컬렉션

2020.3

일반적 컬렉션

- Vector
- String
- HashMap
- heap에 저장되므로 run-time 에 추가/변경/삭제가 가능하다.

벡터

```rust
let v: Vec<i32> = Vec::new();
let v = vec![1, 2, 3];

let mut v = Vec::new();
v.push(5);
v.push(6);
v.push(7);
```

```rust
{
    let v = vec![1, 2, 3, 4];
    // v를 가지고 뭔가 합니다
} // <- v가 스코프 밖으로 벗어났고, 여기서 해제됩니다. heap에서도 해제됩니다.

let v = vec![1, 2, 3, 4, 5];
let does_not_exist = &v[100]; // panic
let does_not_exist = v.get(100); // Some(_) 또는 None

let mut v = vec![1, 2, 3, 4, 5];
let first = &v[0];
v.push(6);
```

벡터와 열거형

벡터는 같은 타입만 저장 가능, 다양한 타입을 열거형으로 묶어서 저장.

```
enum SpreadsheetCell { // 힙 메모리가 얼만큼 필요한지 컴파일 타임에 결정이 가능하므로
    Int(i32), //  새로운 엘리먼트를 추가하는 경우, 얼만큼 필요한지 알아야 함.
    Float(f64), // 10번째 엘리먼트를 꺼내 오는데, 엘리먼트 크기가 가변적인 경우 ???
    Text(String),
}

let row = vec![
    SpreadsheetCell::Int(3),
    SpreadsheetCell::Text(String::from("blue")),
    SpreadsheetCell::Float(10.12),
];
```

Trait object : 어떤 타입을 담을지 예상할 수 없을 때, 라이브러리를 작성하는 경우

스트링 ~ 1

- String, &str,
- OsString, OsStr, CString, CStr

```
let mut s = String::from("foo");
s.push_str(“bar”); // 파라메터의 소유권을 가져오지 않음.

let mut s = String::from("lo");
s.push(‘l'); // 한글자를 추가함.

let s1 = String::from("Hello, ");
let s2 = String::from("world!");
let s3 = s1 + &s2; // s1은 여기서 이동되어 더이상 쓸 수 없음을 유의하세요
```

스트링~2

fn add(self, s: &str) -> String { // self의 소유권을 소비하고 거기에 문자열을 추가해서 그 소유권을 리턴함. 복사본을 만드는 것이 아님.

let len = String::from(“Hola").len(); // 담고 있는 벡터의 바이트 수.

let s1 = String::from("hello");
let h = s1[0]; // 허용되지 않음. 컴파일 에러.

let hello = “Здравствуйте";
let s = &hello[0..4]; // 각 글자가 2바이트인 경우임.

&hello[0..1] : thread 'main' panicked at 'index 0 and/or 1 in `Здравствуйте` do not lie on character boundary', ../src/libcore/str/mod.rs:1694

```
for c in "नमस्ते".chars() {
    println!("{}", c);
}
```

```
for b in "नमस्ते".bytes() {
    println!("{}", b);
}
```

해쉬 맵 ~ 1

```
use std::collections::HashMap;
let mut scores = HashMap::new(); // 저장될 타입은 추론됨.
scores.insert(String::from("Blue"), 10);
scores.insert(String::from("Yellow"), 50);

use std::collections::HashMap;
let field_name = String::from("Favorite color");
let field_value = String::from("Blue");
let mut map = HashMap::new();
map.insert(field_name, field_value);
// field_name과 field_value은 이 지점부터 유효하지 않습니다. 소유권이 이동했으므로,

scores.insert(String::from("Blue"), 10);
scores.insert(String::from("Yellow"), 50);
let team_name = String::from("Blue");
let score = scores.get(&team_name); // Option<&V>를 반환함.
```

해쉬 맵~2

```
scores.insert(String::from("Blue"), 10);
scores.insert(String::from("Blue"), 25); // 덮어 쓰기

scores.insert(String::from("Blue"), 10);
scores.entry(String::from(“Yellow")).or_insert(50); // 없으므로 삽입
scores.entry(String::from(“Blue")).or_insert(50); // 있으므로 무시함
```

```
let text = "hello world wonderful world";

let mut map = HashMap::new();

for word in text.split_whitespace() {
    let count = map.entry(word).or_insert(0);
    *count += 1;
}

println!("{:?}", map);
```

```
{"world": 2, "hello": 1, "wonderful": 1}
```