

Programmazione in Python

Lezione 1

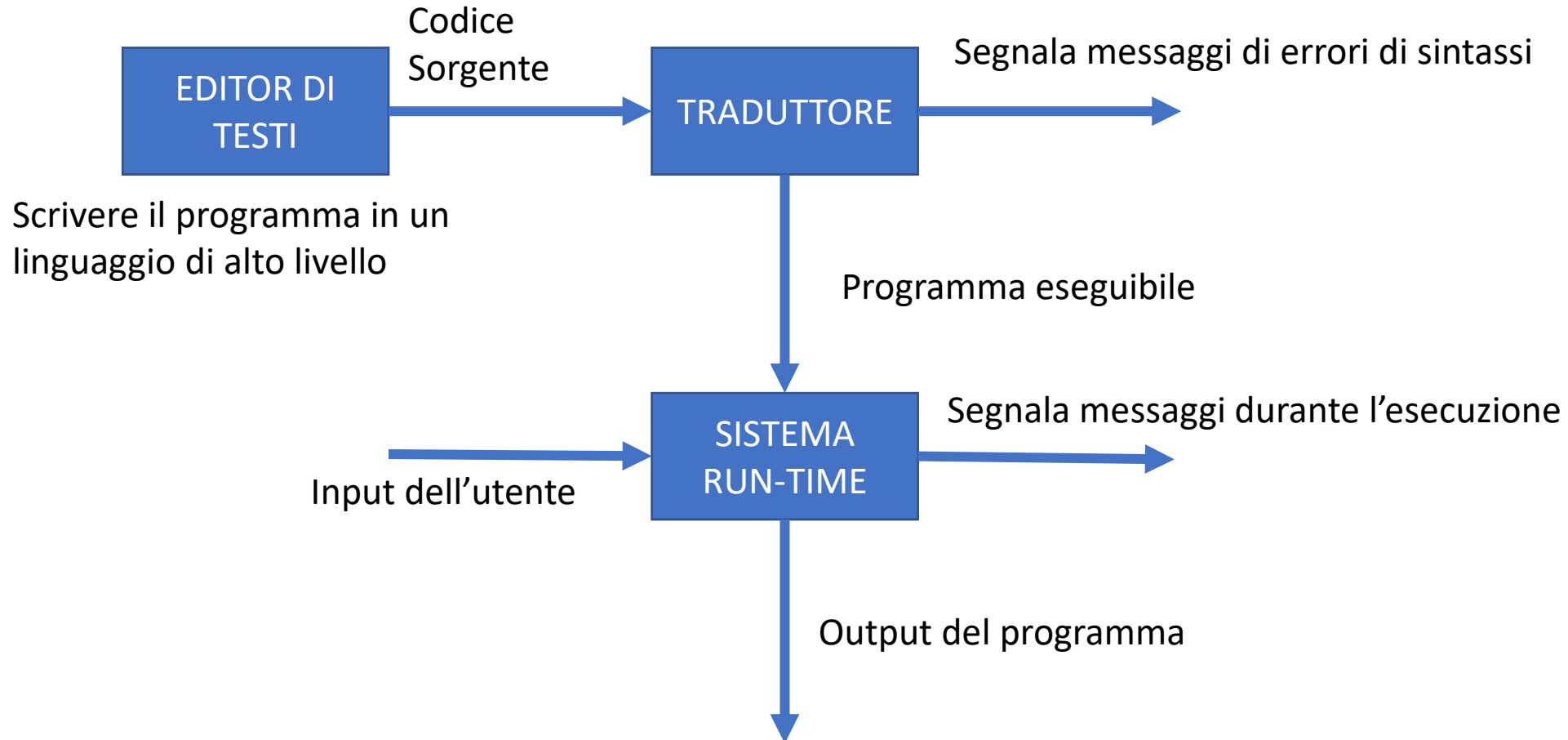


Python

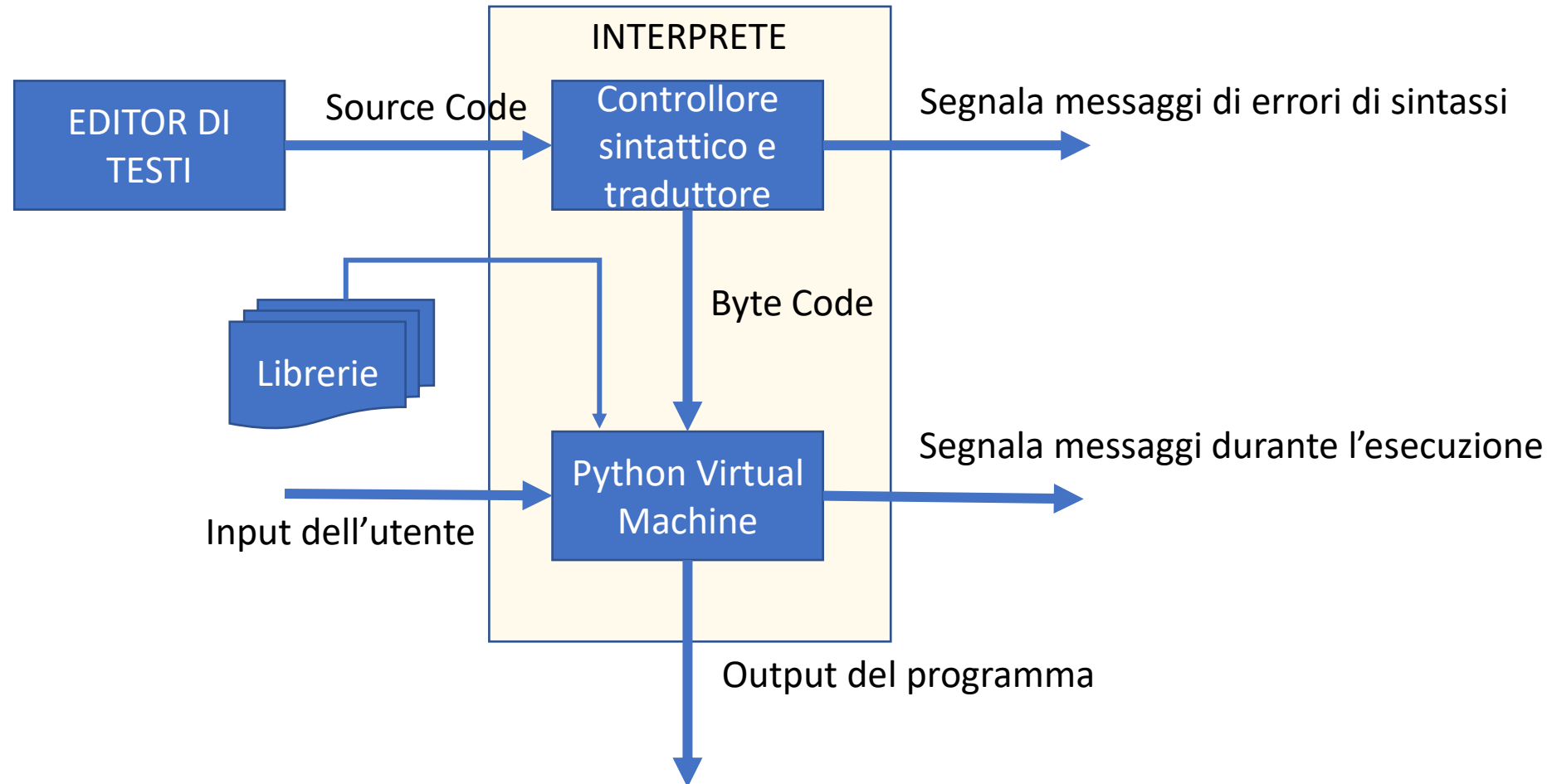
- Linguaggio di programmazione di alto livello
- Sviluppato agli inizi degli anni '90 da Guido Van Rossum
- Pensato per programmare velocemente e modificare i programmi altrettanto velocemente
- Oggi è uno dei linguaggi più largamente utilizzati
 - Accademia, applicazioni professionali, ecc.
- Sintassi più semplice di altri linguaggi
- Portabilità

Useremo Python3

Fasi dell'attività di programmazione



Fasi dell'attività di programmazione in Python



Programmare in Python

- Scrivere il programma con l'ausilio di un editor di testo o di un *ambiente di sviluppo integrato*

```
# il mio primo programma  
print('HELLO WORLD!')
```

- Eseguire il programma
 - Come fare dipende dall'ambiente utilizzato
 - Viene eseguito usando l'*interprete* Python
 - Shell interattiva

```
HELLO WORLD!
```

Programma

- Contiene istruzioni o enunciati
- Vengono tradotti ed eseguiti dall'interprete Python
- Attenzione alla distinzione tra maiuscole e minuscole
 - Esempio: posso scrivere `print('Ciao')` ma non `Print('Ciao')`
- Attenzione alla indentazione
 - Gli enunciati all'interno di uno stesso blocco devono iniziare alla stessa colonna

```
print( 'HELLO WORLD!' )  
    print( 'Non va bene' )
```

Errori

- ***Errori di sintassi***

- Riconosciuti durante la fase di compilazione
- Non consentono la generazione del programma eseguibile

- ***Errori di esecuzione (o logici)***

- Il programma eseguibile viene creato
- Una volta eseguito il programma termina inaspettatamente producendo un'eccezione (ad esempio quando si divide per zero) oppure
- Produce un risultato inaspettato
- Segnale che qualcosa non va nella logica del programma

Il mio primo programma

```
# il mio primo programma  
print('HELLO WORLD!')
```

Commento

stringa

Stampa la riga di testo
Hello World!

Sintassi

```
print(valore1, ..., valoren)
```

- `print` è una Funzione
- `valore1, ..., valoren` sono gli argomenti
- **Esempio** `print('La somma di 3 e 2 è', 3+2)`

Stampa
La somma di 3 e 2 è 5

Il tipo di dato

I valori utilizzati all'interno di un programma possono essere di *tipi* diversi

- In Python ogni valore è di uno specifico tipo
- Il tipo di dato associato ad un valore specifica
 - La rappresentazione e memorizzato all'interno del calcolatore
 - Le operazioni possibili
- I tipi messi a disposizione dal linguaggio sono detti *primitivi*
 - E' possibile utilizzare dati definiti dall'utente

I tipi primitivi in Python

- Alcuni dei tipi di dati supportati da Python
 - Numeri
 - Stringhe di testo
 - File
 - Contenitori
- Tra i numeri abbiamo
 - *Numeri interi* (Es. 5, -27, 0): `int`
 - Quali? Dipende dall'implementazioni, generalmente tra $-2^{31}-1$ e $+2^{31}$
 - *Numeri decimali in virgola mobile* (floating point) (Es. 3.57, 0.28, -7.890): `float`
 - Quali? Dipende dall'implementazioni, generalmente tra -10^{308} e $+10^{308}$ e utilizzano 16 cifre per la precision

Operazioni sui valori - espressioni

Espressioni aritmetiche con operatori aritmetici

**	elevamento a potenza: es. $a^{**}b$
-	cambio di segno: es. $-a$
*	moltiplicazione: es. $a*b$
/	divisione: es. a/b
//	quoziente: es. $a//b$
%	resto (o modulo): es. $a\%b$
+	addizione: es. $a+b$
-	sottrazione: es. $a-b$

Precedenza degli operatori nelle espressioni aritmetiche: si applicano le regole dell'algebra

- Le operazioni di uguale precedenza sono associative a sinistra
- L'elevamento a potenza è associativo a destra
 - $2^{**}3^{**}2$ equivale a $2^{**}9$ – $(2^{**}3)^{**}2$ equivale a $8^{**}2$

Le variabili

- Un *identificatore di variabile* o semplicemente *variabile* è un nome dato ad un valore

```
>>> s='pippo'
```

s da questo momento riferisce al valore 'pippo'

```
>>> s
```

```
produce 'pippo'
```

```
>>> t='pluto'
```

Operazione di
inizializzazione

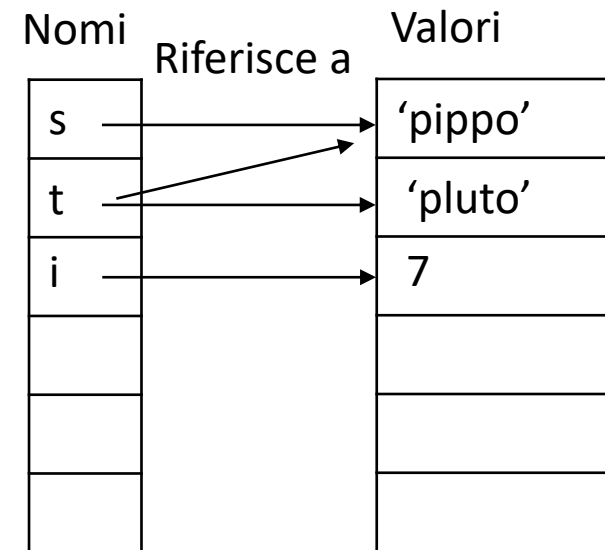
```
>>> i=7
```

```
>>> t=s
```

Operazione di
assegnamento

Quando si assegna un valore ad una variabile si effettua una operazione di *assegnamento*

La prima volta che si effettua questa operazione si parla di *inizializzazione*



Variabili

Quando in una istruzione (non di assegnamento) si incontra una variabile, l'interprete cerca il valore associato alla variabile e si parla di *riferimenti ad una variabile*

```
>>> a=32
```

```
>>> b=27
```

```
>>> a+b
```

```
59
```

Riferimenti ad a e b
Cerca il loro valore e calcola il
risultato

Nomi di Variabili

- Devono iniziare con una lettera o un _
- Possono contenere lettere, cifre e altri caratteri di sottolineatura
 - N1, pippo_7, io_sono_una_variabale
- Spesso si usa la forma cammellare
 - ioSonoUnaVariabile
- Alcune parole sono riservate e non possono essere usate come nomi
- Case sensitive
 - Pippo è diverso da pippo

Programmazione in Python

Lezione 2



Ripassiamo

File area.py

```
##  
# questo programma calcola l'area di un  
# triangolo avente base e altezza fissate  
  
base=3  
altezza=4  
print("l'area è",base*altezza/2) # calcolo e stampo l'area
```

commento utilizzato per
documentare il programma

L'area è 6.0

Inizializzazione
delle variabili
base e altezza

Commento di fine riga

Funzione che si occupa della
stampa

Espressione che calcola l'area
come numero con la virgola

Ripassiamo

File area.py

```
"""  
questo programma calcola l'area di un  
triangolo avente base e altezza fissate  
"""  
  
base=3  
altezza=4  
# calcolo e stampo l'area  
print("l'area è",base*altezza/2)
```

docstring: commento multilinea utilizzato per documentare e creare una guida su moduli, funzioni, oggetti.

L'area è 6.0

Inizializzazione
delle variabili
base e altezza

Commento di fine riga

Funzione che si occupa della
stampa

Espressione che calcola l'area
come numero con la virgola

Prendere dati in input

File area.py

```
"""  
questo programma calcola l'area di un  
triangolo avente base e altezza fissate  
"""  
  
base=3  
altezza=4  
# calcolo e stampo l'area  
print("l'area è",base*altezza/2)
```

Vogliamo generalizzare il problema:

Scrivere un programma che dati in input i valori di base e altezza, calcola l'area del triangolo

Prendere dati in input

```
>>> s = input('Inserisci il tuo nome: ')
```

```
Inserisci il tuo nome: simona
```

```
>>> s
```

```
'simona'
```

```
>>> print(s)
```

```
simona
```

Funzione input

- Riceve un argomento di tipo stringa
- Sintassi `input(stringa)`
- visualizza la stringa ricevuta
- Restituisce la stringa digitata da tastiera
- L'argomento è opzionale `input()`

Prendere dati interi in input

```
>>> numero1=int(input('Inserisci un numero:'))
```

```
Inserisci un numero: 38
```

```
>>> numero2=int(input('Inserisci un altro numero:'))
```

```
Inserisci un altro numero: -47
```

```
>>> numero1+numero2
```

```
-9
```

`int`

- Funzione per la conversione di una stringa in un numero intero

Funzione `input`

- Riceve un argomento di tipo stringa
- Sintassi `input(stringa)`
- visualizza la stringa ricevuta
- Restituisce la **stringa** digitata da tastiera
- L'argomento è opzionale `input()`

Prendere dati interi in input

- Che succede se dimentichiamo int prima di input?
 - I numeri inseriti vengono trattati come stringhe

```
>>> numero1=input('Inserisci un numero:')
```

```
Inserisci un numero: 38
```

```
>>> numero1
```

```
'38'
```

```
>>> numero2=input('Inserisci un altro numero:')
```

```
Inserisci un altro numero: -47
```

```
>>> numero1+numero2
```

```
'38-47'
```

Qui l'operatore + non è la somma tra interi, perché opera su due stringhe, e produce la concatenazione

Esercizio - l'area del triangolo

Scrivere un programma che dati in input i valori di base e altezza, calcola l'area del triangolo

File area.py

```
"""  
questo programma calcola l'area di un  
triangolo con base e altezza presi da input  
"""  
  
base=int(input('Inserisci la base: '))  
altezza=int(input("Inserisci l'altezza: "))  
# calcolo e stampo l'area  
print("L'area è",base*altezza/2)
```

Notate l'uso di '
oppure di " per
racchiudere le
stringhe

```
Inserisci la base: 7  
Inserisci l'altezza: 5  
L'area è 17.5
```

Altro sulle stringhe

- Sequenze di escape

- Usate per esprimere caratteri speciali
- Esempi `\n` (carattere di a capo), `\t` tabulazione, `\'` virgoletta semplice etc.

```
>>> stringa='le foglie dell\'albero'
```

```
>>> stringa
```

```
"le foglie dell'albero"
```

- Concatenamento di stringhe

- Operatore +

```
>>> "ciao " + "mondo!"
```

```
'ciao mondo'
```

- Operatore * concatena ripetizioni di una stringa

```
>>> 2*'ciao'
```

```
'ciaociao'
```

I caratteri nelle stringhe

- Una stringa è una sequenza di caratteri UNICODE
- Si può accedere ad ogni carattere tramite il suo indice
 - Nella stringa `'simona'` `s` è il carattere di indice 0, `i` di indice 1, e così via fino ad `a` di indice 5

```
>>> nome='simona'
```

```
>>> nome[3]
```

```
'o'
```

- Si può usare la funzione `len` per determinare la lunghezza di una stringa

```
>>> frase = input('Introduci una frase:')
```

```
Introduci una frase: la vita è bella
```

```
>>> lunghezza = len(frase)
```

```
>>> print("L'ultimo carattere di", frase, "è", frase[lunghezza-1])
```

```
L'ultimo carattere di la vita è bella è a
```


Aritmetica con tipi di dati misti

- Che succede se si scrive un'espressione aritmetica su dati i cui tipi sono diversi?
Esempio: somma tra un intero e un decimale

```
>>> -98.32 + 27  
-71.32
```

Il tipo meno generale `int` viene automaticamente convertito (temporaneamente) nel tipo più generale `float`

- Si possono effettuare conversioni di tipo utilizzando delle funzioni di conversione
 - `int(argomento)` e `float(argomento)` – L'argomento può essere un numero o una stringa

```
>>> int(3.5)  
3  
  
>>> int('-47') - 3  
-50
```

L'interprete controlla il valore degli operandi prima di effettuare un'operazione e segnala errore nel caso in cui un valore non sia appropriato

Le conversioni di tipo

- Funzioni di conversione verso tipi numerici `int(argomento)` e `float(argomento)`
 - L'argomento può essere un numero o una stringa

```
>>> float(3)
```

```
3.0
```

```
>>> float('-32.5')+2.4
```

```
-30.1
```

- Funzione di conversione verso tipo stringa `str(argomento)`
 - L'argomento può essere un valore qualsiasi

```
>>> str(34+15)
```

```
'49'
```

Funzioni e Moduli

- Una **funzione** è un blocco di codice, al quale viene assegnato un nome, che svolge un determinato compito
- Può ricevere degli argomenti, ossia dati necessari per svolgere il compito
 - Es. `print(valore1, ..., valoren)`
- Può restituire un valore
 - Es. `a=input('messaggio')`
 - Es. `x=round(3.6)`
- Alcune funzioni possono avere argomenti facoltativi e altri obbligatori
 - Es. `a=input()` #OK, non produce nessun messaggio
 - Es. `x=round()` #NO, errore
 - Es. `x=round(3.789,2)` #OK, x diventa 3.79

Si può consultare la documentazione di una funzione usando la funzione `help`
Es. `help(round)`

Funzioni e Moduli

A volte i programmatori usano
`from math import *`
Per evitare l'uso del qualificatore

- Python fornisce moltissimi funzioni utili, per gli scopi più vari
- Sono organizzate in librerie chiamate **Moduli**
 - Es. il modulo `math`
- Per essere utilizzate devono essere importate dai rispettivi moduli

- Si può importare tutto il modulo

```
>>> import math  
>>> x=math.sqrt(16)
```

Qui il nome della funzione va preceduto dal
qualificatore `math` seguito da `.`

- Si possono importare solo le funzioni di interesse

```
>>> from math import sqrt, log10  
>>> x=sqrt(16)  
>>> z=log10(100)
```

Notare la differenza
nell'invocazione di `sqrt`

Funzioni e Moduli

- Si può consultare l'elenco delle risorse contenuto in un modulo

```
>>> import math  
>>> dir(math)
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',  
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb',  
'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp',  
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma',  
'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt',  
'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'perm',  
'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan',  
'tanh', 'tau', 'trunc']
```

- Si può consultare la documentazione di tutto il modulo o di una singola risorsa

```
>>> help(math)  
>>> help(math.sqrt)
```

Programmazione in Python

Lezione 3



Prendere decisioni

- I programmi visti finora sono composti da istruzioni (anche molte) ma che devono essere eseguite tutte, una dopo l'altra
- A volte invece potrebbe diventare necessario prendere delle *decisioni* e scegliere tra due possibili azioni alternative
- Per questo è necessario esaminare e valutare se una determinata condizione è vera o falsa e in base al risultato ottenuto seguire una strada piuttosto che un'altra
- Es. Il voto preso all'esame è maggiore di 18?
 - Sì, stampa "complimenti! Hai superato l'esame"
 - No, stampa "peccato! Riprova al prossimo appello"

ISTRUZIONI DI
SELEZIONE

Le istruzioni di selezione - Rappresentazione grafica

Diagramma di flusso di una istruzione di selezione a una via

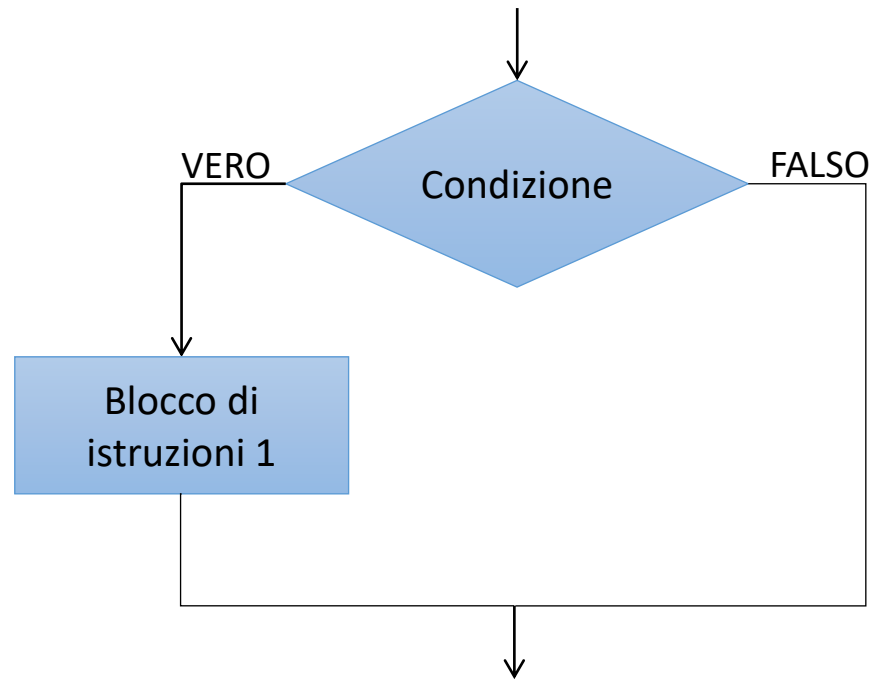
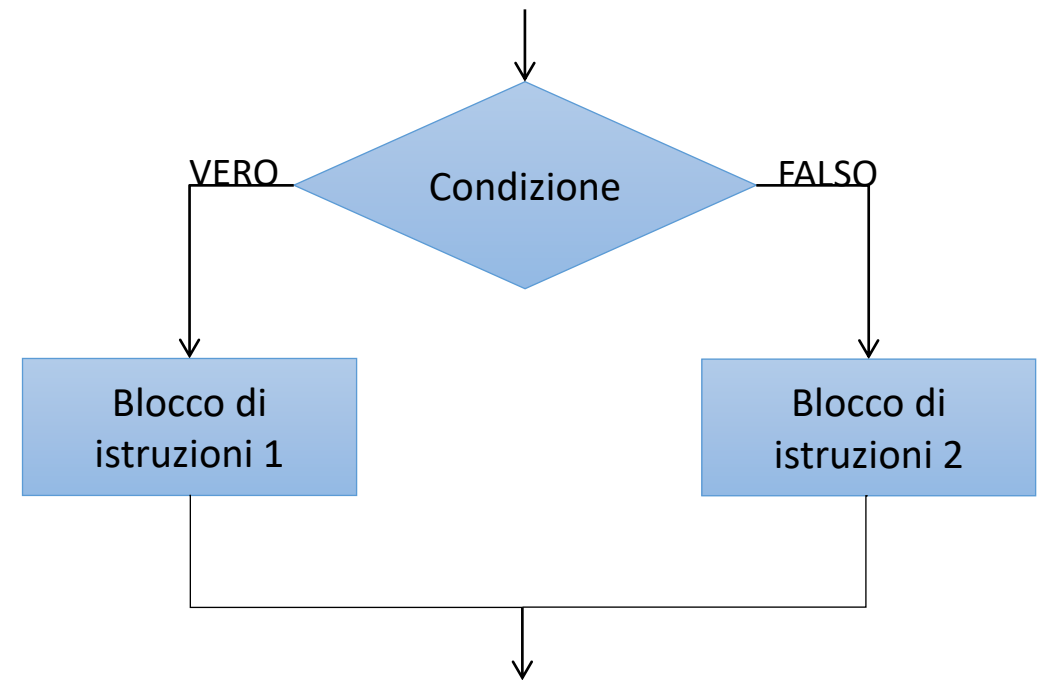


Diagramma di flusso di una istruzione di selezione a due vie



Le condizioni

Il termine deriva dal nome del
Matematico inglese George
Boole (XIX secolo)

Sono espressioni **booleane**, ossia espressioni il cui risultato può essere vero o falso (valore di verità)

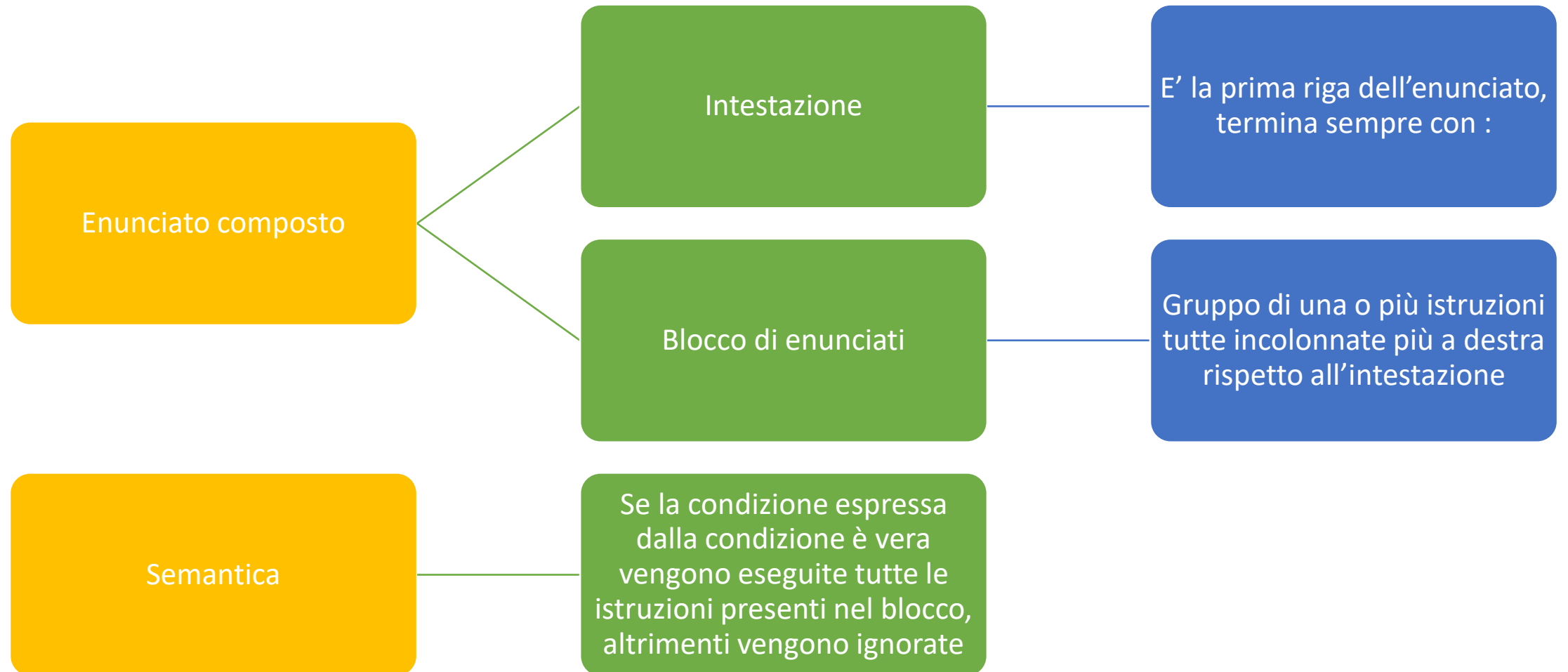
Spesso richiedono di confrontare valori tramite operatori relazionali

- $A > B$ operatore maggiore
- $A \geq B$ operatore maggiore uguale
- $A < B$ operatore minore
- $A \leq B$ operatore minore uguale
- $A == B$ operatore uguale
- $A != B$ operatore diverso

Possono essere utilizzati per confrontare sia numeri che stringhe

Hanno una precedenza più bassa degli operatori aritmetici

Prendere decisioni - L'enunciato `if`



L'enunciato `if`

Sintassi:

selezione a una via

```
if condizione:  
    blocco di istruzioni 1
```

selezione a due vie

```
if condizione:  
    blocco di istruzioni 1  
else:  
    blocco di istruzioni 2
```

Esempio:

```
if voto >= 18:  
    print('complimenti! Hai superato l'esame')  
else:  
    print('peccato! Riprova al prossimo appello')
```

Esercizio 1

- Scrivere un programma Python che dato in input un numero N, verifica se N è pari

File pari.py

```
"""
questo programma verifica se un numero N
preso da input è pari
"""

N=int(input('Inserisci un numero: '))
# verifico se N è un numero pari
if N%2 == 0:
    print(N,"è un numero pari")
else:
    print(N,"NON è un numero pari")
```

Esercizio 2

- Scrivere un programma Python che dati in input due numeri A e B, verifica se A è divisibile per B

File
divisibili.py

```
"""  
questo programma prende da input due interi  
A e B e verifica se A è divisibile per B  
"""  
  
A=int(input('Inserisci un numero: '))  
B=int(input('Inserisci un numero: '))  
# verifico se A è divisibile per B  
if A%B == 0:  
    print(A,"è divisibile per",B)  
else:  
    print(A,"non è divisibile per",B)
```

```
Inserisci un numero: 14  
Inserisci un numero: 2  
14 è divisibile per 2
```

Esercizio 2

- Scrivere un programma Python che dati in input due numeri A e B, verifica se A è divisibile per B

File
divisibili.py

```
"""  
questo programma prende da input due interi  
A e B e verifica se A è divisibile per B  
"""
```

```
A=int(input('Inserisci un numero: '))  
B=int(input('Inserisci un numero: '))  
# verifico se A è divisibile per B  
if A%B == 0:  
    print(A,"è divisibile per",B)  
else:  
    print(A,"non è divisibile per",B)
```

Che succede se
per B si introduce
0?

```
Inserisci un numero: 14  
Inserisci un numero: 0
```

Esercizio 2

- Scrivere un programma Python che dati in input due numeri A e B, verifica se A è divisibile per B

File
divisibili.py

```
"""  
questo programma verifica se due interi  
A e B sono divisibili tra loro  
"""
```

Errore in fase di esecuzione

```
A=14  
B=0  
#  
if  
    if A%B == 0:  
else:  
    print(A, "non è divisibile per", B)
```

Traceback (most recent call last):

File "C:/Users/Simona/Documents/Didattica/PythonProgs/divisibili.py",
line 8, in <module>

if A%B == 0:

ZeroDivisionError: integer division or modulo by zero

Che succede se

ro: 14

ro: 0

Esercizio 2

- Aggiungiamo un controllo
- Eseguiamo la divisione solo se B è diverso da 0

If innestati

```
"""
questo programma prende da input due interi
A e B e verifica se A è divisibile per B
"""

A=int(input('Inserisci un numero: '))
B=int(input('Inserisci un numero: '))
if B != 0:
    # verifico se A è divisibile per B
    if A%B == 0:
        print(A,"è divisibile per",B)
    else:
        print(A,"non è divisibile per",B)
else:
    print('non è possibile dividere per 0')
```

File
divisibili.py

```
Inserisci un numero: 14
Inserisci un numero: 0
non è possibile dividere per 0
```


If innestati

- All'interno di un enunciato `if` si può inserire un altro enunciato `if`
- Si parla di **if innestati** o **annidati**

```
A=int(input('Inserisci un numero: '))
B=int(input('Inserisci un numero: '))
if B != 0:
    # verifico se A è divisibile per B
    if A%B == 0:
        print(A,"è divisibile per",B)
    else:
        print(A,"non è divisibile per",B)
else:
    print('non è possibile dividere per 0')
```



Occhio
all'indentazione

If a più vie

- A volte è necessario prendere decisioni tra più possibili alternative
- Esempio: convertire un voto da 0 a 10 in giudizio
 - Se il voto è minori di 5 stampare insufficiente,
 - Altrimenti, se il voto è 6 stampare sufficiente,
 - Altrimenti, se il voto è 7 stampare discreto,
 - Altrimenti, se il voto è 8 stampare buono,
 - Altrimenti, se il voto è 9 stampare distinto,
 - Altrimenti, stampare ottimo

If a più vie

```
voto=int(input('Inserisci il voto (un numero tra 0 e 10)'))
if voto<=5:
    print('Insufficiente')
else:
    if voto==6:
        print('Sufficiente')
    else:
        if voto==7:
            print('Discreto')
        else:
            if voto==8:
                print('Buono')
            else:
                if voto==9:
                    print('Distinto')
                else:
                    print('Ottimo')
```

Una lunga catena di if-else

If a più vie

```
voto=int(input('Inserisci il voto (un numero tra 0 e 10)'))
if voto<=5:
    print('Insufficiente')
elif voto==6:
    print('Sufficiente')
elif voto==7:
    print('Discreto')
elif voto==8:
    print('Buono')
elif voto==9:
    print('Distinto')
else:
    print('Ottimo')
```

Si può riscrivere in maniera
più compatta utilizzando un if
a più vie

If a più vie

```
voto=int(input('Inserisci il voto (un numero tra 0 e 10)'))
```

```
if voto < 5:
```

```
p
```

```
elif voto < 7:
```

```
p
```

```
elif voto < 9:
```

```
p
```

```
elif voto < 10:
```

```
p
```

```
elif voto < 11:
```

```
p
```

```
else:
```

```
p
```

Sintassi:

```
if condizione 1:
```

```
    blocco di istruzioni 1
```

```
elif condizione 2:
```

```
    blocco di istruzioni 2
```

```
...
```

```
elif condizione n:
```

```
    blocco di istruzioni n
```

```
else:
```

```
    blocco di istruzioni di default
```

Si può riscrivere in maniera
più compatta utilizzando un if
a più vie

Le espressioni booleane composte

- Due o più espressioni booleane possono essere combinate per ottenere una **espressione booleana composta**
- Si utilizzano gli operatori logici `and`, `or` e `not`
- `and` e `or` sono operatori binari
 - `A and B`
 - L'espressione composta è vera se sia A che B sono vere
 - `A or B`
 - L'espressione composta è vera se almeno una tra A e B è vera
- `not` è un operatore unario
 - `not A`
 - L'espressione composta è vera se A è falso
- Hanno una precedenza più bassa degli operatori di confronto

Tavole di verità per gli operatori and, or e not

A	B	A and B
Vero	Vero	Vero
Vero	Falso	Falso
Falso	Vero	Falso
Falso	Falso	Falso

A	B	A or B
Vero	Vero	Vero
Vero	Falso	Vero
Falso	Vero	Vero
Falso	Falso	Falso

A	Not A
Vero	Falso
Falso	Vero

Esempio:

Si noti l'utilizzo dell'operatore `or`

```
voto=int(input('Inserisci il voto '))
if voto < 0 or voto > 30:
    print('voto non valido')
elif voto >= 18:
    print('Esame Superato')
else:
    print('Bocciato')
```


Valutazione di Cortocircuito

Le espressioni composte vengono valutate da sinistra verso destra

A volte è possibile conoscere il valore di verità dell'espressione composta prima che tutte le sue componenti siano state valutate

Nella valutazione di $A \text{ and } B$, viene valutata prima A . Se A è falsa, la valutazione di B non viene effettuata perché l'espressione composta è certamente falsa.

Analogamente, nella valutazione di $A \text{ or } B$, viene valutata prima A . Se A è vera, la valutazione di B non viene effettuata perché l'espressione composta è certamente vera.

Valutazione di Cortocircuito

- Il cortocircuito può essere utile in molti casi
- Ad esempio

```
if n != 0 and m%n == 0:  
    print('m è multiplo di n')
```

Se n è pari a 0 la prima parte della condizione è falsa e la seconda viene saltata evitando così la divisione per 0

Il tipo bool

- E' possibile assegnare il risultato di una espressione booleana ad una variabile.

```
>>> n=int(input('inserisci un numero: '))
```

```
Inserisci un numero: 38
```

```
>>> pari = n%2==0
```

```
>>> pari
```

```
True
```

- La variabile `pari` è una variabile booleana
 - può assumere solo due valori: `False` e `True`
- Il tipo corrispondente è `bool`

Programmazione in Python

Lezione 4



Esercizio – il massimo tra tre interi

Scrivere un programma che, dati in input tre numeri interi, determina e stampa il valore più grande

File
massimo.py

```
a=int(input("inserisci il primo numero "))
b=int(input("inserisci il secondo numero "))
c=int(input("inserisci il terzo numero "))

if a>b:
    if a>c:
        print("Il massimo è",a)
    else:
        print("Il massimo è",c)
elif b>c:
    print("Il massimo è",b)
else:
    print("Il massimo è",c)
```

Versione 1
Notate l'uso di if
innestati e if a più
vie

Esercizio – il massimo tra tre interi

Scrivere un programma che, dati in input tre numeri interi, determina e stampa il valore più grande

File
massimo.py

```
a=int(input("inserisci il primo numero "))
b=int(input("inserisci il secondo numero "))
c=int(input("inserisci il terzo numero "))

massimo=a
if massimo<b:
    massimo=b
if massimo<c:
    massimo=c

print("Il massimo è",massimo)
```

Versione 2
Notate l'uso della
variabile massimo e
i due if

Esercizio – il massimo tra quattro interi

Cosa cambia se gli interi sono 4?

File
massimo.py

```
a=int(input("inserisci il primo numero "))
b=int(input("inserisci il secondo numero "))
c=int(input("inserisci il terzo numero "))
d=int(input("inserisci il quarto numero "))

massimo=a
if massimo<b:
    massimo=b
if massimo<c:
    massimo=c
if massimo<d:
    massimo=d

print("Il massimo è",massimo)
```

4 variabili e 3 if

E se fossero 5?
Oppure 100?
E se fossero N?

Esercizio – il massimo tra N interi

- Scrivere un programma che, preso da input un numero intero N e altri N numeri interi, determina e stampa il valore più grande
- Occorre generalizzare il metodo risolutivo dell'esercizio precedente
 - Possiamo avere una variabile per ogni numero in input?
 - Possiamo avere un enunciato if per ogni numero in input?
 - Quali problemi vedete?
- E' possibile individuare un insieme di operazioni che si ripetono
 - Ad, esempio, potrei riscrivere il programma che calcola il massimo tra 4 numeri, usando una sola variabile, e ripetendo alcune istruzioni più volte

Esercizio – il massimo tra 4 interi

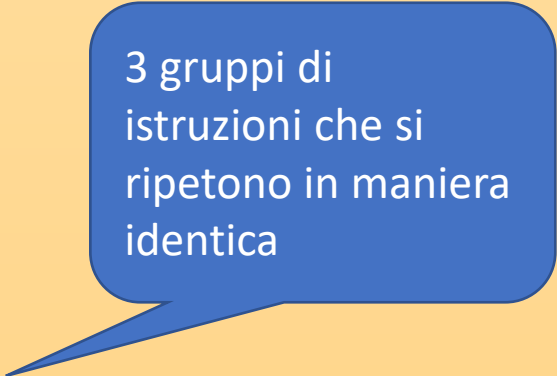
```
massimo=int(input("inserisci un numero "))

a=int(input("inserisci un numero "))
if massimo<a:
    massimo=a

a=int(input("inserisci un numero "))
if massimo<a:
    massimo=a

a=int(input("inserisci un numero "))
if massimo<a:
    massimo=a

print("Il massimo è",massimo)
```

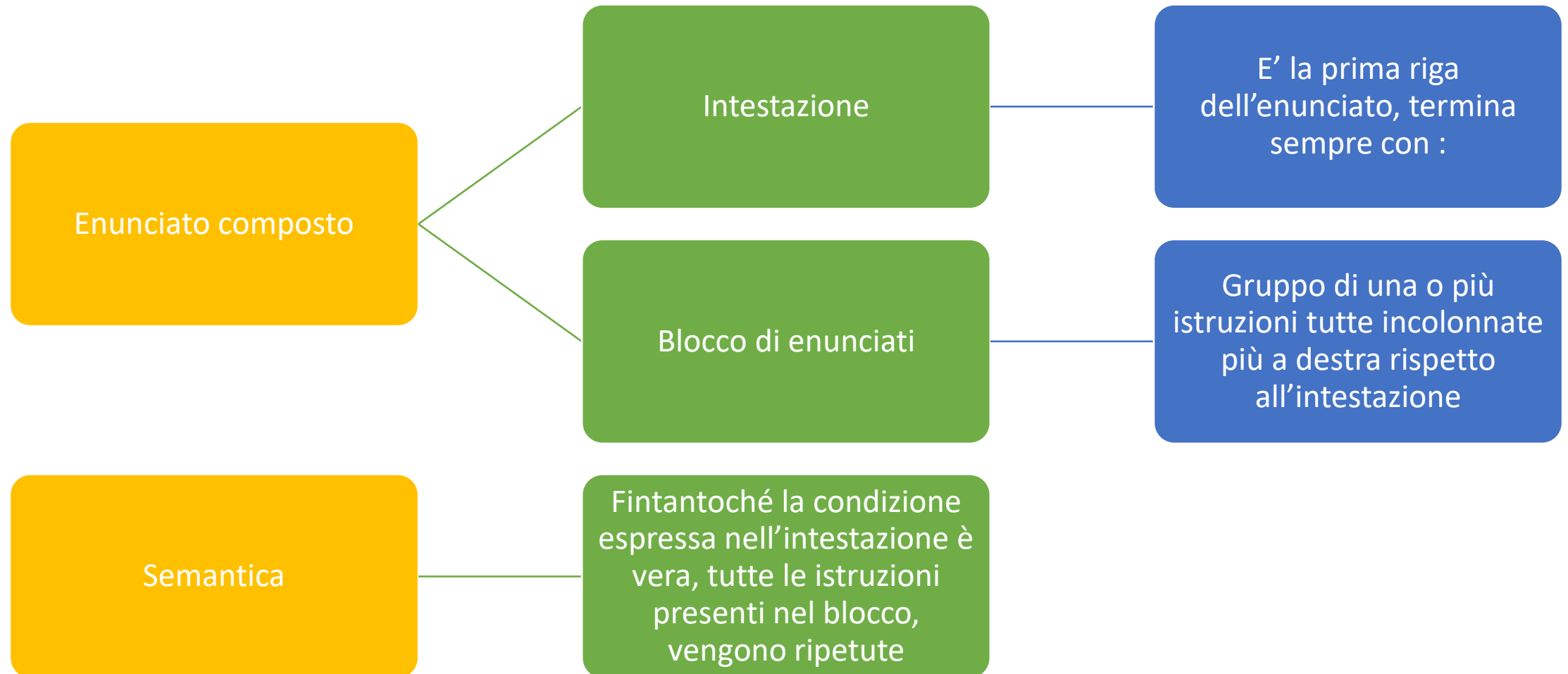


3 gruppi di istruzioni che si ripetono in maniera identica

L'iterazione

- Per ripetere una o più azioni un certo numero di volte, si utilizzano gli enunciati di **Iterazione**, anche detti **cicli**
- **Iterazione definita**
 - le azioni vengono ripetute un determinato numero di volte
 - Es. ripeti le azioni N volte
- **Iterazione indefinita**
 - le azioni vengono ripetute fintantoché durante l'esecuzione una certa condizione è vera
 - Es. ripeti finché numero > 0 dove numero cambia durante l'esecuzione
- Enunciati di Iterazione
 - `for`
 - `while`

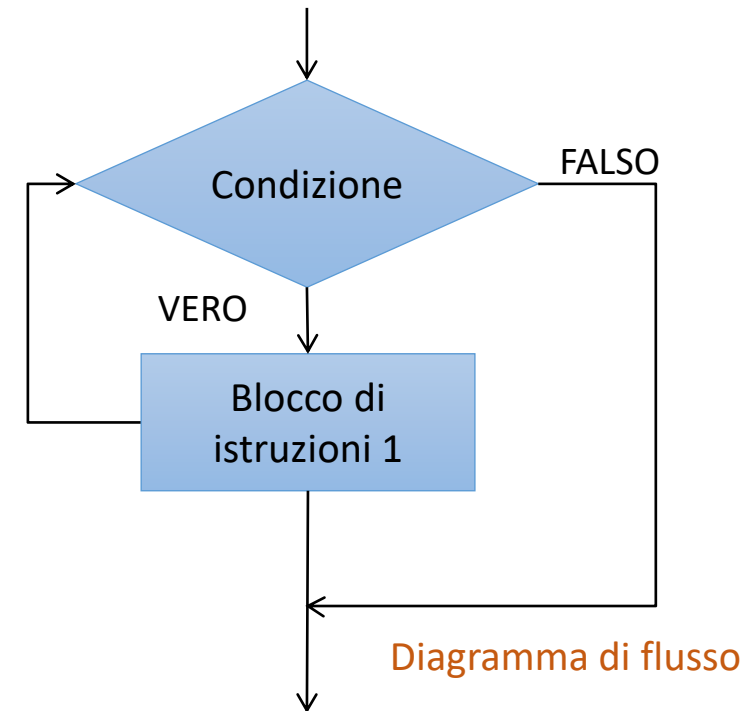
Ripetere istruzioni - L'enunciato `while`



Il ciclo while

Sintassi:

```
while condizione:  
    blocco di istruzioni
```



Attenzione tra le istruzioni nel blocco ce ne devono essere alcune che aggiornano i dati coinvolti nella condizione in modo che prima o poi la condizione diventa falsa e si può interrompere la ripetizione.

Un ciclo progettato male può portare ad un numero di iterazioni «teoricamente»
infinito

Esercizio – il massimo tra N interi

```
N=int(input("Quanti interi vuoi confrontare "))
massimo=int(input("inserisci il primo numero "))

cont=1
while cont < N:
    a = int(input("inserisci un altro numero "))
    if massimo<a:
        massimo=a
    cont=cont+1

print("Il massimo è",massimo)
```

La condizione del ciclo dipende dal valore della variabile cont

La variabile cont viene aggiornata all'interno del ciclo

Le istruzioni nel blocco vengono ripetute fino a quando cont non raggiunge N

Ciclo regolato da contatore

Ciclo regolato da contatore

- L'esempio precedente mostra l'utilizzo di un ciclo while controllato da una variabile **contatore**
- Questo accade quando le istruzioni devono essere ripetute un determinato numero di volte e quindi è necessario contare le iterazioni (**iterazione definita**)

```
cont=1
while cont <= 10:
    print(cont,end=' ')
    cont=cont+1
```

Stampa

1 2 3 4 5 6 7 8 9 10

Inizializzazione
del contatore

Condizione che dipende
dal valore del contatore

Aggiornamento
del contatore

Esercizi

- Scrivere un programma che stampa la somma di N numeri presi da input
- Scrivere un programma che stampa la media di N numeri presi da input
- Scrivere un programma che presi da input N numeri interi stampa la somma di tutti i numeri positivi e la somma di quelli negativi

Ciclo controllato da un valore sentinella

Esercizio: Scrivere un programma che legga da input una sequenza di numeri interi e ne stampi la somma. La lettura dei numeri in input deve andare avanti fino a quando non viene introdotto il valore 0.

```
somma=0
a=int(input("inserisci un numero (0 per terminare): "))
while a!=0:
    somma=somma+a
    a=int(input("inserisci un numero (0 per terminare): "))
print("La somma è",somma)
```

Notare il punto in cui
il primo valore della
sequenza viene
preso da input

0 è il valore ***sentinella***

Serve per indicare che la sequenza in input termina

```
inserisci un numero (0 per terminare): 10
inserisci un numero (0 per terminare): -15
inserisci un numero (0 per terminare): 15
inserisci un numero (0 per terminare): 0
La somma è 10
```


Ciclo controllato da un evento

- Le istruzioni nel blocco vengono ripetute fino a quando non si verifica un determinato evento (iterazione indefinita)
 - Non si conosce a priori il numero di iterazioni che saranno effettuate
 - Ad esempio, un ciclo regolato da un valore sentinella è un caso di iterazione indefinita



Esercizio: Algoritmo di Euclide per il calcolo del MCD tra due interi N ed M

Ripetere le seguenti istruzioni finchè M è diverso da 0

- Calcolare il resto della divisione tra N ed M
- Sostituire N con M
- Sostituire M con il resto

Il MCD sarà dato dall'ultimo valore di N

```
N=int(input("introduci un numero intero"))
M=int(input("introduci un altro intero"))
while M!= 0:
    resto=N%M
    N=M
    M=resto
print("Il MCD è:",N)
```

Hand Tracing

- Nella fase di sviluppo di un programma, per verificarne il corretto funzionamento, si può utilizzare la tecnica di Hand-Tracing
- Si esegue mentalmente una istruzione dopo l'altra e si tiene traccia del valore iniziale delle variabili e di come queste cambino durante l'esecuzione

Esercizi

- Scrivere un programma che presa da input una sequenza di numeri terminata da un valore <0 stampa la somma dei numeri pari
- Scrivere un programma che presa da input una sequenza di numeri terminata da un valore <0 stampa "almeno un numero pari" se nella sequenza è contenuto almeno un numero pari
- Scrivere un programma che dato in input un numero stampa le sue cifre al contrario

Programmazione in Python

Lezione 5



Il ciclo for

- Molto utile quando le istruzioni devono essere ripetute un determinato numero di volte
 - Ad esempio, come nei cicli regolati da contatore
- L'utilizzo del for può essere legato all'utilizzo della funzione **range**
- range genera una sequenza di numeri interi
- Può essere invocata con uno, due oppure tre argomenti
 - con un argomento Y, genera tutti i numeri interi da 0 a Y-1
 - range(4) genera 0 1 2 3
 - Con due argomenti X e Y, genera tutti i numeri compresi tra X e Y-1
 - range(1,4) genera 1 2 3
 - Con tre argomenti X, Y e P genera valori compresi tra X e Y-1 con un incremento di P tra un valore e l'altro
 - range(1,4,2) genera 1 3
 - range(4,0,-1) genera i valori tra 4 e 1 in ordine decrescente 4 3 2 1

Il ciclo for

- Sintassi

```
for variabile in range(...):  
    blocco di istruzioni
```

- Ad ogni iterazione, la variabile indicata nell'intestazione assume uno dei valori indicati da range, in ordine dal primo all'ultimo
- Esempi: Stampa dei numeri da 1 a 10 con while (sinistra) e for(destra)

```
cont=1  
while cont <= 10:  
    print(cont,end=' ')  
    cont=cont+1
```

```
for cont in range(1,11):  
    print(cont,end=' ')
```

Notare la
compattezza
del for

Esempio – somma di N numeri con ciclo `for`

```
N=int(input("Quanti interi vuoi sommare "))  
  
somma=0  
  
for cont in range(N):  
    somma=somma+int(input("Inserisci un numero"))  
  
print("La somma è",somma)
```

La variabile `cont`
assumerà tutti i
valori da 0 ad N-1

Operatori di assegnamento incrementale

E' possibile combinare l'operatore di assegnamento con operatori aritmetici e di concatenamento

```
>>> a=3
```

```
>>>a+=2
```

Sta per a=a+2

```
>>>a
```

```
5
```

```
>>> corso='fondamenti di'
```

```
>>> corso+=' programmazione 1'
```

```
>>> corso
```

```
fondamenti di programmazione 1
```

Esempi:

*=

+=

%=

-=

Numeri casuali

- A volte, specialmente nei giochi, è richiesto la gestione della **casualità**
 - Es. lancio dei dadi, lancio di una moneta, ecc.
- La casualità può essere **SIMULATA** nella programmazione e spesso i linguaggi forniscono opportune risorse per gestirle
- Python fornisce una serie di funzionalità disponibili nel modulo `random`
- Ad esempio, la funzione `randint` può essere usata per generare numeri interi pseudocasuali
 - Riceve due argomenti `a` e `b` e restituisce un numero pseudocasuale compreso tra `a` e `b` (estremi inclusi)

```
>>> randint(1,6)
```

```
5
```

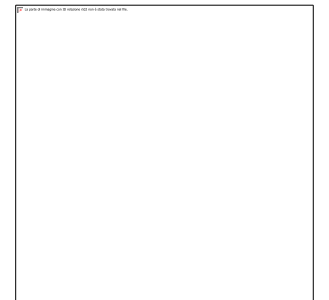
Simulare il lancio dei dadi

```
from random import randint

dado1=randint(1,6)
dado2=randint(1,6)

print(dado1, dado2)
```

4 3



Stringhe

- Una stringa, diversamente da un numero intero, è una struttura dati
- è composta da una sequenza di caratteri

```
>>> frase="Natale si avvicina"
```

- La funzione `len` può essere utilizzata per conoscere la sua lunghezza

```
>>>len(frase) 18
```

- I caratteri sono numerati in base alla loro posizione nella stringa, partendo dal primo che ha indice 0

N	a	t	a	l	e		s	i		a	v	v	i	c	i	n	a
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

- Si può accedere ad ogni carattere tramite l'operatore **subscript**

```
>>>frase[5] 'e'
```

Sottostringhe

- Sono porzioni della stringa di partenza
 - Ad esempio "tale si a" è una sottostringa della stringa "Natale si avvicina"
- Si possono estrarre sottostringhe utilizzando l'operatore di subscript e i due punti (:)

```
>>> frase="Natale si avvicina"
```

```
>>> frase[2:11]
```

```
'tale si a'
```

```
>>> frase[2:]
```

```
'tale si avvicina'
```

```
>>> frase[:11]
```

```
'Natale si a'
```

```
>>> frase[-4:]
```

```
'cina'
```

```
>>> frase[:-4]
```

```
'Natale si avvi'
```

Cercare una sottostringa

- Per verificare se una sottostringa è presente in una stringa data si può usare l'operatore `in` (lo stesso che si usa nel `for`)

```
frase="Natale si avvicina"  
daCercare="tale"  
if daCercare in frase:  
    print(daCercare,"è presente in",frase)
```

tale è presente in Natale si avvicina

startswith ed endswith

- Sono **metodi** che possono essere utilizzati per verificare se una stringa inizi o finisca con una determinata sottostringa.
- Vengono invocati su una stringa *S*, ricevono come argomento la sottostringa da cercare *SS* e restituiscono un valore booleano
 - Esempio:

```
>>> presente = s.endswith(SS)
```

```
frase = "Natale si avvicina"  
if frase.startswith("Natale"):  
    print(frase, "comincia con", "Natale")
```

Si tratta di metodi
non di funzioni.
Qual è la
differenza?

Altri metodi per stringhe e sottostringhe

```
>>> stringa = "blablabla"  
>>> conta=stringa.count("bla")  
>>> print("bla è presente",conta,"volte")
```

```
bla è presente 3 volte
```

Il metodo count
conta il numero di
occorrenze di una
sottostringa

```
>>> primaOccorrenza=stringa.find("bla")  
>>> print("La prima occorrenza di bla parte dall'indice",primaOccorrenza)
```

```
La prima occorrenza di bla parte dall'indice 0
```

Il metodo find restituisce l'indice dal quale inizia la
prima occorrenza di una sottostringa
Se la sottostringa non è presente restituisce -1

Stringhe, lettere e cifre

- Le stringhe sono sequenze di caratteri e perciò possono contenere lettere dell'alfabeto oppure cifre.
- Esistono dei metodi per analizzare e riconoscere la presenza di lettere e cifre all'interno delle stringhe.
- Esempi:
 - `isalpha()` restituisce `true` se la stringa è non vuota e contiene solo lettere
 - `isdigit()` restituisce `true` se la stringa è non vuota e contiene solo cifre
 - `isalnum()` restituisce `true` se la stringa è non vuota e contiene solo lettere o cifre
 - `islower()` restituisce `true` se la stringa è non vuota e tutte le lettere che contiene sono minuscole
 - `isupper()` simile ad `islower()` ma controlla che le lettere siano tutte maiuscole

Programmazione in Python

Lezione 6



Problema: le temperature

Per semplicità si supponga che per ogni giorno venga indicato un solo valore e per il calcolo della media si usi la divisione intera

Data una sequenza di valori interi che rappresentano le temperature registrate in una città in un mese stampi:

- La temperatura media
- L'elenco dei giorni in cui la temperatura è stata più alta della media

Esempio: se le temperature fossero

-5, 2, 0, -3, 1, 2, **4**, -2, 0, 2, 3, 2, **4**, **6**, -1, 0, **4**, **6**, **7**, **6**, **10**, **13**, **12**, **6**, **5**, 2, -1, 0, **4**, **6**, -1

E' necessario analizzare tutti i dati almeno due volte, una volta per calcolare la media e una volta per identificare i giorni da stampare

I dati vanno memorizzati tutti

La media è 3
giorno 7: 4°
giorno 13: 4°
giorno 14: 6°
giorno 17: 4°
giorno 18: 6°
giorno 19: 7°
giorno 20: 6°
giorno 21: 10°
giorno 22: 13°
giorno 23: 12°
giorno 24: 6°
giorno 25: 5°
giorno 29: 4°
giorno 30: 6°

Memorizzare raccolte di dati: le liste

- **Struttura Dati:**

- raggruppa e organizza diversi dati in un'unità
- Es. Una stringa è una struttura dati che raggruppa sequenze di caratteri

- **Lista:**

- Struttura dati
- Consente di memorizzare raccolte di dati di qualsiasi tipo (**elementi**)
- Gli elementi sono disposti in ordine sequenziale
- Ogni elemento ha un indice che identifica la sua posizione (il primo ha indice pari a 0)

Liste

- Sequenza di dati separata da virgole, racchiusi tra parentesi quadre

- `[1, 18, 22, 44]`
- `["fondamenti", "analisi", "inglese", "economia"]`
- `[]` (lista vuota)

- Può essere assegnata ad una variabile

```
>>> esami = ["fondamenti", "analisi", "inglese", "economia"]  
>>> esami
```

```
['fondamenti', 'analisi', 'inglese', 'economia']
```

- Si può accedere agli elementi tramite la loro posizione (come nelle stringhe)

```
>>> esami[2]
```

```
'inglese'
```

ATTENZIONE

l'indice deve essere valido:
Se la lista contiene n elementi
l'indice deve essere compreso tra
0 e n-1

Liste

- Per conoscere il numero di elementi di una lista si può utilizzare la funzione `len`

```
>>> esami = ["fondamenti", "analisi", "inglese", "economia"]  
>>> len(esami)
```

```
4
```

- Si possono estrarre parti di lista utilizzando l'operatore di subscript e i due punti (:)

```
>>> esami[1:3]
```

```
['analisi', 'inglese']
```

- Si possono concatenare liste con gli operatori `+` e `*`
- Si possono confrontare con l'operatore `==` (si noti che la lista `[1,3]` è diversa da `[3,1]`)
- Si possono creare liste di liste

```
>>> esami2 = ["linguaggi", "fisica", "architettura"]  
>>> [esami, esami2]
```

```
[['fondamenti', 'analisi', 'inglese', 'economia'], ['linguaggi',  
'fisica', 'architettura']]
```

Liste

Liste e stringhe hanno alcune similitudini ma anche alcune differenze:

La lista e' una struttura dati **Mutable** : struttura e contenuto possono cambiare

- È possibile inserire nuovi elementi
- È possibile eliminare elementi
- È possibile sostituire gli elementi già esistenti

La stringa è una struttura dati **Immutable**: struttura e contenuto **NON** possono cambiare

Liste

```
>>> esami = ["fondamenti", "analisi", "inglese", "economia"]  
>>> esami[3]="fisica"  
>>> esami
```

```
['fondamenti', 'analisi', 'inglese', 'fisica']
```

```
>>> parola "economia"  
>>> parola[0]="E"
```

Traceback (most recent call last):

File "<pyshell#42>", line 1, in <module>

parola[0]="E"

TypeError: 'str' object does not support item assignment

Metodi di lista per inserimento ed eliminazione

- `L.append(elemento)`
 - Inserisce un elemento alla fine della lista
- `L.insert(indice, elemento)`
 - Inserisce un elemento alla posizione corrispondente ad indice, se indice è minore della lunghezza di L, altrimenti inserisce alla fine della lista

```
>>> L=[1,3,5]
```

```
>>> L.insert(2,4)
```

```
[1,3,4,5]
```

- `L.pop()`
 - Elimina e restituisce l'elemento alla fine della lista
- `L.pop(indice)`
 - Elimina e restituisce l'elemento alla posizione corrispondente ad indice

La lista non deve essere vuota

indice deve essere un valore tra 0
len(L)

Scansione degli elementi della lista

- Si possono scandire gli elementi di una lista con un ciclo
 - Accedendo ad ogni elemento tramite il proprio indice

```
>>> L=[1,5,7,8,2,-9]
>>> for i in range (len(L)):
    print(i , L[i])
```

0	1
1	5
2	7
3	8
4	2
5	-9

- Accedendo direttamente agli elementi

```
>>> for elemento in L:
    print(elemento)
```

1
5
7
8
2
-9

Problema: le temperature

```
ngiorni=31
elenco=[]
for i in range(0,ngiorni):
    elenco.append(int(input()))

somma=0
for i in range(0,ngiorni):
    somma+=elenco[i]

media=somma//ngiorni

print("La media è", media)
for i in range(0,ngiorni):
    if elenco[i] > media:
        print("giorno ", i+1, ": ", elenco[i], "°", sep='')
```

Possiamo evitare un ciclo for

Problema: le temperature

```
ngiorni=31
elenco=[]
somma=0

for i in range(0,ngiorni):
    elenco.append(int(input()))
    somma+=elenco[i]

media=somma//ngiorni

print("La media è", media)
for i in range(0,ngiorni):
    if elenco[i] > media:
        print("giorno ", i+1, ": ", elenco[i], "°", sep='')

```

Programmazione in Python

Lezione 7



Ricerca di un elemento in una lista

- Per verificare se un elemento è presente in una lista si può utilizzare l'operatore `in`

```
>>> lista=[90,2,9,4,9]
>>> if 9 in lista:
    print("il 9 è presente")
```

- Se è necessario conoscere la posizione in cui l'elemento è presente, si può utilizzare il metodo `index`

- Restituisce la posizione della prima occorrenza
- Può essere invocato se l'elemento è presente, altrimenti si ha un'eccezione
- Riceve come argomento l'elemento da cercare

```
>>> if 9 in lista:
    print("il 9 è presente in posizione ", lista.index(9))
```

il 9 è presente in posizione 2

Si può utilizzare in un ciclo per conoscere le posizioni di tutte le occorrenze

- Può ricevere l'indice della posizione da cui iniziare la ricerca

```
>>> print("il 9 è presente in posizione ", lista.index(9, 3))
```

il 9 è presente in posizione 4

Attenzione che qui almeno un elemento deve essere presente dopo la posizione 3

Eliminare un elemento

- Per rimuovere un elemento data la sua posizione si usa il metodo `pop`
 - `lista.pop()` rimuove l'ultimo
 - `lista.pop(x)` rimuove l'elemento in posizione `x`
- Si può rimuovere un elemento in base al suo valore, senza conoscerne prima il suo valore

- Si usa il metodo `remove`

```
>>> lista=[1,4,6,4]
>>> if 4 in lista:
        lista.remove(4)
>>> lista
```

```
[1, 6, 4]
```

Attenzione che anche qui
l'elemento da rimuovere deve
essere presente

Riferimenti ad una lista (alias)

```
>>> lista1=[47,92,0]
```

- La variabile `lista1` contiene un riferimento alla lista

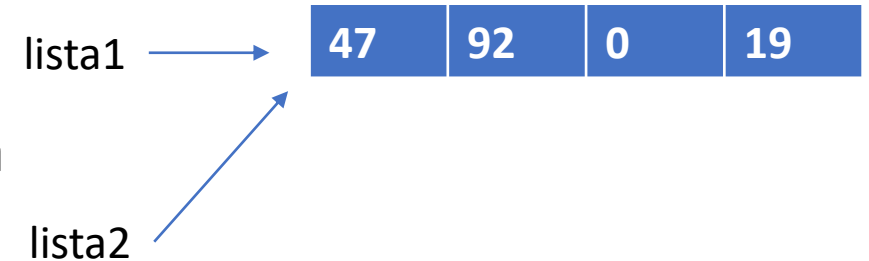
```
>>> lista2= lista1
```

- Ora anche la variabile `lista2` contiene un riferimento alla lista
- Si dice che `lista2` è un *alias* per `lista1`

```
>>> lista2.append(19)
```

```
>>> lista1
```

```
[47, 92, 0, 19]
```



Copia di una lista

```
>>> L=[1,2,3]
```

- Si vuole creare una nuova lista C che contiene gli stessi elementi di L

Metodo1:

```
>>> C=[]  
>>> for elemento in L:  
        C.append(elemento)
```

C inizialmente è vuota e poi vengono inseriti uno alla volta gli elementi di L

Metodo2:

```
>>> C=L[:]
```

Si assegna a C la lista estratta da L

Metodo3:

- Si usa la funzione `list`

La funzione `list`

- Crea una nuova lista concatenando tutti gli elementi di una sequenza

```
>>> L=[1,2,3]
>>> C=list(L)
>>> C
```

```
[1, 2, 3]
```

Qui `list` concatena la sequenza di elementi presenti nella lista `L`

```
>>> nome="Simona"
>>> caratteri=list(nome)
>>> caratteri
```

```
['s', 'i', 'm', 'o', 'n', 'a']
```

Qui `list` concatena la sequenza di caratteri presenti nella stringa `nome`

```
>>> listanumeri=list(range(10,31))
>>> listanumeri
```

```
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]
```

Qui `list` concatena la sequenza di numeri generati dalla funzione `range`

Identità degli oggetti ed Equivalenza Strutturale

- L'operatore `==` restituisce `True` se le liste sono una un alias dell'altra
 - Si parla di **Identità Strutturale**
- L'operatore `==` restituisce `True` se due liste differenti contengono gli stessi oggetti
 - Si parla di **Equivalenza Strutturale**
- Si può verificare l'identità strutturale tramite l'operatore `is`

```
>>> lista1=["fond","analisi"]
```

```
>>> lista2=["fond","analisi"]
```

```
>>> lista1==lista2
```

```
True
```

```
>>> lista2 is lista1
```

```
False
```

Funzioni per le liste

- Calcolare la somma degli elementi di una lista
 - Si può usare la funzione `sum` che riceve la lista e restituisce la somma dei suoi argomenti
 - Gli elementi devono essere tutti numeri

```
>>> L=[1.5, -2, -3]
```

```
>>> somma=sum(L)
```

```
>>> somma
```

```
-3.5
```

Sommare o concatenare gli elementi in una lista

```
L=[1.5,-2,-3]
somma=0.0
for element in L:
    somma+=element
print(somma)
```

-3.5

```
parole=['ciao','luce','scivolo']
concatenazione=""
for stringa in parole:
    concatenazione+= stringa
print(concatenazione)
```

ciaolucescivolo

Funzioni per le liste

- Calcolare il massimo o il minimo degli elementi di una lista
 - Si possono usare le funzioni `max` e `min` che ricevono la lista e restituiscono il massimo e il minimo degli elementi, rispettivamente
 - Gli elementi devono essere tutti numeri o tutte stringhe

```
>>> L=[1.5,-2,-3]
```

```
>>> max(L)
```

```
1.5
```

```
>>> min(L)
```

```
-3
```

```
>>> stringhe=["casa blu","albero","macchina gialla", "novembre"]
```

```
>>> max(stringhe)
```

```
'novembre'
```

```
>>> min(stringhe)
```

```
'albero'
```

Massimo e Minimo

```
stringhe=["casa blu","albero","macchina gialla", "novembre"]
max=stringhe[0]

for i in range(1,len(stringhe)):
    if stringhe[i]>max:
        max=stringhe[i]

print(max)
```

'novembre'

```
L=[1.5,-2,-3, 849, -81, 25.37, -91.2]
min=L[0]
for i in range(1,len(L)):
    if L[i]<min:
        min=L[i]
print(min)
```

'-91.2'

Programmazione in Python

Lezione 8



Cicli annidati

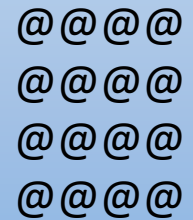
- A volte è necessario utilizzare un ciclo all'interno del blocco di istruzioni di un altro ciclo
- Si parla di cicli annidati (nested loop)
- Ad ogni iterazione del ciclo più esterno si eseguono tutte le iterazioni del ciclo più interno

```
for i in range(3):  
    for j in range(4):  
        print(j,end=" ")  
    print()
```

```
0 1 2 3  
0 1 2 3  
0 1 2 3
```

Esercizio

- Letto da input un numero intero positivo X, stampa un quadrato di lato X di simboli@
 - Esempio: se X è 4, il programma stampa



```
@@@@  
@@@@  
@@@@  
@@@@
```

```
X=int(input())  
  
for i in range(X):  
    for j in range(X):  
        print("@",end=' ')  
    print()
```

Esercizio

- Letto da input un numero intero positivo X, stampa un quadrato di lato X di numeri consecutivi come nell'esempio
 - Esempio: se X è 4, il programma stampa

```
1234
2345
3456
4567
```

```
X=int(input())

for i in range(X):
    for j in range(1,X+1):
        print(j+i,end=' ')
    print()
```

Esercizio

- Data una lista di interi terminata dal tappo 0 verificare se esistono due elementi uguali tra loro

```
elenco=[]
N=int(input())
while N!=0:
    elenco.append(N)
    N=int(input())

duplicati =False
for i in range(len(elenco)-1):
    for j in range(i+1,len(elenco)):
        if elenco[i] == elenco[j]:
            duplicati=True

if duplicati:
    print("Esistono elementi duplicati")
```

Notare l'uso dei cicli innestati:
Per ogni elemento della lista
Si controllano tutti gli elementi successivi

Tabelle / Matrici

- I dati vengono disposti in due dimensioni, righe e colonne
- Esempio

	Lun	Mar	Mer	Gio	Ven	Sab
8:30	Italiano	Arte	Matematica	Italiano	Tecnologia	Storia
9:30	Matematica	Italiano	Matematica	Inglese	Geografia	Storia
10:30	Storia	Italiano	Ed. Fisica	Italiano	Musica	Matematica
11:30	Geografia	Musica	Ed. Fisica	Matematica	Italiano	Arte
12:30	Inglese	Inglese	Tecnologia	Matematica	Inglese	Italiano

- In python si possono rappresentare tramite le liste
 - Si crea una lista, i cui elementi (le righe) sono delle liste

Tabelle / Matrici

```
orario=[  
    ["Italiano", "Arte", "Matematica","Italiano", "Tecnologia", "Storia"],  
    ["Matematica", "Italiano", "Matematica","Inglese", "Geografia", "Storia"],  
    ["Storia", "Italiano", "Ed. Fisica","Italiano", "Musica", "Matematica"],  
    ["Geografia", "Musica", "Ed. Fisica","Matematica", "Italiano", "Arte"],  
    ["Inglese", "Inglese", "Tecnologia","Matematica", "Inglese", "Italiano"]  
]
```

orario → 0	Italiano	Arte	Matematica	Italiano	Tecnologia	Storia
1	Matematica	Italiano	Matematica	Inglese	Geografia	Storia
2	Storia	Italiano	Ed. Fisica	Italiano	Musica	Matematica
3	Geografia	Musica	Ed. Fisica	Matematica	Italiano	Arte
4	Inglese	Inglese	Tecnologia	Matematica	Inglese	Italiano
	0	1	2	3	4	5

Accedere agli elementi di una matrice

- Si accede ad ogni elemento tramite due indici (riga e colonna)

```
>>> print(orario[1][3])
```

Inglese

- Per scandire una matrice si usa un ciclo annidato

```
for i in range(5):  
    for j in range(6):  
        print(orario[i][j], end='  ')  
    print()
```

Italiano Arte Matematica Italiano Tecnologia Storia
Matematica Italiano Matematica Inglese Geografia Storia
Storia Italiano Ed. Fisica Italiano Musica Matematica
Geografia Musica Ed. Fisica Matematica Italiano Arte
Inglese Inglese Tecnologia Matematica Inglese Italiano

A hand-drawn diagram of a 3x2 matrix, represented by a large pair of parentheses containing the numbers 1, 2, 3, 4, 5, and 6. The numbers are arranged in three rows and two columns: the first row contains 1 and 2, the second row contains 3 and 4, and the third row contains 5 and 6. Above the first column, there is a downward-pointing arrow, and above the second column, there is a downward-pointing arrow, indicating the row indices. To the right of the first row, there is a rightward-pointing arrow, indicating the column index.

Creazione di una tabella di dimensione $N \times M$

Per creare una tabella con M righe e N colonne di valori interi pari a 0

```
M=3  
N=4  
matrice=[]  
  
for i in range(M):  
    matrice.append([0]*N)
```

Si crea una lista vuota

Si inseriscono nella lista M righe

Ogni riga è una lista con N zero

Esercizio

- Stampare le lezioni del giovedì

```
orario=[  
    ["Italiano", "Arte", "Matematica","Italiano", "Tecnologia", "Storia"],  
    ["Matematica", "Italiano", "Matematica","Inglese", "Geografia", "Storia"],  
    ["Storia", "Italiano", "Ed. Fisica","Italiano", "Musica", "Matematica"],  
    ["Geografia", "Musica", "Ed. Fisica","Matematica", "Italiano", "Arte"],  
    ["Inglese", "Inglese", "Tecnologia","Matematica", "Inglese", "Italiano"]  
]
```

```
for i in range(5):  
    print(orario[i][3])
```

```
Italiano  
Inglese  
Italiano  
Matematica  
Matematica
```

Esercizio

- Data una matrice di interi di N righe e N colonne verificare se si tratta di una matrice diagonale.

una matrice diagonale è una matrice quadrata in cui solamente i valori della diagonale principale possono essere diversi da 0

Diagonale
Principale

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 81 \end{bmatrix}$$

Matrice Diagonale

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 81 \end{bmatrix}$$

Matrice Non Diagonale

Esercizio - Matrice Diagonale

```
N=int(input())

mat=[]
for i in range(N):
    mat.append([])
    for j in range(N):
        mat[i].append(int(input()))

diagonale=True
for i in range(N):
    for j in range(N):
        if i!=j and mat[i][j]!=0:
            diagonale=False
if diagonale:
    print("Matrice Diagonale")
```

Programmazione in Python

Lezione 9



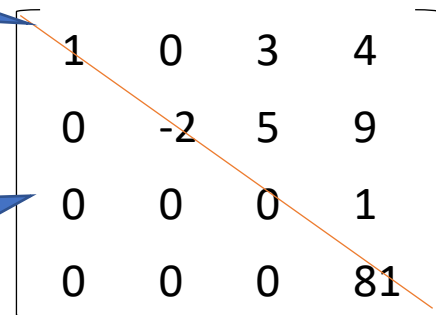
Esercizio

- Data una matrice di interi di N righe e N colonne verificare se si tratta di una matrice triangolare superiore.

Una matrice triangolare superiore è una matrice quadrata che ha tutti gli elementi nulli sotto la diagonale principale

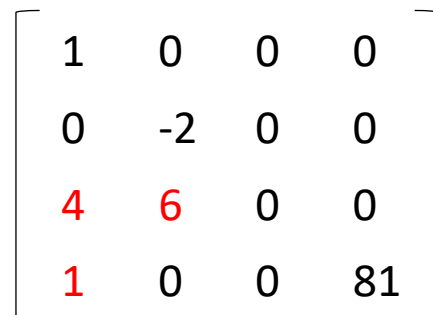
Diagonale
Principale

Elementi
nulli sotto la
diagonale



1	0	3	4
0	-2	5	9
0	0	0	1
0	0	0	81

Matrice Triangolare Superiore



1	0	0	0
0	-2	0	0
4	6	0	0
1	0	0	81

Matrice Non Triangolare

Esercizio - Matrice Triangolare Superiore – versione 1

```
N=int(input())

mat=[]
for i in range(N):
    mat.append([])
    for j in range(N):
        mat[i].append(int(input()))

triangolare=True
for i in range(N):
    for j in range(N):
        if j<i and mat[i][j]!=0:
            triangolare=False
if triangolare:
    print("Matrice triangolare superiore")
```

Esercizio - Matrice Triangolare Superiore – versione 2

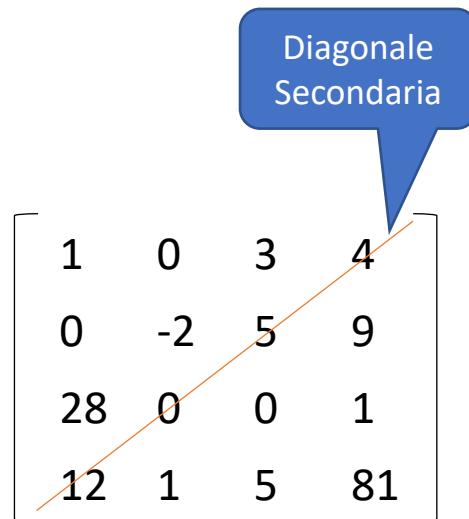
```
N=int(input())

mat=[]
for i in range(N):
    mat.append([])
    for j in range(N):
        mat[i].append(int(input()))

triangolare=True
for i in range(1,N):
    for j in range(0,i):
        if mat[i][j]!=0:
            triangolare=False
if triangolare:
    print("Matrice triangolare superiore")
```

Esercizio

- Data una matrice di interi di N righe e N colonne calcolare la somma degli elementi della diagonale secondaria.



1	0	3	4
0	-2	5	9
28	0	0	1
12	1	5	81

Somma 21 (12+0+5+4)

Cosa fa questo programma?

```
N1=int(input())
N2=int(input())

p1=True
p2=True
for i in range(2,N1//2+1):
    if N1%i==0:
        p1=False

for i in range(2,N2//2+1):
    if N2%i==0:
        p2=False

if p1 and p2 and (N1-N2 == 2 or N2-N1 == 2):
    print("SI")

else:
    print("NO")
```

Cosa fa questo programma?

```
N1=int(input())
N2=int(input())

if numeroPrimo(N1) and numeroPrimo(N2) and (N1-N2 == 2 or N2-N1 == 2):
    print("SI")
else:
    print("NO")
```

Cosa fa questo programma?

```
N1=int(input())
N2=int(input())

if numeroPrimo(N1) and numeroPrimo(N2) and valoreAssoluto(N1-N2)==2:
    print("SI") # sono primi gemelli
else:
    print("NO") # non sono primi gemelli
```

Cosa fa questo programma?

```
def numeroPrimo(N):  
    for i in range(2,N//2+1):  
        if N%i==0:  
            return False  
    return True  
  
def valoreAssoluto(N):  
    if N<0:  
        return -N  
    return N  
  
N1=int(input())  
N2=int(input())  
  
if numeroPrimo(N1) and numeroPrimo(N2) and valoreAssoluto(N1-N2):  
    print("SI") # sono primi gemelli  
else:  
    print("NO") # non sono primi gemelli
```

Cosa fa questo programma?

```
N1=int(input())
N2=int(input())

p1=True
p2=True
for i in range(2,N1//2+1):
    if N1%i==0:
        p1=False

for i in range(2,N2//2+1):
    if N2%i==0:
        p2=False

if p1 and p2 and (N1-N2 == 2 or N2-N1 == 2):
    print("SI")

else:
    print("NO")
```

Codice Ripetuto

Dettagli che complicano la lettura del codice

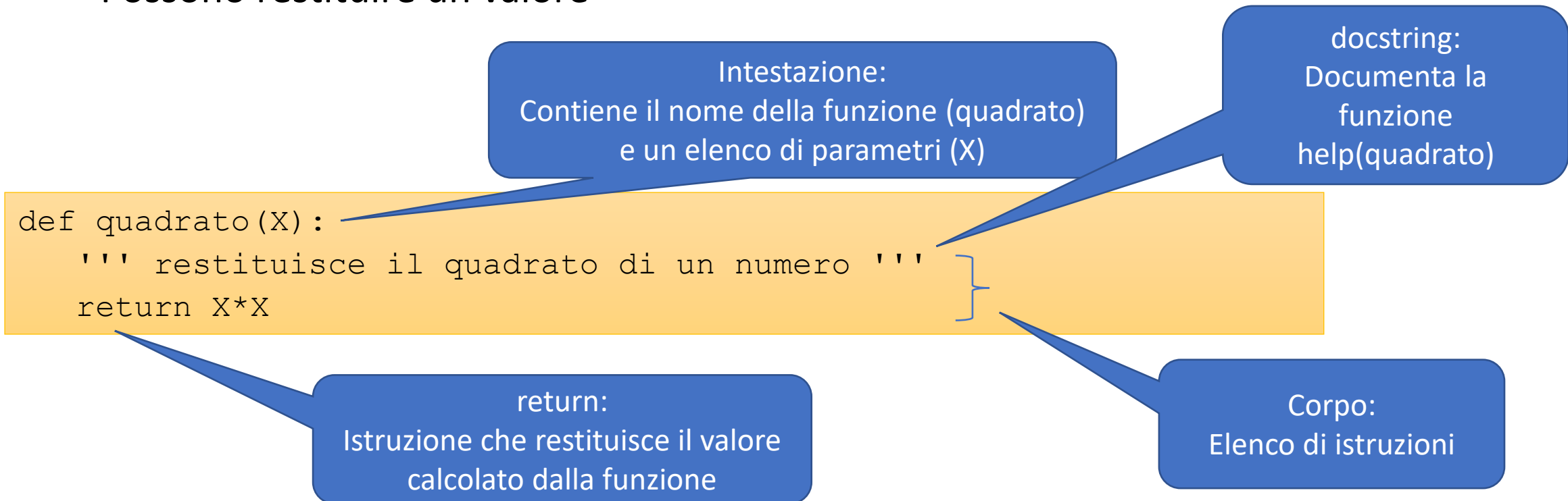
Progettare con le funzioni

Costituiscono un meccanismo di *astrazione* con molti vantaggi

- Consentono di nascondere i dettagli complessi
 - Es.: la verifica di numero primo è nascosta nella funzione
- ```
if numeroPrimo(N1) and numeroPrimo(N2) and valoreAssoluto(N1-N2)==2:
```
- Consentono di evitare codice ridondante o duplicato
  - Consentono di realizzare metodi generali per risolvere classi di problemi
    - Es.: Si specifica come verificare che un qualsiasi N sia primo e poi si applica alle istanze N1 e N2
  - Consentono di scomporre il problema in sotto-problemi da risolvere indipendentemente dal resto
    - Es.: Verificare che un numero è primo, calcolare il valore assoluto di un numero.
    - Le soluzioni dei sotto-problemi vengono poi composte, per ottenere la soluzione del problema di partenza

# Funzioni

- Sono sequenze di istruzioni con un nome
  - Possono ricevere argomenti
  - Possono restituire un valore



# Funzioni

- Sono sequenze di istruzioni con un nome
  - Possono ricevere argomenti
  - Possono restituire un valore

X si chiama *parametro formale*

```
def quadrato(X):
 ''' restituisce il quadrato del numero '''
 return X*X
```

Invocazione:

Si specifica il nome della funzione seguito dagli argomenti su cui la funzione deve essere invocata

```
N = int(input())
N2 = quadrato(N)
```

Il valore restituito dalla funzione viene usato nel programma

N si chiama *parametro attuale*



# Funzioni

- Sono sequenze di istruzioni con un nome
  - Possono ricevere argomenti
  - Possono restituire un valore
- Sintassi:

Se l'istruzione `return` manca, il controllo passa al chiamante dopo che è stata eseguita l'ultima istruzione del corpo e viene restituito il valore speciale `None`

`parametro1, ..., parametron` sono i ***parametri formali***

```
def nomedellafunzione (parametro1, ..., parametron):
 corpo
```

```
return espressione
```

Interrompe l'esecuzione della funzione restituendo al chiamante il valore di `espressione`

# Passaggio di parametri

- Quando si invoca la funzione vengono create delle variabili per i parametri formali
- I parametri formali vengono inizializzati con i valori dei parametri attuali
- L'assegnamento avviene secondo l'ordine in cui appaiono
  - Il primo parametro formale con il primo attuale, il secondo con il secondo e così via.

```
def areaDelTriangolo(B,H):
 ''' restituisce l'area del triangolo '''
 return B*H/2
```

```
base = float(input())
altezza = float(input())
area = areaDelTriangolo(base,altezza)
print(area)
```

I nomi dei parametri attuali e quelli dei parametri formali potrebbero anche essere uguali

- Quando la funzione termina le variabili che sono state create con l'invocazione della funzione vengono eliminate

# Return

- Può essere presente più volte, in percorsi diversi (ad esempio in un if e un else)

```
def numeroPrimo(N):
 for i in range(2, N//2+1):
 if N%i==0:
 return False
 return True
```

- Nelle funzioni che non restituiscono valori in genere l'istruzione return manca
- Si può comunque usare return per interrompere prematuramente la funzione

```
def stampaQuadrato(lato):
 if lato < 0:
 return
 for i in range(lato):
 for j in range(lato):
 print("@", end=' ')
 print()
```

```
stampaQuadrato(x)
```

# Funzioni nei programmi

- Un programma può utilizzare una o più funzioni
- L'invocazione di una funzione deve avvenire dopo che la funzione è stata definita

```
N = int(input())
N2 = quadrato(N) #qui quadrato non è stata ancora definita
```

```
def quadrato(X):
 ''' restituisce il quadrato di un numero '''
 return X*X
```

Questo genera un errore di compilazione  
**Traceback (most recent call last):**  
**File "<pyshell#14>", line 1, in <module>**  
**N2=quadrato(N)**  
**NameError: name 'quadrato' is not defined**

# Funzioni nei programmi

- Un programma può utilizzare una o più funzioni
- L'invocazione di una funzione deve avvenire dopo che la funzione è stata definita
- All'interno di una funzione può essere invocata un'altra funzione

```
def quadrato(X):
 ''' restituisce il quadrato di un numero '''
 return X*X
```

```
def cubo(X):
 ''' restituisce il cubo di un numero '''
 return quadrato(X)*X
```

```
N = int(input())
N2 = cubo(N)
```

La funzione cubo invoca  
la funzione quadrato

# Funzioni nei programmi

- Un programma può utilizzare una o più funzioni
- L'invocazione di una funzione deve avvenire dopo che la funzione è stata definita
- All'interno di una funzione può essere invocata un'altra funzione

```
def cubo(X):
 ''' restituisce il cubo di un numero '''
 return quadrato(X)*X
```

```
def quadrato(X):
 ''' restituisce il quadrato di un numero '''
 return X*X
```

```
N = int(input())
N2 = cubo(N)
```

Non è importante l'ordine  
in cui vengono scritte le  
funzioni

# La funzione main

- Quando si scrivono programmi con funzioni, è utile che tutte le istruzioni siano contenute all'interno di funzioni
- Una di queste funzioni sarà il punto di partenza dell'esecuzione
  - Per uniformità con altri linguaggi questa funzione può essere chiamata `main`

```
def main():
 N = int(input())
 N2 = cubo(N)
```

```
def cubo(X):
 ''' restituisce il cubo di un numero '''
 return quadrato(X)*X
```

```
def quadrato(X):
 ''' restituisce il quadrato di un numero '''
 return X*X
```

```
main()
```

Il programma contiene  
solo l'invocazione della  
funzione main

# Ambito di visibilità delle variabili

- Parte di programma nella quale una variabile è accessibile
- Le variabili definite all'interno di una funzione si chiamano *variabili locali*
  - Sono visibili solo all'interno della funzione
  - Una variabile non può essere usata al di fuori del suo ambito di visibilità

```
def prova():
 N = 27
```

```
print(N)
```

Errore



# Ambito di visibilità delle variabili

- È possibile avere variabili con lo stesso nome ma con ambiti di visibilità diversa
- L'ambito di visibilità di una variabile può anche essere *globale*
  - Una variabile globale è visibile dal punto in cui è definita fino alla fine del file
  - Anche all'interno delle funzioni

```
def prova():
 print(N)
```

```
N=36
prova()
print(N)
```

# Variabili locali e globali

- Una variabile locale può avere lo stesso nome di una variabile globale
  - In questo caso la variabile globale non è visibile nella funzione

```
def prova():
 print(N)
 N=27
```

```
N=36
prova()
print(N)
```

Errore

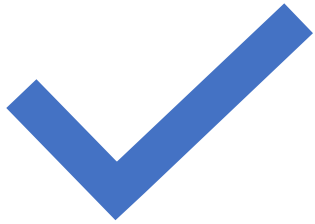
# Variabili locali e globali

- Una variabile locale può avere lo stesso nome di una variabile globale
  - In questo caso la variabile globale non è visibile nella funzione
- Se una funzione deve modificare il valore di una variabile globale bisogna aggiungere una dichiarazione global

```
def prova():
 global N
 print(N)
 N=27
```

```
N=36
prova()
print(N)
```

# Norme di buona programmazione



**Le variabili globali non dovrebbero essere usate all'interno di funzioni**

Utilizzare i parametri  
per la comunicazione con l'esterno



**Usare molte variabili globali**

complica la comprensione del codice  
è spesso fonte di errori

# Programmazione in Python

Lezione 10



# Funzioni e Liste

- Le funzioni possono anche operare su liste
  - Le liste possono essere passate come parametro alla funzione

```
def somma(L):
 somma=0
 for i in range(len(L)):
 somma+=L[i]
 return somma
```

```
L=[3,-3,5]
print(somma(L))
```

5

# Funzioni e Liste

- Le funzioni possono anche operare su liste
  - Le liste possono essere passate come parametro alla funzione
  - Il contenuto della lista può essere modificato all'interno di una funzione

```
def prova(L,x):
 if x not in L:
 L.append(x)
```

```
L=[1,3]
prova(L,23)
print(L)
```

```
[1, 3, 23]
```

# Funzioni e Liste

- Le funzioni possono anche operare su liste
  - Le liste possono essere passate come parametro alla funzione
  - Il contenuto della lista può essere modificato all'interno di una funzione
  - Possono essere restituite da una funzione

```
def estraiPari():
 pari=[]
 for i in range(10):
 x=int(input())
 if x%2==0:
 pari.append(x)
 return pari
```

```
listaPari=estraiPari()
print(listaPari)
```

Se in input si inseriscono:

3 5 7 9 1 2 4 6 8 9

Il programma stampa

[2, 4, 6, 8]



# Esercizio

- Scrivere un programma che, presa da input una sequenza di N numeri, utilizzando una funzione, stampa i numeri **TRIANGOLARI**.
- Un numero si dice triangolare se può essere rappresentato graficamente come un triangolo, ovvero, preso un insieme di elementi con cardinalità pari al numero in oggetto, è possibile disporre i suoi elementi su una griglia regolare, in modo da formare un triangolo equilatero o un triangolo isoscele.



Sono formati dalla somma di numeri consecutivi partendo da 1

# Esercizio

```
def main():
 x=int(input())
 while x!=0:
 if triangolare(x):
 print(x)
 x=int(input())
```

Per ogni numero della  
sequenza si rimanda alla  
funzione la verifica della  
proprietà di essere  
triangolare

```
def triangolare(N):
 somma=0
 i=1
 while somma<N:
 somma+=i
 i+=1
 return somma==N
```

Si sommano i numeri  
consecutivi partendo da 1,  
finchè non si raggiunge o  
supera N

```
main()
```

# Esercizio

- Scrivere un programma che, letta da input una sequenza di N elementi, determinare il massimo e stampa l'elenco dei numeri contenuti nella sequenza con il relativo scarto dal massimo.
- Es. Se N è 7 e la sequenza è 13, 12, 9, -7, 5, 14, 2 il programma dovrebbe stampare

```
13 Scarto 1
12 Scarto 2
9 Scarto 5
-7 Scarto 21
5 Scarto 9
14 Scarto 0
2 Scarto 12
```

# Esercizio

- Quali sotto-problemi possiamo individuare?
  - Creazione della sequenza con N valori presi da input
  - Calcolo del massimo della sequenza
  - Stampa ogni valore della sequenza con il relativo scarto dal massimo
- Progettiamo la soluzione affidando ogni sotto-problema ad una funzione
- Pseudocodice
  - Leggi da input N
  - Assegna ad una lista S una sequenza di N numeri presi da input
  - Assegna ad M il massimo della lista S
  - Stampa gli scarti da M degli elementi di S

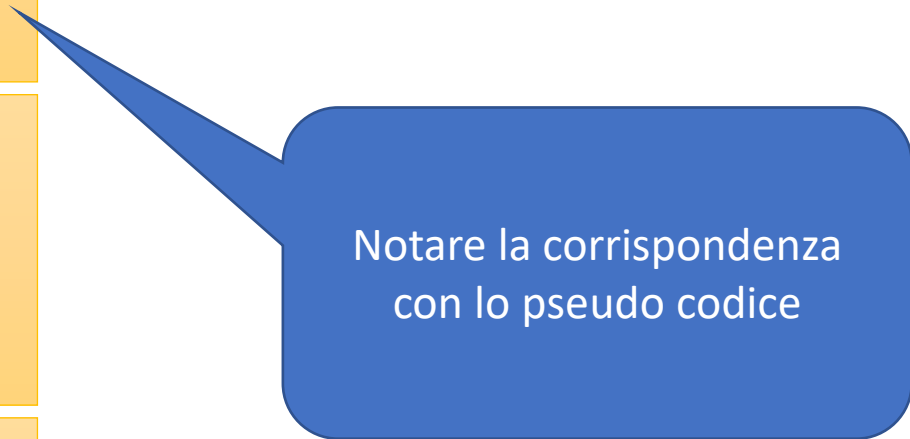
```
def main():
 N=int(input())
 sequenza=leggiSequenza(N)
 massimo=max(sequenza)
 stampaScarti(sequenza,massimo)
```

```
def leggiSequenza(N):
 sequenza=[]
 for i in range(N):
 sequenza.append(int(input()))
 return sequenza
```

```
def stampaScarti(S, max):
 for i in range(len(S)):
 print(S[i], "Scarto", max-S[i]);
```

```
def max(S):
 max=S[0]
 for i in range(len(S)):
 if S[i]>max:
 max=S[i]
 return max;
```

```
main()
```



Notare la corrispondenza  
con lo pseudo codice

# Ricerca di un element in un elenco:

## Ricerca Lineare e Ricerca Binaria

- Dato un elemento  $x$  ed un elenco, verificare se  $x$  è presente e determinare la sua posizione
- **Ricerca lineare**
  - Confronta ogni elemento dell'elenco con quello da cercare
  - Utile per elenchi di piccole dimensione e nei quali gli elementi non seguono alcun ordine
  - Se gli elementi dell'elenco sono  $n$ , sono necessary al massimo  $n$  passi per rispondere
- **Ricerca Binaria**
  - Si può utilizzare solo su array i cui elementi seguono un qualche ordine
  - Confronta l'elemento da cercare con quello da cercare
    - Se sono uguali, l'elemento è presente
    - Altrimenti, se l'elemento da cercare è più piccolo di quello centrale, si ripete la ricerca nella prima metà dell'elenco
    - Se invece è più piccolo, la ricerca si ripete sulla parte destra
  - Molto veloce: in un elenco di  $n$  elementi, sono necessary al massimo  $\log_2 n$  passi per rispondere
    - Su 30 elementi impiega al massimo 5 passi

# Ricerca Lineare

```
def ricercaLineare(V, x):
 for i in range(len(V)):
 if V[i] == x:
 return i # x e' presente in posizione i
 return -1; # x non e' presente
```

```
def main():
 L=[1,3,5,2,4,9]
 x=5
 pos=ricercaLineare(L,x)
 if pos==-1:
 print(x, "non è presente in ", L)
 else:
 print(x, "è presente in ", L, "in
posizione",pos)
```

```
main()
```

# Ricerca Binaria

```
def ricercaBinaria(V, x):
 inf = 0
 sup = len(V)-1
 while inf <= sup:
 mid = (inf + sup)//2 # Elemento centrale (circa)
 if V[mid] == x:
 return mid # x è presente in posizione mid
 else:
 if V[mid] < x: inf = mid + 1 # cerca a destra di mid
 else: sup = mid -1 # cerca a sinistra di mid
 return -1 # x non è presente
```

```
def main():
 L=[1,3,5,12,14,29]
 x=5
 pos=ricercaBinaria(L,x)
 if pos==-1: print(x, "non è presente in ", L)
 else: print(x, "è presente in ", L, "in posizione",pos)
```

```
main()
```

Si noti che l'elenco è ordinato



# Algoritmi di Ordinamento

- Ordinamento di dati secondo una relazione d'ordine
  - Fondamentale per moltissime applicazioni informatiche
    - elenchi telefonici, biblioteche, dizionari, magazzini, archivi, ecc.
  - Facilita le ricerche successive di elementi all'interno dell'elenco
- Metodi semplici:
  - Facili da capire e da realizzare
  - Generalmente richiedono un numero di confronti dell'ordine di  $n^2$
  - Rapidi per  $n$  abbastanza piccolo ma inutilizzabili per  $n$  grande
- Metodi avanzati:
  - Tecniche più complesse e meno intuitive
  - Richiedono, generalmente, un numero di confronti nell'ordine di  $n \log n$

# Bubble Sort

- Si effettuano una serie di passate sull'elenco fino a che esso non risulti ordinato.
- Ad ogni passata, si confrontano coppie di elementi consecutivi e si scambiano di posto gli elementi che non verificano l'ordinamento.

2 1 4 5 3 → 1 2 4 5 3 → 1 2 4 3 5

Passata 1

1 2 4 3 5 → 1 2 3 4 5

Passata 2

...

...

# Bubble Sort

```
def main():
 N=int(input())
 sequenza=leggiSequenza(N)
 bubbleSort(sequenza)
 print(sequenza)
```

```
def leggiSequenza(N):
 sequenza=[]
 for i in range(N):
 sequenza.append(int(input()))
 return sequenza
```

```
def bubbleSort(S):
 for cont in range(len(S)):
 for j in range(len(S)-1):
 if S[j] > S[j+1]:
 temp=S[j]
 S[j]=S[j+1]
 S[j+1]=temp
 print("passata ",cont, "Elenco->", S)
```

```
main()
```

Se N fosse 5 e l'elenco fosse 1 2 4 5 3 il programma stamperebbe

passata 0 Elenco-> [1, 2, 4, 3, 5]  
passata 1 Elenco-> [1, 2, 3, 4, 5]  
passata 2 Elenco-> [1, 2, 3, 4, 5]  
passata 3 Elenco-> [1, 2, 3, 4, 5]  
passata 4 Elenco-> [1, 2, 3, 4, 5]  
[1, 2, 3, 4, 5]

Si noti che l'elenco è ordinato  
già alla seconda passata

Sono possibili ottimizzazioni

# Bubble Sort – versione 2

- Per rendere più efficiente l'algoritmo si utilizza una variabile booleana `scambi` che segnala se durante una passata sono stati effettuati o meno degli scambi.
- Se durante una passata non sono stati effettuati scambi ( e quindi la variabile `scambi` è `False`) l'array è già ordinato e l'algoritmo termina.

```
def bubbleSort(S):
 i = len(S)
 ordinato = False
 while not ordinato:
 ordinato = True;
 for j in range(i-1):
 if S[j] > S[j+1]:
 temp=S[j]
 S[j]=S[j+1]
 S[j+1]=temp
 ordinato = False;
 print("Elenco->", S)
 i-=1
```

Se N fosse 5 e l'elenco fosse 1 2 4 5 3 il programma stamperebbe

Elenco-> [1, 2, 4, 3, 5]  
Elenco-> [1, 2, 3, 4, 5]  
Elenco-> [1, 2, 3, 4, 5]  
[1, 2, 3, 4, 5]

Alla terza passata non avvengono scambi, l'algoritmo termina

Ogni passata termina prima delle precedenti

Esercizio: Ottimizzare ulteriormente l'algoritmo in modo tale che ogni passata termini alla posizione dove è avvenuto l'ultimo scambio nella passata precedente

# Programmazione in Python



# Selection Sort

- Si cerca il minimo dell'array; si scambia l'elemento minimo con il primo elemento dell'array; poi si cerca il minimo nell'array a partire dal secondo elemento fino alla fine, e lo si scambia con l'elemento al secondo posto, e così via
- Il costo non dipende da come è l'input ma dalla dimensione dell'array
  - Su array ordinati o disordinati non cambia quasi niente.
- Utile per ordinare oggetti grandi poiché ogni oggetto viene spostato al più una volta.

# Selection Sort

```
def minIndex(S,i):
 minIndex = i
 for j in range(i+1, len(S)):
 if S[j] < S[minIndex]:
 minIndex = j
 return minIndex
```

```
def selectionSort(S):
 for i in range(len(S)-1):
 posMin=minIndex(S,i)
 S[i], S[posMin] = S[posMin], S[i] #scambia gli elementi
 print(S)
```

Se N fosse 5 e l'elenco fosse 1 2 4 5 3 il programma stamperebbe

```
[1, 2, 3, 5, 4]
[1, 2, 3, 5, 4]
[1, 2, 3, 5, 4]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
```

Esegue sempre N passate per  
calcolare il minimo sull'array

Si noti come avviene lo scambio

# Tuple

- In python una tupla è una sequenza di elementi immutabile
- Si differenzia dalle liste perché gli elementi sono racchiusi tra parentesi tonde
- Una volta creata, una tupla non si può modificare

```
tuplaInteri = (1,3,5,7)
```

- Si possono usare sulle tuple le funzioni che si usano sulle liste purchè non ne modifichino il contenuto
- Le tuple si possono usare per assegnare più valori a più variabili

```
(A, B) = (1, 2)
```

A=1

B=2

- Le parentesi tonde si possono omettere



# Tuple

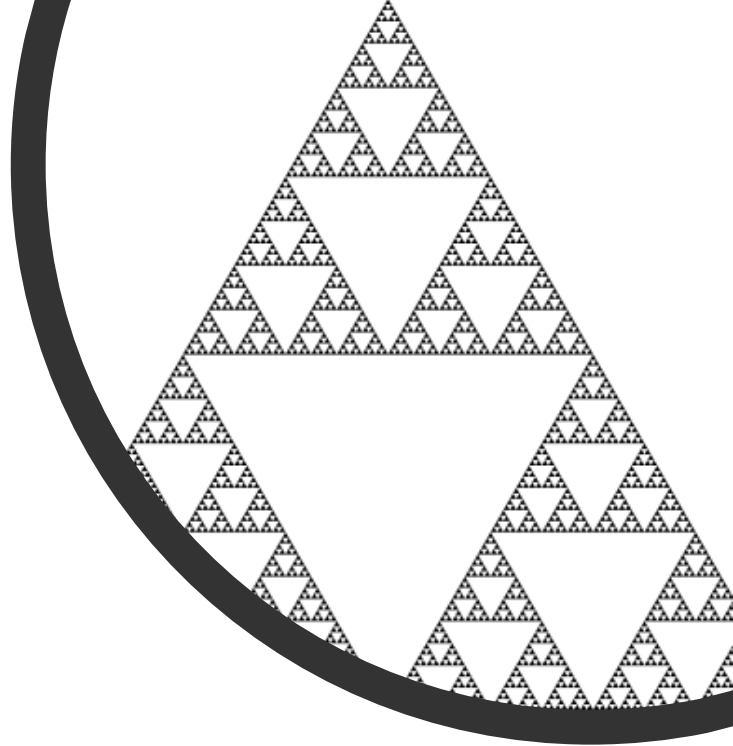
- Le tuple possono essere usate per fare in modo che una funzione restituisca più valori
  - La funzione restituisce una tupla di valori

```
def funzione():
 A = input()
 B = int(input())
 return (A,B)
```

```
Tupla=funzione()
print(Tupla)
```

# Tuple

- Le tuple possono essere usate per definire funzioni che ricevono un numero variabile di argomenti
- To Be Continued ;-)



La ricorsione



# La ricorsione

- E' una *tecnica di programmazione*
- Il problema da risolvere viene suddiviso in sotto-problemi simili a quello originale, ma generalmente più piccoli e più semplici.
- Si studia il problema e si identificano dei *casi base*, casi più semplici che possono essere risolti facilmente;
- La soluzione del problema originale viene ottenuta combinando la soluzione di uno o più problemi sotto-problemi.

# Funzioni Ricorsive

Sono funzioni che richiamano se stesse

Devono essere progettate in maniera tale che:

- alcuni casi speciali più semplici siano risolti direttamente (casi base)
- ogni invocazione ricorsiva semplifica in qualche modo l'elaborazione, ossia:
  - ***risolve un sotto-problema più semplice***
  - ***si avvicina sempre di più a un caso base***

Attenzione: una cattiva progettazione può portare ad una ricorsione infinita

# Funzioni Diretta e Indiretta

- Ricorsione diretta: all'interno del corpo della funzione è presente una chiamata alla funzione stessa

```
def funzione (parametro1, ..., parametron) :
 ...
 funzione (p1, ..., pn)
```

- Ricorsione indiretta: all'interno del corpo della funzione è presente una chiamata ad un'altra funzione che a sua volta richiama la funzione di partenza

```
def funzione1 (parametro1, ..., parametron) :
 ...
 funzione2 (q1, ..., qm)
```

```
def funzione2 (parametro1, ..., parametrom) :
 ...
 funzione1 (p1, ..., pn)
```

# Esempio: il fattoriale

- Il fattoriale di un numero  $n$  è definito come:

$$n! = n * (n - 1) * (n - 2) * \dots * 1$$

## Soluzione Iterativa

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

```
n = int(input())
print(fattoriale(n))
```

# Il fattoriale – approccio ricorsivo

- **IDEA:** Si riconduce il problema del calcolo di  $n!$  ad un sotto-problema simile al problema di partenza ma su un input più piccolo, ossia al calcolo di  $(n-1)!$

$n!$  si può scrivere come  $n * (n - 1)!$

$$5! = 5 * 4!$$

$$4! = 4 * 3!$$

- Ci sono dei casi più semplici che possono essere risolti direttamente

*CASI BASE*

$$1! = 0! = 1$$

- Il calcolo di  $n!$  si riconduce al calcolo di  $(n-1)!$  che a sua volta verrà ricondotto a  $(n-2)!$  e così via fino a quando non si raggiunge un caso base.
- La soluzione del problema originario si ottiene combinando le soluzioni dei vari sottoproblemi.



# Soluzione Ricorsiva

```
def fattorialeRicorsivo(n):
 if n==0 or n==1:
 return 1
 return n*fattorialeRicorsivo(n-1)
```

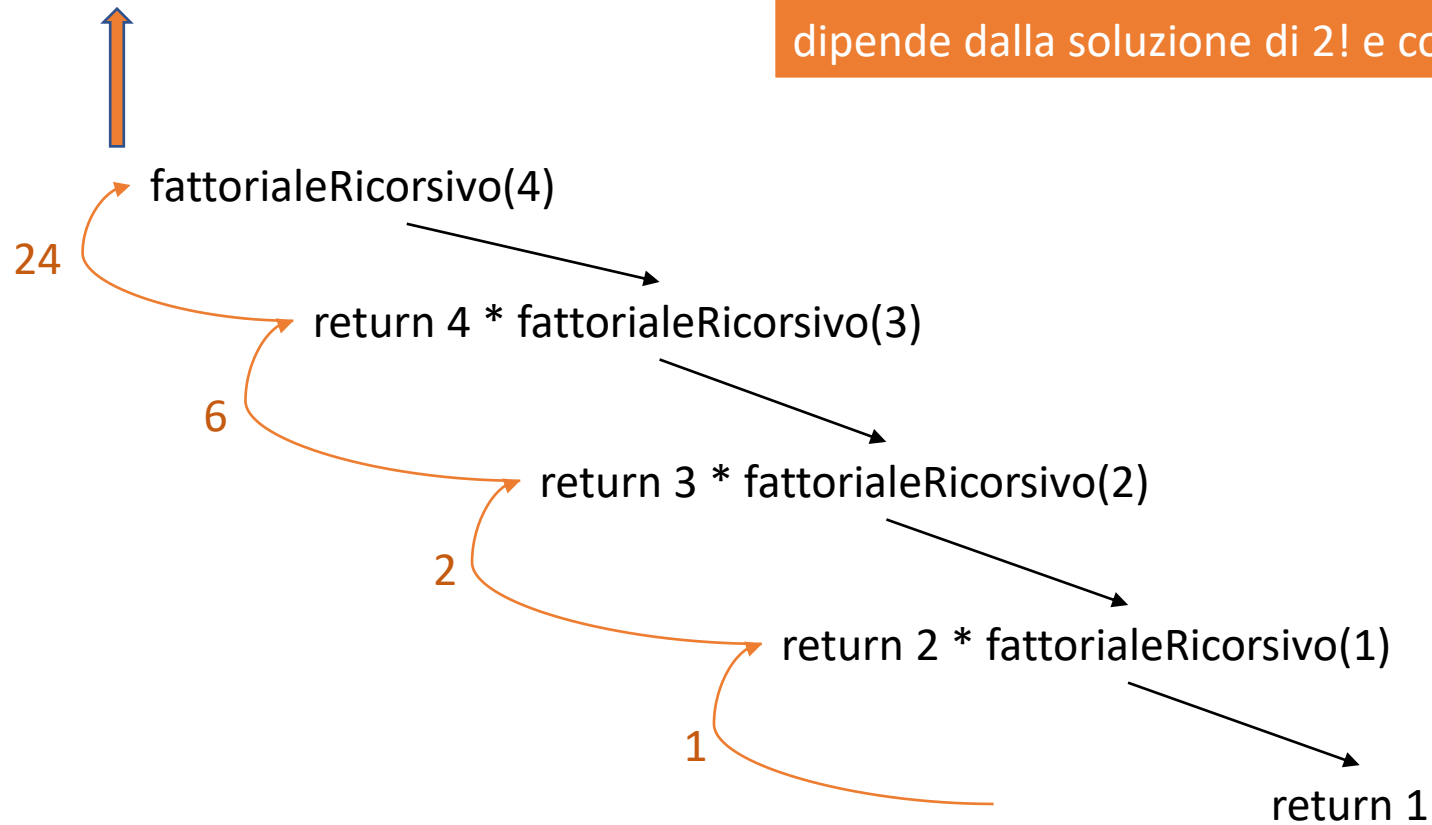
Casi Base

Chiamata Ricorsiva

```
n= int(input())
print(fattorialeRicorsivo(n))
```

# Esempio di Invocazione per 4!

La soluzione di 4! si ottiene dalla soluzione di 3! che a sua volta dipende dalla soluzione di 2! e così via



# I numeri di Fibonacci

- Ogni numero è la somma dei due precedenti
  - 0, 1, 1, 2, 3, 5, 8...

Sono elementi della successione così definita

$$\text{fib}(1) = 1$$

$$\text{fib}(2) = 1$$

$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2) \text{ per ogni } n \neq 1 \text{ e } n \neq 2$$



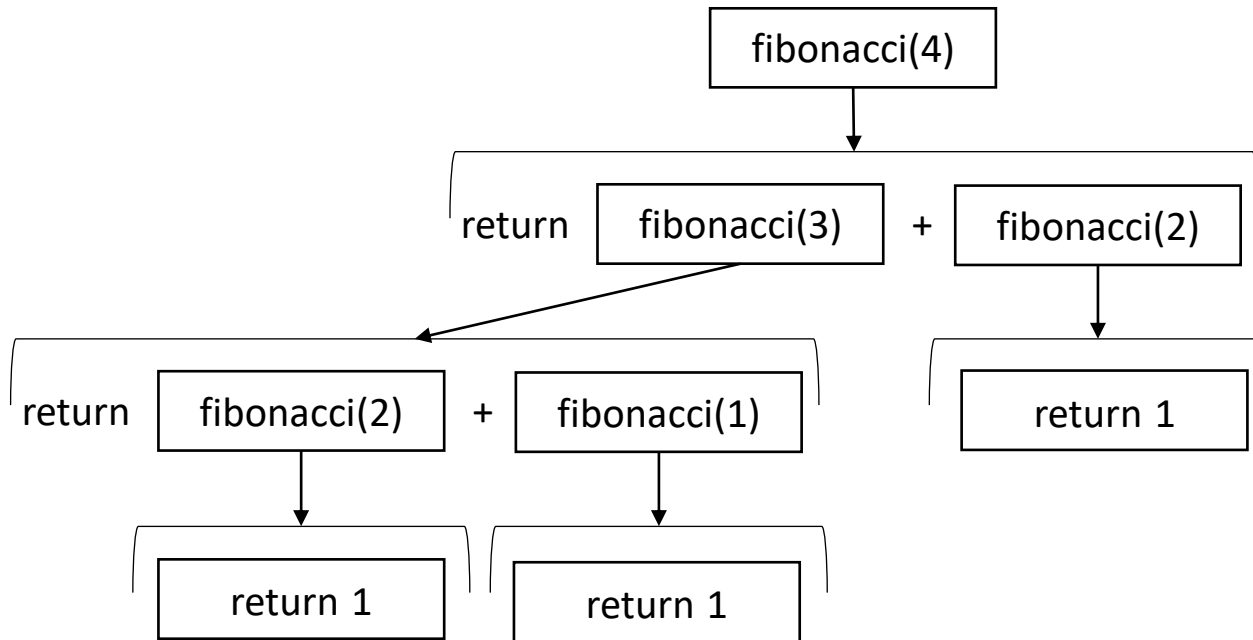
# Funzione Ricorsiva per calcolare i numeri di Fibonacci

```
def fibonacci(n):
 if n==1 or n==2:
 return 1
 else:
 return fibonacci(n-1)+fibonacci(n-2)
```

Casi base

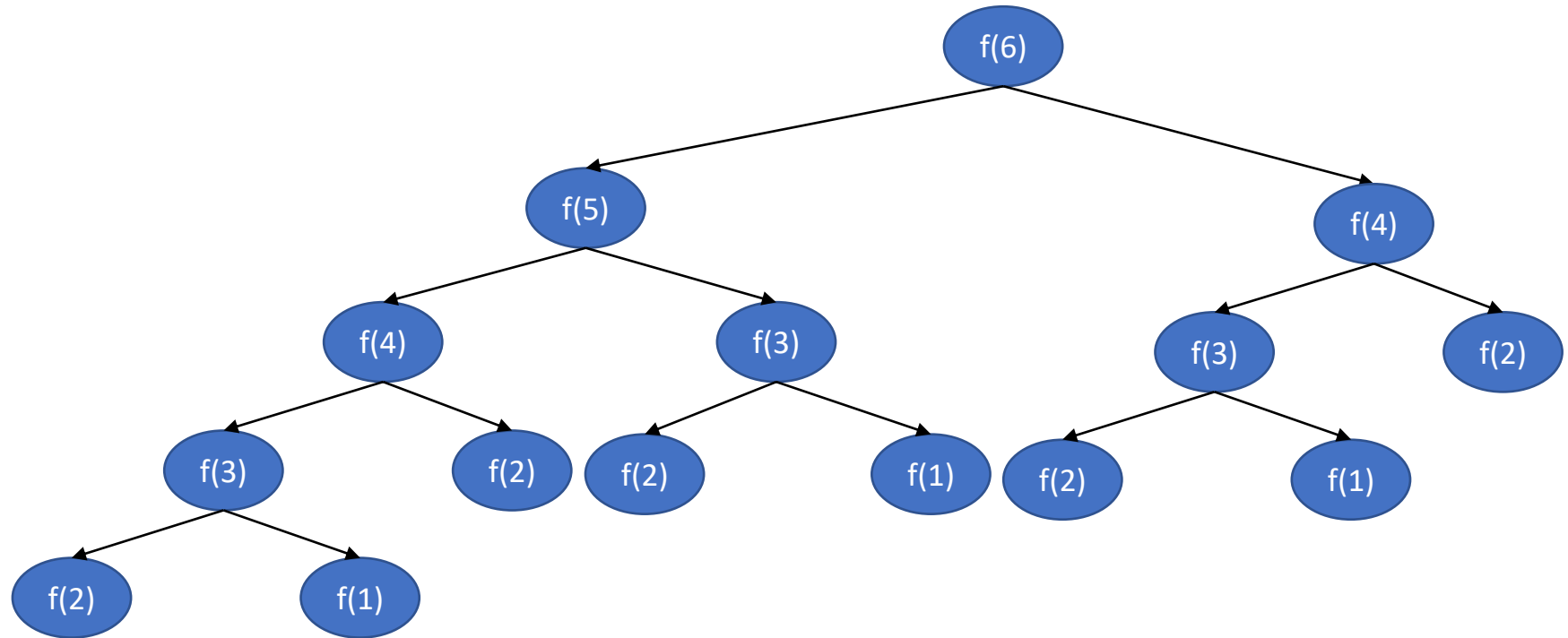
Il calcolo di fibonacci(n) è ricondotto alla  
risoluzione di due sottoproblemi  
fibonacci(n-1) e fibonacci(n-2)

# Esempio di Invocazione per fibonacci di 3



Alcune chiamate  
vengono ripetute più  
volte

# Esempio di Invocazione per fibonacci di 6



# Iterazione vs Ricorsione

- Ripetizione
  - Iterazione: ciclo esplicito
  - Ricorsione: chiamate ripetute alla funzione
- Terminazione
  - Iterazione: fallisce la condizione nel ciclo
  - Ricorsione: raggiungimento del caso base
- In entrambe si può entrare in “cicli infiniti”

Bilancio tra  
performance (iterazione) e  
una buona ingegneria del software (ricorsione)

# Programmazione in Python





# Esempio: Potenza di un numero

- Calcolo ricorsivo di  $N^M$  (con  $M > 0$ )
- IDEA: Si riconduce il problema del calcolo di  $N^M$  al problema del calcolo di  $N^{M-1}$

Infatti  $N^M$  si può scrivere come  $N * N^{M-1}$

Formulazione  
Ricorsiva

$N^0=1$  è un caso risolvibile direttamente  
(ma anche  $N^1=N$ )

Casi Base

# Potenza di un Numero

```
def potenzaRic(N,M):
 if M==0:
 return 1
 return N*potenzaRic(N,M-1)
```

Caso Base

Chiamata Ricorsiva

```
def main():
 N,M=int(input()),int(input())
 print(potenzaRic(N,M))
```

Prima Invocazione

```
main()
```

# Esempio: Ricerca del massimo in una lista

```
def massimoRicorsivo(L, inf, max):
 if inf==len(L):
 return max
 if max < L[inf]:
 max =L[inf]
 return massimoRicorsivo(L,inf+1,max)
```

Caso Base  
(non ci sono elementi)

Si aggiorna il valore del  
massimo corrente

Chiamata Ricorsiva  
Si richiama la funzione per cercare il  
massimo da inf+1 in avanti

```
def main():
 L=[-7,-3,52,4]
 print(massimoRicorsivo(L,1,L[0]))
```

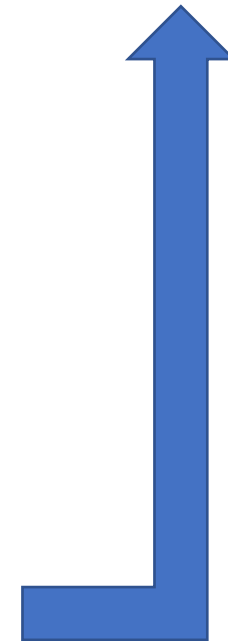
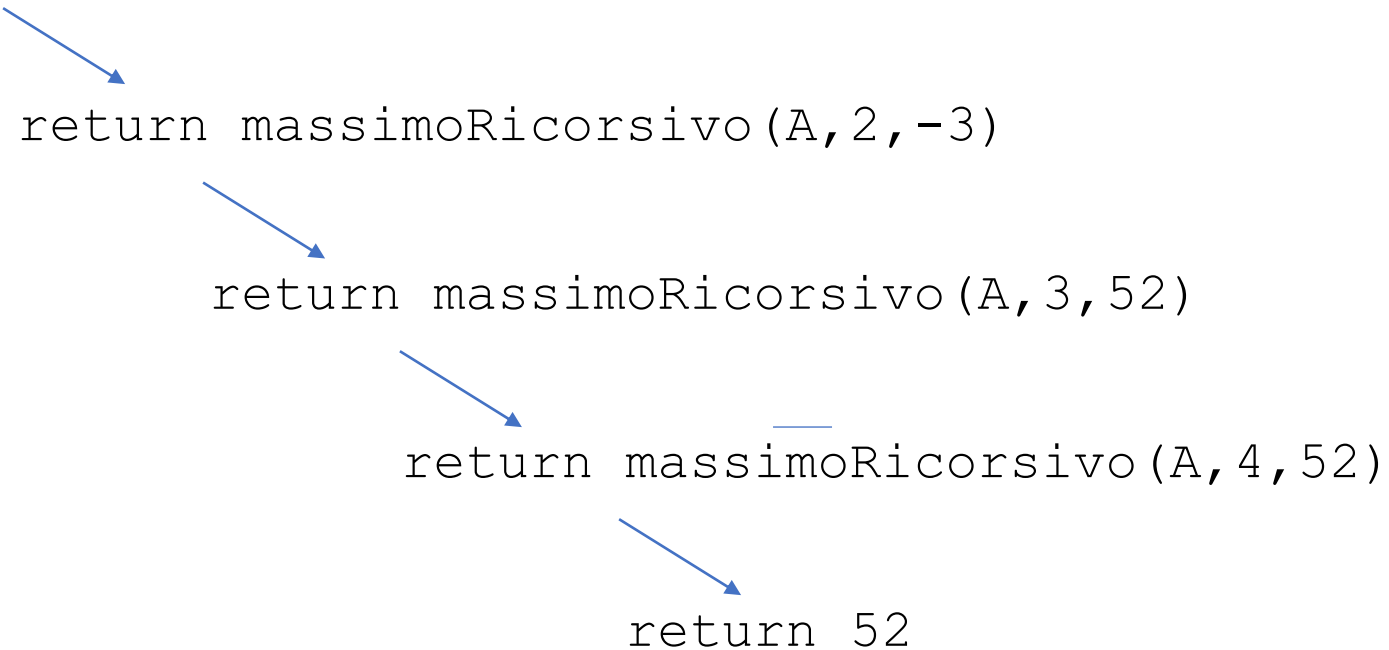
```
main()
```

Prima Invocazione  
Si suppone che il massimo  
è il primo elemento

# Esempio di Invocazione

`L=[-7,-3,52,4]`

`massimoRicorsivo(L,1,-7)`



# Esempio: Somma della diagonale di una matrice

- Scrivere una funzione ricorsiva che data una matrice M quadrata calcoli la somma degli elementi sulla diagonale principale di M

```
def somma_ricorsiva (M,i):
 if i>=len(M):
 return 0
 else:
 return M[i][i] + somma_ricorsiva(M,i+1)
```

```
def main():
 M=[[1,2,5, 3],[0,2,0,2],[0,2,3,2],[1,2,3,4]]
 print(somma_ricorsiva(M,0))
```

```
main()
```

|          |          |          |          |
|----------|----------|----------|----------|
| <u>1</u> | 2        | 5        | 3        |
| 0        | <b>2</b> | 0        | 2        |
| 0        | 2        | <b>3</b> | 2        |
| 1        | 2        | 3        | <b>4</b> |