

ESERCIZI
ARRAY/LISTE
FUNZIONI
REGOLE DI VISIBILITÀ DELLE VARIABILI

INFORMATICA

ESERCIZIO: PRIMALITÀ

IDEA PER RISOLVERE IL PROBLEMA

- ▶ Consideriamo il numero n da testare
- ▶ Sia m il numero di divisori non banali di n trovati.
All'inizio m è zero
- ▶ Per ogni numero k da 2 a $n-1$ controlliamo
 - ▶ Se il resto della divisione di n per k è zero
 - ▶ Incrementiamo il numero di divisori non banali
- ▶ Se m è zero il numero è primo altrimenti è composto

A R R A Y

GLI ARRAY/LISTE

GLI ARRAY/LISTE

- ▶ Spesso non ci serve una sola variabile di un tipo, ma molte
- ▶ Ad esempio per memorizzare 5 valori ci servono 5 variabili...
- ▶ `valore1 = 22; valore2 = 33; ...; valore5 = 78`
- ▶ Questo è poco pratico (e se le variabili fossero 1000?)
- ▶ Per questo abbiamo gli array/liste:
- ▶ `valori = [22, 33, 99, 12, 78]`

UNA NOTA SULLA TERMINOLOGIA

- ▶ Quelli che qui chiamiamo array sono chiamati **liste** in Python
- ▶ Noi utilizziamo il termine array (o, a volte, vettori) perché il termine liste verrà utilizzato per altri tipi di strutture
- ▶ Le liste/array in Python hanno una serie di funzionalità utili ed in comune a tutte le *collezioni* di Python, che però non vedremo qui.

ARRAY: IDEA DI BASE

- ▶ Se ci servono 10 valori diamo un "nome unico" ad un blocco di memoria che li può contenere (un array)
- ▶ Accediamo alle singole celle/valori con il nome dell'array e l'indice della variabile
- ▶ Esempio:
la cella di posizione 5 all'interno dell'array *un_array*

ARRAY: ESEMPI

`valori = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`



`punto = [6, 2, 9]`

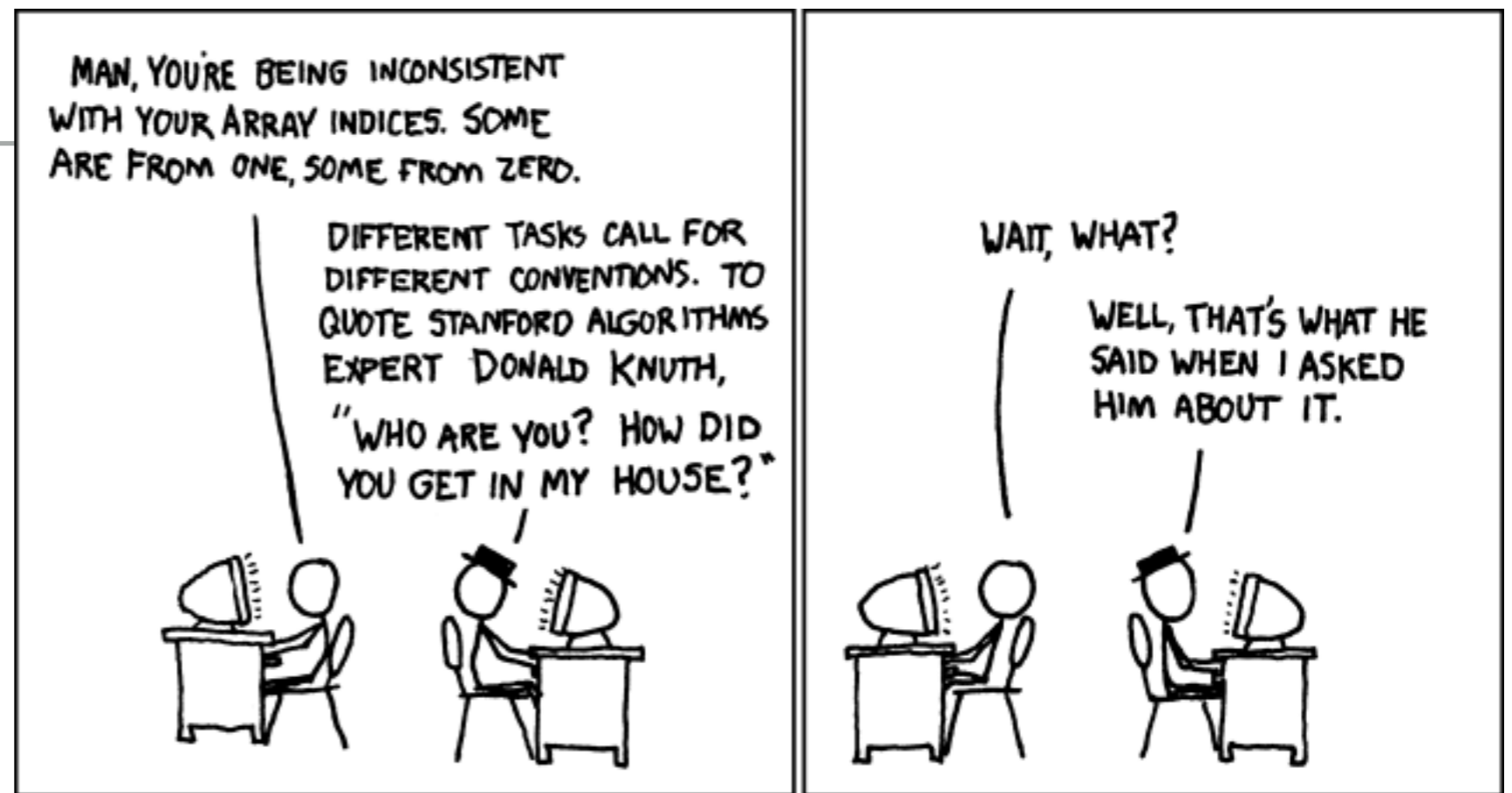


Ok, ma come si accede alle singole celle?

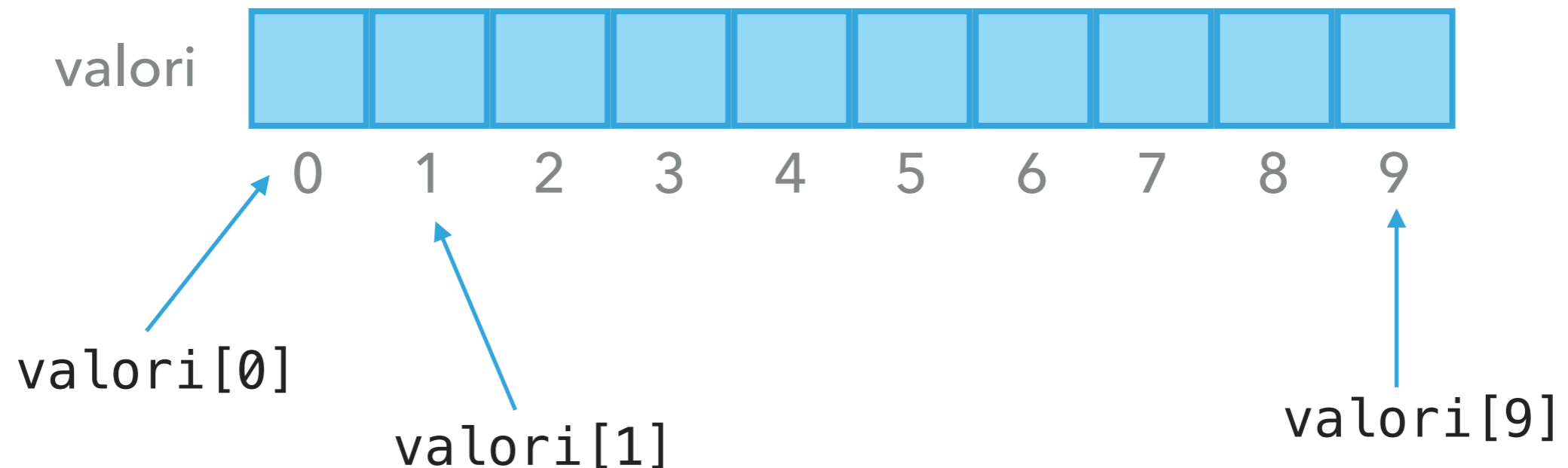
ARRAY

INDICI

Un array con n elementi ha indici $0, 1, 2, \dots, n-1$



Credits: <https://xkcd.com/163/>



ARRAY: ANCORA INDICI

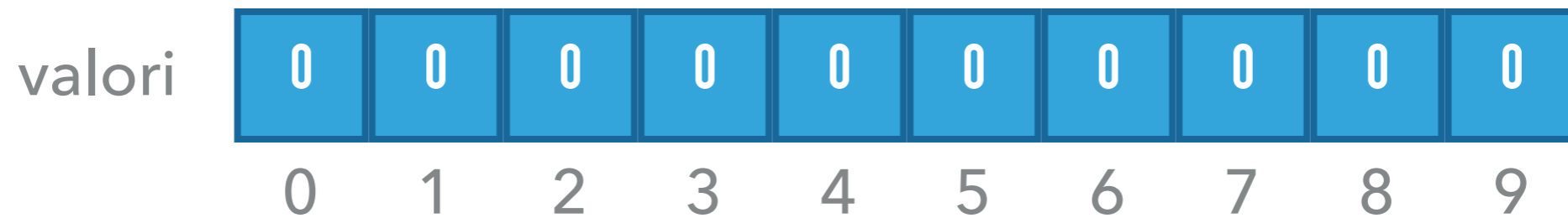
- ▶ Possiamo utilizzare espressioni che restituiscono valori interi per indirizzare gli elementi dell'array:
 - ▶ `a[9] = 5 # OK`
 - ▶ `a[2*3] = 7 # OK`, è come dire `a[6]`
 - ▶ `a = 5 # Funziona`, ma ora "a" non è più un array
 - ▶ `a[2.0 * 3.5] = 8 # NO`, l'indice deve essere un intero
 - ▶ `a[1] = [1,2,3,4] # OK`, può contenere altri array

INDICI

```
valori = [0,0,0,0,0,0,0,0,0,0]  
valori[2] = 4  
valori[5] = 2  
valori[valori[2]] = 3
```

INDICI

```
valori = [0,0,0,0,0,0,0,0,0,0]  
valori[2] = 4  
valori[5] = 2  
valori[valori[2]] = 3
```



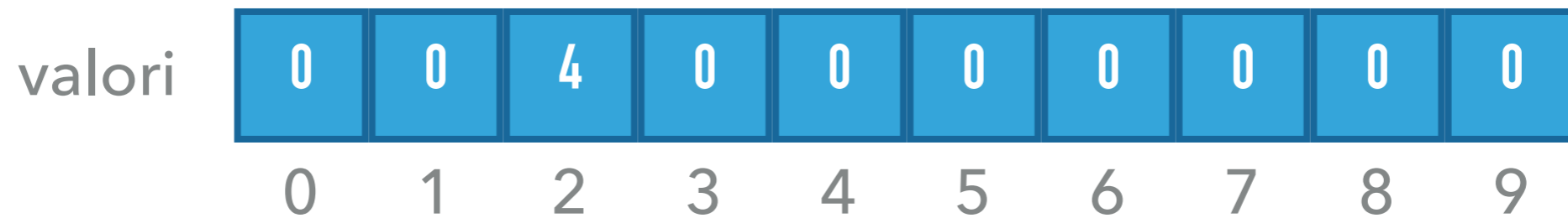
INDICI

```
valori = [0,0,0,0,0,0,0,0,0,0]
```

```
valori[2] = 4
```

```
valori[5] = 2
```

```
valori[valori[2]] = 3
```



INDICI

```
valori = [0,0,0,0,0,0,0,0,0,0]
```

```
valori[2] = 4
```

```
valori[5] = 2
```

```
valori[valori[2]] = 3
```

valori	0	0	4	0	0	2	0	0	0	0
	0	1	2	3	4	5	6	7	8	9

INDICI

```
valori = [0,0,0,0,0,0,0,0,0,0]
```

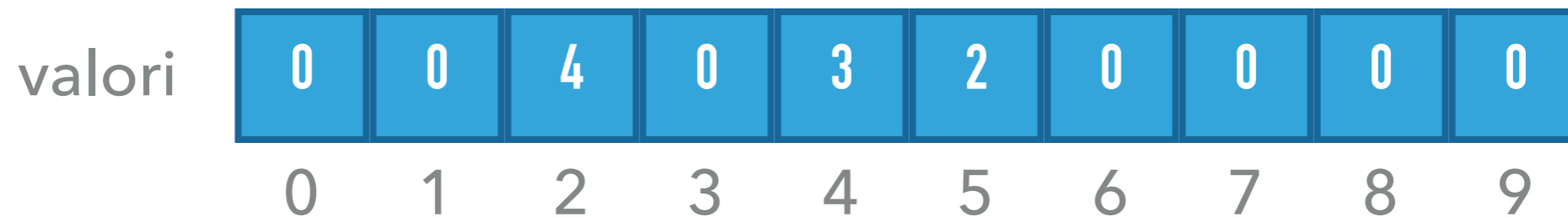
```
valori[2] = 4
```

```
valori[5] = 2
```

```
valori[valori[2]] = 3
```

Equivale a:

```
valori[4] = 3
```



ARRAY: FUNZIONALITÀ UTILI

- ▶ `a = [1,2,-23,5,8]`
- ▶ `len(a)` # restituisce la lunghezza dell'array a, che è 5
- ▶ `[0] * 10` # restituisce un array di 10 elementi identici (tutti zero)
- ▶ `[1,2] + [3,4]` # concatena i due array, ritornando `[1,2,3,4]`
- ▶ `3 in a` # restituisce True se 3 appartiene all'array "a"
- ▶ `3 not in a` # restituisce True se 3 non appartiene ad "a"
- ▶ `a.append(5)` # aggiunge 5 alla fine dell'array

USO ASSIEME AL CICLO FOR

Si usano spesso i cicli for per iterare su tutti i valori di un array:

```
quadrati = [0] * 10
for i in range(0,10):
    quadrati[i] = i * i
```

Riempimento dei valori

```
for i in range(0,10):
    print(str(i) + "*" + str(i) + "=" + str(quadrati[i]))
```

Stampa dei valori

ARRAY: ACCESSO OLTRE I LIMITI

Non accedere oltre i limiti di un array

```
un_array = [0,0,0,0,0]
```

```
un_array[5] = 10
```

Questo indice è oltre quello validi (da 0 a 4)

Ma cosa accade?

Viene generata una eccezione che ci dice che stiamo accedendo ad un elemento fuori dai limiti dell'array

Credits: <https://xkcd.com/292/>



QUIZ SU ARRAY

```
x = [1.5, 2.0, 3.2, 10.3]
s = 0;
for i in range(0,4):
    s = s + x[i]
```

Quale è il valore della variabile s dopo l'esecuzione del codice?

1. Stiamo accedendo fuori dai limiti

2. Stiamo usando float invece di interi come indici

3. 17.0

4. 10.3

QUIZ SU ARRAY

```
x = [0,0,0,0]
x[1] = 0
x[2] = 4
x[0] = x[2]
x[3] = 9
```

Quale è il contenuto dell'array x dopo l'esecuzione del codice?

1. [4,0,4,9]

2. Stiamo accedendo fuori dai limiti

3. [4,0,2,9]

[redacted]

ARRAY MULTIDIMENSIONALI

- ▶ Oltre a "vettori" (array uno-dimensionali), possiamo avere matrici (array bidimensionali), etc.
- ▶ L'accesso ad un array ad 2 dimensioni necessita di due indici, a tre dimensioni di tre indici, etc.
- ▶ Per il resto si comportano in modo simile agli array di una singola dimensione

ARRAY MULTIDIMENSIONALI: ESEMPI

```
valori = [[0, 0, 0],  
          [0, 0, 0],  
          [0, 0, 0],  
          [0, 0, 0],  
          [0, 0, 0]]
```

Array di 5 righe e 3 colonne

L'accesso agli elementi richiede due indici:

```
valori[2][1] = 4
```

```
valori[1][2] = 5
```

valori

4			
3			
2		4	
1			5
0			
	0	1	2



FUNZIONI

FUNZIONI

- ▶ Molto spesso scriviamo codice che fa la stessa cosa ma in punti diversi (o con valori diversi)
- ▶ Sarebbe comodo raggruppare "pezzi di codice" con un unico compito sotto un unico nome
- ▶ E avere "pezzi di codice" riutilizzabile sarebbe utile

ESEMPIO D'USO

n = 4

```
fattoriale = 1
for i in range(1,n+1):
    fattoriale = fattoriale * i
print("Il fattoriale di " + str(n) + " è " + str(fattoriale))
```

Questo pezzo calcola il fattoriale. Non sarebbe utile scriverlo una volta sola e quando ci serve fare:

n = 4

```
fattoriale = calcola_fattoriale(n)
print("Il fattoriale di " + str(n) + " è " + str(fattoriale))
```

A PROCEDURE IS A PATTERN FOR THE LOCAL EVOLUTION OF A COMPUTATIONAL PROCESS. IT SPECIFIES HOW EACH STAGE OF THE PROCESS IS BUILT UPON THE PREVIOUS STAGE.

H. Abelson, G.J. Sussman.
Structure and Interpretation of Computer Programs

STRUTTURA COMUNE DELLE FUNZIONI

Nome della funzione

Dice come chiameremo la funzione nel nostro codice

```
def nome_funzione (parametri formali):  
    # codice della funzione
```

Parametri formali

Un elenco di nomi di variabili: indicano che input riceve la funzione
Possono essercene 0, 1 o più di uno

ESEMPIO D'USO

Qui diciamo come è fatta (come si chiama e che argomenti prende) la funzione `calcola_fattoriale`

```
def calcola_fattoriale(n):  
    fattoriale = 1  
    for i in range(1, n+1):  
        fattoriale = fattoriale * i  
    return fattoriale
```

Qui indichiamo che il valore che dobbiamo fornire è quello contenuto nella variabile "fattoriale"

PARAMETRI FORMALI E ATTUALI

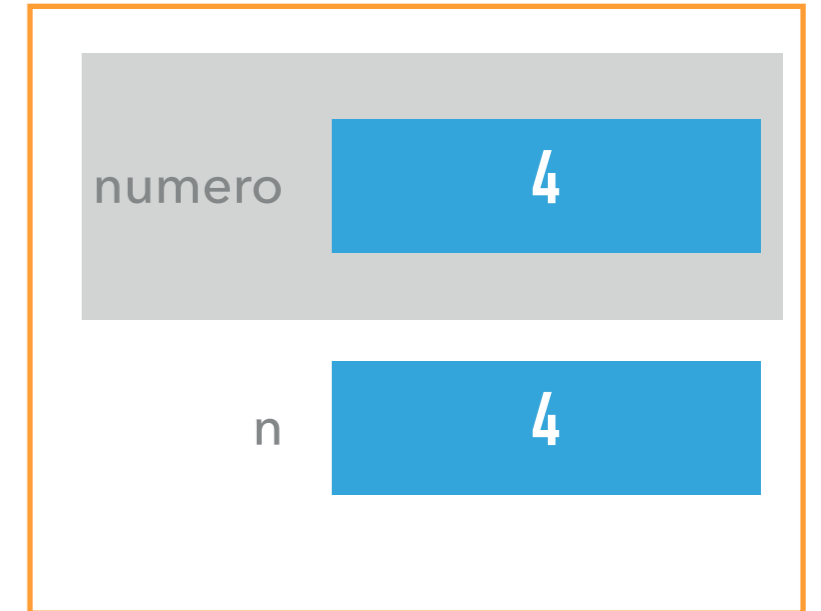
Parametro formale: dice come si chiamerà il valore che passiamo come input alla funzione

```
def calcola_fattoriale(n):  
    ...
```

Parametro attuale: il valore che avrà la variabile n mentre il corpo della funzione viene eseguito

```
calcola_fattoriale(5)
```

RECORD DI ATTIVAZIONE

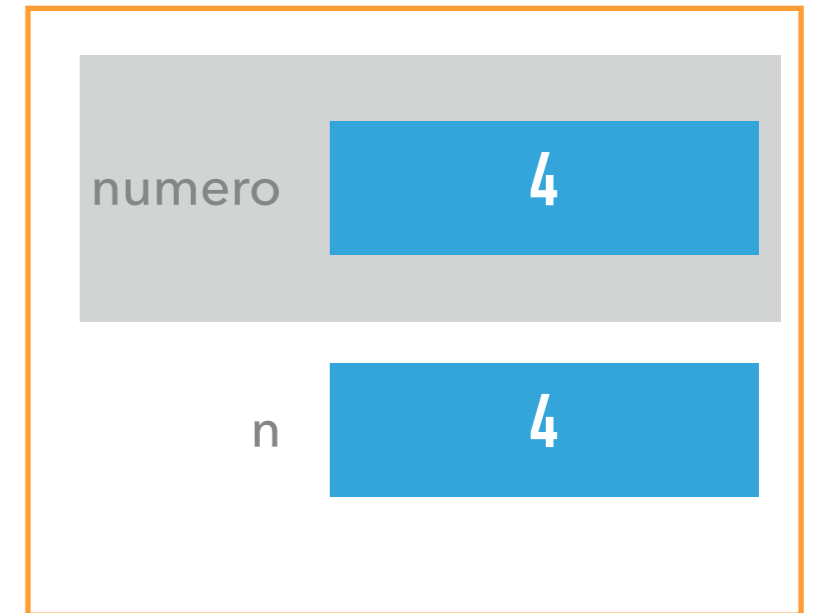


Ogni funzione ha un suo "record di attivazione" che contiene:

- ▶ Le sue variabili locali (quelle definite nel corpo della funzione)
- ▶ I valori dei parametri della funzione
- ▶ L'indirizzo da cui far ripartire l'esecuzione del chiamante

RECORD DI ATTIVAZIONE

Alcune note importanti:



- ▶ Gli argomenti sono passati per assegnamento in Python
- ▶ e.g., `calcola_fattoriale(5)` assegna a n il valore come se si scrivesse $n=5$
- ▶ Quindi modificare n nella funzione non cambia il valore nel chiamante.
- ▶ Questo ha alcune ripercussioni per quanto riguarda gli array

QUIZ SULLE FUNZIONI

```
def funzione(a, b):  
    tmp = (a + b)/2  
    return tmp
```

Quale è l'output di `funzione(4,6)`?

1. 5.0

2. 6.0

3. tmp

4. Non funziona perché a e b non sono inizializzati

QUIZ SULLE FUNZIONI

```
def funzione1():  
    a = 1  
    b = funzione2(a)
```

```
def funzione2(a):  
    a = a + 1  
    return 5
```

Quali sono i valori di a e b dopo l'esecuzione del codice?

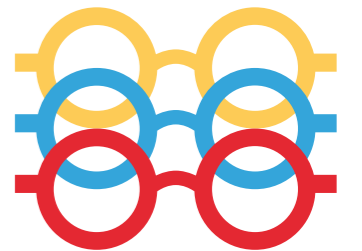
1. a è 2
b è 5

2. a è 1
b è 5

3. Il codice non funziona: stesso nome di variabile

4.

a è CTHUHU b è $\sqrt{2}$



VISIBILITÀ DELLE VARIABILI

COSA È LA VISIBILITÀ (SCOPE) DELLE VARIABILI?

- ▶ Una variabile non "esiste" sempre e non è accessibile da ogni parte del codice
- ▶ Pensate alla scomodità di non poter usare la variabile "i" perché qualcuno l'ha usata dall'altra parte
- ▶ Esistono quindi regole che dicono che "pezzo di codice" "vede" una certa variabile

RULE OF THUMB

- ▶ **Variabili locali:** Una variabile definita nella stessa funzione in cui si utilizza è visibile all'interno di quella funzione
- ▶ È possibile accedere anche alle variabili **globali** definite al di fuori di ogni funzione (ma generalmente sconsigliato)
- ▶ Una funzione non può "vedere" le variabili locali di un'altra funzione, neanche se chiama o è chiamata da quella funzione

QUIZ SULLE FUNZIONI

```
def funzione1():  
    a = 1  
    b = funzione2(a)
```

```
def funzione2(n):  
    c = a + 1  
    return c
```

Quale è il valore di b dopo l'esecuzione del codice?

1. 31

2. 30

3. funzione(30)

4. Il codice non è corretto: la variabile 'a' non è visibile