**What is RUST?**

* Open-Source Language similar to c++
* Designed for performance and reliability
* It is compiled language means uses a compiler that source code into machine code that can run into your system.
* **Statically Typed:** In Rust, the type of a variable is checked at compile time, meaning it must be known before the program runs. This helps catch errors early.
* **Strongly Typed:** Rust ensures that once a variable has a type, it can only hold values of that type. You can't mix different types (like adding a number to text) without explicitly converting them.

**Memory Management in Rust:**

* Rust ensures all memory access is valid, preventing issues like null pointers or invalid references.
* It’s memory-safe by default, without needing a garbage collector.
* These safety features don’t slow down performance.

**Cargo:**

* Manages dependencies for repeatable builds.
* Downloads and build external libraries.
* Calls rustc with correct parameters.

**Variable:**

* Name associated with a value stored in the computer memory.
* Declare in RUST by using **let** keyword.

**Mutability:**

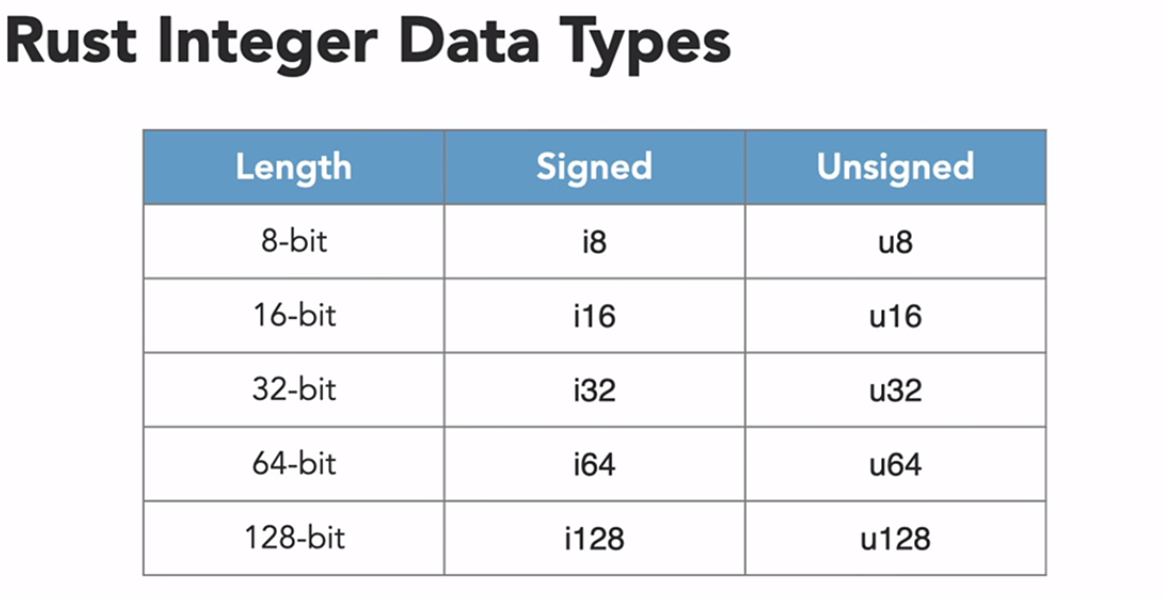
* RUST variables are immutable by default.
* Variable must be explicitly declared as mutable using **mut** keyword.

**Rust Variable Naming Rules:**

* Can contain letters, numeric digits, and the underscore character.
* Must begin with a letter or an underscore.
* Case-sensitive: uppercase and lowercase letters are distinct.
* Variable named cannot be keywords such as **let** or **mut.**

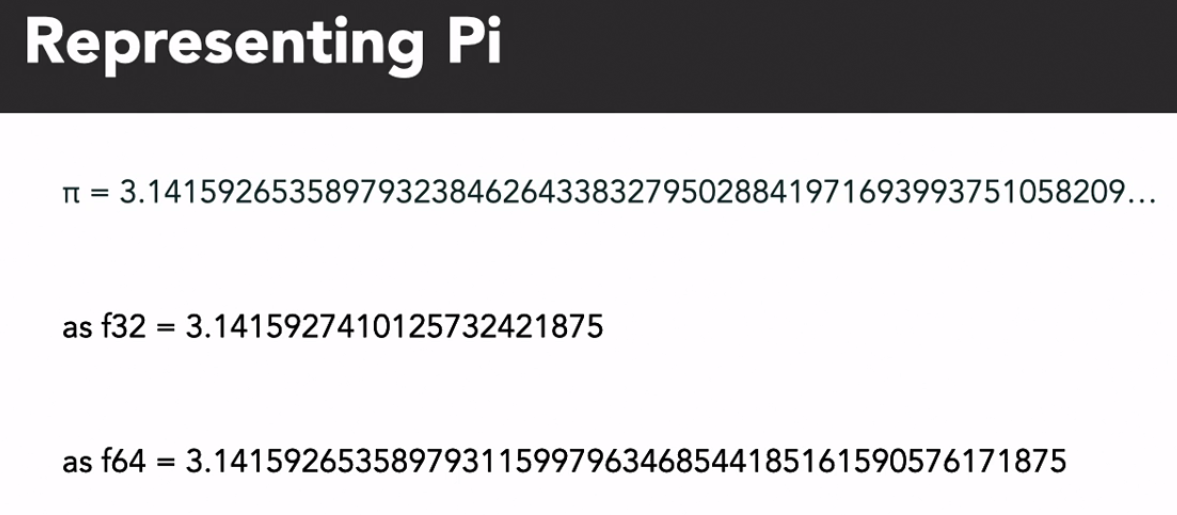
**Data Type:**

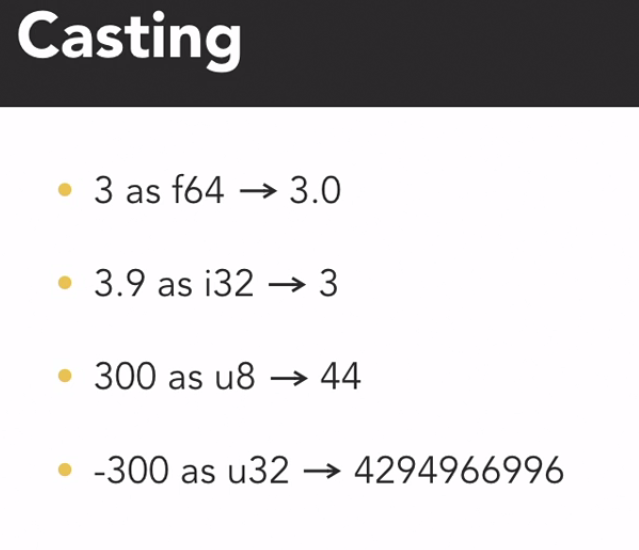
* Defines how data should be stored and interpreted.
* Defines which operation can be used to interpret data.



**Floating-Point Data Types**

* Represent numbers with decimal points using IEEE 754 standard
* Rust has two floating-point types: f32 and f64
* Value stored as fractional and exponential components
* Max f64 value≈ 1.7976931348623157 × 10308.





**Bitwise Operators:**

Logical operations on patterns of bits at the individual bit level

* **NOT**: Flips the bits of a number (1 becomes 0, 0 becomes 1).
* **AND**: Compares two numbers bit by bit, returning 1 only if both bits are 1.
* **OR**: Compares two numbers bit by bit, returning 1 if at least one bit is 1.
* **XOR**: Compares two numbers bit by bit, returning 1 if the bits are different.
* **SHIFT**: Moves the bits of a number left (adds 0s on the right) or right (adds 0s on the left).
  + let result = 0b1010 << 2; // Left shift 1010 (binary) by 2 places becomes 101000

**Boolean Data Types and its Operations:**

* Values can be either true or false.
* Logical Operations: NOT, AND, OR and XOR.
* 1 🡪 True
* 0 🡪 False

**Short Circuiting Logical Operations:**

* **&& (AND):** If the first condition is false, the second condition is not checked, because the result will always be false.
* **|| (OR):** If the first condition is true, the second condition is not checked, because the result will always be true.

**Note:** panic!() macro is used to stop the program immediately when something goes wrong, displaying an error message.

**Char Data Type:**

* Represents single character.
* **Unicode scalar:** Can represent any Unicode character.
* Stored 4 bytes.

**Scalar Types:**

Scalar types represent a single value. Rust has four primary scalar types:

* **Integer:** Whole numbers (e.g., i32 for 32-bit signed integers, u8 for 8-bit unsigned integers).
* **Floating-point:** Numbers with decimals (e.g., f32 for 32-bit, f64 for 64-bit precision).
* **Boolean:** Represents true or false values (true, false).
* **Char:** Represents a single Unicode character (e.g., 'a', '🦀').

**Compound Types:**

Compound types can group multiple values into one type. Rust has two basic compound types:

* **Array:** 
  + A fixed-size collection of values of the same type (e.g., [1, 2, 3] of type [i32; 3]).
  + Stored in contiguous memory locations.
* **Tuple:** 
  + A fixed-size collection of values of different types (e.g., ("hello", 5, true) of type (str, i32, bool)).

**usize Data Type:**

* Size is based on number of bytes needed to reference memory
* Compiling for 32-bit processor→ usize is 4 bytes
* Compiling for 64-bit processor→ usize is 8 bytes

**Function:**

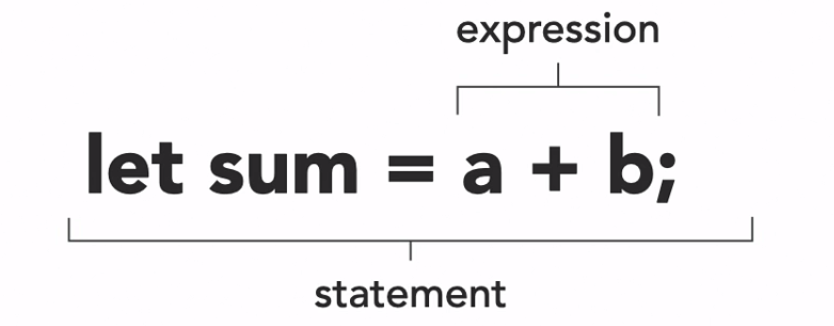
* Organize sections of code into reusable modules.
* **Parameters:** 
  + Variable used to provide input data to a function.
  + Defined in parenthesis of the function signature.

**Statement:**

* Performs an action without returning a value
* Ends with semicolon

**Expression:**

* Evaluates to a resulting value.
* Does not end with semicolon.



**Unit Data Type:**

* Used when there is no other meaningful value that can be returned.
* Represent with ()

**Loops**

**Loop:** Infinite loop that repeats a block of code until explicitly stopped.

**Break Keyword:** Immediately exit the loop and continue execution afterward.

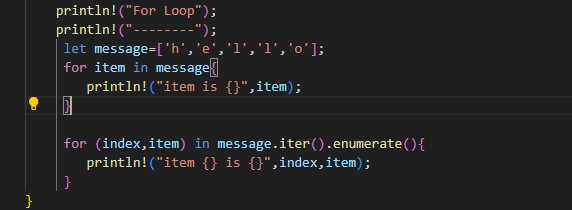
**While Loop:** Loop that repeats a block of code while condition is true

**Loop vs While Loop:**

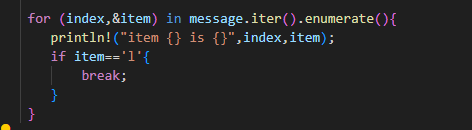


Break keyword in loop can be used to return value while in while loop break keyword does not return value.

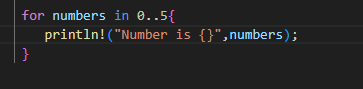
**For Loop:**



* iter() allows you to loop through each item in the collection (like a list or array) one by one.
* iter() gives references to elements.
* enumerate() gives each item an index, so you can access both the index and the item in the loop.



* &item is used because iter() gives references to elements. Writing &item lets you access the value directly instead of a reference.



**Summary of Rust Loops**

**Loop:**

* Repeat a block of code forever.
* Need the loop to return a value.

**While Loop:**

* Continue repeating the block of code as long as condition is true.

**For Loop:**

* Iterate over each item in a collection.
* Repeat a block of code N times -> iterate over range 0..N.