Programmazione (scientifica) in python

Liceo Scientifico e Sportivo Statale "A. Tassoni"

26/02/2018

Before we start...

Github (parte I)

Lo sviluppo software moderno è sempre *collaborativo*. La gestione di progetti importanti avviene su piattaforme dedicate, ad esempio

- sourceforge
- launchpad
- bitbucket
- gitlab

ma soprattutto

- github

Ciascuna di queste piattaforme permette la creazione di *repository* per la gestione del codice.

Github (parte I)

Lo sviluppo software moderno è sempre *collaborativo*. La gestione di progetti importanti avviene su piattaforme dedicate, ad esempio

- sourceforge
- launchpad
- bitbucket
- gitlab

ma soprattutto

- github

Ciascuna di queste piattaforme permette la creazione di *repository* per la gestione del codice.

Alla base di queste piattaforme c'è sempre un **software di controllo versione**, anche se tutte permettono molte altre funzionalità oltre alla gestione del codice.

Il software più popolare è git (ne impareremo le basi durante l'ultima lezione).

Github (parte I)

- Gratuito se si usano solo repository pubblici
- Qualsiasi repository pubblico è *clonabile* e *biforcabile* da chiunque
- Offre molte funzionalità oltre al semplice controllo versione:

    1. Gestione progetti
    2. Gestione degli *issues* (problemi) con il codice
    3. Pagine wiki
    4. Analisi statistica dell'utilizzo del repository e dell'attività dei *contributori*
    5. … e molto altro (specialmente se pagate)…

Github (parte I)

- Gratuito se si usano solo repository pubblici
- Qualsiasi repository pubblico è *clonabile* e *biforcabile* da chiunque
- Offre molte funzionalità oltre al semplice controllo versione:

    1. Gestione progetti
    2. Gestione degli *issues* (problemi) con il codice
    3. Pagine wiki
    4. Analisi statistica dell'utilizzo del repository e dell'attività dei *contributori*
    5. ... e molto altro (specialmente se pagate)...

per ora...

https://github.com/LiceoTassoni/CorsoPython2018

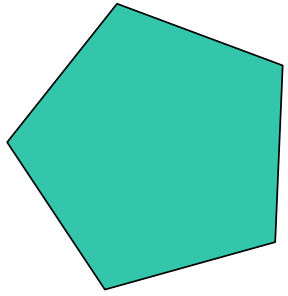
I Jupyter notebook

aprire nel browser il sito:

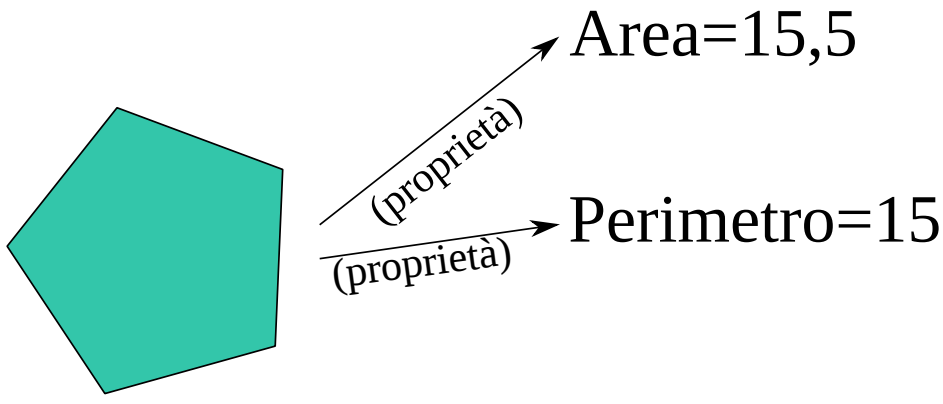
192.168.2.74:8000

e inserisci le credenziali

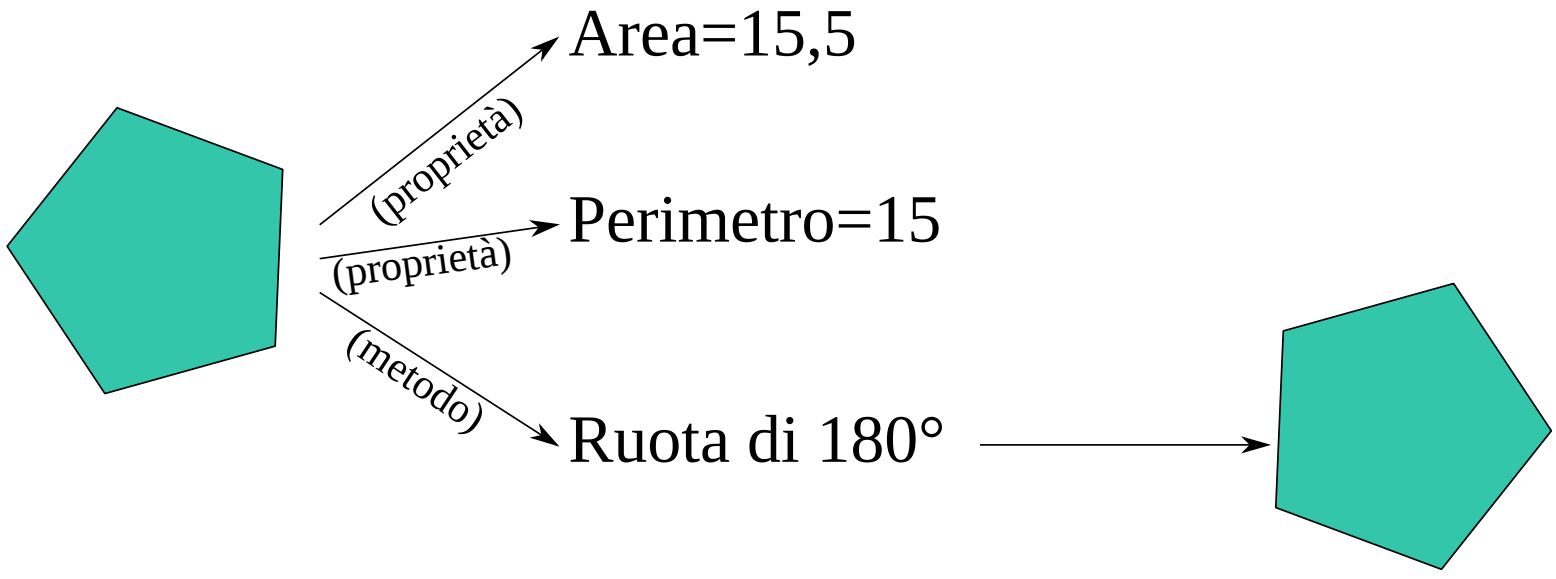
Programmazione "ad oggetti"



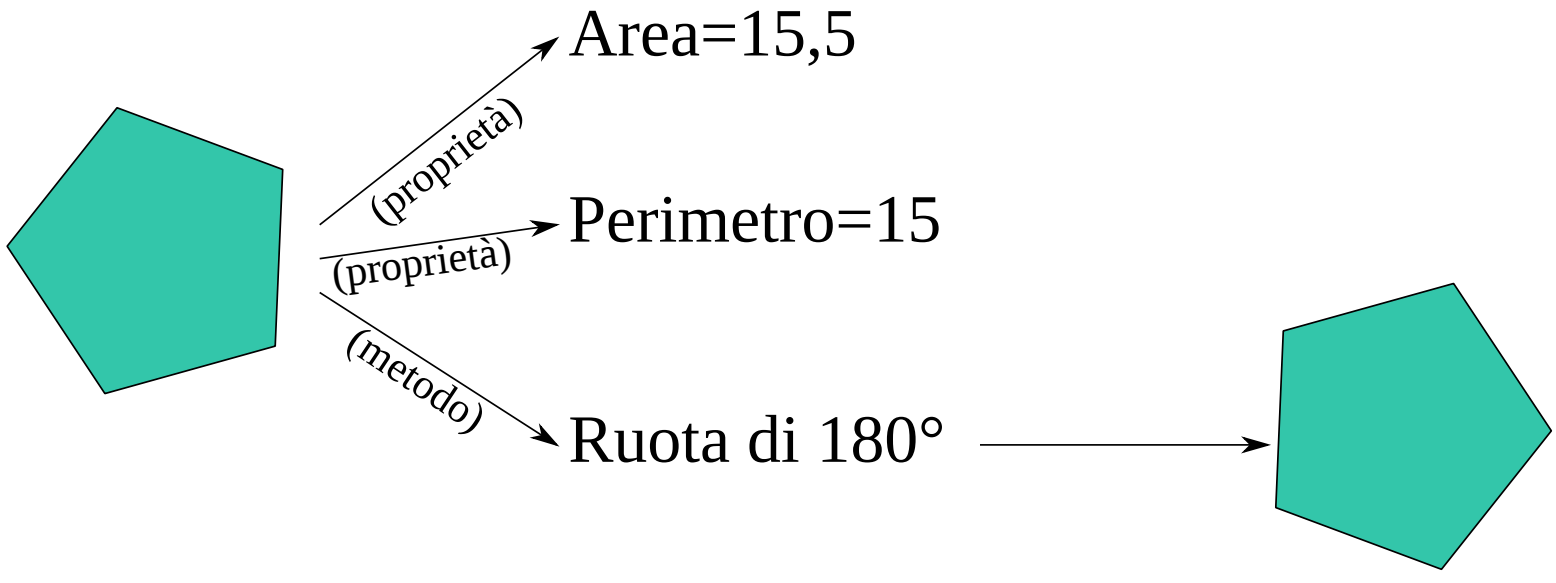
Programmazione "ad oggetti"



Programmazione "ad oggetti"



Programmazione "ad oggetti"



In python, **tutto** è oggetto (quasi), ed ha *proprietà* e *metodi*.

La sintassi per richiamare proprietà e metodi di un oggetto è

```
oggetto.proprietà
```

o

```
oggetto.metodo()
```

Alcuni metodi di uso frequente

Stringhe: `split`

Il metodo `split` restituisce una lista di stringhe

```
x="""Del bel Panaro il pian, sotto due scorte,
A predar vanno i Bolognesi armati;
E da Gherardo altri condotti a morte,"""
print(x.split(" "))
print(x.split(","))
print(x.split("\n"))
```

Alcuni metodi di uso frequente

Stringhe: `find` o `index` (e derivati)

Si usano per trovare uno o più caratteri in una stringa

```
print(x.find("o i"))
print(x.rindex("o i"))
```

Si usa, per esempio, per *affettare* una stringa da un certo carattere in poi (o indietro):

```
a = x.find("o i")
print(x[a::-1])
```

Alcuni metodi di uso frequente

Stringhe: `find` o `index` (e derivati)

Si usano per trovare uno o più caratteri in una stringa

```
print(x.find("o i"))
print(x.rindex("o i"))
```

Si usa, per esempio, per *affettare* una stringa da un certo carattere in poi (o indietro):

```
a = x.find("o i")
print(x[a::-1])
```

Stringhe: `count`

Conta quante volte una sequenza di caratteri compare in un altra stringa:

```
print(x.count("o"))
```

Alcuni metodi di uso frequente

Stringhe: `join`

```
lista_parole = ["Del","bel","Panaro","il","pian","sotto", "due", "scorte"]
print(" ".join(lista_parole))
```

Stringhe: `replace`

Sostituisce una sequenza di caratteri in una stringa

```
x="""Del bel Panaro il pian, sotto due scorte,
A predar vanno i Bolognesi armati;
E da Gherardo altri condotti a morte,"""
print(x.replace("e", "oo"))
print(x)  # <- il contenuto originale non è modificato
```

Alcuni metodi di uso frequente

Liste: `append`

aggiunge elementi ad una lista

```
x = []
x.append("Ciao")
x.append(True)
print(x)
```

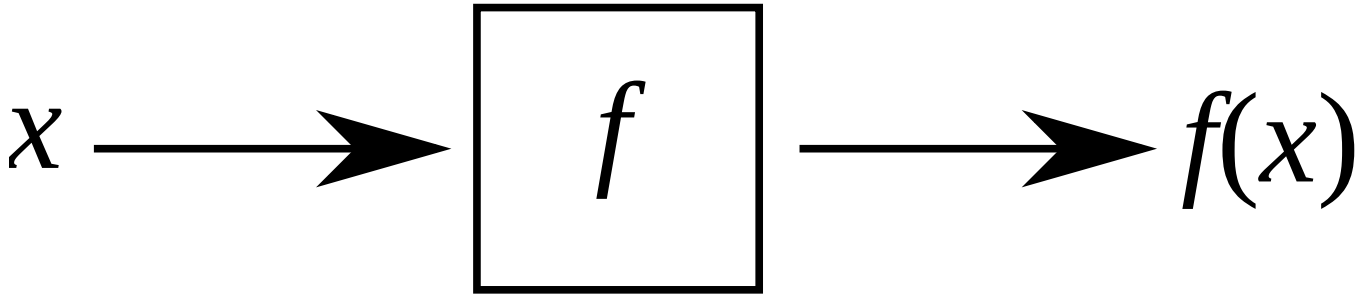
Liste: `sort`

Ordina il contenuto di una lista:

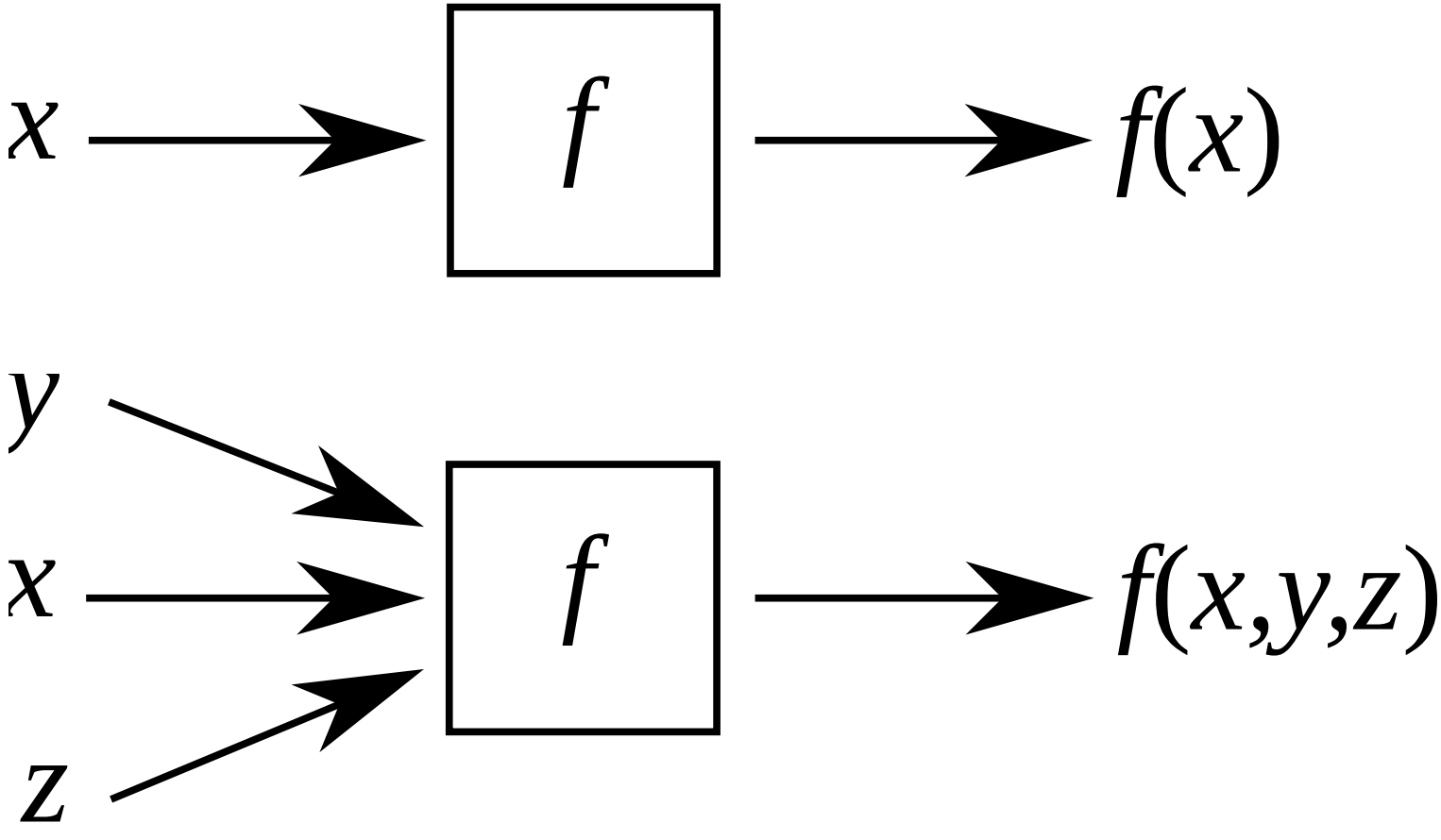
```
y = [5, 123.46, -235, 35e-3, 54, 43]
y.sort()
print(y)
```

A differenza dei metodi visti in precedenza per le stringhe, `sort` **modifica** il contenuto originale.

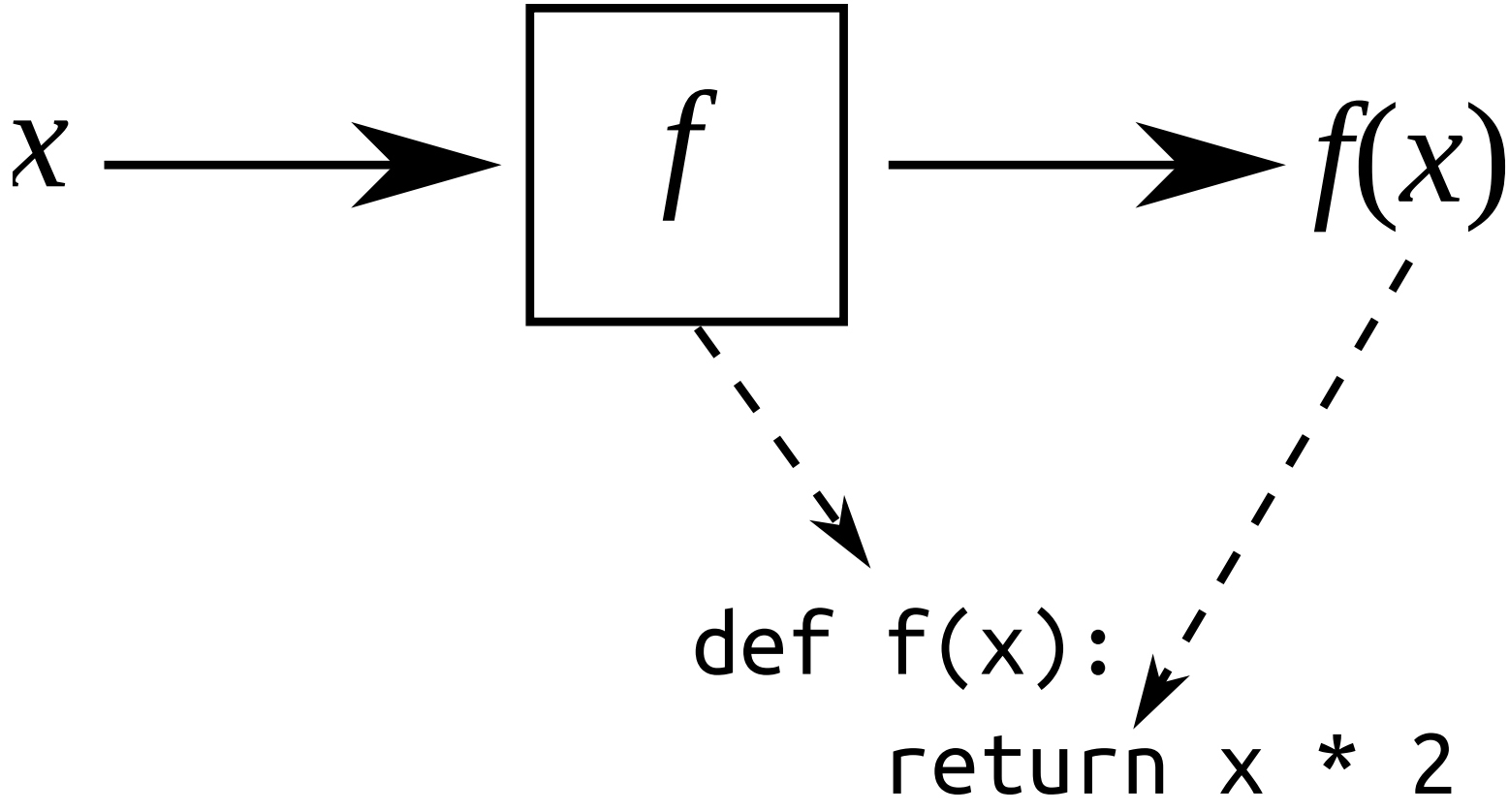
Fuzioni



Fuzioni



Fuzioni



Fuzioni

Sotto la definizione, è bene mettere una *stringa di documentazione (docstring)*

```
def raddoppia(x):
    """Raddoppia l'argomento"""  # <- meglio non ometterla
    return x*2

doppio_numero = raddoppia(5)
doppia_stringa = raddoppia("Ciao! ")
print(doppio_numero)
print(doppia_stringa)
print(raddoppia.__doc__)     # <--- senza parentesi!
```

Fuzioni

Sotto la definizione, è bene mettere una *stringa di documentazione (docstring)*

```
def raddoppia(x):
    """Raddoppia l'argomento"""  # <- meglio non ometterla
    return x*2

doppio_numero = raddoppia(5)
doppia_stringa = raddoppia("Ciao! ")
print(doppio_numero)
print(doppia_stringa)
print(raddoppia.__doc__)     # <--- senza parentesi!
```

Per non far fare nulla ad una funzione si può far restituire un valore qualsiasi (ad es. `None`)

```
def non_far_nulla():     # <--- l'argomento si può omettere
    return None
```

oppure (ancora meglio) usare il comando apposta (`pass`)

```
def non_far_nulla():
    pass
```

Funzioni

Raccomandazioni per l'uso con i notebook:

1. Ogni funzione in una cella a sé stante

2. Ogni volta che si modifica una funzione, bisogna eseguire nuovamente la cella

Funzioni

Attenzione allo **scope**. Tutte le variabili hanno un *ambito di validità*: nel caso delle funzioni, le variabili (compresi gli argomenti) sono definiti *localmente* ed hanno priorità sulle variabili con lo stesso nome definite *esternamente*

```
def raddoppia(k):
    """Raddoppia l'argomento"""
    return k*2
```

```
print(raddoppia(111))
print(k)
```

la variabile x non esiste al di fuori della funzione...

Funzioni

Attenzione allo **scope**. Tutte le variabili hanno un *ambito di validità*: nel caso delle funzioni, le variabili (compresi gli argomenti) sono definiti *localmente* ed hanno priorità sulle variabili con lo stesso nome definite *esternamente*

```
def raddoppia(k):
    """Raddoppia l'argomento"""
    return k*2
```

```
print(raddoppia(111))
print(k)
```

la variabile x non esiste al di fuori della funzione...

... e se esiste:

```
k=12
print(raddoppia(9))
```

Funzioni

Se una variabile *non* è definita all'interno di una funzione, l'interprete va a cercare se esiste una variabile con lo stesso nome definita esternamente:

```
def moltiplica(k):
    """Moltiplica l'argomento per a"""
    return k*m
```

```
# prova prima questo...
print(moltiplica(2))
```

```
# poi questo
m = 31
print(moltiplica(2))
```

Funzioni

Se una variabile *non* è definita all'interno di una funzione, l'interprete va a cercare se esiste una variabile con lo stesso nome definita esternamente:

```
def moltiplica(k):
    """Moltiplica l'argomento per a"""
    return k*m
```

```
# prova prima questo...
print(moltiplica(2))
```

```
# poi questo
m = 31
print(moltiplica(2))
```

Attenzione!! a scrivere codice in questo modo: il risultato di una funzione dovrebbe dipendere *solo* dal valore dei suoi argomenti! Cosa che in questo caso non è vera.

Ci sono ovviamente delle eccezioni, ma le vedremo...

Primi elementi di progammazione

Cicli `while`

```
i, n = 2, 1231
while i < (n//2 + 1):
    if n % i == 0:
        print(n, "è divisibile per", i)
    i = i + 1 # oppure i += 1
```

Primi elementi di progammazione

La sintassi `a in b` si utilizza per verificare se un elemento è presente o no in una lista o in una stringa.

```
lista = [ 1, "Due", 3.0, True, [ 5, "sei"] ]
print(1 in lista)
print(2 in lista)
print(3 in lista)
```

Primi elementi di progammazione

La sintassi `a in b` si utilizza per verificare se un elemento è presente o no in una lista o in una stringa.

```
lista = [ 1, "Due", 3.0, True, [ 5, "sei"] ]
print(1 in lista)
print(2 in lista)
print(3 in lista)
```

La sintassi che utilizza `in` è utile in combinazione con le istruzioni condizionali

```
if True in lista:
    print("c'è un valore \"True\" nella lista")
    # la \ è necessaria per inserire
    # virgolette NELLA stringa
else:
    print("non c'è il \"True\" nella lista")
```