Fondamenti di Informatica (Elettronici)

Da THINK JULIA – Capitolo 3

19 ottobre 2020

Le Funzioni (Seconda parte)[1]

1. Parametri e argomenti
2. Le variabili e i parametri sono locali
3. Diagrammi di stack
4. Funzioni produttive e funzioni vuote
5. Perché le funzioni?
6. Debug
7. Glossario
8. Esercizi

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

Section 1

Parametri e argomenti

Argomenti delle funzioni

Molte funzioni richiedono argomenti.

Ad esempio, con "`sin`" si passa un numero come argomento. Alcune funzioni richiedono più di un argomento: "`parse`" ne accetta due, un tipo numerico e una stringa. All'interno della funzione, gli argomenti sono assegnati a variabili chiamate parametri.

```
function printtwice(bruce)
     println(bruce)
     println(bruce)
end
```

Questa funzione assegna l'argomento a un parametro chiamato `bruce`. Quando la funzione viene chiamata, stampa due volte il valore del parametro. Questa funzione funziona con qualsiasi valore che può essere stampato.

```
julia> printtwice("Spam")
Spam
Spam
julia> printtwice(42)
42
42
julia> printtwice(pi)
pi
pi
```

Espressioni come arogomenti

Le stesse regole di composizione che si applicano alle funzioni primitive si applicano anche alle funzioni definite dal programmatore, quindi possiamo usare qualsiasi tipo di espressione come argomento per `printtwice` :

```
julia> printtwice("Spam "^4)
Spam Spam Spam Spam
Spam Spam Spam Spam
julia> printtwice(cos(pi)) -1.0
-1.0
```

L'argomento viene valutato prima che la funzione venga chiamata, quindi negli esempi le espressioni `"Spam"^4` e `cos(pi)` vengono valutate solo una volta.

Si può anche usare una variabile come argomento:

```
julia> michael = "Eric, the half a bee."
"Eric, the half a bee."
julia> printtwice(michael)
Eric, the half a bee.
```

Section 2

Le variabili e i parametri sono locali

Oggetti locali

Quando si crea una variabile all'interno di una funzione, questa è locale. Esiste solo all'interno della funzione.

```
function cattwice(part1, part2)
    concat = part1 * part2
    printtwice(concat)
end
```

Questa funzione accetta due argomenti, li concatena e stampa il risultato due volte:

```
julia> line1 = "Bing tiddle "
"Bing tiddle "
julia> line2 = "tiddle bang."
"tiddle bang."
julia> cattwice(line1, line2)
Bing tiddle tiddle bang.
Bing tiddle tiddle bang.
```

Anche i parametri sono locali. Al di fuori di `printtwice`, non esistono cose come `bruce`. Quando "cattwice" termina, la variabile locale "concat" viene distrutta.

```
julia> println(concat)
ERROR: UndefVarError: concat not defined
```

Section 3

Diagrammi di stack

Diagrammi di stack

Per tenere traccia di* quali variabili* possano essere utilizzate e dove, possiamo disegnare un diagramma di stack (pila).

I diagrammi a pila mostrano il valore di ogni variabile, ma mostrano anche la funzione a cui appartiene ciascuna variabile.

Ogni funzione è rappresentata da un frame (cornice).

Un frame è un riquadro con accanto il nome di una funzione e al suo interno i parametri e le variabili della funzione.

Il diagramma dello stack per l'esempio precedente è mostrato in Figura.

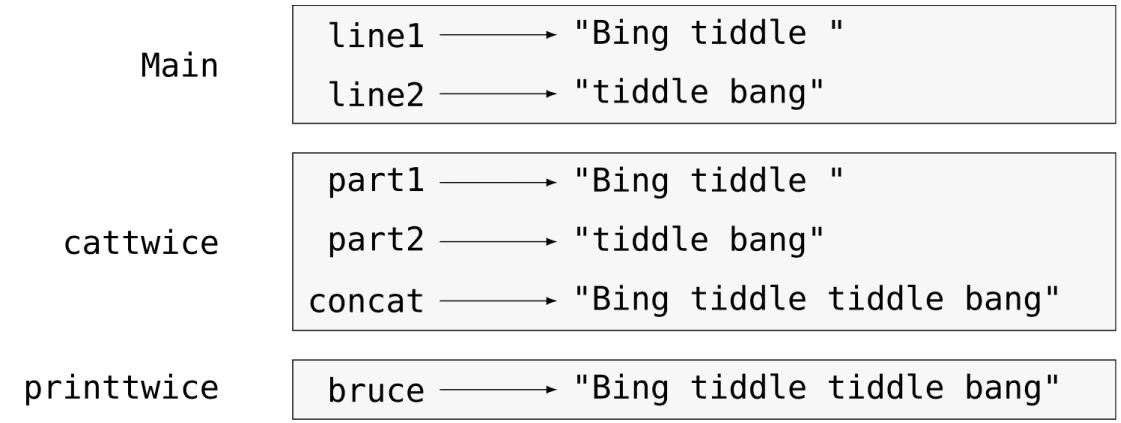

Figure 2. Stack diagram

Framestack

I frame sono disposti in una pila che indica quale funzione ha chiamato quale altra, e così via.

In questo esempio, `printtwice` è stata chiamata da `cattwice`, e `cattwice` è stata chiamata da `Main`, che è un nome speciale per il frame principale.

Quando si crea una variabile al di fuori di qualsiasi funzione, essa appartiene a `Main`.

Ogni parametro fa riferimento allo stesso valore del suo argomento corrispondente.

Quindi, `part1` ha lo stesso valore di `line1`, `part2` ha lo stesso valore di `line2` e `bruce` ha lo stesso valore di `concat`.

Se si verifica un errore durante una chiamata di funzione, Julia stampa il nome della funzione, il nome della funzione che l'ha chiamata e il nome della funzione che ha chiamato quest'ultima, fino a `Main`.

Stacktrace

Ad esempio, se provi ad accedere a `concat` da `printtwice`, si ottiene un 'UndefVarError':

```
  ERROR: UndefVarError: concat not defined
  Stacktrace:
[1] printtwice at ./REPL[1]:2 [inlined]
[2] cattwice(::String, ::String) at ./REPL[2]:3
```

Questo elenco di funzioni è chiamato `stacktrace`.

Ti dice in quale file di programma si è verificato l'errore, in quale riga, e quali funzioni erano in esecuzione in quel momento.

Mostra anche la riga di codice che ha causato l'errore.

L'ordine delle funzioni nello `stacktrace` è l'inverso dell'ordine dei frame nel diagramma dello stack. La funzione attualmente in esecuzione è in alto.

Section 4

Funzioni produttive e funzioni vuote

Funzioni vuote

Alcune delle funzioni che abbiamo utilizzato, come le funzioni matematiche, restituiscono risultati; per mancanza di un nome migliore, le chiamiamo funzioni feconde o produttive.

Altre funzioni, come `printtwice`, eseguono un'azione ma non restituiscono un valore.

Sono chiamate funzioni vuote.

Quando si chiama una funzione feconda, quasi sempre si vuole fare qualcosa con il risultato; per esempio, si potrebbe assegnarlo a una variabile o usarlo come parte di un'espressione:

```
x = cos(radians)
golden = (sqrt(x) + 1) / 2
```

Funzioni produttive

Quando si chiama una funzione in modalità interattiva, Julia visualizza il risultato:

```
julia> sqrt (5)
2.23606797749979
```

Ma in uno script, se chiami una funzione feconda da sola, il valore restituito viene perso per sempre!

```
sqrt(5)
```

Uscite:

```
2.23606797749979
```

Questo script calcola la radice quadrata di 5, ma poiché non memorizza né visualizza il risultato, non è molto utile.

Le funzioni vuote potrebbero visualizzare qualcosa sullo schermo o avere qualche altro effetto, ma non hanno un valore di ritorno.

nothing

Se assegni il risultato a una variabile, ottieni un valore speciale chiamato `nothing`.

```
julia> result = printtwice("Bing")
Bing
Bing
julia> show(result)
nothing
```

Per stampare il valore `nothing`, devi usare la funzione `show` che è come `print` ma può gestire il valore `nothing`.

Il valore `nothing` è diverso dalla stringa "`nothing`". È un valore speciale che il suo tipo:

```
julia> typeof(nothing)
Nothing
```

Le funzioni che abbiamo scritto finora sono tutte vuote. Inizieremo a scrivere funzioni produttive tra pochi capitoli.

Section 5

Perché le funzioni?

Dividi un programma in funzioni

Potrebbe non essere chiaro il motivo per cui vale la pena dividere un programma in funzioni.

Ci sono diversi motivi:

1. La creazione di una nuova funzione ti dà l'opportunità di dare un nome a un gruppo di istruzioni, il che rende il tuo programma più facile da leggere e da "debuggare".

Dividi un programma in funzioni

Potrebbe non essere chiaro il motivo per cui vale la pena dividere un programma in funzioni.

Ci sono diversi motivi:

1. La creazione di una nuova funzione ti dà l'opportunità di dare un nome a un gruppo di istruzioni, il che rende il tuo programma più facile da leggere e da "debuggare".
2. Le funzioni possono ridurre le dimensioni di un programma eliminando il codice ripetitivo.

Dividi un programma in funzioni

Potrebbe non essere chiaro il motivo per cui vale la pena dividere un programma in funzioni.

Ci sono diversi motivi:

1. La creazione di una nuova funzione ti dà l'opportunità di dare un nome a un gruppo di istruzioni, il che rende il tuo programma più facile da leggere e da "debuggare".
2. Le funzioni possono ridurre le dimensioni di un programma eliminando il codice ripetitivo.
3. In seguito, se apporti un cambiamento, devi farlo in un solo posto.

Dividi un programma in funzioni

Potrebbe non essere chiaro il motivo per cui vale la pena dividere un programma in funzioni.

Ci sono diversi motivi:

1. La creazione di una nuova funzione ti dà l'opportunità di dare un nome a un gruppo di istruzioni, il che rende il tuo programma più facile da leggere e da "debuggare".
2. Le funzioni possono ridurre le dimensioni di un programma eliminando il codice ripetitivo.
3. In seguito, se apporti un cambiamento, devi farlo in un solo posto.
4. Dividere un lungo programma in funzioni consente di eseguire il debug delle parti una alla volta e quindi di assemblarle in un insieme funzionante.

Dividi un programma in funzioni

Potrebbe non essere chiaro il motivo per cui vale la pena dividere un programma in funzioni.

Ci sono diversi motivi:

1. La creazione di una nuova funzione ti dà l'opportunità di dare un nome a un gruppo di istruzioni, il che rende il tuo programma più facile da leggere e da "debuggare".
2. Le funzioni possono ridurre le dimensioni di un programma eliminando il codice ripetitivo.
3. In seguito, se apporti un cambiamento, devi farlo in un solo posto.
4. Dividere un lungo programma in funzioni consente di eseguire il debug delle parti una alla volta e quindi di assemblarle in un insieme funzionante.
5. Le funzioni ben progettate sono spesso utili per molti programmi.

Dividi un programma in funzioni

Potrebbe non essere chiaro il motivo per cui vale la pena dividere un programma in funzioni.

Ci sono diversi motivi:

1. La creazione di una nuova funzione ti dà l'opportunità di dare un nome a un gruppo di istruzioni, il che rende il tuo programma più facile da leggere e da "debuggare".
2. Le funzioni possono ridurre le dimensioni di un programma eliminando il codice ripetitivo.
3. In seguito, se apporti un cambiamento, devi farlo in un solo posto.
4. Dividere un lungo programma in funzioni consente di eseguire il debug delle parti una alla volta e quindi di assemblarle in un insieme funzionante.
5. Le funzioni ben progettate sono spesso utili per molti programmi.
6. Una volta scritte ed eseguito il debug, puoi riutilizzarle.

Dividi un programma in funzioni

Potrebbe non essere chiaro il motivo per cui vale la pena dividere un programma in funzioni.

Ci sono diversi motivi:

1. La creazione di una nuova funzione ti dà l'opportunità di dare un nome a un gruppo di istruzioni, il che rende il tuo programma più facile da leggere e da "debuggare".
2. Le funzioni possono ridurre le dimensioni di un programma eliminando il codice ripetitivo.
3. In seguito, se apporti un cambiamento, devi farlo in un solo posto.
4. Dividere un lungo programma in funzioni consente di eseguire il debug delle parti una alla volta e quindi di assemblarle in un insieme funzionante.
5. Le funzioni ben progettate sono spesso utili per molti programmi.
6. Una volta scritte ed eseguito il debug, puoi riutilizzarle.
7. In Julia, le funzioni possono migliorare molto le prestazioni.

Section 6

Debug

Una abilità importante

Una delle abilità più importanti che acquisirai è la capacità di debuggare.

Sebbene possa essere frustrante, il `debugging` è una delle parti della programmazione più intellettualmente ricche, stimolanti e interessanti.

In un certo senso il `debugging` è come un lavoro investigativo.

1. Ti trovi di fronte a indizi e devi dedurre i processi e gli eventi che hanno portato ai risultati che vedi.

Una abilità importante

Una delle abilità più importanti che acquisirai è la capacità di debuggare.

Sebbene possa essere frustrante, il `debugging` è una delle parti della programmazione più intellettualmente ricche, stimolanti e interessanti.

In un certo senso il `debugging` è come un lavoro investigativo.

1. Ti trovi di fronte a indizi e devi dedurre i processi e gli eventi che hanno portato ai risultati che vedi.
2. Una volta che abbiamo un'idea di cosa sta andando storto, modifichiamo il programma e riproviamo.

Una abilità importante

Una delle abilità più importanti che acquisirai è la capacità di debuggare.

Sebbene possa essere frustrante, il `debugging` è una delle parti della programmazione più intellettualmente ricche, stimolanti e interessanti.

In un certo senso il `debugging` è come un lavoro investigativo.

1. Ti trovi di fronte a indizi e devi dedurre i processi e gli eventi che hanno portato ai risultati che vedi.
2. Una volta che abbiamo un'idea di cosa sta andando storto, modifichiamo il programma e riproviamo.
3. Se l'ipotesi era corretta, si può prevedere il risultato della modifica e fare un passo avanti verso un programma funzionante.

Una abilità importante

Una delle abilità più importanti che acquisirai è la capacità di debuggare.

Sebbene possa essere frustrante, il `debugging` è una delle parti della programmazione più intellettualmente ricche, stimolanti e interessanti.

In un certo senso il `debugging` è come un lavoro investigativo.

1. Ti trovi di fronte a indizi e devi dedurre i processi e gli eventi che hanno portato ai risultati che vedi.
2. Una volta che abbiamo un'idea di cosa sta andando storto, modifichiamo il programma e riproviamo.
3. Se l'ipotesi era corretta, si può prevedere il risultato della modifica e fare un passo avanti verso un programma funzionante.
4. Se l'ipotesi era sbagliata, bisogna inventarne una nuova.

Una abilità importante

Una delle abilità più importanti che acquisirai è la capacità di debuggare.

Sebbene possa essere frustrante, il `debugging` è una delle parti della programmazione più intellettualmente ricche, stimolanti e interessanti.

In un certo senso il `debugging` è come un lavoro investigativo.

1. Ti trovi di fronte a indizi e devi dedurre i processi e gli eventi che hanno portato ai risultati che vedi.
2. Una volta che abbiamo un'idea di cosa sta andando storto, modifichiamo il programma e riproviamo.
3. Se l'ipotesi era corretta, si può prevedere il risultato della modifica e fare un passo avanti verso un programma funzionante.
4. Se l'ipotesi era sbagliata, bisogna inventarne una nuova.

Una abilità importante

Una delle abilità più importanti che acquisirai è la capacità di debuggare.

Sebbene possa essere frustrante, il `debugging` è una delle parti della programmazione più intellettualmente ricche, stimolanti e interessanti.

In un certo senso il `debugging` è come un lavoro investigativo.

1. Ti trovi di fronte a indizi e devi dedurre i processi e gli eventi che hanno portato ai risultati che vedi.
2. Una volta che abbiamo un'idea di cosa sta andando storto, modifichiamo il programma e riproviamo.
3. Se l'ipotesi era corretta, si può prevedere il risultato della modifica e fare un passo avanti verso un programma funzionante.
4. Se l'ipotesi era sbagliata, bisogna inventarne una nuova.

Per alcune persone, la programmazione e il debug sono la stessa cosa. In questo senso, la programmazione è il processo di debugging graduale di un programma fino a quando non fa quello che si vuole: Test Driven Development (TDD)

L'idea è che si dovrebbe iniziare con un programma funzionante e apportare piccole modifiche, eseguendo il debug man mano che il programma viene esteso.

Section 7

Glossario

Glossario I

funzione Una sequenza di istruzioni con nome che esegue alcune operazioni utili. Le funzioni possono o non possono accettare argomenti e possono o non possono produrre un risultato.

Glossario I

funzione Una sequenza di istruzioni con nome che esegue alcune operazioni utili. Le funzioni possono o non possono accettare argomenti e possono o non possono produrre un risultato.

definizione di funzione Un'istruzione che crea una nuova funzione, specificandone il nome, i parametri e le istruzioni che contiene.

Glossario I

funzione Una sequenza di istruzioni con nome che esegue alcune operazioni utili. Le funzioni possono o non possono accettare argomenti e possono o non possono produrre un risultato.

definizione di funzione Un'istruzione che crea una nuova funzione, specificandone il nome, i parametri e le istruzioni che contiene.

oggetto funzione Un valore creato da una definizione di funzione. Il nome della funzione è una variabile che fa riferimento a un oggetto funzione.

Glossario I

funzione Una sequenza di istruzioni con nome che esegue alcune operazioni utili. Le funzioni possono o non possono accettare argomenti e possono o non possono produrre un risultato.

definizione di funzione Un'istruzione che crea una nuova funzione, specificandone il nome, i parametri e le istruzioni che contiene.

oggetto funzione Un valore creato da una definizione di funzione. Il nome della funzione è una variabile che fa riferimento a un oggetto funzione.

intestazione La prima riga di una definizione di funzione.

Glossario I

funzione Una sequenza di istruzioni con nome che esegue alcune operazioni utili. Le funzioni possono o non possono accettare argomenti e possono o non possono produrre un risultato.

definizione di funzione Un'istruzione che crea una nuova funzione, specificandone il nome, i parametri e le istruzioni che contiene.

oggetto funzione Un valore creato da una definizione di funzione. Il nome della funzione è una variabile che fa riferimento a un oggetto funzione.

intestazione La prima riga di una definizione di funzione.

corpo La sequenza di istruzioni all'interno di una definizione di funzione.

Glossario I

funzione Una sequenza di istruzioni con nome che esegue alcune operazioni utili. Le funzioni possono o non possono accettare argomenti e possono o non possono produrre un risultato.

definizione di funzione Un'istruzione che crea una nuova funzione, specificandone il nome, i parametri e le istruzioni che contiene.

oggetto funzione Un valore creato da una definizione di funzione. Il nome della funzione è una variabile che fa riferimento a un oggetto funzione.

intestazione La prima riga di una definizione di funzione.

corpo La sequenza di istruzioni all'interno di una definizione di funzione.

parametro Un nome utilizzato all'interno di una funzione per fare riferimento al valore passato come argomento.

Glossario I

funzione Una sequenza di istruzioni con nome che esegue alcune operazioni utili. Le funzioni possono o non possono accettare argomenti e possono o non possono produrre un risultato.

definizione di funzione Un'istruzione che crea una nuova funzione, specificandone il nome, i parametri e le istruzioni che contiene.

oggetto funzione Un valore creato da una definizione di funzione. Il nome della funzione è una variabile che fa riferimento a un oggetto funzione.

intestazione La prima riga di una definizione di funzione.

corpo La sequenza di istruzioni all'interno di una definizione di funzione.

parametro Un nome utilizzato all'interno di una funzione per fare riferimento al valore passato come argomento.

chiamata di funzione Un'istruzione che esegue una funzione. Consiste nel nome della funzione seguito da un elenco di argomenti tra parentesi.

Glossario I

funzione Una sequenza di istruzioni con nome che esegue alcune operazioni utili. Le funzioni possono o non possono accettare argomenti e possono o non possono produrre un risultato.

definizione di funzione Un'istruzione che crea una nuova funzione, specificandone il nome, i parametri e le istruzioni che contiene.

oggetto funzione Un valore creato da una definizione di funzione. Il nome della funzione è una variabile che fa riferimento a un oggetto funzione.

intestazione La prima riga di una definizione di funzione.

corpo La sequenza di istruzioni all'interno di una definizione di funzione.

parametro Un nome utilizzato all'interno di una funzione per fare riferimento al valore passato come argomento.

chiamata di funzione Un'istruzione che esegue una funzione. Consiste nel nome della funzione seguito da un elenco di argomenti tra parentesi.

argomento Un valore fornito a una funzione quando la funzione viene chiamata. Questo valore viene assegnato al parametro corrispondente nella funzione.

Glossario II

variabile locale Una variabile definita all'interno di una funzione. Una variabile locale può essere utilizzata solo all'interno della sua funzione.

Glossario II

variabile locale Una variabile definita all'interno di una funzione. Una variabile locale può essere utilizzata solo all'interno della sua funzione.

valore di ritorno Il risultato di una funzione. Se una chiamata di funzione viene utilizzata come espressione, il valore restituito è il valore dell'espressione.

Glossario II

variabile locale Una variabile definita all'interno di una funzione. Una variabile locale può essere utilizzata solo all'interno della sua funzione.

valore di ritorno Il risultato di una funzione. Se una chiamata di funzione viene utilizzata come espressione, il valore restituito è il valore dell'espressione.

funzione produttiva o feconda Una funzione che restituisce un valore.

Glossario II

variabile locale Una variabile definita all'interno di una funzione. Una variabile locale può essere utilizzata solo all'interno della sua funzione.

valore di ritorno Il risultato di una funzione. Se una chiamata di funzione viene utilizzata come espressione, il valore restituito è il valore dell'espressione.

funzione produttiva o feconda Una funzione che restituisce un valore.

funzione void Una funzione che restituisce sempre `nothing`.

Glossario II

variabile locale Una variabile definita all'interno di una funzione. Una variabile locale può essere utilizzata solo all'interno della sua funzione.

valore di ritorno Il risultato di una funzione. Se una chiamata di funzione viene utilizzata come espressione, il valore restituito è il valore dell'espressione.

funzione produttiva o feconda Una funzione che restituisce un valore.

funzione void Una funzione che restituisce sempre `nothing`.

`nothing` Un valore speciale restituito dalle funzioni vuote.

Glossario II

variabile locale Una variabile definita all'interno di una funzione. Una variabile locale può essere utilizzata solo all'interno della sua funzione.

valore di ritorno Il risultato di una funzione. Se una chiamata di funzione viene utilizzata come espressione, il valore restituito è il valore dell'espressione.

funzione produttiva o feconda Una funzione che restituisce un valore.

funzione void Una funzione che restituisce sempre `nothing`.

`nothing` Un valore speciale restituito dalle funzioni vuote.

composizione Utilizzo di un'espressione come parte di un'espressione più ampia o di un'istruzione come parte di un'istruzione più ampia.

Glossario II

variabile locale Una variabile definita all'interno di una funzione. Una variabile locale può essere utilizzata solo all'interno della sua funzione.

valore di ritorno Il risultato di una funzione. Se una chiamata di funzione viene utilizzata come espressione, il valore restituito è il valore dell'espressione.

funzione produttiva o feconda Una funzione che restituisce un valore.

funzione void Una funzione che restituisce sempre `nothing`.

`nothing` Un valore speciale restituito dalle funzioni vuote.

composizione Utilizzo di un'espressione come parte di un'espressione più ampia o di un'istruzione come parte di un'istruzione più ampia.

flusso di esecuzione Le istruzioni eseguite nell'ordine.

Glossario II

variabile locale Una variabile definita all'interno di una funzione. Una variabile locale può essere utilizzata solo all'interno della sua funzione.

valore di ritorno Il risultato di una funzione. Se una chiamata di funzione viene utilizzata come espressione, il valore restituito è il valore dell'espressione.

funzione produttiva o feconda Una funzione che restituisce un valore.

funzione void Una funzione che restituisce sempre `nothing`.

`nothing` Un valore speciale restituito dalle funzioni vuote.

composizione Utilizzo di un'espressione come parte di un'espressione più ampia o di un'istruzione come parte di un'istruzione più ampia.

flusso di esecuzione Le istruzioni eseguite nell'ordine.

diagramma di stack Una rappresentazione grafica della pila di funzioni, delle loro variabili, e dei valori a cui si riferiscono.

Glossario II

variabile locale Una variabile definita all'interno di una funzione. Una variabile locale può essere utilizzata solo all'interno della sua funzione.

valore di ritorno Il risultato di una funzione. Se una chiamata di funzione viene utilizzata come espressione, il valore restituito è il valore dell'espressione.

funzione produttiva o feconda Una funzione che restituisce un valore.

funzione void Una funzione che restituisce sempre `nothing`.

`nothing` Un valore speciale restituito dalle funzioni vuote.

composizione Utilizzo di un'espressione come parte di un'espressione più ampia o di un'istruzione come parte di un'istruzione più ampia.

flusso di esecuzione Le istruzioni eseguite nell'ordine.

diagramma di stack Una rappresentazione grafica della pila di funzioni, delle loro variabili, e dei valori a cui si riferiscono.

frame Una casella in un diagramma di stack, che rappresenta una chiamata di funzione. Contiene le variabili locali e i parametri della funzione.

Glossario II

variabile locale Una variabile definita all'interno di una funzione. Una variabile locale può essere utilizzata solo all'interno della sua funzione.

valore di ritorno Il risultato di una funzione. Se una chiamata di funzione viene utilizzata come espressione, il valore restituito è il valore dell'espressione.

funzione produttiva o feconda Una funzione che restituisce un valore.

funzione void Una funzione che restituisce sempre `nothing`.

`nothing` Un valore speciale restituito dalle funzioni vuote.

composizione Utilizzo di un'espressione come parte di un'espressione più ampia o di un'istruzione come parte di un'istruzione più ampia.

flusso di esecuzione Le istruzioni eseguite nell'ordine.

diagramma di stack Una rappresentazione grafica della pila di funzioni, delle loro variabili, e dei valori a cui si riferiscono.

frame Una casella in un diagramma di stack, che rappresenta una chiamata di funzione. Contiene le variabili locali e i parametri della funzione.

stacktrace Un elenco delle funzioni in esecuzione, stampato quando si verifica un'eccezione.

Section 8

Esercizi

aaaa

aaaa