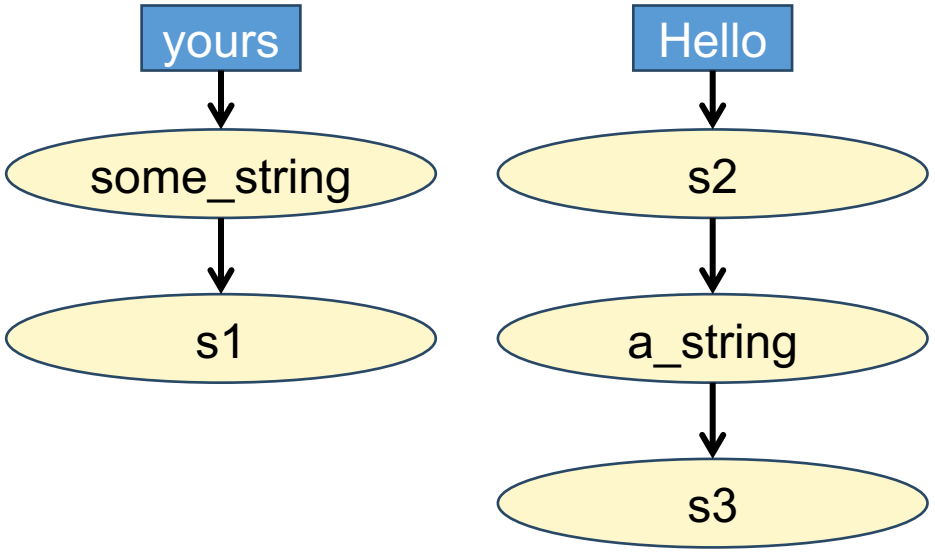
러스트 소유권
참조와 대여 / 슬라이스

https://youtu.be/qLJw9hb3Fts

1. 소유권 리뷰

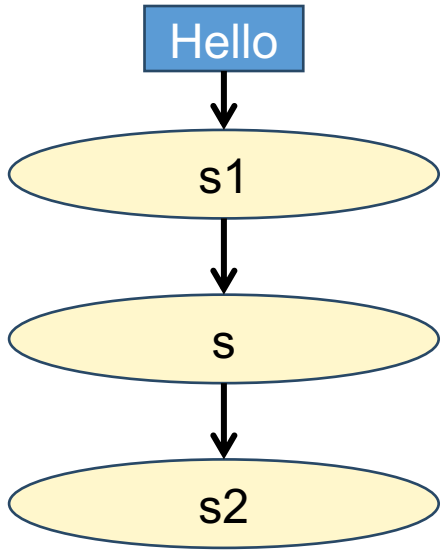
```
fn main() {
    let s1 = gives_ownership();         // gives_ownership이 자신의 반환 값을 s1로
                                        // 이동시킵니다

    let s2 = String::from("hello");     // s2가 스코프 안으로 들어옵니다

    let s3 = takes_and_gives_back(s2);  // s2는 takes_and_gives_back로 이동되는데,
                                        // 이 함수 또한 자신의 반환 값을 s3로
                                        // 이동시킵니다
} // 여기서 s3가 스코프 밖으로 벗어나면서 버려집니다. s2는 이동되어서 아무 일도
  // 일어나지 않습니다. s1은 스코프 밖으로 벗어나고 버려집니다.

fn gives_ownership() -> String {            // gives_ownership은 자신의 반환 값을
                                            // 자신의 호출자 함수로 이동시킬
                                            // 것입니다

    let some_string = String::from("yours"); // some_string이 스코프 안으로 들어옵니다

    some_string                             // some_string이 반환되고
                                            // 호출자 함수 쪽으로
                                            // 이동합니다
}

// 이 함수는 String을 취하고 같은 것을 반환합니다
fn takes_and_gives_back(a_string: String) -> String { // a_string이 스코프 안으로
                                                      // 들어옵니다

    a_string  // a_string이 반환되고 호출자 함수 쪽으로 이동합니다
}
```



1. 소유권 리뷰

```rust
fn main() {
    let s1 = gives_ownership();         // gives_ownership이 자신의 반환 값을 s1로
                                        // 이동시킵니다

    let s2 = String::from("hello");     // s2가 스코프 안으로 들어옵니다

    let s3 = takes_and_gives_back(s2);  // s2는 takes_and_gives_back로 이동되는데,
                                        // 이 함수 또한 자신의 반환 값을 s3로
                                        // 이동시킵니다
} // 여기서 s3가 스코프 밖으로 벗어나면서 버려집니다. s2는 이동되어서 아무 일도
  // 일어나지 않습니다. s1은 스코프 밖으로 벗어나고 버려집니다.

fn gives_ownership() -> String {            // gives_ownership은 자신의 반환 값을
                                            // 자신의 호출자 함수로 이동시킬
                                            // 것입니다

    let some_string = String::from("yours"); // some_string이 스코프 안으로 들어옵니다

    some_string                             // some_string이 반환되고
                                            // 호출자 함수 쪽으로
                                            // 이동합니다
}

// 이 함수는 String을 취하고 같은 것을 반환합니다
fn takes_and_gives_back(a_string: String) -> String { // a_string이 스코프 안으로
                                                      // 들어옵니다

    a_string  // a_string이 반환되고 호출자 함수 쪽으로 이동합니다
}
```

```rust
fn main() {
    let s1: String = String::from("hello");
    let s2: String = temp_function(s1);

    println!("s1 = {}", s1);
    println!("s2 = {}", s2);
}

fn temp_function(s: String) -> String{
    s
}
```



2. 참조와 대여

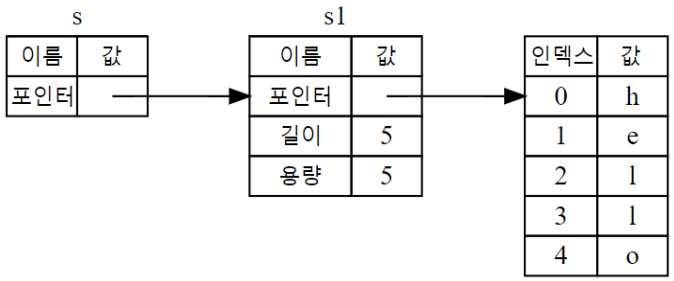
- 참조
  - 참조는 변수가 메모리에 저장된 값에 대한 접근 권한을 가지고 있지만, 소유권은 가지고 있지 않을 때 사용됨
  - 참조를 사용하면 데이터를 직접적으로 이동시키지 않고도 읽거나 수정할 수 있음
  - 불변 참조와 가변 참조
    - 불변 참조(&T) : 불변 참조는 데이터를 읽기만 가능
    - 가변 참조(&mut T) : 가변 참조는 데이터의 수정도 가능, 특정 데이터에 대해 오직 하나의 가변 참조만을 허용

- 대여
  - 참조자를 만드는 행위를 대여(Borrow)라고 한다.

2. 참조와 대여

- 참조
  - 불변 참조(&T) : 불변 참조는 데이터를 읽기만 가능

```rust
fn main() {
    let s1: String = String::from("hello");
    str_info(&s1);

    println!("s1 = {}", s1);
    //println!("s2 = {}", s2);
}

fn str_info(s: &String){
    str_len(s);
}

fn str_len(s: &String) {
    println!("s len = {}", s.len());
}
```




```rust
fn main() {
    let s = String::from("hello");

    change(&s);
}

fn change(some_string: &String) {
    some_string.push_str(", world");
}
```

```
$ cargo run
   Compiling ownership v0.1.0 (file:///projects/ownership)
error[E0596]: cannot borrow `*some_string` as mutable, as it is behind a `&
 --> src/main.rs:8:5
  |
7 | fn change(some_string: &String) {
  |                        ------- help: consider changing this to be a mut
8 |     some_string.push_str(", world");
  |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ `some_string` is a `&` reference, s

For more information about this error, try `rustc --explain E0596`.
error: could not compile `ownership` due to previous error
```

2. 참조와 대여

- 참조
  - 가변 참조(&mut T) : 가변 참조는 데이터의 수정도 가능, 특정 데이터에 대해 오직 하나의 가변 참조만을 허용

```
let mut s = String::from("hello");

let r1 = &mut s;
let r2 = &mut s;

println!("{}, {}", r1, r2);
```

```
fn main() {
    let mut s = String::from("hello");

    change(&mut s);
}

fn change(some_string: &mut String) {
    some_string.push_str(", world");
}
```

```
$ cargo run
   Compiling ownership v0.1.0 (file:///projects/ownership)
error[E0499]: cannot borrow `s` as mutable more than once at a time
 --> src/main.rs:5:14
  |
4 |     let r1 = &mut s;
  |              ------ first mutable borrow occurs here
5 |     let r2 = &mut s;
  |              ^^^^^^ second mutable borrow occurs here
6 |
7 |     println!("{}, {}", r1, r2);
  |                        -- first borrow later used here
```

- 이러한 제약은 데이터 경합을 방지
  - 둘 이상의 포인터가 동시에 같은 데이터에 접근 x

2. 참조와 대여

- 참조
  - 가변 참조(&mut T) : 가변 참조는 데이터의 수정도 가능, 특정 데이터에 대해 오직 하나의 가변 참조만을 허용

```
let mut s = String::from("hello");

{
    let r1 = &mut s;
} // 여기서 r1이 스코프 밖으로 벗어나며, 따라서 아무 문제없이 새 참조자를 만들 수 있습니다.

let r2 = &mut s;
```

```
let mut s = String::from("hello");

let r1 = &s; // 문제없음
let r2 = &s; // 문제없음
let r3 = &mut s; // 큰 문제

println!("{}, {}, and {}", r1, r2, r3);
```

```
fn main() {
    let mut s = String::from("hello");

    let r1 = &s; // 문제없음
    let r2 = &s; // 문제없음
    println!("{} and {}", r1, r2);
    // 이 지점 이후로 변수 r1과 r2는 사용되지 않습니다

    let r3 = &mut s; // 문제없음
    println!("{}", r3);
}
```

2. 참조와 대여

- 댕글링 참조
  - 댕글링 포인터(Dangling pointer)란 어떤 메모리를 가리키는 포인터가 남아있는 상황에서 일부 메모리를 해제해 버림으로써, 다른 개체가 할당 받았을지도 모르는 메모리를 참조하게 된 포인터

```
fn main() {
    let reference_to_nothing = dangle();
}

fn dangle() -> &String {
    let s = String::from("hello");

    &s
}
```

3. 슬라이스

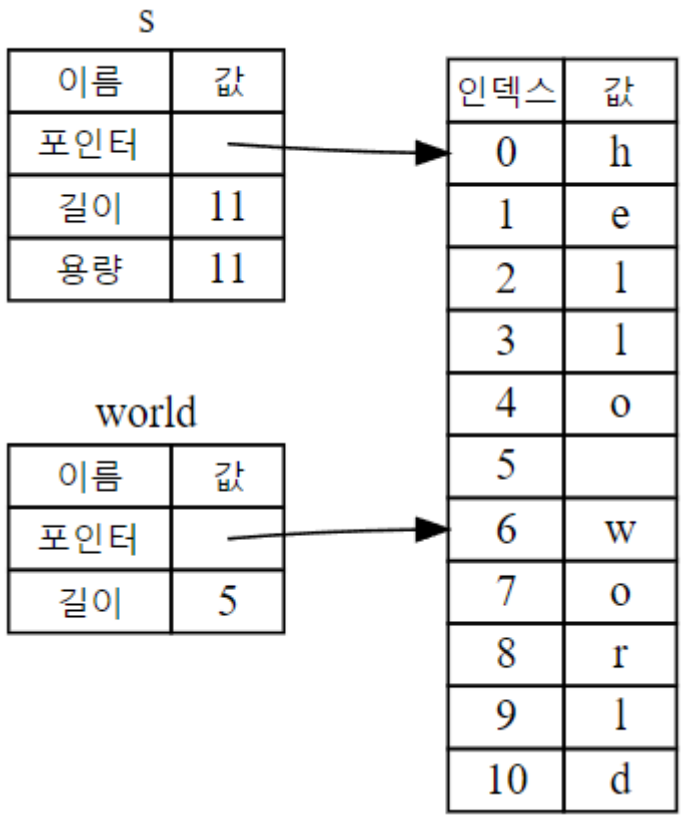
- 슬라이스는 컬렉션을 통째로 참조하는 것이 아닌 컬렛션의 연속된 일련의 요소를 참조.
  - 슬라이스는 참조자의 일종으로 소유권을 갖지 않음.
  - 컬렉션은 여러 값들을 저장할 수 있는 데이터 구조(예 : Vector, String, HashMap, HashSet .)

```
fn first_word(s: &String) -> usize {
    let bytes = s.as_bytes();

    for (i, &item) in bytes.iter().enumerate() {
        if item == b' ' {
            return i;
        }
    }

    s.len()
}
```

3. 슬라이스

- 슬라이스는 컬렉션을 통째로 참조하는 것이 아닌 컬렛션의 연속된 일련의 요소를 참조.

```
let s = String::from("hello world");

let hello = &s[0..5];
let world = &s[6..11];
```



3. 슬라이스

- 슬라이스는 컬렉션을 통째로 참조하는 것이 아닌 컬렉션의 연속된 일련의 요소를 참조.

```
let s = String::from("hello world");

let hello = &s[0..5];
let world = &s[6..11];
```

```
let s = String::from("hello");

let slice = &s[0..2];
let slice = &s[..2];
```

```
let s = String::from("hello");

let len = s.len();

let slice = &s[3..len];
let slice = &s[3..];
```

```
let s = String::from("hello");

let len = s.len();

let slice = &s[0..len];
let slice = &s[..];
```

3. 슬라이스

- 슬라이스는 컬렉션을 통째로 참조하는 것이 아닌 컬렛션의 연속된 일련의 요소를 참조.

```
fn first_word(s: &String) -> usize {
    let bytes = s.as_bytes();

    for (i, &item) in bytes.iter().enumerate() {
        if item == b' ' {
            return i;
        }
    }

    s.len()
}
```

```
fn first_word(s: &String) -> &str {
    let bytes = s.as_bytes();

    for (i, &item) in bytes.iter().enumerate() {
        if item == b' ' {
            return &s[0..i];
        }
    }

    &s[..]
}
```

```
fn main() {
    let mut s = String::from("hello world");

    let word = first_word(&s);

    s.clear(); // 에러!

    println!("the first word is: {}", word);
}
```

감 사 합 니 다