# Passing collection to a function

In Python, you can pass collections (such as lists, tuples, dictionaries, etc.) to functions as arguments. This allows you to operate on entire sets of data within a function, making your code more modular and flexible. Let's explore how to pass different types of collections to a function with examples:

### Passing Lists:

```python
def process_numbers(numbers):
    for num in numbers:
        print(num * 2)

# Passing a list to the function
my_numbers = [1, 2, 3, 4, 5]
process_numbers(my_numbers)
```

In this example, the `process_numbers` function takes a list of numbers as an argument and prints each number multiplied by 2. You can pass any list of numbers to this function.

### Passing Tuples:

```python
def display_info(person_info):
    for key, value in person_info.items():
```

```
    print(f"{key}: {value}")

# Passing a tuple (dictionary) to the function
person_data = {"name": "Alice", "age": 30, "city": "Wonderland"}
display_info(person_data)
```

Here, the `display_info` function takes a dictionary (which is similar to a tuple in this context) representing information about a person and prints each key-value pair. You can pass any dictionary-like structure to this function.

### Passing Sets:

```python
def process_unique_items(unique_items):
    for item in unique_items:
        print(f"Processing: {item}")

# Passing a set to the function
unique_items_set = {"apple", "banana", "orange"}
process_unique_items(unique_items_set)
```

In this example, the `process_unique_items` function takes a set of unique items as an argument and prints each item. You can pass any set to this function.

### Passing Multiple Collections:

```python
def combine_lists(list1, list2):
    combined_list = list1 + list2
    print(combined_list)

# Passing two lists to the function
numbers1 = [1, 2, 3]
numbers2 = [4, 5, 6]
combine_lists(numbers1, numbers2)
```

Here, the `combine_lists` function takes two lists as arguments and concatenates them. You can pass any two lists to this function.

### Unpacking Collections:

You can use the `*` operator to unpack elements of a collection and pass them as separate arguments to a function.

```python
def print_elements(element1, element2, element3):
    print(element1, element2, element3)

# Unpacking a tuple and passing its elements to the function
my_tuple = (1, 2, 3)
```

```
print_elements(*my_tuple)
```

In this example, the `*my_tuple` syntax unpacks the elements of the tuple, and each element is passed as a separate argument to the `print_elements` function.

By understanding how to pass collections to functions, you can create more reusable and versatile code that can handle different sets of data.