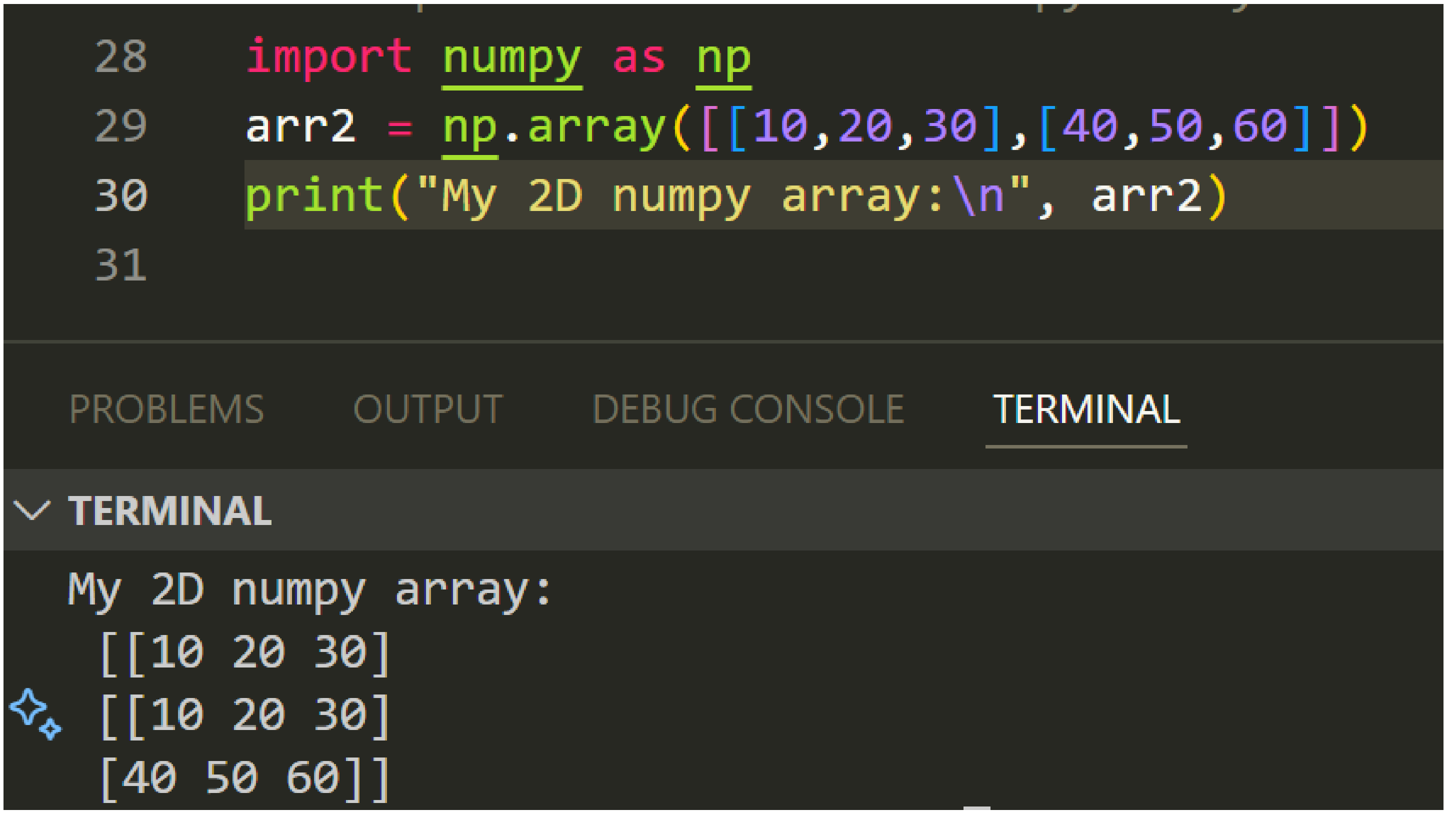
Output



```python
# Example 1: Creation of 1D array
import numpy as np
arr1=np.array([10,20,30])
print("My 1D array:\n",arr1)
```

```
PS D:\MARWADI\YEAR2\SEM3\PYTHON> python -u "d:\MA
My 1D array:
 [10 20 30]
```

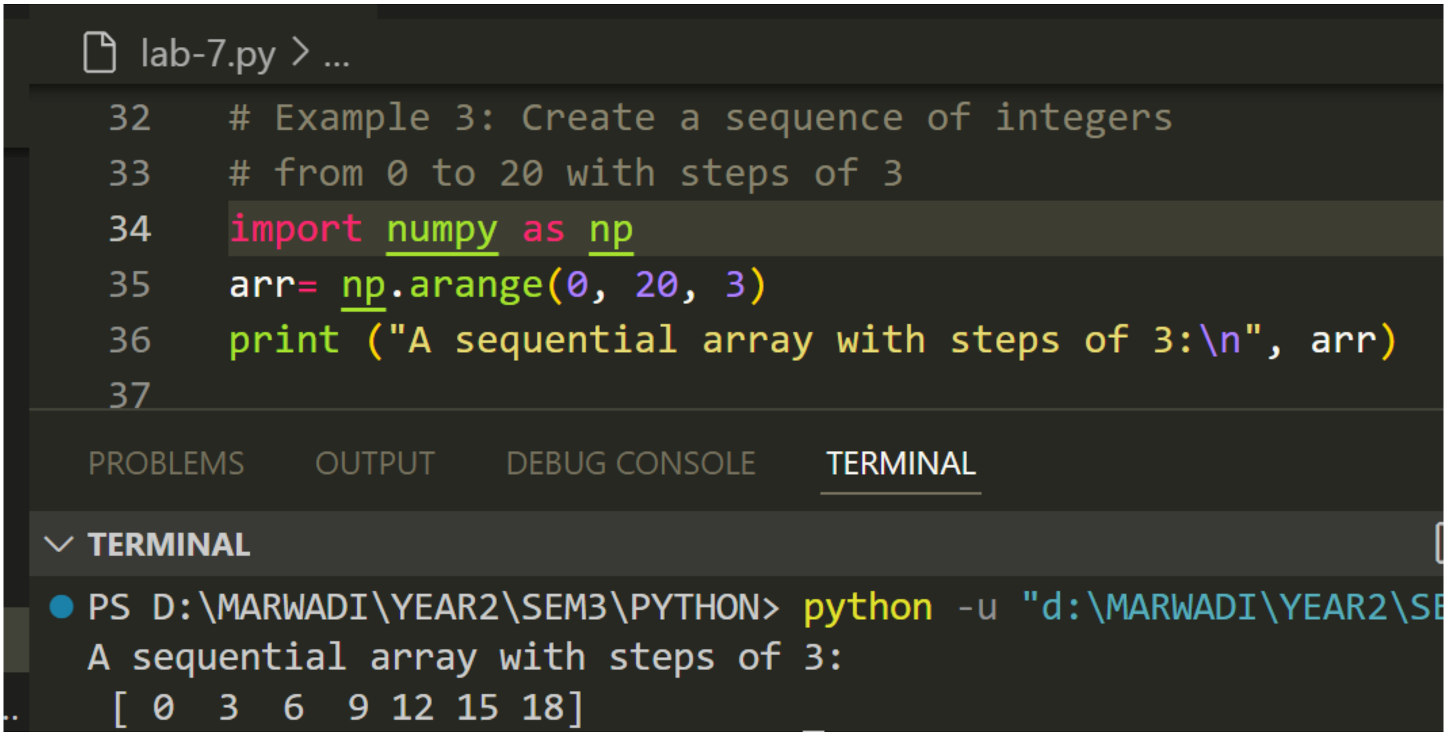
# Example 2: Create a 2D numpy array
arr2 = np.array([[10,20,30],[40,50,60]])
print("My 2D numpy array:\n", arr2)
Output



```
import numpy as np
arr2 = np.array([[10,20,30],[40,50,60]])
print("My 2D numpy array:\n", arr2)
```

```
My 2D numpy array:
 [[10 20 30]
 [[10 20 30]
 [40 50 60]]
```

# Example 3: Create a sequence of integers
# from 0 to 20 with steps of 3
arr= np.arange(0, 20, 3)
print ("A sequential array with steps of 3:\n", arr)
Output



```python
# Example 3: Create a sequence of integers
# from 0 to 20 with steps of 3
import numpy as np
arr= np.arange(0, 20, 3)
print ("A sequential array with steps of 3:\n", arr)
```

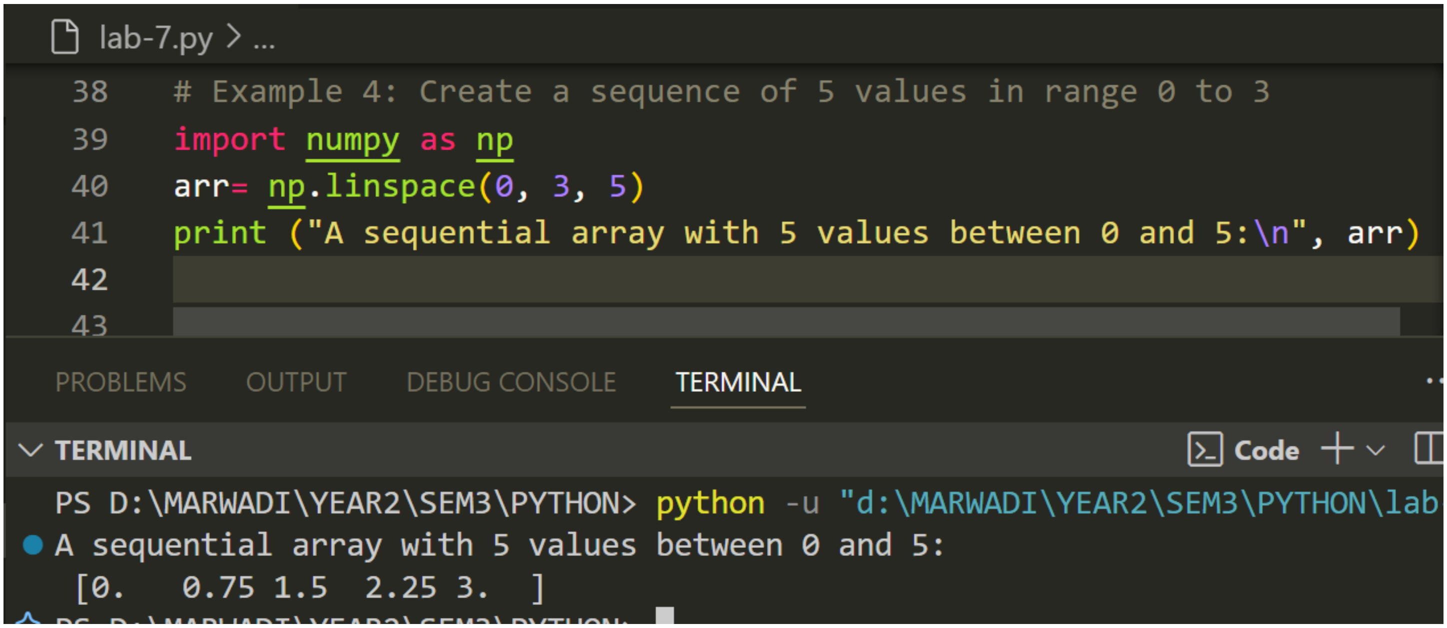
```
PS D:\MARWADI\YEAR2\SEM3\PYTHON> python -u "d:\MARWADI\YEAR2\SE
A sequential array with steps of 3:
 [ 0  3  6  9 12 15 18]
```

# Example 4: Create a sequence of 5 values in range 0 to 3

arr= np.linspace(0, 3, 5)

print ("A sequential array with 5 values between 0 and 5:\n", arr)

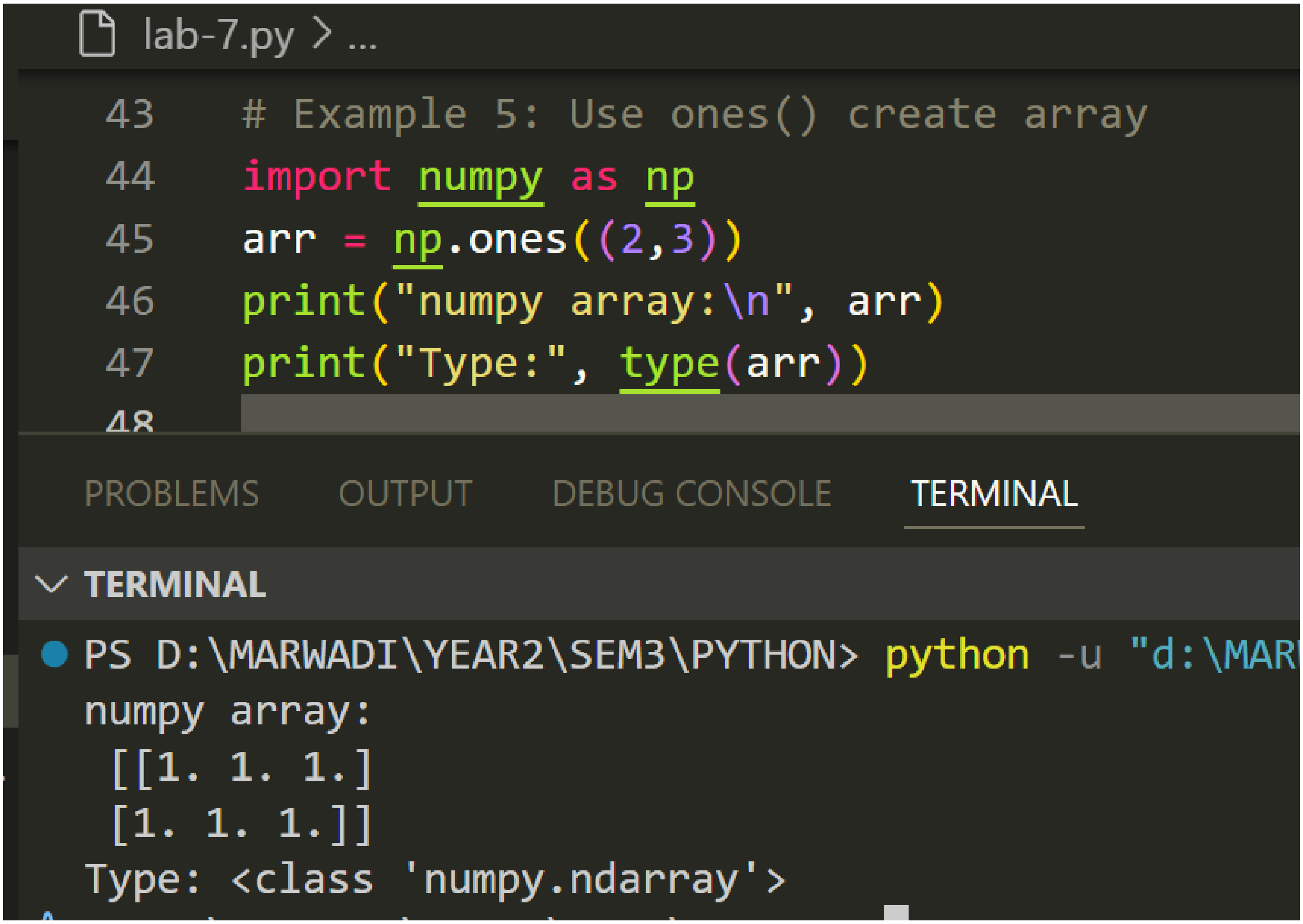
Output



```
# Example 4: Create a sequence of 5 values in range 0 to 3
import numpy as np
arr= np.linspace(0, 3, 5)
print ("A sequential array with 5 values between 0 and 5:\n", arr)
```

```
PS D:\MARWADI\YEAR2\SEM3\PYTHON> python -u "d:\MARWADI\YEAR2\SEM3\PYTHON\lab
A sequential array with 5 values between 0 and 5:
 [0.   0.75 1.5  2.25 3.  ]
PS D:\MARWADI\YEAR2\SEM3\PYTHON>
```

# Example 5: Use ones() create array
arr = np.ones((2,3))
print("numpy array:\n", arr)
print("Type:", type(arr))
Output



```python
# Example 5: Use ones() create array
import numpy as np
arr = np.ones((2,3))
print("numpy array:\n", arr)
print("Type:", type(arr))
```

```
PS D:\MARWADI\YEAR2\SEM3\PYTHON> python -u "d:\MAR
numpy array:
 [[1. 1. 1.]
 [1. 1. 1.]]
Type: <class 'numpy.ndarray'>
```

NumPy Data Types

A data type is a way to specify the type of data that will be stored in an array. For example,

array1 = np.array([2, 4, 6])

NumPy Data Types

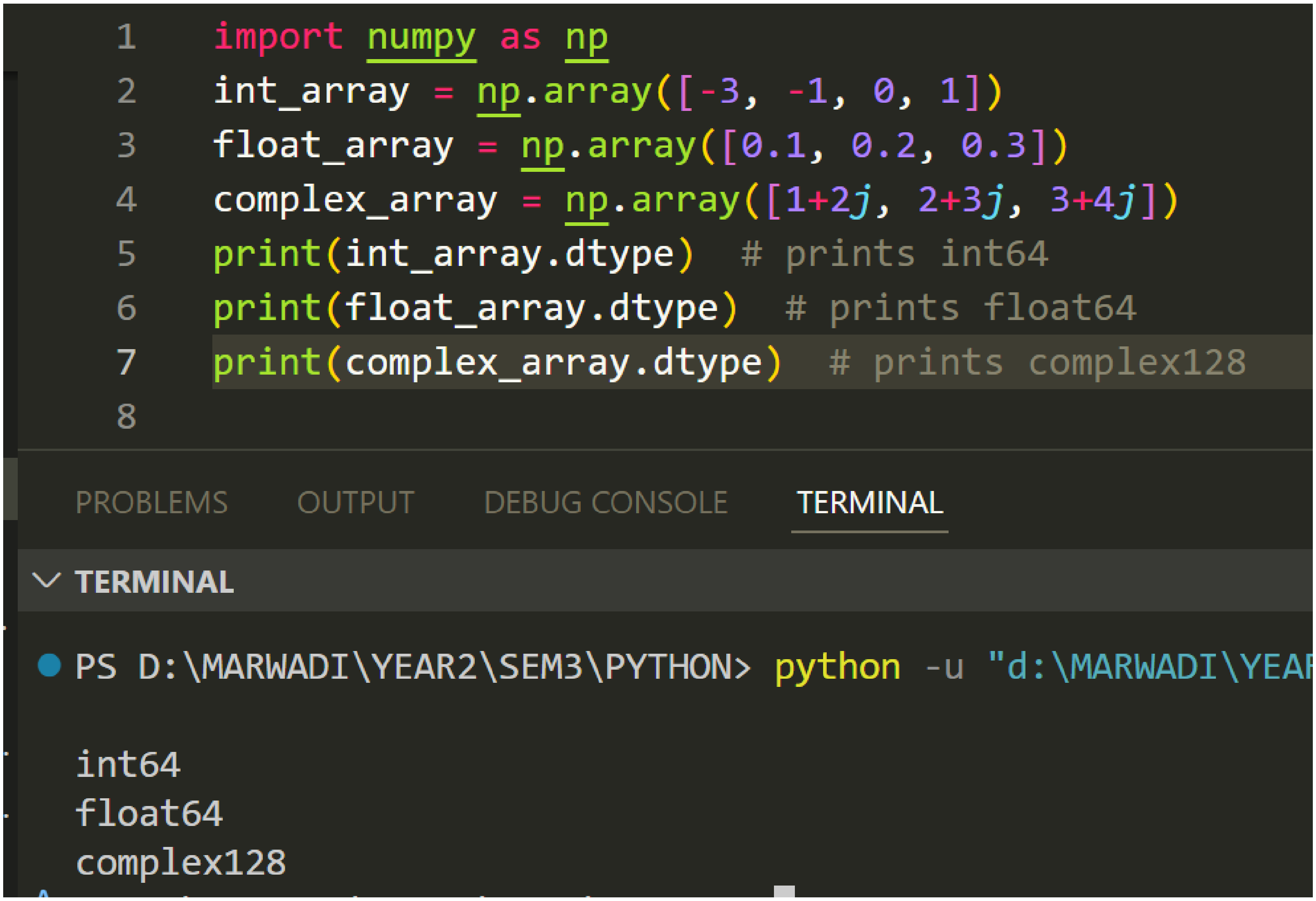
NumPy offers a wider range of numerical data types than what is available in Python. Here's the list of most commonly used numeric data types in NumPy:

1. int8, int16, int32, int64 - signed integer types with different bit sizes
2. uint8, uint16, uint32, uint64 - unsigned integer types with different bit sizes
3. float32, float64 - floating-point types with different precision levels
4. complex64, complex128 - complex number types with different precision levels

Check Data Type of a NumPy Array

```
import numpy as np
# create an array of  integers
int_array = np.array([-3, -1, 0, 1])
# create an array of floating-point numbers
float_array = np.array([0.1, 0.2, 0.3])
# create an array of complex numbers
complex_array = np.array([1+2j, 2+3j, 3+4j])
# check the data type of int_array
print(int_array.dtype)  # prints int64
# check the data type of float_array
print(float_array.dtype)  # prints float64
# check the data type of complex_array
print(complex_array.dtype)  # prints complex128
```

Output



```python
import numpy as np
int_array = np.array([-3, -1, 0, 1])
float_array = np.array([0.1, 0.2, 0.3])
complex_array = np.array([1+2j, 2+3j, 3+4j])
print(int_array.dtype)  # prints int64
print(float_array.dtype)  # prints float64
print(complex_array.dtype)  # prints complex128

```

```
PS D:\MARWADI\YEAR2\SEM3\PYTHON> python -u "d:\MARWADI\YEAR

int64
float64
complex128
```

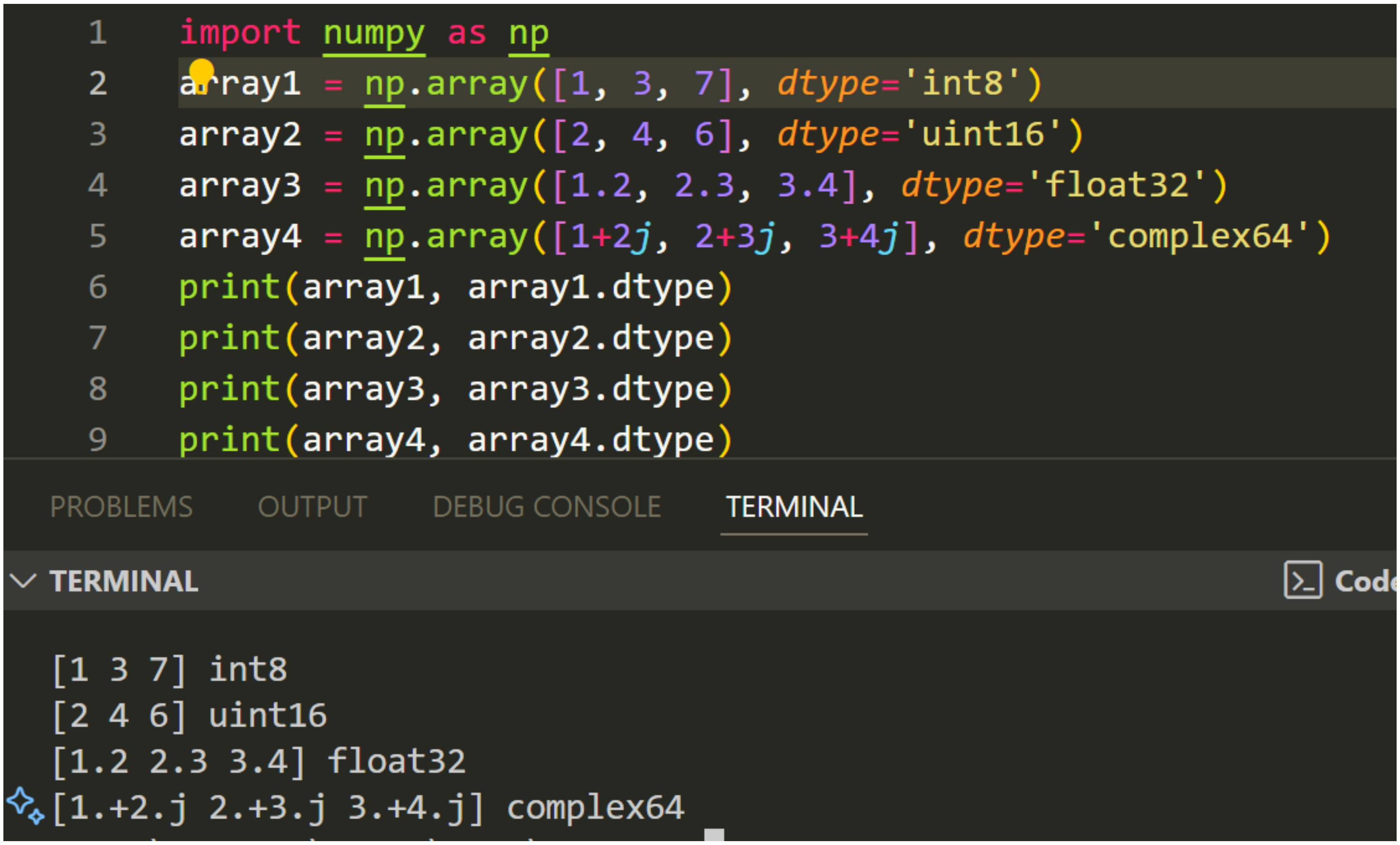
Creating NumPy Arrays With a Defined Data Type

In NumPy, we can create an array with a defined data type by passing the dtype parameter while calling the np.array() function. For example,

import numpy as np
# create an array of 8-bit integers

```
array1 = np.array([1, 3, 7], dtype='int8')
# create an array of unsigned 16-bit integers
array2 = np.array([2, 4, 6], dtype='uint16')
# create an array of 32-bit floating-point numbers
array3 = np.array([1.2, 2.3, 3.4], dtype='float32')
# create an array of 64-bit complex numbers
array4 = np.array([1+2j, 2+3j, 3+4j], dtype='complex64')
# print the arrays and their data types
print(array1, array1.dtype)
print(array2, array2.dtype)
print(array3, array3.dtype)
print(array4, array4.dtype)
```

Output



```
import numpy as np
array1 = np.array([1, 3, 7], dtype='int8')
array2 = np.array([2, 4, 6], dtype='uint16')
array3 = np.array([1.2, 2.3, 3.4], dtype='float32')
array4 = np.array([1+2j, 2+3j, 3+4j], dtype='complex64')
print(array1, array1.dtype)
print(array2, array2.dtype)
print(array3, array3.dtype)
print(array4, array4.dtype)
```

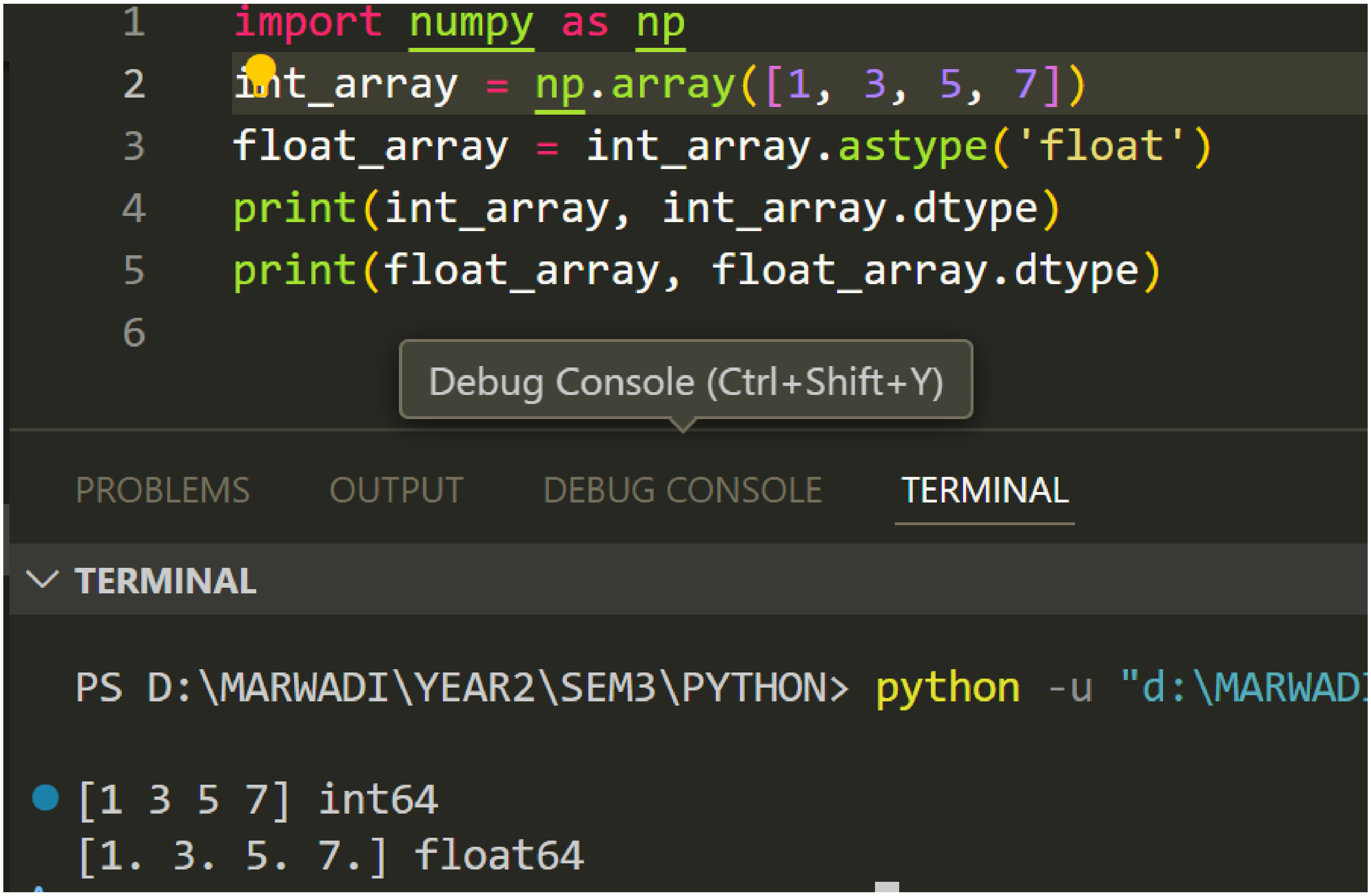
```
[1 3 7] int8
[2 4 6] uint16
[1.2 2.3 3.4] float32
[1.+2.j 2.+3.j 3.+4.j] complex64
```

NumPy Type Conversion

In NumPy, we can convert the data type of an array using the astype() method. For example,

```
import numpy as np
# create an array of integers
int_array = np.array([1, 3, 5, 7])
# convert data type of int_array to float
float_array = int_array.astype('float')
# print the arrays and their data types
print(int_array, int_array.dtype)
print(float_array, float_array.dtype)
```

Output



```
import numpy as np
int_array = np.array([1, 3, 5, 7])
float_array = int_array.astype('float')
print(int_array, int_array.dtype)
print(float_array, float_array.dtype)
```

```
PS D:\MARWADI\YEAR2\SEM3\PYTHON> python -u "d:\MARWADI
[1 3 5 7] int64
[1. 3. 5. 7.] float64
```

NumPy Array Attributes

In NumPy, attributes are properties of NumPy arrays that provide information about the array's shape, size, data type, dimension, and so on.

Common NumPy Attributes

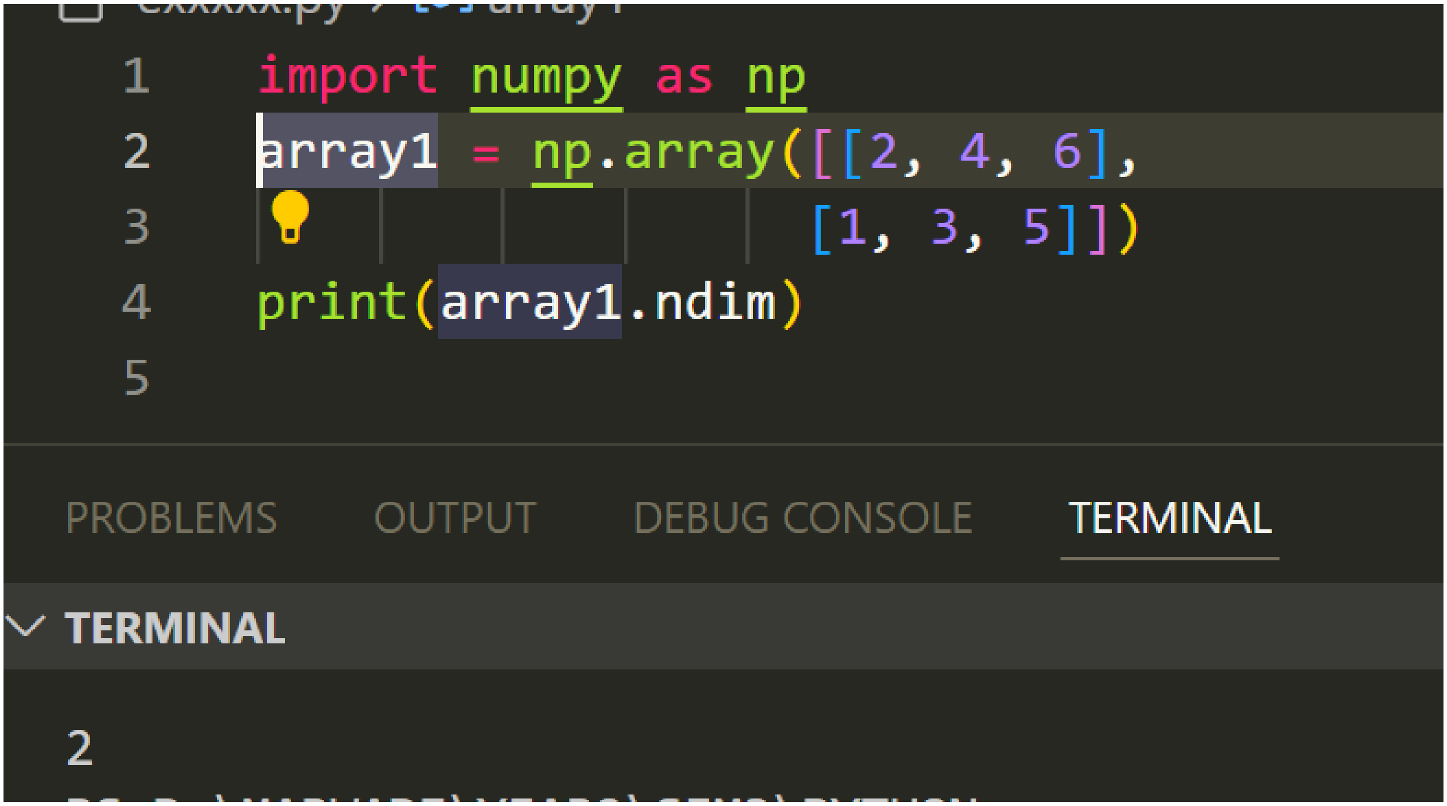
Here are some of the commonly used NumPy attributes:

Attributes	Description
ndim	returns number of dimension of the array
size	returns number of elements in the array
dtype	returns data type of elements in the array
shape	returns the size of the array in each dimension.
itemsize	returns the size (in bytes) of each elements in the array
data	returns the buffer containing actual elements of the array in memory

The ndim attribute returns the number of dimensions in the numpy array. For example,

```
import numpy as np
# create a 2-D array
array1 = np.array([[2, 4, 6],
              [1, 3, 5]])
# check the dimension of array1
print(array1.ndim)
```

Output



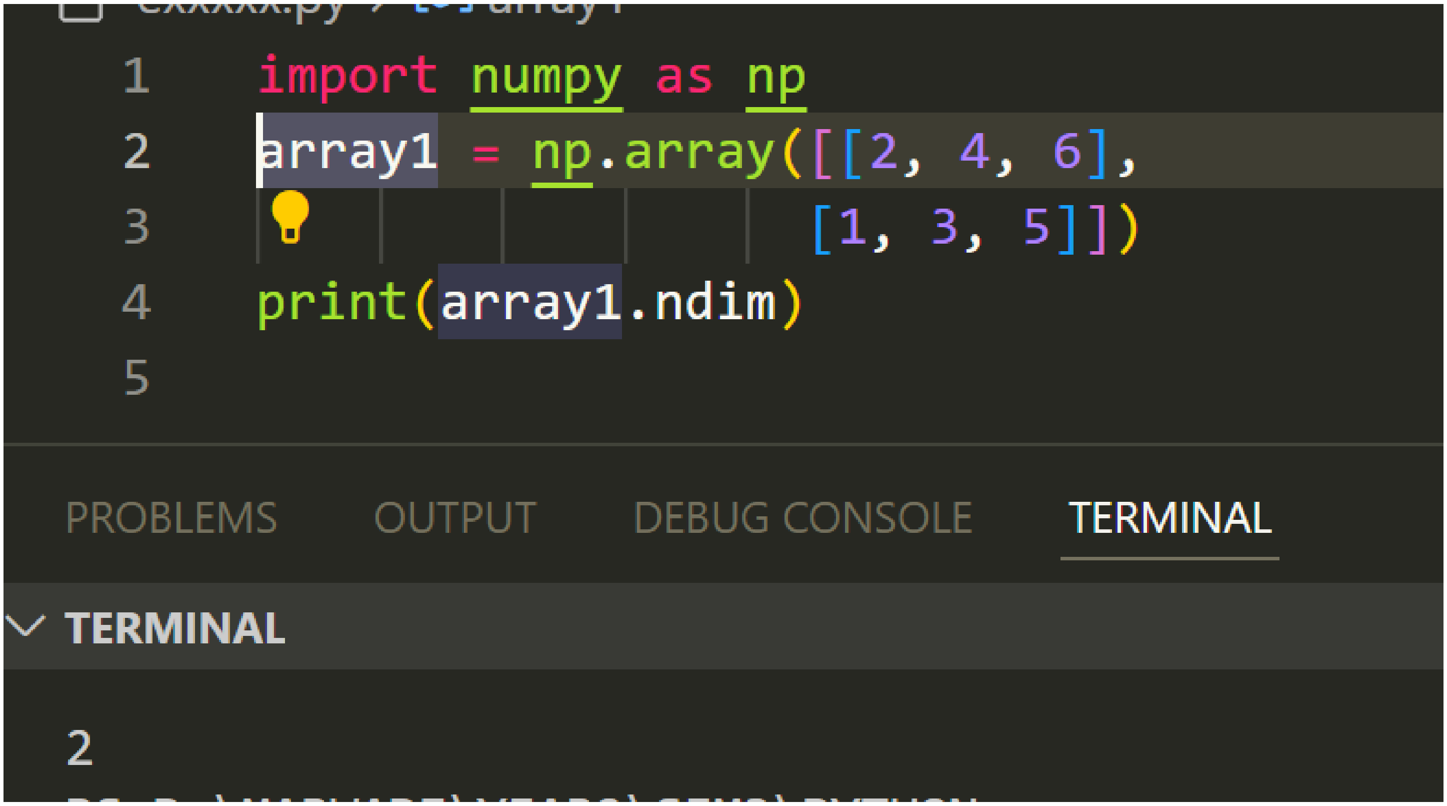
```
import numpy as np
array1 = np.array([[2, 4, 6],
                   [1, 3, 5]])
print(array1.ndim)
```

```
2
```

NumPy Array size Attribute
The size attribute returns the total number of elements in the given array.

```
import numpy as np
array1 = np.array([[1, 2, 3],
              [6, 7, 8]])
# return total number of elements in array1
print(array1.size)
```

Output



```
import numpy as np
array1 = np.array([[2, 4, 6],
                   [1, 3, 5]])
print(array1.ndim)
```

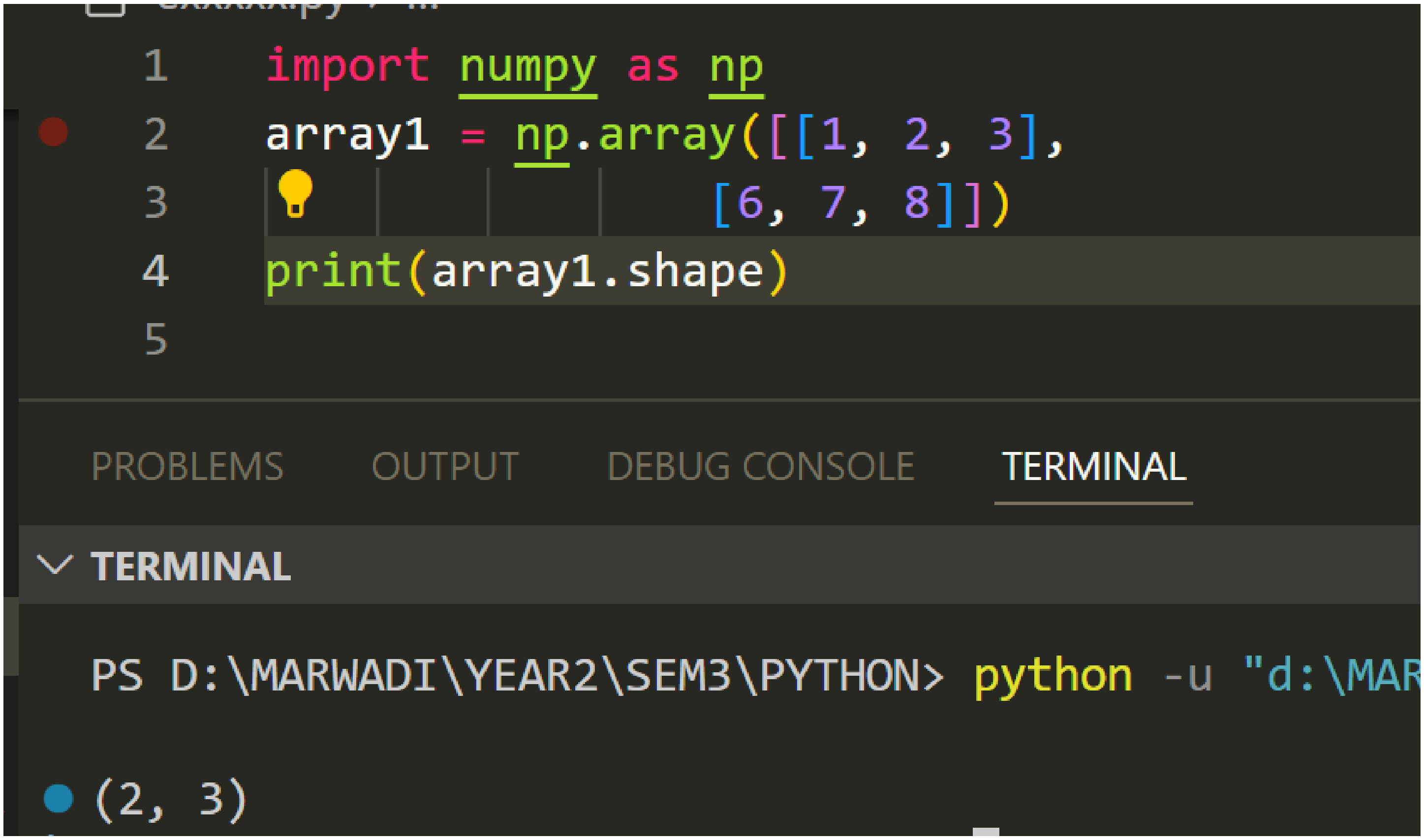
```
2
```

NumPy Array shape Attribute

In NumPy, the shape attribute returns a tuple of integers that gives the size of the array in each dimension. For example,

```
import numpy as np
array1 = np.array([[1, 2, 3],
            [6, 7, 8]])
# return a tuple that gives size of array in each dimension
print(array1.shape)
```

Output



```python
import numpy as np
array1 = np.array([[1, 2, 3],
                   [6, 7, 8]])
print(array1.shape)
```

```
PS D:\MARWADI\YEAR2\SEM3\PYTHON> python -u "d:\MAR
(2, 3)
```

NumPy Array dtype Attribute

We can use the dtype attribute to check the datatype of a NumPy array. For example,

import numpy as np

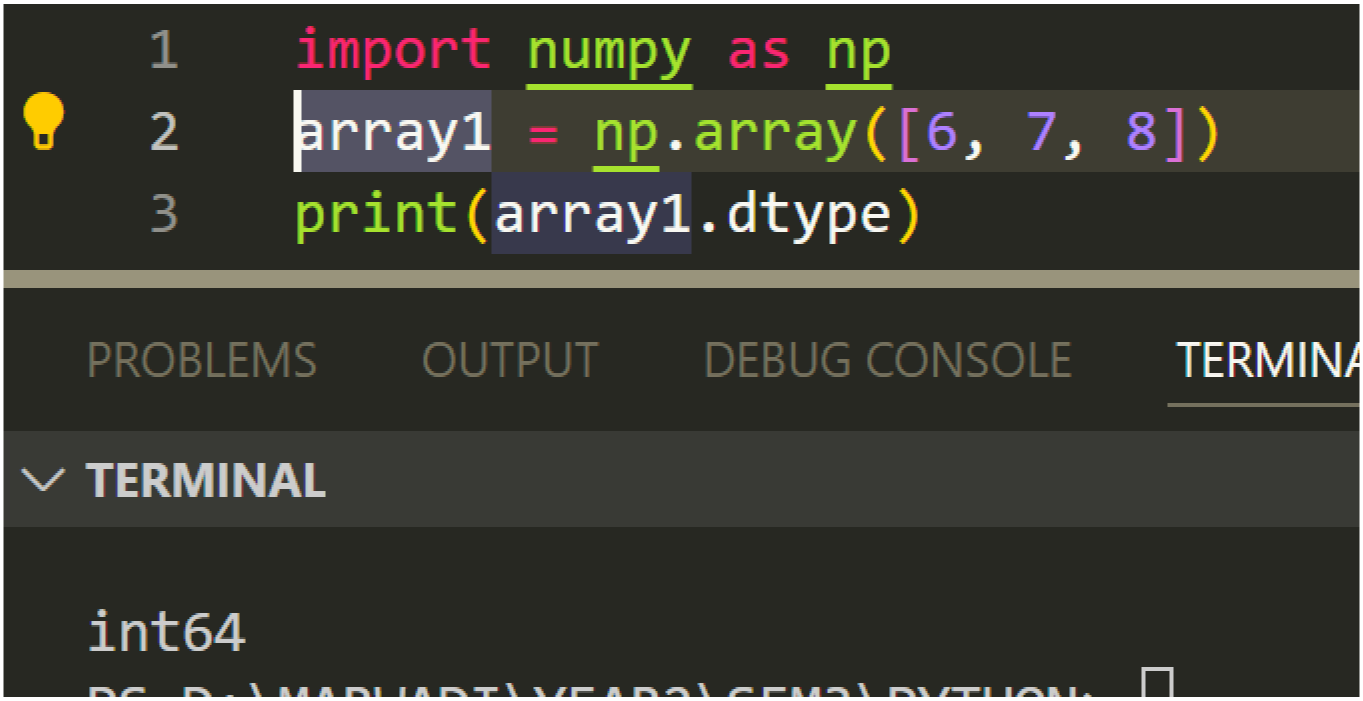
# create an array of integers

array1 = np.array([6, 7, 8])

# check the data type of array1

print(array1.dtype)

Output



```python
import numpy as np
array1 = np.array([6, 7, 8])
print(array1.dtype)
```

```
int64
PS D:\MARWADI\YEAR2\SEM3\PYTHON:
```

NumPy Array itemsize Attribute

In NumPy, the itemsize attribute determines size (in bytes) of each element in the array. For example,

```
import numpy as np
# create a default 1-D array of integers
array1 = np.array([6, 7, 8, 10, 13])
# create a 1-D array of 32-bit integers
array2 = np.array([6, 7, 8, 10, 13], dtype=np.int32)
# use of itemsize to determine size of each array element of array1 and array2
print(array1.itemsize)  # prints 8
print(array2.itemsize)  # prints 4
```

Output

NumPy Array data Attribute

In NumPy, we can get a buffer containing actual elements of the array in memory using the data attribute.

In simpler terms, the data attribute is like a pointer to the memory location where the array's data is stored in the computer's memory.

Subject: Programming With Python (01CT1309)	**Aim:** Practical based on NumPy ndarray	
Experiment No: 07	**Date:**	**Enrollment No:92400133189**

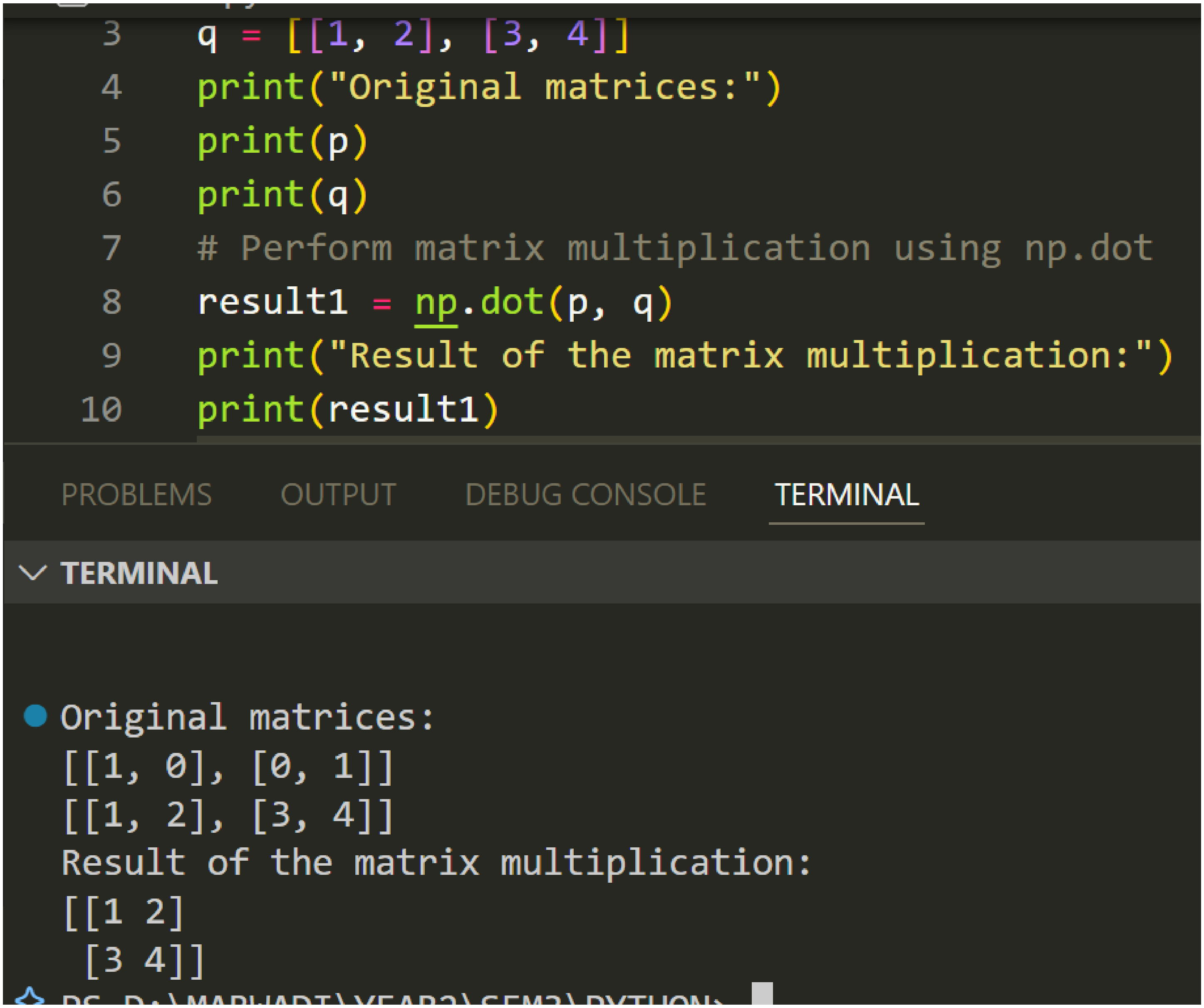
```
import numpy as np
array1 = np.array([6, 7, 8])
array2 = np.array([[1, 2, 3],
                   [6, 7, 8]])
# print memory address of array1's and array2's data
print("\nData of array1 is: ",array1.data)
print("Data of array2 is: ",array2.data)
```

Task

Multiplication of two given matrixes

```
import numpy as np
p = [[1, 0], [0, 1]]
q = [[1, 2], [3, 4]]
print("Original matrices:")
print(p)
print(q)
# Perform matrix multiplication using np.dot
result1 = np.dot(p, q)
print("Result of the matrix multiplication:")
print(result1)
```

Output



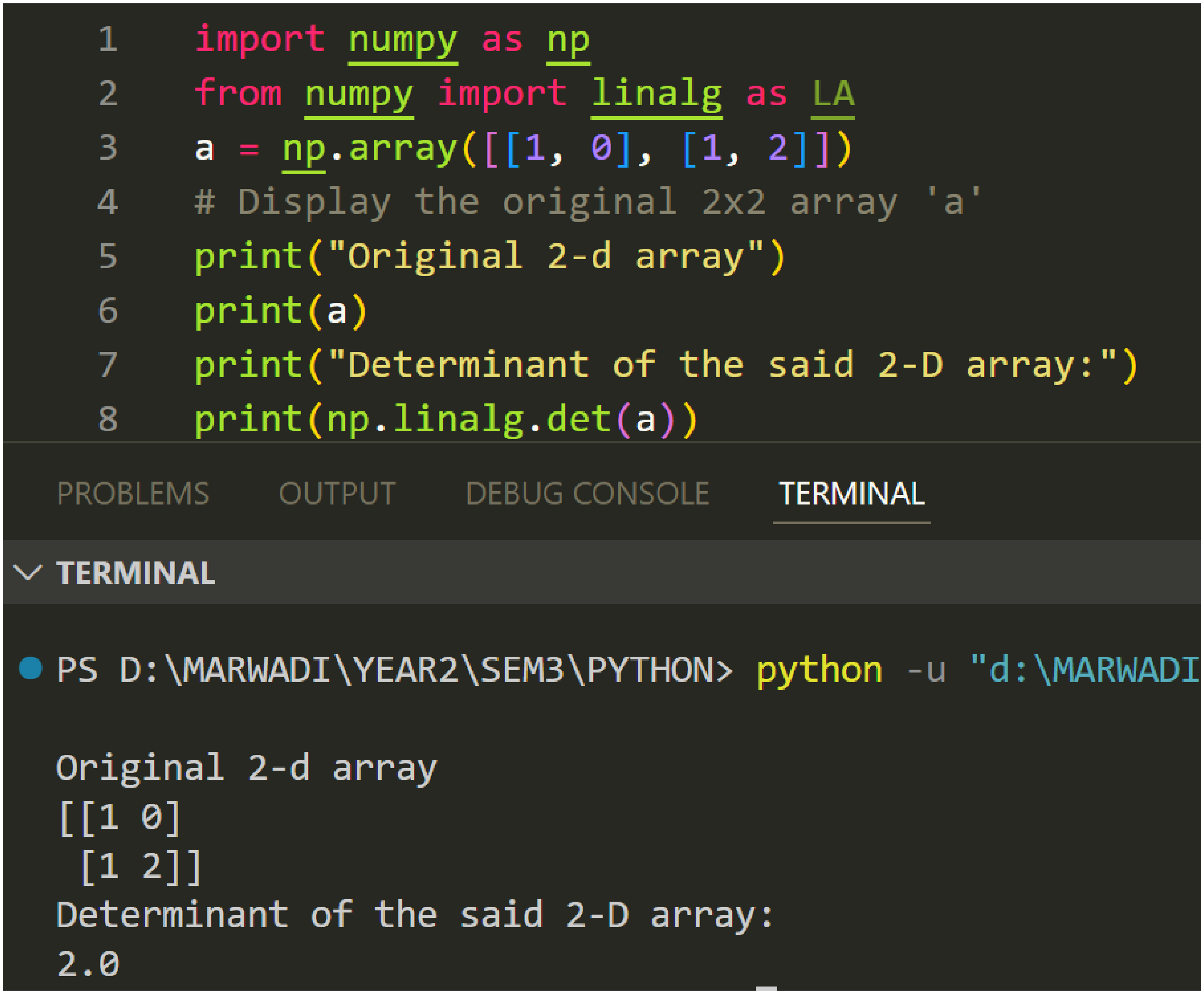
```
q = [[1, 2], [3, 4]]
print("Original matrices:")
print(p)
print(q)
# Perform matrix multiplication using np.dot
result1 = np.dot(p, q)
print("Result of the matrix multiplication:")
print(result1)
```

```
Original matrices:
[[1, 0], [0, 1]]
[[1, 2], [3, 4]]
Result of the matrix multiplication:
[[1 2]
 [3 4]]
PS D:\MARWADI\YEAR2\SEM3\PYTHON>
```

Compute the determinant of a given square array.

```
import numpy as np
from numpy import linalg as LA
a = np.array([[1, 0], [1, 2]])
# Display the original 2x2 array 'a'
print("Original 2-d array")
print(a)
print("Determinant of the said 2-D array:")
print(np.linalg.det(a))
```

Output

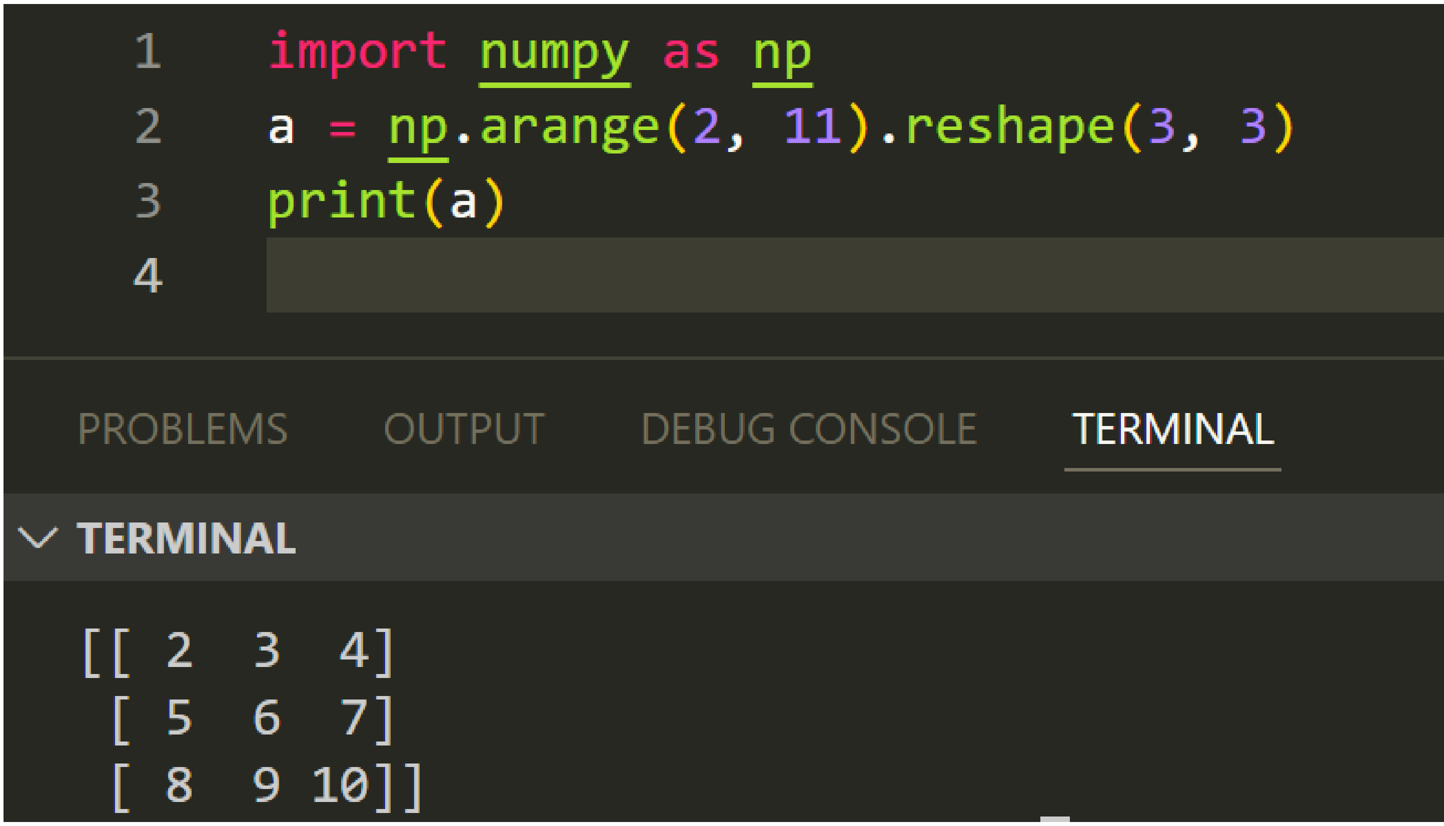


```python
import numpy as np
from numpy import linalg as LA
a = np.array([[1, 0], [1, 2]])
# Display the original 2x2 array 'a'
print("Original 2-d array")
print(a)
print("Determinant of the said 2-D array:")
print(np.linalg.det(a))
```

```
PS D:\MARWADI\YEAR2\SEM3\PYTHON> python -u "d:\MARWADI

Original 2-d array
[[1 0]
 [1 2]]
Determinant of the said 2-D array:
2.0
```

Post Lab Exercise:

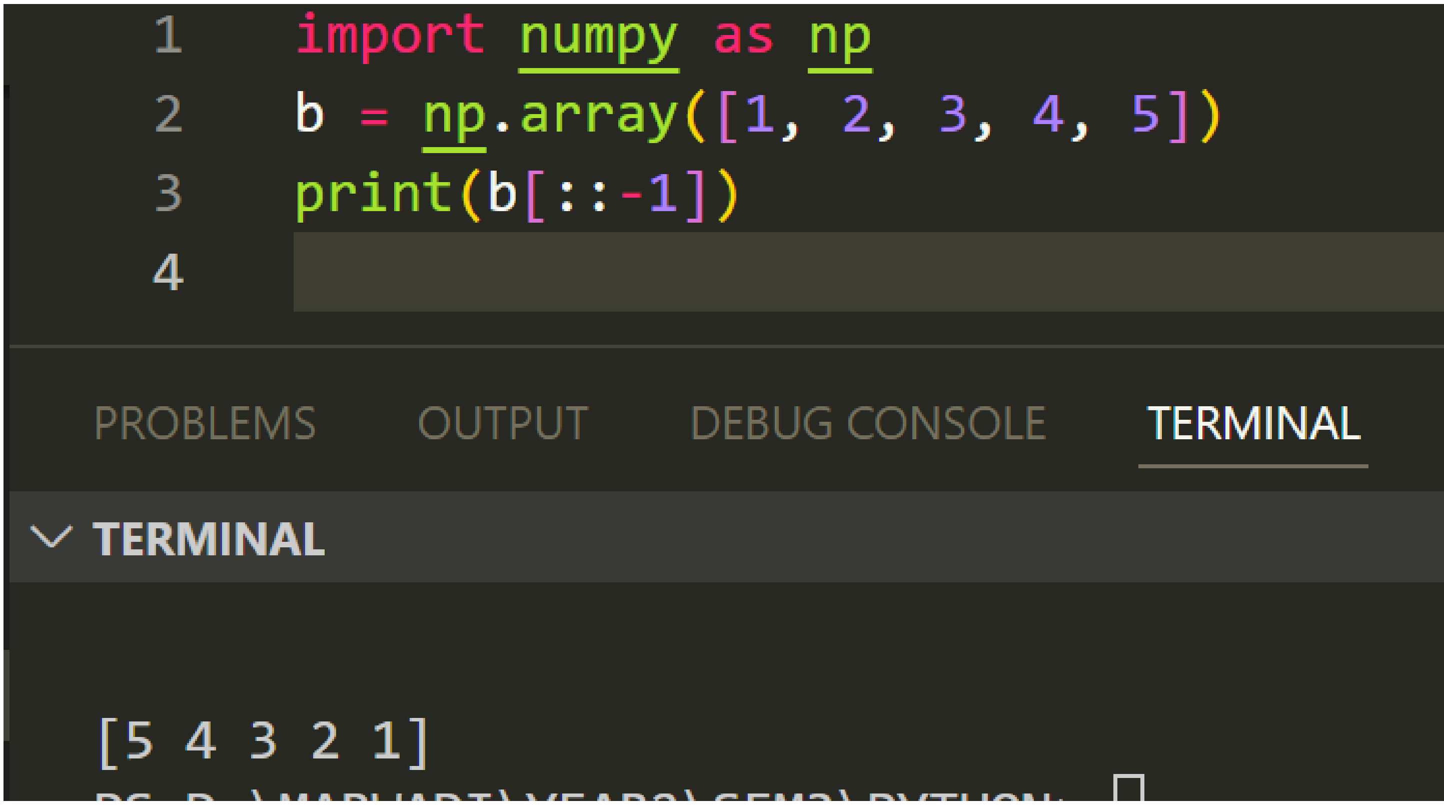
a. Write a NumPy program to create a 3x3 matrix with values ranging from 2 to 10.



```python
import numpy as np
a = np.arange(2, 11).reshape(3, 3)
print(a)
```

```
[[ 2  3  4]
 [ 5  6  7]
 [ 8  9 10]]
```

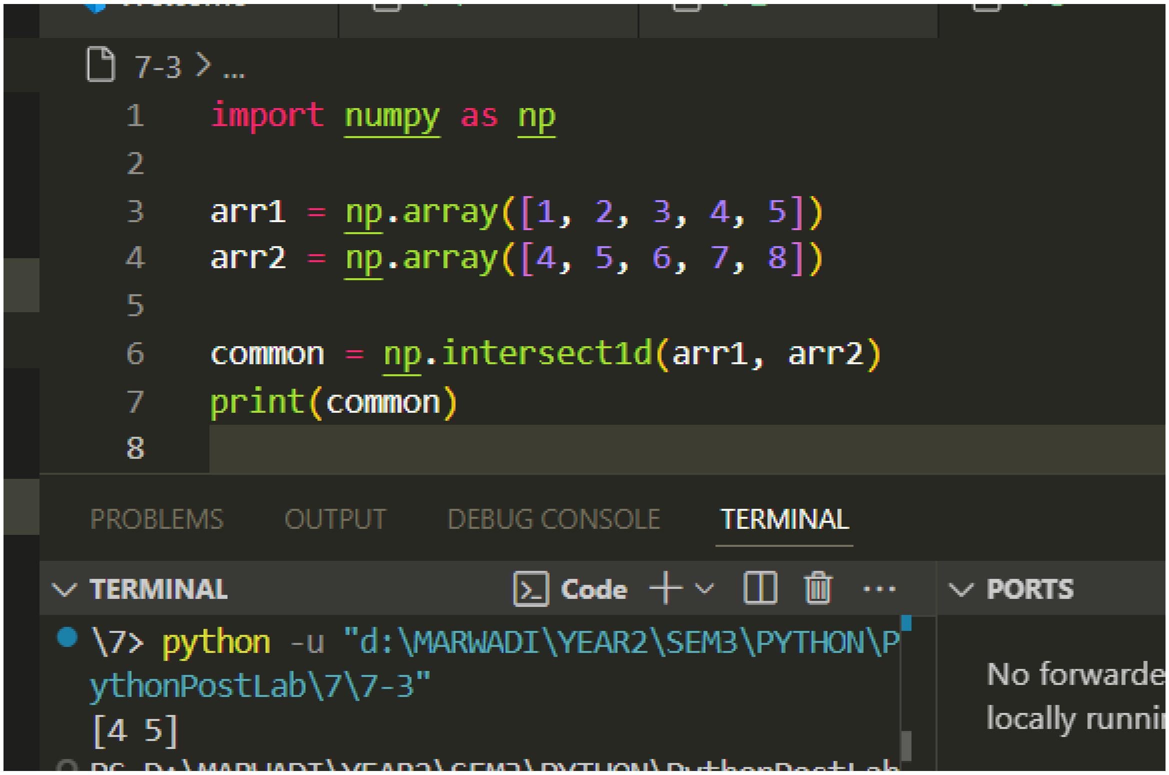
b. Write a NumPy program to reverse an array (the first element becomes the last).



```python
import numpy as np
b = np.array([1, 2, 3, 4, 5])
print(b[::-1])

```

```
[5 4 3 2 1]
PS D:\MARWADI\YEAR2\SEM3\PYTHON>
```

c. Write a NumPy program to find common values between two arrays.

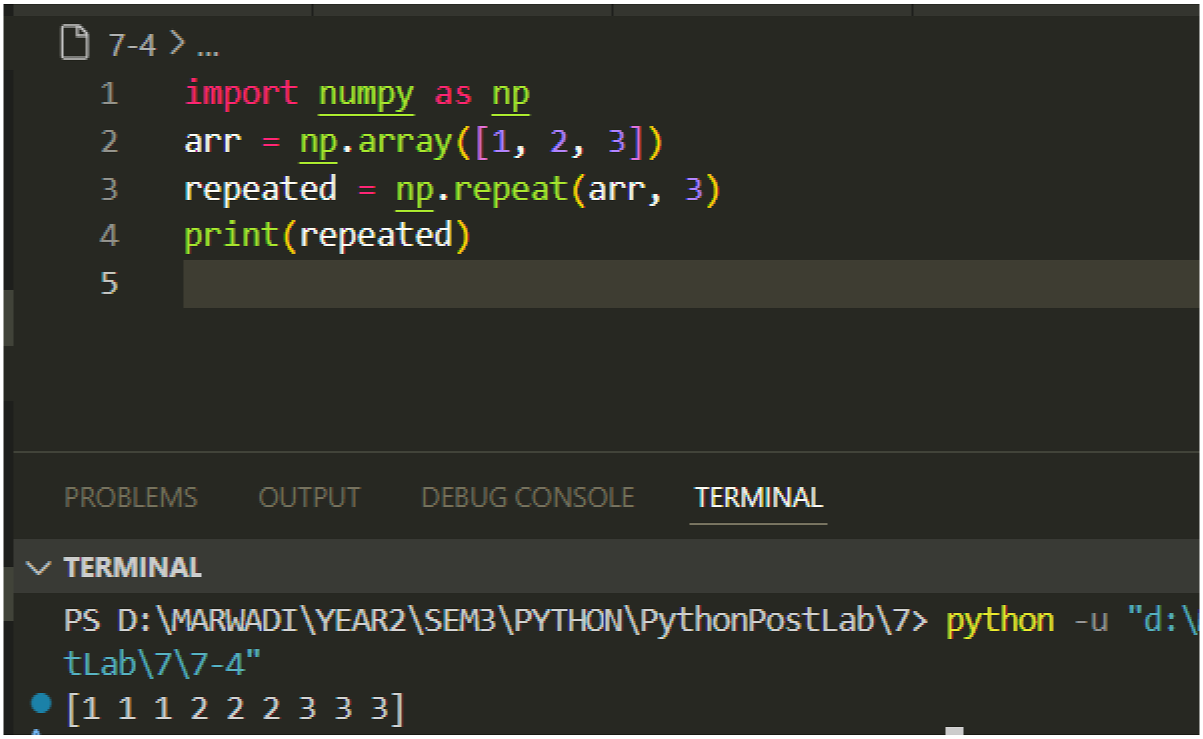


```python
import numpy as np

arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([4, 5, 6, 7, 8])

common = np.intersect1d(arr1, arr2)
print(common)
```

```
\7> python -u "d:\MARWADI\YEAR2\SEM3\PYTHON\PythonPostLab\7\7-3"
[4 5]
PS D:\MARWADI\YEAR2\SEM3\PYTHON\PythonPostLab
```

No forwarde
locally runni

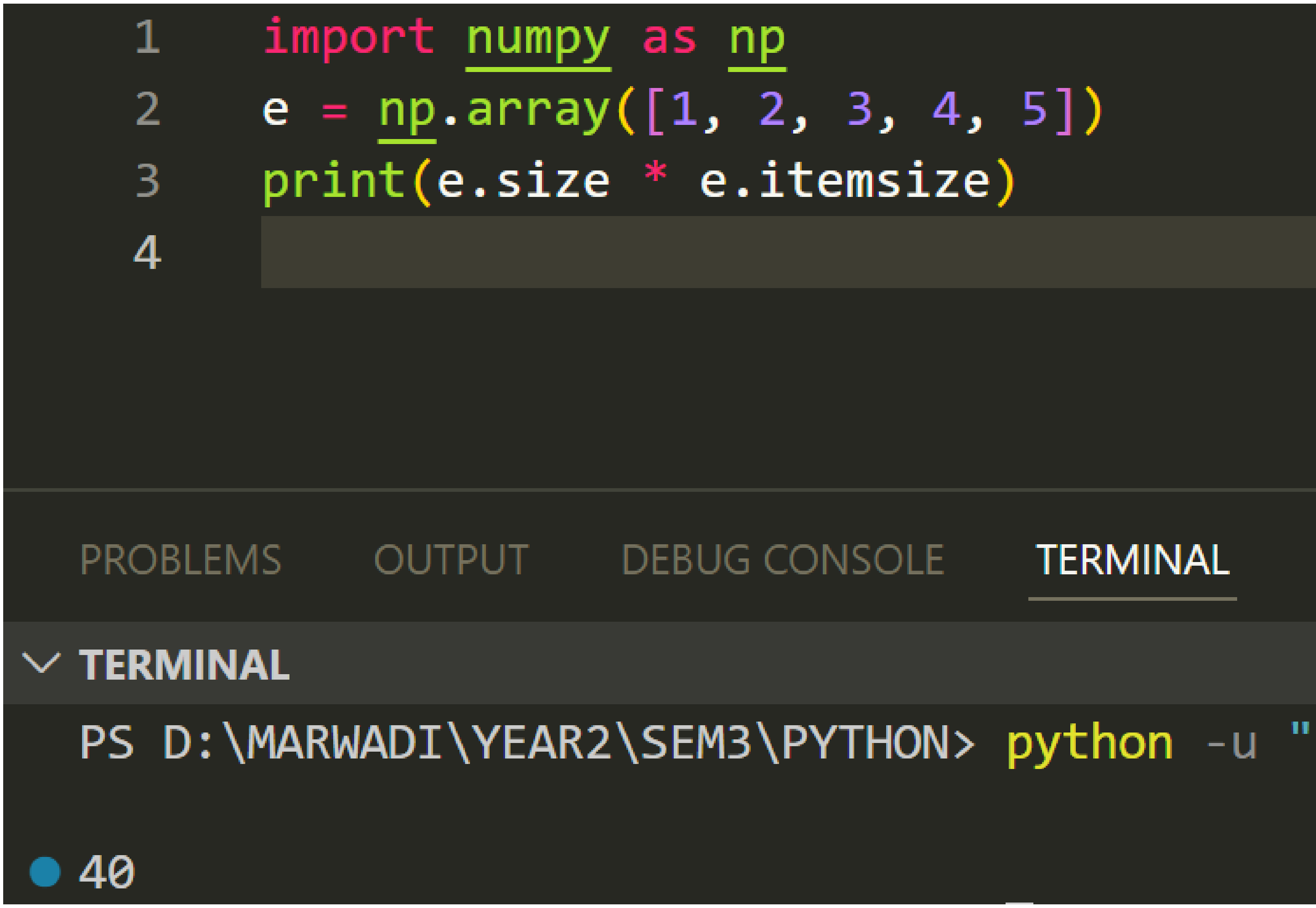
d. Write a NumPy program to repeat array elements.



```python
import numpy as np
arr = np.array([1, 2, 3])
repeated = np.repeat(arr, 3)
print(repeated)
```

```
PS D:\MARWADI\YEAR2\SEM3\PYTHON\PythonPostLab\7> python -u "d:\
tLab\7\7-4"
[1 1 1 2 2 2 3 3 3]
```

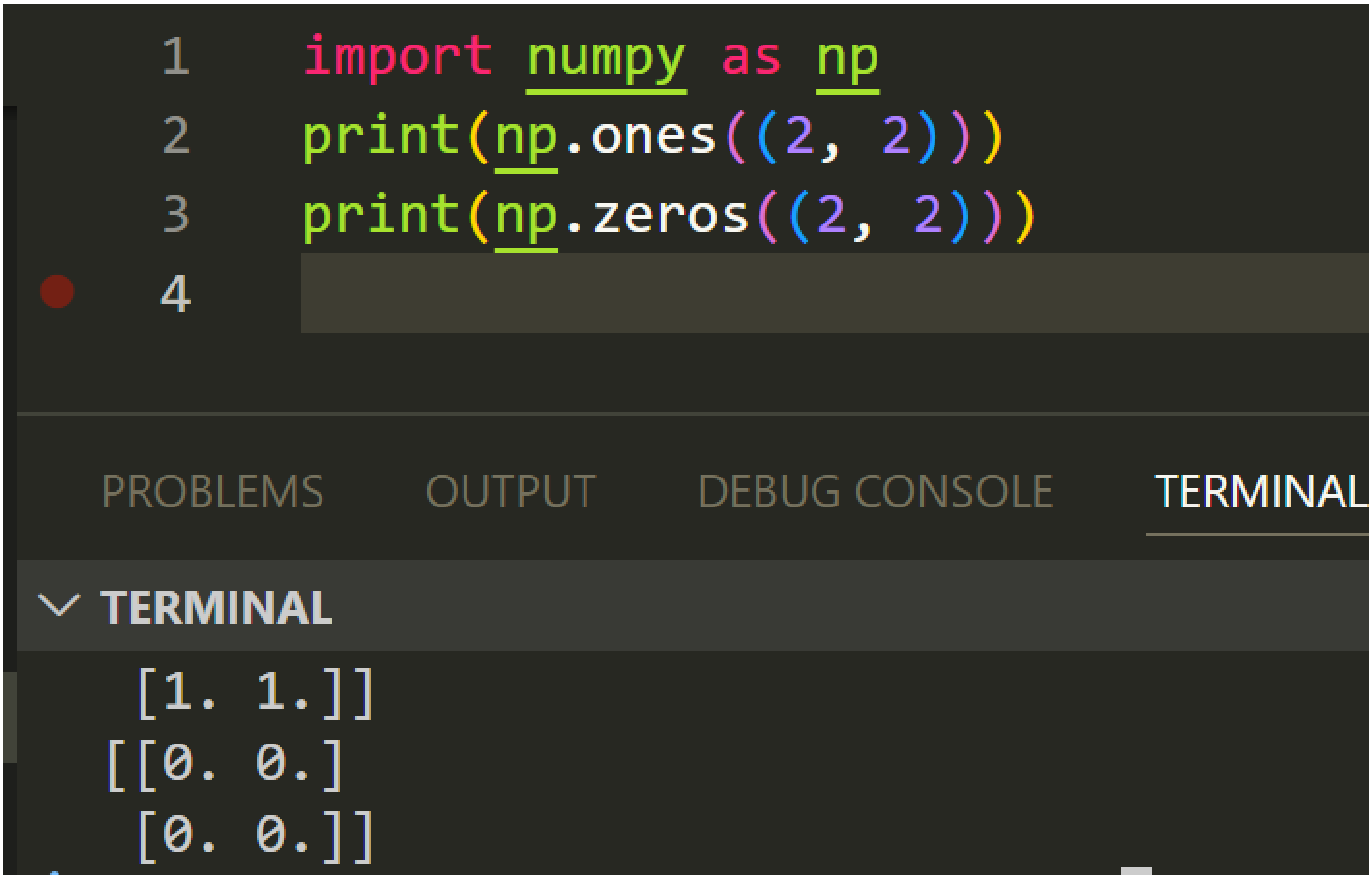
e. Write a NumPy program to find the memory size of a NumPy array.



```python
import numpy as np
e = np.array([1, 2, 3, 4, 5])
print(e.size * e.itemsize)
```

```
PS D:\MARWADI\YEAR2\SEM3\PYTHON> python -u "

40
```

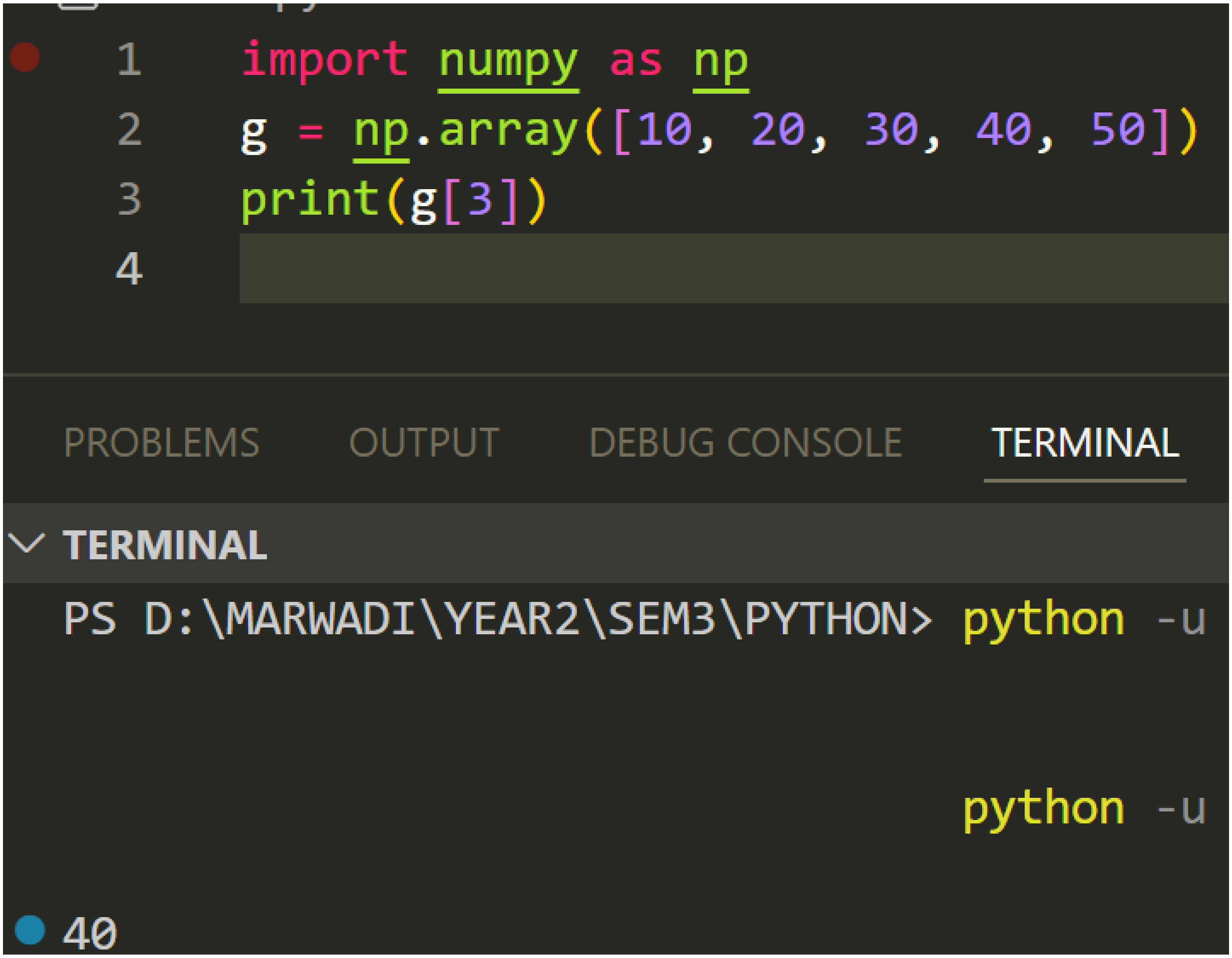
f. Write a NumPy program to create an array of ones and zeros.



```python
import numpy as np
print(np.ones((2, 2)))
print(np.zeros((2, 2)))
```

```
 [1. 1.]]
[[0. 0.]
 [0. 0.]]
```

g. Write a NumPy program to find the 4th element of a specified array.



```python
import numpy as np
g = np.array([10, 20, 30, 40, 50])
print(g[3])

```

```
PS D:\MARWADI\YEAR2\SEM3\PYTHON> python -u

                                 python -u

40
```

Github : PythonPostLab/7 at main · Om-Lathigara/PythonPostLab