# Variables and Types

Epos is statically typed with type inference, making it both safe and expressive.

## Variable Declaration

Variables in Epos are immutable:

```
name: string = "Alice"
age: int = 25
height: float = 5.8
is-active: bool = true
```

You can also use type inference:

```
name := "Alice"       # inferred as string
age := 25             # inferred as int
height := 5.8         # inferred as float
is-active := true     # inferred as bool
```

## Built-in Types

### Primitive Types

- `int`: Integer numbers
- `float`: Floating-point numbers
- `string`: Text strings
- `bool`: Boolean values (true/false)

### String Interpolation

Embed expressions in strings using `#{}`:

```
name := "World"
message: string = "Hello #{name}!"
count := 42
status: string = "There are #{count} items"
```

### Lists

Lists are homogeneous collections:

```
numbers: list(int) = {1, 2, 3, 4, 5}
names: list(string) = {"Alice", "Bob", "Charlie"}
flags: list(bool) = {true, false, true}

# Nested lists
matrix: list(list(int)) = {{1, 2}, {3, 4}, {5, 6}}
```

Ranges are also supported:

```
range: list(int) = 1..10  # 1, 2, 3, ..., 10
```

Get the length of a list using `len()`:

```
len({1, 2, 3}) # => 3
```

Access list elements using `elem()`:

```
first: int = elem(numbers, 0)      # Gets first element
last: int = numbers.elem(len(numbers) - 1)  # Gets last element
```

### Records

Records are structured data types:

```
record Person
  name: string
  age: int
  email?: string    # Optional field
end

person: Person = @{
  name => "Alice",
  age => 30,
  email => "alice@example.com"
}

# Access record fields
print(person.name)      # Dot notation
print(person["age"])    # Bracket notation
```

## Type Aliases

Create aliases for complex types:

```
type UserId = int
type EmailAddress = string
type UserList = list(Person)

user-id: UserId = 123
emails: list(EmailAddress) = {"alice@example.com", "bob@example.com"}
```

## Union Types

Union types allow you to combine different types:

```
type UserId = int | string

user-id: UserId = 123
user-id: UserId = "alice@example.com"
```

## Generics

Epos supports generic types for reusable code:

```
# Generic function that works with any type
fn identity(value: t): t
  value
end

# Generic record
record Pair(a, b)
  first: a
  second: b
end

coord: Pair(int, int) = @{
  first => 10,
  second => 20
}
```

## Visibility

By default, everything is private. Use "*" for public declarations:

```
secret-key: string = "abc123"    # Private variable
name*: string = "public-name"    # Public variable

fn internal-helper(): int
  42
end

fn public-function*(): int
  internal-helper()
end
```

Next, learn about functions in Epos.