Tipus estructurat de dades heterogènies. Tupla

PID_00268287

Autors que han participat col·lectivament en aquesta obra
Lluís Beltrà
Kenneth Capseta
Maria Jesús Marco Galindo
Antonio Ponce

Idea i direcció de l'obra
Maria Jesús Marco Galindo

Disseny i edició gráfica
Asunción Muñoz

Material docent de la UOC

Universitat Oberta de Catalunya

Segona edició: juliol 2020

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

Continguts

Tipus estructurat de dades heterogènies

Tupla

- Sintaxi per declarar una tupla
- Què és una TUPLA?
- Accés als camps d'una tupla
- Concepte clau
- Com podem assignar i consultar els camps d'una tupla?
- Taules de tuples
- Tuples de tuples

Com representem les tuples en JavaScript?

- Sintaxi per declarar una tupla en JS
- Accés als camps d'una tupla en JS
- Què passaria si …

Tipus estructurat de dades heterogènies

Tupla

Fins ara hem estudiat els tipus bàsics (enters, reals, caràcters, booleans, etc.) i hem vist com agrupar dades del mateix tipus (homogènies) en taules.

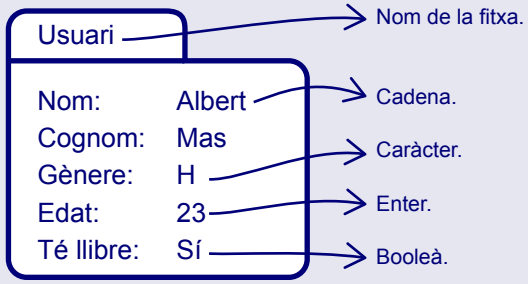
A vegades, però, per representar la realitat ens caldria poder agrupar elements de tipus diferents però que es refereixen a un mateix concepte. Així podrem definir tipus de dades que s'adaptin millor a les dades del món real que el problema ha de tractar.

Què és una TUPLA?

Una TUPLA és un tipus estructurat de dades que permet agrupar informació diversa (hetereogènia).

Exemple

Imaginem que som els encarregats d'una antiga biblioteca, la qual disposa d'un gran arxivador que conté la fitxa de cada usuari. Cada fitxa és similar a aquesta:



Com es pot observar, aquesta fitxa té un nom "Usuari" i en ella s'hi agrupen dades, que poden ser de diferent tipus (enter, cadena, etc.).

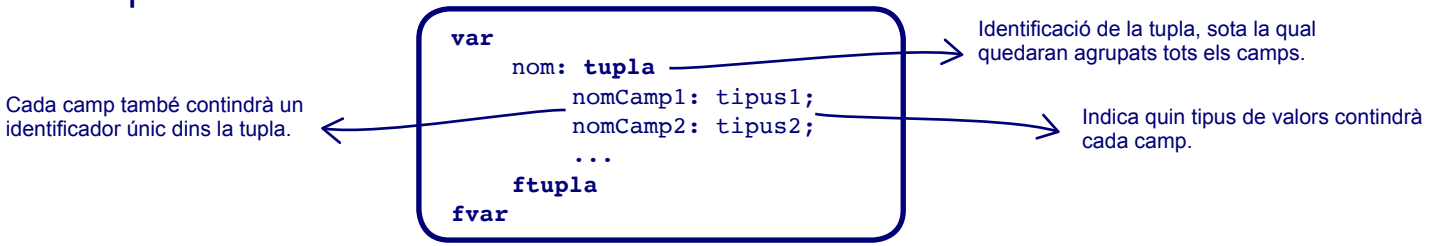
Com representem la fitxa en notació algorísmica?

Aquesta fitxa la podrem representar amb una tupla i les dades a emmagatzemar seran els camps d'aquesta tupla:

```
var
    usuari: tupla
        nom: cadena;
        cognom: cadena;
        gènere: caràcter;
        edat: enter;
        té_llibre: booleà;
    ftupla
fvar
```

Cal tenir present que els camps d'una fitxa no tenen sentit per si sols, no són independents. Per exemple, el camp *nom* d'aquest exemple només tè sentit relacionat amb un usuari. A més, tots els camps de la fitxa (*nom*, *cognom*, etc.), es refereixen al mateix concepte, en aquest cas al mateix usuari.

Sintaxi per declarar una tupla



Accés als camps d'una tupla

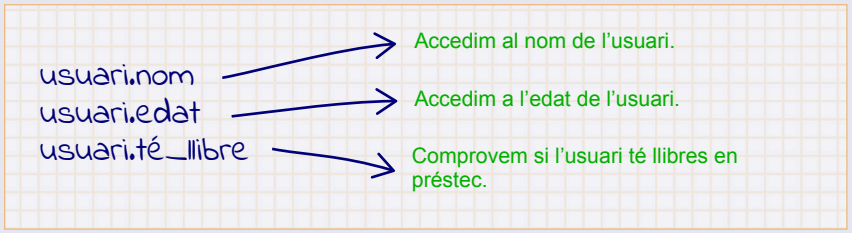
Per fer referència a un camp d'una tupla cal indicar el nom de la variable de tipus tupla seguit d'un punt i del nom del camp. La **sintaxi** és la següent:

```
nomTupla.nomCamp
```

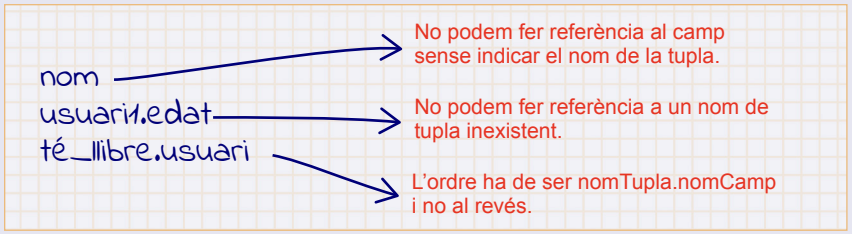
Exemple

Seguint les declaracions que hem vist anteriorment:

Aquestes referències **són correctes**



En canvi, aquestes referències **NO són correctes**



Concepte clau

Una **tupla** és un tipus de dades estructurat que agrupa diferents dades anomenades camps. Cadascun d'aquests subobjectes té un nom que l'identifica dins la tupla, i és d'un tipus de dades concret.

A diferència de les taules, les tuples ens permeten agrupar camps de diferents tipus de dades (estructura heterogènia).

Així, podem emmagatzemar informació relacionada entre si tot creant tuples que es componen de diversos camps, que poden ser de tipus diferents i que, en conjunt, representen un únic concepte estructurat proper a la realitat que volem representar. D'aquesta manera l'algorisme és més llegible i mes entenedor ja que aconseguim una abstracció de la realitat més ajustada al que es necessita per a resoldre el problema.

Ens quins casos ens són útils les tuples?

Quan l'algorisme ha de tractar dades compostes (formades per altres dades) i que no es poden definir amb els tipus de dades bàsics del llenguatge algorísmic. Així podrem crear variables a la mida de les necessitats de l'algorisme i més adequats al concepte o objecte real que representen.

Com podem assignar i consultar els camps d'una tupla?

Exemple

Una vegada coneixem l'estructura d'una tupla i sabem com accedir als seus respectius camps, veiem com podem assignar i consultar els valors de cada camp de la tupla:

```
usuari.nom := "Albert"
usuari.edat := 23;
```

Assignem el nom "Albert" al camp *nom* de la variable *usuari*.

Assignem el valor 23 al camp *edat* de la variable *usuari2*.

La consulta dels valors continguts en els camps d'una tupla també s'ha de fer per a cada camp per separat.

Evidentment, els valors que assignem als camps d'una tupla poden provenir d'un dispositiu d'entrada (lectura) i també podem mostrar per un dispositiu de sortida els camps d'una tupla (escriptura):

```
usuari.cognom:= llegirTeclat()
escriurePantalla(usuari.edat) ;
```

Taules de tuples

Fins ara hem vist com treballar amb una fitxa d'un usuari de la biblioteca però, realment, a la biblioteca hi ha un arxivador ple de fitxes. De fet hi ha tantes fitxes com usuaris registrats té la biblioteca.

Aprofitant el concepte de taula que hem vist a l'anterior unitat, podem combinar aquests dos tipus i donar solució a aquesta situació.

Exemple

Aquest seria l'arxivador de la biblioteca:

Imaginem que a la biblioteca hi ha 58 usuaris registrats i, per tant, l'arxivador conté 58 fitxes amb les seves dades. Ho podem representar algorísmicament així:

La taula representa l'arxivador.

Els 58 registres representen les 58 fitxes.

```
var
    usuarisBiblioteca : taula [58] de tupla
                            nom : cadena;
                            cognom : cadena;
                            gènere : caràcter;
                            edat : enter;
                            té_llibre : booleà;
                        ftupla;
fvar
```

Cada fitxa és de tipus usuari, és a dir, conté dades amb l'estructura de la tupla usuari i representa a un usuari de la biblioteca.

Per exemple, per assignar un nom al tercer usuari simplement accedirem al camp *nom* de la tercera fitxa de l'arxivador:

```
usuarisBiblioteca[3].nom := "Enric"
```

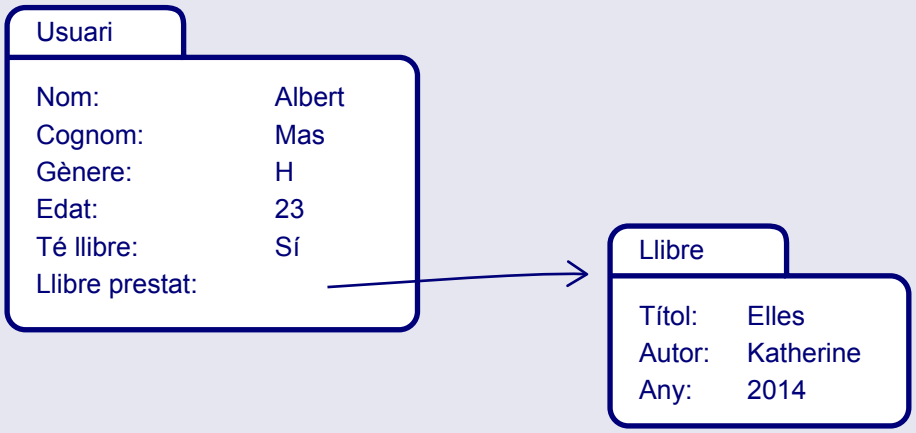
Tuples de tuples

Com ja s'ha comentat, les tuples ens permetran **agrupar diferents tipus** de dades. Això significa que podem tenir un camp de la tupla que sigui de tipus tupla.

Exemple

Seguint l'exemple de la fitxa d'usuari de la biblioteca, veiem que el camp *té_llibre* és un booleà que ens indica si l'usuari té un llibre en préstec, però no dóna cap informació sobre el llibre prestat.

Si volem tenir les dades del llibre, es pot afegir un camp de tipus tupla:



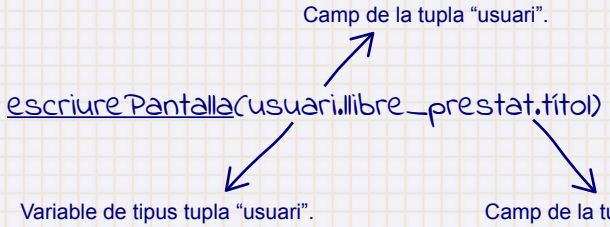
Veiem que s'ha definit la tupla *Usuari* i el camp *Llibre prestat* l'hem definit com una tupla de tipus *Llibre*.

Representem en notació algorísmica (NA) aquesta nova estructura:

```
var
   usuarisBiblioteca : taula [58] de tupla
                                        nom : cadena;
                                        cognom : cadena;
                                        gènere : carècter;
                                        edat : enter;
                                        té_llibre : booleà;
                                        llibre_prestat : tupla
                                                 títol : cadena;
                                                 autor : cadena;
                                                 any : enter;
                                        ftupla;
   ftupla;
fvar
```

Amb aquesta nova estructura podem conèixer les dades d'un usuari i podem saber també les dades del llibre que té en préstec.

La manera d'operar amb les dades serà la mateixa que en el cas de les tuples simples. Per exemple, l'acció que ens mostra el títol del llibre que té un usuari en préstec, podria ser:



Com representem les tuples en JavaScript?

En llenguatge JavaScript (JS) les tuples es representen com a objectes. Igual que un **Array**, un **Object** és una construcció que agrupa un conjunt de dades diverses. De cada **Object** podem definir propietats i mètodes que donen molta potencialitat de codificació. En la seva versió més simple, només indicant les propietats (els camps) ja podem utilitzar els objectes per representar tuples.

Sintaxi per declarar una tupla en JS

```
var nomTupla = {
nomCamp1 : valor1,
nomCamp2 : valor2,
...
nomCamp3 : valorN
} ;
```

Exemples

```
//Declarem els objectes usuari:
var usuari1 = {
nom: "Pere",
cognom: "Llopis",
genere: "H",
edat: 35,
te_llibre: undefined
};
var usuari2 = {
nom: "Maria",
cognom: "Vilanova",
genere: "D",
edat: 18,
te_llibre: undefined
};
var usuari3 = {
nom: "Albert",
cognom: "Mas",
genere: "H",
edat: 23,
te_llibre:undefined
};
//Declarem els llibres de la biblioteca
var llibre1 = {
titol : "Elles",
autor : "Katherine",
any : 2014
};
var llibre2 = {
titol : "La Ilíada",
autor : "Homer",
any : 2003
};
```

```
//Incloem els nostres usuaris a la biblioteca
var usuarisBiblioteca = [usuari1, usuari2, usuari3];
```

A diferència de la definició de tupla que hem vist en notació algorísmica, quan definim un objecte de JavaScript no cal que declarem el tipus dels camps (que en el cas dels objectes es diuen propietats). Ja s'autoassignarà en el moment que l'utilitzem per primer cop. Aquest és un comportament que algorísmicament no es recomana perquè pot portar a error o confusió, però en el cas del llenguatge JavaScript, no es pot fer d'altra manera.

En posteriors assignatures ja veureu com es poden definir objectes amb propietats (camps) i mètodes (funcionalitats) associats, i, per tant, com programar amb aquest llenguatge amb el paradigma de l'orientació a objectes (OO). A la guia Mozilla de JS també ho trobareu detallat.

Accés als camps d'una tupla en JS

Accedirem als valors de cada camp (propietat) fent referència al seu nom després del nom de la tupla i un punt, tal com fem amb la notació algorísmica:

```
nombreObject.nomCamp
```

Recordeu que el llenguatge JavaScript té moltes més opcions que la notació algorísmica. Aquí adoptarem la més senzilla.

Trobareu altres opcions a la guia de JavaScript de Mozilla.

Exemples

```
1 //L'Albert ha llogat un llibre:
2 usuarisBiblioteca[2].te_llibre = llibre1
3
```

```
1 //Comprovem el llibre que ha llogat:
2 console.log(usuarisBiblioteca[2].te_llibre);
3
```

Què passaria si ...

...intentem accedir a un nom de camp que no existeix?

Exemple

Simplement ens retornaria com a valor la paraula clau de JS "undefined".

```
1 var tupla = {
2 camp : "prova",
3 };
4 console.log(typeof(tupla.camp)); //retornarà "string"
5 console.log(typeof(tupla.campInexistent)); // retornarà "undefined"
6
```

Fixeu-vos que el comportament dels objectes en JS permet representar les tuples tal i com les hem definit algorísmicament, però també ofereix altres possibilitats que, de moment, no utilitzarem.

Podeu provar aquests codis a la web de PythonTutor.