Fondamenti di Informatica (Elettronici)

THINK JULIA – Capitolo 12

25 novembre 2020

12. Tuple[1]

[1] Tratto da https://benlauwens.github.io/ThinkJulia.jl/latest/book.html, disponibile sotto Licenza 'Creative Commons Attribution-NonCommercial 3.0 Unported'.

Questo capitolo presenta un altro tipo predefinito, la tupla, e poi mostra come gli array, i dizionari e le tuple lavorano insieme. Presento anche una funzionalità utile per array di argomenti a lunghezza variabile, gli operatori di gather (raccolta) e scatter (dispersione).

Section 1

Le tuple sono immutabili

I valori possono essere di qualsiasi tipo

Una `tupla` è una sequenza di valori. I valori possono essere di qualsiasi tipo e sono indicizzati da numeri interi, quindi da questo punto di vista le tuple sono molto simili agli array.

La differenza importante è che le tuple sono immutabili e che ogni elemento può avere il proprio tipo.

Sintatticamente, una `tupla` è un elenco di valori separati da virgole:

```
julia> t = 'a', 'b', 'c', 'd', 'e'
('a', 'b', 'c', 'd', 'e')
```

Sebbene non sia necessario, è comune racchiudere le tuple tra parentesi:

```
julia> t = ('a', 'b', 'c', 'd', 'e')
('a', 'b', 'c', 'd', 'e')
```

tuple con un singolo elemento

Per creare una tupla con un singolo elemento, devi includere una virgola finale:

```
julia> t1 = ('a',)
('a',)
julia> typeof(t1)
Tuple{Char}
```

WARNING

Un valore tra parentesi senza virgola non è una tupla:

```
julia> t2 = ('a')
'a': ASCII/Unicode U+0061 (category Ll: Letter, lowercase)
julia> typeof(t2)
Char
```

Funzione costruttore di `tuple`

Un altro modo per creare una tupla è la funzione predefinita `tuple`. Senza argomenti, crea una tupla vuota:

```
julia> tuple()
()
```

Se vengono forniti più argomenti, il risultato è una tupla con gli argomenti dati:

```
julia> t3 = tuple(1, 'a', pi)
(1, 'a', (\pi TAB) = 3.1415926535897...)
```

Poiché `tuple` è il nome di una funzione predefinita, dovresti evitare di usarlo come nome di variabile.

Operatore `bracket [ ]`

La maggior parte degli operatori di array lavora anche su tuple.

L'operatore `bracket` indicizza un elemento:

```
julia> t = ('a', 'b', 'c', 'd', 'e');
julia> t[1]
'a': ASCII/Unicode U+0061 (category Ll: Letter, lowercase)
```

L'operatore `slice` seleziona una serie di elementi:

```
julia> t[2:4]
('b', 'c', 'd')
```

Operatori relazionali

Ma se si prova a modificare uno degli elementi della tupla, si ottiene un errore:

```
julia> t[1] = 'A'
ERROR: MethodError: no method matching setindex!(::NTuple{5,Char}, ::Char,
```

Poiché le tuple sono immutabili, non puoi modificare gli elementi.

Gli operatori relazionali lavorano con tuple e altre sequenze; Julia inizia confrontando il primo elemento di ogni sequenza.

Se sono uguali, passa agli elementi successivi, e cos via, finché non trova elementi che differiscono.

Gli elementi successivi non vengono considerati (anche se molto numerosi).

```
julia> (0, 1, 2) < (0, 3, 4)
true
julia> (0, 1, 2000000) < (0, 3, 4)
true
```

Section 2

Assegnazione di tuple

Scambiare due variabili (swapping)

Spesso è utile scambiare i valori di due variabili.

Con le assegnazioni convenzionali, devi usare una variabile temporanea. Ad esempio, per scambiare `a` e `b`:

```
temp = a
a = b
b = temp
```

Questa soluzione è ingombrante; l'assegnazione di tuple è più elegante:

```
a, b = b, a
```

Il lato sinistro è una `tupla` di variabili; il lato destro è una `tupla` di espressioni.

Ogni valore viene assegnato alla rispettiva variabile. Tutte le espressioni sul lato destro vengono valutate prima di qualsiasi assegnazione.

Divisione di tuple

Il numero di variabili a sinistra deve essere inferiore al numero di valori a destra:

```
julia> (a, b) = (1, 2, 3)
(1, 2, 3)
julia> a, b, c = 1, 2
ERROR: BoundsError: attempt to access (1, 2)
    at index [3]
```

Più in generale, il lato destro può essere qualsiasi tipo di sequenza (`string`,`array` o `tuple`). Ad esempio, per dividere un indirizzo e-mail in un nome utente e un dominio, potresti scrivere:

```
julia> addr = "julius.caesar@rome"
`julius.caesar@rome`
julia> uname, domain = split(addr, '@');
```

Il valore restituito da `split` è un array con due elementi; il primo elemento assegnato a `uname`, il secondo a `dominio`.

```
julia> uname
`julius.caesar`
julia> domain
`rome`
```

Section 3

Tuple come valori di ritorno

Returning multiple values

A rigor di termini, una funzione può restituire solo un valore, ma se il valore è una tupla, l'effetto è lo stesso della restituzione di più valori.

Ad esempio, se si desidera dividere due numeri interi e calcolare il quoziente e il resto, non è efficiente calcolare `x ÷ y` e quindi `x % y`. E' meglio calcolarli entrambi contemporaneamente.

La funzione predefinita `divrem` accetta due argomenti e restituisce una tupla di due valori, il quoziente * e ilresto**.

Puoi memorizzare il risultato come una tupla:

```
julia> t = divrem(7, 3)
(2, 1)
```

Assegnazione di tuple

Oppure usa l'assegnazione di tupla per memorizzare gli elementi separatamente:

```
julia> q, r = divrem(7, 3);
julia> @show q r;
q = 2
r = 1
```

Ecco un esempio di una funzione che restituisce una tuple:

```
function minmax(t)
    minimum(t), maximum(t)
end
```

maximum e minimum sono funzioni predefinite che trovano gli elementi più grandi e più piccoli di una sequenza.

minmax calcola entrambi e restituisce una tupla di due valori. La funzione built-in extrema è più efficiente.

Section 4

Tuple di argomenti a lunghezza variabile

Numero variabile di argomenti

Le funzioni possono accettare un numero variabile di argomenti.

Un nome di parametro che termina con `...` raccoglie argomenti in una `tupla`. Ad esempio, `printall` accetta un numero qualsiasi di argomenti e li stampa:

```
function printall(args...)
    println(args)
end
```

Il parametro `gather` può avere qualsiasi nome si voglia, `args` è convenzionale. Ecco come "funziona" la funzione:

```
julia> printall(1, 2.0, '3')
(1, 2.0, '3')
```

gather e scatter

Il complemento di `gather` è `scatter`. Se si dispone di una sequenza di valori e si desidera passarla a una funzione come più argomenti, è possibile utilizzare l'operatore ...

Ad esempio, `divrem` accetta esattamente due argomenti; non funziona con una tupla:

```
julia> t = (7, 3);
julia> divrem(t)
ERROR: MethodError: no method matching divrem(::Tuple{Int64,Int64})
```

Ma se spargi la tupla con un `...`, funziona:

```
julia> divrem(t...)
(2, 1)
```

Tuple di argomenti di lunghezza variabile

Molte delle funzioni predefinite utilizzano tuple di argomenti di lunghezza variabile. Ad esempio, `max` e `min` possono accettare qualsiasi numero di argomenti:

```
julia> max(1, 2, 3)
3
```

Ma `sum` non lo fa:

```
julia> sum(1, 2, 3)
ERROR: MethodError: no method matching sum(::Int64, ::Int64, 
```

Nel mondo Julia, la raccolta è spesso chiamata `slurp` e la dispersione `splat`.

Esercizio 12-1

Scrivi una funzione chiamata `sumall` che accetta un numero qualsiasi di argomenti e restituisce la loro `sum`.

Section 5

Array e tuple

Funzione zipper

zip è una funzione predefinita che accetta due o più sequenze e restituisce una raccolta di tuple dove ogni tupla contiene un elemento da ogni sequenza.

Il nome della funzione si riferisce a una cerniera, che unisce e intercala due file di denti.

Questo esempio zippa una stringa e un array:

```
julia> s = "abc";
julia> t = [1, 2, 3];
julia> zip(s, t)
Base.Iterators.Zip{Tuple{String,Array{Int64,1}}}(("abc", [1, 2
```

L'uso più comune di `zip` è in un ciclo `for`

Il risultato è un oggetto `zip` che sa come iterare attraverso le coppie. L'uso più comune di `zip` è in un ciclo `for`:

```
julia>
for pair in zip(s, t)
    println(pair)
end
('a', 1)
('b', 2)
('c', 3)
```

Un oggetto `zip` è una sorta di iteratore, che sarebbe qualsiasi oggetto che itera attraverso una sequenza.

Gli iteratori sono simili agli array in qualche modo, ma a differenza degli array, non è possibile utilizzare un indice per selezionare un elemento da un iteratore.

Oggetto zip per creare un array

Se vuoi usare gli operatori e le funzioni di array, puoi usare un oggetto `zip` per creare un array:

```
julia> collect(zip(s, t))
3-element Array{Tuple{Char,Int64},1}:
    ('a', 1)
    ('b', 2)
    ('c', 3)
```

Il risultato è un array di`tuple`; in questo esempio, ogni `tupla` contiene un carattere dalla stringa e l'elemento corrispondente dall'array.

Se le sequenze non hanno la stessa lunghezza (`length`), il risultato ha la `length` di quella più corta.

```
julia> collect(zip("Anne", "Elk"))
3-element Array{Tuple{Char,Char},1}:
    ('A', 'E')
    ('n', 'l')
    ('n', 'k')
```

Assegnazione di `tuple` in un ciclo `for`

Puoi usare l'assegnazione di `tuple` in un ciclo `for` per attraversare un array di tuple:

```
julia> t = [('a', 1), ('b', 2), ('c', 3)];
julia> for (letter, number) in t
    println(number, ` `, letter)
end
1 a
2 b
3 c
```

Ogni volta nel ciclo, Julia seleziona la tupla successiva nell'array e assegna gli elementi a `letter` e `number`.

Le parentesi intorno a `(letter, number)` sono obbligatorie.

Idioma utile

Se combini le assegnazioni, `zip`,`for` e `tuple`, ottieni un idioma utile per attraversare due (o più) sequenze allo stesso tempo.

Ad esempio, `hasmatch` accetta due sequenze, `t1` e `t2`, e restituisce `true` se esiste un indice `i` tale che `t1[i] == t2[i]`:

```
function hasmatch(t1, t2)
    for (x, y) in zip(t1, t2)
        if x == y
            return true
        end
    end
    false
end
```

Oggetto enumerate

Se hai bisogno di attraversare gli elementi di una sequenza e dei loro indici, puoi usare la funzione predefinita `enumerate`:

```
julia> for (index, element) in enumerate("abc")
    println(index, ` `, element)
end
1 a
2 b
3 c
```

Il risultato di `enumerate` un oggetto `enumerate`, che itera una sequenza di coppie; ogni coppia contiene un indice (a partire da 1) e un elemento della sequenza data.

Section 6

Dizionari e tuple

Dizionari come iteratori

I dizionari possono essere usati come iteratori che iterano le coppie valore-chiave. Puoi usarlo in un ciclo `for` come questo:

```
julia> d = Dict('a'=>1, 'b'=>2, 'c'=>3);
julia> for (key, value) in d
    println(key, " ", value)
end
a1
c3
b2
```

Come ci si dovrebbe aspettare da un dizionario, gli elementi non sono in ordine.

##Array Matrice di tuple per inizializzare un nuovo dizionario

Andando nella direzione opposta, puoi utilizzare un array di tuple per inizializzare un nuovo dizionario:

Tuple come chiavi nei dizionari

E' comune usare le tuple come chiavi nei dizionari.

Ad esempio, un elenco telefonico potrebbe mappare da coppie `cognome` e `nome` a numeri di telefono.

Supponendo di aver definito `last`, `first` e `number`, potremmo scrivere:

```
directory[last, first] = number
```

L'espressione tra parentesi è una `tupla`. Potremmo usare l'assegnazione di tupla per attraversare questo dizionario.

```
for ((last, first), number) in directory
    println(first, ` `, last, ` `, number)
end
```

Questo ciclo attraversa le coppie chiave-valore nella `directory`, che sono tuple.

Assegna gli elementi della chiave in ogni tupla a `last` e `first`, e il valore a `number`, quindi stampa il `nome` e il numero di telefono corrispondente.

Rappresenta le tuple in un diagramma a stati 1/2

Esistono due modi per rappresentare le tuple in un diagramma di stato.

La versione più dettagliata mostra gli indici e gli elementi proprio come appaiono in un array. Ad esempio, la tupla `(Cleese, John)` apparirà come nel diagramma di stato:

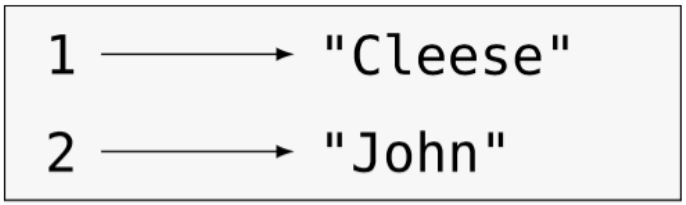

Figure 18. State diagram

Ma in un diagramma pi grande potresti voler omettere i dettagli. Ad esempio, un diagramma della rubrica telefonica potrebbe apparire come nel diagramma di stato qui sotto.

Rappresenta le tuple in un diagramma a stati 2/2

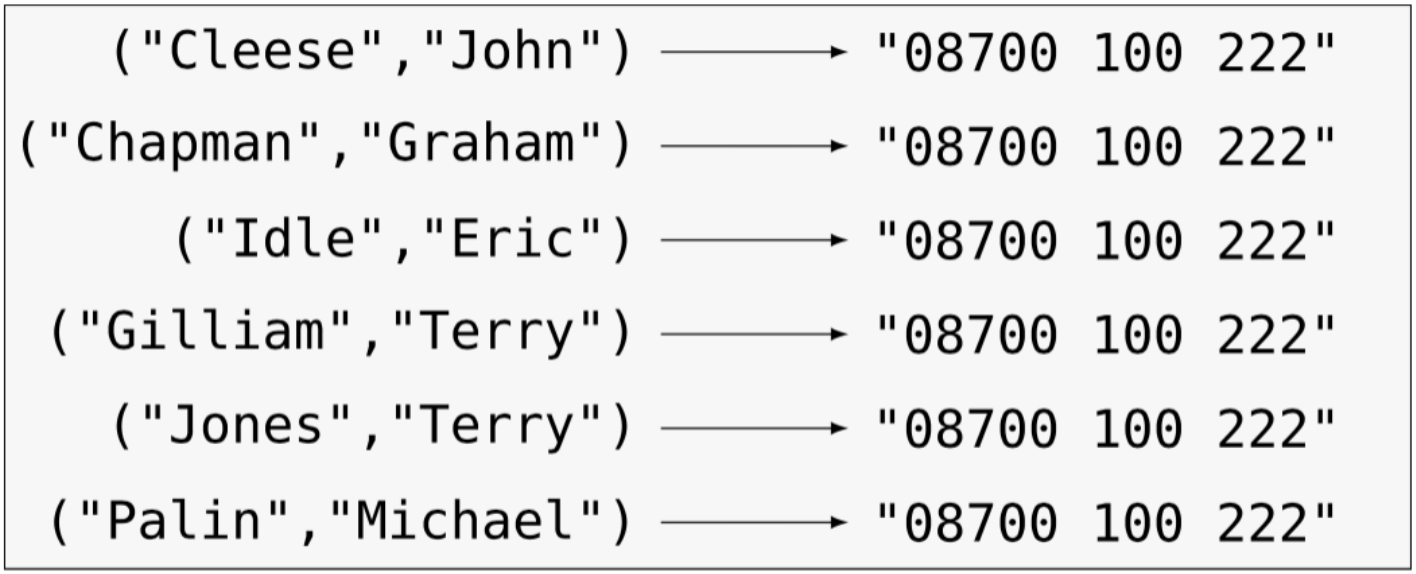

Figure 19. State diagram

Qui le tuple vengono mostrate usando la sintassi di Julia come scorciatoia grafica. Il "numero" di telefono nel diagramma la riga dei reclami per la BBC, quindi per favore non chiamatelo ;-).

Section 7

Sequenze di sequenze

Cosa scegliere?

Mi sono concentrato sugli array di tuple, ma quasi tutti gli esempi in questo capitolo funzionano anche con array di array, tuple di tuple e tuple di array.

Per evitare di enumerare le possibili combinazioni, a volte è più facile parlare di sequenze di sequenze.

In molti contesti, i diversi tipi di sequenze (stringhe, array e tuple) possono essere usati in modo intercambiabile.

Quindi come dovresti sceglierne uno rispetto agli altri?

- Per iniziare con l'ovvio, le stringhe sono più limitate rispetto ad altre sequenze perché gli elementi devono essere caratteri.

Cosa scegliere?

Mi sono concentrato sugli array di tuple, ma quasi tutti gli esempi in questo capitolo funzionano anche con array di array, tuple di tuple e tuple di array.

Per evitare di enumerare le possibili combinazioni, a volte è più facile parlare di sequenze di sequenze.

In molti contesti, i diversi tipi di sequenze (stringhe, array e tuple) possono essere usati in modo intercambiabile.

Quindi come dovresti sceglierne uno rispetto agli altri?

- Per iniziare con l'ovvio, le stringhe sono più limitate rispetto ad altre sequenze perché gli elementi devono essere caratteri.
- Sono anche immutabili.

Cosa scegliere?

Mi sono concentrato sugli array di tuple, ma quasi tutti gli esempi in questo capitolo funzionano anche con array di array, tuple di tuple e tuple di array.

Per evitare di enumerare le possibili combinazioni, a volte è più facile parlare di sequenze di sequenze.

In molti contesti, i diversi tipi di sequenze (stringhe, array e tuple) possono essere usati in modo intercambiabile.

Quindi come dovresti sceglierne uno rispetto agli altri?

- Per iniziare con l'ovvio, le stringhe sono più limitate rispetto ad altre sequenze perché gli elementi devono essere caratteri.
- Sono anche immutabili.
- Se hai bisogno della possibilità di cambiare i caratteri in una stringa (invece di creare una nuova stringa), potresti voler usare invece un array di caratteri.

Altri criteri

Gli array sono più comuni delle tuple, principalmente perché sono mutabili. Ma ci sono alcuni casi in cui potresti preferire le tuple:

- In alcuni contesti, come un'istruzione `return`, è sintatticamente più semplice creare una tupla che un array.

Altri criteri

Gli array sono più comuni delle tuple, principalmente perché sono mutabili. Ma ci sono alcuni casi in cui potresti preferire le tuple:

- In alcuni contesti, come un'istruzione `return`, è sintatticamente più semplice creare una tupla che un array.
- Se stai passando una sequenza come argomento a una funzione, l'uso delle tuple riduce il potenziale di comportamento imprevisto dovuto all'aliasing.

Altri criteri

Gli array sono più comuni delle tuple, principalmente perché sono mutabili. Ma ci sono alcuni casi in cui potresti preferire le tuple:

- In alcuni contesti, come un'istruzione `return`, è sintatticamente più semplice creare una tupla che un array.
- Se stai passando una sequenza come argomento a una funzione, l'uso delle tuple riduce il potenziale di comportamento imprevisto dovuto all'aliasing.
- Per motivi di prestazioni. Il compilatore può specializzarsi sul tipo.

Altri criteri

Gli array sono più comuni delle tuple, principalmente perché sono mutabili. Ma ci sono alcuni casi in cui potresti preferire le tuple:

- In alcuni contesti, come un'istruzione `return`, è sintatticamente più semplice creare una tupla che un array.
- Se stai passando una sequenza come argomento a una funzione, l'uso delle tuple riduce il potenziale di comportamento imprevisto dovuto all'aliasing.
- Per motivi di prestazioni. Il compilatore può specializzarsi sul tipo.

Altri criteri

Gli array sono più comuni delle tuple, principalmente perché sono mutabili. Ma ci sono alcuni casi in cui potresti preferire le tuple:

- In alcuni contesti, come un'istruzione `return`, è sintatticamente più semplice creare una tupla che un array.
- Se stai passando una sequenza come argomento a una funzione, l'uso delle tuple riduce il potenziale di comportamento imprevisto dovuto all'aliasing.
- Per motivi di prestazioni. Il compilatore può specializzarsi sul tipo.

Poiché le tuple sono immutabili, non forniscono funzioni come `sort!` E `reverse!`, Che modificano gli array esistenti.

Altri criteri

Gli array sono più comuni delle tuple, principalmente perché sono mutabili. Ma ci sono alcuni casi in cui potresti preferire le tuple:

- In alcuni contesti, come un'istruzione `return`, è sintatticamente più semplice creare una tupla che un array.
- Se stai passando una sequenza come argomento a una funzione, l'uso delle tuple riduce il potenziale di comportamento imprevisto dovuto all'aliasing.
- Per motivi di prestazioni. Il compilatore può specializzarsi sul tipo.

Poiché le tuple sono immutabili, non forniscono funzioni come `sort!` E `reverse!`, Che modificano gli array esistenti.

Ma Julia fornisce la funzione incorporata `sort`, che accetta un array e restituisce un nuovo array con gli stessi elementi in ordine ordinato, e `reverse`, che accetta qualsiasi sequenza e restituisce una sequenza dello stesso tipo in ordine inverso.

Section 8

Debug

Strutture dati composte

Array, dizionari e tuple sono esempi di strutture dati; in questa lezione abbiamo iniziato a vedere strutture di dati composti, come array di tuple o dizionari che contengono tuple come chiavi e array come valori.

Le strutture di dati composite sono utili, ma sono soggette a quelli che chiamo errori di forma; ovvero, errori causati quando una struttura dati ha tipo, dimensione o struttura errati.

Ad esempio, se aspetti un array con un numero ori di e ti passo un intero semplice (non in un array), non funzionerà.

Julia permette di allegare un tipo agli elementi di una sequenza. Il modo in cui eseguire questa operazione è descritto in dettaglio in Multiple Dispatch. La specifica del tipo elimina molti errori formali.

Section 9

Glossario

Glossario

tupla Una sequenza immutabile di elementi in cui ogni elemento può avere il proprio tipo.

Glossario

tupla Una sequenza immutabile di elementi in cui ogni elemento può avere il proprio tipo.

assegnazione di tupla Un assegnamento con una sequenza a destra e una tupla di variabili a sinistra. Il lato destro viene valutato e quindi i suoi valori vengono assegnati alle variabili a sinistra.

Glossario

tupla Una sequenza immutabile di elementi in cui ogni elemento può avere il proprio tipo.

assegnazione di tupla Un assegnamento con una sequenza a destra e una tupla di variabili a sinistra. Il lato destro viene valutato e quindi i suoi valori vengono assegnati alle variabili a sinistra.

gather (raccogliere) L'operazione di assemblaggio di una tupla di argomenti di lunghezza variabile.

Glossario

tupla Una sequenza immutabile di elementi in cui ogni elemento può avere il proprio tipo.

assegnazione di tupla Un assegnamento con una sequenza a destra e una tupla di variabili a sinistra. Il lato destro viene valutato e quindi i suoi valori vengono assegnati alle variabili a sinistra.

gather (raccogliere) L'operazione di assemblaggio di una tupla di argomenti di lunghezza variabile.

spargere (scatter) L'operazione di trattare una sequenza come un elenco di argomenti.

Glossario

tupla Una sequenza immutabile di elementi in cui ogni elemento può avere il proprio tipo.

assegnazione di tupla Un assegnamento con una sequenza a destra e una tupla di variabili a sinistra. Il lato destro viene valutato e quindi i suoi valori vengono assegnati alle variabili a sinistra.

gather (raccogliere) L'operazione di assemblaggio di una tupla di argomenti di lunghezza variabile.

spargere (scatter) L'operazione di trattare una sequenza come un elenco di argomenti.

oggetto zip Il risultato della chiamata di una funzione integrata zip; un oggetto che itera attraverso una sequenza di tuple.

Glossario

tupla Una sequenza immutabile di elementi in cui ogni elemento può avere il proprio tipo.

assegnazione di tupla Un assegnamento con una sequenza a destra e una tupla di variabili a sinistra. Il lato destro viene valutato e quindi i suoi valori vengono assegnati alle variabili a sinistra.

gather (raccogliere) L'operazione di assemblaggio di una tupla di argomenti di lunghezza variabile.

spargere (scatter) L'operazione di trattare una sequenza come un elenco di argomenti.

oggetto zip Il risultato della chiamata di una funzione integrata zip; un oggetto che itera attraverso una sequenza di tuple.

iteratore Un oggetto che può scorrere una sequenza, ma che non fornisce operatori e funzioni di indici.

Glossario

tupla Una sequenza immutabile di elementi in cui ogni elemento può avere il proprio tipo.

assegnazione di tupla Un assegnamento con una sequenza a destra e una tupla di variabili a sinistra. Il lato destro viene valutato e quindi i suoi valori vengono assegnati alle variabili a sinistra.

gather (raccogliere) L'operazione di assemblaggio di una tupla di argomenti di lunghezza variabile.

spargere (scatter) L'operazione di trattare una sequenza come un elenco di argomenti.

oggetto zip Il risultato della chiamata di una funzione integrata zip; un oggetto che itera attraverso una sequenza di tuple.

iteratore Un oggetto che può scorrere una sequenza, ma che non fornisce operatori e funzioni di indici.

struttura dati Una raccolta di valori correlati, spesso organizzati in array, dizionari, tuple, ecc.

Glossario

tupla Una sequenza immutabile di elementi in cui ogni elemento può avere il proprio tipo.

assegnazione di tupla Un assegnamento con una sequenza a destra e una tupla di variabili a sinistra. Il lato destro viene valutato e quindi i suoi valori vengono assegnati alle variabili a sinistra.

gather (raccogliere) L'operazione di assemblaggio di una tupla di argomenti di lunghezza variabile.

spargere (scatter) L'operazione di trattare una sequenza come un elenco di argomenti.

oggetto zip Il risultato della chiamata di una funzione integrata zip; un oggetto che itera attraverso una sequenza di tuple.

iteratore Un oggetto che può scorrere una sequenza, ma che non fornisce operatori e funzioni di indici.

struttura dati Una raccolta di valori correlati, spesso organizzati in array, dizionari, tuple, ecc.

errore di forma Un errore causato perché un valore ha la forma sbagliata; cioè il tipo o la dimensione sbagliati.

Section 10

Esercizi

aaaa

aaaa