Laboratorio di Informatica

Terza lezione a Python

Dottore Paolo Parisen Toldin - parisent@cs.unibo.it
Dottoressa Sara Zuppiroli - sara.zuppiroli@unibo.it

Ricorsione

- La ricorsione primitiva è una operazione definita sulle funzioni in questo modo:

$$h(x_1, \ldots, x_n, y) = \begin{cases} f(x_1, \ldots, x_n) & \text{se } y = 0 \\ g(x_1, \ldots, x_n, h(x_1, \ldots, x_n, y-1)) & \text{altrimenti} \end{cases}$$

- Quindi, “ricorsione” vuol dire che la funzione richiama sé stessa ed è definita a partire da uno o più “casi base” e poi da uno o più casi che richiamano la funzione con parametri più “semplici”.

- La definizione della funzione risulterà più leggibile e più semplice da trattare!

Ricorsione

- La somma, ad esempio, può essere definita in modo “ricorsivo”, per a$\geq$0 come

$$somma(a,b) = \begin{cases} b & \text{se } a \text{ è } 0 \\ somma(a-1, b+1) & \text{altrimenti} \end{cases}$$

- Il nostro codice python sarà dunque

```
def somma(a,b):
    if a==0 :
        return b
    else :
        return somma(a-1,b+1)
```

Provatelo, funziona? (ovviamente con numeri positivi)

Ricorsione

- L'elevamento a potenza può essere definito in modo ricorsivo come a^b

$$exp(a,b) = \begin{cases} a & \text{se } b \text{ è } 1 \\ a \cdot exp(a, b-1) & \text{altrimenti} \end{cases}$$

- Il nostro codice python sarà dunque

```
def esponenziale(a,b):
    if b==1 :
        return a
    else :
        return a*exponenziale(a,b-1)
```

Provatelo, funziona? (ovviamente con numeri positivi)

Iterazione vs Ricorsione

```
def somma(a,b):
    risultato = b
    for i in range(0,a):
        risultato = risultato + 1
    return risultato
```

```
def somma(a,b):
    if a==0 :
        return b
    else :
        return somma(a-1,b+1)
```

- A sinistra la somma iterata, a destra la somma ricorsiva
- A sinistra abbiamo una istruzione (risultato=risultato+1) che viene eseguita un numero di volte pari al valore di a.
- A destra abbiamo una funzione (somma(a,b)) che viene richiamata più volte, passando, via via, parametri diversi
- Con la ricorsione calcoliamo il risultato in modo “top-down”, cioè generiamo tutte le chiamate ricorsive di funzione che servono per calcolare il risultato
- Con l'iterazione, partiamo dal valore base e costruiamo passo dopo passo il valore finale. Visione “bottom-up”.

Tipi di dato

- definizione: Un tipo di dato è una collezione di valori omogenei ed effettivamente presentati, dotato di un insieme di operazioni che manipolano tali valori

- omogenei: i valori condividono alcune proprietà strutturali, es. i numeri intere tra 3 e 10
- operazioni: a fianco all'insieme dei valori, abbiamo le operazioni che agiscono su tali valori
- effettivamente presentati: i valori che possono essere rappresentati in modo finito. Il tipo real o float non rappresenta tutti i numeri reali, ma solo quelli ottenibili da un algoritmo.

Variabile

- la variabile che può assumere tutti i valori numerici di un insieme prefissato (tipo)
- la variabile è una specie di contenitore che può contenere valori di tipo omogeneo.
- Questi valori possono cambiare nel tempo, e in python possono cambiare anche i tipi omogenei che sono contenuti in essa.

la tipizzazione in python

- Abbiamo visto come in python non dobbiamo esplicitare il tipo di dato di ogni singola variabile. Questo perchè python segue quella che si chiama “tipizzazione a papera” (duck typing). Si tratta di tipizzazione dinamica.

  “Se nuota come un'anatra e starnazza come un'anatra, io chiamo quell'uccello anatra”

  se una variabile la usiamo come numero intero, le associamo numeri interi, allora quella variabile denoterà un numero intero!

  nb: la tipizzazione dinamica è la politica di tipizzazione, ovvero di assegnazione di tipi alle variabili, in cui il controllo del tipo della variabile e effettuato a run-time piuttosto che in fase di compilazione.

Tipo di dato in python

- Tipi di dato in PythonI tipi di dato sono i seguenti:

- Semplici:
  - interi (int), operazioni tra interi
  - interi lunghi (long), operazioni tra interi
  - numeri in virgola mobile (float), operazioni tra float
  - numeri complessi (complex), operazioni tra complessi
  - valori booleani (bool), operazioni tra booleani
- Sequenze:
  - Immutabili:
    - stringhe (str), operazioni su stringhe
    - tuple (tuple), operazioni su tuple
    - insiemi immutabili (frozenset) operazioni su insiemi immutabili
  - Mutabili:
    - liste (list), operazioni su liste
    - dizionari (dict), operazioni su dizionari
    - insiemi (set), operazioni su insiemi
    - file (file), operazioni su file

Tipi di dato in python

- Sebbene non sia necessario denotare esplicitamente il tipo di dato in python, finora abbiamo visto:
  - Interi
  - Reali
  - Valori booleani
  - Stringhe
  - Liste
  - Tuple (o chiamate anche coppie)

Tipo interi, float e complessi

- Rappresentazione interi; x=3
- Rappresentazione float; x=3.0
- Rappresentazione complessi; x=(3+2j)

- Operazioni associate

Esempio	Descrizione
M+M	Somma
M-M	Sottrazione
M*M	Prodotto
M/M	Divisione, con risultato intero o reale a seconda degli operandi
M%M	Resto della divisione intera
M**M	Elevamento a potenza

Tipo booleano

- Rappresentazione booleani; x=True; x=False

- Operazioni associate:

Operazione	Descrizione
M and N	congiunzione
M or N	disgiunzione
M==N	uguaglianza
M!=N M<>N	disuguaglianza
not M	negazione

Tipo tupla

- Una tupla è in realtà una collezione arbitraria di valori che possono essere trattati come una unità. Una tupla è per molti versi simile ad una lista, ma è immutabile.

- In Python le tuple si rappresentano semplicemente come liste di valori separati da virgola racchiuse fra parentesi tonde, cosí:

```
>>> unaTupla = (1,3,5)
>>> print unaTupla[1]    # si usano gli indici come per le liste
3
>> unaTupla[2] = 7       # errore, non si puo` modificare un elemento di una tupla
Traceback (innermost last):
  File "<pyshell >", line 1, in ?
    unaTupla[2] = 7
TypeError: object doesn't support item assignment
```

Tipi di dato in python

- Possiamo comunque sempre forzare il tipo di dato di una variabile ad essere di un determinato tipo. Ad esempio possiamo chiedere che una variabile intera sia vista come un numero reale, o che un numero sia visto come una stringa o una stringa come un numero.

  - float(variabile) = se può, il risultato è un numero reale
  - int(variabile) = se può, il risultato è un numero intero
  - str(variabile) = se può, il risultato è una stringa

Nota come queste “conversioni” possono generare errori a tempo di esecuzione. Ad esempio, non tutte le stringhe possono essere convertite in numeri!!

Operazioni su liste

mialista = [54,2,65,7,32,12,3]

- Possiamo ordinare facilmente una lista con:

  >>mialista.sort()

  >>mialista

  >>[2, 3, 7, 12, 32, 54, 65]

- Quanto lunga è la lista?

  >>len(mialista)

  7

- E se voglio aggiungere un elemento?

  >>mialista.append(“pippo”)

  >>mialista

  [2, 3, 7, 12, 32, 54, 65, 'pippo']

OPPURE, posso usare l'operatore

mialista + [“pippo”]

Otteniamo lo stesso risultato. Provate!

Operazioni su liste

- E se volessi prendere una sottolista?

```
>>mialista[2:7]
>>[7,12,32,54,65]
>>mialista[:7]
>>[2,3,7,12,32,54,65]
>>mialista[2:]
>>[7,12,32,54,65,'pippo']
```

Primo indice = indice primo elemento
Secondo indice = primo elemento scartato

Il due punti serve a dire
“dall'inizio”

Il due punti serve a dire
“fino alla fine”

Operazioni su liste

- Cosa succede se ho una lista di liste? Ad esempio, una lista di nomi e cognomi.

  ```
  >> mia_lista = [["mario","rossi"],["andrea","bianchi"],["eleonora","arrighi"],["giovanna","malpighi"]]
  >> mia_lista[2]
  ["andrea","bianchi"]
  >> mia_lista[2][1]
  'arrighi'
  ```

  Essendo una lista di liste, la prima volta che richiedo l'elemento [1] (secondo elemento della lista), ottengo una nuova lista! Dunque, posso ancora applicare l'operatore “[ ]” e scegliere un nuovo valore all'interno della lista.

Provate ad accedere a tutti i campi
di questa lista di liste.

Esercizio

- Proviamo ad implementare la successione di fibonacci! Sappiamo che la successione di fibonacci è definita nel seguente modo:

$$fib(n) = \begin{cases} 1 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ fib(n-1) + fib(n-2) & \text{altrimenti} \end{cases}$$

- Quindi:

$fib(2) = fib(1) + fib(0) = 1 + 1 = 2$
$fib(3) = fib(2) + fib(1) = 3$
$fib(4) = 5$
$\ldots$

Numero aureo

- Il rapporto tra due numeri di fibonacci consecutivi tende al numero aureo $\phi = 1,61803398875$

$$\phi = \lim_{n \to \text{inf}} \frac{fib(n)}{fib(n-1)} = \lim_{n \to \text{inf}} \frac{fib(n-1) + fib(n-2)}{fib(n-1)} =$$

$$1 + \lim_{n \to \text{inf}} \frac{fib(n-2)}{fib(n-1)} \Rightarrow \phi = 1 + \frac{1}{\phi} \Rightarrow \phi^2 - \phi - 1 = 0$$

Ed otteniamo il numero aureo!

$$\phi = \frac{1 + \sqrt{5}}{2}$$

Esercizio

- Implementare la successione di fibonacci utilizzando la formula di Binet

$$fib(n) = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

E' una soluzione ricorsiva, iterativa o nessuna delle due?

Esercizio

- Implementare la successione di fibonacci usando un metodo iterativo.
  - Questo vuol dire che non potrete usare né la formula di Binet, né potrete richiamare in modo ricorsivo la funzione.
  - Il suggerimento è quello di tenere traccia, tramite due variabili i due valori precedenti e poi iterativamente calcolare il valore successivo aggiornando i precedenti.

Esercizio

- Scrivere un programma che stampi un menù con quattro voci; ogni voce da la possibilità di calcolare la successione di fibonacci in modo diverso:
  - Ricorsione
  - Iterazione
  - Formula di Binet

(chiaramente, il menù deve anche avere la possibilità di chiudere il programma)

Esercizio

- Implementare un programma che, richiesto un numero intero positivo in input da tastiera n, stampa in output il valore fib(n).
  - Implementare il programma senza utilizzare la ricorsione ma usando il costrutto while

    suggerimento: ad ogni ciclo tenete traccia dei valori precedentemente calcolati ed aggiornateli ad ogni iterazione.

Esercizio

- Basandosi sull'esercizio precedente, scrivere un programma che chiede in input un valore numerico n > 0. Dopodichè, stampa tutti i valori da fib(0) a fib(n).
  - Implementare la soluzione richiamando le tre funzioni appena viste
  - Confrontare i tempi di risposta delle tre soluzioni per $n \in \{1, 10, 20, 30, 40, 50, 80, 100\}$

Complessità

- *L'informatica*, intesa come *scienza*, si occupa anche di studiare la complessità dei programmi.
  - Complessità può essere inteso come sinonimo di “difficoltà” nell'eseguire il dato programma/algoritmo.
  - Essa studia principalmente le *risorse tempo* (numero di istruzioni eseguite?) e *spazio* (quanta memoria richiede?)
  - Il calcolo è fatto rispetto ai valori in input

Esempio

- Conderiamo il seguente programma

```
import random
def swap(index_a,index_b, myarr):
	#la funzione scambia due valori
	temp=myarr[index_a]
	myarr[index_a] = myarr[index_b]
	myarr[index_b] = myarr[index_b]
	return myarr

#definisco un array di valori casuali
my_array=[]
for i in range(0,miavar)
	my_array = my_array + [int(round(random.random()*100))]



miavar= input(“inserisci valore”)

for i in range(0,miavar)
	for j in range (0,miavar)
			my_array = swap(i,j,my_array)
```

Facciamo i conti: quanti passi vengono eseguiti?

Il programma viene dato in allegato nella cartella comune

Esempio

- Apriamo la soluzione dell'esercizio dell'ordinamento e controlliamo assieme la complessità dell'algoritmo.

  Sorprendentemente, “quel” programma ha complessità sia in *spazio* che in *tempo* costante!

- Ma se misuriamo la sua complessità rispetto alla grandezza dell'array, troveremo la stessa complessità del programma precedente!
- La differenza è che il programma precedente non fa niente di utile, mentre quest'ultimo ordina dei numeri.

nb: è stato dimostrato che ordinare una serie di n numeri non può mai richiedere meno di $n \log n$ passi di calcolo!

Complessità

- Non esiste una sola soluzione per un dato problema. Anzi! È dimostrato che esistono *infinite* soluzioni per ogni problema.
  - Ovvero, ci possono essere infiniti programmi diversi che potreste scrivere per risolvere un determinato problema
  - Quale scegliere? Ovviamente quello che ha una “complessità” minore.

Esercizio

- Creare un programma che tramite un piccolo menù vi dia la possibilità di calcolare i primi n numeri della successione di fibonacci.

  (nb: il menù avra solo due voci: “calcola fibonacci” ed “esci”)

- Estensione facoltativa: il programma deve anche dirvi quali numeri sono primi o meno.

  Ricordo che per vedere se un numero dispari n è primo, è sufficiente vedere se è divisibile o meno per tutti i valori fino a $\sqrt{n}$

Esercizio

- Scrivere un programma che:
  - Chieda in input nome e cognome (due input separati)
  - Salvi la coppia nome e cognome, concatenandolo ad una lista di nomi e cognomi già definita all'inizio: [[Marco, Rossi],[Giovanna, Bianchi],[Lucia, Verde]]
  - Si preveda che il programma continui a chiedere in input nome e cognome fintantochè viene inserito “STOP”.
  - A questo punto il programma stampa la lista di tutti i nomi e cognomi inseriti.

Esercizio

- Create un programma per la gestione di candidati. Il programma deve tenere traccia di:
  - Nome
  - Cognome
  - Età
  - Elenco certificati presentati (nb: trattasi di una lista ad-hoc)

  Implementare la possibilità di inserire nuovi candidati, cancellarli, cercarli e visualizzarli tutti.

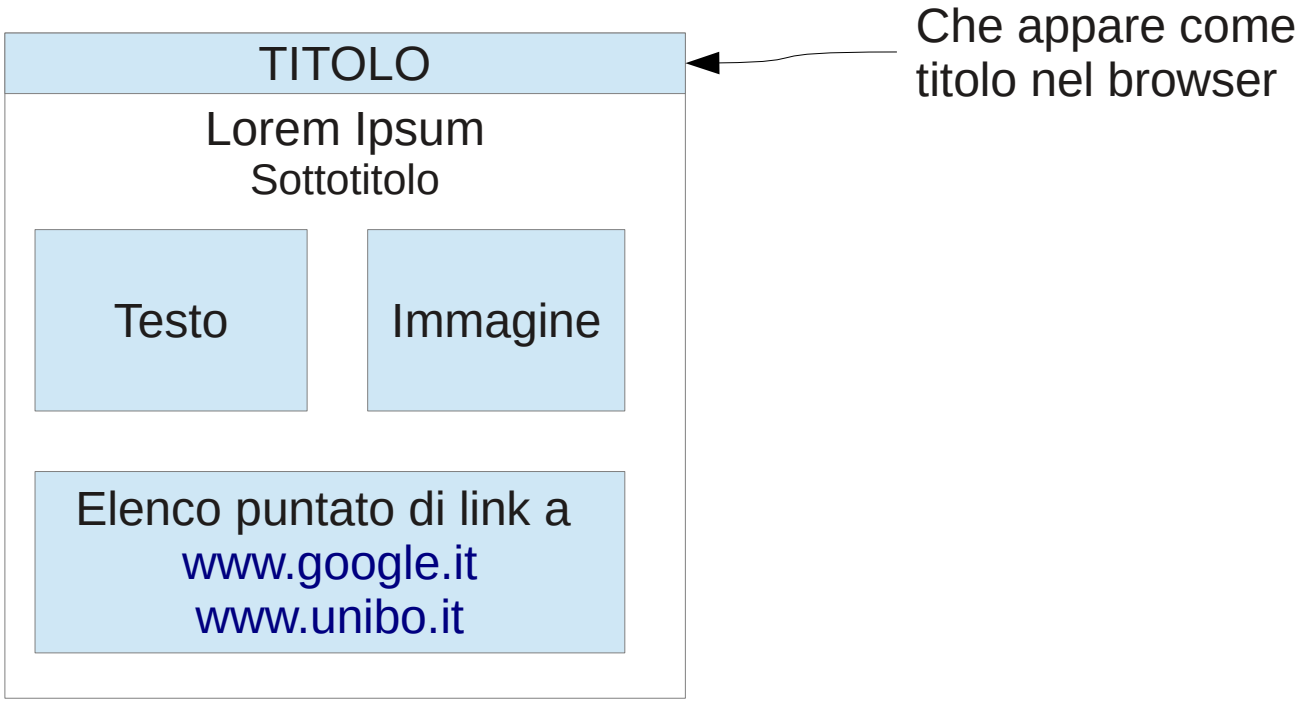
nb: in questo esercizio dovrete gestire una lista all'interno di un'altra lista, a sua volta ancora dentro un'altra!!

Esercizio

- Estendere l'esercizio precedente con:
  - Ricerca per titoli. Il programma deve ritornare il nome e cognome dei canditati con tali titoli
  - Il risultato dev'essere ordinato in modo lessicografico.
  - Aggiungere la possibilità di poter inserire nuovi titoli/certificati

Esercizio

- Scrivere una pagina HTML che appaia come in figura:



- Riempite la vostra pagina con del testo “lorem Ipsum” o con del testo a vostra scelta. http://it.lipsum.com/

Esercizio

- Scrivere un funzione che preso in input xmi restituisca il valore della funzione fattoriale.

  f(x) =x!

  - Risolvetelo usando la ricorsione e usando l'iterazione. Quale delle due risulta avere complessità minore?

Esercizio

- Presa una lista L e una variabile x in input, definire una funzione che restituisca il numero delle occorrenze di x in L

Esercizio

- Scrivere una funzione che mi calcoli il valore medio in una lista di k elementi.

Esercizio

- Scrivere una funzione che mi restituisca il primo valore presente nella lista che minimizzi la distanza al valore medio in una lista di k elementi.

  es. L=[1,5,9]

  media = 7.5

  d(1,7.5)= |1-7.5| = 6.5

  d(5, 7.5)= |5-7.5| = 2.5

  d(9, 7.5)= |9-7.5| = 2.5

  elemento trovato = 5

-

Esercizio

- Scrivere una funzione che prese due liste in input mi restituisca la lista con gli elementi intersezione delle due liste in input.

  Es. L_1 = [1,2,3,4,5]

  L_2 = [1,2,7]

  L_intersezione = [1,2]

Esercizio

- Scrivere una funzione che ritorni la lista differenza tra due liste L1, L2.

  Es. L_1 [1,2,3,4,5]

  L_2 [1,2,7]

  L_D [3,4,5]

Esercizio

- Implementare la funzione di ackermann

$$A(m,n) = \begin{cases} n+1 & \text{se } m = 0 \\ A(m-1,1) & \text{se } m > 0 \text{ e } n = 0 \\ A(m-1, A(m,n-1)) & \text{se } m > 0 \text{ e } n > 0 \end{cases}$$

- Nota bene: la funzione decisamente cresce molto velocemente. Provate a ragionare su un pezzo di carta: quante chiamate ricorsive vengono fatte?

A(0,0) = 1
A(0,1) = 2
A(2,1) = 5
A(3,5) = 253
A(4,5) > $2^{2^{65533}}$ = python finisce la memoria disponibile