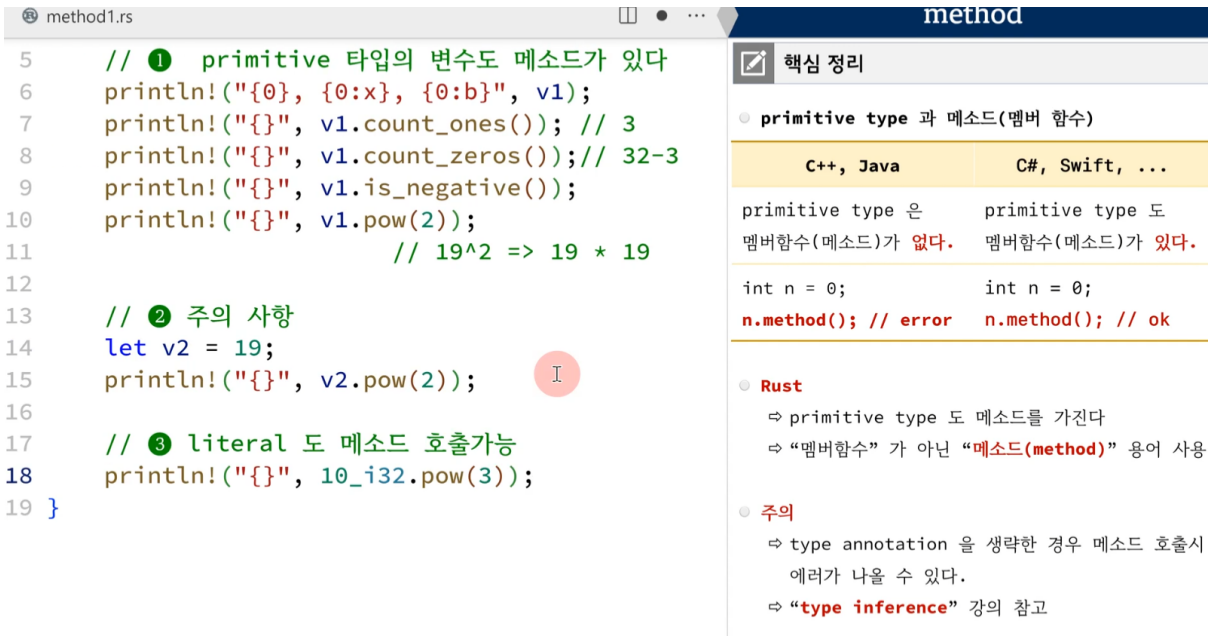
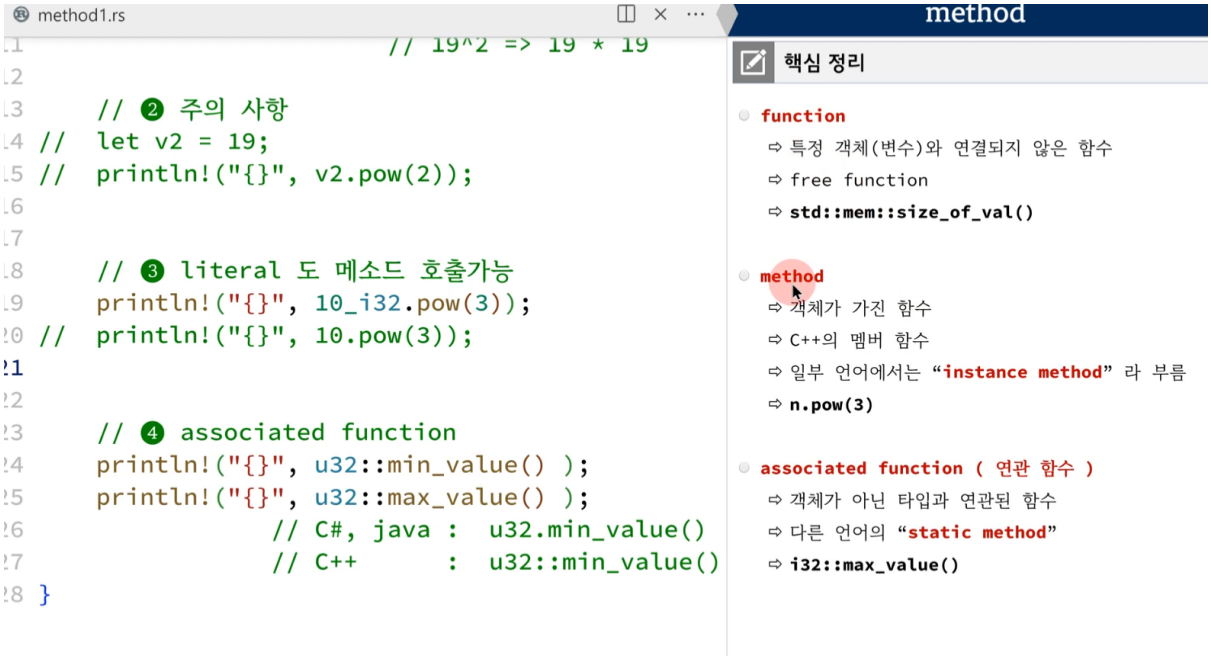
04. method


method1.rs
// ❶ primitive 타입의 변수도 메소드가 있다
println!("{0}, {0:x}, {0:b}", v1);
println!("{}", v1.count_ones()); // 3
println!("{}", v1.count_zeros());// 32-3
println!("{}", v1.is_negative());
println!("{}", v1.pow(2));
// 19^2 => 19 * 19
// ❷ 주의 사항
let v2 = 19;
println!("{}", v2.pow(2));
// ❸ literal 도 메소드 호출가능
println!("{}", 10_i32.pow(3));
}
method
핵심 정리
primitive type 과 메소드(멤버 함수)
C++, Java
C#, Swift, ...
primitive type 은 멤버함수(메소드)가 없다.
primitive type 도 멤버함수(메소드)가 있다.
int n = 0;
n.method(); // error
int n = 0;
n.method(); // ok
Rust
⇨ primitive type 도 메소드를 가진다
⇨ "멤버함수" 가 아닌 "메소드(method)" 용어 사용
주의
⇨ type annotation 을 생략한 경우 메소드 호출시 에러가 나올 수 있다.
⇨ "type inference" 강의 참고


method1.rs
// 19^2 => 19 * 19
// ❷ 주의 사항
// let v2 = 19;
// println!("{}", v2.pow(2));
// ❸ literal 도 메소드 호출가능
println!("{}", 10_i32.pow(3));
// println!("{}", 10.pow(3));
// ❹ associated function
println!("{}", u32::min_value());
println!("{}", u32::max_value());
// C#, java : u32.min_value()
// C++ : u32::min_value()
}
method
핵심 정리
function
⇨ 특정 객체(변수)와 연결되지 않은 함수
⇨ free function
⇨ std::mem::size_of_val()
method
⇨ 객체가 가진 함수
⇨ C++의 멤버 함수
⇨ 일부 언어에서는 "instance method" 라 부름
⇨ n.pow(3)
associated function (연관 함수)
⇨ 객체가 아닌 타입과 연관된 함수
⇨ 다른 언어의 "static method"
⇨ i32::max_value()

method2.rs

```rust
fn main()
{
    let n1 : i32 = -10;

    // n1의 절대값을 출력 하고 싶다.
    println!("{}", n1.abs() );


    // 24와 18의 최대 공약수를 찾고 싶다.
    println!("{}", 24_u32.rem_euclid(18) );
}
```

method

핵심 정리

- 타입이 가진 **method**를 검색 하려면 ?
  - ⇨ "**THE STANDARD LIBRARY**" 문서 참고
  - ⇨ https://doc.rust-lang.org/std/index.html

method3.rs

```rust
{
    let n1 = 10;

    // ❶ 암시적 변환 허용 안됨
//  let f1 : f64 = n1;


    // ❷ 명시적 변환
    let f2 = n1 as f64;
    let f3 = f64::from(n1);

    let n3 = 3.4 as i32;
}
```

casting

핵심 정리

- 정수 <=> 실수 타입 변환
  - ⇨ 암시적 변환 안됨
  - ⇨ "**as**" 또는 "**from**" associated function 사용

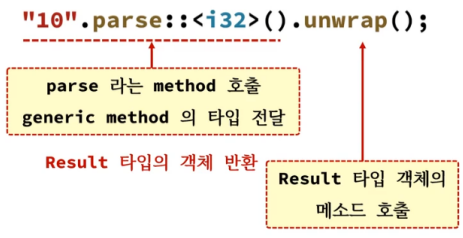
method3.rs

```rust
    let f2 = n1 as f64;
    let f3 = f64::from(n1);

    let n3 = 3.4 as i32;


    // ❸ 문자열 => 정수 또는 실수
    let a = "10".parse::<i32>().unwrap();
    let b = "3.4".parse::<f64>().unwrap();

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

casting

핵심 정리

- 문자열 => 정수, 실수 변환



- **Result** 타입
  - ⇨ Rust 에서 실패 가능성이 있는 함수(메소드) 호출시 반환 값으로 주로 사용하는 타입.
  - ⇨ "**Enum**" 으로 만든 타입
  - ⇨ "**Result**" 강좌 참고