Sottoprogrammi
Funzioni

Problema

- Abbiamo visto sequenze di istruzioni che risolvono particolari “sottoproblemi”: controllo della primalità di un numero, confronto di stringhe, etc.
- È possibile riutilizzare queste sequenze?
- È possibile consentire ad altri di riutilizzare queste sequenze?
- È possibile dare a queste sequenze un nome che ne indichi la funzionalità?
- È possibile “svincolare” lo sviluppo di soluzioni a sottoproblemi di questo tipo dallo sviluppo di una soluzione “complessa”?

Se fosse possibile…

…fare queste cose, i vantaggi sarebbero molti:

- Potendo riutilizzare facilmente le sequenze, aumenterebbe la produttività (potremmo scrivere più codice in meno tempo)
- Potremmo consentire ad altri di aumentare la loro produttività
- Potremmo rendere il codice più leggibile
- Potremmo “spezzare” la risoluzione di un problema in sottoproblemi

Le funzioni

- Python offre, in risposta a queste esigenze, lo strumento delle *funzioni*
- Una funzione è una sequenza di comandi che:
  - ha un nome
  - può essere *invocata* (cioè può esserne richiesta l'*esecuzione*)
  - può ricevere dei *parametri* che ne influenzano l'esecuzione
  - produce un valore *risultato*

Motivazioni

- ***Modularità*** nello sviluppo del codice
  - Affrontare il problema per ***raffinamenti successivi***
- ***Riusabilità***
  - Scrivere una sola volta il codice e usarlo più volte
  - Esempio: un algoritmo di ordinamento
- ***Astrazione***
  - Esprimere in modo sintetico operazioni complesse
  - Definire operazioni specifiche dei tipi di dato definiti dal programmatore
    - Esempio: calcolo "*totale* + *iva*" di un ordine
    - **punti, segmenti, poligoni, numeri complessi**

Sulla **necessità** dei sottoprogrammi

I sottoprogrammi consentono di ***scomporre*** problemi complessi in moduli più semplici, sfruttabili poi, anche singolarmente, per la risoluzione di ***problemi diversi***.
Se strutturati nel modo corretto, ***nascondono*** al resto del programma dettagli implementativi che non è necessario che esso conosca; in tal modo rendono ***più chiaro*** il programma nel suo complesso, e assai ***più facile*** la sua manutenzione [… o la sua *scrittura corretta*! NdA].

B. W. Kernighan
D. M. Ritchie

Definizione di funzioni

La sintassi per definire funzioni è molto semplice. Ad esempio possiamo definire una funzione che restituisce `True` se un numero è pari o `False` se è dispari:

```
def is_even(n):
 if n%2 == 0:
   return True
 else:
   return False
```

La funzione è introdotta dalla parola chiave `def`, dopo `def` appare il nome della funzione, in questo caso `is_even`; dopo il nome della funzione viene specificata tra parentesi tonde la lista dei parametri accettati dalla funzione; dopo la lista dei parametri ci sono i due punti (`:`) che introducono un blocco di codice indentato; il blocco di codice può contenere diverse istruzioni e 0 o più `return`.

Quando chiamiamo questa funzione possiamo passare un numero qualsiasi che verrà assegnato a n. Il corpo della funzione viene poi eseguito e, a seconda del valore di n, la funzione restituisce True se il numero è pari o False se è dispari:

```
>>> def is_even(n):
...     # se il resto di n/2 è 0, n è pari
...     if n%2 == 0:
...         return True
...     else:
...         return False
...
>>> is_even(4)
True
>>> is_even(5)
False
>>> is_even(-7)
False
```

È anche possibile documentare una funzione usando una docstring, cioè una stringa (in genere racchiusa tra """..."""") che si trova come prima istruzione all'interno di una funzione:

```
def is_even(n):
    """Return True if n is even, False otherwise."""
    if n%2 == 0:
        return True
    else:
        return False
```

La funzione builtin help() è in grado di estrarre e mostrare questa stringa automaticamente:

```
>>> help(is_even)
Help on function is_even in module __main__:

is_even(n)
    Return True if n is even, False otherwise.
```

Passaggio parametri

```
>>> def say_hello():
...     print('Hello World!')
...
>>> say_hello()
Hello World!
```

In questo primo esempio abbiamo definito una funzione con 0 parametri, che quindi non è in grado di accettare nessun argomento.

Prima di vedere più in dettaglio come definire una funzione, è utile approfondire il passaggio di argomenti. Quando una funzione viene chiamata, è possibile passare 0 o più argomenti. Questi argomenti possono essere passati per posizione o per nome:

```
>>> def calc_rect_area(width, height):
...     """Return the area of the rectangle."""
...     return width * height
...
>>> calc_rect_area(3, 5)
15
>>> calc_rect_area(width=3, height=5)
15
>>> calc_rect_area(height=5, width=3)
15
>>> calc_rect_area(3, height=5)
15
```

Valori di ritorno

- La parola chiave return viene usata per restituire un valore al chiamante, che può assegnarlo a una variabile o utilizzarlo per altre operazioni

```
>>> def square(n):
...     return n**2
...
>>> x = square(5)
>>> x
25
>>> square(square(5))
625
>>> square(3) + square(4) == square(5)
True
```

Una funzione può contenere 0 o più return, e una volta che un return viene eseguito, la funzione termina immediatamente.

Questo vuol dire che solo uno dei return viene eseguito ad ogni chiamata:

```
>>> def abs(n):
...     if n < 0:
...         return -n  # eseguito se n è negativo
...     return n  # eseguito se n è positivo o nullo
...
>>> abs(-5)
5
>>> abs(5)
5
```

return è in genere seguito dal valore di ritorno, ma è anche possibile omettere il valore e usare return per terminare la funzione: in questo caso None viene restituito automaticamente. Se si raggiunge il termine della funzione senza incontrare neanche un return, None viene restituito automaticamente:

```
>>> def print_twice(text):
...     if not text:
...         # termina immediatamente se text è una stringa vuota
...         return
...     print(text)
...     print(text)
...     # restituisce None automaticamente al termine della funzione
...
>>> # stampa 2 volte e restituisce None al termine della funzione
>>> res = print_twice('Python')
Python
Python
>>> print(res)
None
>>> # entra nell'if e restituisce None prima di stampare
>>> res = print_twice('')
>>> print(res)
None
```

Nel caso sia necessario restituire più valori, è possibile fare:

```
>>> def midpoint(x1, y1, x2, y2):
...     """Return the midpoint between (x1; y1) and (x2; y2)."""
...     xm = (x1 + x2) / 2
...     ym = (y1 + y2) / 2
...     return xm, ym
...
>>> x, y = midpoint(2, 4, 8, 12)
>>> x
5.0
>>> y
8.0
```

Il valore restituito è sempre uno: una singola tupla di 2 elementi.
Python supporta un'operazione chiamata unpacking, che ci permette di assegnare contemporaneamente diversi valori a più variabili, permettendo quindi operazioni come la seguente:

```
x, y = midpoint(2, 4, 8, 12)
```

Il primo valore della tupla è associato a x e il secondo a y.

Scope delle variabili

Tutti i parametri e le variabili create all'interno di una funzione possono essere usate solo da codice all'interno della funzione.

```
>>> def calc_circle_area(r):
...     pi = 3.14
...     return pi * r**2
...
>>> calc_circle_area(5)
78.5
>>> r
Traceback (most recent call last): File "<stdin>", line 1, in
<module> NameError: name 'r' is not defined
>>> pi
Traceback (most recent call last): File "<stdin>", line 1, in
<module> NameError: name 'pi' is not defined
```

Esempio

```
n=int(input("Inserire un numero: "))
i=2
while i <= n:
    j=2
    while i%j != 0:
        j=j+1
    if j == i:
        print(i)
    i=i+1
```

- La porzione di codice evidenziata calcola se il valore della variabile `i` è un numero primo
- Il valore su cui 'lavora' è il valore della variabile `i`
- Il suo 'risultato' è il valore della variabile `primo`

Definizione della funzione primo

```
def primo(num):
    for n in range(2,num):
        if num % n == 0:
            return False
    else:
        return True
```

Esempio

```
n=int(input("Inserire un numero: "))
i=2
while i <= n:
    if primo(i):
        print(i)
    i=i+1
```

Sintassi dell'invocazione

- Una funzione può essere vista come un'*espressione*, che può essere *valutata* (es: possiamo immaginare una funzione **`somma(a,b)`** equivalente ad **`a+b`**)
- Quando, nell'esecuzione del codice, è necessario 'valutare' un'espressione-funzione, la funzione viene *invocata*
- La sintassi per la formulazione di una espressione-funzione è semplicemente:

  <nome funzione>(<lista argomenti>),
- dove <lista argomenti> è una lista di espressioni di tipo corrispondente a quelle della definizione

Semantica dell'invocazione

- Quando una funzione viene invocata:
  1. Le espressioni che ne costituiscono gli argomenti vengono valutate, e il valore reso disponibile alla funzione
  2. La funzione viene eseguita, fino a quando non viene incontrato un comando return
  3. Il "valore" dell'espressione-funzione è il valore dell'argomento del return che ha causato la terminazione

Esempio di codice

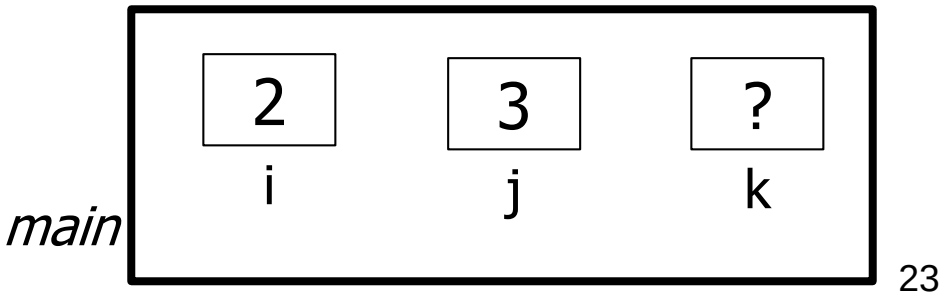
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

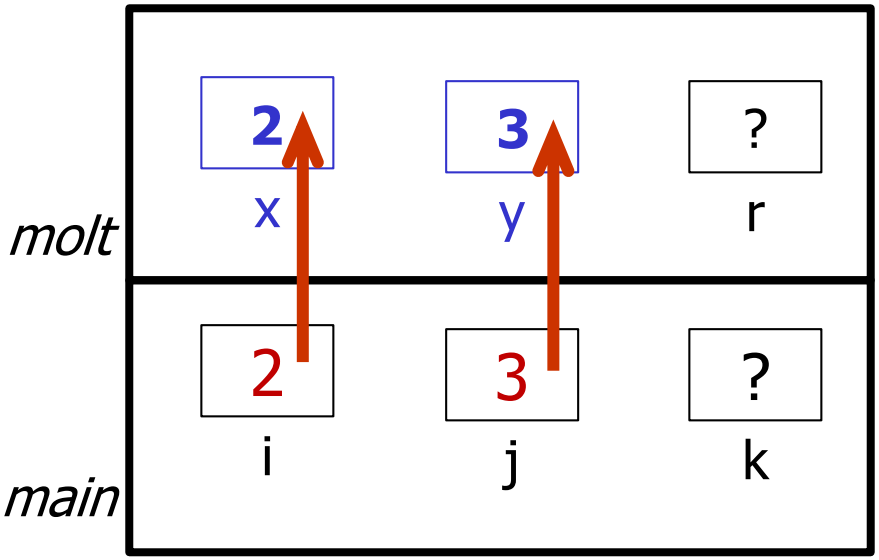
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

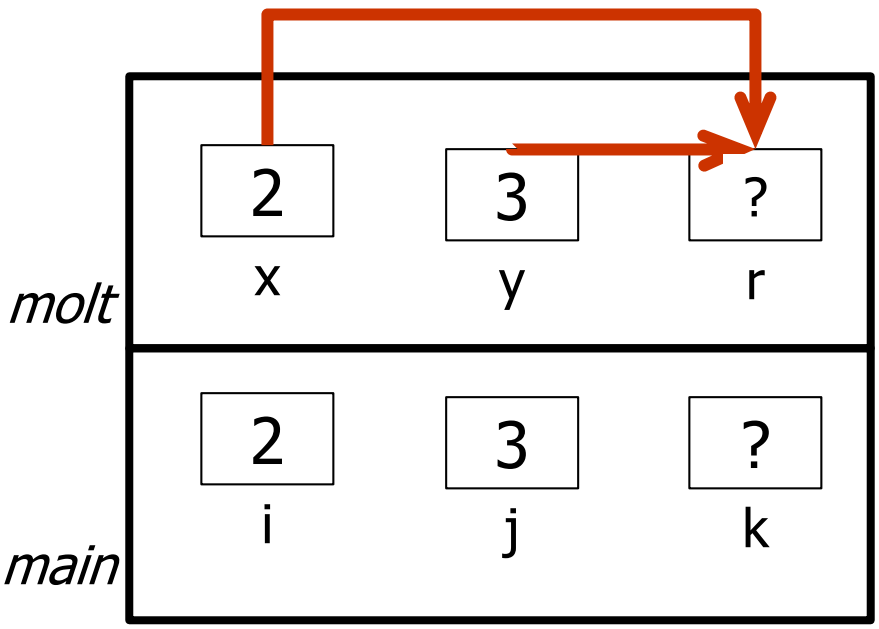
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

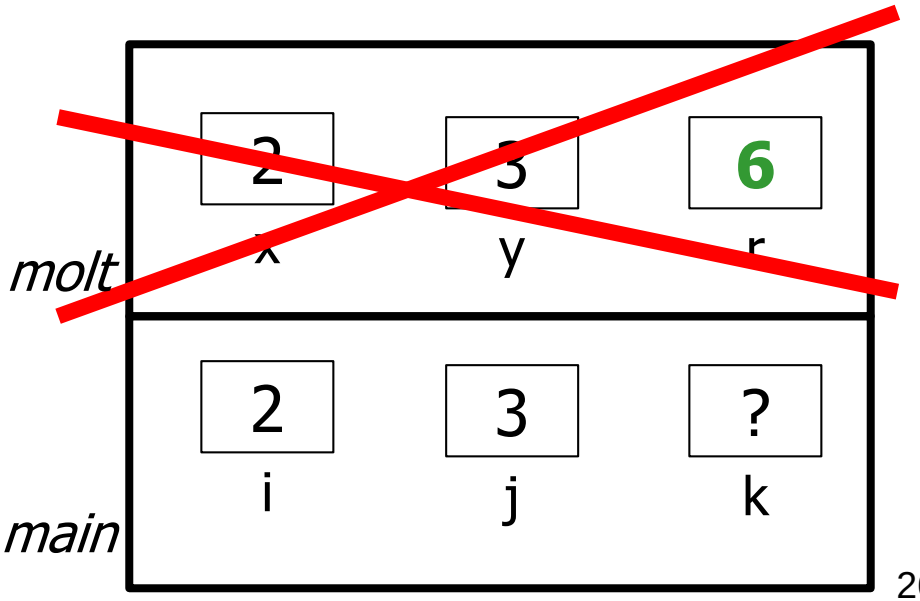
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

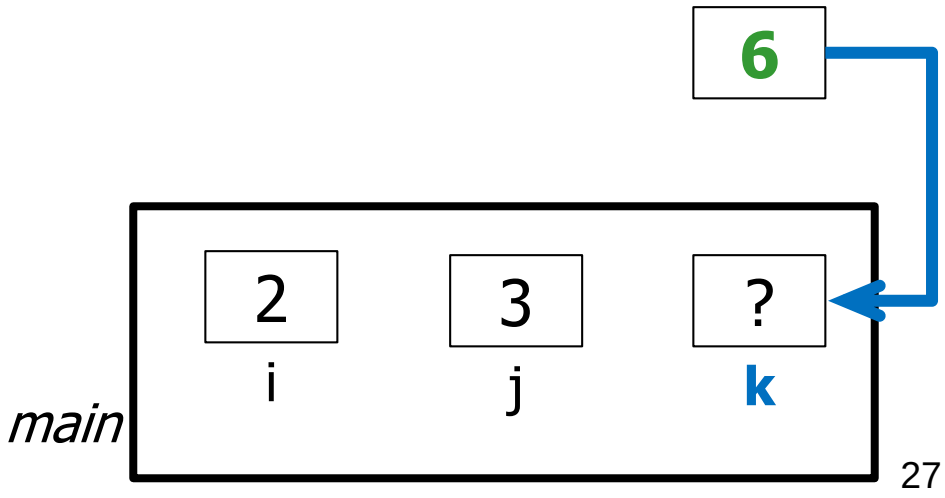
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

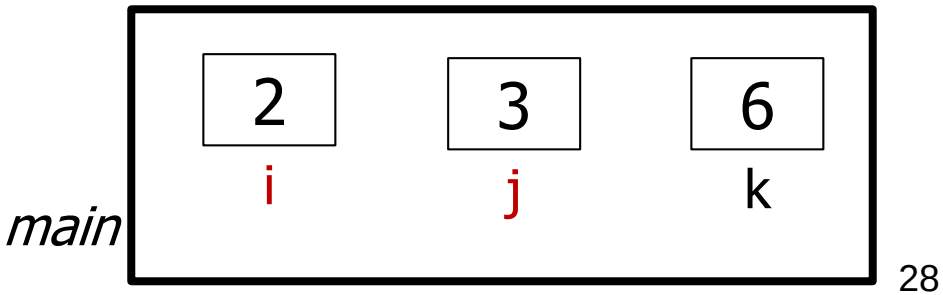
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

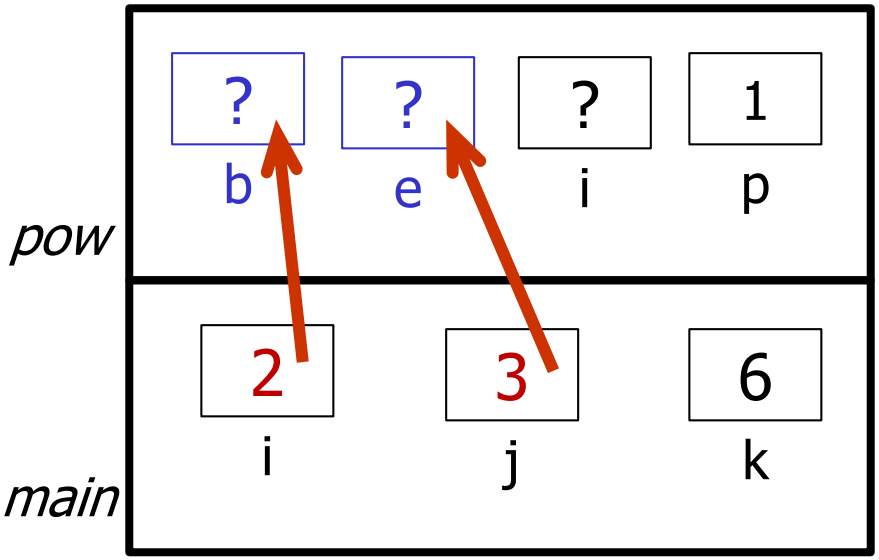
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

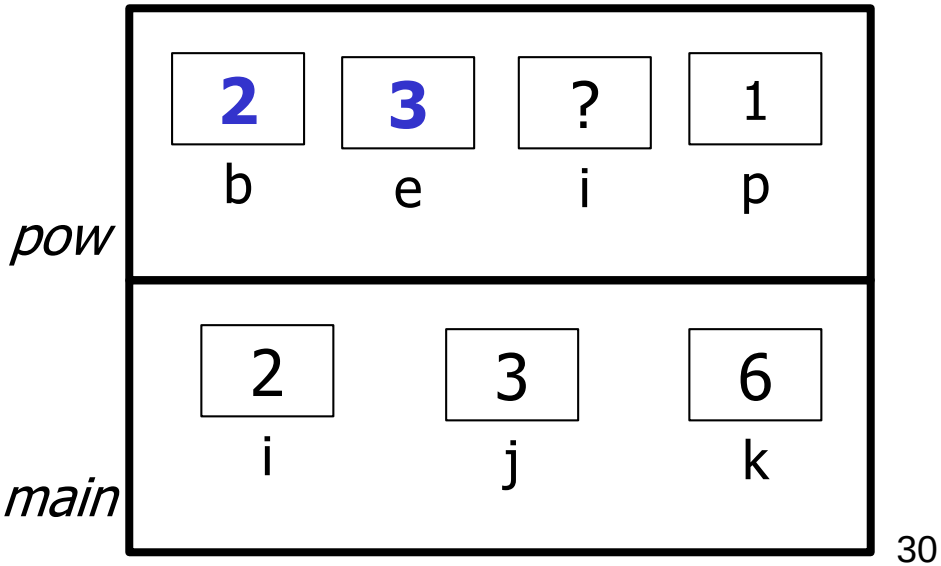
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

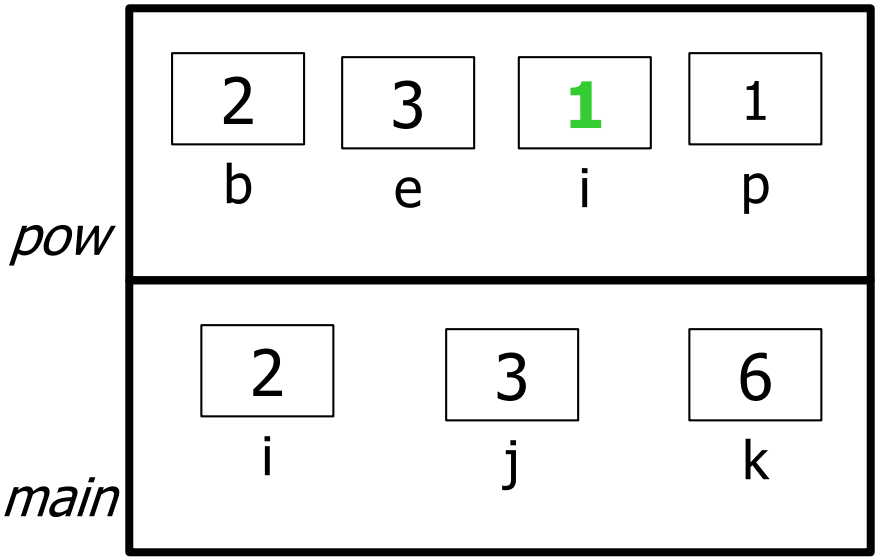
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

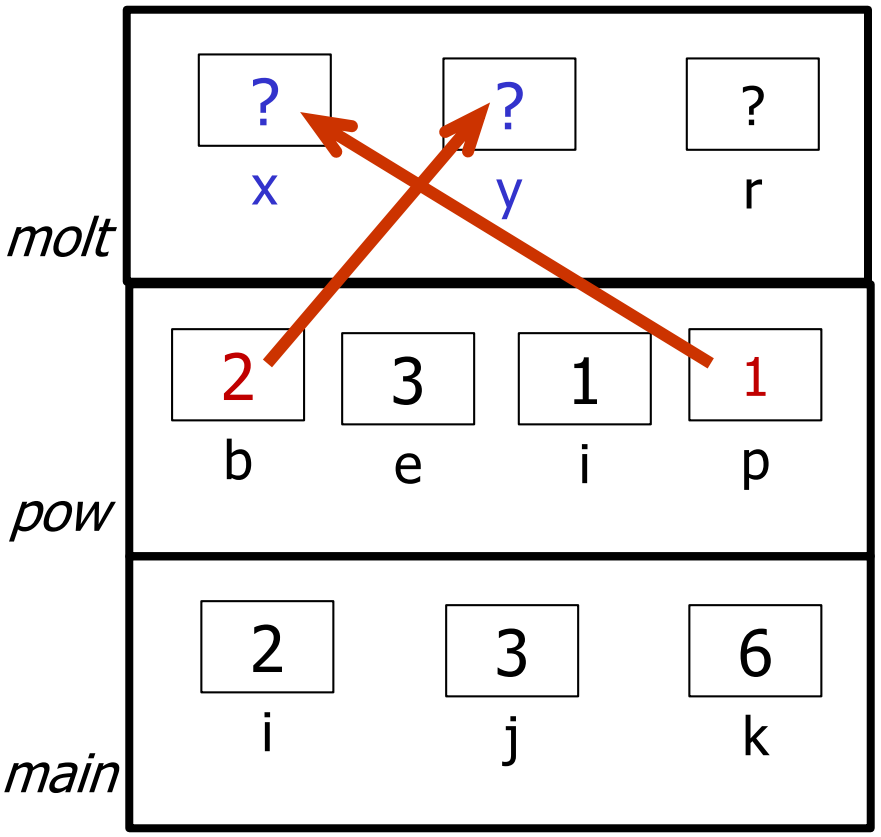
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

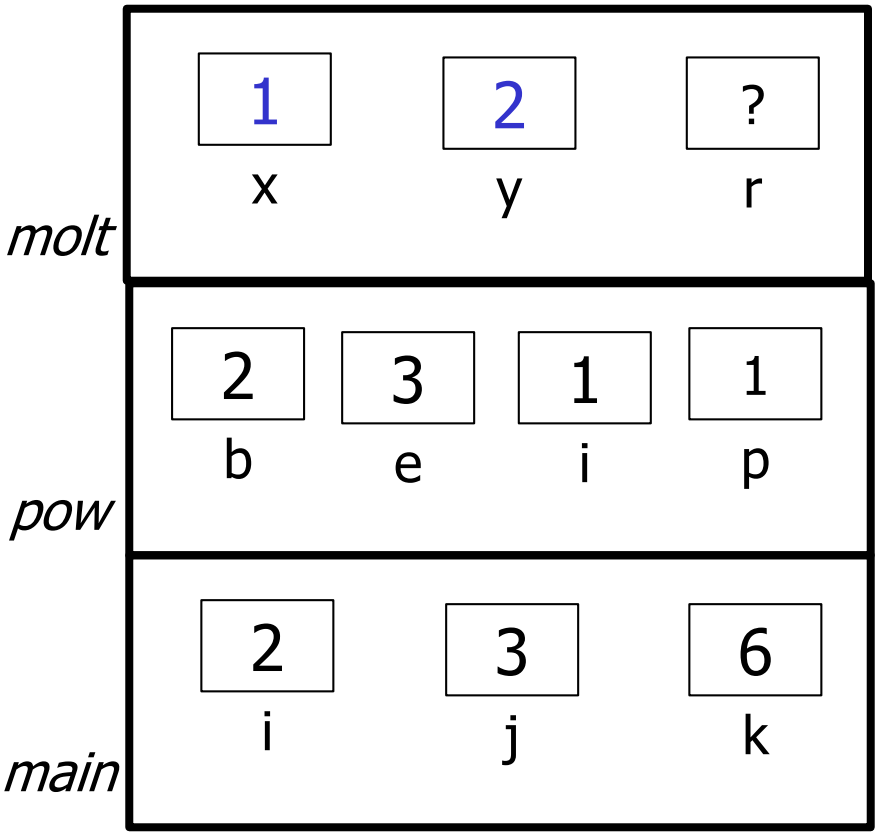
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

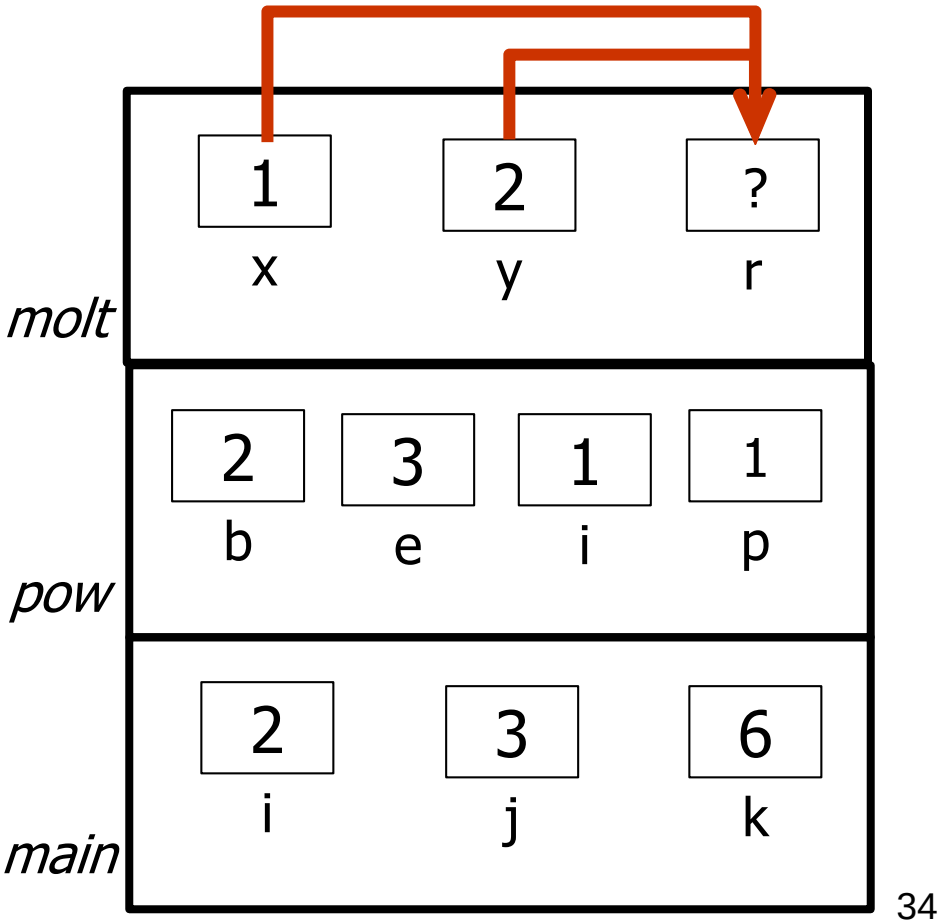
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

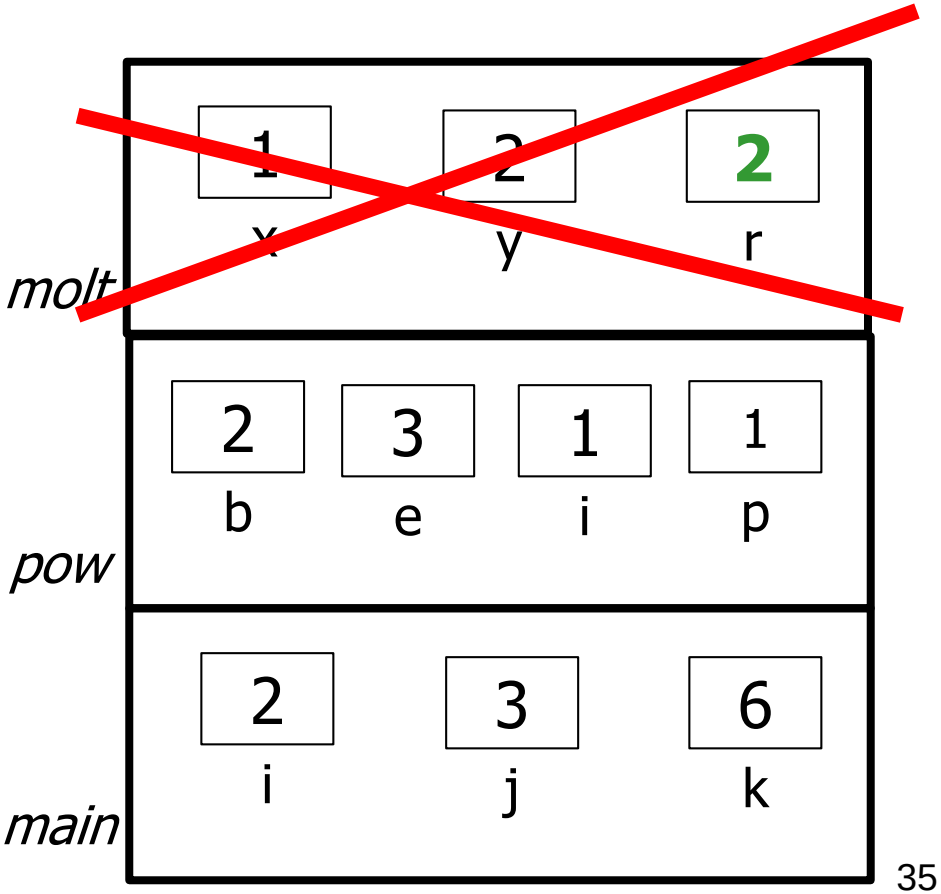
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

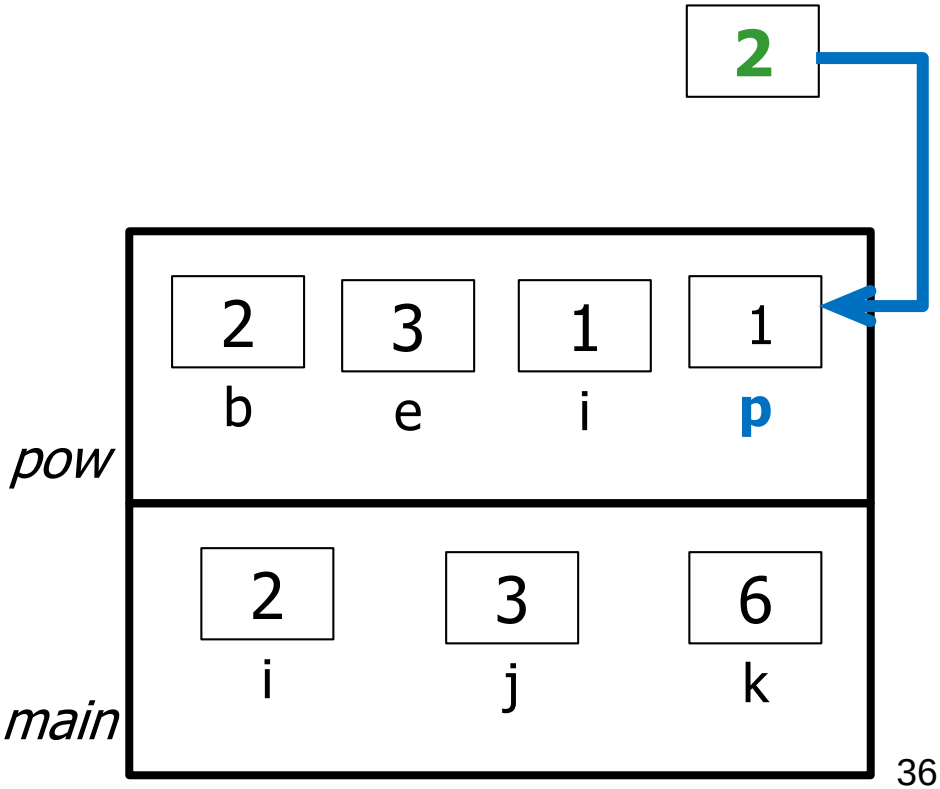
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

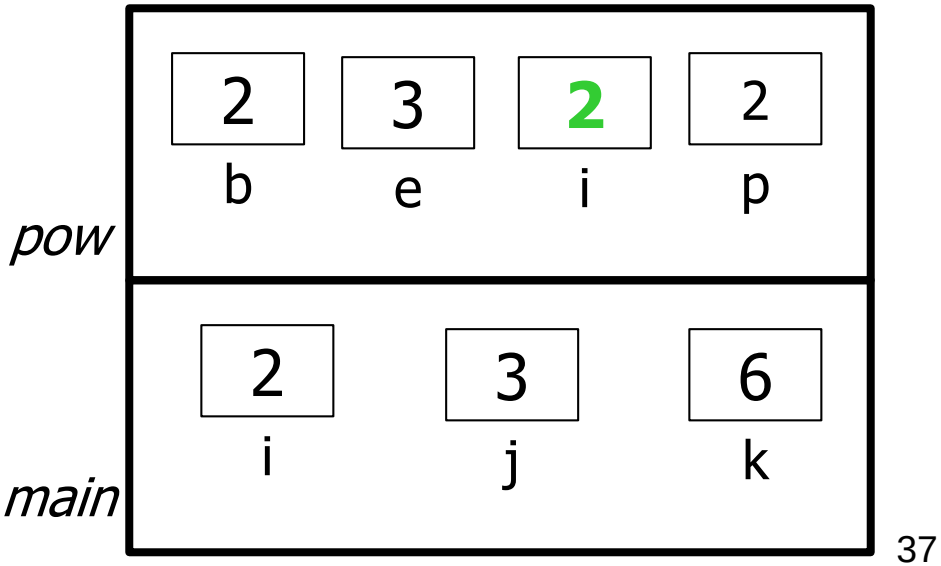
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

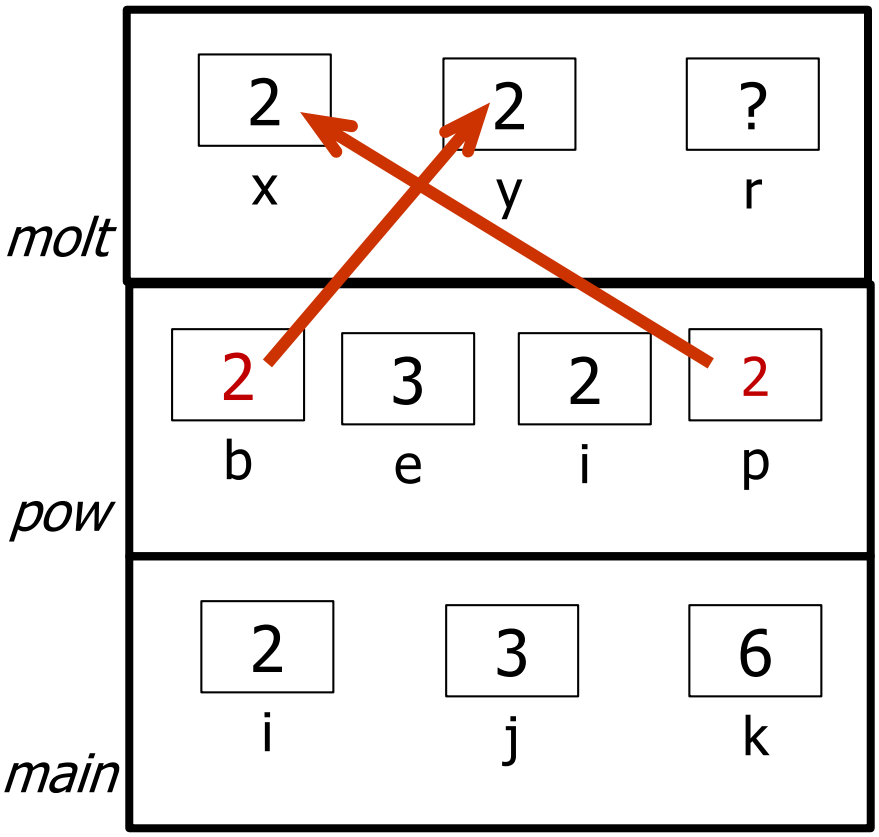
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

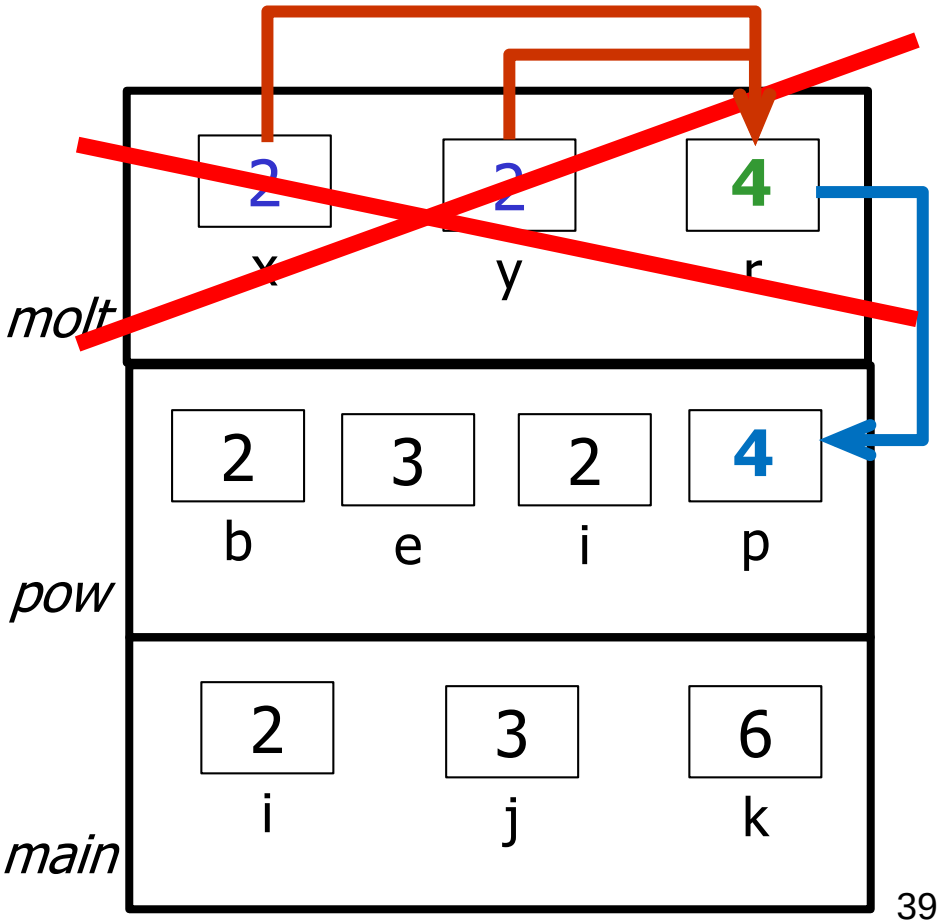
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

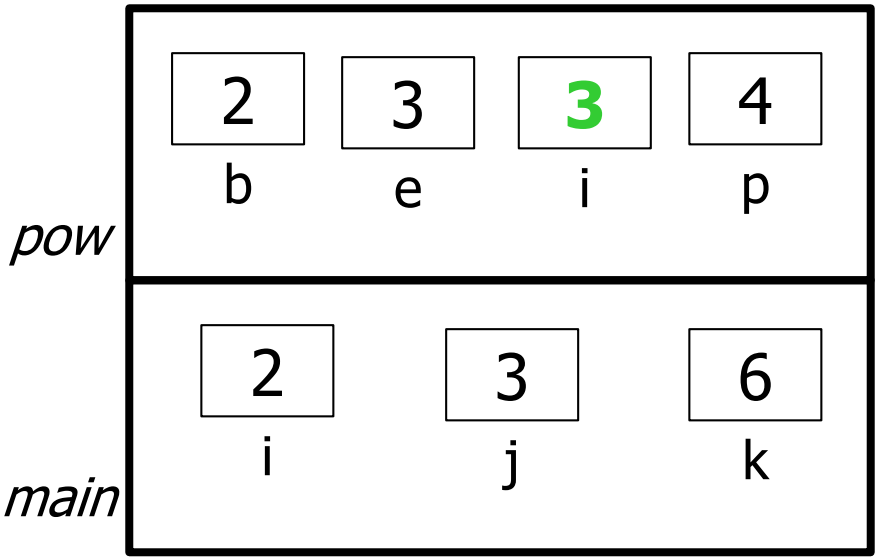
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

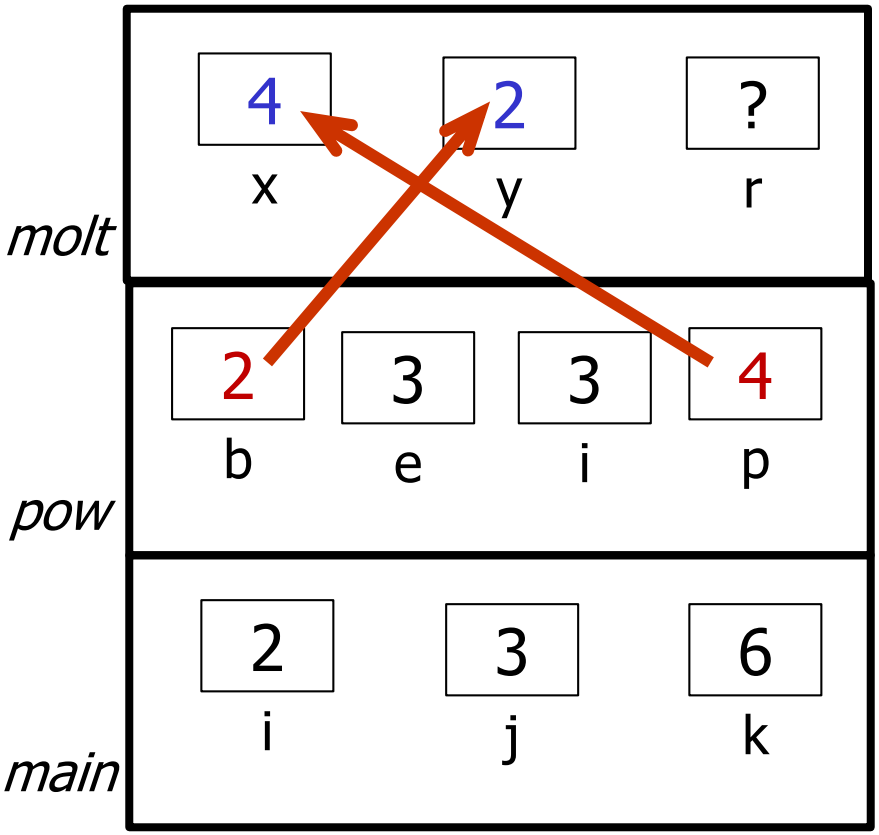
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

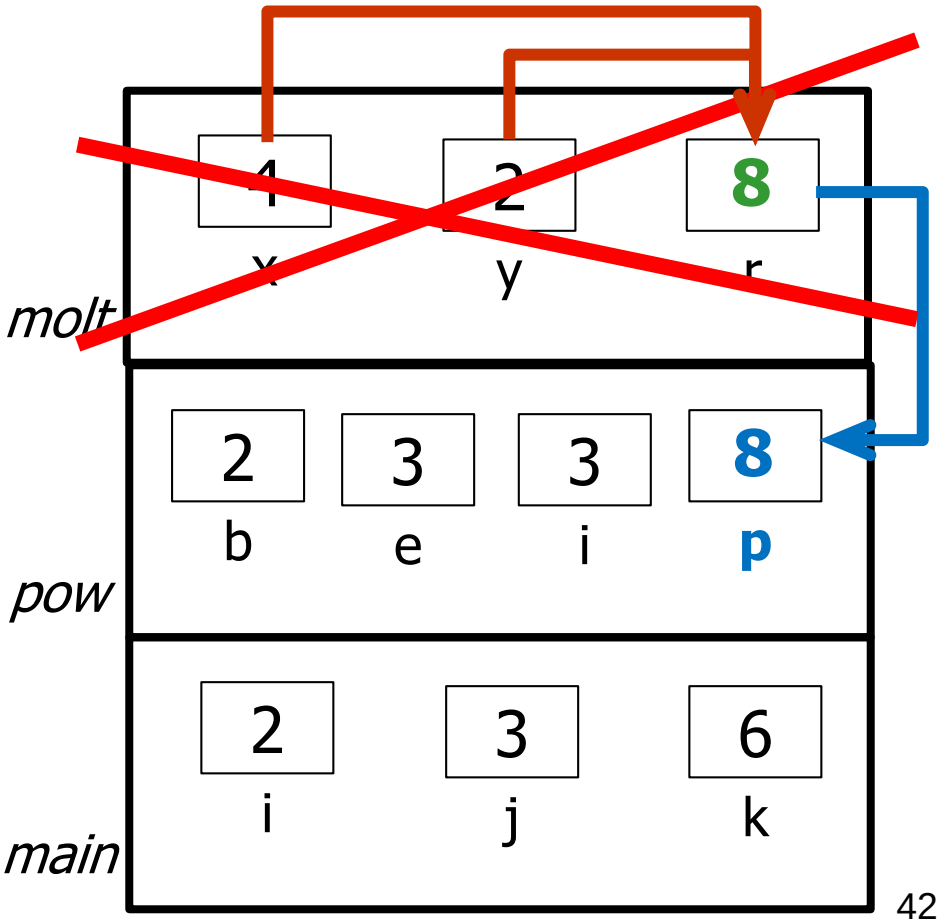
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

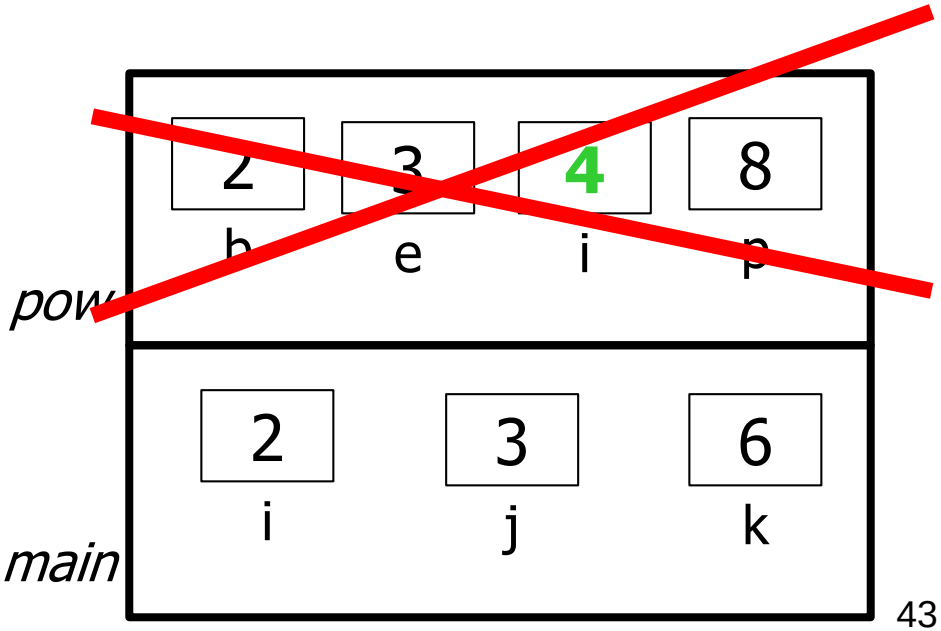
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

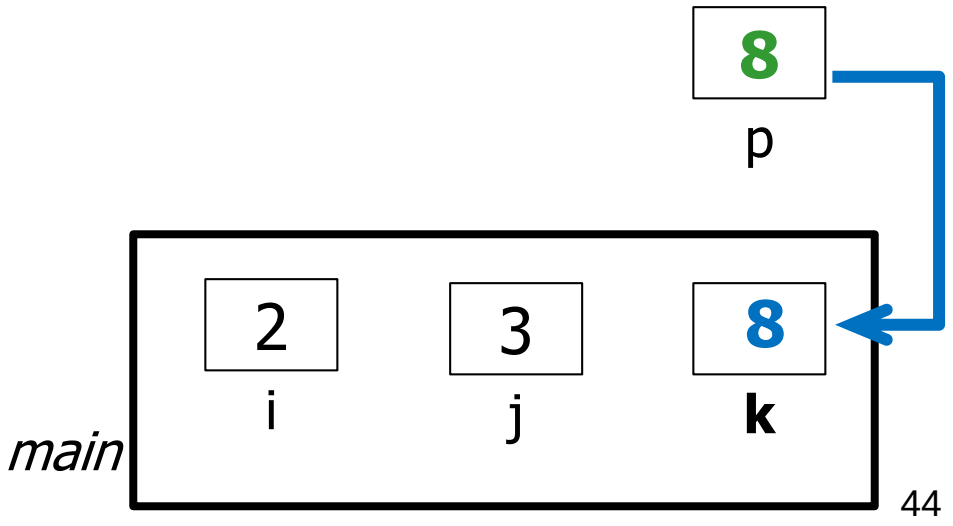
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Esempio di codice

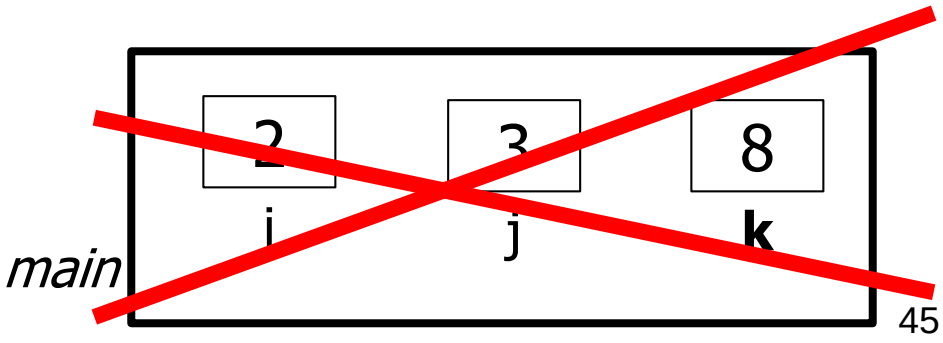
```
def moltiplica(x,y):
    return x*y

def power(b,e):
    p=1
    for i in range(1,e+1):
        p=moltiplica(p,b)
    return p

i=2
j=3

k=moltiplica(i,j)
print(k)

k=power(i,j)
print(k)
```



Record di attivazione

- Ogni sottoprogramma (incluso il main) ha associato un **record di attivazione**. Contiene:
  - tutti i dati relativi all'ambiente locale del sottoprogramma
  - l'indirizzo di ritorno nel programma chiamante
  - altri dati utili
- **Per ogni attivazione di sottoprogramma si crea un nuovo record di attivazione**

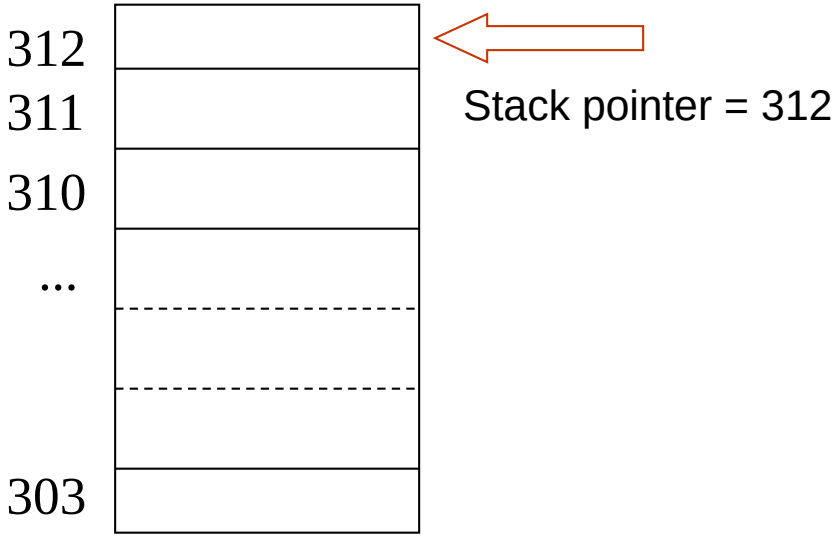
Perché è necessario?

- In generale, una funzione può essere chiamata un numero imprecisato di volte
- Ogni chiamata a procedura richiede allocazione di spazio di memoria per le sue variabili locali
  - Il compilatore potrebbe “preparare” un ambiente per ogni funzione definita? In generale no...
- **Vi sono procedure che richiamano se stesse** (vedremo: ricorsione)
  - **Possono esistere più “istanze” di una funzione, “addormentate” in attesa della terminazione di una “gemella” per riprendere l’esecuzione**
- In questo ultimo caso il compilatore non può sapere quanto spazio allocare per le variabili del programma (nei vari ambienti)

La pila (stack) di sistema

- Una porzione della memoria di lavoro, chiamata **stack** (***pila***)***:*** modalità **LIFO** (Last in First Out) permette al sistema operativo di gestire i processi e di eseguire le chiamate a sottoprogramma
- **Lo Stack Pointer (puntat. alla pila)** è un registro che contiene **l'indirizzo della parola di memoria da leggere nello stack**

- Operazione di ***inserimento:***
  - incremento SP
  - scrittura in parola indirizzata da SP
- Operazione di ***estrazione:***
  - lettura da parola indirizzata da SP
  - decremento SP



Map e filter

In Python ci sono 2 funzioni bultin che sono in grado di svolgere un ruolo simile alle comprehension:

map(func, seq): applica la funzione func a tutti gli elementi di seq e restituisce un nuovo iterabile;
filter(func, seq): restituisce un iterabile che contiene tutti gli elementi di seq per cui func(elem) è true.

map(func, seq) è simile a [func(elem) for elem in seq]:

```
>>> # definisco una funzione che restituisce il quadrato di un numero
>>> def square(n):
...     return n**2
...
>>> squares = map(square, range(10))
>>> squares  # map restituisce un oggetto iterabile
<map object at 0xb6730d8c>
>>> list(squares)  # convertendolo in lista si possono vedere gli elementi
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> # la seguente listcomp è equivalente a usare list(map(func, seq))
>>> [square(x) for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

filter(func, seq) è simile a [elem for elem in seq if func(elem)]:

```
>>> # definisco una funzione che restituisce True se il numero è pari
>>> def is_even(n):
...     if n%2 == 0:
...         return True
...     else:
...         return False
...
>>> even = filter(is_even, range(10))
>>> even  # filter restituisce un oggetto iterabile
<filter object at 0xb717b92c>
>>> list(even)  # convertendolo in lista si possono vedere gli elementi
[0, 2, 4, 6, 8]
>>> # la seguente listcomp è equivalente a usare list(filter(func, seq))
>>> [x for x in range(10) if is_even(x)]
[0, 2, 4, 6, 8]
```

Esercizi su Funzioni

Esercizio

- Si codifichi un programma che legge dallo standard input una sequenza (di lunghezza arbitraria) di interi positivi terminata dal valore 0 e, al termine della sequenza, visualizza su standard output un messaggio che indica *quante* ***coppie*** *di numeri consecutivi che siano diversi, pari* ***e*** *il cui prodotto sia un quadrato perfetto* sono contenute nella sequenza. Ecco un esempio:

  2 50 13 16 8 7 8 2 18 6 6 16 4 1 25 0

  Le coppie di numeri cercati sono 4 (2 50, 8 2, 2 18, 16 4)

- Si noti che un numero può anche appartenere a due coppie. Nella soluzione **si utilizzi** la funzione:

  dppqp(a, b) /* Restituisce 1 se i parametri sono **d**iversi, **p**ari, e il loro **p**rodotto è un **q**uadrato **p**erfetto, 0 altrimenti */

Esercizio

Le matrici quadrate N x N possono essere navigate in diversi modi. Due modi tradizionali sono il nested loop e il merge scan. Il nested loop è la navigazione riga per riga (o colonna per colonna), come esemplificato sotto (nella matrice i numeri rappresentano l'ordine di navigazione).

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

a) Si scriva una funzione che riceve in input due matrici quadrate A e B e un intero k. A e B sono matrici NxN e k è compreso tra 1 e NxN (non serve verificare, k è garantito essere compreso tra 1 e NxN). La funzione deve attraversare, secondo l'ordine del nested loop per righe, le prime k posizioni sia di A che di B e restituire il numero di interi uguali che si trovano nella stessa posizione in A e in B. (2 punti).

Esempio: con k=12 e le matrici

6	***8***	13	***87***	78	5	***9***	23
12	***45***	32	12				

5	***8***	12	***87***	56	6	***9***	11
13	***45***	11	32				

restituisce 4, perché sono uguali l'8 in posizione (1,2), l'87 in (1,4), il 9 in (1,7) e il 45 in (2,2).

Il metodo merge scan invece è quello che si muove lungo le diagonali partendo dal punto più in alto a sinistra, come esemplificato sotto (nella matrice i numeri rappresentano l'ordine di esplorazione).

1	2	4	7	11	16	22	29
3	5	8	12	17	23	30	37
6	9	13	18	24	31	38	44
10	14	19	25	32	39	45	50
15	20	26	33	40	46	51	55
21	27	34	41	47	52	56	59
28	35	42	48	53	57	60	62
36	43	49	54	58	61	63	64

b) Si scriva una funzione che riceve in input due matrici quadrate A e B e un intero k. A e B sono matrici NxN e k è compreso tra 1 e NxN (non serve verificare, k è garantito essere compreso tra 1 e NxN). La funzione deve attraversare, secondo l'ordine del merge scan, le prime k posizioni sia di A che di B e restituire il numero di interi uguali che si trovano nella stessa posizione in A e in B. (2 punti).
Esempio, con k=12 e le matrici

23	45	12	***3***	11
2	***32***	1	65	
1	9			
10				

3	43	87	***3***	9
1	***32***	3	100	
2	54			
10				

restituisce 3, perché sono uguali il 3 in posizione (1,4), il 32 in (2,2) e il 10 in (4,1).

Esercizio

- I risultati della finale olimpica di nuoto della staffetta 4x100 stile libero sono rappresentati mediante una matrice di dimensioni 4x8 di triplette di elementi che rappresentano minuti, secondi e millisecondi
- Ogni cella rappresenta il risultato di un singolo frazionista e i risultati sono disposti nella matrice secondo la corsia (e.g. corsia 1 nella prima colonna)
- Scrivere una funzione che riceve in ingresso la matrice che contiene il risultato e restituisce il numero della corsia vincente
- Scrivere una funzione che riceve in ingresso la matrice che contiene il risultato e una variabile che contiene il record mondiale di staffetta e che restituisce True se il record mondiale è stato battuto, False altrimenti.

Esercizio

- Scrivere una funzione che riceve in input una matrice di interi di dimensione N*N e restituisce un'opportuna struttura dati contenente tutti i punti della matrice contenenti un valore uguale al massimo valore contenuto nella matrice stessa

Esercizio

- Scrivere un programma che definendo opportune funzioni, risolva il seguente problema. Date due stringhe S1 e S2, stabilire se S2 compare come sottosequenza di S1 e, in caso affermativo, a partire da quale indice. Creare quindi una nuova stringa S3 costruita come S1 meno la prima occorrenza della stringa S2. Non è permesso utilizzare le funzioni di libreria del linguaggio relative alla manipolazione di stringhe.
- Esempio: S1 = “melograno”, S2 = “grano”;

  restituisce 5 (S2 è inclusa in S1 a partire dal 5° carattere) e

  S3 = “melo”.
- Se fosse S1 = “sperpero”, S2 = “per”;

  il risultato è 2 e S3 = “spero”.

Esercizio

- Si considerino due matrici di interi A e B, di uguali dimensioni (R e C, costanti, che indicano il numero di righe e colonne). Diciamo che A *domina* B se, confrontando i valori in posizioni corrispondenti, risulta che il numero dei valori in A maggiori dei corrispondenti valori in B è più grande del numero di quelli di B maggiori dei corrispondenti in A e inoltre gli elementi corrispondenti non sono mai uguali (se due elementi corrispondenti sono uguali la dominanza non è definita).
- Si codifiche la funzione **domina(**...**)** che riceve le matrici come parametri e restituisce 1 se la prima domina la seconda, -1 se la seconda domina la prima, 0 altrimenti.

Esercizio

- *SimpleScrabble* è una versione del gioco *Scrabble* dove il valore delle parole dipende solo dalle lettere che le compongono. Ogni lettera ha un valore (da 1 a 10) ed è disponibile in un certo numero di esemplari (da 1 a 12). Il valore di una parola è la somma dei valori delle sue lettere. Gli array **valore** e **numero**, definiti nell'ambiente globale come segue, indicano il valore e il numero di esemplari disponibili di ognuna delle 26 lettere, in ordine alfabetico. Consideriamo per semplicità solo le lettere maiuscole.
- numero = [9,2,2,4,12,2,3,2,9,1,1,4,2,6,8,2,1,6,4,6,4,2,2,1,2,1]
- valore[1,2,3,2,1,4,2,4,1,8,5,1,3,1,1,3,10,1,1,1,1,4,4,8,4,10]
- Ci sono quindi 9 A da 1 punto, 2 B da 2 pt, 2 C da 3 pt, 4 D da 2 pt, 12 E da 1 pt, …, 1 Z da 10 pt.
- Una parola è valida se è di almeno 2 lettere ed è ottenibile con le lettere a disposizione (CAB vale 3+1+2=6 pt, KARAOKE non è valida, perché disponiamo di una sola K).
- Una funzione **valida(**...**)**, che restituisce 1 se la parola è valida, 0 altrimenti
- Una funzione **punteggio(…)** calcola e restituisce il valore della parola ricevuta come parametro, assegnando 0 alle parole *non valide*.
- Si codifichino le funzioni **valida(…)** e **punteggio(**...**)**

Esercizio

- Definiamo *totalmente diverse (TD)* due parole che non hanno lettere in comune. Ad esempio "fuoco" e "aria" sono TD, mentre "fuoco" e "acqua" no (entrambe hanno 'c', anche se in posizioni diverse).
- Si definisca una funzione **… td(…)** che riceve come parametri due stringhe (che supponiamo valide e ben formate) e restituisce **1** se le stringhe rappresentano due parole TD, **0** altrimenti
- Si scriva un programma che legge da stdin una sequenza (di lunghezza arbitraria, anche enorme) di parole, ognuna di massimo 50 caratteri, e al termine della sequenza indichi su stdout il numero di parole della sequenza e il numero di coppie di parole TD **consecutive**. La sequenza è terminata dalla parola "fine" (che non deve essere considerata parte della sequenza)

Esercizio

- Si consideri una matrice quadrata di NxN punti del piano cartesiano
- Gli N punti della diagonale, quelli di ogni riga e quelli di ogni colonna definiscono linee spezzate di N-1 lati. Diciamo che una matrice di punti è *regolare* se la lunghezza della spezzata definita dalla *diagonale principale* è maggiore della lunghezza di *tutte* le spezzate definite dalle righe e dalle colonne della matrice.
- Si codifichi funzione **regolare(**...**)** che, data una matrice, restituisce 1 se è regolare, 0 altrimenti

Esercizio

- Scrivere in linguaggio C, un programma che letta una sequenza di N numeri complessi dallo standard input (rappresentati con parte reale e parte immaginaria) stampi video la sequenza ordinata in maniera crescente secondo il valore dei loro moduli.

Esercizio

- Sia dato un vettore di N ≥ 2 elementi. Ogni elemento del vettore è un numero intero. Si chiede di progettare e codificare una funzione che verifichi se, a partire da un elemento di indice i ≥ 0 fino alla fine del vettore, ogni elemento abbia valore inferiore alla somma di tutti gli elementi collocati alla sua destra, cioè alla somma di tutti gli elementi aventi indice superiore al suo (beninteso questa verifica non s'applica all'ultimo elemento, che non ha niente alla sua destra).
- Per esempio, il vettore seguente (con N = 5) soddisfa alla condizione in questione, a partire dall'elemento di indice 1 (ma non a partire da quello di indice 0).
  indice 0 1 2 3 4
  elemento 9 3 0 1 4
- La funzione ha come argomento il vettore e l'indice i da cui partire con la verifica. Essa restituisce 1 se il vettore soddisfa alla condizione così descritta, 0 altrimenti.

Esercizio

- Un numero si dice doppiamente primo se è primo e tutte le sue cifre sono numeri primi.
- Si scriva una funzione f che riceve una matrice NxN di interi e restituisce 1 se tutti i numeri doppiamente primi che contiene sono circondati solo da numeri che non sono doppiamente primi nelle otto direzioni possibili per le caselle centrali, nelle cinque o tre possibili per le caselle sul bordo della matrice, 0 altrimenti.
- Un insieme di celle di una matrice si dice connesso se esiste un percorso che partendo da una delle celle permette di raggiungere le altre muovendosi di un passo alla volta nelle otto direzioni attraversando solo celle dell'insieme stesso.
- Si scriva una funzione g che riceve una matrice NxN di interi e restituisce 1 se tutti i numeri doppiamente primi che contiene appartengono a un unico insieme connesso di celle, 0 altrimenti.

Esercizio

- Dopo che il codificatore ha composto il codice, il decodificatore fa il suo primo tentativo, cercando di indovinare il codice. Il codificatore, appena il suo avversario ha completato il tentativo, fornisce degli aiuti comunicando: il numero di cifre giuste al posto giusto, cioè le cifre del tentativo che sono effettivamente presenti nel codice al posto tentato con degli 1 e il numero di cifre giuste al posto sbagliato (cioè le cifre del tentativo che sono effettivamente presenti nel codice ma non al posto tentato) con dei 2.
- Non bisogna comunicare quali cifre sono giuste o sbagliate ma solo quante.
- Si scriva una funzione che riceve in input tre array di dimensione 4 che rappresentano 1) il codice da decodificare, 2) il tentativo di indovinarlo e 3) gli aiuti dati in risposta al tentativo.
- La funzione compara l'array tentativo con l'array codice e riempie l'array risposta in base alle regole del mastermind (nell'array risposta le caselle che non devono essere riempite siano messe tutte a 0).

- Esempi
- Se codice=”2 4 6 3” e tentativo=”1 9 5 7”, risposta dovrà contenere “0 0 0 0”
- Se codice=”2 4 6 3” e tentativo=”2 6 1 7”, risposta dovrà contenere “1 2 0 0”
- Se codice=”2 4 6 3” e tentativo=”7 5 6 3”, risposta dovrà contenere “1 1 0 0”
- Se codice=”2 4 6 3” e tentativo=”2 4 6 3”, risposta dovrà contenere “1 1 1 1”
- Se codice=”2 4 6 3” e tentativo=”6 3 4 2”, risposta dovrà contenere “2 2 2 2”
- Se codice=”2 4 6 3” e tentativo=”7 3 9 3”, risposta dovrà contenere “1 0 0 0”
- Se codice=”2 4 4 3” e tentativo=”6 3 4 2”, risposta dovrà contenere “1 2 2 0”
- Se codice=”2 4 4 3” e tentativo=”2 9 2 0”, risposta dovrà contenere “1 0 0 0”