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*.

Tuples differ from arrays in several respects. Most notably, tuples can comprise elements with heterogeneous types, without any impact on performance. Moreover, they're characterized by their fixed size and immutability. Both aspects imply that, once a tuple is created, its elements cannot be added, removed, or modified.

Importantly, tuples are only suitable for **storing a small number of elements**. Large tuples will result in slow computations at best, or directly trigger fatal errors. This is why vectors must be opted for when it comes to large collections. While these restrictions might initially seem limiting, they also bring substantial performance gains.

At this stage, we'll limit our discussion to the basic features of tuples, introducing only what's necessary to develop the fundamentals of Julia. A more in-depth exploration will follow in Part II, after we've developed the necessary tools to appreciate their role in high-performance applications.

DEFINITION OF TUPLES

Tuples are defined by enclosing their elements within parentheses `()`, in contrast to the square brackets `[]` used in vectors. When a tuple contains more than one element, the parentheses are optional and often omitted.

Single-element tuples, instead, 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 distinguishes it from the expression `(10)`, which would simply be interpreted as the number `10`.

As with vectors, the syntax for accessing the *i*-th element of a tuple `x` is `x[i]`.

```
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 can be used to assign values to multiple variables at once. This operation is commonly referred to as **unpacking** or **destructuring**. It's accomplished by placing a tuple on the left-hand side of `=` and a collection on the right-hand side, which may be either another tuple or a vector.

```
(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
```

This technique is commonly employed when a function returns multiple values. It provides a convenient syntax to unpack outputs into individual variables.