1) **HashSet:** A hash-based set that stores unique elements. It does not guarantee any order of elements. Provides fast operations: add, remove, contains (typically O(1)).

Ex:
```
import 'dart:collection';
 void main()
{
   var hashSet = HashSet();
   hashSet.addAll([3, 1, 2]);
   print(hashSet); // {1, 2, 3}
}
```

2) **LinkedHashSet:** A hash-based set that maintains insertion order. Elements are iterated in the order they were added. Fast operations similar to HashSet.

Ex:
```
import 'dart:collection';
 void main()
{
  var linkedHashSet = LinkedHashSet();
  linkedHashSet.addAll([3, 1, 2]);
  print(linkedHashSet); // {3, 1, 2}
}
```

3) **SplayTreeSet:** A set based on a self-balancing binary search tree. Elements are always sorted. Operations like add, remove, contains take O(log n) time.

Ex:
```
import 'dart:collection';
 void main()
{
   var SplayTreeSet = SplayTreeSet();
   SplayTreeSet.addAll([3, 0, 21]);
   print(SplayTreeSet); // {0,3,21}
}
```

Set Creation:

1) Using set literal:

Ex: void main()

```
{
    Set<int> a={1,2,3,4,5};
    print(a); //{1,2,3,4,5}
}
```

2) Using set constructor:

Ex: void main()

```
{
   Set<int> b=set();
   b.add(1);
   b.add(2);
   Print(b);  //{1,2}
```

3) Using Set.from():

Ex: void main()

```
{
  var list = [1, 2, 2, 3];
  var n = Set.from(list);
  print(n); // {1, 2, 3}
}
```

Map: Collection of key-value pairs.

Map Creation:

1) Empty map: A map with no key-value pairs. Created using either a literal {} or the Map() constructor.

Ex: void main()

```
{
     var emptyMap = <String, int>{};
    print(emptyMap); // {}
```

```
}
```

2) Map Literal: Define a map by directly specifying key-value pairs inside {}. Keys and values can be any types, and Dart infers the types if you don't specify.

Ex:
```
void main()
{
    var mapLit = <String,int>{'apple': 3, 'banana': 5};
    print(mapLit); // {apple: 3, banana: 5}
}
```

3) Map.from():Creates a new map by copying key-value pairs from an existing map. Useful to clone or create a map from another.

Ex:
```
void main()
{
    var or = {'x': 10, 'y': 20};
    var cop = Map<String, int>.from(or);
    print(cop); // {x: 10, y: 20}
}
```

4) Map.of(): Creates a new map by copying the key-value pairs from **any other Map** (similar to Map.from()). The difference is mostly semantic; Map.of() expects a Map argument and is often preferred for readability.

Ex:
```
void main()
{
    var original = {'a': 1, 'b': 2};
    var copy = Map.of(original);
    print(copy); // {a: 1, b: 2}
}
```

5) Map.fromEntries(): Creates a new map from a list of key-value MapEntry objects.

Ex:
```
void main()
```

```
 {
    var entries = [
    MapEntry('apple', 10),
    MapEntry('banana', 20),
    MapEntry('orange', 30),
         ];
    var fruitPrices = Map.fromEntries(entries);
    print(fruitPrices); //  {apple: 10, banana: 20, orange: 30}
}
```