Types

```
// ---------- TYPE ----------
    // i8, i16, i32, i64, i128 => signed integer (positive and negative) with size of integer specified in number of bits
    // u8, u16, u32, u64, u128 => unsigned integer (positive) with size of integer specified in number of bits
    // f32, f64 => single-precision and double-precision floating point numbers
    // bool => true, false
    // char => character declared with '' single quotation marks
    // &str => immutable string literal, stored on the stack with "" double quotation marks
    // String => mutable string vector, stored as a Vec<u8> on the heap with "" double quotation marks
```

Control structures

```
// ---------- CONTROL STRUCTURE ----------

// ---------- CONDITIONALS ----------

// IF ELSE IF ELSE

let number:i16 = 42;
if number < 0 {
    println!("number is negative");
} else if number == 0 {
    println!("number is zero");
} else {
    println!("number is positive");
}

// MATCH EXPRESSION
    // rust's powerful pattern-matching construct similar to switch case in other languages
    // match and => define a match expression, where every match expression evaluates to a single value since matches are exhaustive and each match-arm (=>) points to an expression
    // _ => match-all pattern which acts as the default case for match expressions, required in every match expression to cover every possible match-arm since matches are exhaustive

fn im_feeling_lucky(feeling_lucky:bool) -> i32 {
    match feeling_lucky {
        true => 100,
        false => 0,
    }
} // a match expression within a function

enum Coin {
    Penny,
    Nickel,
    Dime,
    Quarter,
} // create the enum Coin

fn value_in_cents(coin:Coin) {
    match coin {
        Coin::Penny => {
            println!("Lucky penny!");
            1
        },
        Coin::Nickel => 5,
        Coin::Dime => 10,
        Coin::Quarter => 25,
    }
} // match expressions can be used alongside enums to leverage on powerful pattern-matching capabilities

let some_u8_value = 0u8;
match some_u8_value {
    1 => println!("one"),
    2 => println!("two"),
    3 => println!("three"),
    _ => (),
}

// UNDERSCORE
    // _ => catch-all pattern that specifies a value to be discarded and can be used for destructuring, also a match-all pattern in match expressions
    // prefixing a variable with _ will indicate to the compiler to ignore it even if its unused
```