# Module 02 – Exercise

# Numpy

Nguyen Quoc Thai

# Objectives

## Basic Numpy

- ❖ Array Creation
- ❖ Array Attributes
- ❖ Indexing
- ❖ Slicing
- ❖ Reshaping
- ❖ Operations

## Applications

- ❖ Convert Color Image to Grayscale
- ❖ Tabular Data Analysis

# Outline

SECTION 1

## Numpy Review

SECTION 2

## Image Processing

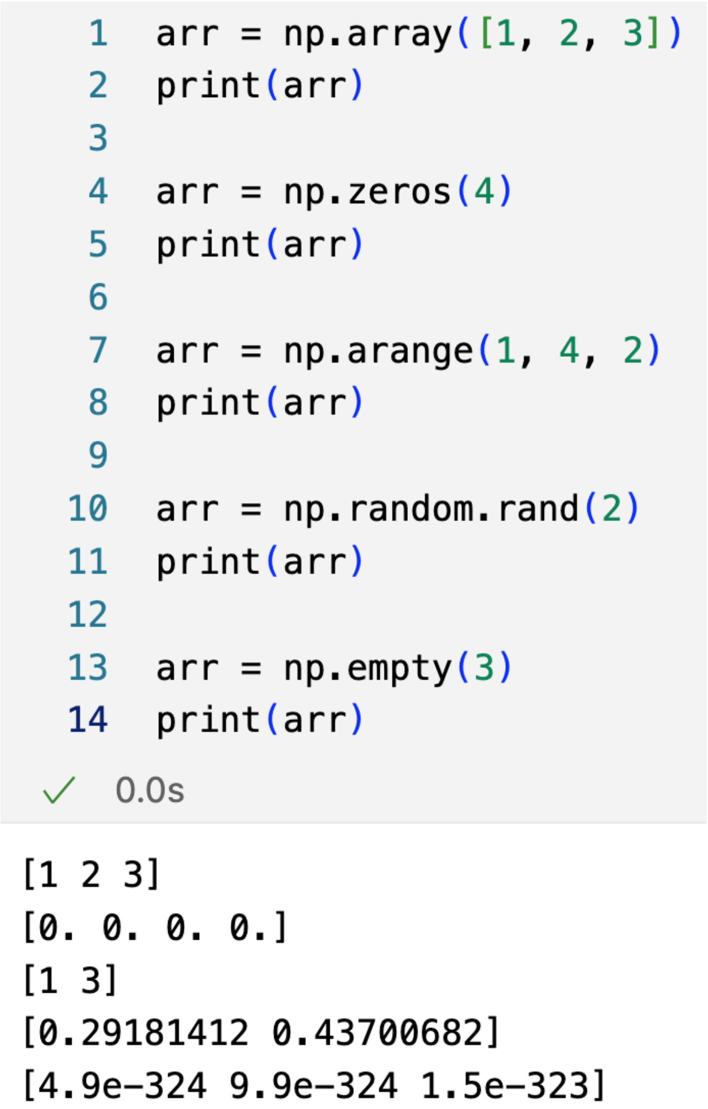
SECTION 3

## Tabular Data Analysis

# Numpy Review

## Numpy Array Creation

```
np.array(python_list)
np.zeros(n)- np.ones(n)
np.arange(min, max, step)
np.random.rand(n)
np.empty(n)
```



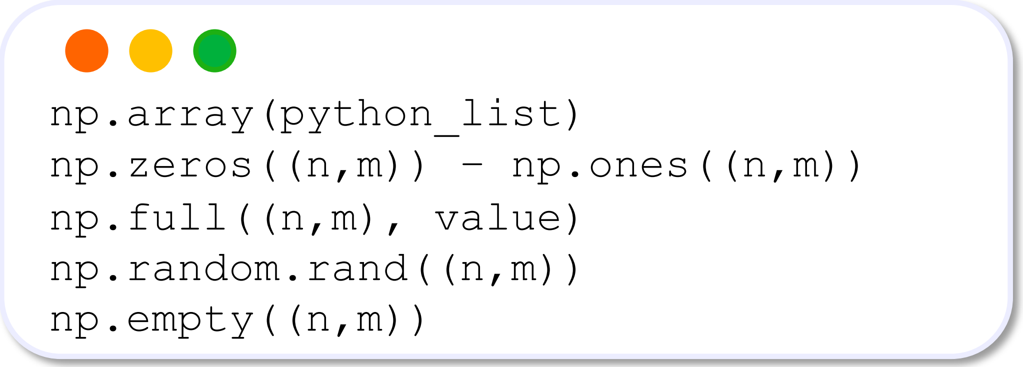
```
arr = np.array([1, 2, 3])
print(arr)

arr = np.zeros(4)
print(arr)

arr = np.arange(1, 4, 2)
print(arr)

arr = np.random.rand(2)
print(arr)

arr = np.empty(3)
print(arr)
```

✓ 0.0s

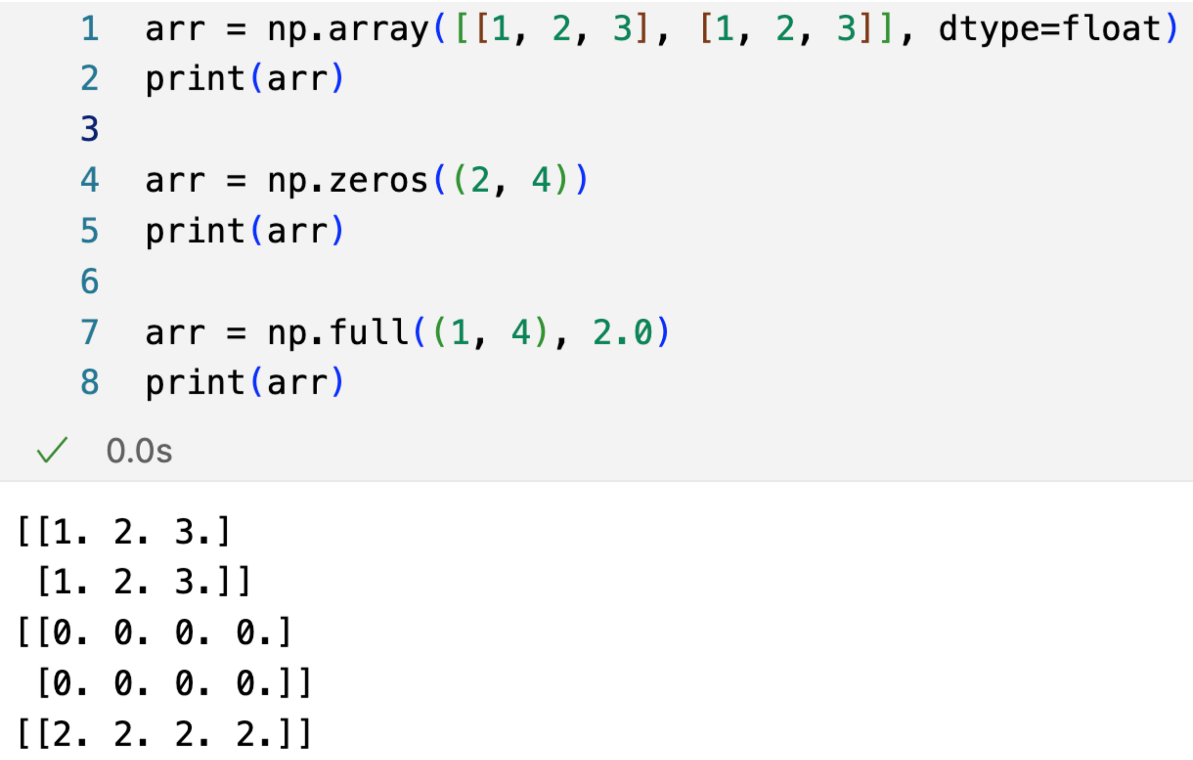
```
[1 2 3]
[0. 0. 0. 0.]
[1 3]
[0.29181412 0.43700682]
[4.9e-324 9.9e-324 1.5e-323]
```

# Numpy Review

## Numpy N-D Array Creation



```
np.array(python_list)
np.zeros((n,m)) – np.ones((n,m))
np.full((n,m), value)
np.random.rand((n,m))
np.empty((n,m))
```



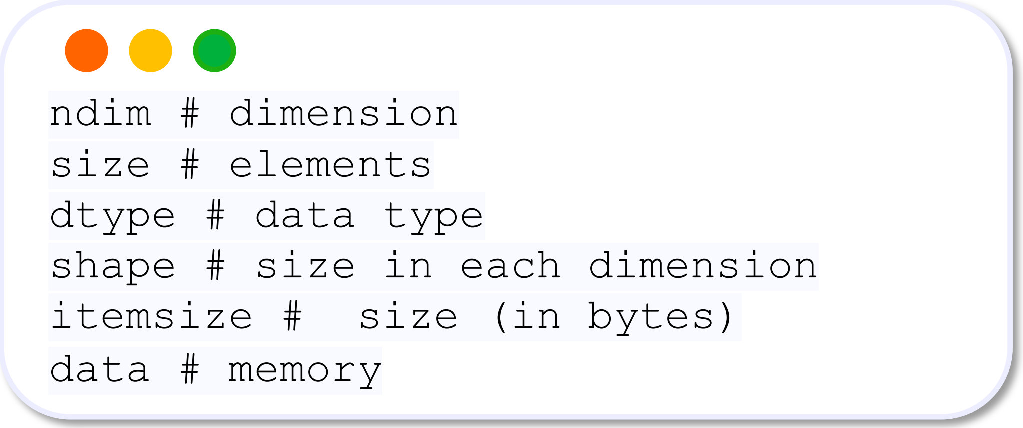
```python
arr = np.array([[1, 2, 3], [1, 2, 3]], dtype=float)
print(arr)

arr = np.zeros((2, 4))
print(arr)

arr = np.full((1, 4), 2.0)
print(arr)
```

✓ 0.0s

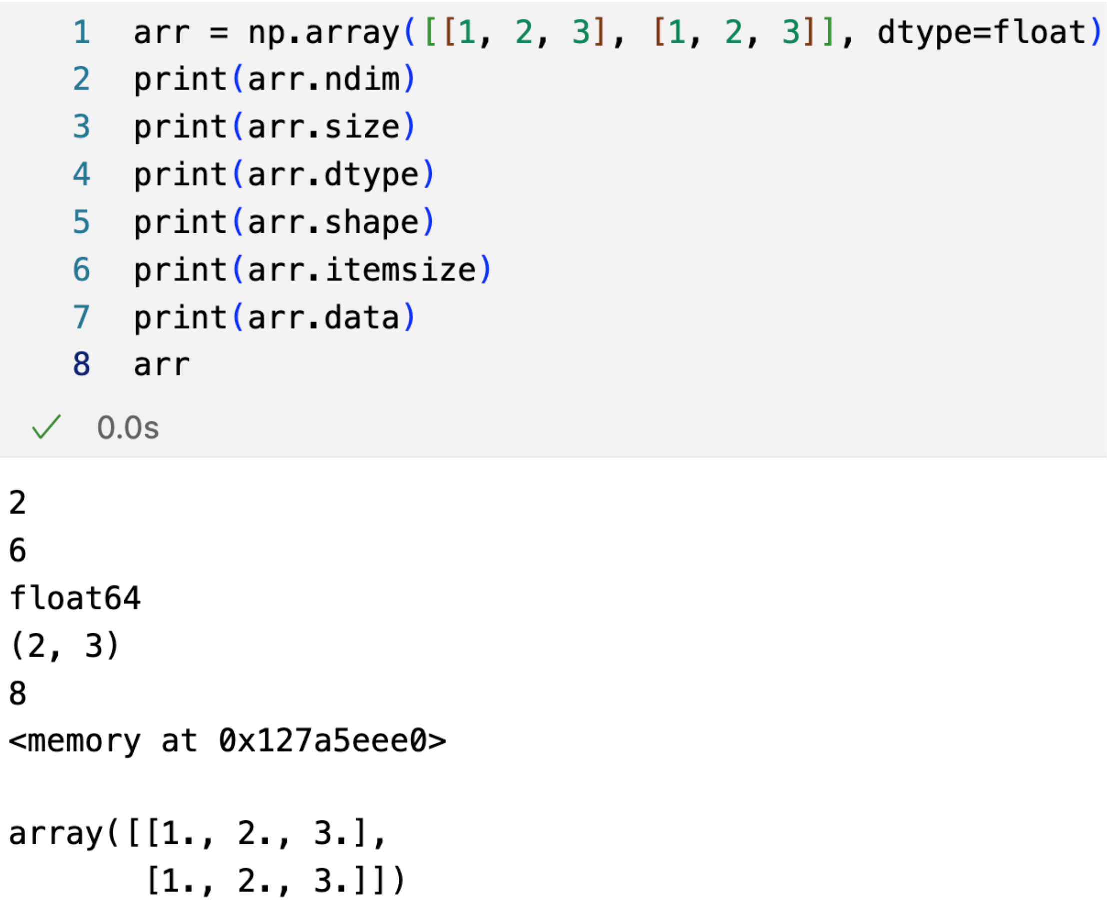
```
[[1. 2. 3.]
 [1. 2. 3.]]
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]]
[[2. 2. 2. 2.]]
```

# Numpy Review

## Numpy Array Attributes



```
ndim # dimension
size # elements
dtype # data type
shape # size in each dimension
itemsize #  size (in bytes)
data # memory
```



```
arr = np.array([[1, 2, 3], [1, 2, 3]], dtype=float)
print(arr.ndim)
print(arr.size)
print(arr.dtype)
print(arr.shape)
print(arr.itemsize)
print(arr.data)
arr
```

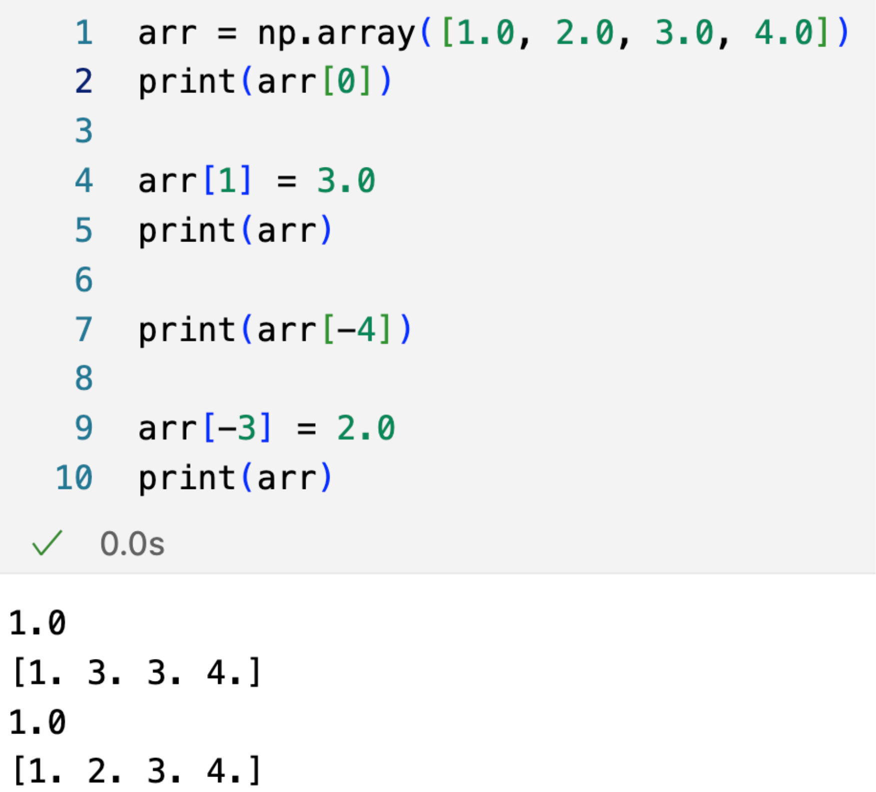
✓ 0.0s

```
2
6
float64
(2, 3)
8
<memory at 0x127a5eee0>

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

# Numpy Review

## Numpy Array Indexing

| | | | | |
|---|---|---|---|---|
| | 1.0 | 2.0 | 3.0 | 4.0 |
| Index | 0 | 1 | 2 | 3 |
| Negative Index | -4 | -3 | -2 | -1 |



```
arr = np.array([1.0, 2.0, 3.0, 4.0])
print(arr[0])

arr[1] = 3.0
print(arr)

print(arr[-4])

arr[-3] = 2.0
print(arr)
```

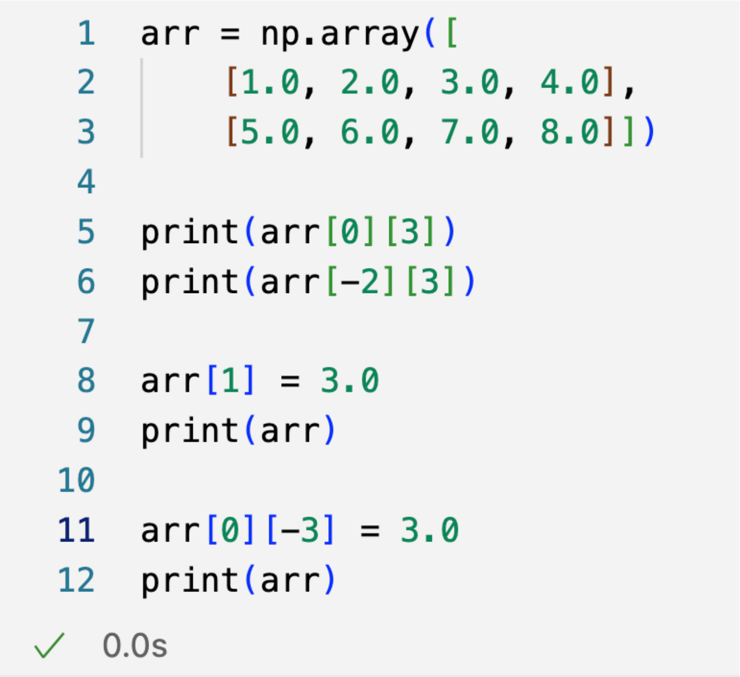
✓ 0.0s

```
1.0
[1. 3. 3. 4.]
1.0
[1. 2. 3. 4.]
```

# Numpy Review

## Numpy Array Indexing

|  |  |  |  |  |  |
|---|---|---|---|---|---|
|  |  | -4 | -3 | -2 | -1 |
|  |  | 0 | 1 | 2 | 3 |
| -2 | 0 | 1.0 | 2.0 | 3.0 | 4.0 |
| -1 | 1 | 5.0 | 6.0 | 7.0 | 8.0 |



```python
arr = np.array([
    [1.0, 2.0, 3.0, 4.0],
    [5.0, 6.0, 7.0, 8.0]])

print(arr[0][3])
print(arr[-2][3])

arr[1] = 3.0
print(arr)

arr[0][-3] = 3.0
print(arr)
```

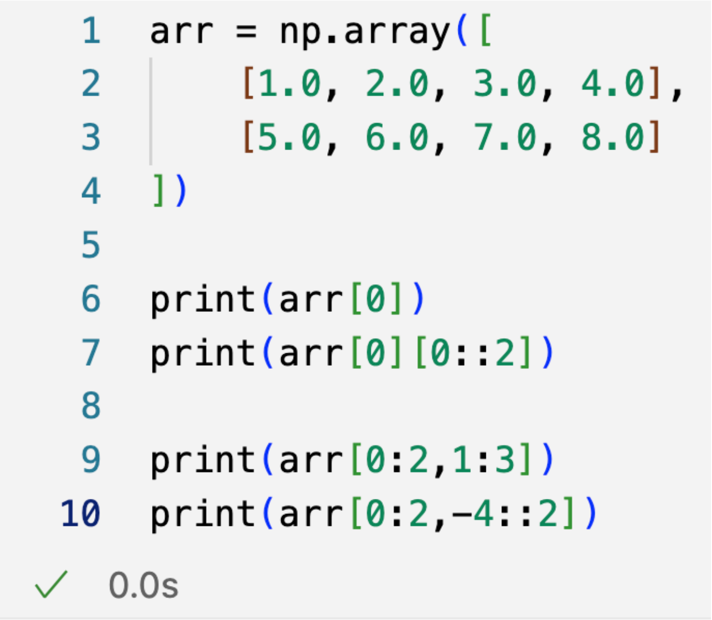
```
4.0
4.0
[[1. 2. 3. 4.]
 [3. 3. 3. 3.]]
[[1. 3. 3. 4.]
 [3. 3. 3. 3.]]
```

# Numpy Review

## Numpy Array Slicing

```
array[start:stop:step]
```

|  |  |  |  |  |  |
|---|---|---|---|---|---|
|  |  | -4 | -3 | -2 | -1 |
|  |  | 0 | 1 | 2 | 3 |
| -2 | 0 | 1.0 | 2.0 | 3.0 | 4.0 |
| -1 | 1 | 5.0 | 6.0 | 7.0 | 8.0 |



```
arr = np.array([
        [1.0, 2.0, 3.0, 4.0],
        [5.0, 6.0, 7.0, 8.0]
])

print(arr[0])
print(arr[0][0::2])

print(arr[0:2,1:3])
print(arr[0:2,-4::2])
```

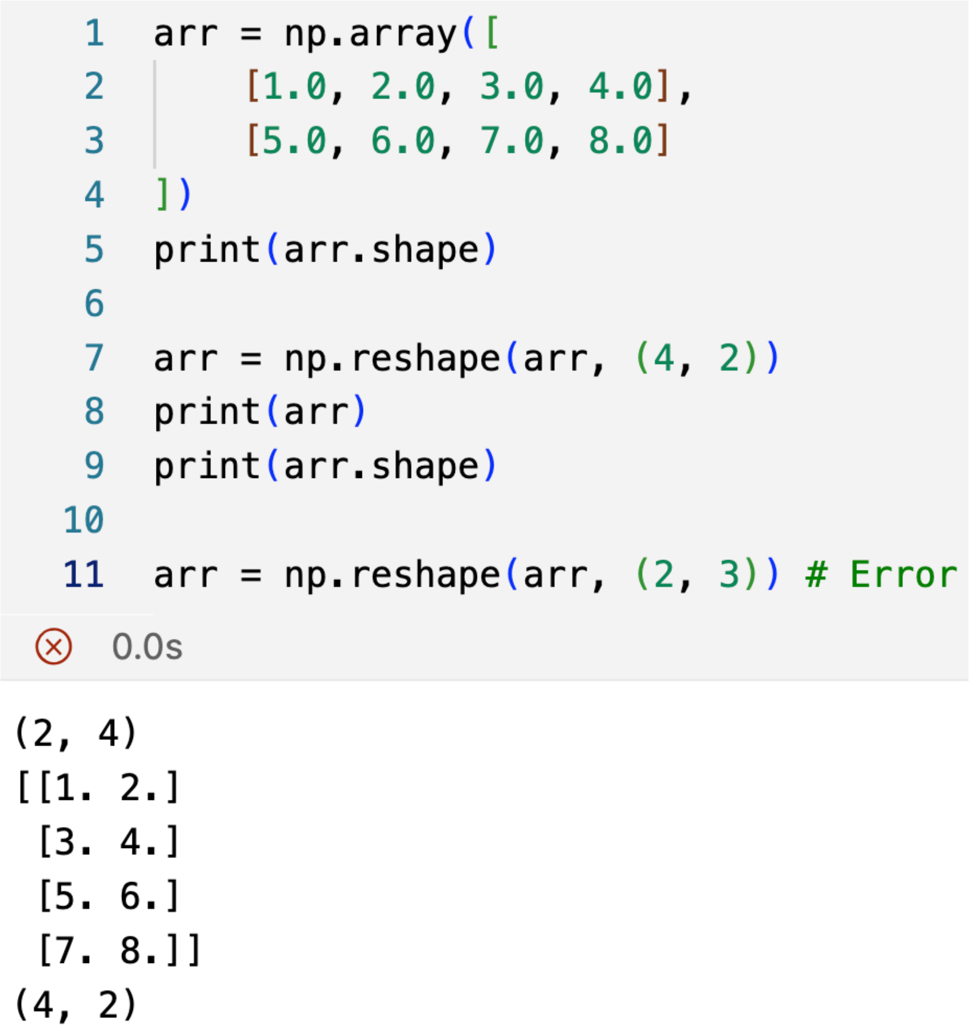
✓ 0.0s

```
[1. 2. 3. 4.]
[1. 3.]
[[2. 3.]
 [6. 7.]]
[[1. 3.]
 [5. 7.]]
```

# Numpy Review

## Numpy Array Reshaping

```
np.reshape(array, newshape)
```



```
arr = np.array([
    [1.0, 2.0, 3.0, 4.0],
    [5.0, 6.0, 7.0, 8.0]
])
print(arr.shape)

arr = np.reshape(arr, (4, 2))
print(arr)
print(arr.shape)

arr = np.reshape(arr, (2, 3)) # Error
```

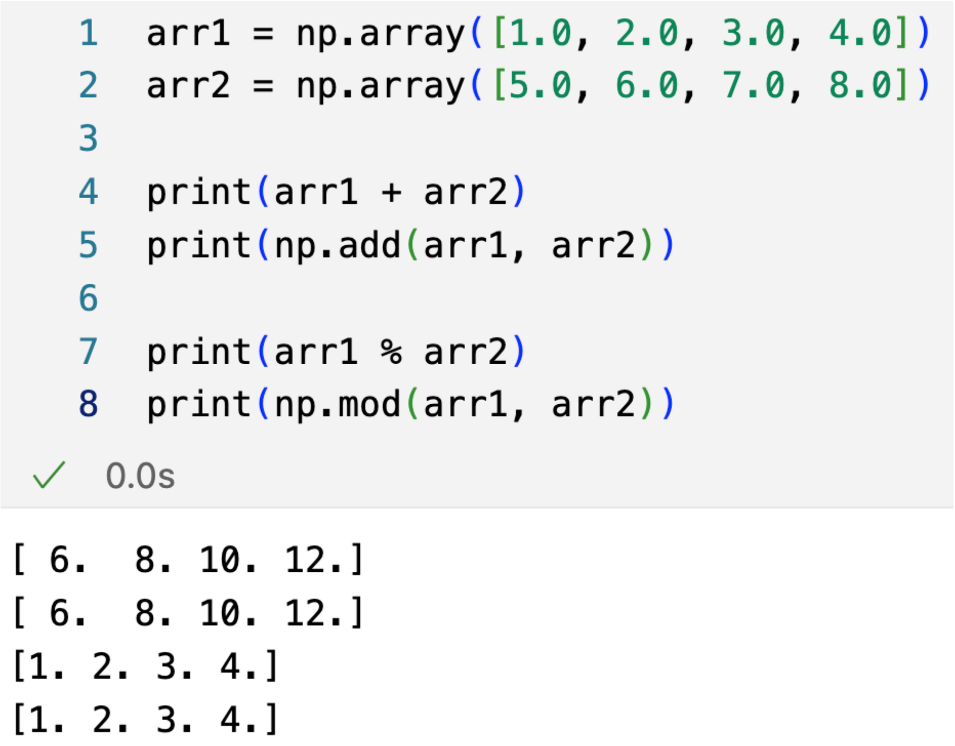
⊗ 0.0s

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

# Numpy Review

## Arithmetic Array Operations

| Element-wise Operation | Operator | Function |
|---|---|---|
| Addition | + | add() |
| Subtraction | - | subtract() |
| Multiplication | * | multiply() |
| Division | / | divide() |
| Exponentiation | ** | power() |
| Modulus | % | mod() |



```
arr1 = np.array([1.0, 2.0, 3.0, 4.0])
arr2 = np.array([5.0, 6.0, 7.0, 8.0])

print(arr1 + arr2)
print(np.add(arr1, arr2))

print(arr1 % arr2)
print(np.mod(arr1, arr2))
```

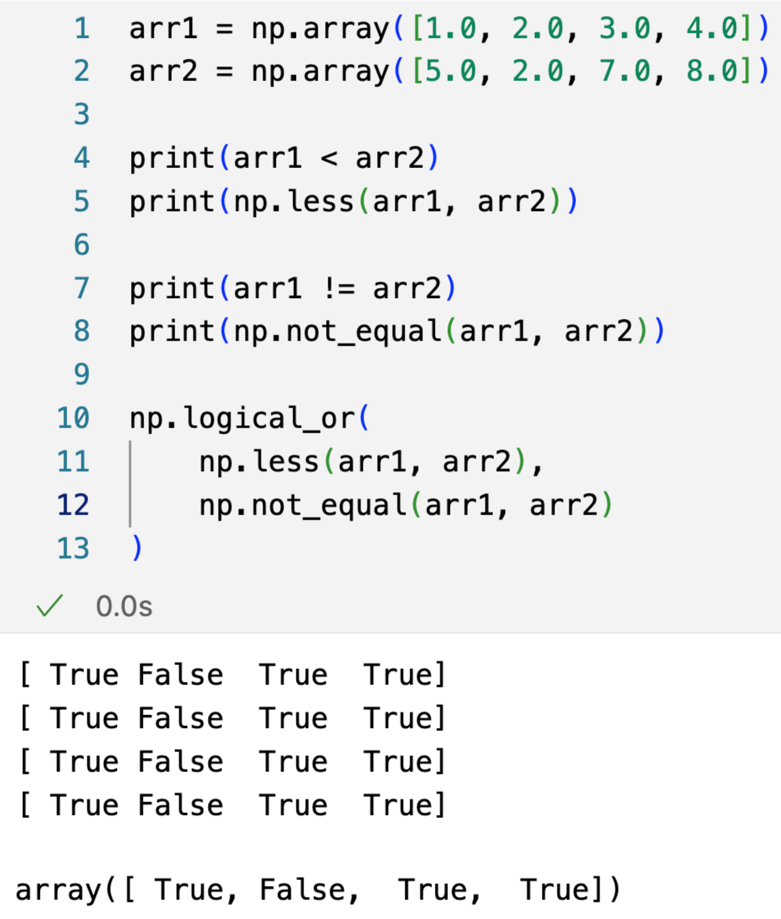
✓ 0.0s

```
[ 6.  8. 10. 12.]
[ 6.  8. 10. 12.]
[1. 2. 3. 4.]
[1. 2. 3. 4.]
```

# Numpy Review

## Arithmetic Array Operations

| Operator | Function |
|---|---|
| < | less() |
| <= | less_equal() |
| > | greater() |
| >= | greater_equal() |
| == | equal() |
| != | not_equal() |
| and | logical_and() |
| or | logical_or() |
| not | logical_not() |
| xor | logical_xor() |



```python
arr1 = np.array([1.0, 2.0, 3.0, 4.0])
arr2 = np.array([5.0, 2.0, 7.0, 8.0])

print(arr1 < arr2)
print(np.less(arr1, arr2))

print(arr1 != arr2)
print(np.not_equal(arr1, arr2))

np.logical_or(
    np.less(arr1, arr2),
    np.not_equal(arr1, arr2)
)
```

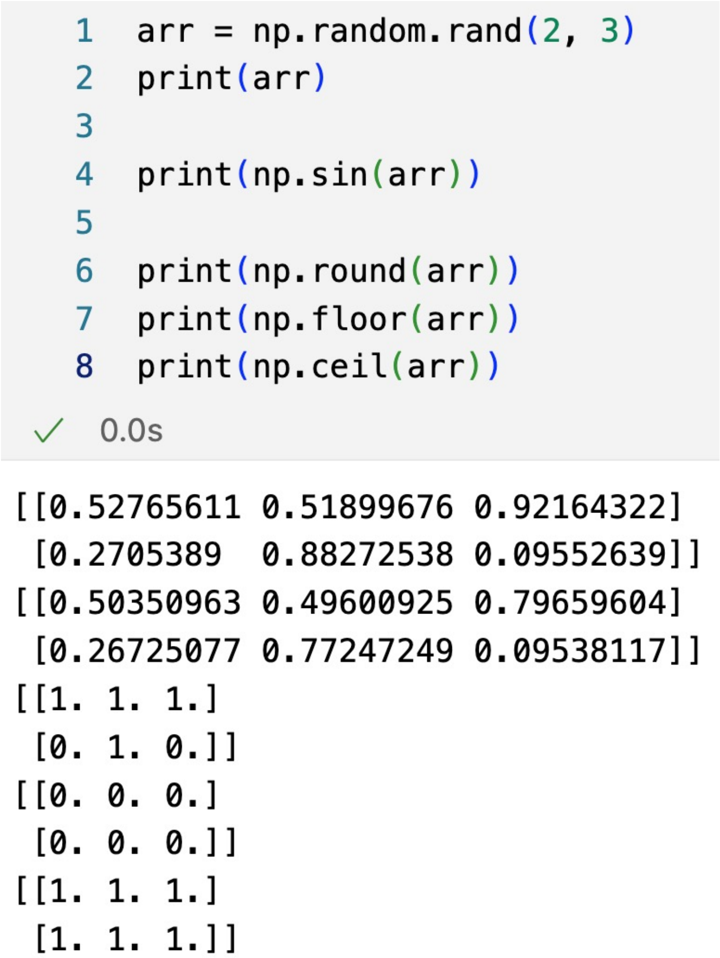
✓ 0.0s

```
[ True False  True  True]
[ True False  True  True]
[ True False  True  True]
[ True False  True  True]

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

# Numpy Review

## Math Functions

| Type | Function |
|---|---|
| Trigonimetric | sin(), cos(), tan(),… |
| Arithmetic | add(), subtract(),… |
| Rounding | round(), floor(), ceil() |



```python
arr = np.random.rand(2, 3)
print(arr)

print(np.sin(arr))

print(np.round(arr))
print(np.floor(arr))
print(np.ceil(arr))
```

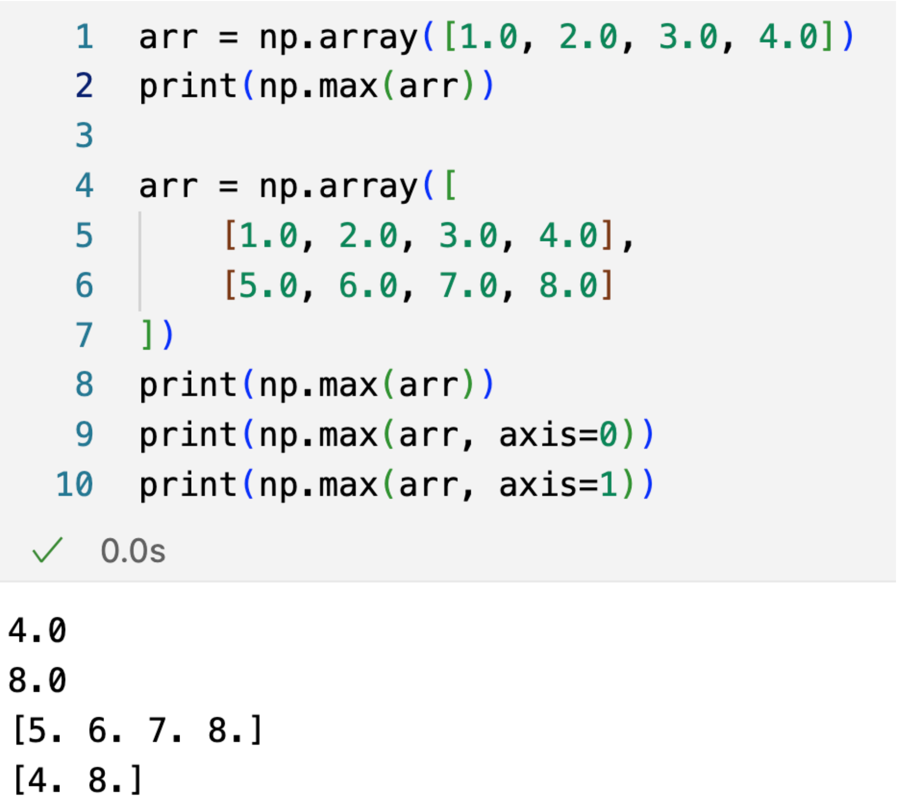
✓ 0.0s

```
[[0.52765611 0.51899676 0.92164322]
 [0.2705389  0.88272538 0.09552639]]
[[0.50350963 0.49600925 0.79659604]
 [0.26725077 0.77247249 0.09538117]]
[[1. 1. 1.]
 [0. 1. 0.]]
[[0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1.]
 [1. 1. 1.]]
```

# Numpy Review

## ! Statistical Functions

| Function | Description |
|---|---|
| median() | return the median of an array |
| mean() | return the mean of an array |
| min() | return the minimum element of an array |
| max() | return the maximum element of an array |



```python
arr = np.array([1.0, 2.0, 3.0, 4.0])
print(np.max(arr))

arr = np.array([
        [1.0, 2.0, 3.0, 4.0],
        [5.0, 6.0, 7.0, 8.0]
])
print(np.max(arr))
print(np.max(arr, axis=0))
print(np.max(arr, axis=1))
```

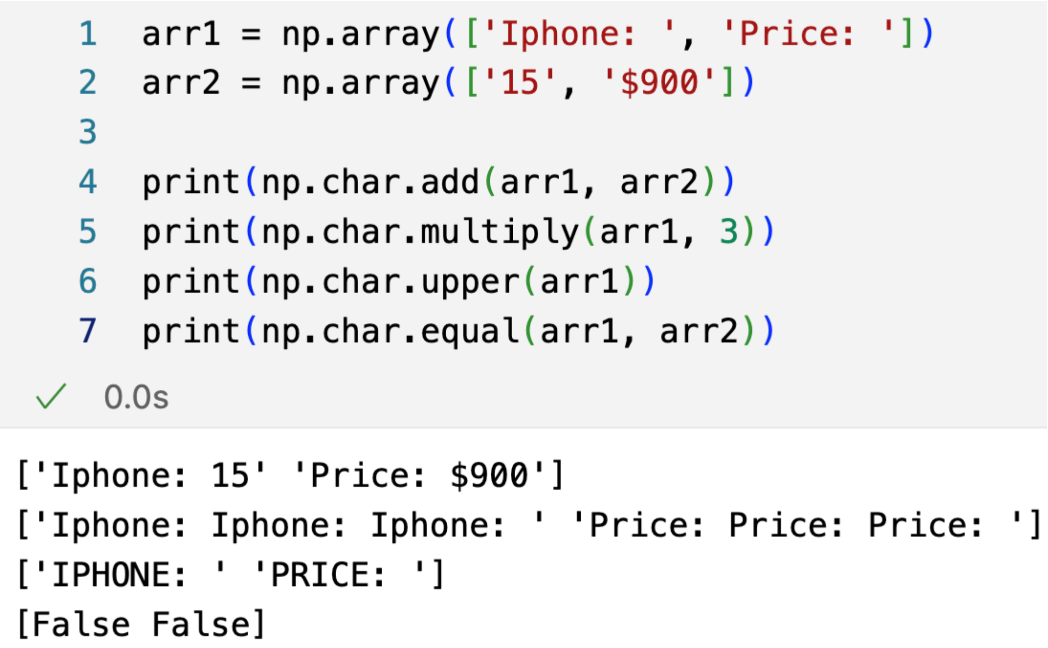
✓ 0.0s

```
4.0
8.0
[5. 6. 7. 8.]
[4. 8.]
```

# Numpy Review

## String Functions

| Function | Description |
|---|---|
| add() | concatenates two strings |
| multiply() | repeats a string for a specified number of times |
| capitalize() | capitalizes the first letter of a string |
| lower() | lowercasing |
| upper() | uppercasing |
| join() | joins a sequence of strings |
| equal() | checks if two strings are equal or not |
| add() | concatenates two strings |



```python
arr1 = np.array(['Iphone: ', 'Price: '])
arr2 = np.array(['15', '$900'])

print(np.char.add(arr1, arr2))
print(np.char.multiply(arr1, 3))
print(np.char.upper(arr1))
print(np.char.equal(arr1, arr2))
```

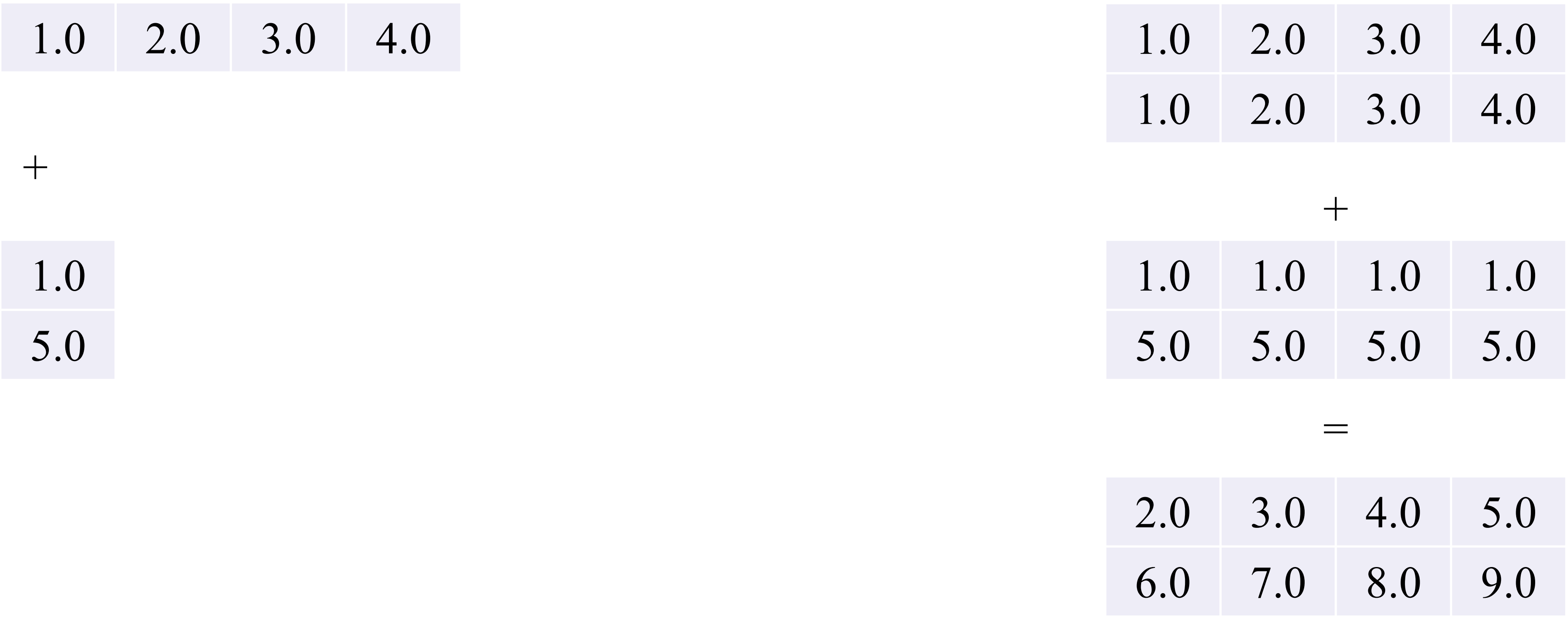
✓ 0.0s

```
['Iphone: 15' 'Price: $900']
['Iphone: Iphone: Iphone: ' 'Price: Price: Price: ']
['IPHONE: ' 'PRICE: ']
[False False]
```

# Numpy Review

## Broadcasting

- Perform mathematical operations on arrays of different shapes
- Small shape => expand to match the large shape



| | | | |
|---|---|---|---|
| 1.0 | 2.0 | 3.0 | 4.0 |

+

| |
|---|
| 1.0 |
| 5.0 |

| | | | |
|---|---|---|---|
| 1.0 | 2.0 | 3.0 | 4.0 |
| 1.0 | 2.0 | 3.0 | 4.0 |

+

| | | | |
|---|---|---|---|
| 1.0 | 1.0 | 1.0 | 1.0 |
| 5.0 | 5.0 | 5.0 | 5.0 |

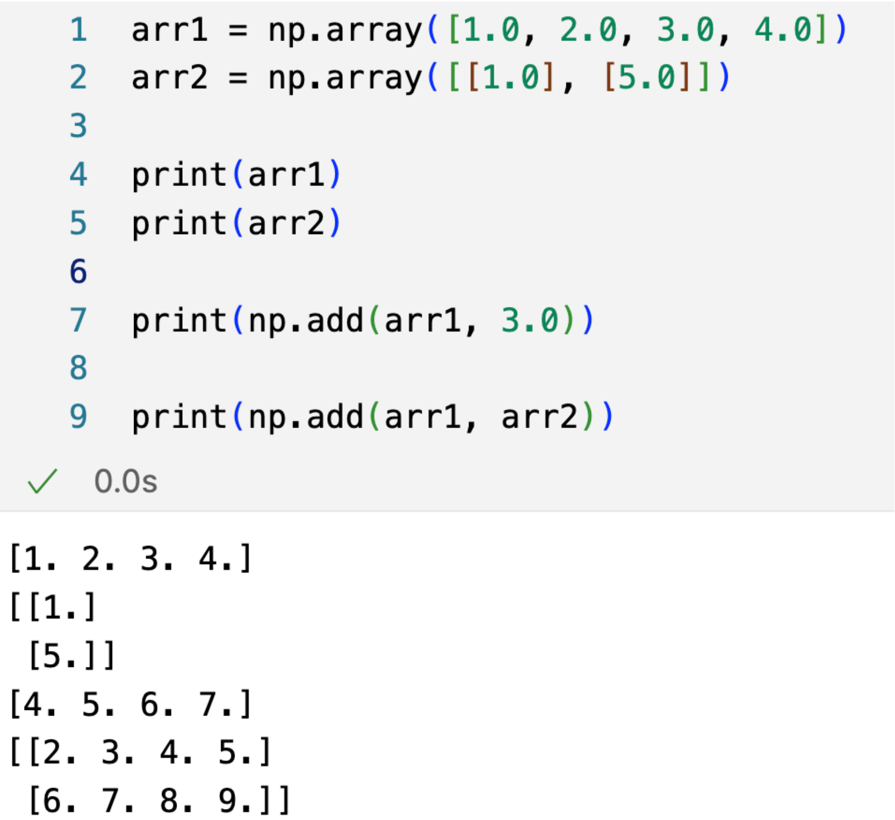
=

| | | | |
|---|---|---|---|
| 2.0 | 3.0 | 4.0 | 5.0 |
| 6.0 | 7.0 | 8.0 | 9.0 |

# Numpy Review

## Broadcasting

- Perform mathematical operations on arrays of different shapes
- Small shape => expand to match the large shape



```
arr1 = np.array([1.0, 2.0, 3.0, 4.0])
arr2 = np.array([[1.0], [5.0]])

print(arr1)
print(arr2)

print(np.add(arr1, 3.0))

print(np.add(arr1, arr2))
```

✓ 0.0s

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

| | | | |
|---|---|---|---|
| 1.0 | 2.0 | 3.0 | 4.0 |
| 1.0 | 2.0 | 3.0 | 4.0 |

+

| | | | |
|---|---|---|---|
| 1.0 | 1.0 | 1.0 | 1.0 |
| 5.0 | 5.0 | 5.0 | 5.0 |

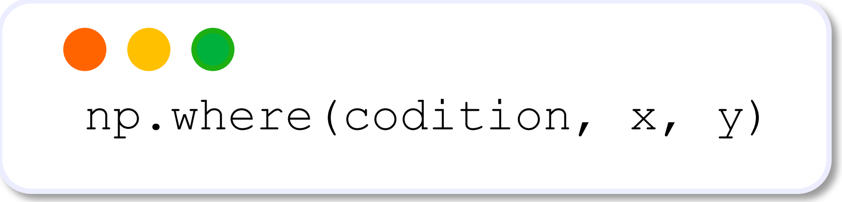
=

| | | | |
|---|---|---|---|
| 2.0 | 3.0 | 4.0 | 5.0 |
| 6.0 | 7.0 | 8.0 | 9.0 |

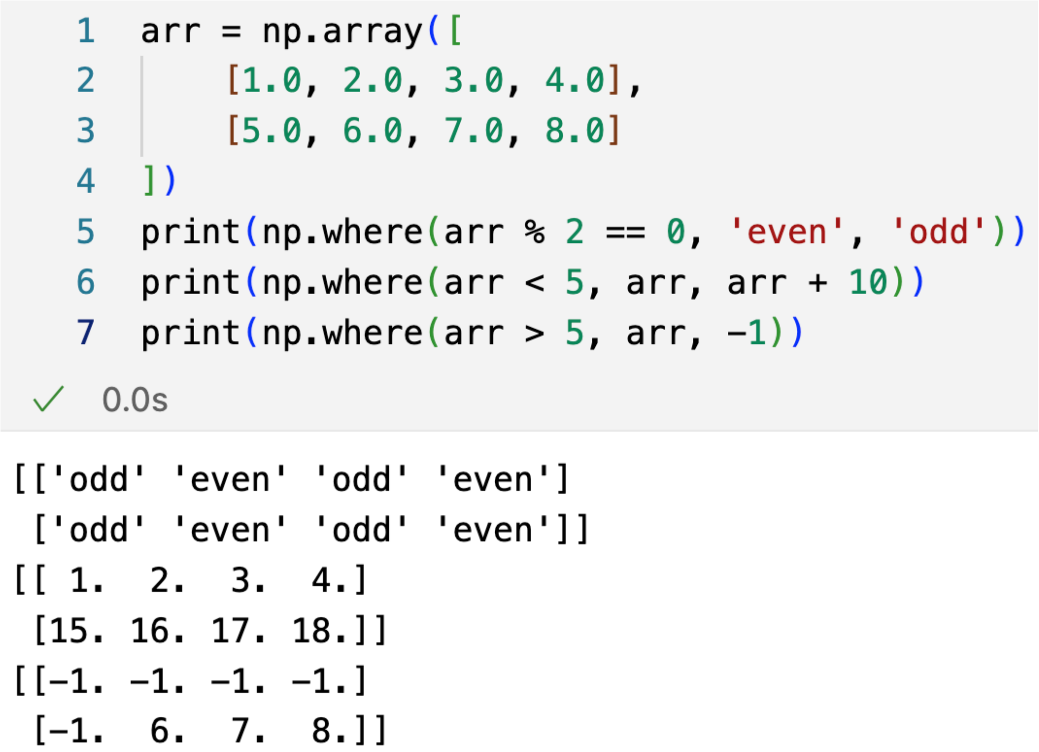
# Numpy Review

## Where Function

- Return elements chosen from x or y depending on condition



```
np.where(codition, x, y)
```



```
arr = np.array([
    [1.0, 2.0, 3.0, 4.0],
    [5.0, 6.0, 7.0, 8.0]
])
print(np.where(arr % 2 == 0, 'even', 'odd'))
print(np.where(arr < 5, arr, arr + 10))
print(np.where(arr > 5, arr, -1))
```

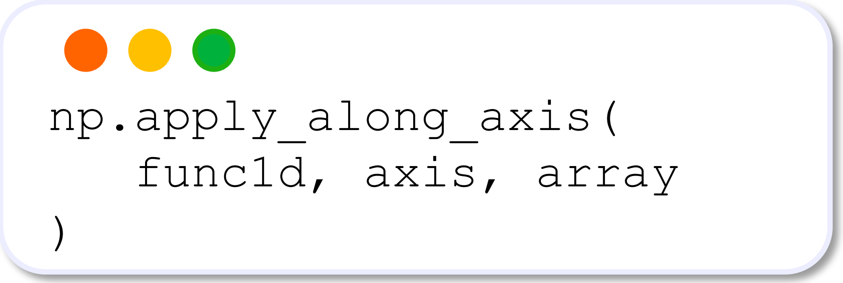
✓ 0.0s

```
[['odd' 'even' 'odd' 'even']
 ['odd' 'even' 'odd' 'even']]
[[ 1.  2.  3.  4.]
 [15. 16. 17. 18.]]
[[-1. -1. -1. -1.]
 [-1.  6.  7.  8.]]
```

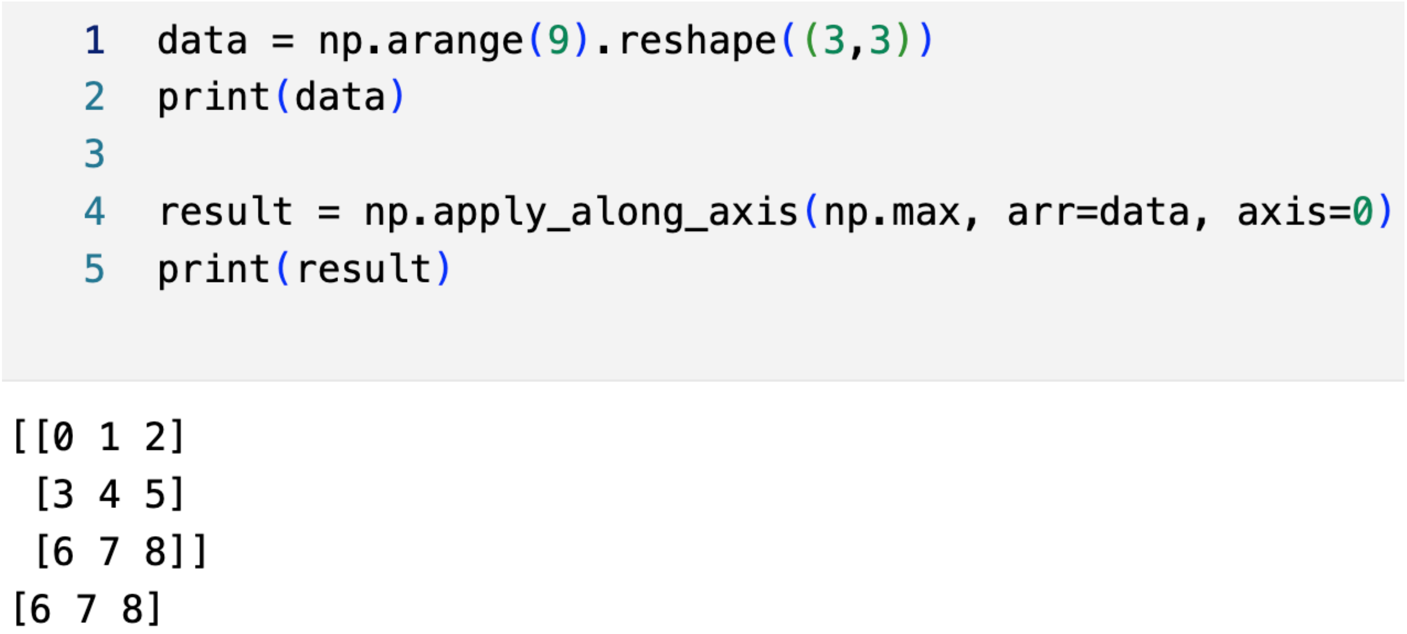
# Numpy Review

## apply_along_axis Function

- Apply a function to 1-D slices along given axis



```
np.apply_along_axis(
    func1d, axis, array
)
```



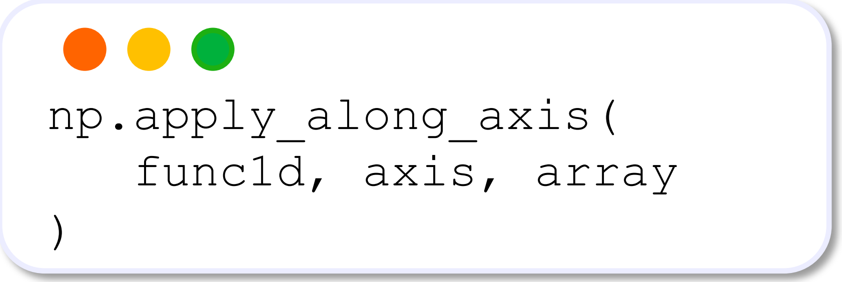
```
data = np.arange(9).reshape((3,3))
print(data)

result = np.apply_along_axis(np.max, arr=data, axis=0)
print(result)
```

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

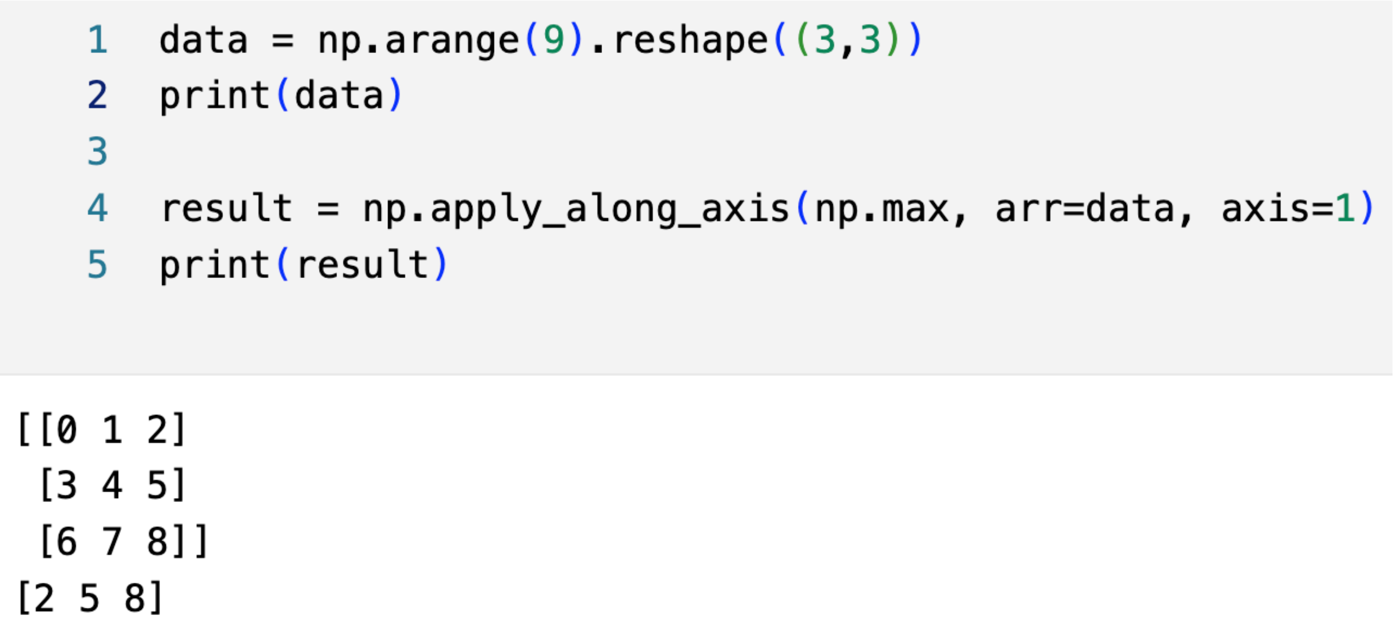
# Numpy Review

## apply_along_axis Function

- Apply a function to 1-D slices along given axis



```
np.apply_along_axis(
    func1d, axis, array
)
```



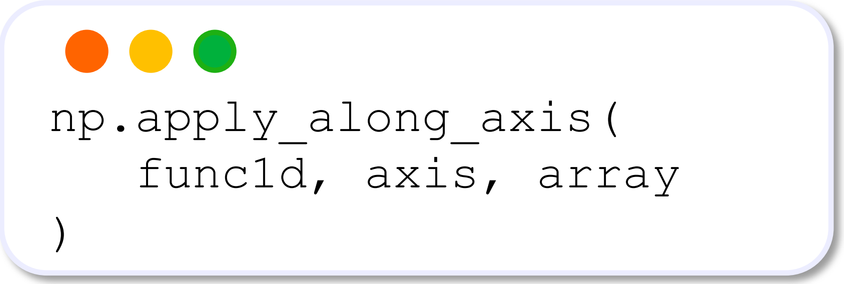
```
data = np.arange(9).reshape((3,3))
print(data)

result = np.apply_along_axis(np.max, arr=data, axis=1)
print(result)
```

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

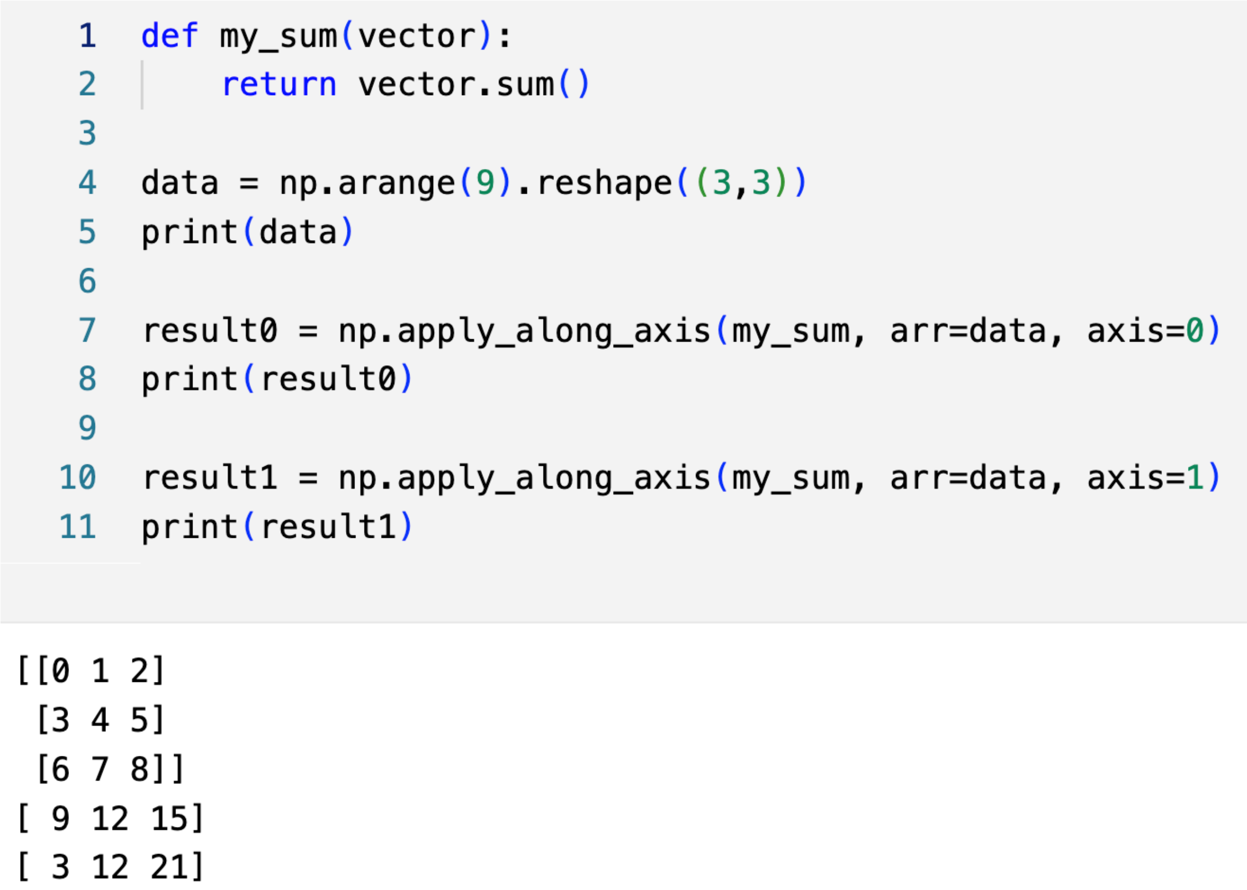
# Numpy Review

## apply_along_axis Function

- Apply a function to 1-D slices along given axis



```
np.apply_along_axis(
    func1d, axis, array
)
```



```python
def my_sum(vector):
    return vector.sum()

data = np.arange(9).reshape((3,3))
print(data)

result0 = np.apply_along_axis(my_sum, arr=data, axis=0)
print(result0)

result1 = np.apply_along_axis(my_sum, arr=data, axis=1)
print(result1)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[ 9 12 15]
[ 3 12 21]
```

# QUIZ TIME

# Outline

SECTION 1

## Numpy Review

SECTION 3

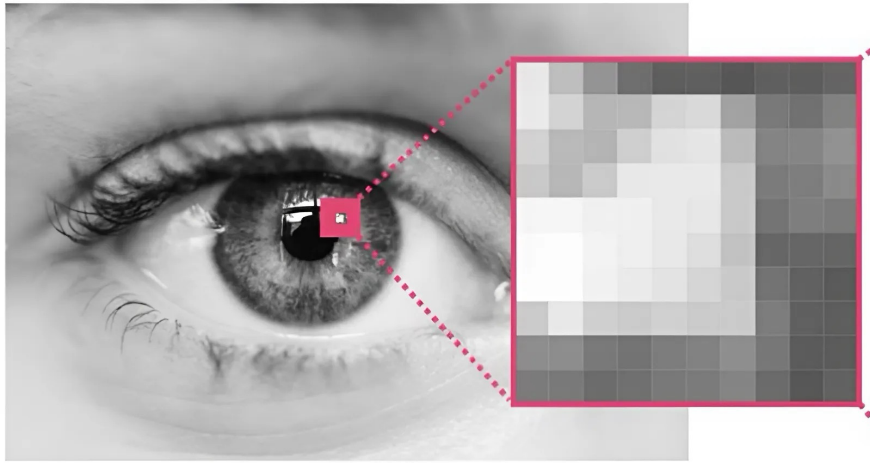
## Tabular Data Analysis

SECTION 2

## Image Processing

# Image Processing

## Grayscale Image



|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| 230 | 194 | 147 | 108 | 90 | 98 | 84 | 96 | 91 | 101 |
| 237 | 206 | 188 | 195 | 207 | 213 | 163 | 123 | 116 | 128 |
| 210 | 183 | 180 | 205 | 224 | 234 | 188 | 122 | 134 | 147 |
| 198 | 189 | 201 | 227 | 229 | 232 | 200 | 125 | 127 | 135 |
| 249 | 241 | 237 | 244 | 232 | 226 | 202 | 116 | 125 | 126 |
| 251 | 254 | 241 | 239 | 230 | 217 | 196 | 102 | 103 | 99 |
| 243 | 255 | 240 | 231 | 227 | 214 | 203 | 116 | 95 | 91 |
| 204 | 231 | 208 | 200 | 207 | 201 | 200 | 121 | 95 | 95 |
| 144 | 140 | 120 | 115 | 125 | 127 | 143 | 118 | 92 | 91 |
| 121 | 121 | 108 | 109 | 122 | 121 | 134 | 106 | 86 | 97 |

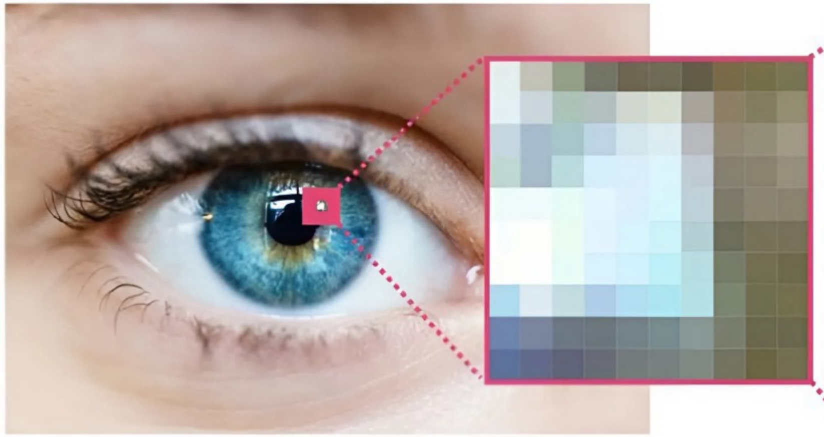
Pixel p = scalar

$$0 \leq p \leq 255$$

AI VIET NAM
@aivietnam.edu.vn

# Image Processing

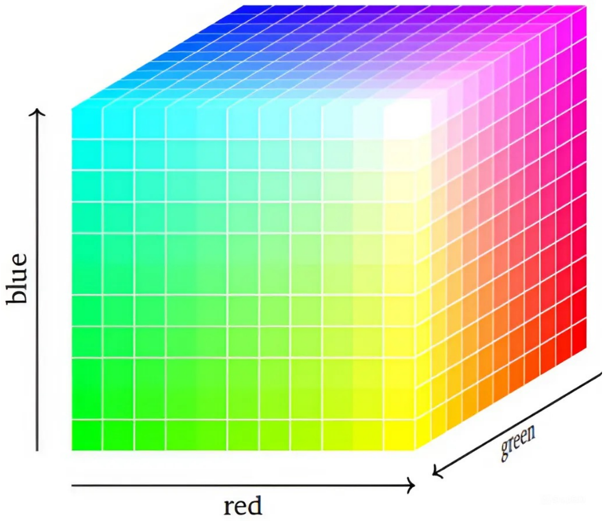
## Color Image



|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| 233 | 188 | 137 | 96 | 90 | 95 | 63 | 73 | 73 | 82 |
|  |  |  |  |  |  |  |  |  | 109 |
|  |  |  |  |  |  |  |  |  | 131 |
|  |  |  |  |  |  |  |  |  | 122 |
|  |  |  |  |  |  |  |  |  | 115 |
|  |  |  |  |  |  |  |  |  | 87 |
|  |  |  |  |  |  |  |  |  | 74 |
|  |  |  |  |  |  |  |  |  | 74 |
|  |  |  |  |  |  |  |  |  | 74 |
|  |  |  |  |  |  |  |  |  | 82 |

|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| 237 | 202 | 159 | 120 | 105 | 110 | 88 | 107 | 112 | 121 |
|  |  |  |  |  |  |  |  |  | 142 |
|  |  |  |  |  |  |  |  |  | 155 |
|  |  |  |  |  |  |  |  |  | 140 |
|  |  |  |  |  |  |  |  |  | 133 |
|  |  |  |  |  |  |  |  |  | 111 |
|  |  |  |  |  |  |  |  |  | 107 |
|  |  |  |  |  |  |  |  |  | 113 |
|  |  |  |  |  |  |  |  |  | 105 |
|  |  |  |  |  |  |  |  |  | 113 |

|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| 226 | 191 | 147 | 110 | 101 | 112 | 98 | 123 | 110 | 119 |
| 221 | 191 | 176 | 182 | 203 | 214 | 169 | 144 | 133 | 145 |
| 185 | 160 | 161 | 184 | 205 | 223 | 186 | 137 | 147 | 161 |
| 181 | 174 | 189 | 207 | 206 | 215 | 194 | 136 | 142 | 151 |
| 246 | 237 | 237 | 231 | 208 | 206 | 192 | 122 | 143 | 144 |
| 254 | 254 | 241 | 224 | 199 | 192 | 181 | 99 | 122 | 117 |
| 239 | 248 | 232 | 207 | 187 | 182 | 184 | 110 | 114 | 110 |
| 193 | 215 | 193 | 167 | 158 | 164 | 181 | 114 | 112 | 111 |
| 113 | 119 | 110 | 111 | 113 | 123 | 135 | 120 | 108 | 106 |
| 93 | 97 | 91 | 103 | 107 | 111 | 122 | 112 | 104 | 114 |



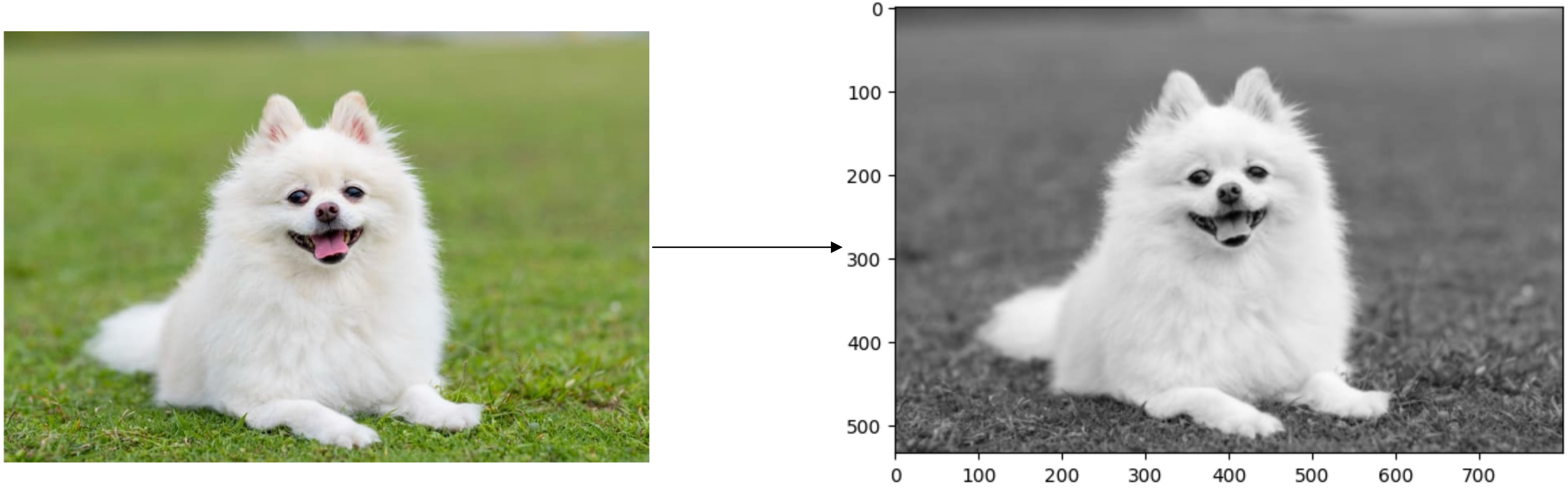
(Height, Width, Channel)

$$\text{Pixel p} = \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

$$0 \leq r, g, b \leq 255$$

# Image Processing

## ! Convert Color Image to Grayscale Image



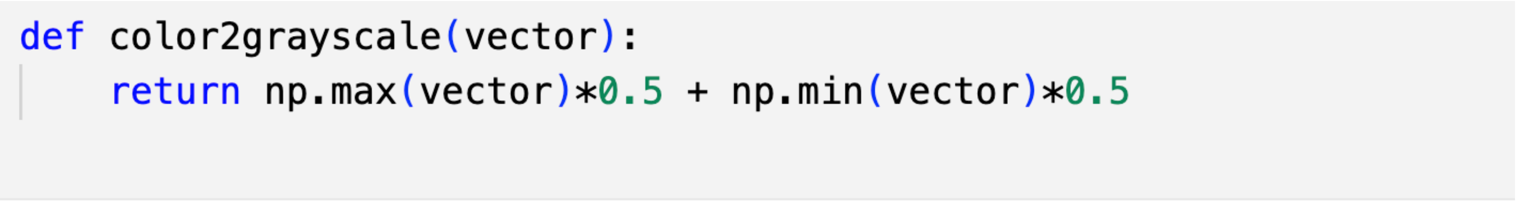
# Image Processing

## Convert Color Image to Grayscale Image

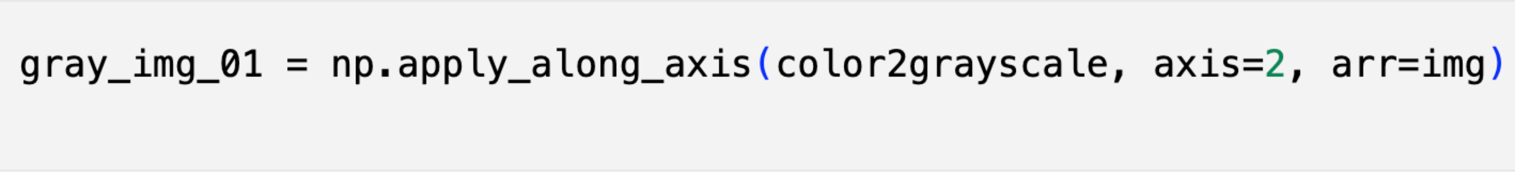
- The lightness method

Average the most prominent and least prominent

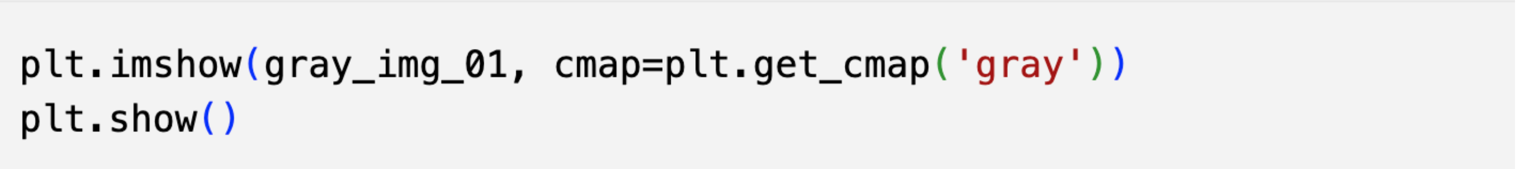
$$\frac{\max(R, G, B) + \min(R, G, B)}{2}$$



```python
def color2grayscale(vector):
    return np.max(vector)*0.5 + np.min(vector)*0.5
```



```python
gray_img_01 = np.apply_along_axis(color2grayscale, axis=2, arr=img)
```



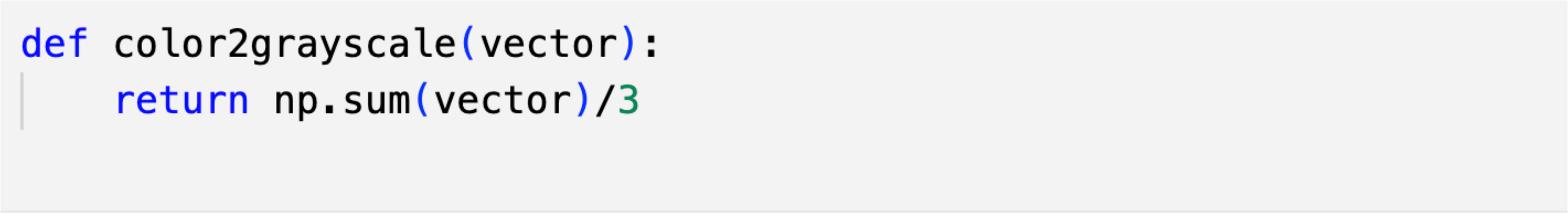
```python
plt.imshow(gray_img_01, cmap=plt.get_cmap('gray'))
plt.show()
```

# Image Processing

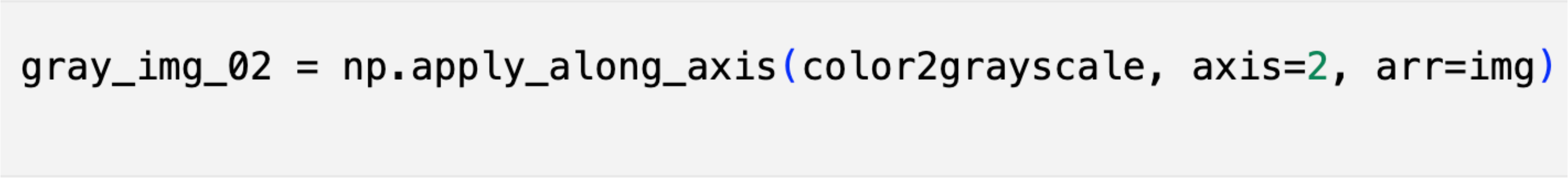
## Convert Color Image to Grayscale Image

- The average method

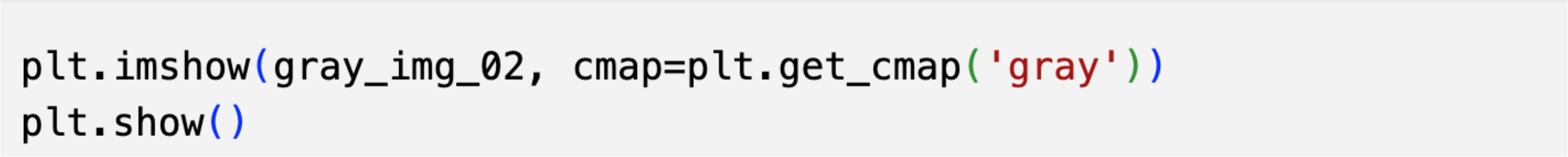
$$\frac{R + G + B}{3}$$



```python
def color2grayscale(vector):
    return np.sum(vector)/3
```



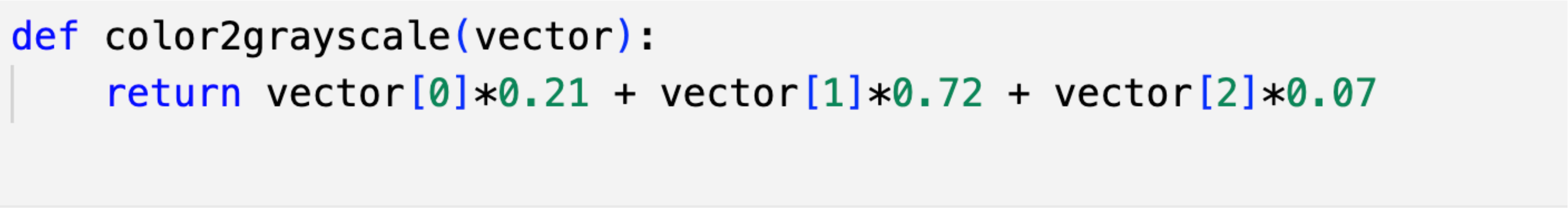
```python
gray_img_02 = np.apply_along_axis(color2grayscale, axis=2, arr=img)
```



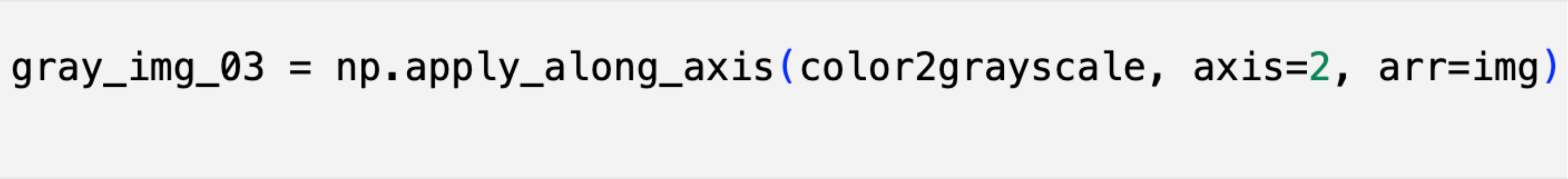
```python
plt.imshow(gray_img_02, cmap=plt.get_cmap('gray'))
plt.show()
```

## Convert Color Image to Grayscale Image

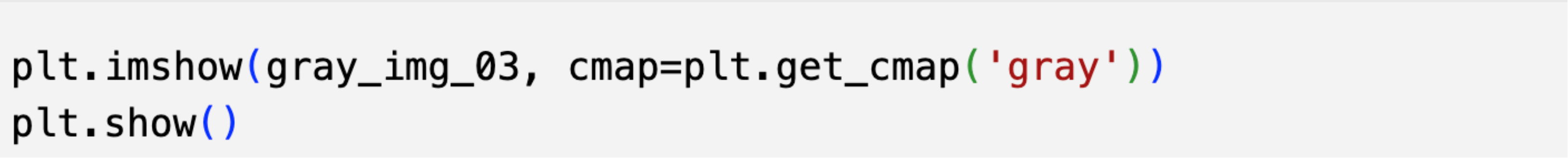
- The luminosity method $0.21 * R + 0.72 * G + 0.07 * B$



```python
def color2grayscale(vector):
    return vector[0]*0.21 + vector[1]*0.72 + vector[2]*0.07
```



```python
gray_img_03 = np.apply_along_axis(color2grayscale, axis=2, arr=img)
```



```python
plt.imshow(gray_img_03, cmap=plt.get_cmap('gray'))
plt.show()
```

# Outline

SECTION 1

Numpy Review

SECTION 2

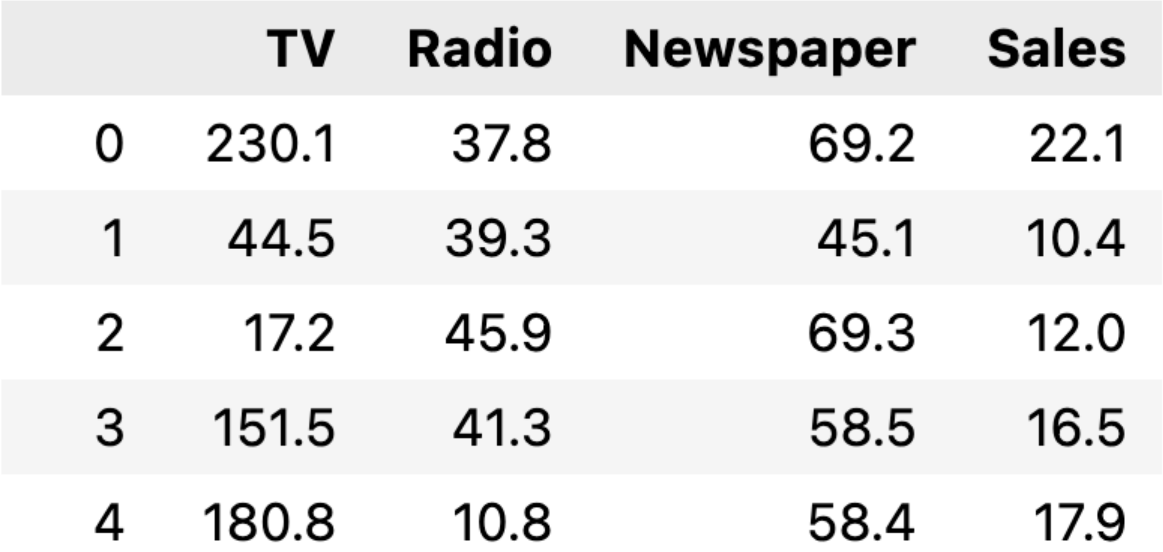
Image Processing

SECTION 3

Tabular Data Analysis

# Tabular Data Analysis

## Tabular Data

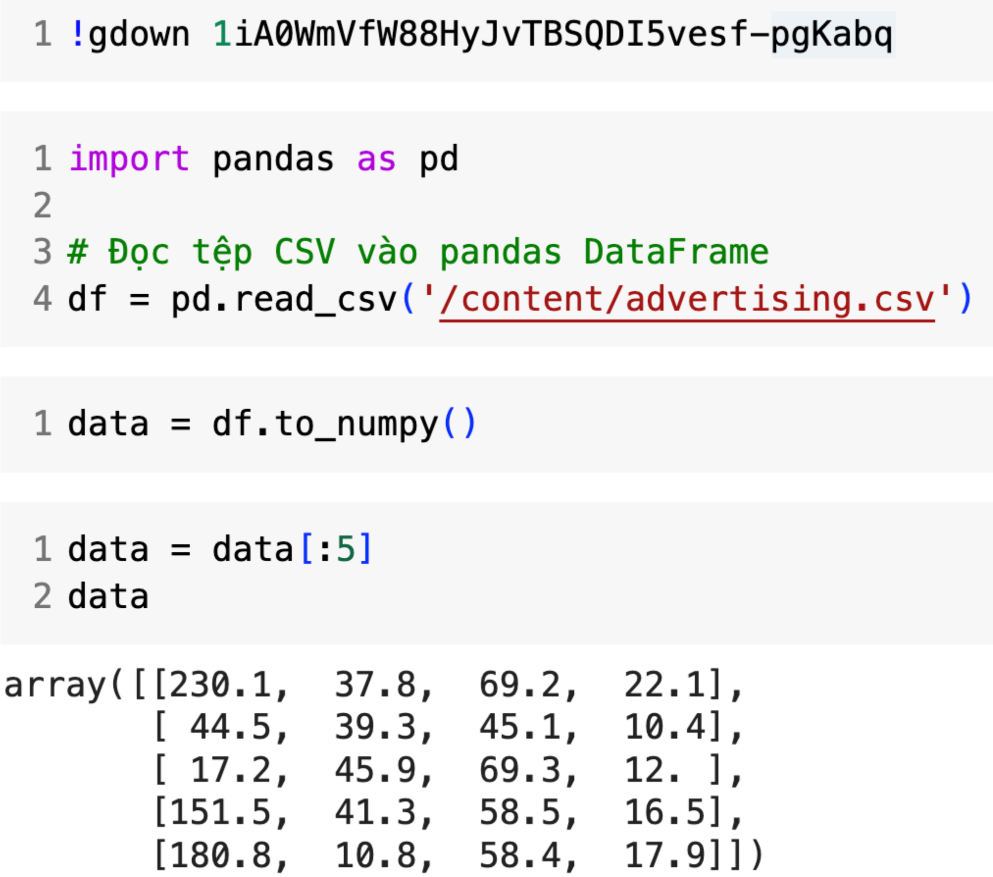


| | TV | Radio | Newspaper | Sales |
|---|---|---|---|---|
| 0 | 230.1 | 37.8 | 69.2 | 22.1 |
| 1 | 44.5 | 39.3 | 45.1 | 10.4 |
| 2 | 17.2 | 45.9 | 69.3 | 12.0 |
| 3 | 151.5 | 41.3 | 58.5 | 16.5 |
| 4 | 180.8 | 10.8 | 58.4 | 17.9 |

# Tabular Data Analysis

## Tabular Data

- Load data



```
!gdown 1iA0WmVfW88HyJvTBSQDI5vesf-pgKabq
```

```
import pandas as pd

# Đọc tệp CSV vào pandas DataFrame
df = pd.read_csv('/content/advertising.csv')
```

```
data = df.to_numpy()
```

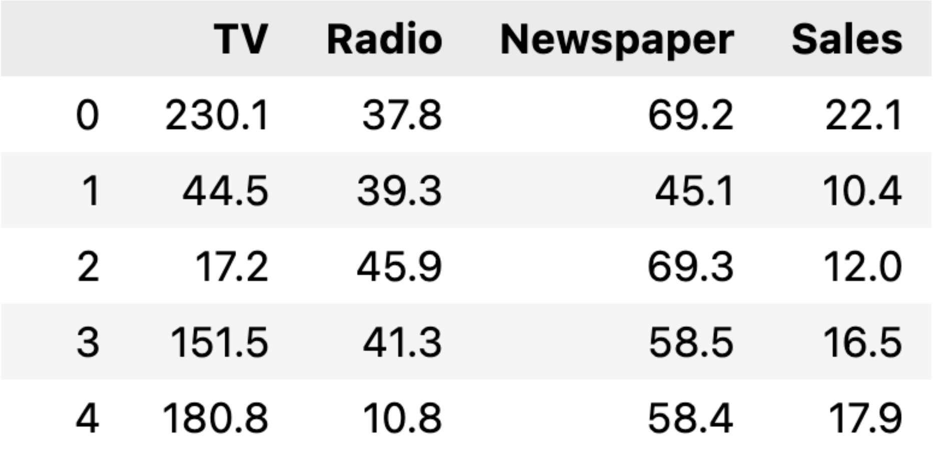
```
data = data[:5]
data
```

```
array([[230.1,  37.8,  69.2,  22.1],
       [ 44.5,  39.3,  45.1,  10.4],
       [ 17.2,  45.9,  69.3,  12. ],
       [151.5,  41.3,  58.5,  16.5],
       [180.8,  10.8,  58.4,  17.9]])
```

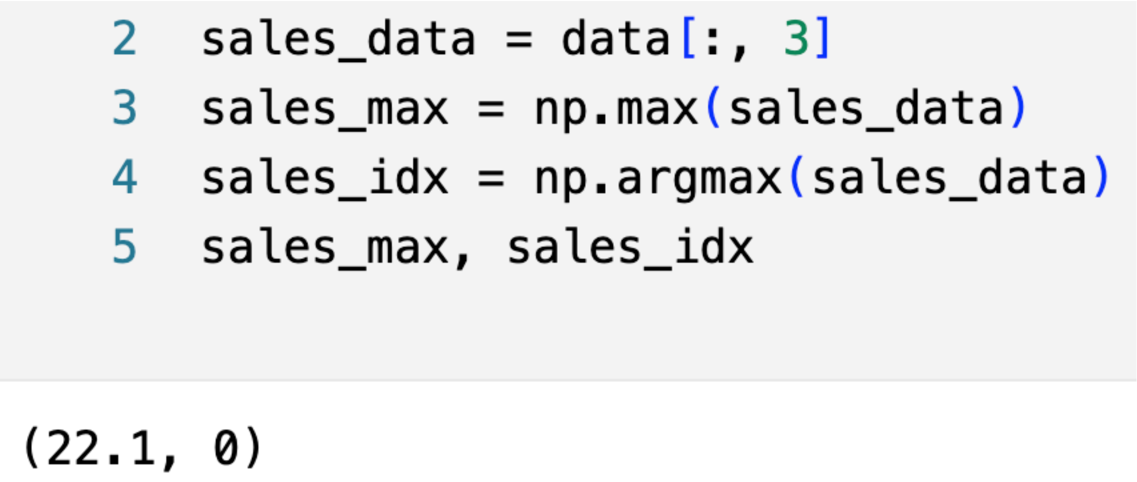
# Tabular Data Analysis

## Tabular Data

- Get the maximum value of the Sales column and its corresponding index



|   | TV | Radio | Newspaper | Sales |
|---|---|---|---|---|
| 0 | 230.1 | 37.8 | 69.2 | 22.1 |
| 1 | 44.5 | 39.3 | 45.1 | 10.4 |
| 2 | 17.2 | 45.9 | 69.3 | 12.0 |
| 3 | 151.5 | 41.3 | 58.5 | 16.5 |
| 4 | 180.8 | 10.8 | 58.4 | 17.9 |



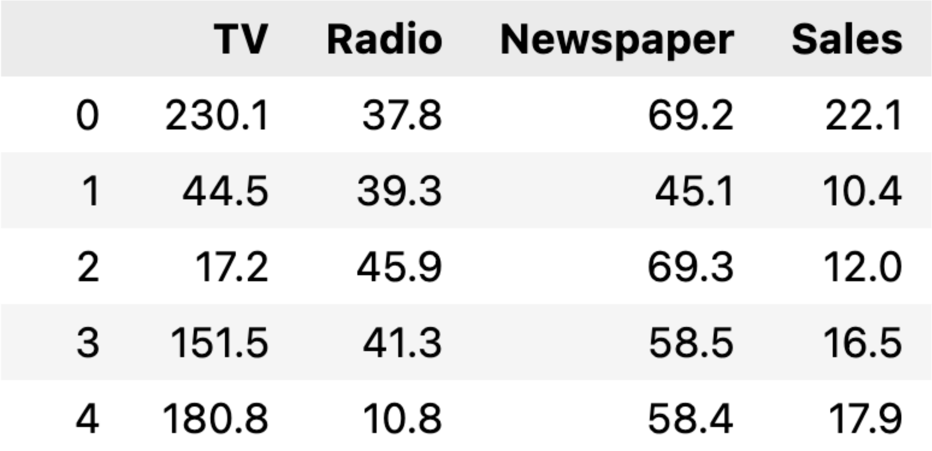
```
sales_data = data[:, 3]
sales_max = np.max(sales_data)
sales_idx = np.argmax(sales_data)
sales_max, sales_idx
```

```
(22.1, 0)
```

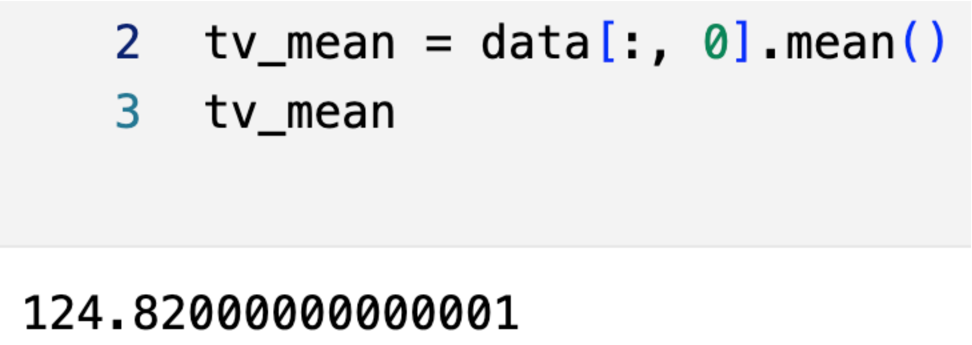
# Tabular Data Analysis

## Tabular Data

- Calculate the average value of the TV column



|   | TV | Radio | Newspaper | Sales |
|---|---|---|---|---|
| 0 | 230.1 | 37.8 | 69.2 | 22.1 |
| 1 | 44.5 | 39.3 | 45.1 | 10.4 |
| 2 | 17.2 | 45.9 | 69.3 | 12.0 |
| 3 | 151.5 | 41.3 | 58.5 | 16.5 |
| 4 | 180.8 | 10.8 | 58.4 | 17.9 |



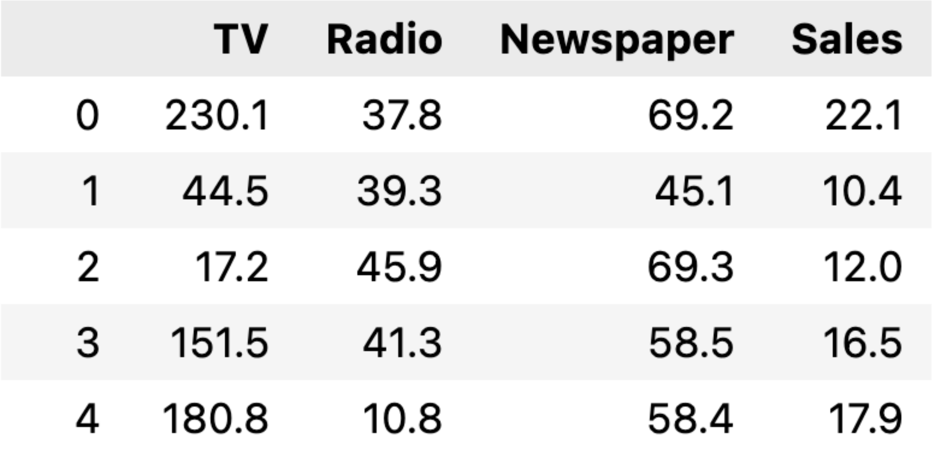
```
tv_mean = data[:, 0].mean()
tv_mean
```

```
124.82000000000001
```

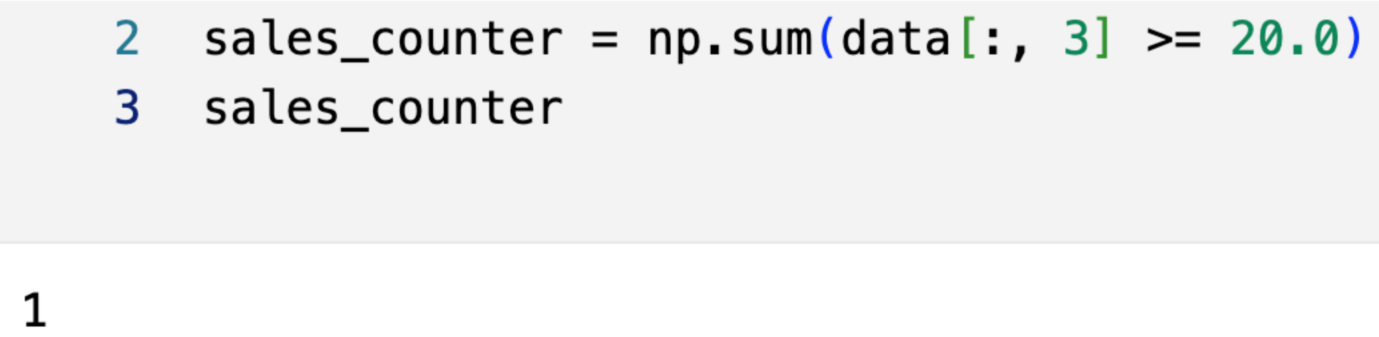
# Tabular Data Analysis

## Tabular Data

- Count the number of rows with Sales >= 20



|   | TV | Radio | Newspaper | Sales |
|---|---|---|---|---|
| 0 | 230.1 | 37.8 | 69.2 | 22.1 |
| 1 | 44.5 | 39.3 | 45.1 | 10.4 |
| 2 | 17.2 | 45.9 | 69.3 | 12.0 |
| 3 | 151.5 | 41.3 | 58.5 | 16.5 |
| 4 | 180.8 | 10.8 | 58.4 | 17.9 |



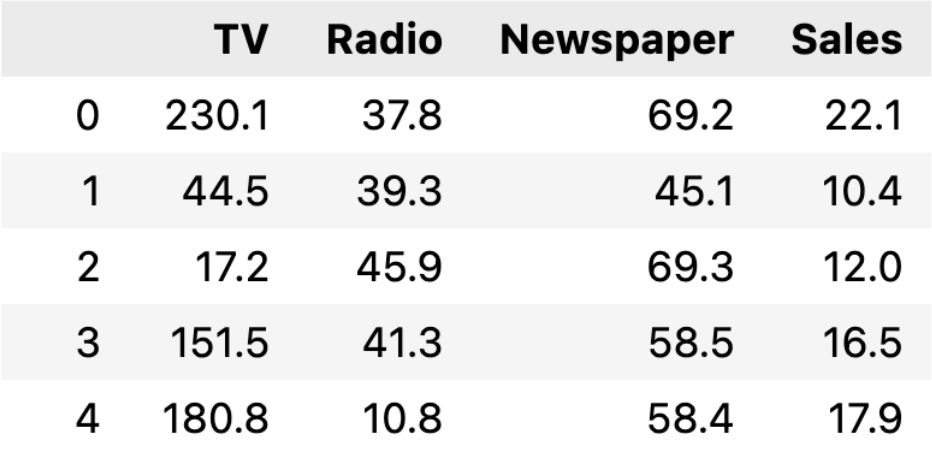
```
sales_counter = np.sum(data[:, 3] >= 20.0)
sales_counter
```

```
1
```

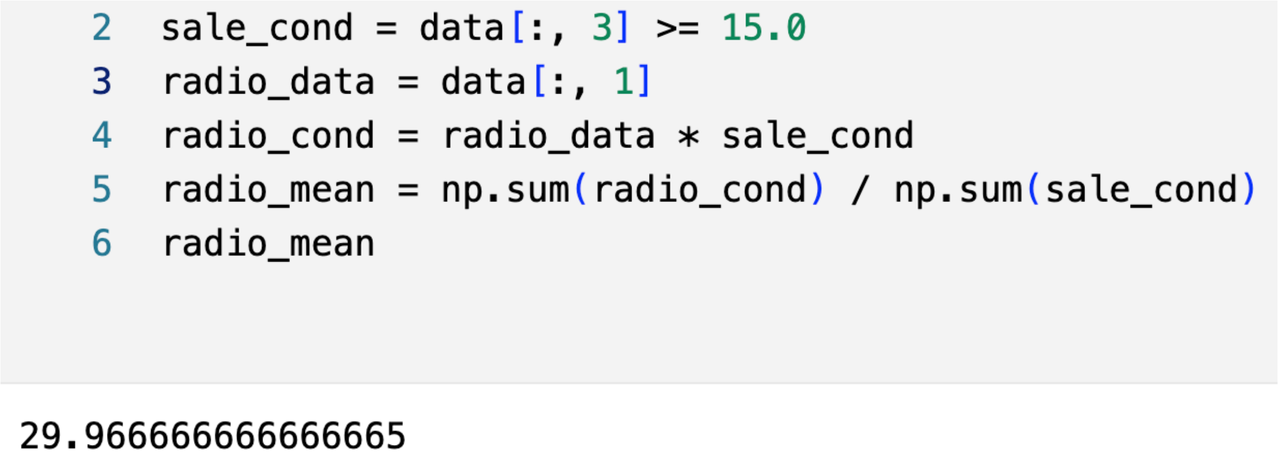
# Tabular Data Analysis

## Tabular Data

- Calculate the average value of the Radio column with Sales column value >= 15



|   | TV | Radio | Newspaper | Sales |
|---|---|---|---|---|
| 0 | 230.1 | 37.8 | 69.2 | 22.1 |
| 1 | 44.5 | 39.3 | 45.1 | 10.4 |
| 2 | 17.2 | 45.9 | 69.3 | 12.0 |
| 3 | 151.5 | 41.3 | 58.5 | 16.5 |
| 4 | 180.8 | 10.8 | 58.4 | 17.9 |



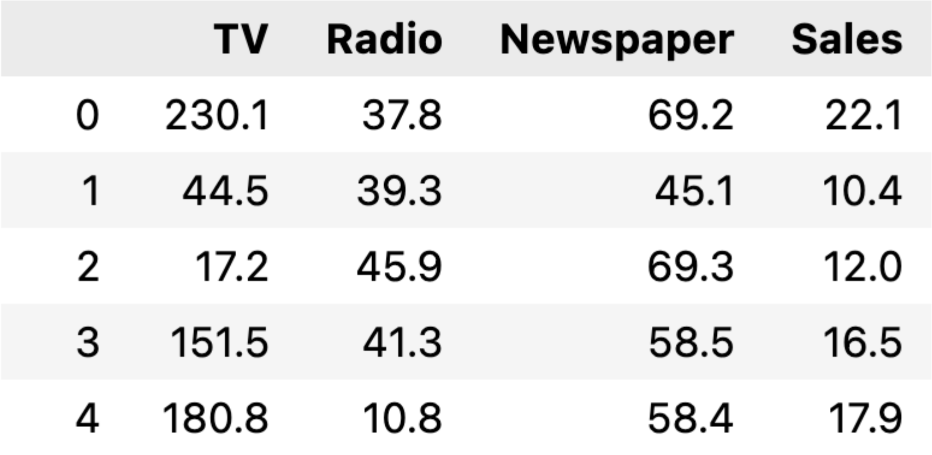
```python
sale_cond = data[:, 3] >= 15.0
radio_data = data[:, 1]
radio_cond = radio_data * sale_cond
radio_mean = np.sum(radio_cond) / np.sum(sale_cond)
radio_mean
```

```
29.966666666666665
```

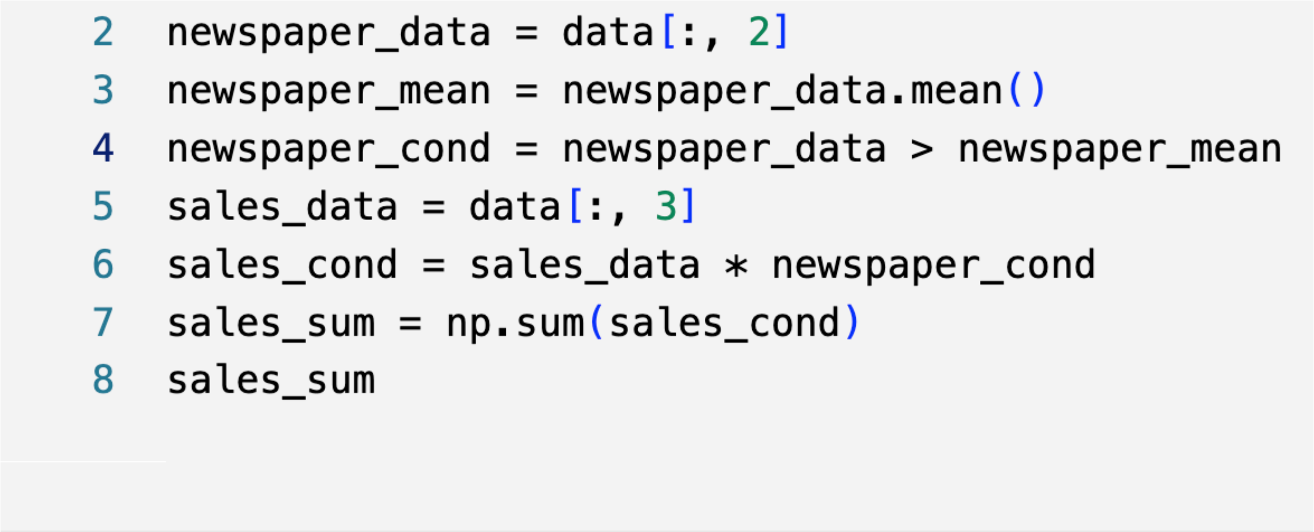
# Tabular Data Analysis

## Tabular Data

- Calculate the total values of Sales (Condition: Newpaper > the average value of all numbers in Newpaper)



| | TV | Radio | Newspaper | Sales |
|---|---|---|---|---|
| 0 | 230.1 | 37.8 | 69.2 | 22.1 |
| 1 | 44.5 | 39.3 | 45.1 | 10.4 |
| 2 | 17.2 | 45.9 | 69.3 | 12.0 |
| 3 | 151.5 | 41.3 | 58.5 | 16.5 |
| 4 | 180.8 | 10.8 | 58.4 | 17.9 |



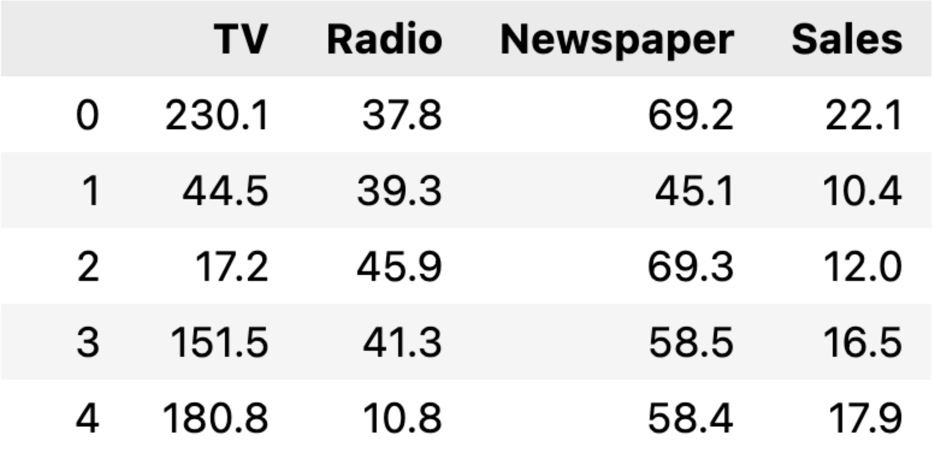
```
newspaper_data = data[:, 2]
newspaper_mean = newspaper_data.mean()
newspaper_cond = newspaper_data > newspaper_mean
sales_data = data[:, 3]
sales_cond = sales_data * newspaper_cond
sales_sum = np.sum(sales_cond)
sales_sum
```

```
34.1
```

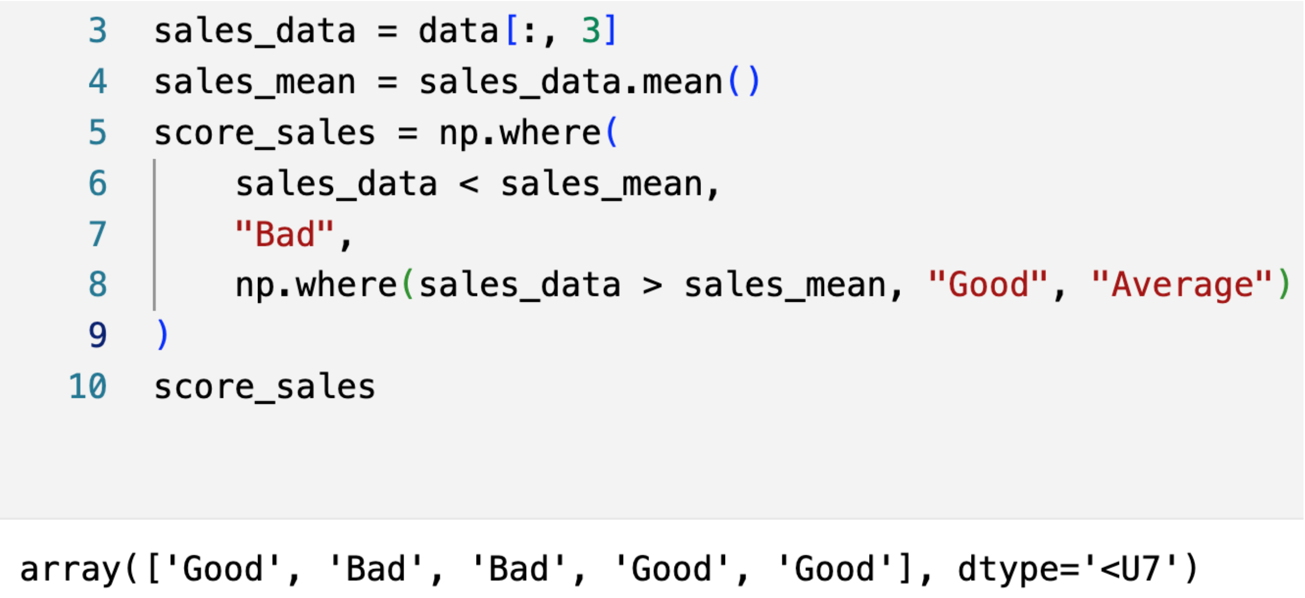
# Tabular Data Analysis

## Tabular Data

- Create new array that contains three values : Good, Average, Bad
- Value > the average of Sales => Good, < => Bad, = => Average



|   | TV | Radio | Newspaper | Sales |
|---|---|---|---|---|
| 0 | 230.1 | 37.8 | 69.2 | 22.1 |
| 1 | 44.5 | 39.3 | 45.1 | 10.4 |
| 2 | 17.2 | 45.9 | 69.3 | 12.0 |
| 3 | 151.5 | 41.3 | 58.5 | 16.5 |
| 4 | 180.8 | 10.8 | 58.4 | 17.9 |



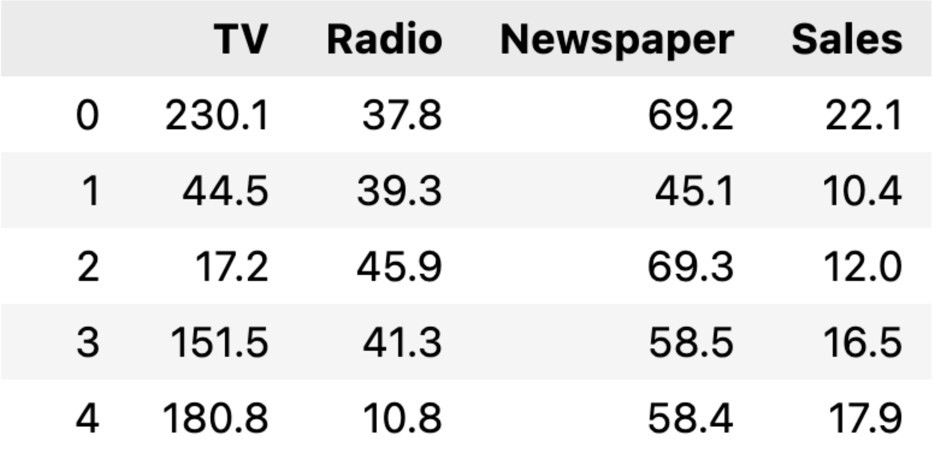
```python
sales_data = data[:, 3]
sales_mean = sales_data.mean()
score_sales = np.where(
    sales_data < sales_mean,
    "Bad",
    np.where(sales_data > sales_mean, "Good", "Average")
)
score_sales
```

```
array(['Good', 'Bad', 'Bad', 'Good', 'Good'], dtype='<U7')
```

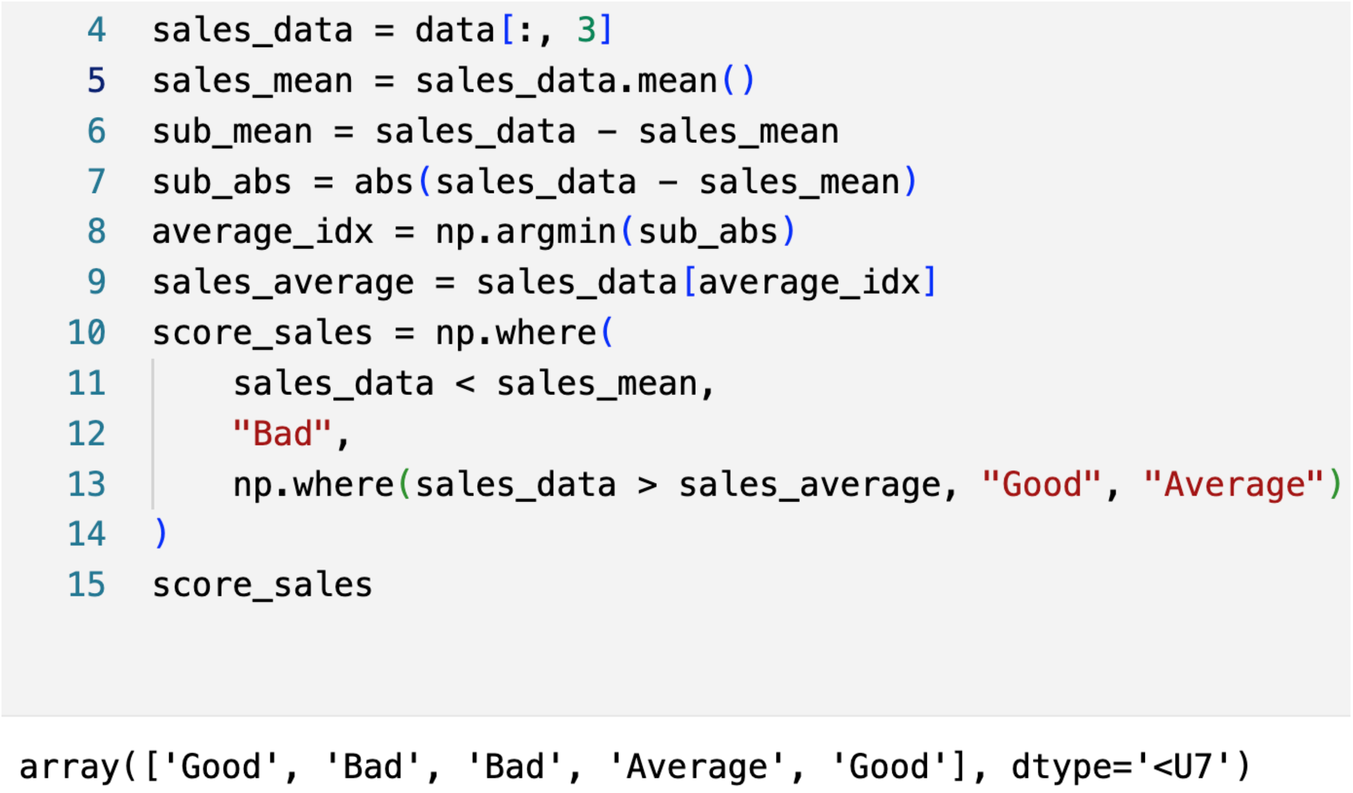
# Tabular Data Analysis

## Tabular Data

- Create new array that contains three values : Good, Average, Bad
- Value > the closest value to the average of Sales => Good, < => Bad, = => Average



| | TV | Radio | Newspaper | Sales |
|---|---|---|---|---|
| 0 | 230.1 | 37.8 | 69.2 | 22.1 |
| 1 | 44.5 | 39.3 | 45.1 | 10.4 |
| 2 | 17.2 | 45.9 | 69.3 | 12.0 |
| 3 | 151.5 | 41.3 | 58.5 | 16.5 |
| 4 | 180.8 | 10.8 | 58.4 | 17.9 |



```python
sales_data = data[:, 3]
sales_mean = sales_data.mean()
sub_mean = sales_data - sales_mean
sub_abs = abs(sales_data - sales_mean)
average_idx = np.argmin(sub_abs)
sales_average = sales_data[average_idx]
score_sales = np.where(
    sales_data < sales_mean,
    "Bad",
    np.where(sales_data > sales_average, "Good", "Average")
)
score_sales
```

```
array(['Good', 'Bad', 'Bad', 'Average', 'Good'], dtype='<U7')
```

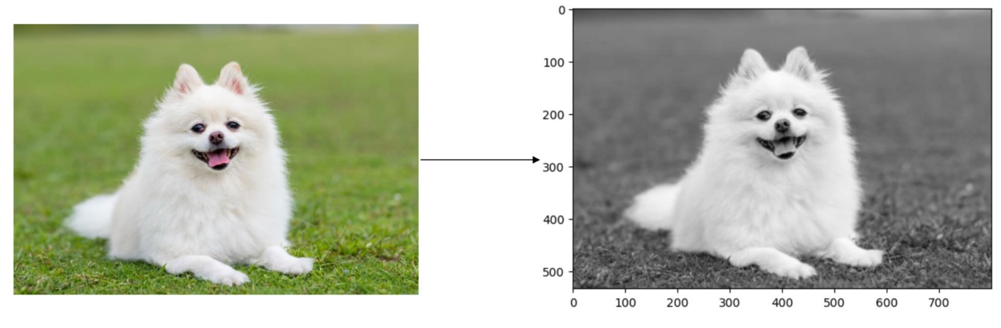
# Summary

## Grayscale Image

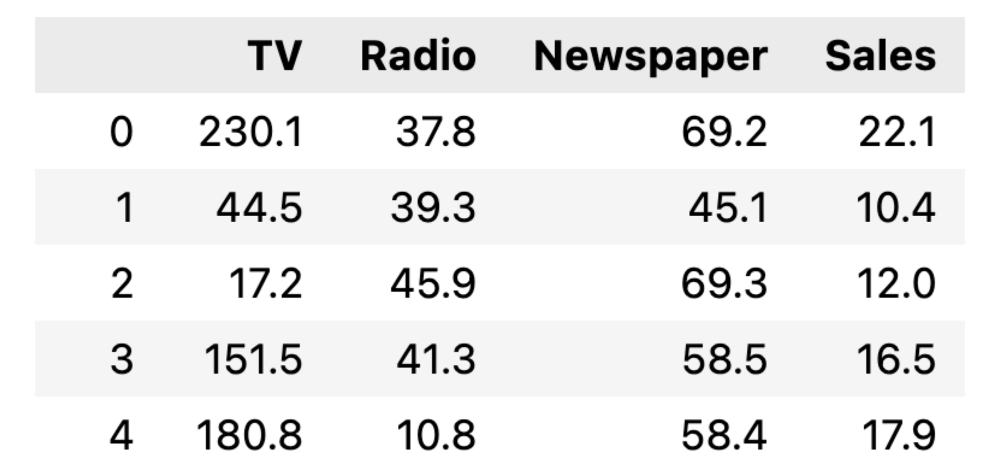
$$\frac{\max(R,G,B)+\min(R,G,B)}{2}$$

$$\frac{R+G+B}{2}$$

$$\frac{0.21*R+0.72*G+0.07*B}{2}$$



## Tabular Data Analysis



| | TV | Radio | Newspaper | Sales |
|---|---|---|---|---|
| 0 | 230.1 | 37.8 | 69.2 | 22.1 |
| 1 | 44.5 | 39.3 | 45.1 | 10.4 |
| 2 | 17.2 | 45.9 | 69.3 | 12.0 |
| 3 | 151.5 | 41.3 | 58.5 | 16.5 |
| 4 | 180.8 | 10.8 | 58.4 | 17.9 |

# Thanks!

## Any questions?