# EX1:

The bug in the `id_to_fruit` function is that it likely returns the fruit name at an incorrect index because sets in Python are zero-indexed.

Here's the breakdown of the problem:

- **Sets are zero-indexed:** When you access elements in a set using an index, Python starts counting from 0, not 1. So, the first element in the set has index 0, the second element has index 1, and so on.
- **Function assumes 1-based indexing:** The provided code snippet expects the `fruit_id` to correspond to the actual position of the fruit in the set (`"orange"` at index 1, `"kiwi"` at index 3, etc.).

This mismatch between assumptions and how sets work in Python leads to the wrong fruit being returned.

Here's how to fix it:

Python
```
def id_to_fruit(fruit_id, fruits):
  """
  Returns the fruit name by getting the string at the correct index of the
set.

  Args:
      fruit_id (int): The id of the fruit to get (1-based indexing).
      fruits (Set[str]): The set of fruits to choose the id from.

  Returns:
      str: The string corresponding to the index fruit_id.
  """

  # Subtract 1 from fruit_id to get the correct zero-based index for the
set
  return list(fruits)[fruit_id - 1]
```

By subtracting 1 from `fruit_id`, we convert it to a zero-based index that aligns with how Python handles sets. This ensures that the function returns the fruit corresponding to the provided `fruit_id`.

Now, running the example with the fix:

```
>>> name1 = id_to_fruit(1, {"apple", "orange", "melon", "kiwi",
"strawberry"})
>>> name3 = id_to_fruit(3, {"apple", "orange", "melon", "kiwi",
"strawberry"})
>>> name4 = id_to_fruit(4, {"apple", "orange", "melon", "kiwi",
"strawberry"})

>>> print(name1)  # Output: orange
>>> print(name3)  # Output: kiwi
>>> print(name4)  # Output: strawberry
```

This should print the correct fruits as expected: "orange", "kiwi", and "strawberry".

## EX2:

There are two bugs in the provided `swap` function:

1. **Missing copy:** The first bug is that the function doesn't create a copy of the input array before modifying it. When you modify the elements of the original `coords` array, you're also modifying the `swapped_coords` array since they both point to the same memory location.
2. **Incorrect swapping:** The second bug is in how the swapping is done. Slicing with a step of 2 (`[:, [1, 0, 2, 3, 4]]`) creates a view of the original array with columns reordered, but it doesn't swap the elements within each row.

Here's how to fix the code:

Python
```python
import numpy as np

def swap(coords):
  """
```

```
    Swaps the x and y coordinates in a numpy array of bounding boxes.

    Args:
        coords (ndarray): A numpy array of bounding box coordinates.

    Returns:
        ndarray: A new array with the x and y coordinates flipped.
    """

    # Create a copy of the input array to avoid modifying the original
    swapped_coords = coords.copy()

    # Swap x and y coordinates within each row
    swapped_coords[:, [0, 1]] = swapped_coords[:, [1, 0]]

    return swapped_coords
```

Explanation of the fix:

- **`coords.copy()`:** This line creates a new copy of the `coords` array. Now, modifications to `swapped_coords` won't affect the original `coords` array.
- **`swapped_coords[:, [0, 1]] = swapped_coords[:, [1, 0]]`:** This line swaps the first two columns (0 and 1) within each row of the `swapped_coords` array. This effectively swaps the x and y coordinates as desired.

With these changes, the `swap` function will correctly create a new array with flipped x and y coordinates in each row, without modifying the original input array.

---

EX3 :

The bug in the `plot_data` function is likely related to how it interprets the data from the CSV file. Here's the potential problem and a fix:

**Problem:** The code writes the header row ("precision", "recall") but might not be correctly parsing it when plotting. Plotting libraries often treat the first row (excluding the header) as the actual data points.

**Fix is to Skip the header row when reading the data**

Python

```
import csv
import matplotlib.pyplot as plt  # Assuming matplotlib is used for
plotting

def plot_data(csv_file_path):
  """
  Plots the precision-recall curve based on data from a .csv file.

  Args:
      csv_file_path (str): The CSV file containing the data to plot.
  """

  precision = []
  recall = []

  with open(csv_file_path, 'r') as f:
    reader = csv.reader(f)
    # Skip the header row
    next(reader)
    for row in reader:
      precision.append(float(row[0]))
      recall.append(float(row[1]))

  # Plot the data
  plt.plot(precision, recall)
  plt.xlabel("Precision")
  plt.ylabel("Recall")
  plt.title("Precision-Recall Curve")
  plt.show()

# Example usage (assuming data_file.csv exists)
plot_data('data_file.csv')
```

This approach skips the first row (header) using `next(reader)` and then reads the remaining rows, converting them to floats before appending to separate lists (`precision` and `recall`).