

03. vector #1

```

1 fn main()
2 {
3     let mut a = [1,2,3];
4     let mut v = vec![1,2,3];
5
6     // ❶ 요소 변경 - 배열, vector 모두 가능
7     a[0] = 0; // ok
8     v[0] = 0; // ok
9
10    // ❷ 항목의 추가
11    a.push(20); // error
12    v.push(20); // ok
13
14    // ❸ 크기 변경
15    a.resize(10, 0); // error
16    v.resize(10, 0); // ok
17
18    println!("{:?}", v);
19 }

```

vector

핵심 정리

- array vs vector

	요소 변경 c[0] = 10	요소추가/삭제 c.push(7)	크기 변경 c.resize(10,0)
mutable array	0	X	X
mutable vector	0	0	0

- array vs vector 의 메모리 구조

```

10 // ❷ 항목의 추가
11 // a.push(20); // error
12 v.push(20); // ok
13
14 // ❸ 크기 변경
15 // a.resize(10, 0); // error
16 v.resize(10, 0); // ok
17
18 println!("{:?}", v);
19
20 println!("{:?}", std::mem::size_of_val(&a));
21 println!("{:?}", std::mem::size_of_val(&v));
22
23 println!("{:p}", &a);
24 println!("{:p}, {:p}", &v, v.as_ptr());
25 }

```

vector

핵심 정리

- array vs vector

	요소 변경 c[0] = 10	요소추가/삭제 c.push(7)	크기 변경 c.resize(10,0)
mutable array	0	X	X
mutable vector	0	0	0

- array vs vector 의 메모리 구조

`std::mem::size_of_val(&a)` : 스택에 할당된 a의 배열 크기

`std::mem::size_of_val(&v)` : 스택에 할당된 v의 벡터 정보(구조체?) 크기

`println!("{:p}", &a)` : 스택에 할당된 a의 배열 주소

`println!("{:p}", &v)` : 스택에 할당된 v의 벡터 정보(구조체?) 주소

println!("{:p}", v.as_ptr()) : 스택에 할당된 v의 벡터 정보(구조체?)가 가리키는 힙에 할당된 배열 주소. (위 이미지에서 (*))

vector2.rs

```
2 {
3     // ❶ vector 를 생성하는 방법
4     let v1 : std::vec::Vec<i32> = Vec::new();
5     let v2 : Vec<i32> = Vec::new();
6     // let v3          = Vec::new(); // error
7     let v4            = Vec::<i32>::new();
8     let v5            = Vec::from([1,2,3]);
9     // let v6 : Vec<i32> = [1,2,3].into();
10    let v6 : Vec<_> = [1,2,3].into();
11
12
13    // ❷ vec! 매크로 사용
14    let v7 = vec![5, 6, 7];
15    let mut v8 = vec![5, 6, 7];
16
17 }
18
```

vector

핵심 정리

- vector 의 타입
 - ⇒ `std::vec::Vec<T>`
 - ⇒ generic
 - ⇒ 모듈명 등을 표기하지 않아도 사용가능(`prelude`)
- Rust 에서 “객체를 생성하는 코딩 관례”

primitive type	let n1 = 10; let arr = [1,2,3];
user define type	let s = String::new(); let v = Vec::<i32>::new();

- ⇒ 타입이 가진 **associated function** 을 사용하는 경우가 많음.
- ⇒ `::new()`, `::from()` 등이 관례적으로 널리 사용.
- ⇒ vector 는 Generic 이므로 타입인자 필요