# Fondamenti di Informatica (Elettronici)

THINK JULIA – Capitolo 5

25 ottobre 2020

# 5. Condizionali (parte a)[1]

1. Divisione intera e modulo
2. Espressioni booleane
3. Operatori logici
4. Esecuzione condizionale
5. Esecuzione alternativa
6. Condizionali concatenati
7. Condizionali annidati

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

## Contenuti del capitolo

L'argomento principale di questo capitolo è l'istruzione `if`, che esegue codice differente a seconda dello stato del programma.

## Contenuti del capitolo

L'argomento principale di questo capitolo è l'istruzione `if`, che esegue codice differente a seconda dello stato del programma.

## Due nuovi operatori

Ma prima si vuole introdurre due nuovi operatori: divisione intera e operazione modulo, ovvero resto della divisione intera.

Section 1

# Divisione intera e modulo

# Divisione intera

```
julia> minutes = 105
105
julia> minutes / 60
1.75
```

Normalmente non scriviamo le ore con punti decimali. La divisione intera `floor` (in basso) (simbolo ÷) restituisce il numero intero di ore, arrotondando per difetto:

```
julia> hours = minutes ÷ 60
1
```

Inoltre, si possono estrarre la cifra (le cifre) più a destra da un numero. Ad esempio, "x % 10" restituisce la cifra più a destra di un intero "x" (in base 10).

Allo stesso modo "x % 100" restituisce le ultime due cifre. Per ottenere il resto in minuti, possiamo sottrarre il numero di ore (=1) in minuti:

```
julia> remainder = minutes - hours * 60
45
```

# Operatore modulo

Un'alternativa è usare l'operatore modulo, %, che divide due numeri e restituisce il resto

```
julia> remainder = minutes % 60
45
```

**Suggerimento**

L'operatore modulo è più utile di quanto sembri. Ad esempio, possiamo verificare se un numero è divisibile per un altro: se `x % y` è zero, `x` è divisibile per `y`.

Inoltre, possiamo estrarre la cifra o le cifre più a destra da un numero. Ad esempio, `x % 10` restituisce la cifra più a destra di un intero `x` (in base 10). Allo stesso modo, `x % 100` restituisce le ultime due cifre.

Section 2

# Espressioni booleane

# Valori di verità

Un'espressione booleana è un'espressione che può essere `true` o `false`. I seguenti esempi usano l'operatore `==`, che confronta due operandi e produce `true` se sono uguali e `false` altrimenti:

```
julia> 5 == 5
true
julia> 5 == 6
false
```

`true` e `false` sono valori speciali che appartengono al tipo `Bool`; non sono stringhe:

```
julia> typeof(true)
Bool
julia> typeof(false)
Bool
```

# Operatori relazionali

L'operatore == è uno degli operatori relazionali; gli altri sono:

```
x != y # x is not equal to y
x (\ne TAB) y # x is not equal to y
x > y # x is greater than y
x < y # x is less than y
x >= y # x is greater than or equal to y x$\ge$y # (\ge TAB)
x <= y # x is less than or equal to y x$\le$y # (\le TAB)
```

# Operatori relazionali

L'operatore == è uno degli operatori relazionali; gli altri sono:

```
x != y # x is not equal to y
x (\ne TAB) y # x is not equal to y
x > y # x is greater than y
x < y # x is less than y
x >= y # x is greater than or equal to y x$\ge$y # (\ge TAB)
x <= y # x is less than or equal to y x$\le$y # (\le TAB)
```

**Nota bene**

Sebbene queste operazioni ti siano probabilmente familiari, i simboli Julia sono diversi dai simboli matematici.
Un errore comune consiste nell'usare un singolo segno di uguale (=) invece di un doppio segno di uguale (==). Ricorda che = è un operatore di assegnazione e == è un operatore relazionale. Non esistono cose come =< o =>.

Section 3

# Operatori logici

# Operatori logici

Ci sono tre operatori logici: `&&` (And), `||` (Or) e `!` (Not). La semantica (significato) di questi operatori è simile al loro significato in inglese.

Ad esempio, `x > 0 && x < 10` è `true` solo se `x` è maggiore di 0 e minore di 10.

`n % 2 == 0 || n % 3 == 0` è vero se una o entrambe le condizioni sono vere, cioè se il numero è divisibile per 2 o 3.

Sia `&&` che `||` si associano a destra, ma `&&` ha una precedenza maggiore di `||`.

Infine, l'operatore `!` nega un'espressione booleana, quindi `!(x > y)` è vera se `x > y` è falsa, cioè se `x` è minore o uguale a `y`.

Section 4

# Esecuzione condizionale

# La forma più semplice è l'istruzione `if`

Per scrivere programmi utili, abbiamo quasi sempre bisogno della capacità di controllare le condizioni e modificare di conseguenza il comportamento del programma.

Le dichiarazioni condizionali ci danno questa capacità.

```
if x > 0
    println ("x è positivo")
end
```

- L'espressione booleana dopo `if` è chiamata condizione. Se è vera, viene eseguita l'istruzione indentata. In caso contrario, non accade nulla.

# La forma più semplice è l'istruzione `if`

Per scrivere programmi utili, abbiamo quasi sempre bisogno della capacità di controllare le condizioni e modificare di conseguenza il comportamento del programma.

Le dichiarazioni condizionali ci danno questa capacità.

```
if x > 0
    println ("x è positivo")
end
```

- L'espressione booleana dopo `if` è chiamata condizione. Se è vera, viene eseguita l'istruzione indentata. In caso contrario, non accade nulla.
- Le istruzioni `if` hanno la stessa struttura delle definizioni di funzione: un'intestazione seguita dal corpo che termina con la parola chiave `end`.

# La forma più semplice è l'istruzione `if`

Per scrivere programmi utili, abbiamo quasi sempre bisogno della capacità di controllare le condizioni e modificare di conseguenza il comportamento del programma.

Le dichiarazioni condizionali ci danno questa capacità.

```
if x > 0
    println ("x è positivo")
end
```

- L'espressione booleana dopo `if` è chiamata condizione. Se è vera, viene eseguita l'istruzione indentata. In caso contrario, non accade nulla.
- Le istruzioni `if` hanno la stessa struttura delle definizioni di funzione: un'intestazione seguita dal corpo che termina con la parola chiave `end`.
- Dichiarazioni come questa sono chiamate dichiarazioni composte. Non c'è limite al numero di istruzioni che possono apparire nel corpo.

# La forma più semplice è l'istruzione `if`

Per scrivere programmi utili, abbiamo quasi sempre bisogno della capacità di controllare le condizioni e modificare di conseguenza il comportamento del programma.

Le dichiarazioni condizionali ci danno questa capacità.

```
if x > 0
    println ("x è positivo")
end
```

- L'espressione booleana dopo `if` è chiamata condizione. Se è vera, viene eseguita l'istruzione indentata. In caso contrario, non accade nulla.
- Le istruzioni `if` hanno la stessa struttura delle definizioni di funzione: un'intestazione seguita dal corpo che termina con la parola chiave `end`.
- Dichiarazioni come questa sono chiamate dichiarazioni composte. Non c'è limite al numero di istruzioni che possono apparire nel corpo.
- Occasionalmente, è utile avere un corpo senza istruzioni (di solito come segnaposto per il codice che non hai ancora scritto).

Section 5

# Esecuzione alternativa

# If (then) else

Una seconda forma dell'istruzione `if` è "l'esecuzione alternativa", in cui ci sono due possibilità e la condizione logica determina quale viene eseguita. La sintassi è simile a questa:

```
if x % 2 == 0
    println("x is even")
else
    println("x is odd")
end
```

- Se il resto è 0 quando `x` viene diviso per 2, allora sappiamo che `x` è pari e il programma visualizza un messaggio appropriato.

# If (then) else

Una seconda forma dell'istruzione `if` è "l'esecuzione alternativa", in cui ci sono due possibilità e la condizione logica determina quale viene eseguita. La sintassi è simile a questa:

```
if x % 2 == 0
    println("x is even")
else
    println("x is odd")
end
```

- Se il resto è 0 quando `x` viene diviso per 2, allora sappiamo che `x` è pari e il programma visualizza un messaggio appropriato.
- Se la condizione è `false`, viene eseguita la seconda serie di istruzioni.

# If (then) else

Una seconda forma dell'istruzione `if` è "l'esecuzione alternativa", in cui ci sono due possibilità e la condizione logica determina quale viene eseguita. La sintassi è simile a questa:

```
if x % 2 == 0
    println("x is even")
else
    println("x is odd")
end
```

- Se il resto è 0 quando `x` viene diviso per 2, allora sappiamo che `x` è pari e il programma visualizza un messaggio appropriato.
- Se la condizione è `false`, viene eseguita la seconda serie di istruzioni.
- Poiché la condizione deve essere `true` o `false`, verrà eseguita esattamente una delle alternative.

# If (then) else

Una seconda forma dell'istruzione `if` è "l'esecuzione alternativa", in cui ci sono due possibilità e la condizione logica determina quale viene eseguita. La sintassi è simile a questa:

```
if x % 2 == 0
    println("x is even")
else
    println("x is odd")
end
```

- Se il resto è 0 quando `x` viene diviso per 2, allora sappiamo che `x` è pari e il programma visualizza un messaggio appropriato.
- Se la condizione è `false`, viene eseguita la seconda serie di istruzioni.
- Poiché la condizione deve essere `true` o `false`, verrà eseguita esattamente una delle alternative.
- Le alternative sono chiamate rami, perché sono rami nel flusso di esecuzione.

Section 6

# Condizionali concatenati

# Condizionali concatenati 1/2

A volte ci sono più di due possibilità e abbiamo bisogno di più di due rami. Un modo per esprimere un calcolo del genere è un condizionale concatenato:

```
if x < y
    println ("x è minore di y")
elseif x > y
    println ("x è maggiore di y")
else
    println ("x e y sono uguali")
end
```

Di nuovo, verrà eseguito esattamente un ramo. Non c'è limite al numero di istruzioni `elseif`.

Se c'è una clausola `else`, deve essere alla fine, e una soltanto.

# Condizionali concatenati 2/2

```
if choice == "a"
    draw_a()
elseif choice == "b"
    draw_b()
elseif choice == "c"
    draw_c()
end
```

Ogni condizione viene controllata in ordine. Se il primo è `false`, viene controllata la successiva e così via.

Se una di loro è `true`, il ramo corrispondente viene eseguito e l'istruzione termina.

Anche se più di una condizione è `true`, viene eseguito solo il primo ramo `true`.

Section 7

# Condizionali annidati

# Condizionali annidati all'interno di un altro

Un condizionale può anche essere annidato all'interno di un altro. Avremmo potuto scrivere l'esempio nella sezione precedente in questo modo:

```
if x == y
    println("x and y are equal")
else
    if x < y
        println("x is less than y") else
        println("x is greater than y") end
end
```

Il condizionale esterno contiene due rami:

- il primo ramo contiene una semplice dichiarazione;

# Condizionali annidati all'interno di un altro

Un condizionale può anche essere annidato all'interno di un altro. Avremmo potuto scrivere l'esempio nella sezione precedente in questo modo:

```
if x == y
    println("x and y are equal")
else
    if x < y
        println("x is less than y") else
        println("x is greater than y") end
end
```

Il condizionale esterno contiene due rami:

- il primo ramo contiene una semplice dichiarazione;
- il secondo ramo contiene un'altra istruzione `if`, che ha due rami propri.

# Condizionali annidati all'interno di un altro

Un condizionale può anche essere annidato all'interno di un altro. Avremmo potuto scrivere l'esempio nella sezione precedente in questo modo:

```
if x == y
    println("x and y are equal")
else
    if x < y
        println("x is less than y") else
        println("x is greater than y") end
end
```

Il condizionale esterno contiene due rami:

- il primo ramo contiene una semplice dichiarazione;
- il secondo ramo contiene un'altra istruzione `if`, che ha due rami propri.

# Condizionali annidati all'interno di un altro

Un condizionale può anche essere annidato all'interno di un altro. Avremmo potuto scrivere l'esempio nella sezione precedente in questo modo:

```
if x == y
    println("x and y are equal")
else
    if x < y
        println("x is less than y") else
        println("x is greater than y") end
end
```

Il condizionale esterno contiene due rami:

- il primo ramo contiene una semplice dichiarazione;
- il secondo ramo contiene un'altra istruzione `if`, che ha due rami propri.

# Evitare condizionali annidati

Sebbene l'indentazione non obbligatoria delle istruzioni renda la struttura evidente, i condizionali annidati diventano difficili da leggere molto rapidamente. È una buona idea evitarli quando puoi.

Gli operatori logici spesso forniscono un modo per semplificare le istruzioni condizionali annidate. Ad esempio, possiamo riscrivere il codice seguente utilizzando un singolo condizionale:

```
if 0 < x
    if x < 10
    println("x is a positive single-digit number.") end
end
```

L'istruzione print viene eseguita solo se superiamo entrambi i condizionali, quindi possiamo ottenere lo stesso effetto con l'operatore &&:

```
if 0 < x && x < 10
    println("x is a positive single-digit number.")
end
```

Per questo tipo di condizione, Julia fornisce una sintassi più concisa:

```
if 0 < x < 10
    println("x is a positive single-digit number.")
end
```