

# Linguaggi di Programmazione – Programming Languages AA-AY 2024-2025

Progetto luglio 2025 – July 2025 Project

## Quantità fisiche con “dimensioni”

Marco Antoniotti, Fabio Sartori  
Dipartimento di Informatica, Sistemistica e Comunicazione  
Università degli Studi di Milano Bicocca

### Scadenza

La consegna del progetto è fissata per il giorno 27 luglio 2025 entro le 23:55 GMT+1.

## 1 Introduzione

Le quantità fisiche sono composte da una *magnitudine* numerica e da una *dimensione*. La dimensione è espressa in *unità* stabilite dal *Système International d’Unités* (Sistema Internazionale – S.I.) o derivate. Ad esempio,  $42 \text{ N m}$ , oppure  $9.8 \text{ m s}^{-2}$ .

Fare operazioni con *quantità* richiede di controllare la correttezza e la compatibilità delle dimensioni. Come sempre, non si possono confrontare pere con mele.

Lo scopo di questo progetto è la costruzione di due librerie (in Prolog ed in Common Lisp) per la manipolazione – per l’appunto – di quantità fisiche dimensionate.

### 1.1 Unità del S.I.

Il S.I. prescrive sette unità di base mostrate nella Tabella 1. Potete guardare la pagina Wikipedia [https://en.wikipedia.org/wiki/International\\_System\\_of\\_Units](https://en.wikipedia.org/wiki/International_System_of_Units) per trovare molte più unità derivate, più o meno standardizzate.

Symbol	name
kg	kilogram
m	metre
s	second
A	Ampere
K	Kelvin
cd	candela
mol	mole

Table 1: Le sette unità di base del S.I.

Come potrete notare, nella pagina Wikipedia a cui facciamo riferimento, ad ogni unità sono associati, come minimo, un *nome* ed un *simbolo*; altre caratteristiche sono associeribili ad ogni unità, in particolare, per le unità derivate, la loro *espansione* in termini delle unità base.

Una quantità può essere scritta in modi diversi. Ad esempio:

$42 \text{ N m}$  o  $42 \text{ m N}$  o anche  $\text{n M } 42$ .

Symbol	Name
Bq	Becquerel
°C	degree Celsius
C	Coulomb
F	Farad
Gy	Gray
Hz	Hertz
H	Henry
J	Joule
kat	Katal
lm	lumen
lx	lux
N	Newton
Ω	Ohm
Pa	Pascal
rad	radian
S	Siemens
Sv	Sievert
sr	steradian
T	Tesla
V	Volt
W	Watt
Wb	Weber

Table 2: Le principali unità S.I. derivate. In ordine alfabetico.

Sorge quindi il problema di come scrivere le quantità in modo “canonico”; ai fini di questo progetto, il modo canonico è quello che rappresenta le quantità come  $\langle \text{num} \rangle \langle \text{dimensione} \rangle$ , dove  $\langle \text{dimensione} \rangle$  è una moltiplicazione di unità, potenzialmente esponenziate (e.g.,  $\text{kg m s}^{-2}$ ). Inoltre, e questo è un punto molto importante, l’ordine un cui le unità appaiono è quello della Tabella 1 per le unità di base e quello della Tabella 2 per le unità derivate (ovvero alfabetico, secondo la terminologia inglese). In altre parole dovete assicurarsi che una quantità data in input sia poi sempre ritornata in formato canonico. Ad esempio:

$$\begin{aligned} 42 \text{ N m} &\Rightarrow 42 \text{ m N}, \\ 42 \text{ m s}^{-2} \text{ kg} &\Rightarrow 42 \text{ kg m s}^{-2} \end{aligned}$$

*Buona parte di questo progetto consiste nel costruire delle procedure di riordinamento delle dimensioni (con le loro potenze) in modo da ottenere un risultato canonico.*

## 2 Operazioni da implementare e rappresentazione

Le librerie che implemerete dovranno contenere alcune operazioni standard per la manipolazione delle varie quantità.

- *Ispezione.*
- *Riscrittura.*
- *Operazioni aritmetiche.*

La rappresentazione delle quantità dovrà essere la seguente.

**Prolog.** Le quantità devono essere rappresentate da termini siffatti:

`q(Number, Dimension)`

per i quali si può scrivere il predicato:

```
is_quantity(q(N, D)) :- number(N), is_dimension(D).
```

Il predicato `is_dimension(D)` controlla semplicemente che l'argomento sia una “dimensione”, con alcune libertà; ad esempio:

```
?- is_dimension(m).
true

?- is_dimension(m * s).
true

?- is_dimension(s * m).
true
% is_dimension non controlla che la dimensione sia in forma canonica.

?- is_dimension(m * 'A').
true

?- is_dimension(m * 'A' * z ** 4).
false
% 'z' non e' una dimensione riconosciuta.

?- is_dimension(m * ('A' ** 2) ** 4).
true
% Notate che anche questa e' una dipartita dalla forma canonica.
```

Come potete notare, una dimensione è rappresentata con i simboli S.I. Di conseguenza, dovete quotare i simboli maiuscoli, come '`A`' (ampere) negli esempi precedenti.

**Common Lisp.** In questo caso la rappresentazione risulta più semplice dato che Common Lisp, di default, rende tutti i nomi maiusoli. Le quantità devono essere rappresentate (analogamente al caso del Prolog) con oggetti siffatti:

```
(q number dimension)
```

per i quali si può scrivere il predicato:

```
(defun is-quantity (q)
  (and (listp q)
        (eq 'q (first q))
        (let ((n (second q))
              (d (third q)))
          )
        (and (numberp n)
             (is-dimension d))))
```

Il predicato `is-dimension` controlla semplicemente che il suo argomento sia una “dimensione”, con alcune libertà; ad esempio:

```
cl-prompt> (is-dimension 'm)
T

cl-prompt> (is-dimension '(* m s))
true

cl-prompt> (is-dimension '(* s m))
true
```

```
;;; is-dimension non controlla che la dimensione sia in forma canonica.

cl-prompt> (is-dimension '(* m A))
true

cl-prompt> (is-dimension '(* m A (expt z 4)))
false
;;; 'z' non e' una dimensione riconosciuta.

cl-prompt> (is-dimension '( * m (expt (expt A 2) 4)))
true
;;; Notate che anche questa e' una dipartita dalla forma canonica.
```

Anche in questo caso, una dimensione è rappresentata con i simboli S.I.

## 2.1 Operazioni da implementare

Le vostre librerie dovranno implementare le seguenti operazioni. *Notate che dovete anche trattare i multipli delle varie unità di misura: cm, microfarad, etc.*

**Prolog.** I predicati che dovete implementare (oltre a quelli descritti sopra) servono a ispezionare le varie strutture dati e a fare calcoli con quantità.

### Predicate **is\_si\_unit(*S*)**

Il predicato **is\_si\_unit** è vero quando *S* è un simbolo che denota un'unità S.I. (base o derivata).

### Predicate **is\_base\_si\_unit(*S*)**

Il predicato **is\_si\_unit** è vero quando *S* è un simbolo che denota un'unità base S.I.

### Predicate **si\_unit\_name(*S, N*)**

Il predicato **si\_unit\_name** è vero quando *N* è il nome dell'unità il cui simbolo è *S*.

### Predicate **si\_unit\_symbol(*N, S*)**

Il predicato **si\_unit\_symbol** è vero quando *N* è il nome dell'unità il cui simbolo è *S*.

### Predicate **si\_unit\_base\_expansion(*S, Expansion*)**

Il predicato **si\_unit\_base\_expansion** è vero quando *Expansion* è l'espansione *in forma canonica* dell'unità *S*; l'espansione deve contenere solo unità base.

### Predicate **is\_dimension(*D*)**

Il predicato **is\_dimension** è vero quando *D* è una “dimensione”.

### Predicate **is\_quantity(*Q*)**

Il predicato **is\_quantity** è vero quando *Q* è una “quantità”.

### Predicate **compare\_units(*Result, U1, U2*)**

Il predicato **compare\_units** è vero quando *Result* è uno dei simboli <, >, o =; ovvero *Result* rappresenta la relazione d'ordine tra le unità *U1* e *U2*.

### **Predicate norm(Dim, NewDim)**

Il predicato **norm** è vero quando *ResultNewDim* è la forma canonica della dimensione *D*.

**NB.** *NewDim* deve essere un’unità, un’unità elevata ad un dato esponente, oppure una moltiplicazione dei suddetti; i termini che in Prolog rappresentano delle moltiplicazioni nelle forme canoniche in questione *devono* essere *associativi a sinistra*. Infatti:

```
?- a * b * c = a * (b * c).
false

?- a * (b * c) = a * (b * c).
true

?- a * b * c = a * b * c.
true

?- (a * b) * c = a * b * c.
true
```

### **Predicate qadd(Q1, Q2, QR)**

Il predicato **qadd** è vero quando *QR* è il risultato (in forma canonica) della somma delle quantità *Q1* e *Q2*. La somma è valida solo se le due quantità hanno dimensioni compatibili.

### **Predicate qsub(Q1, Q2, QR)**

Il predicato **qsubtract** è vero quando *QR* è il risultato (in forma canonica) della sottrazione della quantità *Q2* da *Q1*. La sottrazione è valida solo se le due quantità hanno dimensioni compatibili.

### **Predicate qmul(Q1, Q2, QR)**

Il predicato **qmul** è vero quando *QR* è il risultato (in forma canonica) della moltiplicazione delle quantità *Q1* e *Q2*.

### **Predicate qdiv(Q1, Q2, QR)**

Il predicato **qdiv** è vero quando *QR* è il risultato (in forma canonica) della moltiplicazione delle quantità *Q1* e *Q2*.

### **Predicate qexp(Q, N, QR)**

Il predicato **qexp** è vero quando *QR* è il risultato (in forma canonica) dell’elevamento alla potenza *N* delle quantità *Q*.

**Common Lisp.** Le funzioni che dovete implementare (oltre a quelle descritte sopra) servono a ispezionare le strutture dati e a fare calcoli simbolici con “quantità”.

Si noti che in Common Lisp sarà necessario costruire anche delle funzioni che servono ad estrarre parti delle varie strutture dati che rappresentano unità e quantità. In Prolog possiamo usare l’unificazione per ottenere questo risultato, in Common Lisp no<sup>1</sup>.

### **Function is-si-unit S → Boolean**

La funzione **is-si-unit** ritorna T quando *S* è un simbolo che denota un’unità S.I. (base o derivata). La funzione ritorna NIL in caso contrario.

### **Function is-base-si-unit S → Boolean**

La funzione **is-base-si-unit** ritorna T quando *S* è un simbolo che denota un’unità S.I. base. La funzione ritorna NIL in caso contrario.

<sup>1</sup>A meno di implementare un “unificatore” per CL, ovviamente.

**Function `si-unit-name`  $S \rightarrow N$**

La funzione `si-unit-name` ritorna il nome  $N$  (un simbolo Common Lisp) dell’unità il cui simbolo è  $S$ , o `NIL` se non si trova l’associazione.

**Function `si-unit-symbol`  $N \rightarrow S$**

La funzione `si-unit-symbol` ritorna il simbolo  $S$  (un simbolo Common Lisp) dell’unità il cui nome è  $N$ , o `NIL` se non si trova l’associazione.

**Function `si-unit-base-expansion`  $S \rightarrow Expansion$**

La funzione `si-unit-base-expansion` ritorna l’espansione *in forma canonica* dell’unità  $S$ ; l’espansione deve contenere solo unità base.

**Function `is-dimension`  $D \rightarrow Boolean$**

La funzione `is-dimension` ritorna `T` quando  $D$  è una “dimensione”; in caso contrario ritorna `NIL`.

**Function `is-quantity`  $Q \rightarrow Boolean$**

La funzione `is-quantity` ritorna `T` quando  $Q$  è una “quantità”; in caso contrario ritorna `NIL`.

**Function `q`  $N, D \rightarrow Result$**

Questa funzione (un “costruttore”) ritorna una quantità in forma `(Q N D)`, dove  $D$  è l’argomento  $D$  in forma canonica.

**Function `compare-units`  $U1, U2 \rightarrow Result$**

La funzione `compare-units` ritorna come *Result* uno dei simboli `<`, `>`, o `=`; ovvero *Result* rappresenta la relazione d’ordine tra le unità  $U1$  e  $U2$ .

**Function `norm`  $Dim \rightarrow NewDim$**

La funzione `norm` ritorna come *ResultNewDim* la forma canonica della dimensione  $D$ .

**NB.** In Common Lisp la forma canonica è una lista con operatore `*` e con operandi delle unità o delle espressioni del tipo `(expt U E)`.

**Function `qadd`  $Q1, Q2 \rightarrow QR$**

La funzione `qadd` ritorna il risultato (in forma canonica) della somma delle quantità  $Q1$  e  $Q2$ . La somma è valida solo se le due quantità hanno dimensioni compatibili; in caso contrario la funzione genera un errore (chiamando la funzione `error`).

**Function `qsub`  $Q1, Q2 \rightarrow QR$**

La funzione `qsub` ritorna il risultato (in forma canonica) della sottrazione della quantità  $Q2$  dalla quantità  $Q1$ . La sottrazione è valida solo se le due quantità hanno dimensioni compatibili; in caso contrario la funzione genera un errore (chiamando la funzione `error`).

**Function `qmul`  $Q1, Q2 \rightarrow QR$**

La funzione `qmul` ritorna il risultato (in forma canonica) della moltiplicazione delle quantità  $Q1$  e  $Q2$ .

**Function `qdiv`  $Q1, Q2, QR \rightarrow L$**

a funzione `qdiv` ritorna il risultato (in forma canonica) della divisione delle quantità  $Q1$  e  $Q2$ . Se  $Q2$  ha valore 0, viene segnalato un errore mediante la funzione `error`.

### **Function `qexp` $Q, N \rightarrow QR$**

La funzione `qexp` ritorna il risultato (in forma canonica) dell'elevamento alla potenza  $N$  delle quantità  $Q$ .

## **3 Esempi**

Questi sono alcuni esempi di come si può usare questa libreria.

**NB.** Dovete naturalmente essere preparati a calcolare anche altri esempi.

### **Common Lisp**

```
cl-prompt> (defparameter q1 (q 42 'm)
Q1

cl-prompt> q1
(Q 42 M)

cl-prompt> (defparameter q2 (q 1/2 '(* (expt s 3) (expt m -3))))
Q2

cl-prompt> q2
(Q 1/2 (* (expt M -3) (expt S 3)) ; Notate la forma canonica.

cl-prompt> (qmul q1 q2)
(Q 21 (* (expt M -2) (expt S 3)))

cl-prompt> (qadd q1 q2)
ERROR: Incompatible dimensions M and (* (expt M -3) (expt S 3)).
...
cl-prompt> (si-unit-symbol 'ohm)
OMEGA ; Notate questo caso speciale.

cl-prompt> (si-unit-symbol 'degrecelsius)
DC ; Notate questo altro caso speciale.

cl-prompt> (norm '(* (expt m -2) A (expt s -2) kg (expt m 2) (expt K 2) (expt s -1)))
(* KG (expt S -3) A (expt K 2))
```

Attenzione alle seguenti possibilità.

```
cl-prompt> (qadd (q 2 'm) (q 20 'cm))
(Q 2.2 M) ; Metres and centimetres are ‘‘compatible’’.
```

## Prolog

```
?- qmul(q(42, m), q(0.5, (s ** 3) * (m ** -3)), R).
R = q(21, (m ** -2) * (s ** 3)).

?- qadd(q(42, m), q(0.5, (s ** 3) * (m ** -3)), R).
false

?- si_unit_symbol(ohm, S).
S = 'Omega' ; Notare questo caso speciale.

?- si_unit_symbol(degreecelsius, S).
S = dc ; Notare questo altro caso speciale.

?- norm((m ** -2) * 'A' * (s ** -2) * kg * (m ** 2) * ('K' ** 2) * (s ** -1), ND).
ND = kg * (s ** -3) * 'A' * ('K' ** 2)
```

Attenzione alle possibilità seguenti.

```
?- qadd(q(2, m), q(20, cm), R).
R = q(2.2, m) ; Metres and centimetres are ‘‘compatible’’.
```

## 4 Conclusioni

La libreria di funzioni che avrete costruito è un primo passo verso la costruzione di un sistema di *Computer Algebra* quali Mathematica<sup>TM</sup> Maxima, Axiom etc.

La rappresentazione di quantità non è necessariamente la migliore e sono molte le variazioni sul tema; lo scopo di questa rappresentazione è di coniugare semplicità e flessibilità, oltre ad essere facile da manipolare<sup>2</sup>. Qualora si vogliano fare operazioni più sofisticate con varie quantità, potrebbe valer la pena di implementare una diversa rappresentazione sia in Prolog che in Common Lisp.

---

<sup>2</sup>Specie per il correttore.

## 5 Da consegnare...

LEGGERE ATTENTAMENTE LE ISTRUZIONI QUI SOTTO  
(IN ITALIANO!).

PRIMA DI CONSEGNARE, CONTROLLATE **ACCURATAMENTE**  
CHE TUTTO SIA NEL FORMATO E CON LA STRUTTURA DI CARTELLE  
RICHIESTI.

RIPETIAMO! RILEGGETE BENE TUTTO IL TESTO (e le istruzioni di consegna).

Dovete consegnare:

Uno **.zip** file dal nome <Cognome>\_<Nome>\_<matricola>\_SIUNITS\_LP\_202507.zip che conterrà una cartella dal nome <Cognome>\_<Nome>\_<matricola>\_SIUNITS\_LP\_202507.

Se il vostro nome e cognome sono: Gian Giacomo Pier Carl Luca Serbelloni Lupmann Vien Dal Mare, il nome del file sarà:

Serbelloni\_Lupmann\_Vien\_Dal\_Mare\_Gian\_Giacomo\_Pier\_Carl\_Luca\_123456\_SIUNITS\_LP\_202507.zip.

Inoltre...

- Nella cartella dovete avere un file **Group.txt** e due sottocartelle: una di nome **Lisp** e l'altra di nome **Prolog**. Il file **Group.txt** deve contenere, uno per riga, i nomi e le matricole dei componenti del gruppo. *Anche se il gruppo è di una persona.*
- Nella directory **Lisp** dovete avere:

– un file dal nome **siunits.lisp** che contiene il codice di della libreria.

- \* Le prime linee del file **devono essere dei commenti con il seguente formato**, ovvero devono fornire le necessarie informazioni secondo le regole sulla collaborazione pubblicate su Moodle.

```
;;;, <Matricola> <Cognome> <Nome>  
;;;, <eventuali collaborazioni>
```

Il contenuto del file deve essere ben commentato.

– Un file **README** in cui si spiega come si possono usare le funzioni definite nel programma.

- Nella directory **Prolog** dovete avere:

– un file dal nome **siunits.pl** che contiene il codice di della libreria.

- \* Le prime linee del file **devono essere dei commenti con il seguente formato**, ovvero devono fornire le necessarie informazioni secondo le regole sulla collaborazione pubblicate su Moodle.

```
%%% <Matricola> <Cognome> <Nome>  
%%% <eventuali collaborazioni>
```

Il contenuto del file deve essere ben commentato.

– Un file **README** (si! Anche qui anche se è una ripetizione) in cui si spiega come si possono usare i predicati definiti nel programma.

**ATTENZIONE!** Consegname solo dei files e directories con nomi costruiti come spiegato. Niente spazi extra e soprattutto niente **.rar** or **.7z** o **.tgz** – solo **.zip**!  
Repetita juvant! **NON CONSEGNARE FILES .rar!!!!**

**PRIMA DI CONSEGNARE...** Ricontrollate il contenuto del vostro file **.zip**. Al suo interno ci deve essere UN SOLO ELEMENTO, ovvero la cartella che contiene i vostri elaborati.

**Esempio:**

File **.zip**:

**Antoniotti\_Marco\_424242\_SIUNITS\_LP\_202507.zip**

Che contiene:

```
prompt$ unzip -l Antoniotti_Marco_424242_SIUNITS_LP_202507.zip
Archive: Antoniotti_Marco_424242_SIUNITS_LP_202507.zip
      Length      Date    Time     Name
-----  -----
          0 12-02-25 09:59  Antoniotti_Marco_424242_SIUNITS_LP_202507/
        80 12-04-25 09:55  Antoniotti_Marco_424242_SIUNITS_LP_202507/Group.txt
          0 12-04-25 09:55  Antoniotti_Marco_424242_SIUNITS_LP_202507/Lisp/
      4783 12-04-25 09:51  Antoniotti_Marco_424242_SIUNITS_LP_202507/Lisp/siunits.lisp
    10598 12-04-25 09:53  Antoniotti_Marco_424242_SIUNITS_LP_202507/Lisp/README.txt
          0 12-04-25 09:55  Antoniotti_Marco_424242_SIUNITS_LP_202507/Prolog/
      4623 12-04-25 09:51  Antoniotti_Marco_424242_SIUNITS_LP_202507/Prolog/siunits.pl
    10622 12-04-25 09:53  Antoniotti_Marco_424242_SIUNITS_LP_202507/Prolog/README.txt
-----  -----
      30626                           7 files
```

## 5.1 Valutazione e scadenza

Il programma sarà valutato sulla base di una serie di test standard. In particolare si valuterà la copertura e correttezza delle operazione di base sulle varie quantità.

La consegna del progetto è fissata per il giorno 27 luglio 2025 entro le 23:55 GMT+1.