Fondamenti di Informatica (Elettronici)

THINK JULIA – Chapter 3

14 ottobre 2020

3. Funzioni (a)[1]

1. Chiamate di funzione
2. Funzioni matematiche
3. Composizione
4. Aggiunta di nuove funzioni
5. Definizioni e uso
6. Flusso di esecuzione

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

Funzioni

Nel contesto della programmazione, una funzione è una sequenza denominata di istruzioni che esegue un calcolo.

Funzioni

Nel contesto della programmazione, una funzione è una sequenza denominata di istruzioni che esegue un calcolo.

Funzione: definizione

Quando si definisce una funzione, si specifica il nome, la sequenza di parametri, e la sequenza di istruzioni.

Funzione: chiamata

Successivamente, si può "chiamare" la funzione per nome.

Section 1

Chiamate di funzione

Terminologia

Abbiamo già visto un esempio di chiamata di funzione:

```
julia> println("Hello, World!")
Hello, World!
```

- Il nome della funzione è `println`

Terminologia

Abbiamo già visto un esempio di chiamata di funzione:

```
julia> println("Hello, World!")
Hello, World!
```

- Il nome della funzione è `println`
- L'espressione tra parentesi è chiamata argomento della funzione.

Terminologia

Abbiamo già visto un esempio di chiamata di funzione:

```
julia> println("Hello, World!")
Hello, World!
```

- Il nome della funzione è `println`
- L'espressione tra parentesi è chiamata argomento della funzione.
- È comune dire che una funzione "accetta" un argomento e "restituisce" un risultato.

Terminologia

Abbiamo già visto un esempio di chiamata di funzione:

```
julia> println("Hello, World!")
Hello, World!
```

- Il nome della funzione è `println`
- L'espressione tra parentesi è chiamata argomento della funzione.
- È comune dire che una funzione "accetta" un argomento e "restituisce" un risultato.
- Il risultato è anche chiamato valore di ritorno.

Conversione di valori

Julia fornisce funzioni che convertono i valori da un tipo a un altro.

- La funzione `parse` prende una stringa e la converte in qualsiasi tipo di numero, se possibile, o si lamenta altrimenti:

```
julia> parse(Int64, "32")
32
julia> parse(Float64, "3.14159")
3.14159
julia> parse(Int64, "Hello")
ERROR: ArgumentError: invalid base 10 digit 'H' in "Hello"
```

```
julia> trunc(Int64, 3.99999)
 3
julia> trunc(Int64, -2.3)
-2
```

Conversione di valori

Julia fornisce funzioni che convertono i valori da un tipo a un altro.

- La funzione `parse` prende una stringa e la converte in qualsiasi tipo di numero, se possibile, o si lamenta altrimenti:

```
julia> parse(Int64, "32")
32
julia> parse(Float64, "3.14159")
3.14159
julia> parse(Int64, "Hello")
ERROR: ArgumentError: invalid base 10 digit 'H' in "Hello"
```

- `trunc` può convertire valori a virgola mobile in numeri interi, ma non vengono arrotondati; taglia la parte frazionaria:

```
julia> trunc(Int64, 3.99999)
 3
julia> trunc(Int64, -2.3)
-2
```

Converting values

- `float` converte interi in numeri in virgola mobile (floating-point):

```
julia> float(32)
32.0
```

```
julia> string(32)
"32"
julia> string(3.14159)
"3.14159"
```

Converting values

- `float` converte interi in numeri in virgola mobile (floating-point):

```
julia> float(32)
32.0
```

- Infine, `string` converte il suo argomento in una stringa:

```
julia> string(32)
"32"
julia> string(3.14159)
"3.14159"
```

Section 2

Funzioni matematiche

In Julia, è direttamente disponibile la maggior parte delle funzioni matematiche

```
ratio = signal_power / noise_power
decibels = 10 * log10(ratio)
```

Questo primo esempio usa `log10` per calcolare un rapporto segnale/rumore in decibel (assumendo che `signal_power` e `noise_power` siano definiti). Viene fornito anche `log`, che calcola i logaritmi naturali.

```
radians = 0.7
height = sin(radians)
```

Questo secondo esempio trova il seno di `radians`.

Il nome della variabile indica che `sin` e le altre funzioni trigonometriche (`cos`, `tan`, ecc.) Accettano argomenti in radianti.

Esempi

Per convertire i gradi in radianti, dividi per 180 e moltiplica per π:

```
julia> degrees = 45
45
julia> radians = degrees / 180 * pi
0.7853981633974483
julia> sin(radians)
0.7071067811865475
```

Il valore della costante `pi` è un'approssimazione in virgola mobile di π, con una precisione di circa 16 cifre.

Se conosci la trigonometria, puoi controllare il risultato precedente confrontandolo con la radice quadrata di due diviso due: $(\sqrt{2})/2$

```
julia> sqrt(2)/2
0.7071067811865476
```

Section 3

Composizione

Composizione delle espressioni

Una delle caratteristiche più utili dei linguaggi di programmazione è la loro capacità di prendere piccoli blocchi di istruzioni e comporli.

L'argomento di una funzione può essere qualsiasi espressione, inclusi gli operatori aritmetici:

```
x = sin(degrees / 360 * 2 * pi)
```

E anche chiamate di funzione:

```
x = exp(log(x+1))
```

Ogni espressione rappresenta il suo valore

Quasi ovunque si possa mettere un valore, si può mettere un'espressione arbitraria,

con un'eccezione: il lato sinistro di un'istruzione di assegnazione deve essere un nome di variabile. Qualsiasi altra espressione sul lato sinistro è un errore di sintassi (vedremo le eccezioni a questa regola più avanti).

```
julia> minutes = hours * 60 # right
120
julia> hours * 60 = minutes # wrong!
ERROR: syntax: "60" is not a valid function argument name
```

Semantica dell'istruzione di assegnazione

1. Valuta la espressione a destra;

Ogni espressione rappresenta il suo valore

Quasi ovunque si possa mettere un valore, si può mettere un'espressione arbitraria,

con un'eccezione: il lato sinistro di un'istruzione di assegnazione deve essere un nome di variabile. Qualsiasi altra espressione sul lato sinistro è un errore di sintassi (vedremo le eccezioni a questa regola più avanti).

```
julia> minutes = hours * 60 # right
120
julia> hours * 60 = minutes # wrong!
ERROR: syntax: "60" is not a valid function argument name
```

Semantica dell'istruzione di assegnazione

1. Valuta la espressione a destra;
2. Assegna il valore valutato alla variabile a sinistra.

Section 4

Aggiunta di nuove funzioni

Sintassi della definizione della funzione

Una definizione di funzione specifica il nome di una nuova funzione e la sequenza di istruzioni che vengono eseguite quando la funzione viene chiamata. Ecco un esempio:

```
function printlyrics()
    println("I'm a lumberjack, and I'm okay.")
    println("I sleep all night and I work all day.")
end
```

1. **function** è una parola chiave che indica che questa è una definizione di funzione. Il nome della funzione è `printlyrics`.

Sintassi della definizione della funzione

Una definizione di funzione specifica il nome di una nuova funzione e la sequenza di istruzioni che vengono eseguite quando la funzione viene chiamata. Ecco un esempio:

```
function printlyrics()
    println("I'm a lumberjack, and I'm okay.")
    println("I sleep all night and I work all day.")
end
```

1. **function** è una parola chiave che indica che questa è una definizione di funzione. Il nome della funzione è `printlyrics`.
2. Le regole per i nomi delle funzioni sono le stesse dei nomi delle variabili.

Sintassi della definizione della funzione

Una definizione di funzione specifica il nome di una nuova funzione e la sequenza di istruzioni che vengono eseguite quando la funzione viene chiamata. Ecco un esempio:

```
function printlyrics()
    println("I'm a lumberjack, and I'm okay.")
    println("I sleep all night and I work all day.")
end
```

1. **function** è una parola chiave che indica che questa è una definizione di funzione. Il nome della funzione è `printlyrics`.
2. Le regole per i nomi delle funzioni sono le stesse dei nomi delle variabili.
3. Dovresti evitare di avere una variabile e una funzione con lo stesso nome.

Sintassi della definizione della funzione

Una definizione di funzione specifica il nome di una nuova funzione e la sequenza di istruzioni che vengono eseguite quando la funzione viene chiamata. Ecco un esempio:

```
function printlyrics()
    println("I'm a lumberjack, and I'm okay.")
    println("I sleep all night and I work all day.")
end
```

1. **function** è una parola chiave che indica che questa è una definizione di funzione. Il nome della funzione è `printlyrics`.
2. Le regole per i nomi delle funzioni sono le stesse dei nomi delle variabili.
3. Dovresti evitare di avere una variabile e una funzione con lo stesso nome.
4. Le parentesi vuote dopo il nome indicano che questa funzione non ha argomenti.

Sintassi della definizione della funzione

Una definizione di funzione specifica il nome di una nuova funzione e la sequenza di istruzioni che vengono eseguite quando la funzione viene chiamata. Ecco un esempio:

```
function printlyrics()
    println("I'm a lumberjack, and I'm okay.")
    println("I sleep all night and I work all day.")
end
```

1. **function** è una parola chiave che indica che questa è una definizione di funzione. Il nome della funzione è `printlyrics`.
2. Le regole per i nomi delle funzioni sono le stesse dei nomi delle variabili.
3. Dovresti evitare di avere una variabile e una funzione con lo stesso nome.
4. Le parentesi vuote dopo il nome indicano che questa funzione non ha argomenti.
5. La prima riga di definizione è chiamata intestazione; il resto è chiamato corpo.

Sintassi della definizione della funzione

Una definizione di funzione specifica il nome di una nuova funzione e la sequenza di istruzioni che vengono eseguite quando la funzione viene chiamata. Ecco un esempio:

```
function printlyrics()
    println("I'm a lumberjack, and I'm okay.")
    println("I sleep all night and I work all day.")
end
```

1. **function** è una parola chiave che indica che questa è una definizione di funzione. Il nome della funzione è `printlyrics`.
2. Le regole per i nomi delle funzioni sono le stesse dei nomi delle variabili.
3. Dovresti evitare di avere una variabile e una funzione con lo stesso nome.
4. Le parentesi vuote dopo il nome indicano che questa funzione non ha argomenti.
5. La prima riga di definizione è chiamata intestazione; il resto è chiamato corpo.
6. Il corpo termina con la parola chiave `end`. Può contenere un numero qualsiasi di dichiarazioni.

Sintassi della definizione della funzione

Una definizione di funzione specifica il nome di una nuova funzione e la sequenza di istruzioni che vengono eseguite quando la funzione viene chiamata. Ecco un esempio:

```
function printlyrics()
    println("I'm a lumberjack, and I'm okay.")
    println("I sleep all night and I work all day.")
end
```

1. **function** è una parola chiave che indica che questa è una definizione di funzione. Il nome della funzione è `printlyrics`.
2. Le regole per i nomi delle funzioni sono le stesse dei nomi delle variabili.
3. Dovresti evitare di avere una variabile e una funzione con lo stesso nome.
4. Le parentesi vuote dopo il nome indicano che questa funzione non ha argomenti.
5. La prima riga di definizione è chiamata intestazione; il resto è chiamato corpo.
6. Il corpo termina con la parola chiave `end`. Può contenere un numero qualsiasi di dichiarazioni.
7. Per una maggiore leggibilità, il corpo della funzione deve essere rientrato.

Sintassi della chiamata = sintassi delle funzioni primitive

```
julia> printlyrics()
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
```

Una volta definita una funzione, è possibile utilizzarla all'interno di un'altra:

```
julia> function repeatlyrics()
    printlyrics()
    printlyrics()
end
```

E poi chiama `repeatlyrics`:

```
julia> repeatlyrics()
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
```

Section 5

Definizioni e uso

Utilizzo delle definizioni di funzione

Mettendo insieme i frammenti di codice precedenti, l'intero programma diventa:

```
function printlyrics()
    println("I'm a lumberjack, and I'm okay.")
    println("I sleep all night and I work all day.")
end
function repeatlyrics()
    printlyrics()
    printlyrics()
end
repeatlyrics()
```

Questo programma contiene due definizioni di funzioni: `printlyrics` e `repeatlyrics`. Le definizioni di funzione vengono eseguite proprio come le altre istruzioni, ma l'effetto è creare oggetti funzione.

Le istruzioni all'interno della funzione non vengono eseguite finché la funzione non viene chiamata e la definizione della funzione non genera alcun output.

Come ci si potrebbe aspettare, è necessario creare una funzione prima di poterla eseguire. La definizione della funzione deve essere eseguita prima della chiamata.

Section 6

Flusso di esecuzione

Flusso di esecuzione

Per assicurarci che una funzione sia definita prima del suo primo utilizzo, devi conoscere le istruzioni eseguite nell'ordine. Questo è chiamato flusso di esecuzione.

1. L'esecuzione inizia sempre dalla prima istruzione del programma.

Flusso di esecuzione

Per assicurarci che una funzione sia definita prima del suo primo utilizzo, devi conoscere le istruzioni eseguite nell'ordine. Questo è chiamato flusso di esecuzione.

1. L'esecuzione inizia sempre dalla prima istruzione del programma.
2. Le dichiarazioni vengono eseguite una alla volta, nell'ordine dall'alto verso il basso.

Flusso di esecuzione

Per assicurarci che una funzione sia definita prima del suo primo utilizzo, devi conoscere le istruzioni eseguite nell'ordine. Questo è chiamato flusso di esecuzione.

1. L'esecuzione inizia sempre dalla prima istruzione del programma.
2. Le dichiarazioni vengono eseguite una alla volta, nell'ordine dall'alto verso il basso.
3. Le definizioni di funzione non alterano il flusso di esecuzione del programma, ma le istruzioni all'internodella funzione vengono eseguite solo quando la funzione viene chiamata.

Flusso di esecuzione

Per assicurarci che una funzione sia definita prima del suo primo utilizzo, devi conoscere le istruzioni eseguite nell'ordine. Questo è chiamato flusso di esecuzione.

1. L'esecuzione inizia sempre dalla prima istruzione del programma.
2. Le dichiarazioni vengono eseguite una alla volta, nell'ordine dall'alto verso il basso.
3. Le definizioni di funzione non alterano il flusso di esecuzione del programma, ma le istruzioni all'internodella funzione vengono eseguite solo quando la funzione viene chiamata.
4. Una chiamata di funzione è come una deviazione nel flusso di esecuzione.

Stack (pila) di esecuzione

Alla chiamata di una funzione, invece di passare all'istruzione successiva, il flusso salta al corpo della funzione, esegue le istruzioni lì contenute e poi torna per riprendere da dove si era interrotto.

- Sembra semplice, ricordando che una funzione può chiamarne un'altra.

Stack (pila) di esecuzione

Alla chiamata di una funzione, invece di passare all'istruzione successiva, il flusso salta al corpo della funzione, esegue le istruzioni lì contenute e poi torna per riprendere da dove si era interrotto.

- Sembra semplice, ricordando che una funzione può chiamarne un'altra.
- Mentre si trova nel mezzo di una funzione, il programma potrebbe dover eseguire le istruzioni in un'altra funzione.

Stack (pila) di esecuzione

Alla chiamata di una funzione, invece di passare all'istruzione successiva, il flusso salta al corpo della funzione, esegue le istruzioni lì contenute e poi torna per riprendere da dove si era interrotto.

- Sembra semplice, ricordando che una funzione può chiamarne un'altra.
- Mentre si trova nel mezzo di una funzione, il programma potrebbe dover eseguire le istruzioni in un'altra funzione.
- Quindi, durante l'esecuzione di quella nuova funzione, il programma potrebbe dover eseguire ancora un'altra funzione!

Stack (pila) di esecuzione

Alla chiamata di una funzione, invece di passare all'istruzione successiva, il flusso salta al corpo della funzione, esegue le istruzioni lì contenute e poi torna per riprendere da dove si era interrotto.

- Sembra semplice, ricordando che una funzione può chiamarne un'altra.
- Mentre si trova nel mezzo di una funzione, il programma potrebbe dover eseguire le istruzioni in un'altra funzione.
- Quindi, durante l'esecuzione di quella nuova funzione, il programma potrebbe dover eseguire ancora un'altra funzione!

Stack (pila) di esecuzione

Alla chiamata di una funzione, invece di passare all'istruzione successiva, il flusso salta al corpo della funzione, esegue le istruzioni lì contenute e poi torna per riprendere da dove si era interrotto.

- Sembra semplice, ricordando che una funzione può chiamarne un'altra.
- Mentre si trova nel mezzo di una funzione, il programma potrebbe dover eseguire le istruzioni in un'altra funzione.
- Quindi, durante l'esecuzione di quella nuova funzione, il programma potrebbe dover eseguire ancora un'altra funzione!

Fortunatamente, Julia è capace di tenere traccia di dove si trova. Pertanto, ogni volta che una funzione viene completata, il programma riprende da dove era stato interrotto nella funzione chiamante.

Stack (pila) di esecuzione

Alla chiamata di una funzione, invece di passare all'istruzione successiva, il flusso salta al corpo della funzione, esegue le istruzioni lì contenute e poi torna per riprendere da dove si era interrotto.

- Sembra semplice, ricordando che una funzione può chiamarne un'altra.
- Mentre si trova nel mezzo di una funzione, il programma potrebbe dover eseguire le istruzioni in un'altra funzione.
- Quindi, durante l'esecuzione di quella nuova funzione, il programma potrebbe dover eseguire ancora un'altra funzione!

Fortunatamente, Julia è capace di tenere traccia di dove si trova. Pertanto, ogni volta che una funzione viene completata, il programma riprende da dove era stato interrotto nella funzione chiamante.

Quando arriva alla fine del programma, il flusso di esecuzione termina. Quando il controllo passa da una funzione a un'altra, un record con i dati di ritorno viene inserito nello stack di esecuzione. Quando la funzione termina, il record viene rimosso (estratto) dallo stack.