Rust

숫자 관련 타입

길이	Signed	Unsigned
Integer 타입	i8, i16, i32, i64, i128	u8, u16, u32, u64, u128
Floating-point 타입	f32, f64	

```
fn main() {
    let x = 1.0; // f64
    let y: f32 = 10.0; // f32
}
```

```
let sum = 1 + 2;                // 더하기
let difference = 10.1 - 0.2;  // 빼기
let multiplication = 10 * 20; // 곱하기
let division = 50.5 / 30.5;    // 나누기
let remainder = 53 % 5;       // 나머지
```

문자 타입

```
let hello='a';
let hello2: char = 'b';
```

주석

```
//
/* */
//! //
```

조건부 할당

```
let n = 1;
let positive = if n > 0 { true } else { false }; // true
```

상수 타입

```
const ELEVATION: u32 = 8_849; // 상수 변수
println!("{}m", ELEVATION); // 8849m
```

튜플 타입

종류	내용
(String, i32)	서로 다른 타입으로 구성 가능

```
let tuple = ("hello", 3, 'a');
tuple.0      // “hello”
tuple.1      // 3
tuple.2      // a
```

```
let tuple: (i32, f64, u8) = (500, 6.4, 1);
tuple.0      // 500
tuple.1      // 6.4
tuple.2      // 1
```

if 문

```
fn main() {
    let n = 1;
    if n < 0 {
        print!("{}는 음수 입니다.", n);
    } else if n > 0 {
        print!("{}은 양수 입니다.", n); // 출력
    } else {
        print!("{}은 0입니다.", n);
    }
}
```

Enum 타입

```
enum Numbers {
    ONE = 1, TWO = 2,
}
fn main() {
    let one=Numbers::ONE as u8;
    println!("{}", one); // 1
    println!("{}", Numbers::TWO as u8); // 2
}
```

배열 타입

종류	내용
고정 배열	let x: [i32; 5] = [1, 2, 3, 4, 5];
배열(5 size)을 6으로 초기화	let y: [i32; 5] = [6; 5];

```
x[0]          // 1
x[4]          // 5
x[x.len()-1]  // 5
x.len())      // 5
y[0]          // 6
y[4]          //  6
y[y.len()-1]  // 6
y.len()       // 5
```

모듈 상수

```
pub mod media_type {
    pub const JSON: &'static str = "application/json";
    pub const TXT: &'static str = "text/plain";
}
fn main() {
    println!("{}", media_type::JSON); // application/json
    println!("{}", media_type::TXT);  // text/plain
}
```

cargo 명령어 옵션

```
cargo new <패키지명> // 새로운 패키지 생성
cargo check // 현재 패키지 에러 체크
cargo run // 로컬패키지 실행
cargo build // 빌드 수행 (--release 옵션을 붙이면 optimized 고려)
cargo test // 테스트 실행
cargo clean // target 디렉터리 삭제
cargo doc // 패키지 문서 생성(./target/doc)
cargo update // 의존성 업데이트 (Cargo.lock 기준)
cargo search <검색어> // Rust 레지스트리에서 패키지(crate) 검색
```

Rust

모듈 선언 / Privacy

```
mod message {          // 모듈 선언
    pub fn h() { h2(); }  // pubic 함수
    fn h2() {}            // private 함수
}
fn main() {
    message::h();      // 호출 가능(public)
    // message::h2();  // ERROR, 호출 불가(private)
}
```

Vec<T>

```
let mut v: Vec<i32> = vec![1, 2]; // Vec=확장 가능한 배열 타입
println!("len={}, cap={}",v.len(),v.capacity()); // len=2, cap=2
v.push(3); // [1, 2, 3], 마지막 element에 3추가
println!("len={}, cap={}",v.len(),v.capacity()); // len=3, cap=4
v.pop(); // [1, 2], 마지막 element 삭제
v.remove(1); // [1], 1번 index 값 삭제
println!("len={}",v.len()); // len=1
println!("ptr={:?}",v.as_ptr()); //ptr=0x7fa61ac02c90
println!("{}",v.is_empty()); // false
print!("{}\n",v[0]); // 1
println!("{:?}",v); // [1]
```

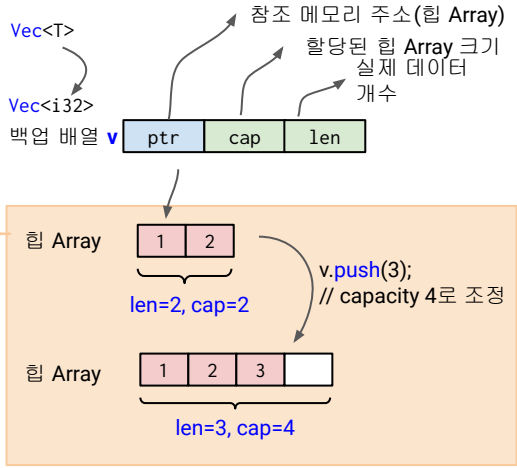
포맷팅

```
println!("{}", 1); // 1, 일반 출력 형식
println!("{:?}", 1); // 1, 디버그용 출력 형식
println!("{a} {b} {c}",c="C",b="B",a="A"); // A B C
println!("{n:>w$}", n=1, w=6);
// "     1", 왼쪽에서 6번째부터 출력
println!("{number:>0width$}", number=1, width=6); // 000001
```

for 문

```
for x in 0..2 {
        println!("x={}", x); // x=0, x=1
}
```

```
for (i,j) in (1..3).enumerate() {
        println!("i={}, j={}", i, j); // i=0, j=1 출력후 i=1, j=2 출력됨
}
```



while 문

```
let mut n = 1;
while n < 3 {
    println!("{}", n); // 1, 2 순으로 출력
     n += 1;
}
```

match (switch 문과 비슷)

```
fn main() {
    let n = 1;
    match n {
        n if n<0 => print!("{}<0", n),
        n if n>0 => print!("{}>0", n), // 매칭된 case
        1 => print!("{}", n), // 1과 일치 case
        _ => print!("{}", n), // 매칭이 없는 case
    }
}
```

함수

```
fn main() {
    let value = sum(1,2);
    print!("{}",value); // 3
}
fn sum(a: i32,b: i32) -> i32 {
    return a+b
}
```

HashMap

```
use std::collections::HashMap;
fn main() {
    let mut map = HashMap::new();
    map.insert(String::from("Hallasan"), 1947);
    let key = String::from("Hallasan");
    println!("{:?}",map.get(&key));//Some(1947)
    match map.get(&key) {
        Some(&v) => println!("{}", v), // 1947
        _ => println!("not found"),
    }
}
```

Rust

struct (튜플과 유사, 각 요소에 이름을 붙일 수 있음)

```
struct User { name: String, id: i32, active: bool}
fn main() {
    let user= User{
        name: String::from("happy"), id: 100, active: true,
    };
    println!("{}",user.name); // happy
    println!("{}",user.id); // 100
    println!("{}",user.active); // true
}
```

각 요소에 이름 부여 가능

File / Read

```
use std::fs;
fn main() {
    let s = fs::read_to_string("hi.txt")
        .expect("예외 상황(panic)에 대한 출력할 메시지");
    println!("{}", s); // hi!, 읽은 파일 컨텐츠를 출력함.
}
```

File / Write

```
use std::fs::File;
use std::io::prelude::*;
fn main() -> std::io::Result<()> {
    let mut file = File::create("hi.txt")?;
    file.write_all(b"Hello")?; // hi.txt에 hello 저장
    Ok(())
}
```

에러 핸들링 연산자

loop를 돌아 입력에 대한 전체 write 수행

Regex / match, replace

```
[dependencies]
regex = "1"
```

Cargo.toml에 의존성 추가

```
use regex::Regex;
fn main() {
    let re = Regex::new(r"^\d{4}-\d{2}-\d{2}$").unwrap();
    println!("{}",re.is_match("2021-01-01")); // true, 패턴이 일치함

    let re2 = Regex::new(r"\d{2}").unwrap();
    println!("{}",re2.replace("20 21","99")); // 99 21, 매칭된 첫번째만 치환
    println!("{}",re2.replace_all("20 21","99")); // 99 99
}
```

정규표현식

에러시 panic! 매크로 호출

매칭된 모든 케이스 치환

네이밍 컨벤션

Crates - snake_case
Modules - snake_case
Types - CamelCase
Traits - CamelCase
Enum variants - CamelCase
Functions - snake_case
Methods - snake_case
Local 변수 - snake_case
Static 변수 - SNAKE_CASE
Constant 변수 - SNAKE_CASE
Type 파라메터 - CamelCase
Lifetimes short, lowercase - 'a

문자열 처리 / push, push_str, trim, trim_start, trim_end

```
fn main() {
    let mut hello = String::from("Hello");
    println!("{}",hello); // hello
    hello.push(' '); // 공백 문자 추가 (char 타입만 push 가능)
    hello.push_str(" world");
    println!("{}",hello); // hello wrold

    println!("<{}>"," hello world ".trim()); // <hello world>
    println!("<{}>"," hello world ".trim_start()); // <hello world >
    println!("<{}>"," hello world ".trim_end()); // < hello world>
}
```

양쪽 trim

왼쪽 trim

오른쪽 trim

&mut String

문자열 추가
- push_str (&str)
- insert_str (usize, &str)

문자 추가, 삭제
- push (char)
- pop () -> Option<char>
- insert (usize, char)
- remove (usize) -> char

문자열 / contains, starts_with, ends_with

```
let greet = "hello";
println!("{}",greet.contains("ll")); // true
println!("{}",greet.starts_with("he")); // true
println!("{}",greet.ends_with("lo")); // true
```

문자열 / Split

```
let split = "a b c d".split(" ");
let vec: Vec<&str> = split.collect();
println!("{:?}",vec); // ["a", "b", "c", "d"]
```

Rust

test / 단위 테스트

```
pub fn sum(a: i32,b: i32) -> i32 { return a+b; }
fn main() {
    println!("{}",sum(1,2));
}
#[cfg(test)]
mod tests {
    use super::*;
    #[test]
    fn test_sum() {
        assert_eq!(sum(1, 2), 3);
    }
}
```

단위 테스트를 수행하는 모듈인 경우 선언

단위 테스트를 수행하는 함수인 경우 선언

```
$ cargo test

running 1 test
test tests::test_sum ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out
```

Casting / 숫자↔문자열

```
fn main() {
    let x: u32 = 10;
    let y = "11".to_string();
    println!("{}", x.to_string()); // 10, 숫자를 문자열로 캐스팅
    println!("{}", y.parse::<i32>().unwrap()); // 11, 문자열을 숫자로 파싱해 캐스팅
}
```

배열 / 최소값, 최대값, 합계

```
let items = [1, 2, 3];
println!("{:?}", items.iter().max()); // Some(3)
println!("{:?}", items.iter().min()); // Some(1)
let sum: u8 = items.iter().sum();
println!("{:?}", sum); // 6 (1+2+3의 합계)
```