Variables and Mutability

Learn to Code with Rust / Section Review

Variables

- A **variable** is a name for a value in the program.
- Declare a variable with the **let** keyword.
- Rust will infer a variable's type from its initial value. We can also explicitly annotate its data type with a colon and the type.
- Rust infers an **i32** for an integer (a whole number) and an **f64** for a float (a decimal-point number)

Interpolation

- Use **{ }** to inject dynamic values into a string.
- Write a value inside the **{ }** or pass the value in as an argument to the **println!** macro.
- Rust assigns a numeric position to each argument after the string. The count starts at 0.
- The curly braces can reference arguments by their numeric position **{0}**.

Mutability

- Rust variables are immutable by default. An **immutable variable** cannot change its value.
- Use **let mut** to make a variable mutable. Its value can now change over the program's execution.
- A mutable variable can change its value but not its *type*.

Constants

- A **constant** is a hardcoded immutable value that is known at compile time.
- Constants can be declared at the file level and be used across multiple functions.
- Constants require an explicit type annotation and an initial value.

Variable Shadowing

- **Variable shadowing** is the re-declaration of an existing variable with the **let** keyword.
- The technique is common when transforming a value to the same representation in a different data type.
- The new declaration overshadows ("towers above") the previous declaration.

Scopes

- A **scope** is the boundary or region in which a name in the program is valid.
- A pair of curly braces creates a **block**. A **block** creates a scope.
- A function has a block, so it creates a scope. We can also create *nested* blocks within functions with **{ }**.

Compiler Directives

- A c**ompiler directive** is an annotation that tells the compiler how to parse the code.
- Compiler directives can apply to individual lines, functions, and files.
  - #[allow(unused_variables)] for lines/functions
  - #![allow(unused_variables)] for files

Miscellaneous

- A **type alias** defines a new name for an existing type.
- Every error has a unique error code. The compiler + the Rust error codes index can explain each one.
  - rustc --explain E0384
- Prefix an underscore to a variable name to silence the compiler warning about it being unused.