Tipos estructurados de datos heterogéneos. Tupla

PID_00268294

Autores que han participado colectivamente en esta obra
Lluís Beltrà
Kenneth Capseta
Maria Jesús Marco Galindo
Antonio Ponce

Idea y dirección de la obra
Maria Jesús Marco Galindo

Diseño y edición gráfica
Asunción Muñoz

Material docente de la UOC

Universitat Oberta de Catalunya

Segunda edición: julio 2020

Av. Tibidabo, 39-43, 08035 Barcelona
Realización editorial: FUOC

Contenidos

Tipos estructurados de datos heterogéneos

Tupla

Hasta ahora hemos estudiado los tipos básicos (enteros, reales, caracteres, booleanos, etc.) y hemos visto cómo agrupar datos del mismo tipo (homogéneos) en tablas.

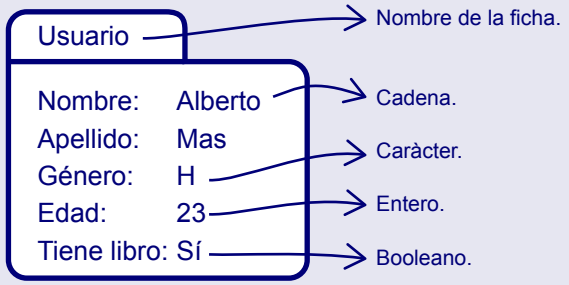
A veces, sin embargo, para representar la realidad nos haría falta poder agrupar elementos de tipos distintos pero que se refieren a un mismo concepto. Así podremos definir tipos de datos que se adapten mejor a los datos del mundo real que el problema debe tratar.

¿Qué es una TUPLA?

Una TUPLA es un tipo estructurado de datos que permite agrupar información diversa (heterogénea).

Ejemplo

Imaginemos que somos los encargados de una antigua biblioteca, la cual dispone de un gran archivador que contiene la ficha de cada usuario. Cada ficha es similar a esta:



Como se puede observar, esta ficha tiene un nombre "Usuario" y en ella se agrupan los datos, que pueden ser de diferente tipo (entero, cadena, etc.).

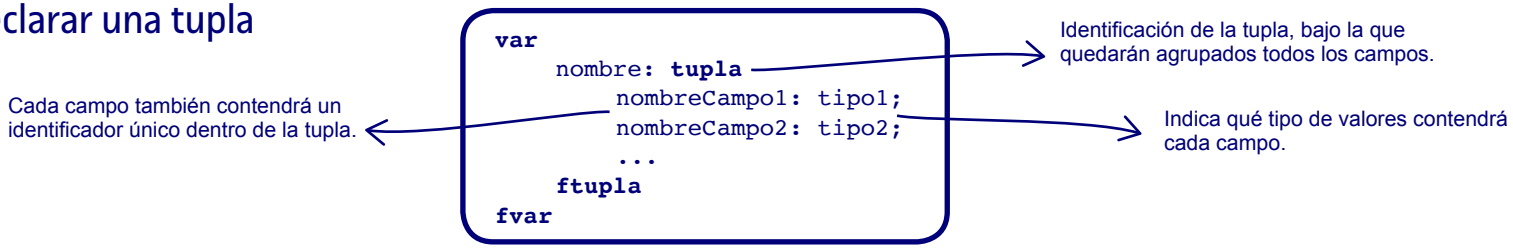
¿Cómo representamos la ficha en notación algorítmica?

Esta ficha la podremos representar con una tupla y los datos a almacenar serán los campos de esta tupla:

```
var
   usuario: tupla
      nombre: cadena;
      apellido: cadena;
      género: carácter;
      edad: entero;
      tiene_libro: booleano;
   ftupla
fvar
```

Hay que tener presente que los campos de una ficha no tienen sentido por sí mismos, no son independientes. Por ejemplo, el campo *nombre* de este ejemplo solo tiene sentido relacionado con un usuario. Además, todos los campos de la ficha (*nombre, apellido*, etc.), se refieren al mismo conceptoo, en este caso al mismo usuario.

Sintaxis para declarar una tupla



Acceso a los campos de una tupla

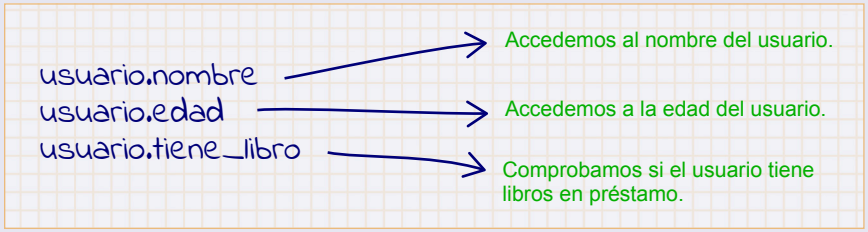
Para hacer referencia a un campo de una tupla se indica con el nombre de la variable de tipo tupla seguido de un punto y, a continuación, el tipo del campo. La **sintaxis** es la siguiente:

```
nombreTupla.nombreCampo
```

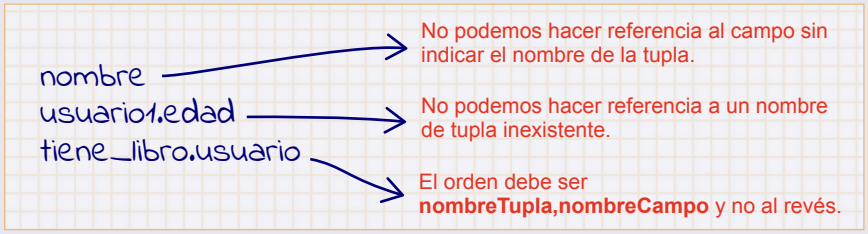
Ejemplo

Siguiendo las declaraciones que hemos visto anteriormente:

Estas referencias **son correctas**



En cambio, estas referencias **NO son correctas**



Concepto clave

Una **tupla** es un tipo de datos estructurado que contiene diferentes datos llamados campos. Cada uno de estos campos tiene un nombre que lo identifica dentro de la tupla, y es de un tipo de datos concreto.

A diferencia de las tablas, las tuplas nos permiten agrupar campos de diferentes tipos de datos (estructura heterogénea) y hacer el algoritmo más legible, más comprensible.

Así, podemos almacenar información relacionada entre sí creando tuplas que se componen de diferentes campos, que pueden ser de tipos diferentes y que, en conjunto, representan un único concepto estructurado cercano a la realidad que queremos representar. De esta forma el algoritmo es más legible y se entiende mejor ya que conseguimos una abstracción de la realidad más ajustada a la que se necesita para resolver el problema.

¿En qué casos nos son útiles las tuplas?

Cuando el algoritmo tiene que tratar datos compuestos (formados por otros datos) y que no se pueden definir con los tipos de datos básicos del lenguaje algorítmico. Así podremos crear variables a medida de las necesidades del algoritmo y más adecuados al concepto u objeto real que representan.

¿Cómo podemos asignar y consultar los campos de una tupla?

Ejemplo

Una vez conocemos la estructura de una tupla y sabemos cómo acceder a sus respectivos campos, vemos cómo podemos asignar y consultar los valores de cada campo de la tupla:

```
usuario.nombre := "Alberto"
usuario.edad := 23;
```

Asignamos el nombre "Albert" al campo *nombre* de la variable *usuario*.

Asignamos el valor 23 al campo *edad* de la variable *usuario*.

La consulta de los valores contenidos en los campos de una tupla también debe hacerse para cada campo por separado, y es necesario escribir la referencia del campo después de la referencia de la tupla.

Evidentemente, los valores que asignamos a los campos de una tupla pueden provenir de un dispositivo de entrada (lectura) y también podemos mostrar por un dispositivo de salida los campos de una tupla (escritura):

```
usuario.apellido:= leerTeclado()
escribirPantalla(usuario.edad) ;
```

Tablas de tuplas

Hasta ahora hemos visto cómo trabajar con una ficha de un usuario de la biblioteca pero, en realidad, en la biblioteca hay un archivo lleno de fichas. De hecho, hay tantas fichas como usuarios registrados tiene la biblioteca.

Aprovechando el concepto de tabla que hemos visto en la anterior unidad, podemos combinar estos dos tipos y dar solución a esta situación.

Ejemplo

Este sería el archivador de la biblioteca:

Imaginemos que en la biblioteca hay 58 usuarios registrados y, por lo tanto, el archivador contiene 58 fichas con sus datos. Lo podemos representar así:

La tabla representa el archivador.

Los 58 registros representan las 58 fichas.

```
var
    usuariosBiblioteca : tabla [58] de tupla
                         nombre : cadena;
                         apellido : cadena;
                         género : carácter;
                         edad : entero;
                         tiene_llibro : booleano;
                         ftupla;
fvar
```

Cada ficha es de tipo de usuario, es decir, contiene datos con la estructura de la tupla usuario y representa a un usuario de la biblioteca.

Por ejemplo, para asignar un nombre al tercer usuario simplemente accederemos al campo "nombre" de la tercera ficha del archivador:

```
usuariosBiblioteca[3].nombre := "Enrique"
```

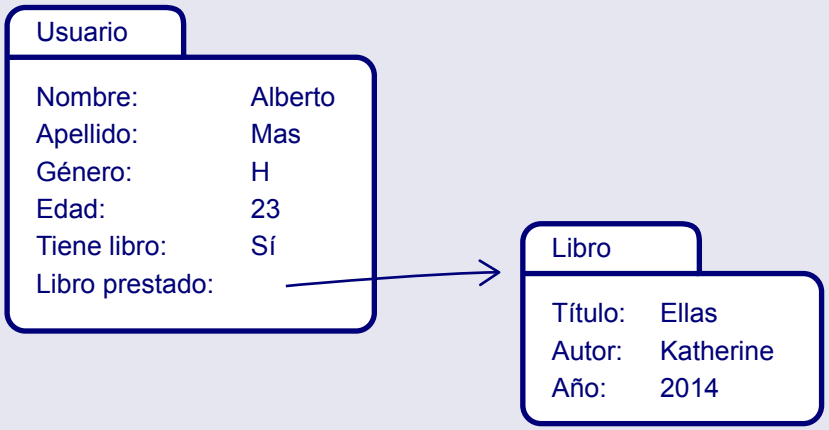
Tuplas de tuplas

Como ya se han mencionado, las tuplas nos permitirán **agrupar diferentes tipos de datos**. Esto significa que podemos tener un campo de la tupla que sea de tipo tupla.

Ejemplo

Siguiendo el ejemplo del archivo de usuario de la biblioteca, vemos que el campo *tiene_libro* es un booleano que nos indica si el usuario tiene un libro en préstamo, pero no ofrece ninguna información sobre el libro.

Si queremos tener los datos del libro, se puede añadir un campo de tipo tupla:



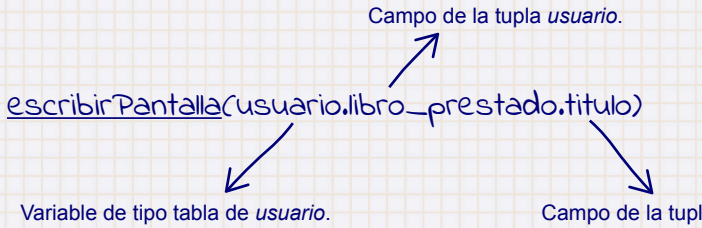
Vemos que se ha definido la tupla *Usuario* y el campo *Libro prestado* lo hemos definido como una tupla de tipo “Libro”.

Representamos en notación algorítmica (NA) esta nueva estructura:

```
var
   usuariosBiblioteca : tabla [58] de tupla
                                       nombre : cadena;
                                       apellido : cadena;
                                       género : caracter;
                                       edad : entero;
                                       tiene_librovbooleano;
                                       libro_prestado : tupla
                                                titulo : cadena;
                                                autor : cadena;
                                                año : entero;
                                       ftupla;
   ftupla;
fvar
```

Con esta nueva estructura podemos conocer los datos de un usuario y podemos saber también los datos del libro que tiene en préstamo.

El modo de operar con los datos será el mismo que en el caso de las tuplas simples. Por ejemplo, la acción que nos muestra el título del libro que tiene un usuario en préstamo, podría ser:



¿Cómo representamos las tuplas en JavaScript?

En lenguaje JavaScript (JS) las tuplas se representan como objetos. Igual que un **Array**, un **Object** es una construcción que agrupa un conjunto de datos diversos. De cada **Object** podemos definir propiedades y métodos que ofrecen mucha potencialidad de codificación. En su versión más simple, solo indicando las propiedades (los campos) ya podemos utilizar los objetos para representar tuplas.

Sintaxis para declarar una tupla en JS

```
var nombreTupla = {
nombreCampo1 : valor1,
nombreCampo2 : valor2,
...
nombreCamp3 : valorN
} ;
```

Ejemplos

```
1  //Declaramos los objetos usuario
2  var usuario1 = {
3  nombre: "Pedro",
4  apellido: "Romero",
5  genero: "H",
6  edad: 35,
7  tiene_libro: undefined
8  };
9  var usuario2 = {
10 nombre: "María",
11 apellido: "Villanueva",
12 genero: "D",
13 edad: 18,
14 tiene_libro: undefined
15 };
16 var usuario3 = {
17 nombre: "Alberto",
18 apellido: "Mas",
19 genero: "H",
20 edad: 23,
21 tiene_libro:undefined
22 };
23 //Declaramos los libros de la biblioteca
24 var libro1 = {
25 titulo : "Ellas",
26 autor : "Katherine",
27 any : 2014
28 };
29 var libro2 = {
30 titulo : "La Ilíada",
31 autor : "Homero",
32 any : 2003
33 };
34
```

```
1  //Incluimos nuestros usuarios en la biblioteca
2  var usuariosBiblioteca = [usuario1, usuario2, usuario3];
3
```

A diferencia de la definición de tupla que hemos visto en notación algorítmica, para un objeto en JavaScript no es necesario que declaremos el tipo, que ya quedará claro en el momento en que lo utilicemos por primera vez. Este es un comportamiento que algorítmicamente no se recomienda ya que puede llevar a error o confusión, pero en el caso de este lenguaje, no se puede realizar de otro modo.

En posteriores asignaturas ya veréis cómo se pueden definir objetos con propiedades (campos) y métodos (funcionalidades) asociados, y, por lo tanto, cómo programar con este lenguaje con el paradigma de la orientación a objetos (OO). En la guia Mozilla de JS también lo encontraréis detallado.

Acceso a los campos de una tupla en JS

Accederemos a los valores de cada campo (propiedad) haciendo referencia a su nombre después del nombre de la tupla y un punto, como hacemos con la notación algorítmica:

```
nombreObject.nombreCampo
```

Recordad que el lenguaje JavaScript tiene muchas más opciones que la notación algorítmica. Aquí adoptaremos la más sencilla.

Encontraréis otras opciones en la guia de JavaScript de Mozilla.

Ejemplos

```
1 //Alberto ha cogido prestado un libro
2 usuariosBiblioteca[2].tiene_libro = libro1;
3
```

```
1 //Comprobamos el libro que ha cogido prestado
2 console.log(usuariosBiblioteca[2].tiene_libro);
3
```

¿Qué pasaría si...

...intentamos acceder a un nombre de campo que no existe?

Ejemplo

Simplemente nos devolvería como valor la palabra clave de JS “undefined”.

```
1 var tupla = {
2 campo : "prueba",
3 };
4 console.log(typeof(tupla.campo)); //nos devolverá "string"
5 console.log(typeof(tupla.campoInexistente)); //nos devolverá "undefined"
6
```

Fijaos que el comportamiento de los objetos en JS permite representar las tuplas tal como las hemos definido algorítmicamente, pero también otras posibilidades que de momento no utilizaremos.

Podéis probar estos códigos en la web de PythonTutor.