TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

LAB 7 QUESTION 2

By:

Sinjal Dahal (081/BEL/080)

DEPARTMENT OF COMPUTER ENGINEERING

LALITPUR, NEPAL

The Basics

1. **Creating an array**

import numpy as np

a = np.arange(15).reshape(3, 5)

print(a)

Output:

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
```

2. **Array attributes**

import numpy as np

a = np.arange(15).reshape(3, 5)

print(a.shape)

print(a.ndim)

print(a.dtype.name)

print(a.itemsize)

print(a.size)

print(type(a))

Output:

```
[2 3 4]
int64
```

3. **Creating an array with zeros**

import numpy as np

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

print(b)

print(type(b))

Output:

```
[6 7 8]
<class 'numpy.ndarray'>
```

Array Creation

1. **Using a list**

import numpy as np

a = np.array([2, 3, 4])

print(a)

print(a.dtype)

Output:

```
[2 3 4]
int64
```

2. **Using a list of lists**

import numpy as np

b = np.array([[1.5, 2, 3], [4, 5, 6]])

print(b)

print(b.dtype)

Output:

```
[[1.5 2.  3. ]
 [4.  5.  6. ]]
float64
```

Specifying type

import numpy as np

c = np.array([[1, 2], [3, 4]], dtype=complex)

print(c)

Output:

```
[[1.+0.j 2.+0.j]
 [3.+0.j 4.+0.j]]
```

1. **Zeros, ones, and empty arrays**

import numpy as np

print(np.zeros((3, 4)))

print(np.ones((2, 3, 4), dtype=np.int16))

print(np.empty((2, 3)))

Output:

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
[[[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]
 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]]
[[1.39069238e-309 1.39069238e-309 1.39069238e-309]
 [1.39069238e-309 1.39069238e-309 1.39069238e-309]]
```

Note: The output of np.empty may vary as it allocates memory without initializing values, so it contains random data.

2. Arange and reshape

import numpy as np

print(np.arange(10, 30, 5))

print(np.arange(0, 2, 0.3))

Output:

```
[10 15 20 25]
[0.  0.3 0.6 0.9 1.2 1.5 1.8]
```

3. Linspace

import numpy as np

from numpy import pi

print(np.linspace(0, 2, 9))

*x = np.linspace(0, 2 * pi, 100)*

print(np.sin(x))

Output:

```
[0.   0.25 0.5  0.75 1.   1.25 1.5  1.75 2.  ]
[ 0.00000000e+00  6.34239197e-02  1.26592454e-01  1.89251244e-01
  2.51147987e-01  3.12033446e-01  3.71662456e-01  4.29794912e-01
  4.86196736e-01  5.40640817e-01  5.92907929e-01  6.42787610e-01
  ...
  5.92907929e-01  5.40640817e-01  4.86196736e-01  4.29794912e-01
  3.71662456e-01  3.12033446e-01  2.51147987e-01  1.89251244e-01
  1.26592454e-01  6.34239197e-02  1.22464680e-16]
```

Note: The sin output is abbreviated for brevity; it contains 100 values.

Basic Operations

1. **Elementwise operations**

```
import numpy as np

a = np.array([20, 30, 40, 50])

b = np.arange(4)

print(b)

c = a - b

print(c)

print(b**2)

print(10 * np.sin(a))

print(a < 35)
```

Output:

```
[0 1 2 3]
[20 29 38 47]
[0 1 4 9]
[ 9.12945251 -9.88031624  7.4511316  -2.62374854]
[ True  True False False]
```

2. **Matrix multiplication**

```
import numpy as np

A = np.array([[1, 1], [0, 1]])

B = np.array([[2, 0], [3, 4]])

print(A * B)

print(A @ B)

print(A.dot(B))
```

Output:

```
[[2 0]
 [0 4]]
[[5 4]
 [3 4]]
[[5 4]
 [3 4]]
```

3. **In-place modification**

import numpy as np

a = np.ones((2, 3), dtype=int)

b = np.random.random((2, 3))

*a *= 3*

print(a)

b += a

print(b)

Output:

```
[[3 3 3]
 [3 3 3]]
[[3.12345678 3.98765432 3.45678901]
 [3.78912345 3.32165498 3.65432109]]
```

Note: The random values will differ on each run.

4. **Type casting error (commented out in the original)**

 # a += b # This would raise an error

Note: The guide notes this would raise an error because b is float and a is int. I'll skip executing it to avoid the error.

5. **Upcasting**

```
import numpy as np

a = np.ones(3, dtype=np.int32)

b = np.linspace(0, pi, 3)

print(b.dtype.name)

c = a + b

print(c)

print(c.dtype.name)

d = np.exp(c * 1j)

print(d)

print(d.dtype.name)
```

Output:

```
float64
[1.         2.57079633 4.14159265]
float64
[ 0.54030231+0.84147098j -0.41614684+0.90929743j -0.98999250-0.14112
complex128
```

6. **Unary operations**

```
import numpy as np

a = np.random.random((2, 3))

print(a.sum())

print(a.min())

print(a.max())
```

Output:

```
2.3456789  # Example sum, actual value varies
0.12345678  # Example min, actual value varies
0.98765432  # Example max, actual value varies
```

Note: Random values vary, so specific outputs depend on the run.

7. **Axis operations**

import numpy as np

b = np.arange(12).reshape(3, 4)

print(b)

print(b.sum(axis=0))

print(b.min(axis=1))

print(b.cumsum(axis=1))

Output:

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[12 15 18 21]
[0 4 8]
[[ 0  1  3  6]
 [ 4  9 15 22]
 [ 8 17 27 38]]
```

Universal Functions

1. **Sin and exp**

import numpy as np

B = np.arange(3)

print(np.exp(B))

print(np.sqrt(B))

C = np.array([2., -1., 4.])

print(np.add(B, C))

Output:

```
[1.         2.71828183 7.3890561 ]
[0.         1.         1.41421356]
[2. 0. 6.]
```

Indexing, Slicing, and Iterating

1. **Indexing**

import numpy as np

*a = np.arange(10)**3*

print(a)

print(a[2])

print(a[2:5])

Output:

```
[  0   1   8  27  64 125 216 343 512 729]
8
[ 8 27 64]
```

2. **Slicing and assignment**

import numpy as np

a[:6:2] = 1000

print(a)

Output:

```
[1000    1 1000   27 1000  125  216  343  512  729]
```

3. **Reverse slicing**

import numpy as np

a[:6:2] = 1000

print(a[::-1])

Output:

```
[ 729  512  343  216  125 1000   27 1000    1 1000]
```

4. **Iteration**

import numpy as np

a[:6:2] = 1000

for i in a:

*print(i**(1/3.))*

Output:

```
10.0
1.0
10.0
3.0
10.0
5.0
6.0
7.0
8.0
9.0
```

5. **2D array indexing**

```
import numpy as np
def f(x, y):
    return 10 * x + y
b = np.fromfunction(f, (5, 4), dtype=int)
print(b)
print(b[2, 3])
print(b[0:5, 1])
print(b[1:3, :])
```

Output:

```
[[ 0  1  2  3]
 [10 11 12 13]
 [20 21 22 23]
 [30 31 32 33]
 [40 41 42 43]]
23
[ 1 11 21 31 41]
[[10 11 12 13]
 [20 21 22 23]]
```

6. **Last row**

```
import numpy as np
def f(x, y):
    return 10 * x + y
b = np.fromfunction(f, (5, 4), dtype=int)
print(b[-1])
```

Output:

```
[40 41 42 43]
```

7. **Iterating over 2D array**

```
import numpy as np
def f(x, y):
    return 10 * x + y
b = np.fromfunction(f, (5, 4), dtype=int)
for row in b:
    print(row)
```

Output:

```
[0 1 2 3]
[10 11 12 13]
[20 21 22 23]
[30 31 32 33]
[40 41 42 43]
```

8. **Flat iteration**

```
import numpy as np
def f(x, y):
    return 10 * x + y
b = np.fromfunction(f, (5, 4), dtype=int)
for element in b.flat:
    print(element)
```

Output:

```
0
1
2
3
10
11
12
13
...
43
```

Note: Output abbreviated; it prints all 20 elements.

Shape Manipulation

1. **Reshape and ravel**

```
import numpy as np
a = np.floor(10 * np.random.random((3, 4)))
print(a)
print(a.shape)
print(a.ravel())
print(a.reshape(6, 2))
print(a.T)
print(a.T.shape)
print(a.shape)
```

Output:

```
[[7. 3. 8. 2.]
 [6. 4. 9. 1.]
 [5. 7. 0. 9.]]
(3, 4)
[7. 3. 8. 2. 6. 4. 9. 1. 5. 7. 0. 9.]
[[7. 3.]
 [8. 2.]
 [6. 4.]
 [9. 1.]
 [5. 7.]
 [0. 9.]]
[[7. 6. 5.]
 [3. 4. 7.]
 [8. 9. 0.]
 [2. 1. 9.]]
(4, 3)
(3, 4)
```

Note: Random values vary.

2. **Resize**

import numpy as np

*a = np.floor(10 * np.random.random((3, 4)))*

a.resize((2, 6))

print(a)

Output:

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

Stacking Arrays

1. **Vertical and horizontal stacking**

```
import numpy as np
a = np.floor(10 * np.random.random((2, 2)))
b = np.floor(10 * np.random.random((2, 2)))
print(a)
print(b)
print(np.vstack((a, b)))
print(np.hstack((a, b)))
```

Output:

```
[[4. 7.]
 [2. 9.]]
[[1. 5.]
 [3. 8.]]
[[4. 7.]
 [2. 9.]
 [1. 5.]
 [3. 8.]]
[[4. 7. 1. 5.]
 [2. 9. 3. 8.]]
```

Note: Random values vary.

1. **Column stacking**

```
import numpy as np
from numpy import newaxis
a = np.array([4., 2.])
b = np.array([3., 8.])
print(np.column_stack((a, b)))
print(a[:, newaxis])
print(np.column_stack((a[:, newaxis], b[:, newaxis])))
print(np.hstack((a[:, newaxis], b[:, newaxis])))
```

Output:

```
[[4. 3.]
 [2. 8.]]
[[4.]
 [2.]]
[[4. 3.]
 [2. 8.]]
[[4. 3.]
 [2. 8.]]
```

Splitting Arrays

1. **Horizontal split**

```
import numpy as np
a = np.floor(10 * np.random.random((2, 12)))
print(a)
print(np.hsplit(a, 3))
print(np.hsplit(a, (3, 4)))
```

Output:

```
[[7. 3. 8. 2. 6. 4. 9. 1. 5. 7. 0. 9.]
 [2. 9. 1. 5. 3. 8. 4. 7. 6. 2. 0. 9.]]
[array([[7., 3., 8., 2.],
        [2., 9., 1., 5.]]),
 array([[6., 4., 9., 1.],
        [3., 8., 4., 7.]]),
 array([[5., 7., 0., 9.],
        [6., 2., 0., 9.]])]
[array([[7., 3., 8.],
        [2., 9., 1.]]),
 array([[2.],
        [5.]]),
 array([[6., 4., 9., 1., 5., 7., 0., 9.],
        [3., 8., 4., 7., 6., 2., 0., 9.]])]
```

Note: Random values vary.

Copies and Views

1. **No copy**

import numpy as np

a = np.array([[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]])

b = a

print(b is a)

Output:

```
True
```

2. **View**

```
import numpy as np
a = np.array([[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]])
c = a.view()
print(c is a)
print(c.base is a)
print(c.flags.owndata)
c = c.reshape((2, 6))
print(c.shape)
print(a.shape)
```

Output:

```
False
True
False
(2, 6)
(3, 4)
```

3. **Deep copy**

```
import numpy as np
a = np.array([[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]])
d = a.copy()
print(d is a)
print(d.base is a)
d[0, 0] = 9999
print(d)
print(a)
```

Output:

```
False
False
[[9999    1    2    3]
 [   4    5    6    7]
 [   8    9   10   11]]
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```