Common Collections

Vector

Allows you to store variable number of values next to each other

To create an empty vector you call the `Vec::new` function

```
let v: Vec<i32> = Vec::new();
```

Since we are not inserting any initial values into the vector we will have to provide type annotations otherwise Rust doesn't know what type of vector this is for. There is the `vec!` macro that will create a new vector that holds the values you give it:

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

So you would rarely need to do the type annotation yourself.

Updating a vector

To add elements to the vector you would use the `push` method:

```
let mut v = Vec::new();

v.push(5);
v.push(6);
```

Reading elements of vectors

Two ways of getting elements out of the vector, via indexing or using the `get` method.

```
let v = vec![1, 2, 3, 4, 5];

let ele = third: &i32 = &v[2]; // indexing
let ele: Option<&i32> = v.get(2); // get method

match ele {
Some(num) => println!("The number is {num}"),
```

```
    None => println!("It doesn't exist"),
}
```

Iterating over a vector

This is with immutable reference

```
let v = vec![100, 32, 57];
for i in &v {
    println!("{i}");
}
```

This is with mutable reference, in order to change the value of that mutable referene you have to use `*` dereference operator to get the value in `i` before the `+=` operator.

```
let mut v = vec![100, 32, 57];
for i in &mut v {
    *i += 50;
}
```

String

A collection of characters, they are stored on the heap. This is different than String literals which is `str` or `&str` slice type.

Hash map

Allows you to associate a value with a key.

Revision #1
Created 30 January 2023 20:45:43 by Tamarine
Updated 30 January 2023 21:56:44 by Tamarine