Functions and Expressions and Statements

Statements

- A line of code that doesn't return anything and simply declares a value
- `let y = 1;` is a statement, it doesn't return a value
- `let x = (let y = 3);` is **NOT** valid

Expressions

- Anything that returns a value is an expression
    - `3 + 5, 5 < 6, 2 != 3` are examples of expressions
    - Another expression type uses a body `{}`

```
let x = {
    let y = 5; // this is a statement
    y+1 // this is an expression
};
```

        - **Note** the last line `y+1` has no semi colon, denoting its expression and returning a value rather than being a statement

Functions

- Functions convention in rust is to use snake case
    - Name a funtion using : `fn snake_case_name(){}`
    - Location of the function doesn't matter, it can be defined anywhere in the program
    - Parameters **MUST** be explicitly type defined
        - `fn snake_case_name(x:<type>, y:<type>){}`
    - Example:

```
fn test_run(){
    println!("test_run has been called");
}

fn add_numbers(x:i32, y:i32){
    pritnln!("add_numbers {} + {} = {}", x, y, (x+y));
}
```

    - Returning in a function

- Returns are done by the `-> <type>` operator
- `fn test() -> i32{}` returns an i32 type
- **Note** No matter which way you use to return, if the return statement is in the middle of the function (i.e. within a loop or conditional) you **NEED** to use the second method with the keyword
    - One way to return is by an expression
        - i.e, similar to an expression body, if an expression is the last statement of the function, then it returns the result of the expression.
    - Example:

```
fn mult_numbers(x:i32, y:i32) -> i32{
    x * y //note this is an expression no semicolon
}
```

    - The other way to return is by using the `return` keyword
        - Note that this way is a either statement or expression, the semicolon doesn't matter
        - Example:

```
fn sub_numbers(x:i32, y:i32) -> i32{
    let result = x - y;
    if result < 10 {
        return result+10 //returning with an expression
    }
    return result //returning with a return expression
}
```