## Welcome to ABT/HYD 182

# **Lab 3**: for..loops, numpy arrays, and conditional statements

# Due Date: Jan 29 | 11:59 PM

---

## Session I

### NumPy

- NumPy (Numerical Python) is an open source Python library that's used in almost every field of science and engineering. The NumPy library contains multidimensional array and matrix data structures.
- NumPy is already installed, so no need to install it again. You only need to import it to your notebook. The most common way to import numpy is: `import numpy as np`
- We shorten the imported name to np for better code readability using NumPy. This widely adopted convention makes your code more readable for everyone working on it.

Recourses:

- NumPy: the absolute basics for beginners
- Learn NumPy : NumPy 50 exercises and solution
- Image manipulation and processing using Numpy and Scipy

### Image manipulation using Numpy

1- Downloading a Raster file.

Download a UAV-based multispectral image from Box at https://ucdavis.box.com/s/4tqpao2lrzrc97pmx6fvm5jcckguzjgy. This is an orthomosaic image generated from stitching multispectral images with five bands captured by Micasense RedEdge camera from a vineyard. The order of bands is blue, green, red, red edge, and near-infrared. As an example, I created a folder called "ABT182" on my Google Drive and pasted the multispectral image in this folder.

*Example: G:\My Drive\ABT182 ===> 'multispectral_image.tif'*

2- Installing the required packages

Google Colab comes with the most common packages. We need to install rasterio library, which is a Python library for the analysis of raster files. To install rasterio, you just need to run the following cell!

In [ ]:
```
# to install rasterio, run this cell.
!pip install rasterio

# !pip uninstall rasterio  # to uninstall rasterio
```

3- Importing the required packages

Here, we will import the required packages for this assignment. Once you installed rasterio, now you can import it. The other packages are already installed and come with Google Colab.

In [1]:
```
# here we import the packages (libraries) we need for this assignment
import rasterio
import numpy as np
import matplotlib.pyplot as plt # a comprehensive library for creating static, animate
```

4- Mounting Google Drive

You should mount your Google Drive to Colab so you can have access to the files (like the image you downloaded for this lab) in your Google Drive.

- Run the below cell
- Go to the URL shown in the cell using a browser (the link is shown once you run the cell)
- Sign in with your Google account, then hit “Allow”. Copy this code by clicking on the icon on the right side.
- Paste (Ctrl + V) it in the notebook where it says “Enter your authorization code”:
- Hit enter.
- To check it is successfully mounted, click on the folder icon on the left side of the Colab. Your Google Drive (myGdrive) should be shown there.

*Note:* to unmount your Google Drive you can run this line of code in a code cell: drive.flush_and_unmount(). But we don't need to do it for this assignment.

- A tutorial showing how to mount Google Drive in Google Colab: https://www.marktechpost.com/2019/06/07/how-to-connect-google-colab-with-google-drive/

In [ ]:
```
# Run this cell to let the notebook have access to your Google Drive.
from google.colab import drive
drive.mount('/content/myGdrive')
```

5- Reading the multispectral image

- Once your Google Drive is mounted, from the left panel (Files > myGdrive) navigate to the folder you saved the multispectral image. Right click on the image and copy the path (right click > Copy path). If you need more instruction, please check HERE.
- Define a variable called image_path and assign the path you copied to it.
- Note the path is a string and therefore, it should be inside a quotation.
- Note: if you want to do the assignment in Spyder, we need to use letter r, which stands for raw, before the path to make backslashes in the string path be interpreted as actual backsplashes and not as python special characters.

  ```
  example

  Google colab:
  image_path = '/content/myGdrive/MyDrive/ABT182/multispectral_image.tif'

  Spyder:
  image_path = r'G:\My Drive\ABT182\multispectral_image.tif'

  Pay attention to the difference in slash and backslash.
  ```
- Open the multispectral image using rasterio. hint: use `open()` method in rasterio to open the file a rasterio object. You can find an example of how to open and read a file here. Please note `open()` method returns an object (not an image).
- Now read the image as a numpy array (matrix). Hint: use `read()` to read the multispectral image as a numpy array as assign it to a variable called image_array.

Please note we want to read all bands (5 bands) of the multispectral image into a 3-dimensional ndarray.

In [ ]:
```
# write your code here

image_path = 'PathToFolderYourSavedTheImage'

'''
# example
# image_path = '/content/myGdrive/MyDrive/ABT182/multispectral_image.tif'

# in spyder or other ide
# image_path = r'G:\My Drive\ABT182\multispectral_image.tif'
'''
image_rio = rasterio.open(image_path)       # use 'open()' to read the file a rasteric
image_array = image_rio.read()              # use 'read()' to  read the multispectral
```

**Exercise 1**

Access to geospatial and other information of a raster file

Write a program to answer the following questions.

- What is the size of the image (height, width, and number of bands)? Hint: use `np.shape()` to find the size of an array. Note that image_array is a 3-dimensional ndarray and interpretation of these 3 axes is (bands, rows, columns). We will transpose it to a regular order (rows, columns, bands) later.

- What is the coordinate reference system (CRS) of the image? Hint: You may want to check the attributes of our rasterio object (examples), which is image_rio in our notebook (e.g., image_rio.bounds returns image extent in cartesian coordinates).
- What is the affine tranform?
- What is the pixel value for no data pixels?
- What is the data types?
- Print the image metadata.

*NOTE:* You may use print function to print the information e.g., print("\n", "Image bounds: ", image_rio.bounds) # "\n" skips a line and prints the result in a new line.

In [5]:
```
# Example
# what is the size of the image_array? Hint: use 'shape' to find the size of an array.
print("\n", "Image bounds: ", image_rio.bounds)
```

```
 Image bounds:  BoundingBox(left=269879.183, bottom=4046995.9311, right=270027.95, to
p=4047172.7481)
```

In [6]:
```
# write your code here

# Image size (bands, rows, columns)
print("\n", "Image size (bands, rows, columns): ", np.shape(image_array))
print("Number of bands:", np.shape(image_array)[0])
print("Height (rows):", np.shape(image_array)[1])
print("Width (columns):", np.shape(image_array)[2])

# Coordinate Reference System (CRS)
print("\n", "Coordinate Reference System (CRS): ", image_rio.crs)

# Affine transform
print("\n", "Affine transform: ", image_rio.transform)

# No data value
print("\n", "No data value: ", image_rio.nodata)

# Data type
print("\n", "Data type: ", image_rio.dtypes)

# Image bounds
print("\n", "Image bounds: ", image_rio.bounds)

# Print all metadata
print("\n", "Image metadata:")
print(image_rio.meta)
```

```
 Image size (bands, rows, columns):  (5, 3467, 2917)
Number of bands: 5
Height (rows): 3467
Width (columns): 2917

 Coordinate Reference System (CRS):  EPSG:32611

 Affine transform:  | 0.05, 0.00, 269879.18|
| 0.00,-0.05, 4047172.75|
| 0.00, 0.00, 1.00|

 No data value:  -10000.0

 Data type:  ('float32', 'float32', 'float32', 'float32', 'float32')

 Image bounds:  BoundingBox(left=269879.183, bottom=4046995.9311, right=270027.95, to
p=4047172.7481)

 Image metadata:
{'driver': 'GTiff', 'dtype': 'float32', 'nodata': -10000.0, 'width': 2917, 'height':
3467, 'count': 5, 'crs': CRS.from_epsg(32611), 'transform': Affine(0.050999999999974
4, 0.0, 269879.183,
       0.0, -0.050999999999944125, 4047172.7481)}
```

**Exercise 2**

Slice the image to five bands and assign a name to each band

- First, we need to transpose the image array (image_array) to m (row) by n (col) by b (bands) image, which is a format expected by most other software and packages, like matplotlib.
- To change the axis order, we will use `reshape_as_image()` method from rasterio. You can find more information and an example here.
- Assign the transposed image (row, col, bands) to a variable called `image`.
- Print the size (shape) of the variable image to check if it is now in the form of row, column, and band.
- Define variables called blue, green, red, rededge, nir and assign a single band to each of them. The multispectral image has five bands (blue, green, red, red edge, and NIR).

Please note python indexing starts at *zero*! So the index of the first band (blue) is zero.

Please check this page to learn about numpy array slicing and indexing.

Example:

```
    blue = image[:,:,0] # all the rows and columns of the first band
(index 0) is assigned to blue band.
```

In [7]:
```python
# write your code here

# Transpose the image array to (rows, columns, bands) format
from rasterio.plot import reshape_as_image
```

```
image = reshape_as_image(image_array)

# Print the shape to verify
print("image size:", image_array.shape)
print("\nimage shape after tranpose:", image.shape)
```

```
image size: (5, 3467, 2917)

image shape after tranpose: (3467, 2917, 5)
```

In [8]:
```
# write your code here

# Extract individual bands
blue = image[:,:,0]      # Blue band (index 0)
green = image[:,:,1]    # Green band (index 1)
red = image[:,:,2]      # Red band (index 2)
rededge = image[:,:,3]  # Red edge band (index 3)
nir = image[:,:,4]      # Near-infrared band (index 4)

# Print shapes to verify
print("Blue band shape:", blue.shape)
print("Green band shape:", green.shape)
print("Red band shape:", red.shape)
print("Red edge band shape:", rededge.shape)
print("NIR band shape:", nir.shape)
```

```
Blue band shape: (3467, 2917)
Green band shape: (3467, 2917)
Red band shape: (3467, 2917)
Red edge band shape: (3467, 2917)
NIR band shape: (3467, 2917)
```

**Exercise 3**

Visualization of bands

- Write a for loop to display each band.
- Define the figure size, e.g., figsize=(10, 8).
- Select a colormap, e.g., cmap = 'viridis'
- Assign an appropriate title to each figure. You may want to create a list of titles (title_list = ['blue band', 'green band', ...]) and call each element (title) in your loop.
- Once the cell is run, all bands are shown.
- Use `matplotlib` We already imported in above cells

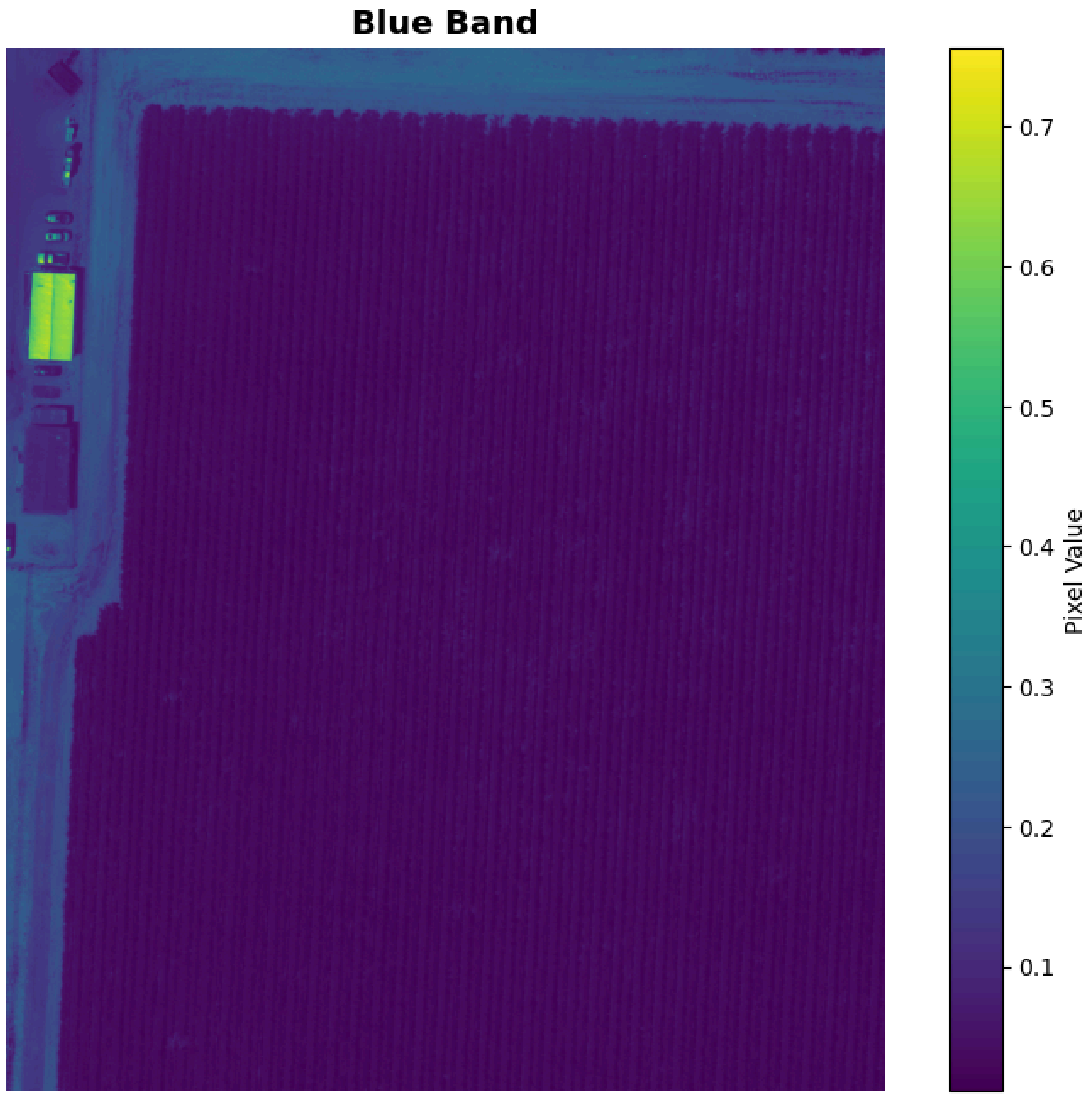
  `import matplotlib.pyplot as plt` `plt.figure(figsize=(10, 8), dpi=100)`

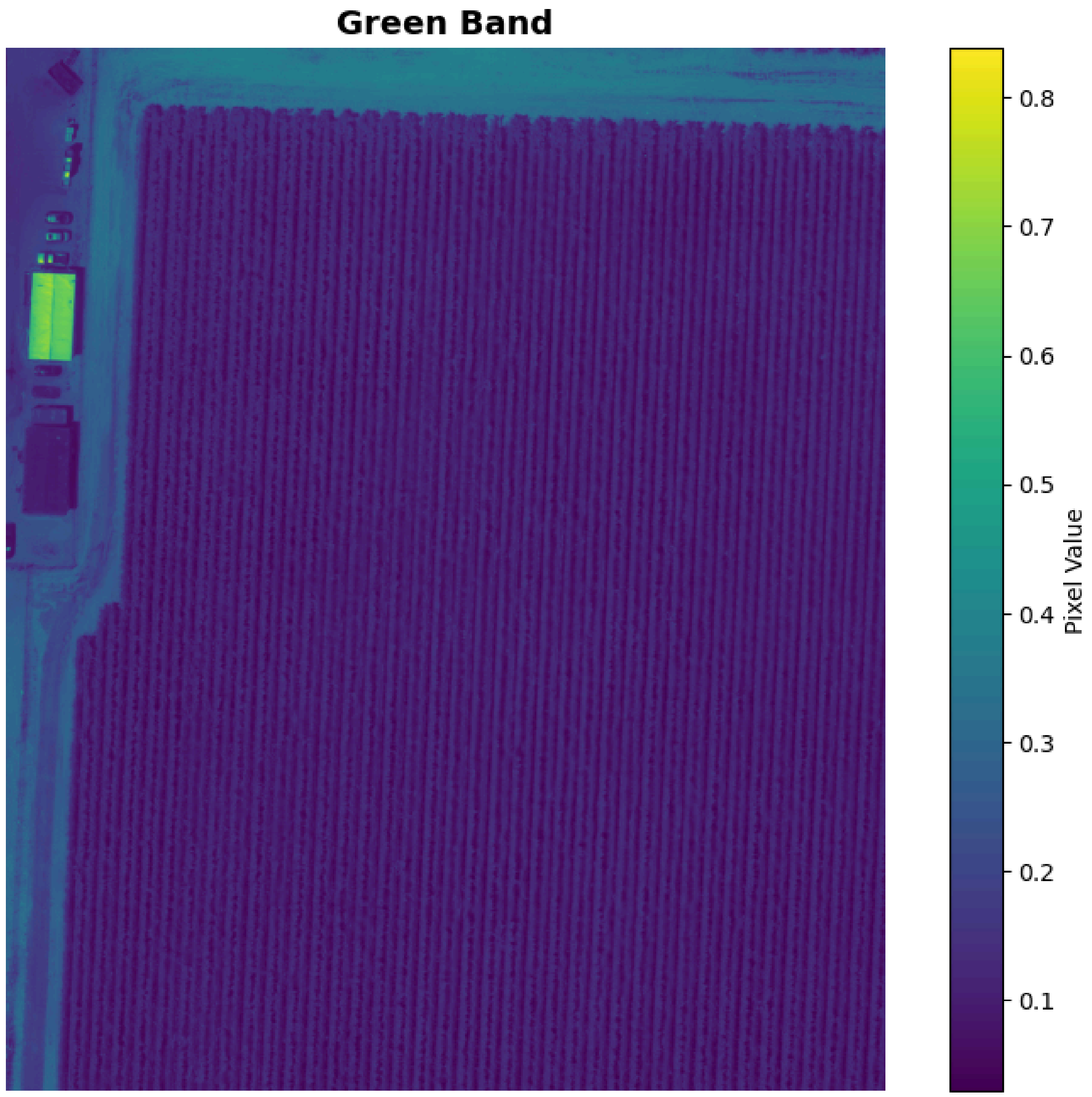
Resources:

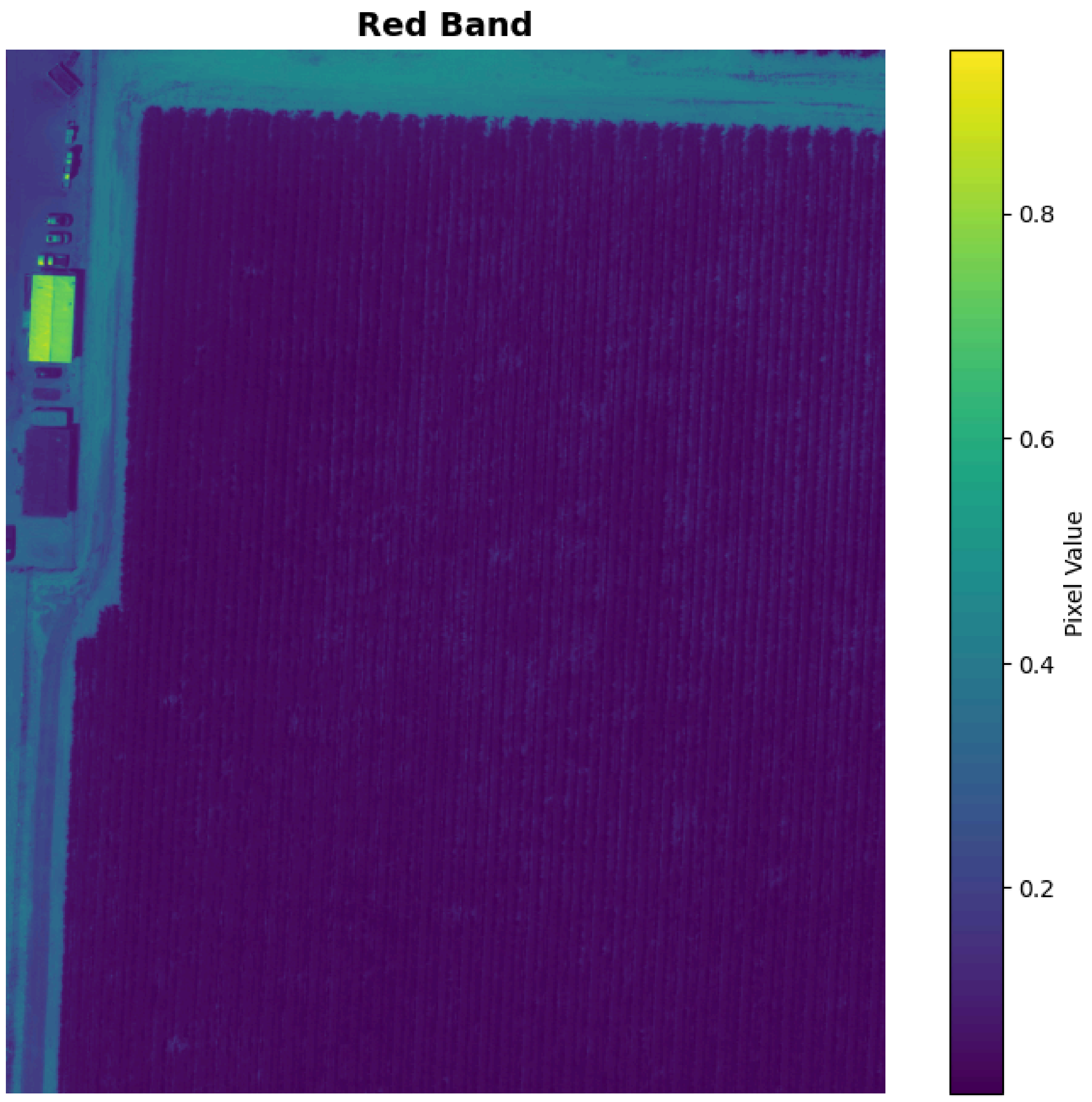
- https://matplotlib.org/stable/tutorials/images.html
- https://matplotlib.org/stable/plot_types/index.html

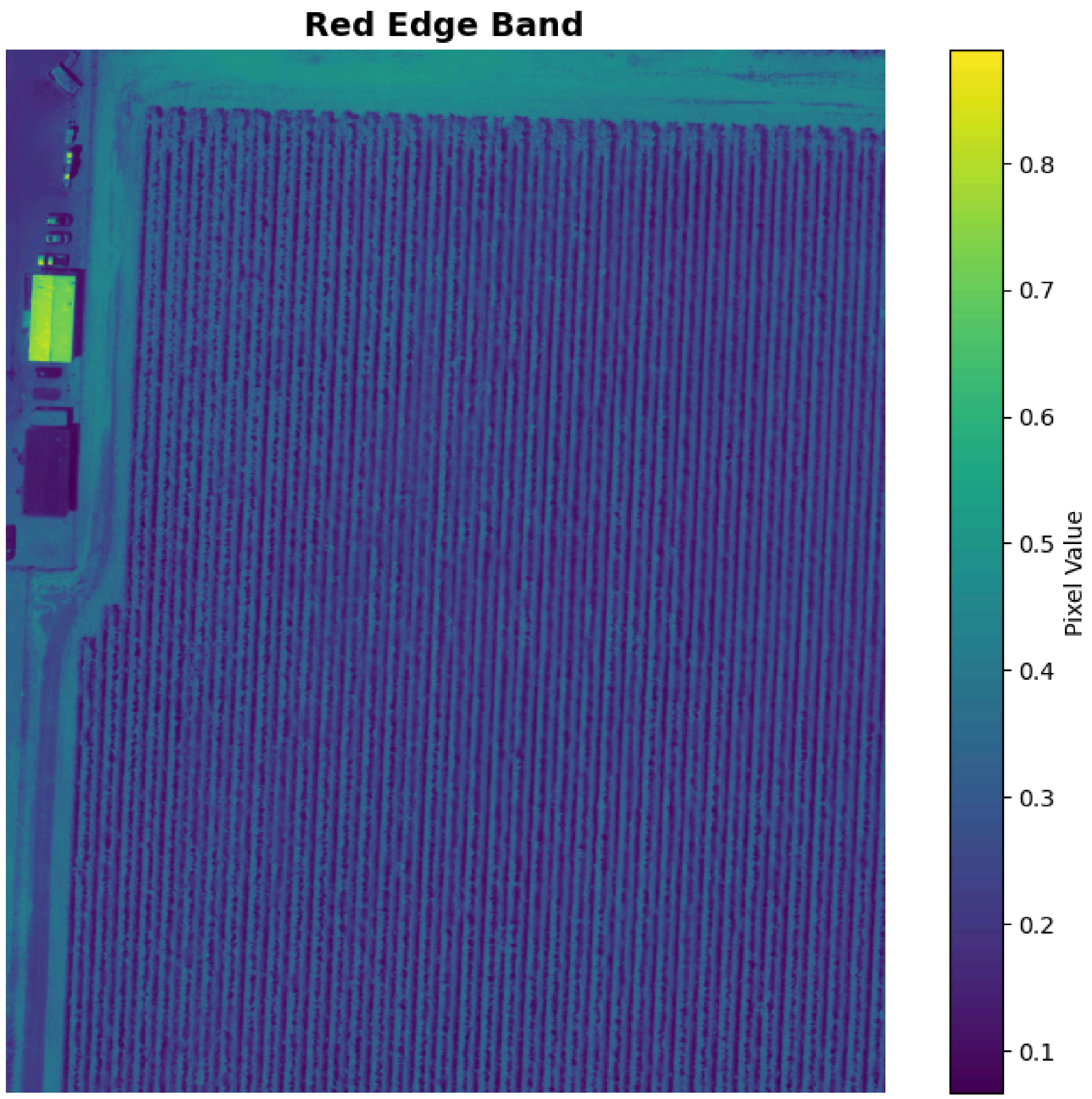
In [9]:
```
# write your code here (10 points)
# set the backend to display the output of plotting commands inline within frontends
%matplotlib inline

# Create a list of band titles
```

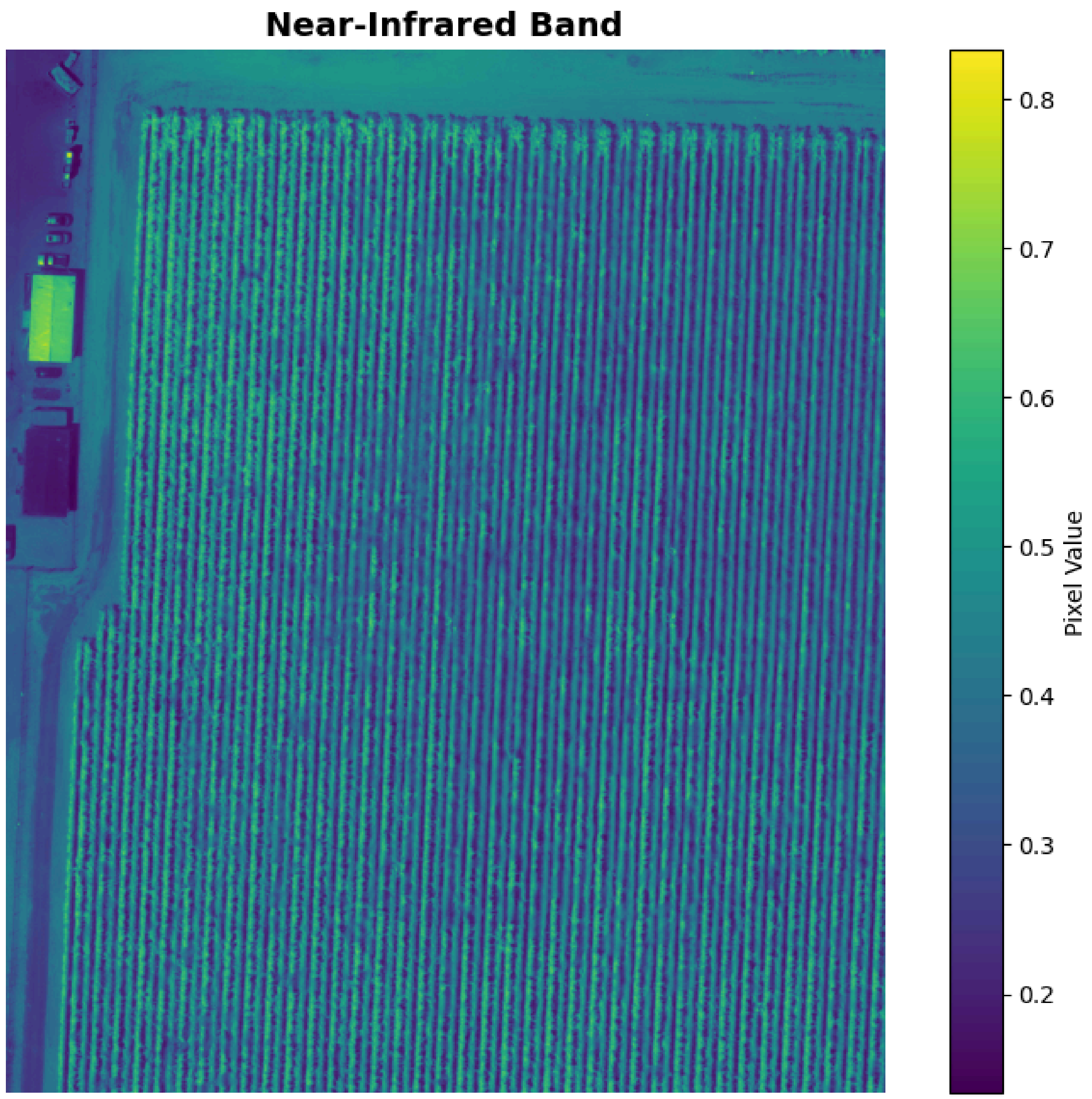
```
title_list = ['Blue Band', 'Green Band', 'Red Band', 'Red Edge Band', 'Near-Infrared E
band_list = [blue, green, red, rededge, nir]

# Loop through each band and display it
for i in range(len(band_list)):
    plt.figure(figsize=(10, 8), dpi=100)
    plt.imshow(band_list[i], cmap='viridis')
    plt.title(title_list[i], fontsize=14, fontweight='bold')
    plt.colorbar(label='Pixel Value')
    plt.axis('off')  # Remove axes for cleaner display
    plt.show()
```




Green Band
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
Pixel Value


Red Band
0.8
0.6
0.4
0.2
Pixel Value


Red Edge Band
Pixel Value
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1



## Session II

### Loops and conditional statements

**Exercise 4**

Part A. We want to create a list of odd and even numbers from the numbers between 1 to 30.

- create a list of numbers between 1 and 30. (hint: you can use `range()` to generate the numbers)
- print this list to ensure it contains the values you expect.
- loop through the items in the list and make decisions if it's odd or even.
- add the item to the list of odd numbers or even numbers accordingly. (hint: you can create an empty list and then use `append()` to add items to it)

Part B. In part A, your program loops through each item and has `if` statements to create a new list. This is about 5 lines of code. Can we create a list of odd and even numbers with only two lines of code, one line for creating odd and another line for even list. (hint: you may use List Comprehension)

Parc C. Write a program that creates a list with all the numbers in the even list raised to the power of 2. You may again use list comprehension and do this in just one line of code or use for loops.

In [10]:

```python
# Part A

# Create a list of numbers between 1 and 30
numbers = list(range(1, 31))
print("Numbers list:", numbers)

# Create empty lists for odd and even numbers
even_list = []
odd_list = []

# Loop through the numbers and categorize them
for num in numbers:
    if num % 2 == 0:  # Check if number is even
        even_list.append(num)
    else:  # Number is odd
        odd_list.append(num)

print("even list: ", even_list)
print("odd list: ", odd_list)
```

```
Numbers list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30]
even list:  [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]
odd list:  [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
```

In [11]:

```python
# Part B
# write your code here

# Using list comprehension to create odd and even lists in just 2 lines
numbers = list(range(1, 31))
print(numbers)

even_list = [x for x in numbers if x % 2 == 0]
odd_list = [x for x in numbers if x % 2 != 0]

print("even list: ", even_list)
print("odd list: ", odd_list)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 2
4, 25, 26, 27, 28, 29, 30]
even list:  [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]
odd list:  [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
```

In [12]:

```python
# Part C
# add your code here

# Using list comprehension to create a list with even numbers raised to power of 2
even_squared = [x**2 for x in even_list]
print(even_squared)
```

```
# Alternative using for loop:
# even_squared = []
# for x in even_list:
#     even_squared.append(x**2)
# print(even_squared)
```

```
[4, 16, 36, 64, 100, 144, 196, 256, 324, 400, 484, 576, 676, 784, 900]
```

**Exercise 5**

- Create two new lists each with the first 5 numbers in your even and odd lists (list slicing).
- Write a program that multiplies each number of your new even list (with the first 5 even numbers) to the other list (with the first five odd numbers) and print the results in the terminal as follows: [hint: you may use nested loop]

```
2 * 1 =  2
2 * 3 =  6
2 * 5 =  10
2 * 7 =  14
2 * 9 =  18
4 * 1 =  4
4 * 3 =  12
4 * 5 =  20
4 * 7 =  28
4 * 9 =  36
6 * 1 =  6
6 * 3 =  18
6 * 5 =  30
6 * 7 =  42
6 * 9 =  54
8 * 1 =  8
8 * 3 =  24
8 * 5 =  40
8 * 7 =  56
8 * 9 =  72
10 * 1 =  10
10 * 3 =  30
10 * 5 =  50
10 * 7 =  70
10 * 9 =  90
```

In [13]:
```
# add your code here
# feel free to add more code cells if needed.

# Create two new lists with the first 5 numbers from even and odd lists
even_first5 = even_list[:5]
odd_first5 = odd_list[:5]

# Nested loop to multiply each even number with each odd number
for even_num in even_first5:
    for odd_num in odd_first5:
        result = even_num * odd_num
        print(f"{even_num} * {odd_num} = {result:3d}")
```

```
2 * 1 =   2
2 * 3 =   6
2 * 5 =  10
2 * 7 =  14
2 * 9 =  18
4 * 1 =   4
4 * 3 =  12
4 * 5 =  20
4 * 7 =  28
4 * 9 =  36
6 * 1 =   6
6 * 3 =  18
6 * 5 =  30
6 * 7 =  42
6 * 9 =  54
8 * 1 =   8
8 * 3 =  24
8 * 5 =  40
8 * 7 =  56
8 * 9 =  72
10 * 1 =  10
10 * 3 =  30
10 * 5 =  50
10 * 7 =  70
10 * 9 =  90
```

**Exercise 6**

- Write a program to check if a given coordinate is within a specified bounding box. The bounding box is defined by its southwest (lower left) and northeast (upper right) corners.

```
# a dictionary containing 6 points
point_dic = {
            'p1': (40.7189, -73.9981),  # Latitude, Longitude
            'p2': (40.7292, -73.9904),  # Latitude, Longitude
            'p3': (40.6890, -73.9977),  # Latitude, Longitude
            'p4': (40.7065, -73.9795),  # Latitude, Longitude
            'p5': (40.6580, -74.0532),  # Latitude, Longitude
            "p6":  (40.7128, -74.0060)  # Latitude, Longitude
            }

bbox_southwest = (40.7000, -74.0200) # Lower left corner
bbox_northeast = (40.7300, -73.9900) # Upper right corner
```

Expected outcome that should be printed in the terminal:

```
The point p1 is within the bounding box.
The point p2 is within the bounding box.
The point p3 is outside the bounding box.
The point p4 is outside the bounding box.
The point p5 is outside the bounding box.
The point p6 is within the bounding box.
```

In [14]:
```
# add your code here
# a dictionary containing 6 points
point_dic = {
            'p1': (40.7189, -73.9981),  # Latitude, Longitude
            'p2': (40.7292, -73.9904),  # Latitude, Longitude
            'p3': (40.6890, -73.9977),  # Latitude, Longitude
            'p4': (40.7065, -73.9795),  # Latitude, Longitude
            'p5': (40.6580, -74.0532),  # Latitude, Longitude
            "p6":  (40.7128, -74.0060)  # Latitude, Longitude
            }

bbox_southwest = (40.7000, -74.0200) # Lower left corner (lat_min, lon_min)
bbox_northeast = (40.7300, -73.9900) # Upper right corner (lat_max, lon_max)

# Loop through each point and check if it's within the bounding box
for point_name, (lat, lon) in point_dic.items():
    # Check if point is within bounding box
    # Latitude must be between bbox_southwest[0] and bbox_northeast[0]
    # Longitude must be between bbox_southwest[1] and bbox_northeast[1]
    if (bbox_southwest[0] <= lat <= bbox_northeast[0] and
        bbox_southwest[1] <= lon <= bbox_northeast[1]):
        print(f"The point {point_name} is within the bounding box.")
    else:
        print(f"The point {point_name} is outside the bounding box.")
```

```
The point p1 is within the bounding box.
The point p2 is within the bounding box.
The point p3 is outside the bounding box.
The point p4 is outside the bounding box.
The point p5 is outside the bounding box.
The point p6 is within the bounding box.
```

### Exercise 7

- Given the elevation and land type of a location, write a program that determines if it is suitable for constructing a building. A location is suitable if the elevation is below 3000 meters and the land type is not 'Wetland' or 'Forest'.

Please note these are synthesized criteria and may not be accurate.

```
locations = {
    'location1': {'elevation': 2000, 'land_type': 'Grassland'},  #
Suitable for construction
    'location2': {'elevation': 4000, 'land_type': 'Mountain'},   #
Not suitable (high elevation)
    'location3': {'elevation': 1500, 'land_type': 'Forest'},     #
Not suitable (land type is Forest)
    'location4': {'elevation': 1000, 'land_type': 'Plains'}      #
Suitable for construction
}
```

- This is a nested dictionary. You can have access to each dictionary inside `locations` by looping through `location.items()`, which returns the dictionary's key-value pairs (in this case the inside dictionary).

Expected outcome that should be printed in the terminal:

```
location1 is suitable for construction.
location2 is not suitable for construction.
location3 is not suitable for construction.
location4 is suitable for construction.
```

In [15]:
```
# add your code here

locations = {
    'location1': {'elevation': 2000, 'land_type': 'Grassland'},  # Suitable for constr
    'location2': {'elevation': 4000, 'land_type': 'Mountain'},   # Not suitable (high
    'location3': {'elevation': 1500, 'land_type': 'Forest'},     # Not suitable (land
    'location4': {'elevation': 1000, 'land_type': 'Plains'}      # Suitable for constr
}

# Loop through each location
for loc_name, loc_info in locations.items():
    elevation = loc_info['elevation']
    land_type = loc_info['land_type']

    # Check if location is suitable: elevation < 3000 AND land_type not 'Wetland' or '
    if elevation < 3000 and land_type not in ['Wetland', 'Forest']:
        print(f"{loc_name} is suitable for construction.")
    else:
        print(f"{loc_name} is not suitable for construction.")
```

```
location1 is suitable for construction.
location2 is not suitable for construction.
location3 is not suitable for construction.
location4 is suitable for construction.
```

**Exercise 8**

- Write a program that determines if a route is optimal for a delivery service. A route is considered optimal if it is either shorter than 10 kilometers or if it has low traffic. In addition, the route will not be considered optimal if it passes through a restricted area. In other words, for a route to be deemed optimal, it must be either short (less than 10 km) or have low traffic. However, if the route passes through a restricted area, it negates these conditions, and the route is no longer considered optimal regardless of its length or traffic conditions.
- If a route is suitable, your code should print "Route is optimal." Otherwise, it should print "Route is not optimal."
- Check your program for the following route:
  ```
  route_length = 9.5 # in kilometers
  traffic_condition = 'Low'
  passes_through_restricted_area = False
  ```

In [16]:
```
# write your code here

route_length = 9.5 # in kilometers
traffic_condition = 'Low'
passes_through_restricted_area = False

# A route is optimal if:
```

```
# 1. It is either shorter than 10 km OR has low traffic
# 2. AND it does NOT pass through a restricted area

# Check if route meets the basic conditions (short OR low traffic)
is_short_or_low_traffic = (route_length < 10) or (traffic_condition == 'Low')

# Route is optimal only if it meets basic conditions AND doesn't pass through restrict
if is_short_or_low_traffic and not passes_through_restricted_area:
    print("Route is optimal.")
else:
    print("Route is not optimal.")
```

```
Route is optimal.
```

**Exercise 9**

**Temperature Anomaly Detection**

- Given a NumPy array representing yearly average temperatures, write a program that identifies years with temperature anomalies. In this case, an anomaly is defined as a temperature that is more than one standard deviations away from the mean.
- Print the following in the terminal:

```
- the temperature range that is not considered anomaly
- temperature mean
- temperature standard deviation
- anomalies outside of the range
```

- hint: Compute the mean and standard deviation using NumPy's `mean` and `std` functions, and then use `numpy.where()` (or np.where) to identify the years where the temperature falls outside the specified range.

In [17]:

```
# write your code here

# Example temperature data (yearly average temperatures)
temperatures = np.array([15.2, 16.1, 14.8, 17.3, 18.5, 16.9, 19.2, 15.7, 20.1, 14.5, 1

# Calculate mean and standard deviation
temp_mean = np.mean(temperatures)
temp_std = np.std(temperatures)

# Temperature range that is NOT considered anomaly (within 1 standard deviation)
lower_bound = temp_mean - temp_std
upper_bound = temp_mean + temp_std

print(f"Temperature range that is not considered anomaly: {lower_bound:.2f} to {upper_
print(f"Temperature mean: {temp_mean:.2f}")
print(f"Temperature standard deviation: {temp_std:.2f}")

# Find anomalies (temperatures outside the range)
anomaly_indices = np.where((temperatures < lower_bound) | (temperatures > upper_bound)
anomalies = temperatures[anomaly_indices]

print(f"\nAnomalies outside of the range: {anomalies}")
print(f"Anomaly years (indices): {anomaly_indices}")
```

```
Temperature range that is not considered anomaly: 15.35 to 18.83
Temperature mean: 17.09
Temperature standard deviation: 1.74

Anomalies outside of the range: [15.2 14.8 19.2 20.1 14.5 19.6]
Anomaly years (indices): [ 0  2  6  8  9 14]
```

**Exercise 10**

- We have several numpy array, each representing the monthly average temperatures (in Celsius) for a year in different California cities. The arrays are named after the cities (e.g., los_angeles, san_francisco). Write a program to:
    - Convert these arrays into one NumPy arrays. This will be a 2D array in which rows represent temp of cities and columns represent temp in each month.
    - Calculate the annual average temperature for each city.
    - Identify the city with the highest annual average temperature. Please note the program should do this, not the user.
    - For this city, find the month with the maximum temperature, assuming the months are ordered from January to December.

Hints:

- create a list of cities, `cities = ["Los Angeles", ...]`
- create a list of months, `months = ['January', 'February', ....]`
- once you create a 2D array, then take the mean along the row axis using `numpy.mean(a, axis=None, dtype=None, out=None, keepdims=<no value>, *, where=<no value>)` and assign it to a variable called `temp_mean`. This should return four values, each is the mean for our cities.
- Find the index of the maximum mean temperature in `temp_mean` using `np.argmax()`. With this index, you can find which city in your city list has the highest temperature and what the average tempreatuer is for this city.
- Using this index, you can slice the temperature of all 12 months for this city from your 2D numpy array. Then, find the maximum temperature and its index in this row of values. Using this new index, you can identify which month has the highest temperature. Please note that index 0 refers to January, 1 to February, etc.

```
# City temperature lists

los_angeles = np.array([20, 20, 21, 22, 22, 23, 24, 25, 24, 23, 21,
20])
san_francisco = np.array([15, 16, 16, 17, 17, 18, 18, 19, 19, 18, 17,
16])
san_diego = np.array([19, 19, 20, 21, 21, 22, 23, 24, 23, 22, 21,
20])
sacramento = np.array([12, 13, 15, 17, 19, 22, 25, 26, 25, 21, 16,
13])
```

Expected outcome printed in the terminal:

```
The city with the max temperature is Los Angeles with 22.08 degree of
centigrade.
In Los Angeles, the highest temperature is in August.
```

In [18]:
```
# add your code here

# City temperature lists
los_angeles = np.array([20, 20, 21, 22, 22, 23, 24, 25, 24, 23, 21, 20])
san_francisco = np.array([15, 16, 16, 17, 17, 18, 18, 19, 19, 18, 17, 16])
san_diego = np.array([19, 19, 20, 21, 21, 22, 23, 24, 23, 22, 21, 20])
sacramento = np.array([12, 13, 15, 17, 19, 22, 25, 26, 25, 21, 16, 13])

# Create lists of cities and months
cities = ["Los Angeles", "San Francisco", "San Diego", "Sacramento"]
months = ['January', 'February', 'March', 'April', 'May', 'June',
          'July', 'August', 'September', 'October', 'November', 'December']

# Convert arrays into one 2D NumPy array (rows = cities, columns = months)
temp_array = np.array([los_angeles, san_francisco, san_diego, sacramento])

# Calculate annual average temperature for each city (mean along column axis, axis=1)
temp_mean = np.mean(temp_array, axis=1)

# Find the index of the city with the highest average temperature
max_temp_index = np.argmax(temp_mean)
max_city = cities[max_temp_index]
max_avg_temp = temp_mean[max_temp_index]

print(f"The city with the max temperature is {max_city} with {max_avg_temp:.2f} degree

# Get the temperature data for the city with max temperature
city_temp_data = temp_array[max_temp_index]

# Find the month with the maximum temperature for this city
max_month_index = np.argmax(city_temp_data)
max_month = months[max_month_index]

print(f"In {max_city}, the highest temperature is in {max_month}.")
```

```
The city with the max temperature is Los Angeles with 22.08 degree of centigrade.
In Los Angeles, the highest temperature is in August.
```

## Exercise 11

### "FizzBuzz Interview Question"

Write a program that prints the numbers from 1 to 50. For multiples of three, instead of the number, print "Fizz". For multiples of five, print "Buzz". If a number is a multiple of both three and five, print "FizzBuzz".

- If the number is a multiple of 3 the output should be "Fizz" .
- If the number given is a multiple of 5, the output should be "Buzz".
- If the number given is a multiple of both 3 and 5, the output should be "FizzBuzz"
- If the number is not a multiple of either 3 or 5, the number should be output on its own as shown in the examples below.

Your task is to write the code that accomplishes this and prints the desired output. An additional challenge is to write the program with the least number of lines.

**Example**

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz

# we can write this in one line of code altough it's a bit long. Here
is the example:

print("\n".join(["FizzBuzz" if x%3==0 and x%5 == 0 else "Fizz" if
x%3==0 else "Buzz" if x%5==0 else str(x) for x in range(1,51)]))

# or instead of having two conditional statement (x%3==0 and x%5 ==
0), you can use x%15==0, which does the same thing!

print("\n".join(["FizzBuzz" if x%15==0 else "Fizz" if x%3==0 else
"Buzz" if x%5==0 else str(x) for x in range(1,51)]))

# please pay attention to the other of conditional statements. What
happen if we start from checking x%3==0?

 "\n".join(some_sequence) takes all the elements in some_sequence
(where some_sequence is an iterable of strings), and concatenates
them into a single string, inserting a newline character between each
element. But you don't need to use this as you loop through all
numbers and a simple print statement per each condition should work.
```

In [19]:
```python
# write your code here

# FizzBuzz solution using a loop
for num in range(1, 51):
    if num % 15 == 0:  # Check for multiples of both 3 and 5 first (or use num % 3 ==
        print("FizzBuzz")
    elif num % 3 == 0:  # Check for multiples of 3
        print("Fizz")
    elif num % 5 == 0:  # Check for multiples of 5
        print("Buzz")
    else:
        print(num)

# Alternative one-line solution (commented out):
# print("\n".join(["FizzBuzz" if x%15==0 else "Fizz" if x%3==0 else "Buzz" if x%5==0 e
```

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
19
Buzz
Fizz
22
23
Fizz
Buzz
26
Fizz
28
29
FizzBuzz
31
32
Fizz
34
Buzz
Fizz
37
38
Fizz
Buzz
41
Fizz
43
44
FizzBuzz
46
47
Fizz
49
Buzz
```

### Exercise 12

### breake vs continue

Part A.

- Given a list of cities and a separate list indicating whether each city is coastal, use a loop with a `break` statement to print the name of the first coastal city.

```
cities = ["Sacramento", "Los Angeles", "San Francisco", "Fresno"]
is_coastal = [False, True, True, False]
```

Part B.

- You have a list of environmental readings (temperature, humidity, etc.). Some readings are marked as None to indicate missing data. Iterate through the list,
  - skip the None values using 'continue'.
  - append the not None values into another list to create a cleaned list without missing data.
  - outside of the loop, print your cleaned list which doesn't have None.
  - outside of the loop, print the number of missing data (you need to keep track of the number of times you skip the loop)

```
readings = [20.1, 21.3, None, 19.8, None, 22.5, 23.0, None, 24.5,
18.7, None, 21.9]
```

In [20]:

```
# Example of using break

'''
We have a list of terrain types in a hiking trail. Print each terrain type until you e
'''

terrains = ["forest", "meadow", "forest", "mountain", "plains"]

# Iterate through each terrain in the terrains list
for terrain in terrains:
    if terrain == "mountain":                         # Check if the current terrain is 
        print("Reached a mountain, stopping here.") # Print a message indicating a mou
        break                                       # Break out of the loop, stopping 
    print(f"Terrain: {terrain}")                    # If the current terrain is not 'm
```

```
Terrain: forest
Terrain: meadow
Terrain: forest
Reached a mountain, stopping here.
```

In [21]:

```
# Example for using continue

'''
Given a list of water bodies and their types ('lake' or 'ocean'), print the names of a
'''

water_bodies = ["Tahoe", "Pacific", "Atlantic", "Huron"]
body_types = ["lake", "ocean", "ocean", "lake"]

# Looping through each index in the range of the length of the water_bodies list
for i in range(len(body_types)):                # looping through all indecis for boc
```

```
    if body_types[i] == "lake":
        continue                                    # If it is 'lake', skip the rest of t
    print(f"{water_bodies[i]} is an ocean.")      # If the body type is not 'lake', pri
```

```
Pacific is an ocean.
Atlantic is an ocean.
```

In [22]:

```
# Part A. write your code
# write your code here

cities = ["Sacramento", "Los Angeles", "San Francisco", "Fresno"]
is_coastal = [False, True, True, False]

# Loop through cities and find the first coastal city
for i in range(len(cities)):
    if is_coastal[i]:
        print(f"The first coastal city is {cities[i]}.")
        break  # Stop the loop once we find the first coastal city
```

```
The first coastal city is Los Angeles.
```

In [23]:

```
# Part B. write your code

readings = [20.1, 21.3, None, 19.8, None, 22.5, 23.0, None, 24.5, 18.7, None, 21.9]

# Initialize cleaned list and counter for missing data
cleaned_readings = []
missing_count = 0

# Iterate through the readings
for reading in readings:
    if reading is None:  # Check if reading is None (missing data)
        missing_count += 1  # Increment missing data counter
        continue  # Skip this iteration and move to the next
    cleaned_readings.append(reading)  # Append non-None values to cleaned list

# Print results outside the loop
print("Cleaned list (without None):", cleaned_readings)
print("Number of missing data:", missing_count)
```

```
Cleaned list (without None): [20.1, 21.3, 19.8, 22.5, 23.0, 24.5, 18.7, 21.9]
Number of missing data: 4
```

In [ ]: