



OVERVIEW & INSTALL

RPG modern, expressive programming language

```
# Check version
```

```
rpg --version
```

Install via official installer,

package manager, or version manager

Popular package managers: cargo,

pip, gem, npm, brew

BASIC SYNTAX

Variables, control flow, expressions

```
// Variable declaration
```

```
let x = 10;
```

```
// Conditional
```

```
if x > 5 { ... } else { ... }
```

```
// Loop
```

```
for item in collection { ... }
```

DATA TYPES

Primitive: integers, floats,

booleans, strings

Composite: arrays, maps, tuples,

structs

```
// Typed declaration
```

```
let name: String = "Alice";
```

```
let numbers: Vec<i32> = vec![1, 2, 3];
```

Strong typing prevents common

runtime errors

FUNCTIONS

```
fn greet(name: &str) -> String {
```

```
    format!("Hello, {}!", name)
```

```
}
```

```
let result = greet("World");
```

First-class functions;

lambdas/closures supported

```
let double = |x| x * 2;
```

OOP / MODULES

Structs, classes, or records with

methods

```
struct User { name: String, age: u32 }
```

```
impl User {
```

```
    fn new(name: &str) -> Self { ... }
```

```
    fn greet(&self) -> String { ... }
```

```
}
```

Interfaces / traits / protocols for

polymorphism

COLLECTIONS

```
// Array / slice
```

```
let arr = [1, 2, 3];
```

```
// Dynamic list
```

```
let mut list = Vec::new();
```

```
list.push(42);
```

```
// Map / dictionary
```

```
let mut map = HashMap::new();
```

```
map.insert("key", "value");
```

ERROR HANDLING

Result/Option types or exceptions

```
fn read_file(path: &str) ->
```

```
    Result<String, Error> {
```

```
    Ok(std::fs::read_to_string(path)?)
```

```
}
```

```
match result {
```

```
    Ok(data) => process(data),
```

```
    Err(e) => eprintln!("Error: {}", e),
```

```
}
```

CONCURRENCY / ASYNC

Threads, async/await, or actor model

```
async fn fetch_data(url: &str) ->
```

```
    Result<String> {
```

```
    let resp = client.get(url).await?;
```

```
    resp.text().await
```

```
}
```

```
tokio::main / async_std / smol
```

Green threads or OS threads

depending on runtime

TESTING

Built-in or framework test support

```
#[test]
```

```
fn test_greet() {
```

```
    assert_eq!(greet("Alice"), "Hello,
```

```
    Alice!");
```

```
}
```

```
cargo test / run test suite
```

Unit tests co-located with source

code

ECOSYSTEM & PACKAGES

Package registry and dependency

management

```
# Add dependency
```

```
cargo add serde
```

```
# In Cargo.toml:
```

```
[dependencies]
```

```
serde = { version = "1.0", features =
```

```
    ["derive"] }
```

Curated ecosystem with popular

libraries for HTTP, JSON, DB

CLI & BUILD TOOLS

```
cargo build    compile project
```

```
cargo run      build and run
```

```
cargo test     run tests
```

```
cargo fmt      format code
```

```
cargo clippy   linting
```

```
cargo doc      generate documentation
```

Integrated toolchain for build,

test, and docs

COMMON GOTCHAS

Follow language-specific naming

conventions

Understand ownership/borrowing/GC

model for this language

Use idiomatic patterns don't port

code from other languages

Check community guidelines for error

handling approach

Profile before optimizing

correctness first