Crystal101 – Complete Encyclopedia & Masterclass

Table of Contents

- Iterate over array
- Exclusive range
- Each - iterate with block
- Map - transform elements
- Select/Filter - keep matching elements
- Reduce - combine all elements
- Any/All - check conditions
    - **?** Conditional Statements
- Inline unless
- With ranges
- Nested ternary (use sparingly)
    - □ Functions & Sub-Functions
- Simple function
- Function with parameters
- Function with return type
- Call with keyword arguments
- Main function
- Sub-function 1
- Sub-function 2
    - □ Object-Oriented Programming
- Create object
- Access properties
    - □ Collections
- Array creation
- Common methods
- Adding/removing elements
- Iteration with transformation
- Hash creation
- Accessing values
- Adding/modifying
- Hash methods
- Iterating
- Set creation (no duplicates)
- Set operations
- Set algebra
- Fixed-size, heterogeneous collection
- Accessing elements
- Pattern matching
- Named tuple - like a lightweight object
- Pattern matching

- File Operations
- JSON
- HTTP
- Time

PDF Creator: Rishi
Date: November 17, 2025
Subtitle: From Beginner to Crystal Mastery

Cover & Introduction

What is Crystal?

Crystal is a **compiled, statically-typed programming language** with a syntax inspired by Ruby, designed for high-performance systems programming. It combines the elegance of Ruby with the speed of C/C++, making it ideal for building fast, reliable applications while maintaining developer productivity.

Key Philosophy: Beautiful syntax doesn't mean sacrificing performance. Crystal proves you can have both.

Key Features

- **Compiled Language:** Crystal compiles to machine code via LLVM, delivering C-like performance
- **Ruby-like Syntax:** Clean, readable code that feels familiar to Ruby developers
- **Static Typing with Type Inference:** Catch errors at compile-time without verbose type declarations
- **Concurrency First:** Built-in support for Fibers and channels for concurrent programming
- **Macros:** Metaprogramming capabilities for code generation and powerful abstractions
- **Zero-cost Abstractions:** Write elegant code without runtime overhead
- **Cross-platform:** Runs on Linux, macOS, and Windows (via WSL)

Learning Objectives for Crystal101

✓ Master Crystal syntax and fundamentals
✓ Build proficiency with data types, operators, and control flow
✓ Understand OOP principles and functional programming paradigms
✓ Work with standard libraries and external packages
✓ Develop concurrent applications using Fibers and channels
✓ Create real-world projects demonstrating mastery
✓ Optimize performance and write production-ready code

Installation & Setup

Windows (WSL2)

```
# Install WSL2 if not already installed<a></a>
wsl --install

# In WSL terminal, update package manager<a></a>
sudo apt update &amp;&amp; sudo apt upgrade -y

# Install Crystal<a></a>
curl -fsSL https://crystal-lang.org/install.sh | bash
```

```
# Verify installation<a></a>
crystal --version
```

Linux (Ubuntu/Debian)

```
# Add Crystal repository<a></a>
curl -fsSL https://keybase.io/crystal/pgp_keys.asc | sudo apt-key add -
echo "deb https://dist.crystal-lang.org/apt crystal main" | sudo tee /etc/apt/sources.list

# Install<a></a>
sudo apt update
sudo apt install crystal

# Verify<a></a>
crystal --version
```

macOS

```
# Using Homebrew<a></a>
brew install crystal

# Or with MacPorts<a></a>
sudo port install crystal

# Verify<a></a>
crystal --version
```

VS Code Setup

1. Install extension: **"Crystal Language"** by Elias Perez
2. Install **Code Runner** extension for quick execution
3. Create `.vscode/settings.json`:

```
{
  "crystal.linter": "ameba",
  "crystal.logLevel": "info",
  "[crystal]": {
    "editor.formatOnSave": true,
    "editor.defaultFormatter": "crystal-lang-tools.crystal"
  }
}
```

Running & Compiling Crystal Files

```
# Run directly (slower, interpreted)<a></a>
crystal run hello.cr

# Compile to executable<a></a>
crystal build hello.cr --release

# Execute compiled binary<a></a>
./hello

# Debug build (includes debug info)<a></a>
crystal build hello.cr
```

```
# Release build (optimized)<a></a>
crystal build hello.cr --release
```

 Tips:

- Use `--release` flag for production builds
- Release builds are 3-10x faster than debug builds
- Compilation time is longer but worth it for performance

⚡ Commands & Syntax Reference

Basic Commands

Command	Purpose	Example
`crystal run`	Execute Crystal file directly	`crystal run app.cr`
`crystal build`	Compile to executable	`crystal build app.cr --release`
`crystal spec`	Run tests	`crystal spec`
`shards init`	Initialize new project	`shards init my_project`
`shards install`	Install dependencies	`shards install`
`crystal play`	Interactive REPL	`crystal play` (opens web interface)

Advanced Flags

```
# Specify output filename<a></a>
crystal build app.cr -o my_app

# Set threads<a></a>
crystal build app.cr --threads 4

# Show time spent on compilation<a></a>
crystal build app.cr --stats

# Inline all possible code<a></a>
crystal build app.cr --inline-threshold 10000

# Link libraries<a></a>
crystal build app.cr -L /usr/local/lib
```

Data Types & Variables

Primitive Data Types

Type	Description	Example
`Int32` / `Int64`	Integer numbers	`x = 42`, `y = 9223372036854775807`
`Float32` / `Float64`	Floating-point numbers	`pi = 3.14159`, `e = 2.718`
`String`	Text data	`name = "Rishi"`

Type	Description	Example
`Char`	Single character	`letter = 'A'`
`Bool`	True/false	`flag = true`, `active = false`
`Symbol`	Immutable, interned strings	`:status`, `:success`
`Array`	Ordered collection	`arr = [1, 2, 3]`
`Hash`	Key-value store	`map = {"a" =&gt; 1, "b" =&gt; 2}`

Type Inference vs Explicit Typing

```
# Type inference - Crystal figures out types automatically<a></a>
x = 10                    # Crystal infers Int32
y = 3.14                  # Crystal infers Float64
name = "Crystal"          # Crystal infers String

# Explicit typing - you specify types explicitly<a></a>
x : Int32 = 10
y : Float64 = 3.14
name : String = "Crystal"

# Variable declaration with types<a></a>
def greet(name : String) : String
  "Hello, #{name}!"
end
```

Working with Variables

```
# Underscore prefix ignores value<a></a>
_unused = 42

# Reassignment<a></a>
age = 20
age = 21  # ✅ valid

# Constants (must be CAPITALIZED)<a></a>
PI = 3.14159
MAX_USERS = 1000
```

➕➖ Operators

Arithmetic Operators

```
a = 10
b = 3

result = a + b      # 13 (addition)
result = a - b      # 7 (subtraction)
result = a * b      # 30 (multiplication)
result = a / b      # 3 (integer division)
result = a % b      # 1 (modulo)
result = a ** b     # 1000 (exponentiation)
```

Comparison Operators

```
a = 5
b = 10

a == b              # false (equal to)
a != b              # true (not equal to)
a &lt; b               # true (less than)
a &lt;= b              # true (less than or equal)
a &gt; b               # false (greater than)
a &gt;= b              # false (greater than or equal)

# Spaceship operator (returns -1, 0, or 1)<a></a>
a &lt;=&gt; b             # -1 (a is less than b)
```

Logical Operators

```
true &amp;&amp; true        # true (AND)
true || false       # true (OR)
!true               # false (NOT)

# Short-circuit evaluation<a></a>
x = false &amp;&amp; expensive_function()  # expensive_function() NOT called
y = true || expensive_function()   # expensive_function() NOT called
```

Assignment Operators

```
x = 10
x += 5              # x = 15
x -= 3              # x = 12
x *= 2              # x = 24
x /= 4              # x = 6
x %= 3              # x = 0
```

Operator Precedence (Highest to Lowest)

1. `**` (exponentiation)
2. `!`, `~` (logical/bitwise NOT)
3. `*`, `/`, `%` (multiplication, division, modulo)
4. `+`, `-` (addition, subtraction)
5. `<<`, `>>` (bitwise shift)
6. `&` (bitwise AND)
7. `|`, `^` (bitwise OR, XOR)
8. `>`, `>=`, `<`, `<=` (comparison)
9. `==`, `!=`, `===`, `!==` (equality)
10. `&&` (logical AND)
11. `||` (logical OR)
12. `? :` (ternary)
13. `=`, `+=`, `-=`, etc. (assignment)

Loops & Iteration

For Loop

```
# Iterate over range<a></a>
for i in 1..5
  puts i              # prints 1, 2, 3, 4, 5
end

# Iterate over array<a></a>
arr = ["a", "b", "c"]
for item in arr
  puts item
end

# Exclusive range<a></a>
for i in 1...5
  puts i              # prints 1, 2, 3, 4 (excludes 5)
end
```

While Loop

```
i = 0
while i &lt; 5
  puts i
  i += 1
end
```

Until Loop (opposite of while)

```
i = 0
until i &gt;= 5
  puts i
  i += 1
end
```

Loop Do (infinite loop with break)

```
counter = 0
loop do
  puts counter
  counter += 1
  break if counter &gt;= 5
end
```

Enumerable Methods

```
arr = [1, 2, 3, 4, 5]

# Each - iterate with block<a></a>
arr.each { |x| puts x }

# Map - transform elements<a></a>
squared = arr.map { |x| x ** 2 }    # [1, 4, 9, 16, 25]

# Select/Filter - keep matching elements<a></a>
evens = arr.select { |x| x % 2 == 0 }  # [2, 4]
```

```
# Reduce - combine all elements<a></a>
sum = arr.reduce(0) { |acc, x| acc + x }  # 15

# Any/All - check conditions<a></a>
has_even = arr.any? { |x| x % 2 == 0 }    # true
all_positive = arr.all? { |x| x &gt; 0 }      # true
```

Nested Loops with Flow Control

```
for i in 1..3
  for j in 1..3
    next if i == j              # skip current iteration
    break if i * j &gt; 6           # exit inner loop
    puts "#{i} × #{j} = #{i * j}"
  end
end
```

❓ Conditional Statements

If, Elsif, Else

```
age = 20

if age &lt; 13
  puts "Child"
elsif age &lt; 18
  puts "Teenager"
elsif age &lt; 65
  puts "Adult"
else
  puts "Senior"
end
```

Unless (opposite of if)

```
status = "active"

unless status == "inactive"
  puts "Processing..."
end

# Inline unless<a></a>
puts "Active!" unless status == "inactive"
```

Case Statement

```
grade = 'A'

case grade
when 'A'
  puts "Excellent!"
when 'B'
  puts "Good!"
when 'C'
  puts "Average"
when 'D'
```

```
  puts "Poor"
else
  puts "Invalid grade"
end

# With ranges<a></a>
score = 85
case score
when 90..100
  puts "A"
when 80..89
  puts "B"
when 70..79
  puts "C"
else
  puts "F"
end
```

Ternary Operator

```
age = 20
status = age &gt;= 18 ? "Adult" : "Minor"

# Nested ternary (use sparingly)<a></a>
category = score &gt;= 90 ? "A" : score &gt;= 80 ? "B" : "C"
```

Inline Conditionals

```
x = 10
x += 5 if x &lt; 20          # conditional assignment
puts "Valid" if x &gt; 0

puts "Done" unless errors.empty?
```

Functions & Sub-Functions

Basic Function Declaration

```
# Simple function<a></a>
def greet
  puts "Hello!"
end

greet()                 # Call function

# Function with parameters<a></a>
def add(a, b)
  a + b
end

result = add(3, 5)      # result = 8

# Function with return type<a></a>
def multiply(a : Int32, b : Int32) : Int32
  a * b
end
```

Default Parameters & Optional Arguments

```
def introduce(name : String, age : Int32 = 20, city : String = "Unknown")
  "#{name} is #{age} years old and lives in #{city}"
end

introduce("Alice")                    # uses defaults
introduce("Bob", 25)                  # custom age
introduce("Charlie", 30, "New York")  # all custom
```

Variable Arguments (*args)

```
def sum(*numbers : Int32) : Int32
  total = 0
  numbers.each { |n| total += n }
  total
end

sum(1, 2, 3, 4, 5)     # 15
```

Keyword Arguments

```
def create_user(name : String, email : String, active : Bool = true)
  # ...
end

# Call with keyword arguments<a></a>
create_user(name: "Alice", email: "alice@example.com")
create_user(email: "bob@example.com", name: "Bob", active: false)
```

Nested Functions & Sub-Functions

```
def outer(x : Int32)
  def inner(y : Int32)
    x + y                       # has access to outer's x
  end

  inner(10)
end

result = outer(5)               # result = 15
```

Function Call Hierarchy

```
# Main function<a></a>
def calculate_total(items : Array(Int32)) : Int32
  subtotal = compute_subtotal(items)
  tax = calculate_tax(subtotal)
  subtotal + tax
end

# Sub-function 1<a></a>
def compute_subtotal(items : Array(Int32)) : Int32
  items.reduce(0) { |sum, item| sum + item }
end

# Sub-function 2<a></a>
```

```
def calculate_tax(amount : Int32) : Int32
  (amount * 0.1).to_i
end

result = calculate_total([100, 200, 300])
```

Blocks & Yield

```
def with_timing
  start = Time.now
  yield                          # execute the block
  elapsed = Time.now - start
  puts "Took #{elapsed.total_milliseconds}ms"
end

with_timing do
  sleep 1
  puts "Did something"
end
```

Best Practices for Functions

- ✅ Keep functions focused on a single responsibility
- ✅ Use descriptive names that indicate purpose
- ✅ Specify parameter and return types for clarity
- ✅ Use default parameters for optional behavior
- ✅ Document complex functions with comments
- ✅ Test functions with various inputs

🏗️ Object-Oriented Programming

Classes & Objects

```
class User
  # Properties with type annotations
  @name : String
  @age : Int32
  @email : String

  # Constructor (initialize method)
  def initialize(name : String, age : Int32, email : String)
    @name = name
    @age = age
    @email = email
  end

  # Getter methods
  def name
    @name
  end

  def age
    @age
  end

  # Instance method
  def display_info
```

```
    "#{@name} (#{@age}) - #{@email}"
  end

  # Setter method
  def age=(new_age : Int32)
    @age = new_age
  end
end

# Create object<a></a>
user = User.new("Alice", 25, "alice@example.com")
puts user.display_info

# Access properties<a></a>
puts user.name
user.age = 26
```

Property Shortcuts

```
class Person
  # Automatic getter and setter
  property name : String

  # Automatic getter only
  getter age : Int32

  # Automatic setter only
  setter email : String

  def initialize(name : String, age : Int32, email : String)
    @name = name
    @age = age
    @email = email
  end
end

person = Person.new("Bob", 30, "bob@example.com")
person.name = "Robert"        # uses setter
puts person.age               # uses getter
```

Inheritance

```
class Animal
  @name : String

  def initialize(name : String)
    @name = name
  end

  def speak
    "#{@name} makes a sound"
  end
end

class Dog &lt; Animal
  def speak
    "#{@name} barks: Woof!"
  end

  def fetch(item : String)
    "#{@name} fetches the #{item}"
  end
```

```
end

dog = Dog.new("Buddy")
puts dog.speak                # "Buddy barks: Woof!"
puts dog.fetch("ball")        # "Buddy fetches the ball"
```

Modules & Mixins

```
module Walkable
  def walk
    "Walking..."
  end
end

module Swimmable
  def swim
    "Swimming..."
  end
end

class Dog
  include Walkable
  include Swimmable
end

dog = Dog.new
puts dog.walk                 # "Walking..."
puts dog.swim                 # "Swimming..."
```

Encapsulation

```
class BankAccount
  @balance : Float64

  def initialize(initial_balance : Float64)
    @balance = initial_balance
  end

  # Private method - can't be called from outside
  private def validate_amount(amount : Float64) : Bool
    amount &gt; 0
  end

  def deposit(amount : Float64)
    if validate_amount(amount)
      @balance += amount
      "Deposited $#{amount}"
    else
      "Invalid amount"
    end
  end

  def balance
    @balance
  end
end

account = BankAccount.new(1000)
puts account.deposit(500)      # works
puts account.validate_amount(100)  # ✖ Error: private method
```

Collections

Arrays

```
# Array creation<a></a>
arr = [1, 2, 3, 4, 5]
arr = Array(Int32).new
arr = [1, 2, 3]

# Common methods<a></a>
arr.size              # 3
arr.first             # 1
arr.last              # 3
arr.empty?            # false
arr.includes?(3)      # true

# Adding/removing elements<a></a>
arr.push(6)           # [1, 2, 3, 4, 5, 6]
arr &lt;&lt; 7              # [1, 2, 3, 4, 5, 6, 7]
arr.pop               # removes last
arr.shift             # removes first
arr.unshift(0)        # adds to beginning

# Iteration with transformation<a></a>
doubled = arr.map { |x| x * 2 }
evens = arr.select { |x| x % 2 == 0 }
sum = arr.reduce(0) { |acc, x| acc + x }
```

Hashes

```
# Hash creation<a></a>
user = {
  "name" =&gt; "Alice",
  "age" =&gt; 25,
  "city" =&gt; "NYC"
}

user = {} of String =&gt; Int32    # empty typed hash

# Accessing values<a></a>
user["name"]          # "Alice"
user.fetch("age", 0)  # 25 (or 0 if key missing)

# Adding/modifying<a></a>
user["email"] = "alice@example.com"
user["age"] = 26

# Hash methods<a></a>
user.keys             # ["name", "age", "city", "email"]
user.values           # ["Alice", 26, "NYC", "alice@example.com"]
user.size             # 4
user.empty?           # false
user.has_key?("name") # true

# Iterating<a></a>
user.each { |key, value| puts "#{key}: #{value}" }
user.keys.each { |key| puts key }
```

Sets

```
# Set creation (no duplicates)<a></a>
numbers = {1, 2, 3, 4, 5}.to_set
numbers = Set(Int32).new([1, 2, 3])

# Set operations<a></a>
numbers.add(6)
numbers.includes?(3)  # true
numbers.size          # number of unique elements

# Set algebra<a></a>
set_a = {1, 2, 3}.to_set
set_b = {3, 4, 5}.to_set

union = set_a | set_b              # {1, 2, 3, 4, 5}
intersection = set_a &amp;amp; set_b       # {3}
difference = set_a - set_b         # {1, 2}
```

Tuples

```
# Fixed-size, heterogeneous collection<a></a>
point = {10, 20}
point_3d = {10, 20, 30}

# Accessing elements<a></a>
x = point[0]         # 10
y = point[1]         # 20

# Pattern matching<a></a>
x, y = point         # x = 10, y = 20
```

NamedTuples

```
# Named tuple - like a lightweight object<a></a>
person = {name: "Alice", age: 25, city: "NYC"}

person[:name]        # "Alice"
person["name"]       # "Alice"

# Pattern matching<a></a>
{name: n, age: a} = person
puts "#{n} is #{a}"
```

Blocks, Closures & Iterators

Block Syntax

```
# Block with single parameter<a></a>
[1, 2, 3].each { |x| puts x }

# Block with multiple lines<a></a>
[1, 2, 3].each do |x|
  squared = x ** 2
  puts squared
end

# Block parameters<a></a>
```

```
hash = {"a" =&gt; 1, "b" =&gt; 2}
hash.each { |key, value| puts "#{key}: #{value}" }
```

Passing Blocks to Functions

```
def process_with_callback
  puts "Before"
  yield                          # execute the block
  puts "After"
end

process_with_callback do
  puts "Inside block"
end

# With parameters<a></a>
def with_value
  yield(42)
end

with_value { |n| puts "Got #{n}" }

# Multiple yields<a></a>
def triple_processing
  yield "first"
  yield "second"
  yield "third"
end

triple_processing { |n| puts n }
```

Lambdas & Procs

```
# Lambda (typed)<a></a>
square = -&gt;(x : Int32) { x ** 2 }
result = square.call(5)        # 25

# Proc (flexible)<a></a>
add = -&gt;(a : Int32, b : Int32) { a + b }
puts add.call(3, 7)            # 10

# Passing lambdas<a></a>
def apply_twice(func : Proc(Int32, Int32))
  x = func.call(5)
  func.call(x)
end

result = apply_twice(-&gt;(n : Int32) { n * 2 })  # 20
```

Closures

```
def make_counter
  count = 0

  return -&gt;{
    count += 1           # closure captures count variable
    count
  }
end
```

```
counter = make_counter
puts counter.call()     # 1
puts counter.call()     # 2
puts counter.call()     # 3
```

Custom Iterators

```
class Range
  def each_with_index
    i = 0
    each do |value|
      yield value, i
      i += 1
    end
  end
end

(1..3).each_with_index do |val, idx|
  puts "#{idx}: #{val}"
end
```

⚠ Error Handling

Begin, Rescue, Ensure

```
begin
  file = File.read("data.txt")
  lines = file.split('\n')
  count = lines.size
rescue
  puts "Error reading file"
rescue ex : Exception
  puts "Caught exception: #{ex.message}"
ensure
  puts "Cleanup code runs regardless"
end
```

Raising Exceptions

```
def divide(a : Int32, b : Int32) : Float64
  if b == 0
    raise "Cannot divide by zero"
  end
  a / b.to_f
end

begin
  result = divide(10, 0)
rescue ex : Exception
  puts "Error: #{ex.message}"
end
```

Custom Exceptions

```
class InvalidAgeError &lt; Exception
end

def set_age(age : Int32)
  if age &lt; 0 || age &gt; 150
    raise InvalidAgeError.new("Age must be between 0 and 150")
  end
end

begin
  set_age(-5)
rescue ex : InvalidAgeError
  puts "Age error: #{ex.message}"
end
```

Best Practices for Error Handling

✓ Catch specific exception types when possible
✓ Provide meaningful error messages
✓ Use ensure for cleanup operations
✓ Don't swallow exceptions silently
✓ Create custom exceptions for domain-specific errors

Concurrency

Fibers (Lightweight Threads)

```
def task(name : String)
  3.times do |i|
    puts "#{name} - #{i}"
    Fiber.yield               # yield control to other fibers
  end
end

# Create multiple fibers<a></a>
f1 = spawn { task("Task 1") }
f2 = spawn { task("Task 2") }

# Wait for completion<a></a>
Fiber.yield until f1.dead? &amp;&amp; f2.dead?
```

Channels (Communication)

```
channel = Channel(String).new

# Send data to channel<a></a>
spawn do
  channel.send("Hello")
  channel.send("World")
  channel.close
end

# Receive data<a></a>
while msg = channel.receive?
```

```
  puts msg
end
```

Multiple Senders & Receivers

```
channel = Channel(Int32).new

# Multiple producers<a></a>
3.times do |i|
  spawn do
    5.times do |j|
      channel.send(i * 10 + j)
      sleep 0.1
    end
  end
end

# Consumer<a></a>
15.times do
  puts channel.receive
end
```

✨ Macros & Compile-Time Programming

Macro Basics

```
macro greet(name)
  puts "Hello, {{ name }}"
end

greet("Alice")          # compiled as: puts "Hello, Alice"

# Macros with string interpolation<a></a>
macro define_method(name, body)
  def {{ name }}
    {{ body }}
  end
end

define_method(say_hello, puts "Hello!")
say_hello()
```

Looping Macros

```
macro define_getters(*names)
  {% for name in names %}
    def {{ name }}
      @{{ name }}
    end
  {% end %}
end

class Person
  define_getters(name, age, email)
end
```

Compile-Time Evaluation

```
macro if_debug
  {% if flag?(:debug) %}
    puts "Debug mode"
  {% else %}
    puts "Release mode"
  {% end %}
end

if_debug()
# Compile with: crystal build --debug<a></a>
```

Libraries & Packages

Standard Library - Key Modules

Library	Purpose	Example
Array	Array operations	`arr.map`, `arr.select`
Hash	Hash/dictionary operations	`hash.each`, `hash.keys`
String	String manipulation	`str.upcase`, `str.split`
File	File I/O operations	`File.read("file.txt")`
IO	Input/output	`puts`, `print`, `gets`
JSON	JSON parsing/generation	`JSON.parse`, `obj.to_json`
HTTP	HTTP requests	`HTTP::Client.get(url)`
Time	Date and time	`Time.now`, `Time.parse`
Math	Mathematical functions	`Math.sqrt`, `Math.sin`

Working with JSON

```
require "json"

# Parse JSON<a></a>
json_string = %({"name":"Alice","age":25})
data = JSON.parse(json_string)
puts data["name"]

# Generate JSON<a></a>
obj = {name: "Bob", age: 30}
json = obj.to_json
puts json

# Custom serialization<a></a>
class Person
  def initialize(@name : String, @age : Int32)
  end

  def to_json(builder : JSON::Builder)
    builder.object do
      builder.field "name", @name
      builder.field "age", @age
```

```
    end
  end
end
```

HTTP Requests

```
require "http/client"

# GET request<a></a>
response = HTTP::Client.get("https://api.example.com/users")
puts response.status_code
puts response.body

# POST request<a></a>
response = HTTP::Client.post(
  "https://api.example.com/users",
  headers: HTTP::Headers{"Content-Type" =&gt; "application/json"},
  body: {name: "Alice", age: 25}.to_json
)
```

External Libraries (Shards)

Popular Crystal packages installed via shards:

Shard	Purpose	Installation
Kemal	Web framework	`dependencies: kemal: "*"`
Sidekiq	Job queue	`dependencies: sidekiq: "*"`
Pg	PostgreSQL driver	`dependencies: pg: "*"`
DB	Database abstraction	`dependencies: db: "*"`
Ameba	Code linter	`dependencies: ameba: "*"`

Mini Projects

Project 1: Temperature Converter CLI

Purpose: Convert temperatures between Celsius, Fahrenheit, and Kelvin

Functions Used:

- `celsius_to_fahrenheit(c : Float64) : Float64`
- `celsius_to_kelvin(c : Float64) : Float64`
- `fahrenheit_to_celsius(f : Float64) : Float64`

Key Code Snippets:

```
def celsius_to_fahrenheit(c : Float64) : Float64
  (c * 9/5) + 32
end

def celsius_to_kelvin(c : Float64) : Float64
  c + 273.15
end
```

```
puts "Enter temperature in Celsius:"
temp = gets.not_nil!.to_f

puts "°F: #{celsius_to_fahrenheit(temp).round(2)}"
puts "K: #{celsius_to_kelvin(temp).round(2)}"
```

Output Example:

```
Enter temperature in Celsius:
25
°F: 77.0
K: 298.15
```

Project 2: To-Do List Manager

Purpose: Simple CLI task management system

Libraries: File I/O, JSON

Key Features:

- Add tasks
- List all tasks
- Mark tasks complete
- Delete tasks

Core Structure:

```
class Task
  property id : Int32
  property title : String
  property completed : Bool

  def initialize(@id, @title, @completed = false)
  end

  def to_json(builder : JSON::Builder)
    builder.object do
      builder.field "id", @id
      builder.field "title", @title
      builder.field "completed", @completed
    end
  end
end

class TodoList
  @tasks : Array(Task) = []

  def add_task(title : String) : Task
    id = @tasks.empty? ? 1 : @tasks.last.id + 1
    task = Task.new(id, title)
    @tasks &lt;&lt; task
    task
  end

  def list_tasks
    @tasks.each { |t| puts "#{t.id}. [#{t.completed ? "x" : " "}] #{t.title}" }
  end

  def complete_task(id : Int32)
```

```
    task = @tasks.find { |t| t.id == id }
    task.completed = true if task
  end
end
```

Project 3: HTTP Weather Client

Purpose: Fetch and display weather data from an API

Libraries: HTTP::Client, JSON

Key Features:

- Fetch weather by city name
- Display temperature, humidity, conditions
- Error handling for invalid cities

Implementation:

```
require "http/client"
require "json"

def get_weather(city : String)
  begin
    url = "https://api.open-meteo.com/v1/forecast?latitude=0&amp;longitude=0&amp;current_we
    response = HTTP::Client.get(url)

    if response.status_code == 200
      data = JSON.parse(response.body)
      puts "Weather data retrieved for #{city}"
    else
      puts "Error: Could not fetch weather data"
    end
  rescue ex
    puts "Exception: #{ex.message}"
  end
end

get_weather("London")
```

Project 4: File Word Counter

Purpose: Count words, lines, and characters in files

Libraries: File I/O

Key Functions:

- `count_words(file_path : String) : Int32`
- `count_lines(file_path : String) : Int32`
- `count_chars(file_path : String) : Int32`

Code:

```
def analyze_file(file_path : String)
  begin
    content = File.read(file_path)
```

```
    words = content.split.size
    lines = content.lines.size
    chars = content.size

    puts "File Analysis: #{file_path}"
    puts "Lines: #{lines}"
    puts "Words: #{words}"
    puts "Characters: #{chars}"
  rescue ex
    puts "Error: #{ex.message}"
  end
end

analyze_file("document.txt")
```

Project 5: Simple Calculator with History

Purpose: Calculator with operation history tracking

Features:

- Basic arithmetic operations
- Operation history (last 10 operations)
- Clear history

Implementation:

```
class Calculator
  @history : Array(String) = []

  def calculate(a : Float64, b : Float64, op : String) : Float64
    result = case op
    when "+"
      a + b
    when "-"
      a - b
    when "*"
      a * b
    when "/"
      b == 0 ? raise "Division by zero" : a / b
    else
      raise "Invalid operation"
    end

    @history &lt;&lt; "#{a} #{op} #{b} = #{result}"
    result
  end

  def show_history
    @history.each { |entry| puts entry }
  end
end

calc = Calculator.new
puts calc.calculate(10, 5, "+")
puts calc.calculate(20, 4, "*")
calc.show_history
```

Project 6: Concurrent Download Manager

Purpose: Download multiple files concurrently using Fibers

Libraries: HTTP::Client, Fiber

Key Concept: Managing concurrent downloads with error handling

```
require "http/client"

class DownloadManager
  @downloads : Array(Fiber) = []

  def add_download(url : String, filename : String)
    fiber = spawn do
      download(url, filename)
    end
    @downloads &lt;&lt; fiber
  end

  private def download(url : String, filename : String)
    begin
      response = HTTP::Client.get(url)
      File.write(filename, response.body)
      puts "✓ Downloaded #{filename}"
    rescue ex
      puts "✗ Failed to download #{filename}: #{ex.message}"
    end
  end

  def wait_all
    @downloads.each { |f| Fiber.yield until f.dead? }
  end
end

manager = DownloadManager.new
manager.add_download("https://example.com/file1.txt", "file1.txt")
manager.add_download("https://example.com/file2.txt", "file2.txt")
manager.wait_all
puts "All downloads complete!"
```

Advanced Topics & Tips

Performance Optimization

```
# Release build is essential for performance<a></a>
crystal build app.cr --release

# Use profiling to find bottlenecks<a></a>
crystal build app.cr --stats

# Inline frequently called small methods<a></a>
def small_operation
  # code that gets inlined
end
```

Project Structure Best Practices

```
my_project/
├── shard.yml
├── src/
│   ├── main.cr
│   ├── models/
│   ├── helpers/
│   └── utils/
├── spec/
│   └── *_spec.cr
└── README.md
```

Debugging Tips

```
# Print debug info<a></a>
puts "DEBUG: value = #{value}"

# Use exception messages<a></a>
begin
  dangerous_operation
rescue ex : Exception
  puts "Exception occurred: #{ex.message}"
  puts "Backtrace: #{ex.backtrace}"
end

# Compile with debug symbols<a></a>
crystal build app.cr --debug
```

Concurrency Best Practices

- ✅ Use Channels for thread-safe communication
- ✅ Avoid sharing mutable state between Fibers
- ✅ Use `spawn` for independent tasks
- ✅ Always close channels when done
- ✅ Handle deadlocks by structuring communication patterns

📝 Personal Notes & Observations

Challenges & Solutions

Challenge 1: Type Errors at Compile Time

- *Solution:* Embrace type checking; it catches bugs early that would be runtime errors in Ruby
- *Tip:* Use type inference when obvious, explicit types for function signatures

Challenge 2: Understanding Fibers vs Threads

- *Solution:* Fibers are cooperative multitasking; channels are for safe inter-fiber communication
- *Tip:* Think of Fibers as green threads managed by Crystal runtime

Challenge 3: Macro Complexity

- *Solution:* Macros execute at compile-time; use them for code generation, not runtime logic
- *Tip:* Start simple; advanced metaprogramming requires deep understanding

Tips for Beginners

1. **Start with Ruby-like syntax** but remember you're writing compiled code
2. **Use type annotations early** to catch errors during development
3. **Test concurrency carefully** - spawn tasks incrementally and verify correctness
4. **Read Crystal's documentation** - it's excellent and has many examples
5. **Use shard.yml** for dependencies even in small projects
6. **Profile before optimizing** - use `--stats` flag to see compilation costs
7. **Leverage standard library** - it's comprehensive and well-designed

Ideas for New Projects

- Web framework exploration (Kemal, Amber)
- WebSocket real-time chat application
- Machine learning with Crystal
- System monitoring dashboard
- Event-driven distributed system
- Game development with libraries like Raylib bindings

Mastering Crystal

The path to mastery involves:

- ✓ Fluent syntax knowledge
- ✓ Deep understanding of type system
- ✓ Proficiency with concurrency patterns
- ✓ Experience optimizing for performance
- ✓ Building real-world applications
- ✓ Contributing to Crystal ecosystem

Appendix

Complete Command Reference

```
# Project management<a></a>
shards init PROJECT_NAME        # Initialize new project
shards install                  # Install dependencies
shards update                   # Update dependencies

# Compilation<a></a>
crystal run FILE.cr             # Run directly
crystal build FILE.cr           # Debug build
crystal build FILE.cr --release # Optimized build
crystal play                    # Interactive REPL

# Testing &amp;amp; Quality<a></a>
crystal spec                    # Run tests
ameba                          # Lint code
crystal docs                   # Generate documentation

# Flags for optimization<a></a>
--release                       # Production build
```

```
--threads 4                     # Specify thread count
--static                        # Static linking
--cross-compile linux x86_64    # Cross compilation
```

Functions & Sub-Functions Quick Reference

Category	Function	Purpose
String	`upcase`, `downcase`, `capitalize`	Case conversion
String	`split`, `strip`, `reverse`	Manipulation
Array	`map`, `select`, `reduce`	Transformation
Array	`sort`, `uniq`, `compact`	Organization
Hash	`keys`, `values`, `merge`	Access
Hash	`each`, `select`, `reject`	Iteration
Math	`sqrt`, `sin`, `cos`, `tan`	Mathematical
Time	`now`, `parse`, `format`	Date/Time

Libraries & Usage Examples

```
# File Operations<a></a>
require "file_utils"
File.read("input.txt")
File.write("output.txt", content)
Dir.glob("*.cr")

# JSON<a></a>
require "json"
data = JSON.parse(json_string)
json = obj.to_json

# HTTP<a></a>
require "http/client"
response = HTTP::Client.get(url)

# Time<a></a>
require "time"
now = Time.now
formatted = now.to_s("%Y-%m-%d")
```

End of Crystal101 Complete Encyclopedia & Masterclass

Created with dedication to mastering the Crystal programming language. Happy coding! ✨