Understanding Basic Collections

Zachary Bennett
Lead Software Developer

@z_bennett_ | zachbennettcodes.com

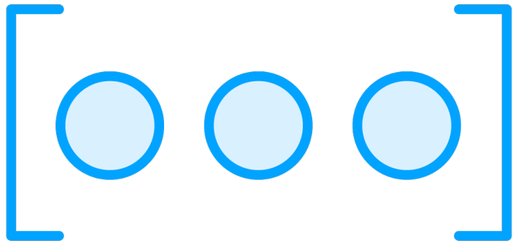
Major Collection Categories

Sequences

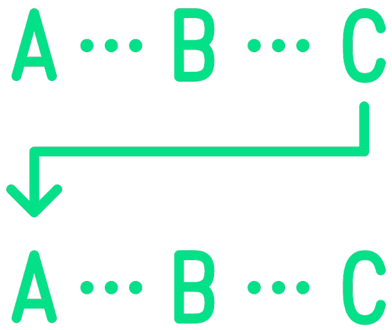
Maps

Sets

Sequence Types



Vec – a generic and growable sequence of data

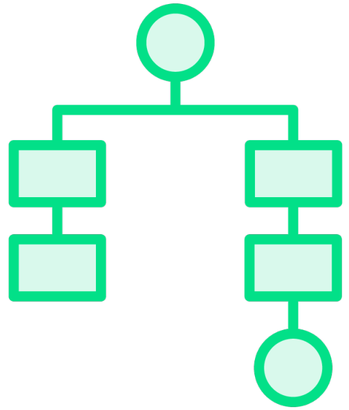


VecDeque – a double-ended queue

LinkedList – a doubly-linked list containing a series of nodes

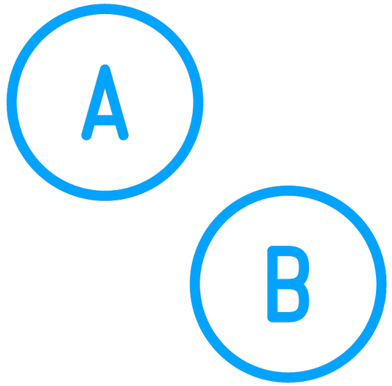
Map Types

HashMap
A key-value store

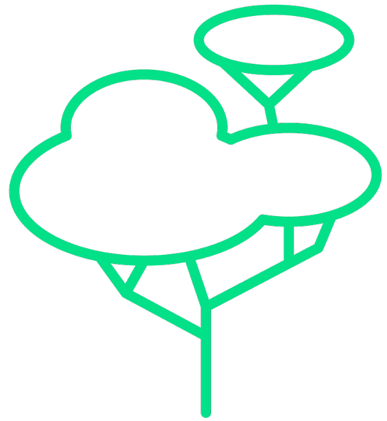


BTreeMap
A map optimized for search

Set Types



HashSet
A set containing no duplicates implemented as a HashMap



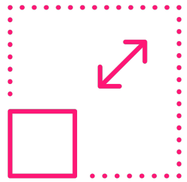
BTreeSet
An ordered set – optimized for search

Fundamental Collections

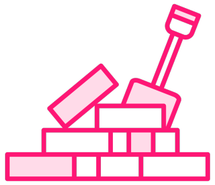
Vec

HashMap

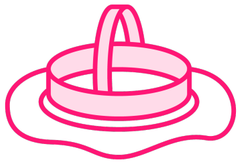
Vec Key Features



Size can grow at runtime



Memory is allocated on the heap



Generic, homogenous data structure

Vec

```
// Creating vectors
let mut vec_a: Vec<i32> = Vec::new();

let vec_b = vec![1, 2, 3];

let vec_c = Vec::from([1, 2, 3]);
```

- Stores elements by key-value pairs
- Fast access of elements
- Great for in-memory caches
- Stores data on the heap

HashMap

```
// Creating hashmaps
let mut map_a: HashMap<&str, i32> = HashMap::new();

let map_b = HashMap::from([(“Hello”, 1)]);
```

Demo

Creating and using Vecs

Creating and using HashMaps

Iterators

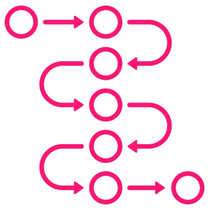
Iterator

An iterator is an abstraction around accessing elements in a collection.

Iterator Key Points

Iterators are lazy

Many collections like Vec use iterators



You can create your own iterators via the Iterator trait

Implicit Iterators

```
let my_nums = vec![1, 2, 3];
for num in my_nums {
    println!(“ Number: {}”, num);
}

// Behind the scenes, the code above is using iterators implicitly
// The “behind-the-scenes” code might look like the following
let my_nums = vec![1, 2, 3];

let nums_iterator = my_nums.iter();
for num in nums_iterator {
    println!(“ Number: {}”, num);
}
```

The Laziness of Iterators

```
let my_nums = vec![1, 2, 3];

// This doesn't really do anything until you start consuming the iterator
let nums_iterator = my_nums.iter();
```

```
let my_nums = vec![1, 2, 3];
for num in my_nums {
    println!(“ Number: {}”, num);
}

let my_map = HashMap::from([(“key1”, 1)]);
for (key, value) in my_map {
    println!(“ Number: {}”, value);
}
```

Custom Iterators

You can create your own iterators by implementing the Iterator trait that Rust provides. These can be used to create custom iterators for your own collections.

For example, the “for/in” syntax can be used on both Vec and HashMap in the same way although the internals of how these iterations happen look different.

Demo

Iterator basics

- Using iterators