anno 2014-15
Introduzione all'Algoritmica per i Licei

2 - Programmazione imperativa: i concetti base.

Elio Giovannetti
Dipartimento di Informatica
Università di Torino

versione 1 marzo 2015

Programmazione imperativa: concetti fondamentali.

1)

- variabile;
- assegnazione;
- stringhe;
- liste (vettori), dizionari;
- input-output elementare;

2)

- sequenza di istruzioni;
- controllo del flusso di esecuzione (istruzioni composte):
  - istruzioni if e if-else;
  - istruzione while;
  - istruzione for

La programmazione imperativa

È un modo di concepire e scrivere i programmi che corrisponde abbastanza al modo in cui funziona lo hardware della macchina:

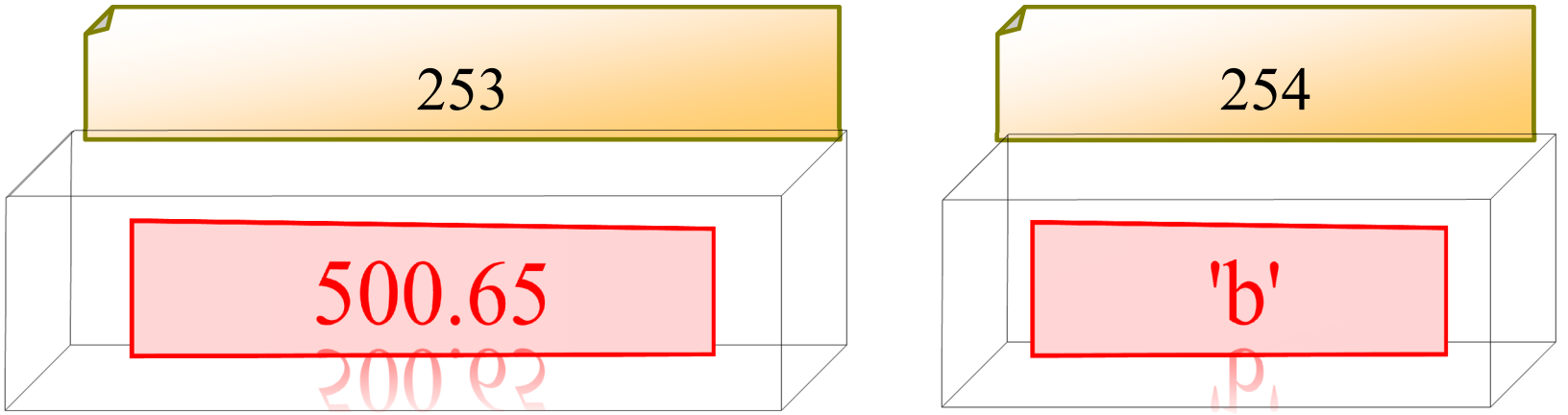
- Un programma è costituto da una sequenza di istruzioni o comandi (per questo si chiama imperativa !).
- La macchina esegue il programma eseguendo un'istruzione dopo l'altra (alcune istruzioni hanno l'effetto di far eseguire un'istruzione successiva piuttosto che un'altra, oppure di far ripetere una sequenza di istruzioni, ecc.).
- La macchina ha uno stato interno, che è costituito dai contenuti delle celle di memoria, o *variabili* (e da qual è la prossima istruzione da eseguire).
- Un' istruzione può modificare tale stato.
- Istruzioni di input/output permettono di comunicare con l'esterno, per ricevere i dati e fornire i risultati.
- Il paradigma imperativo non è l'unico !

Concetti fondamentali: variabili e assegnazione.

variabile (o cella di memoria)

può essere pensata come un contenitore o scatola:

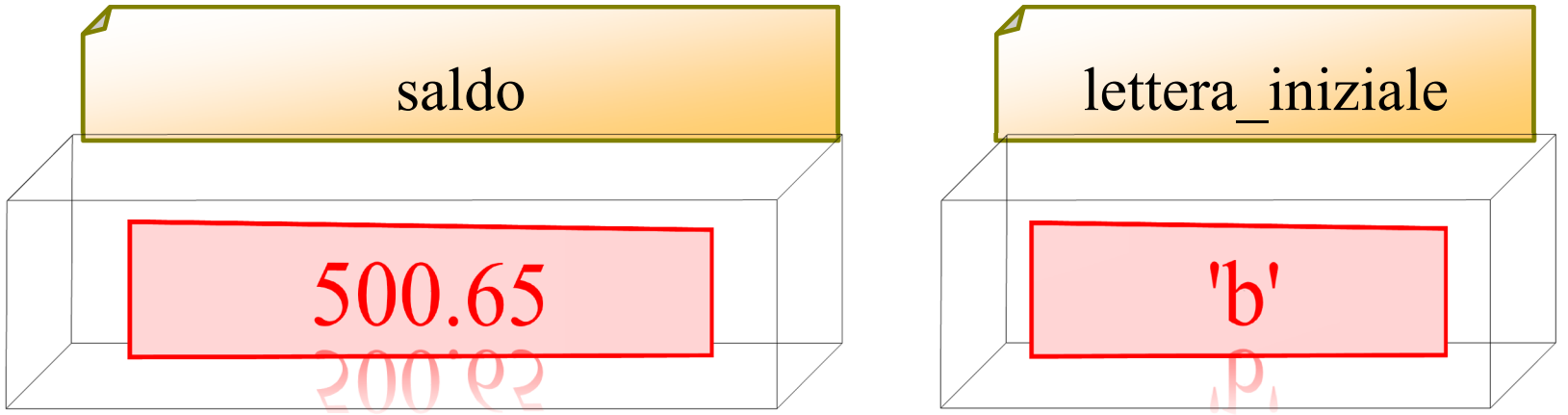
- dotata di un'etichetta esterna che è il suo indirizzo o nome;
- contenente un valore, che è un ente astratto, come un numero, o un carattere, ecc.



Concetti fondamentali: variabili e assegnazione.

variabile (o cella di memoria)

In un programma Java, C, Python, ecc.,
il nome della variabile non è il suo indirizzo numerico, bensì un nome simbolico dichiarato dal programmatore (o, come si vedrà in seguito, un'espressione simbolica di altro genere).
La traduzione di tali nomi in indirizzi è fatta all'atto dell'esecuzione, senza intervento del programmatore.



Identificatori.

I nomi di variabile nei linguaggi di programmazione devono obbedire a certe regole generali, lievemente diverse da linguaggio a linguaggio.

I nomi (di variabile, o, come vedremo, di funzione, ecc.) in un linguaggio di programmazione si chiamano più correttamente identificatori. Le regole più importanti, comuni a tutti i linguaggi, sono che un identificatore

- non può iniziare con una cifra (ma può contenere cifre);
- non può contenere spazi bianchi, segni -, +, ecc.
- non può coincidere con una delle parole che, come vedremo, hanno un significato predefinito nel linguaggio (*if, while*, ecc.).

Esempio: 44gatti, crea-lista, crea lista non sono identificatori permessi (in realtà crea lista sono due distinti identificatori).

gatti44, creaLista, crea_lista sono invece permessi.

Maiuscole e minuscole

Python, come tutti i linguaggi moderni, distingue le minuscole dalle maiuscole. Pertanto gli identificatori:

`totalespese, Totalespese, totaleSpese, TOTalespeSE`

sono interpretati come identificatori tutti fra loro distinti.

Anche `totale_spese` e `totalespese` sono ovviamente identificatori distinti.

In Python sono ammissibili negli identificatori anche le lettere accentate, che noi talora useremo per ragioni didattiche, anche se non è una buona pratica scrivere programmi professionali che contengano caratteri speciali.

Ovviamente `eta, età, etá, etÀ`, ecc.
sono identificatori tutti fra loro diversi.

Regole di stile.

Le regole di stile non sono regole "obbligatorie", nel senso che un programma che le viola ma è sintatticamente corretto può essere eseguito. Tuttavia è importante seguire alcune regole:

- gli identificatori tutti maiuscoli sono considerati pessimo stile, eccetto che per le costanti;
- gli identificatori devono suggerire il significato ad essi associato: occorre trovare un compromesso fra significatività e lunghezza del nome;
- gli indici e i contatori si indicano di solito con le lettere i, j, k;
- per avere identificatori composti di più parole si può seguire una delle due convenzioni più diffuse:
  - ogni parola successiva inizia con maiuscola: **`totaleSpese`**
  - le parole sono separate dal carattere _ : **`totale_spese`**

oppure, qualche volta, … scrivere semplicemente **`totalespese`** !

Concetti fondamentali: variabili e assegnazione.

ASSEGNAZIONE (assignment)

è un'istruzione (cioè un comando all'esecutore) che permette di cambiare il contenuto di una cella di memoria:

Nel vecchio Pascal o nel nuovissimo Grace si scrive così:

```
saldo := saldoIniziale
```

copia il contenuto della cella *saldoIniziale* nella cella *saldo*, cancellando il precedente contenuto di *saldo*;

```
saldo := saldo - prelievo
```

calcola la differenza fra il contenuto della cella *saldo* e quello della cella *prelievo*, e rimette il risultato nella cella *saldo*.

Ma in C, C++, Java, Python, come in quasi tutti i linguaggi di adesso, l'assegnazione è indicata dal simbolo di uguaglianza, pur NON essendo l'uguaglianza !

Concetti fondamentali: variabili e assegnazione.

ASSEGNAZIONE (assignment)

è un'istruzione (cioè un comando all'esecutore) che permette di cambiare il contenuto di una cella di memoria:

In Python si scrive così:

```
saldo = saldoIniziale
```

copia il contenuto della cella *saldoIniziale* nella cella *saldo*, cancellando il precedente contenuto di *saldo*;

```
saldo = saldo - prelievo
```

calcola la differenza fra il contenuto della cella *saldo* e quello della cella *prelievo*, e rimette il risultato nella cella *saldo*.

In C, C++, Java, Python, come in quasi tutti i linguaggi di adesso, l'assegnazione è indicata dal simbolo di uguaglianza, pur NON essendo l'uguaglianza !

Concetti fondamentali: variabili e assegnazione.

NOTA BENE

il simbolo "=" NON indica l'uguaglianza matematica!

la scrittura

a = b

NON è un'espressione affermante che a e b sono uguali;
è un comando o istruzione che copia in a il contenuto di b.
Un simbolo più conveniente per l'assegnazione sarebbe:

a ⟸ b

(l'informazione fluisce da destra verso sinistra)

Subito dopo l'esecuzione dell'istruzione i contenuti di a e di b sono uguali, ma poi possono di nuovo variare indipendentemente l'uno dall'altro.

L'assegnazione non è l'uguaglianza.

La scelta sintattica operata molti anni fa per motivi pratici nel linguaggio C, e poi impostasi quasi universalmente, può creare parecchie difficoltà nei principianti:

il simbolo "**=**", graficamente simmetrico, che in matematica denota la relazione di uguaglianza, che è una relaz. simmetrica, viene usato per indicare un'azione che non è simmetrica!

È difficile, all'inizio, resistere al "subconscio matematico" in cui il simbolo "**=**" denota qualcosa di simmetrico!

E l'uguaglianza allora come si indica?

Con un doppio carattere "=":

==

Ad es., per stabilire se il contenuto di a è uguale a quello di b al fine di eseguire un'istruzione oppure un'altra, si scrive

```
if a == b:
```

Esempio:

```
if a == b: print("a e b sono uguali")
else: print("a e b non sono uguali")
```

Scrivere

```
if a = b:
```

è un errore, appunto perché `a = b` non è un'uguaglianza!

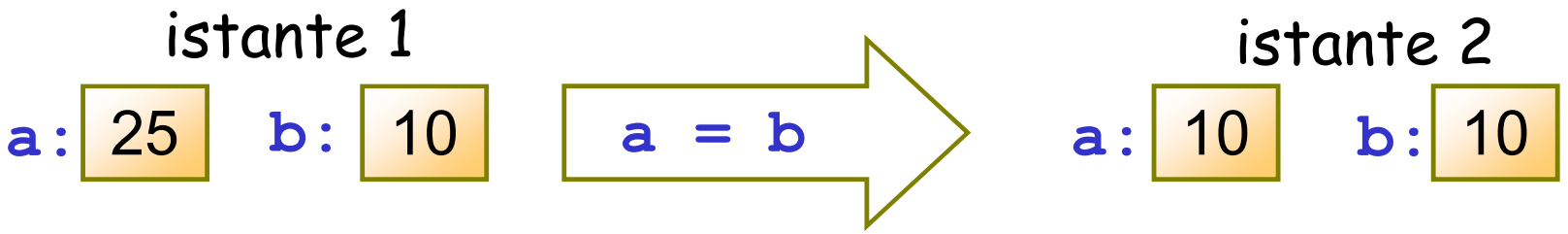
Capita a tutti di farlo, soprattutto all'inizio, ma Python lo segnala ed è immediato correggerlo.

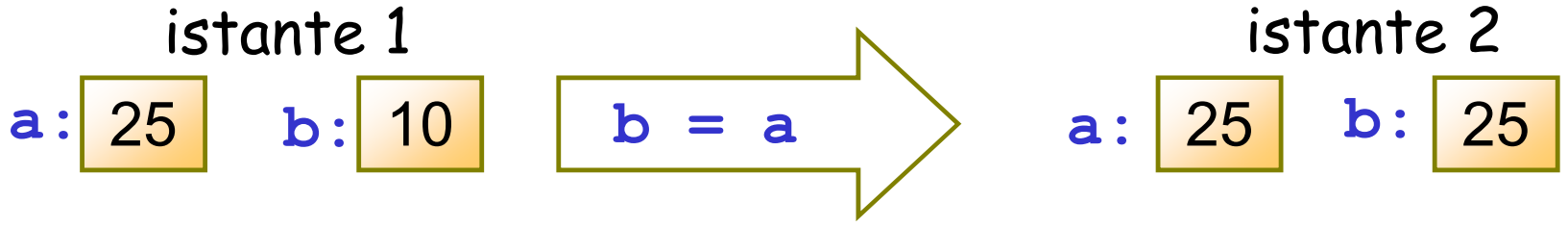
Ricordate: soprattutto in programmazione, sbagliando s'impara!

Variabili e assegnazione.

Dunque l'operazione di assegnazione, a differenza dell'uguaglianza matematica, NON è simmetrica!

`a = b` NON è la stessa cosa che `b = a`





Nota Bene

Nell'istante immediatamente successivo all'esecuzione della istruzione `a = b` i contenuti di `a` e di `b` sono uguali, quindi l'uguaglianza `a == b` è vera.

Ma `a` e `b` non sono diventati sinonimi! Se uno dei due viene successivamente modificato, l'altro rimane invariato.

L'assegnazione non è simmetrica (cont.)

`a = 3` è un'operazione perfettamente legale, che mette il valore 3 nel contenitore a, cancellando il valore precedente

`3 = a` è una scrittura priva di senso, perché 3 non è un contenitore! non si può mettere qualcosa nel numero 3!

`a = b+c` è un'operazione perfettamente legale, che mette in a la somma dei valori contenuti in b e in c

`b+c = a` è una scrittura priva di senso, perché l'*espressione* b+c non denota un contenitore, bensì un numero: la somma dei contenuti di b e c (si possono sommare i contenuti, NON i contenitori)

Ancora sull'asimmetria dell'operatore "="

Osserva, dagli esempi precedenti, che i nomi di variabile vengono interpretati in modo diverso a seconda che siano a sinistra o a destra del simbolo di assegnazione:

- a sinistra del simbolo "=" il nome di una variabile indica proprio il contenitore che ha quel nome;
- a destra del simbolo "=" il nome di una variabile indica il contenuto di quella variabile.

```
a = b
```

vuol dire:

copia il contenuto di b nel contenitore a
(cancellandone il contenuto precedente)

Non vuol dire che a e b diventano lo stesso contenitore!
Non vuol dire che a e b diventano sinonimi!

Assegnazione e uguaglianza

Non confondere

```
a = b
```

che è una istruzione, la cui esecuzione cambia lo stato della macchina, cioè la sua memoria

con

```
a == b
```

che è una espressione, che non viene "eseguita" bensì valutata, e il suo valore è un valore booleano: True o False.

Scrivere `a == b` in una riga al posto di un'istruzione non ha alcun effetto: durante l'esecuzione l'uguaglianza viene valutata, ma il suo valore (True o False) non viene utilizzato.

Invece, come abbiamo già visto, scrivere `if a = b` oppure `while a = b` è un errore, perché `a = b` è un'istruzione, non un'espressione booleana.

Terminologia

In informatica una sequenza di caratteri viene comunemente detta stringa (è un inglesismo, dall'inglese string).

I francesi e gli spagnoli, più inclini alla protezione delle loro lingue, dicono rispettivamente "chaîne" e "cadena".

D'altra parte, i francesi e gli spagnoli chiamano rispettivamente "la souris" e "el ratón" quello che in italiano si chiama "il mouse"!

Stringhe e nomi

In Python, come in tutti i linguaggi di programmazione, per indicare una stringa all'interno di un programma non si può scrivere semplicemente la stringa, perché non si distinguerebbe dai nomi di variabile e da tutte le altre sequenze di caratteri che compongono il programma.

Per scrivere una stringa in un programma occorre racchiuderla fra virgolette doppie o fra virgolette semplici. Esempio.

```
nome = "Adele"
```

`print("nome")` visualizza sullo schermo la parola *nome*
`print(nome)` visualizza sullo schermo la parola *Adele*

L'uso assomiglia a quello della lingua naturale, quando si vuole menzionare o parlare di un'espressione linguistica:

Adele ha 100 lettere dal fidanzato, ma "Adele" ha 5 lettere;
la frase "l'informatica è roba da nerds" è un luogo comune; ecc.

Stringhe in Python e nomi.

L'istruzione

```
print(print)
```

scrive sullo schermo la riga seguente:

```
<built-in function print>
```

che vuol dire che *print* è una funzione pre-definita.

L'istruzione

```
print('print')
```

oppure

```
print("print")
```

scrive sullo schermo: `print`.

L'istruzione

```
print("l'apostrofo")
```

scrive sullo schermo: `l'apostrofo`

L'istruzione

```
print('la parola "print" ha 5 lettere')
```

scrive sullo schermo: `la parola "print" ha 5 lettere`

Stringhe in Python: il + come operatore di concatenazione.

```
a = 'Buon'
b = 'giorno'
print(a + b)
```

scrive sullo schermo `Buongiorno`

Invece, ovviamente:

```
m = 25
n = 10
print(m + n)
```

scrive sullo schermo `35`

Cosa succede se si esegue

```
print(a + m)
```

?

Si genera un errore!

Funzioni: invocare funzioni predefinite.

In Python (come in quasi tutti i linguaggi di programmazione), una funzione è una sequenza di istruzioni, generalmente dotata di un nome, che serve a eseguire un compito specifico.

Più avanti impareremo a definire nuove funzioni; ma in Python, come in ogni linguaggio, vi sono molte funzioni predefinite, e intanto impariamo come si usa una funzione.

Per usare (o invocare) una funzione bisogna scrivere:

- il nome della funzione, ad es. print, oppure input, o int;
- i dati che servono alla funzione per eseguire il suo compito, detti argomenti, racchiusi dentro un'unica coppia di parentesi tonde, e separati l'uno dall'altro mediante virgole; il numero di argomenti dipende dalla funzione.

Esempio. La funzione print prende uno o più argomenti ed esegue il compito di visualizzarli sullo schermo: print(a, 4, 'gatti')

Valutare gli argomenti

Come in matematica, gli argomenti di una funzione non devono essere necessariamente dei valori: possono essere espressioni, che Python valuta prima di passarli alla funzione.

Esempio. Come abbiamo già visto:

`print(25 + 10)` scrive sullo schermo `35`

`print("pip" + "po")` scrive sullo schermo `pippo`

eccetera.

Funzioni che danno un risultato.

Molte funzioni, oltre ad eseguire un compito, danno un risultato, che viene passato al programma.

Esempi.

La funzione sqrt (abbreviazione di square root, radice quadrata) può essere usata come in matematica:

```
r2 = sqrt(2)
```

Il significato è, in realtà, lievemente diverso:

- in un testo matematico una scrittura come la precedente afferma che, da quel punto in poi, il nome r2 è un sinonimo del numero reale che è la radice quadrata di 2;
- in un programma una tale scrittura è un'istruzione eseguendo la quale la funzione sqrt dà come risultato il valore 1.4142… e tale valore viene, per mezzo dell'operatore di assegnazione, depositato nella cella di nome r2.

Restituire … ciò che non si è preso in prestito.

Nella terminologia informatica inglese "dare un risultato" si dice "to return a result", cioè, letteralmente, "restituire un risultato" o anche, con un abbastanza diffuso inglesismo, "ritornare un risultato".

Diremo quindi che la funzione sqrt, invocata con argomento 121, restituisce il numero 11.0, ecc.

Metaforicamente, invocare una procedura è come chiamare un aiutante perché esegua un certo compito; l'aiutante se ne va, esegue il compito (mentre chi l'ha chiamato rimane fermo in attesa, o si addormenta) e poi ritorna al chiamante portandogli il risultato.

A quel punto il chiamante si sveglia e riprende il proprio lavoro.

Funzioni che non restituiscono nessun risultato.

Vi sono anche funzioni che eseguono un compito ma non restituiscono alcun risultato.

Esempio.

La funzione print scrive sullo schermo gli argomenti passatigli, ma non dà alcun risultato.

Si potrebbe definire una funzione ordina che prende come argomento una lista di nomi e la modifica ordinandola in ordine alfabetico.

In questi casi, in realtà, in Python la funzione restituisce un valore speciale che si chiama None (cioè Nessuno, in inglese).

Esempio:

```
risult = print(3+2)
```
→ scrive sullo schermo 5

```
print(risult)
```
→ scrive sullo schermo None.

I vettori o liste.

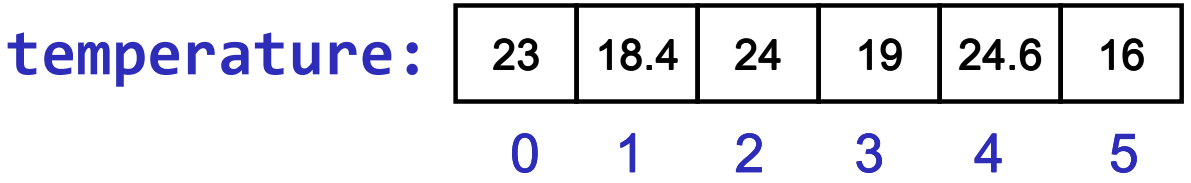
Un vettore (o array) è una sequenza di variabili (cioè di celle di memoria) contigue ognuna delle quali è accessibile tramite un indice in un tempo costante indipendente dalla posizione.

I vettori sono realizzati in modi lievemente diversi, e con sintassi lievemente diverse, nei diversi linguaggi.

In Python si chiamano "liste" e gli elementi si scrivono fra parentesi quadre separate da virgole. Esempio:

```
temperature = [23, 18.4, 24, 19, 24.6, 16]
```

In Python, come nella maggior parte dei linguaggi moderni, l'indicizzazione comincia da 0 e non da 1.



`print(temperature[2])` scrive sullo schermo 24

Vettori (liste).

I contenuti delle celle che compongono un vettore possono venire modificati, come qualunque variabile. Esempio:

```
temperature = [23, 18.4, 24, 19, 24.6, 16]
...
temperature[2] = 17
```

`print(temperature)` scrive sullo schermo:

```
[23, 18.4, 17, 19, 24.6, 16]
```

Un vettore può metaforicamente vedersi come un indirizzo postale che possiede dei "numeri di interno": nell'esempio l'indirizzo generale è `temperature`, i numeri interi da 0 a 5 sono i sei "numeri di interno" di `temperature`.

La primitiva range

La primitiva `range` rappresenta una sequenza ordinata di numeri interi. Esempi:

`range(5)` rappresenta la sequenza `0,1,2,3,4`

`range(1, 5)` rappresenta la sequenza `1,2,3,4`

`range(7, 13)` rappresenta la sequenza `7,8,9,10,11,12`

`range(0, 10, 2)` rappresenta la sequenza `0,2,4,6,8`

`range(0, 9, 2)` rappresenta la stessa sequenza `0,2,4,6,8`

`range(3, 50, 10)` rappresenta la sequenza `3,13,23,33,43`

`range(5, 0, -1)` rappresenta la sequenza `5,4,3,2,1`

`range(5, 1, -1)` rappresenta la sequenza `5,4,3,2`

`range(50, 30, -5)` rappresenta la sequenza `50,45,40,35`

`range(50, 33, -5)` rappresenta la stessa sequenza

ecc.

Range e liste in Python 3

Nelle ultime versioni di Python un range non è direttamente una lista, bensì una sua rappresentazione astratta e compatta (di fatto, è una memorizzazione dei suoi argomenti).

Per generare esplicitamente la lista corrispondente si deve invocare la funzione `list`. Esempio:

`list(range(4,8))` dà come risultato `[4,5,6,7]`

Input-output elementare in Python

In tutti i linguaggi di programmazione per far prendere al programma una stringa (cioè una sequenza di caratteri) digitata sulla tastiera occorre eseguire un procedimento che sospende l'esecuzione del programma e rimane in attesa che l'utente digiti la stringa voluta, seguita dal tasto INVIO.

A quel punto il procedimento passa la stringa al programma, che riprende l'esecuzione.

In Python vi è a tale scopo la funzione `input(…)`: essa scrive sullo schermo la stringa passata come argomento, poi aspetta che l'utente immetta una stringa da tastiera, e infine dà come risultato la stringa immessa.

In Python come in tutti i linguaggi moderni vi sono poi molti altri modi per prendere un input da tastiera, attraverso interfacce grafiche; ma per ora vediamo il modo elementare.

Input-output elementare in Python: esempio.

Consideriamo il mini-programma seguente:

```
nome = input('Come ti chiami?')
print('ciao ' + nome + '!')
```

L'esecuzione della prima riga inizia con l'esecuzione del secondo membro dell'assegnazione: l'esecuzione della funzione **`input`**.

Essa scrive sullo schermo la frase "Come ti chiami?", poi rimane in attesa che l'utente digiti una stringa e pigi INVIO.

A quel punto la funzione dà come risultato al programma la stringa immessa, e viene eseguita l'assegnazione: la stringa viene memorizzata nella cella **`nome`**.

L'istruzione successiva scrive sullo schermo la stringa "ciao " seguita dalla stringa contenuta nella cella **`nome`**, seguita dal punto esclamativo.

Dizionari

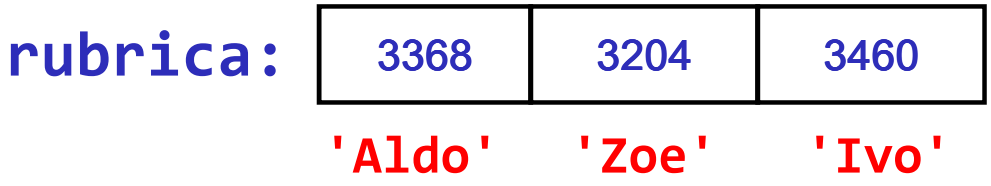
In Python un dizionario è una struttura simile ad una lista in cui però gli indici non sono gli interi consecutivi a partire da 0, ma possono essere delle entità di qualunque tipo, ad es. interi non consecutivi, numeri negativi, stringhe, ecc, purché tutte distinte. Tali "indici generalizzati" vengono chiamati chiavi.

Gli elementi di un dizionario si scrivono fra parentesi graffe, preceduti ognuno dalla propria chiave, separata dai due punti.

Esempio: una rubrica telefonica aziendale (anche disordinata):

```
rubrica = {'Aldo':3368, 'Zoe':3204, 'Ivo':3460}
```

può essere pensata come la struttura qui sotto:



L'istruzione `print(rubrica['Zoe'])` scrive sullo schermo 3204.

Dizionari

Il contenuto di una casella di un dizionario può essere cambiato esattamente come quello di una "casella" di una lista.

Ad es., se Zoe cambia ufficio e quindi numero di interno, potremo scrivere:

```
rubrica['Zoe'] = 3142
```

`print(rubrica)` scrive allora sullo schermo:

```
{'Aldo':3368, 'Zoe':3142, 'Ivo':3460}
```

Nota che mentre le chiavi, come gli indici di una lista, devono essere tutte distinte, caselle diverse possono avere, come in una lista, contenuti uguali. Ad es. se a Zoe viene assegnata la stessa scrivania e lo stesso telefono di Ivo, avremo:

```
rubrica = {'Aldo':3368, 'Zoe':3460, 'Ivo':3460}
```