Crystal Compiled, Ruby-like language

CONCEPTS

Crystal is compiled (via LLVM) with Ruby-like syntax and static types that are mostly inferred. There is no interpreter: run compiles to a cached temp binary each time.

Write src/app.cr
↓ `shards install`
Deps lib/ + shard.lock
↓ `crystal build`
Build bin/app
↓ `crystal spec`
Test spec/*_spec.cr

Plain build is fast but unoptimized (-O0). Benchmark and ship with --release; it is much slower to compile.

RUN AND BUILD

`crystal` **`run`** *`<file.cr>`*
Compile and run a file

`crystal` **`run`** *`<file.cr>`* **`--`** *`<args>`*
Pass arguments to the program

`crystal` *`<file.cr>`*
Shorthand for run

`crystal` **`build`** *`<file.cr>`*
Build a binary in the current dir

`crystal` **`build --release`** *`<file.cr>`*
Optimized build (-O3, single module)

`crystal` **`build -o`** *`<path>`* *`<file.cr>`*
Choose the output path

`crystal` **`build --no-debug`** *`<file.cr>`*
Omit debug symbols

`crystal` **`build --static`** *`<file.cr>`*
Link statically

EVAL AND PLAYGROUND

`crystal` **`eval`** '*`<code>`*'
Compile and run a code string

`echo` '*`<code>`*' **`| crystal eval`**
Read code from stdin

`crystal` **`play`**
Start the browser playground

EVAL AND PLAYGROUND (cont.)

`crystal` **`version`**
Print Crystal, LLVM and target triple

PROJECTS

`crystal` **`init app`** *`<name>`*
Scaffold an application

`crystal` **`init lib`** *`<name>`*
Scaffold a library

init creates src/, spec/ and shard.yml. Commit shard.lock for applications, leave it out for libraries.

`shards init`
Create a shard.yml

SPECS

`crystal` **`spec`**
Run all specs in spec/

`crystal` **`spec`** *`<file>`*
Run one spec file

`crystal` **`spec`** *`<file>`*`:`*`<line>`*
Run the example at a line

`crystal` **`spec -e`** '*`<name>`*'
Run examples matching a name

`crystal` **`spec --fail-fast`**
Stop at the first failure

`crystal` **`spec --order random`**
Randomize example order

FORMAT AND TOOLS

`crystal` **`tool format`**
Format all files in place

`crystal` **`tool format`** *`<path>`*
Format a file or folder

`crystal` **`tool format --check`**
Fail if anything is unformatted (CI)

`crystal` **`docs`**
Generate API docs into ./docs

SHARDS

`shards install`
Install deps into lib/ per the lock

`shards update`
Re-resolve all deps, rewrite shard.lock

SHARDS (cont.)

`shards build`
Install deps, build into bin/

`shards install --production`
Locked only, no dev deps, fail if out of sync

shard.yml

```
name: myapp
version: 0.1.0
targets:
  myapp:
    main: src/myapp.cr
dependencies:
  kemal:
    github: kemalcr/kemal
    version: ~> 1.6
development_dependencies:
  ameba:
    github: crystal-ameba/ameba
```

Add a dependency to shard.yml by hand, then run shards install; there is no shards add command. Then require it by shard name.

LANGUAGE BASICS

Types and methods

```
name = "Ada"
age : Int32 = 36
def greet(who : String) : String
  "Hi, #{who}"
end
puts greet(name)
```

Classes and structs

```
class User
  property name : String
  getter id : Int32
  def initialize(@name, @id); end
end
struct Point
  getter x : Int32, y : Int32
  def initialize(@x, @y); end
end
```

Nil safety

```
# x is String?
x = ENV["HOME"]?
if x
  puts x.size
end
x.try &.size
x.not_nil!
```

`require "./file" / require "json"`
Load a relative file or stdlib module

Type? means Type | Nil. The compiler rejects calls on a possible nil; narrow with if, try, or not_nil! (raises at runtime).