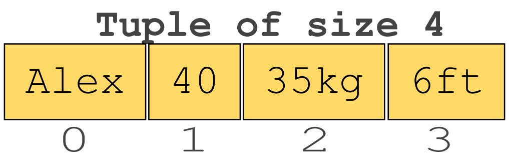
Tuples

This lesson will discuss a compound data type, Tuples.

We'll cover the following

What are Tuples?

Tuples are **heterogeneous sequences of elements**, meaning, each element in a tuple can have a different data type. Just like arrays, tuples are of a fixed length.

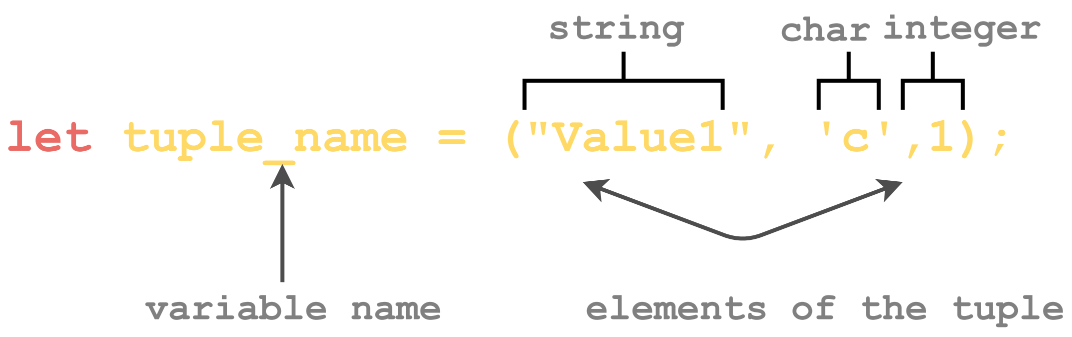


Define a Tuple

A tuple can be defined by writing `let` followed by the name of the tuple and then enclosing the values within the parenthesis.

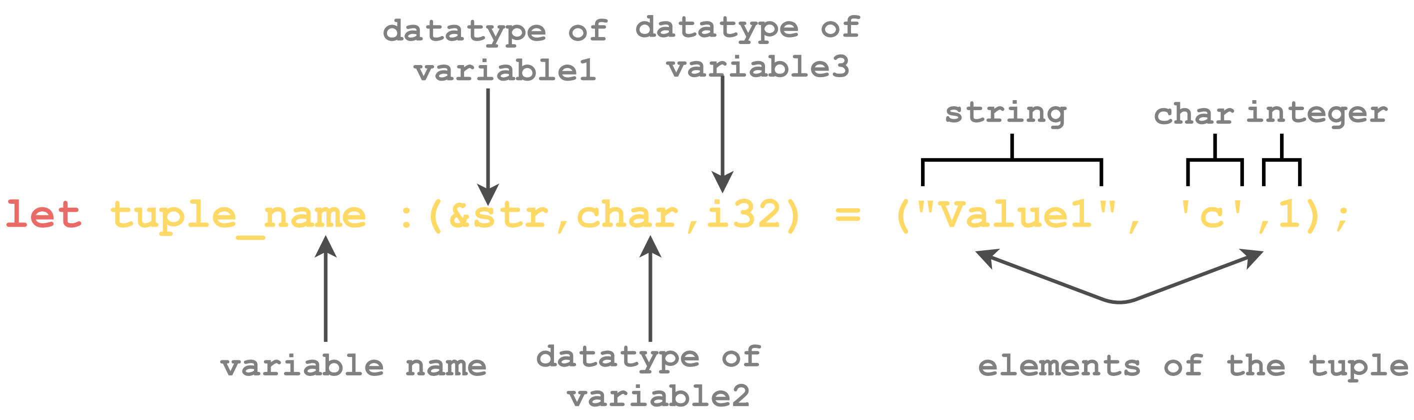
Syntax 1

The syntax below defines a tuple without specifying the type. However, the compiler can infer the type.



Syntax 2

The syntax below defines a tuple by specifying the type.



Example

The following illustration explains the concept:

```
#[allow(unused_variables, unused_mut)]
fn main() {
    //define a tuple
    let person_data = ("Alex", 48, "35kg", "6ft");
    // define a tuple with type annotated
    let person_data : (&str, i32, &str, &str) = ("Alex", 48, "35kg", "6ft");
}
```

Access the Value of the Tuple

- Unlike array which uses [] for accessing an element, the value of the tuple can be accessed using the dot operator (`.`).

```
tuplename.indexvalue
```

- To get the individual values out of a tuple, we can use pattern matching to destructure a tuple value, like this:

```
let person_data = ("Alex", 48, "35kg", "6ft");
let (w, x, y, z) = person_data;
```

```
fn main() {
    //define a tuple
    let person_data = ("Alex", 48, "35kg", "6ft");
    // access value of a tuple
    println!("The value of the tuple at index 0 and index 1 are {} {}",person_data.0,person_data.1

    //define a tuple
    let person_data = ("Alex", 48, "35kg", "6ft");
    // get individual values out of tuple
    let (w ,x, y, z) = person_data;
    //print values
    println!("Name : {}",w);
    println!("Age : {}",x);
    println!("Weight : {}",y);
    println!("Height : {}",z);
}
```

How to Make a Tuple Mutable?

Just like a variable becomes mutable by adding the `mut` keyword after `let`, the same goes for a tuple.

```
fn main() {
    //define a tuple
    let mut person_data = ("Alex", 48, "35kg", "6ft");
    //print the value of tuple
    println!("The value of the tuple at index 0 and index 1 are {} {}", person_data.0, person_data
    //modify the value at index 0
    person_data.0 = "John";
    //print the modified value
    println!("The value of the tuple at index 0 and index 1 are {} {}", person_data.0, person_data
}
```

Print the Tuple

The whole tuple can be traversed using the *debug trait*.

```
fn main() {
    //define a tuple
    let person_data = ("Alex", 48, "35kg", "6ft");
    //print the value of tuple


    println!("Tuple - Person Data : {:?}",person_data);
}
```

Quiz

Test your understanding of tuples in Rust!

Quick Quiz on Tuples!

1 Which of the following statements is not true?

2 What is the output of the following code snippet?

```
let (w ,x, y, z) = ("1","3","2","4");
println!("w : {}",w);
println!("x : {}",x);
println!("y : {}",y);
println!("z : {}",z);
```

Retake Quiz

Now that you have an insight into data types, let’s learn about constant variables in the next lesson.