

**Fundamentals** | Functions

# ■ What are functions?

- ◆ A way to encapsulate program functionality
- ◆ Optionally accept data
- ◆ Optionally return data
- ◆ Utilized for code organization
  - Also makes code easier to read

# ■ Anatomy of a function

```
fn add(a: i32, b: i32) -> i32 {  
    a + b  
}
```

# ■ Anatomy of a function

```
      Name  
      └─┬─  
fn add(a: i32, b: i32) -> i32 {  
    a + b  
}
```

# ■ Anatomy of a function

Parameters

Name

```
fn add(a: i32, b: i32) -> i32 {  
    a + b  
}
```

# ■ Anatomy of a function

```
fn add(a: i32, b: i32) -> i32 {  
    a + b  
}
```

Diagram illustrating the anatomy of a function signature:

- Name:** `add` (indicated by a purple bracket)
- Parameters:** `(a: i32, b: i32)` (indicated by a brown bracket)
- Return Type:** `-> i32` (indicated by a pink bracket)

The function body is enclosed in curly braces `{ ... }` and contains the expression `a + b`.

# ■ Anatomy of a function

Diagram illustrating the anatomy of a function signature and body:

```
fn add(a: i32, b: i32) -> i32 {  
    a + b  
}
```

The components are labeled as follows:

- fn**: Keyword
- add**: Name
- (a: i32, b: i32)**: Parameters
- >**: Arrow
- i32**: Return Type
- {**: Opening brace
- a + b**: Body
- }**: Closing brace

## Using a function

```
fn add(a: i32, b: i32) -> i32 {  
    a + b  
}
```

```
let x = add(1, 1);
```

```
let y = add(3, 0);
```

```
let z = add(x, 1);
```

# ■ Recap

- ◆ Functions encapsulate functionality
- ◆ Useful to organize code
- ◆ Can be executed by “calling” the function
- ◆ Parameters determine what data a function can work with
- ◆ Optionally “returns” data
  - Data sent back from the function