2f. Tuples

Martin Alfaro
PhD in Economics

INTRODUCTION

We continue our exploration of *collections*, defined as objects storing multiple elements. Having previously focused on arrays, we'll now turn our attention to another form of collection known as *tuples*.

The defining characteristic of tuples is their fixed size and immutability. This implies that, once a tuple is created, its elements cannot be added, removed, or modified. While these restrictions might initially seem limiting, they also bring substantial performance gains when working with small collections.

At this point, we'll only touch on the basics. A more in-depth exploration of tuples will follow in Part II, after we've developed the necessary tools to understand the role of tuples in high-performance scenarios.

> **Warning!**
>
> **Tuples should only be used when the collection comprises a small number of elements**. Large tuples will result in slow computations at best, or directly trigger fatal errors. For large vectors, you should keep relying on vectors.

DEFINITION OF TUPLES

The syntax for accessing the *i*-th element of a tuple `x` is `x[i]`, similar to vectors. Likewise, defining tuples requires enclosing their elements in parentheses `()`, which contrasts with the square brackets `[]` used in vectors.

When tuples comprise more than one element, the use of `()` is optional and its omission is in fact a common practice. On the contrary, single-element tuples have stricter syntax rules: you must use parentheses `()` and a trailing comma `,` after the element. For example, a tuple with the single element `10` is represented as `(10,)`. This notation differentiates a tuple from the expression `(10)`, which would be interpreted simply as the number 10.

```
x = (4,5,6)
x =  4,5,6           #alternative notation
```

```
julia> x
(4, 5, 6)
julia> x[1]
4
```

```
x = (10,)    # not x = (10) (it'd be interpreted as x = 10)
```

```
julia> x
(10,)
julia> x[1]
10
```

TUPLES FOR ASSIGNMENTS

Tuples are particularly useful for simultaneously assigning values to multiple variables. This is achieved by placing **a tuple on the left-hand side of** `=` and **a collection on the right-hand side** (either another tuple or a vector). The following examples demonstrate both options.

```
(x,y) = (4,5)
 x,y  =  4,5       #alternative notation
```

```
julia> x
4
julia> y
5
```

```
(x,y) = [4,5]
 x,y  = [4,5]      #alternative notation
```

```
julia> x
4
julia> y
5
```

As we'll see later, this technique is commonly employed when a function returns multiple values, enabling you to unpack the returned values into individual variables.