# Fondamenti di Informatica (Elettronici)

THINK JULIA – Capitolo 8a (prima parte)

6 novembre 2020

# 8. Stringhe (a)[1]

1. Caratteri
2. Una stringa è una sequenza
3. Lunghezza
4. Visita (Traversal)
5. Slices (fette) di stringa
6. Le stringhe sono immutabili

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

Le stringhe non sono come gli interi, i `float` e i booleani.

Una stringa è una sequenza, il che significa che è una collezione ordinata di altri valori. In questo capitolo vedremo come accedere ai caratteri che compongono una stringa, e impareremo alcune delle funzioni di supporto delle stringhe fornite da Julia.

Section 1

# Caratteri

# Standard ASCII

I parlanti di lingua inglese hanno familiarità con caratteri come le lettere dell'alfabeto (A, B, C, ...), i numeri e la punteggiatura comune.

Questi caratteri sono standardizzati e mappati su valori interi compresi tra 0 e 127 dallo standard `ASCII` (American Standard Code for Information Interchange).

Ci sono, ovviamente, molti altri caratteri usati in lingue diverse dall'inglese, comprese varianti dei caratteri `ASCII` con accenti e altre modifiche, script correlati come il cirillico e il greco, e script completamente estranei ad `ASCII` e inglese, inclusi Arabo, cinese, ebraico, hindi, giapponese e coreano.

# Standard Unicode

Lo standard Unicode affronta le complessità di cosa sia esattamente un carattere, ed è generalmente accettato come lo standard definitivo che affronta questo problema.

Fornisce un numero univoco per ogni carattere su scala mondiale.

Un valore di tipo `Char` rappresenta un singolo carattere ed è racchiuso tra virgolette singole:

```
julia> 'x'
'x': ASCII/Unicode U+0078 (category Ll: Letter, lowercase)
julia> ' '
' ': Unicode U+01f34c (category So: Symbol, other)
julia> typeof('x')
Char
```

Anche gli emoji fanno parte dello standard Unicode. (`\:banana:` TAB)

Section 2

# Una stringa è una sequenza

# String = sequenza di caratteri

Una stringa è una sequenza di caratteri. Puoi accedere ai caratteri uno alla volta con l'operatore parentesi quadre:

```
julia> fruit = "banana"
"banana"
julia> letter = fruit[1]
'b': ASCII/Unicode U+0062 (category Ll: Letter, lowercase)
```

La seconda istruzione seleziona il carattere numero 1 d1 `fruit` e lo assegna a `letter`. L'espressione tra parentesi quadre è chiamata indice. L'indice indica quale carattere nella sequenza si desidera (da cui il nome).

Tutta l'indicizzazione in Julia è basata su 1: il primo elemento di qualsiasi oggetto indicizzato con numeri interi si trova all'indice 1 e l'ultimo elemento all'indice `end`.

# Accesso alle stringhe tramite indice

```
julia> fruit[end]
'a': ASCII/Unicode U+0061 (category Ll: Letter, lowercase)
```

Come indice si può usare un'espressione che contiene variabili e operatori:

```
julia> i = 1
1
julia> fruit[i+1]
'a': ASCII/Unicode U+0061 (category Ll: Letter, lowercase)
julia> fruit[end-1]
'n': ASCII/Unicode U+006e (category Ll: Letter, lowercase)
```

Ma il valore dell'indice deve essere un numero intero. Altrimenti si ottiene:

```
julia> letter = fruit[1.5]
ERROR: MethodError: no method matching getindex(::String, ::Float64)
```

Section 3

# Lunghezza

# Restituisce il numero di elementi di un tipo indicizzato

```
julia> fruits = "🍌🍎🍐"
"🍌🍎🍐"

julia> len = length(fruits)
3
```

# Le stringhe sono codificate con UTF-8

Le stringhe vengono codificate utilizzando la codifica UTF-8. UTF-8 è una codifica a larghezza variabile, il che significa che non tutti i caratteri sono codificati nello stesso numero di byte.

La funzione `sizeof` fornisce il numero di byte in una stringa:

```
julia> sizeof("🍌")
4
```

Poiché un'`emoji` è codificata in 4 byte e l'indicizzazione delle stringhe è basata sui byte, il quinto elemento di `fruits` è l'inizio del carattere successivo.

# Usa “nextind” per ottenere gli indici delle stringhe

Ciò significa anche che non ogni indice di byte in una stringa UTF-8 è necessariamente un indice valido per un carattere. Se indicizzi in una stringa con un indice di byte non valido, viene generato un errore:

```
julia> fruits[2]
ERROR: StringIndexError("🍌 🍎 🍐", 2)
```

Nel caso di `fruits`, il carattere è una sequenza di quattro byte, quindi gli indici 2, 3 e 4 non sono validi e l'indice del carattere successivo è 5; questo successivo indice valido può essere calcolato da `nextind (fruits, 1)`, e l'indice successivo al successivo da `nextind (fruits, 5)` e così via.

Section 4

# Visita (Traversal)

# String traversal

Molti calcoli comportano l'elaborazione di una stringa un carattere alla volta.

Spesso iniziano dall'inizio, selezionano ogni carattere alla volta, fanno qualcosa e continuano fino alla fine.

Questo modello di elaborazione è chiamato "traversal" (visita). Un modo per scrivere un `traversal` è con un ciclo `while`:

```
index = firstindex(fruits)
while index <= sizeof(fruits)
    letter = fruits[index]
    println(letter)
    global index = nextind(fruits, index)
end
```

Questo ciclo attraversa la stringa e visualizza ogni lettera su una riga da sola. La condizione del ciclo è `index <= sizeof (fruit)`, quindi quando `index` è maggiore del numero di byte nella stringa, la condizione è `false` e il corpo del ciclo non viene eseguito.

La funzione `firstindex` restituisce il primo byte valido di `index`.

La parola chiave `global` prima di `index` indica che si desidera riassegnare la variabile `index` definita in `Main` (vedere Variabili globali).

Section 5

# Slices (fette) di stringa

# Intervallo di indici

Un segmento di una stringa è chiamato slice. La selezione di una fetta è simile alla selezione di un carattere:

```
julia> str = "Julius Caesar";
julia> str[1:6] "Julius"
```

L'operatore `[n:m]` restituisce la parte della stringa dal byte "`n`" al byte "`m`". Quindi è necessaria la stessa cautela della indicizzazione semplice.

La parola chiave `end` può essere utilizzata per indicare l'ultimo byte della stringa:

```
julia> str[8:end]
"Caesar"
```

# La stringa vuota

Se il primo indice è maggiore del secondo il risultato è una stringa vuota, rappresentata da due virgolette doppie:

```
julia> str[8:7]
""
```

Una stringa vuota non contiene caratteri e ha lunghezza 0, ma a parte questo, è uguale a qualsiasi altra stringa.

**Esercizio 8-3**

Continuando l'esempio, cosa pensi che significhi `str[:]` ? Provalo e guarda.

Section 6

# Le stringhe sono immutabili

# Avviso sui tipi immutabili

Si è tentati di usare l'operatore "" sul lato sinistro di una assegnazione, con l'intenzione di cambiare un carattere in una stringa. Per esempio:

```
julia> greeting = "Hello, world!"
"Hello, world!"
julia> greeting[1] = 'J'
ERROR: MethodError: no method matching setindex!(::String, ::Char,
```

Il motivo dell'errore è che le stringhe non sono modificabili, il che significa che non è possibile modificare una stringa esistente.

# Avviso sui tipi immutabili

Si è tentati di usare l'operatore "" sul lato sinistro di una assegnazione, con l'intenzione di cambiare un carattere in una stringa. Per esempio:

```
julia> greeting = "Hello, world!"
"Hello, world!"
julia> greeting[1] = 'J'
ERROR: MethodError: no method matching setindex!(::String, ::Char,
```

Il motivo dell'errore è che le stringhe non sono modificabili, il che significa che non è possibile modificare una stringa esistente.

Puoi creare una nuova stringa che sia una variazione dell'originale:

```
julia> greeting = "J" * greeting[2:end]
"Jello, world!"
```

Questo esempio concatena una nuova prima lettera con una slice di `greeting`. Non ha effetto sulla stringa originale.