

# CMSC 330: Organization of Programming Languages

---

## Working with Rust

# Installing Rust

---

- Instructions, and stable installers, here:  
<https://www.rust-lang.org/en-US/install.html>
- On a Mac or Linux (VM), open a terminal and run  
`curl https://sh.rustup.rs -sSf | sh`
- On Windows, download+run `rustup-init.exe`  
<https://static.rust-lang.org/rustup/dist/i686-pc-windows-gnu/rustup-init.exe>

# Rust compiler, build system

---

- Rust programs can be compiled using `rustc`
  - Source files end in suffix `.rs`
  - Compilation, by default, produces an executable
    - No `-c` option
- Preferred: Use the `cargo` package manager
  - Will invoke `rustc` as needed to build files
  - Will download and build dependencies
  - Based on a `.toml` file and `.lock` file
    - You won't have to mess with these for this class
  - Like `ocamlbuild`

# Using rustc

---

- Compiling and running a program

main.rs:

```
fn main() {  
    println!("Hello, world!")  
}
```

```
% rustc main.rs
```

```
% ./main
```

```
Hello, world!
```

```
%
```

# Using cargo

---

- Make a project, build it, run it

```
% cargo new hello_cargo --bin
```

```
% cd hello_cargo
```

```
% ls
```

```
Cargo.toml  src/
```

```
% ls src
```

```
main.rs
```

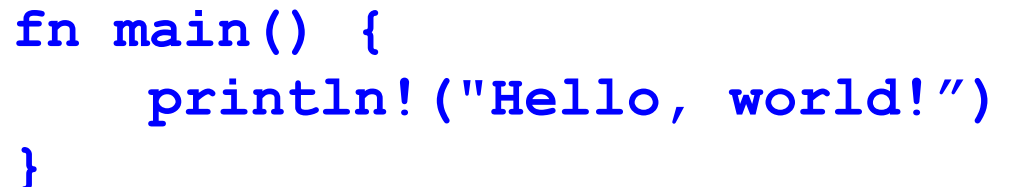
```
% cargo build
```

```
Compiling hello_cargo v0.1.0 (file:///...)
```

```
Finished dev [unoptimized + debuginfo] ...
```

```
% ./target/debug/hello_cargo
```

```
Hello, world!
```



```
fn main() {  
    println!("Hello, world!")  
}
```

# Rust, interactively

---

- Rust has no top-level *a la* OCaml or Ruby
- There is an in-browser execution environment
  - See, for example,  
<https://rustbyexample.com/hello.html>

## Hello World

This is the source code of the traditional Hello World program.

```
// This is the main function
fn main() {
    // The statements here will be executed when the compiled binary is called

    // Print text to the console
    println!("Hello World!");
}
```

```
Hello World!
```