1. 4) NumPy Ndarray Creation Routines.

1) **empty().**

Returns a new array of given shape and type, with random values.

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

x = np.empty([3, 3, 3], np.uint8)
print(x)
print("\n------------------")
```

```
[[[224 156  38]
  [  1 144 143]
  [ 38   1  40]]

 [[158  38   1]
  [200 159  38]
  [  1 200 159]]

 [[ 38   1 200]
  [159  38   1]
  [200 159  38]]]

------------------
```

2) **eye().**

Returns a 2-D array with ones on the diagonal and zeros elsewhere.

In [3]:
```
y = np.eye(5, dtype = np.uint8)
print(y)
print("\n------------------")
```

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

------------------
```

The eye() function may take a third argument as an optional argument, which is k. This argument shifts the main diagonal upwards or downwards according to the value of itself.

In [4]:
```
print("\n1------------------\n")
# The position of diagonal elements will be shifted upwards.
y = np.eye(5, dtype = np.uint8, k = 1)
print(y)
print("\n-------------------\n")


print("\n2------------------\n")
# The position of diagonal elements will be shifted downwards.
y = np.eye(5, dtype = np.uint8, k = -1)
print(y)
print("\n-------------------\n")
```

```
1------------------

[[0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]
 [0 0 0 0 0]]

-------------------


2------------------

[[0 0 0 0 0]
 [1 0 0 0 0]
 [0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 1 0]]

-------------------
```

3) **identity().**

Returns the identity array.

In [6]:
```
x = np.identity(5, dtype = np.uint8)
print(x)
print("\n------------------")
```

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

------------------
```

4) **ones().**

Returns a new array of given shape and type, filled with ones.

In [7]:
```
x = np.ones((2, 2, 2), dtype = np.int16)
print(x)
print("\n------------------")
```

```
[[[1 1]
  [1 1]]

 [[1 1]
  [1 1]]]

------------------
```

5) **zeros().**

Returns a new array of given shape and type, filled with zeros.

In [8]:
```
x = np.zeros((2, 2, 2), dtype = np.int16)
print(x)
print("\n------------------")
```

```
[[[0 0]
  [0 0]]

 [[0 0]
  [0 0]]]

------------------
```

6) **full().**

Returns a new array of given shape and type, filled with the given value.

In [10]:
```
x = np.full((3, 3, 3), dtype = np.int16, fill_value = 5)
print(x)
print("\n------------------")
```

```
[[[5 5 5]
  [5 5 5]
  [5 5 5]]

 [[5 5 5]
  [5 5 5]
  [5 5 5]]

 [[5 5 5]
  [5 5 5]
  [5 5 5]]]

------------------
```

7) **tri().**

An array with ones at and below the given diagonal and zeros elsewhere.

In [11]:
```
x = np.tri(3, 3, k = 0, dtype = np.uint16)
print(x)
print("\n------------------")
```

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

------------------
```

The tri() function may take a third argument as an optional argument, which is k. This argument shifts the main diagonal upwards or downwards according to the value of itself.

In [13]:
```
# The position of diagonal elements will be shifted upwards.
print("\n1------------------\n")
x = np.tri(3, 3, k = 1, dtype = np.uint16)
print(x)
print("\n-------------------\n")

# The position of diagonal elements will be shifted downwards.
print("\n2------------------\n")
x = np.tri(3, 3, k = -1, dtype = np.uint16)
print(x)
print("\n-------------------\n")
```

```
1------------------

[[1 1 0]
 [1 1 1]
 [1 1 1]]

-------------------


2------------------

[[0 0 0]
 [1 0 0]
 [1 1 0]]

-------------------
```

8) **tril().**

Lower triangle of an array. Returns a copy of an array with elements above the k-th diagonal zeroed.

In [14]:
```
x = np.ones((5, 5), dtype = np.int8)
y = np.tril(x, k = -1)
print(y)
print("\n-------------------\n")
```

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

-------------------
```

9) **triu().**

Upper triangle of an array. Returns a copy of a matrix with the elements below the k-th diagonal zeroed.

In [15]:
```
x = np.ones((5, 5), dtype = np.int8)
y = np.triu(x, k = 1)
print(y)
```

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

In [ ]: