Rust Cheatsheet

Abyan Majid, 2023

Cargo

Concept	Snippets/Command/Explanation/Examples
Create a new project	`$ cargo new <project_name>`
Compile into an executable	`$ cargo build`
Compile into an executable and then run	`$ cargo run`
Check if code will compile, without building an executable	`$ cargo check`
Build for release	`$ cargo build --release`
Update dependencies	`$ cargo update`
Build the documentation for your dependencies	`$ cargo doc --open`

Variables

Concept	Snippets/Command/Explanation/Examples

Declare immutable variable	`// can only be declared in functions`
	`let x = 5;`

Declare mutable variable	`// can only be declared in functions`
	`let mut x = 5;`

Constants	`// can be in functions or global scope`
	`const x: u32 = 5;`

Shadowing	`let x = 5;`
	`let x = "hello";`

Statements vs. Expressions	`// statements end with a semicolon, expressions do not.`
	`// besides a syntactic scope, {/* */} also denotes an expression.`
	`let y = {`
	`    let x = 3; // statement`
	`    x + 1 // expression`
	`}; // statement`

Data Types

Concept	Snippets/Command/Explanation/Examples
(SCALAR) Integer	Options: (see table below) Default: i32
(SCALAR) Float	Options: f32 (single precision), f64 (double precision) Default: f64
(SCALAR) Boolean	Options: true, false (see code below)
(SCALAR) Char	(see code below)
(COMPOUND) Tuple	(see code below)
(COMPOUND) Array	(see code below)

(SCALAR) Integer

Options:

Length	Signed	Unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

Default: i32

(SCALAR) Float

Options: f32 (single precision), f64 (double precision)
Default: f64

(SCALAR) Boolean

Options: true, false

```
let x = true;
let y: bool = false; // with explicit type annotation
```

(SCALAR) Char

```
// use single quotes
let c = 'z';
let z: char = 'ℤ'; // with explicit type annotation
```

(COMPOUND) Tuple

```
// fixed size; cannot grow or shrink, the elements may have different types
let tup: (i32, f64, u8) = (500, 6.4, 1);

// destructure a tuple
let (x, y, z) = tup;

// indexing tuples
let a = tup.0; // a = 500

// empty tuple “()” is called a unit
```

(COMPOUND) Array

```
// fixed size; cannot grow or shrink, the elements must have the same type
let arr = [1, 2, 3, 4, 5];

// with type and length annotation
let arr: [i32; 5] = [1, 2, 3, 4, 5]; // type: i32, length: 5

// initialise an array of the same values
let arr = [3; 5]; // the same as arr = [3, 3, 3, 3, 3]

// indexing arrays
let first = arr[0]'
```

Numeric Operations

Concept	Snippets/Command/Explanation/Examples
Addition	`let sum = 5 + 10;`
Subtraction	`let difference = 95.5 - 4.3;`
Multiplication	`let product = 4 * 30;`

Division	`// division truncates towards zero`
	`let quotient = 56.7 / 32.2;`
	`let truncated = -5 / 3; // Results in -1`

Remainder	`let remainder = 43 % 5;`

Functions

Concept	Snippets/Command/Explanation/Examples

Function	`// if not a return type is not specified, all functions default to returning a unit/empty tuple ie. ()`
	
	`fn plus_one(arg:i32) -> i32 {`
	`    return arg + 1;`
	`}`
	
	`fn main() {`
	`    let x = plus_one(5);`
	`    println!("The value of x is {x}."); // The value of x is 6.`
	`}`