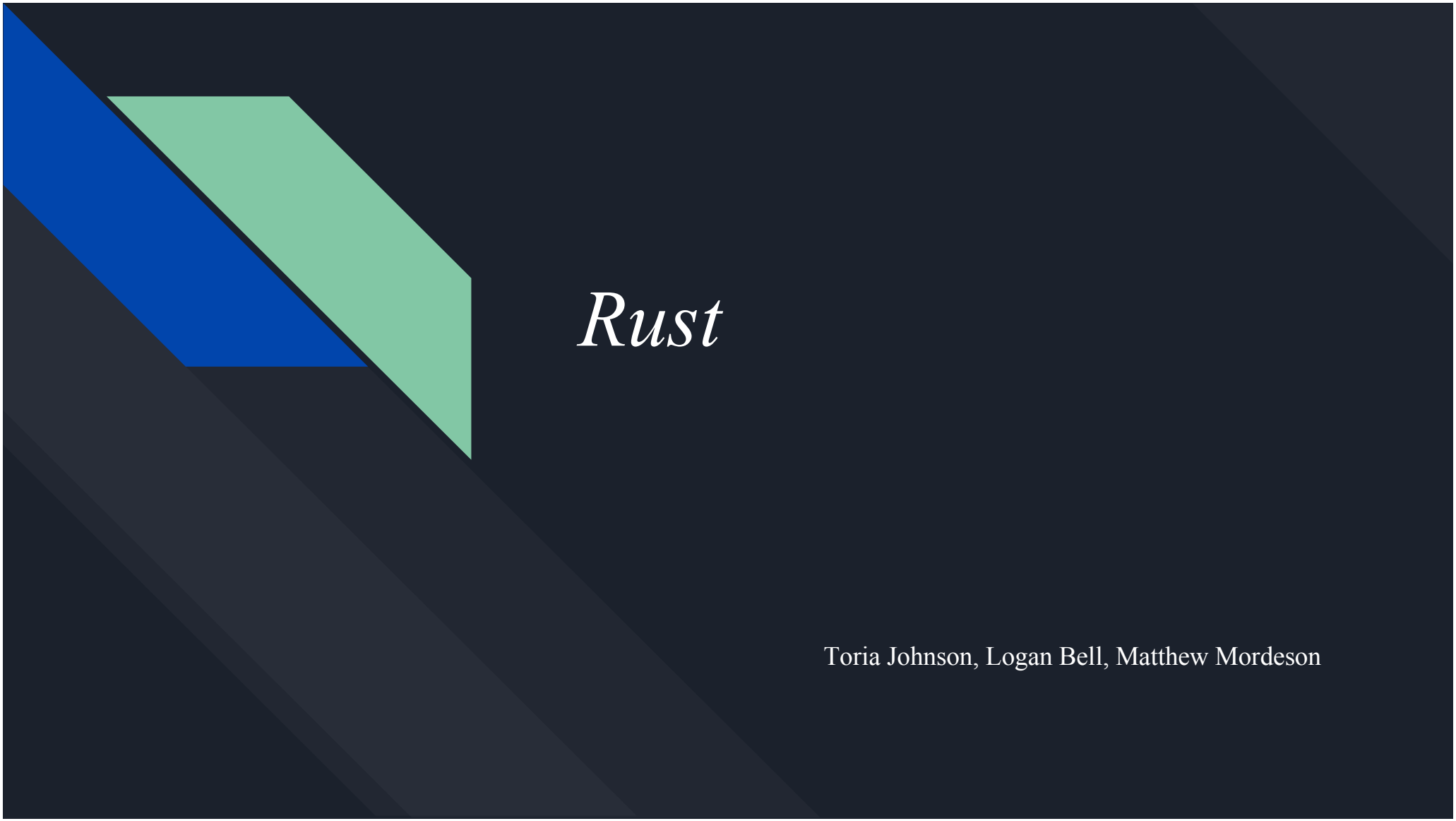
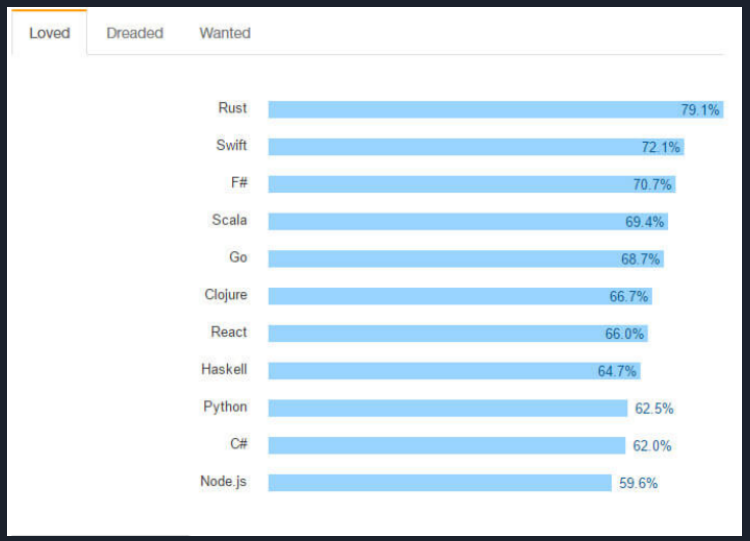

Rust
Toria Johnson, Logan Bell, Matthew Mordeson

Creation, Development, and History

- Started 2006 by Graydon Hoare
- Mozilla Started funding it in 2009
- Announced in 2010
- Based on C++ but also influenced by other languages such as LISP
- Compiled
- Capable of imperative and functional programming



Key Features and Design Goals

- Focused on safety
  - Uses ownership for memory control
  - No null pointers
- Supports Object Oriented programming
  - No inheritance, instead shared traits
  - Structs and implementations can be combined to function like objects
- Functional Language features
  - Closures - functions that can be passed as parameters or saved in a variable

Object-Oriented Example

```
use rand::{Rng, thread_rng };
fn main(){
    let mut die = DIE::new();
}

pub struct DIE {
    face:f64,
}

impl Die{
    pub fn new() -> Die {
        Die{
            let mut rng = rand::thread_rng);
            if rng.gen() {
                let x: f64 = rnd.gen_range(0,10);
            }
        }
    }
    fn roll (&mut self) {
        self.face = rnd.gen_range(0,10);
}
}
```

Usage and Applications

- Developed to be suitable for systems programming with “control of a low-level language, but with the powerful abstractions of a high-level language”
- Designed to be used for the same tasks at C/C++ but with enhanced memory safety
- Dropbox uses rust in Magic Pocket (Dropbox’s file storage system)
- Mozilla uses it to write Servo, it’s experimental web browser engine

Assignments and Parameters

- Variables are immutable by default
- Assignments are done with the “let” keyword
  - Statically typed with inference
- Much like C++, if it can go on the stack, it will
- Can be allocated on to the heap by using Box:
- Passing parameters located on the stack -> by value
- Passing parameters on the heap -> by value (like Java) with a twist...

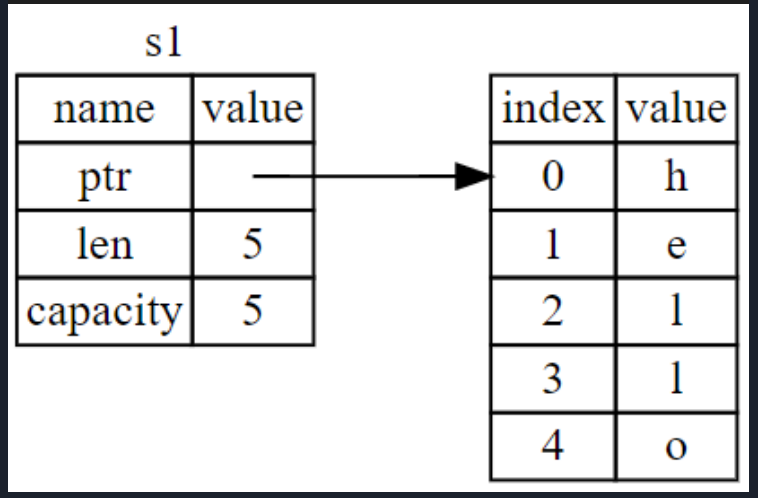
```
fn main(){
    let x : i32;
    let mut y = 76;
    let z = Box::new(5);
}
```

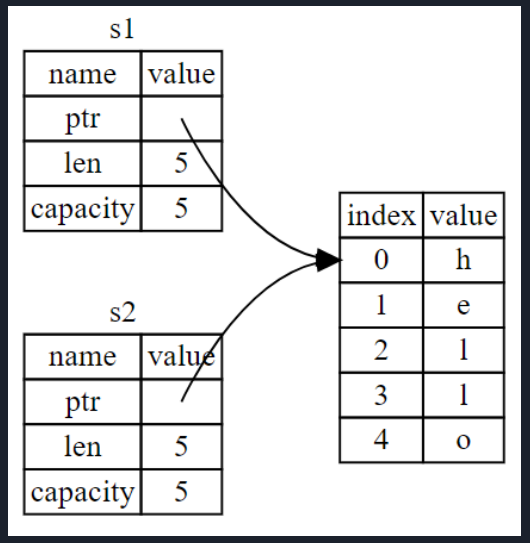
Memory

- Primitives are allocated on the stack by default, complex data is stored as a pointer on the heap, as in Java.
- Uses ownership to guarantee memory safety, and it is the key element of Rust
  - Rust makes use of the “Drop” function to remove data from the heap
  - Drop is automatically called on all variables declared in a scope upon exiting it
  - To ensure multiple drops are not called on the same data, rust uses “moves”
  - This makes multiple assignments behave in strange ways.

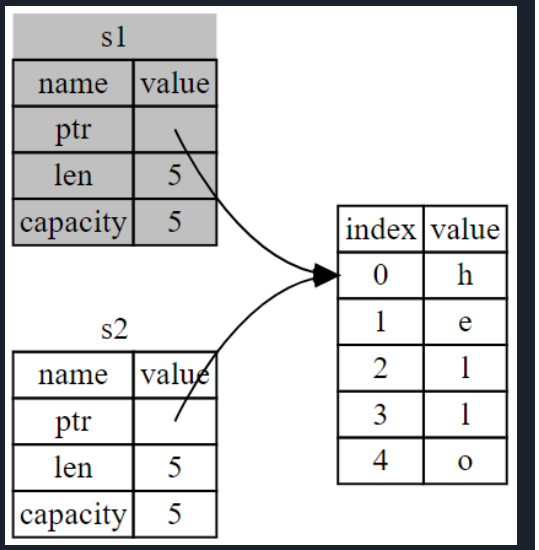
Memory Cont.

```
fn main() {
    let s1 = String::from("hello");
    s1.push_str(", world!");
    let s2 = s1;
    println!("{}",s1);
}
```





VS



Memory Example

```rust
fn main() {
    let nums = vec![2,14,51,37];
    let mut avg : i32;
    avg = getAvg(nums);
    println!("The average of the list {:?} is {:?}",nums,avg);
}

fn getAvg(num_list: Vec<i32>) -> (i32){
    let mut tot = 0;
    let mut size = 0;
    for num in num_list.iter() {
        tot = tot + num;
        size = size + 1;
    }
    return tot/size;
}
```

```
error[E0382]: use of moved value: `nums`
 --> main.rs:4:53
  |
3 |     let avg = getAvg(nums);
  |                      ---- value moved here
4 |     println!("The average of the list {:?} is {:?}",nums,avg);
  |                                                     ^^^^ value used here aft
er move
  |
  = note: move occurs because `nums` has type `std::vec::Vec<i32>`, which does n
ot implement the `Copy` trait

error: aborting due to previous error
```

The Solution - Borrowing

- Borrowing is a way to preserve passed values in Rust
- Upon termination, all references are deleted except for borrowed ones

```
fn main() {
    let nums = vec![2,14,51,37];
    let mut avg : i32;
    avg = getAvg(&nums);
    println!("The average of the list {:?} is {:?}",nums,avg);
}

fn getAvg(num_list: &Vec<i32>) -> (i32){
    let mut tot = 0;
    let mut size = 0;
    for num in num_list.iter() {
        tot = tot + num;
        size = size + 1;
    }
    return tot/size;
}
```

Data Types

- Number assignments are type inferenced, can declare with memory that needs to be allocated, or pointer size.
  - Signed int : i8, i16, i32 (default), i64
  - Unsigned int: u8, u16, u32, u64
  - Floating point: f32, f64(default)
  - Char : 4 bytes each
  - Booleans are binary types
- Examples of invalid and valid assignments:

Control Structures

- If , else , and else if
  - Rust will not automatically try to convert non-boolean types to boolean (such as [not 0] or [0] in JavaScript)
  - If statements can be used in *let* statements, the result of the if determining the value of the variable:

```
let number = if condition {
    5
} else {
    6
};
```

- Loops:
  - Loop- loops until told to stop with break (Ctrl-C).
  - While- repeats while condition is true.
  - For- loop through a collection or range.
- Loops cannot return values like *if*s can, instead returning (), the 'unit' or equivalent of *nil* or *null* in other languages.

Developing With Rust: Cargo

- Cargo is a package manager for Rust which allows for easy access to code dependencies and issuing builds.
  - Building a new project uses “cargo new _____” with an optional --bin flag for creating an executable instead of a library, and a --vcs flag for disabling version control
- By default Rust initializes a git repository for the project
  - **Version control (primarily, git) is heavily central to Rust and its development workflow!**
- Git configuration is determined per-package via Cargo.toml

Concurrency

- Ownership and type security are innately predisposed for building parallel programs.
  - Errors can be moved to compile time instead of runtime: *fearless concurrency*, as Rust developers call it
- Many languages have a *1:1* model or *M:N*, Rust standard library only contains 1:1 threading examples
  - Rust will try to have speed over deference
- thread::spawn function used to initialize a new thread, thread::sleep forces a thread to stall for a duration
- Join handles are the return values of a thread, saved in a variable.
  - The JoinHandle will, when" join" is called, wait for the thread to finish before the main program continues progression
- Concurrency is far more complicated than we can cover here--synchronization, message passing, and variable mutation in parallel!

Summary

- Rust is focused on safety in environments where small errors can compound
- Rust is used by major players like Dropbox and Mozilla
- Rust has unique memory management capabilities
- Rust is seamlessly integrated into Git
- Rust is well-suited to parallel programming applications

Sources

- https://doc.rust-lang.org/book/second-edition
- https://thenewstack.io/safer-future-rust/
- https://medium.com/mozilla-tech/why-rust-is-the-most-loved-language-by-developers-666add782563