





# Python 101

Curs 2 - Paradigme de Programare

# Pentru început...

-  Zoom + Discord + Moodle
  - Curs
  - Materiale și anunțuri
-  Marți, ora 18:00 - 21:00
  - Quiz de recapitulare din cursul precedent
  - Curs + Demo
  - Laborator
-  Puneți întrebări oricând
-  Feedback la fiecare curs

# Join us on Whatsapp!



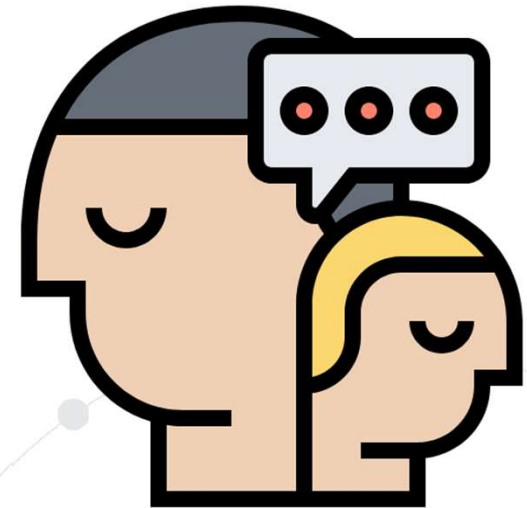
<https://bit.ly/3TNP1sJ>

The background features several decorative elements: a large blue shape on the left, a blue semi-circle at the bottom right, and thin grey arcs with small dots connecting different parts of the design.

# Quiz time!

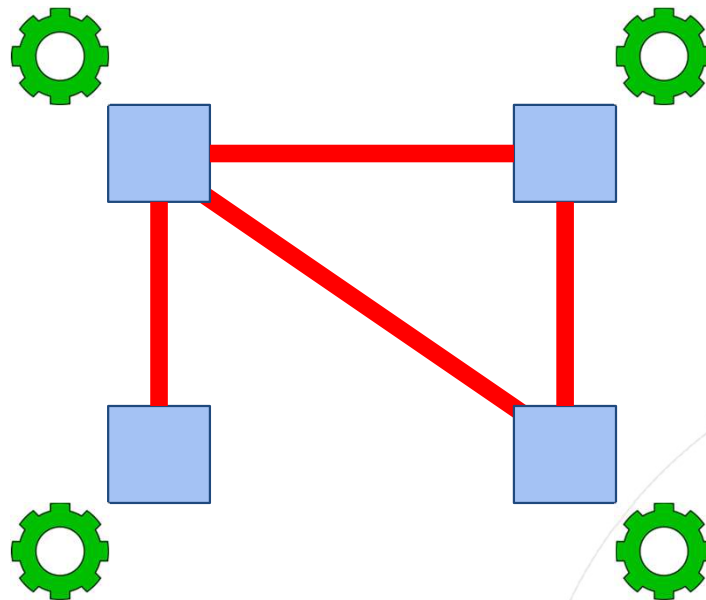
# Paradigme de programare

O paradigma dictează cum se formulează o problemă și cum sunt interpretate rezultatele.



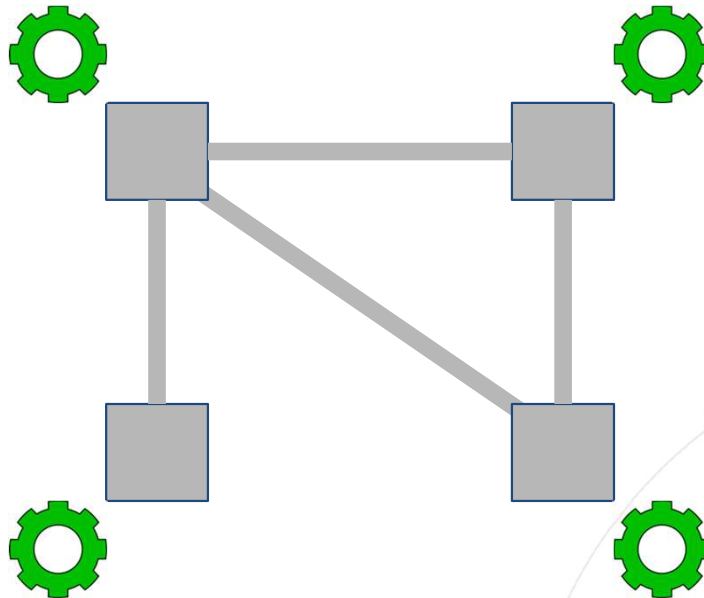
# Paradigme de programare

Paradigma	Accent pe	Rezultat



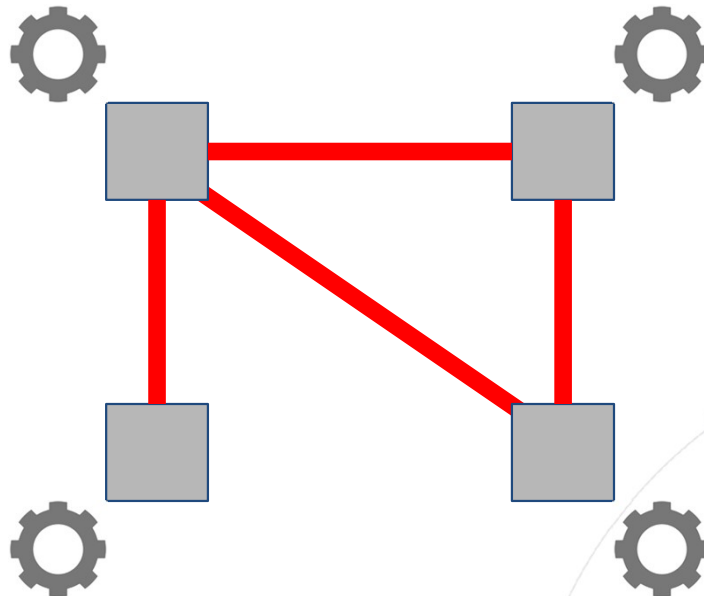
# Paradigme de programare

Paradigma	Accent pe	Rezultat
Procedurală	Acțiuni	Proceduri



# Paradigme de programare

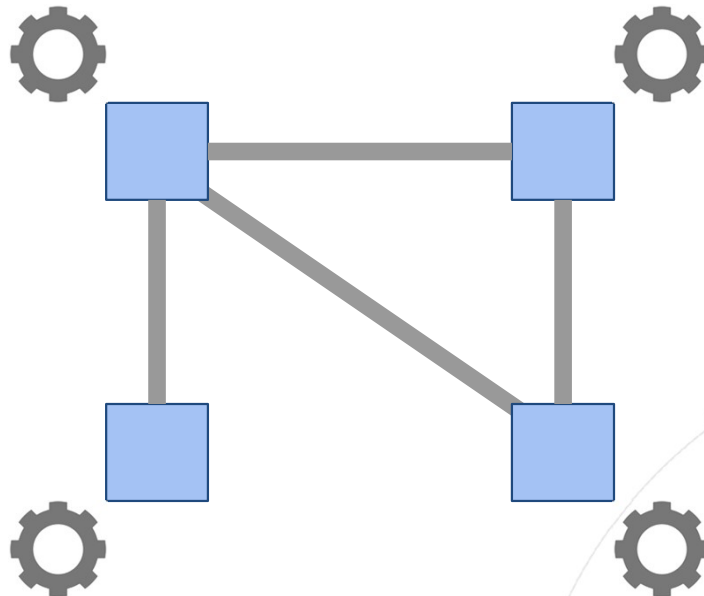
Paradigma	Accent pe	Rezultat
Procedurală	Acțiuni	Proceduri
Funcțională	Relații	Funcții în sens matematic





# Paradigme de programare

Paradigma	Accent pe	Rezultat
Procedurala	Actiuni	Proceduri
Funcțională	Relații	Funcții în sens matematic
Orientată obiect	Entități	Clase și obiecte



# Paradigme de programare

- cele 3 paradigme sunt echivalente între ele
- orice program scris într-o paradigmă poate fi tradus într-un program din oricare altă paradigmă



# Paradigmele limbajului Python

- python este un limbaj multiparadigmă
- îmbină toate cele 3 tipuri:
  - Procedural
  - Funcțional
  - Orientat obiect

# Funcții

- o funcție este un subprogram sau o procedură
- permite reutilizarea codului și evitarea duplicării lui
- se definește cu cuvântul cheie **def**

```
def beautiful_print(name) :  
    print(f"Hello, {name}!")
```

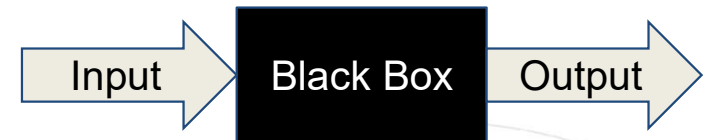
Semnătura funcției

Corpul funcției

# Funcții

- o funcție returnează implicit **None**
- o bună practică este ca funcția să fie văzută ca un black box, nu modifică nimic din afara corpului ei

```
def add_elem(l, e):  
    l.append(e)  
    return l
```



# Funcții

Se pot modifica și variabile ce nu aparțin corpului funcției.

```
no_calls = 0
```

```
def call(phone_number):  
    global no_calls  
    no_calls += 1
```

# Funcții

Definirea unei funcții fără corp este posibilă folosind cuvântul cheie **pass**.

```
def count_atoms_on_jupiter():  
    pass
```



# Parametrii impliciți

Parametrii impliciți se definesc cu =.

```
def greet(name='Monica') :  
    print(f"Hi, {name}!")
```

```
greet()                # Hi,  
Monica!  
greet('Daniel')        # Hi, Daniel!
```



# Număr variabil de parametrii

- o funcție a cărei număr de parametrii este necunoscut, se numește funcție cu număr variabil de parametrii
- parametrii sunt văzuți ca o listă

```
def my_sum(*args):  
    sum = 0  
    for num in args:  
        sum += num  
    return sum  
print(my_sum(1, 2, 3, 4))           # 10
```

# Parametrii cheie valoare

- parametrii unei funcții pot fi direct specificați prin nume
- parametrii sunt văzuți ca un dicționar

```
def capitals(**kwargs):  
    for country, capital in kwargs.items():  
        print(f'{country}: {capital}')
```

```
capitals(Romania='Bucharest', Italy='Rome')  
# Romania: Bucharest  
# Italy: Rome
```

# Funcții anonime

- sunt folosite în special în **programarea funcțională**
- o funcție care nu are nume se numește funcție anonimă sau **funcție lambda**
- evită definirea unei funcții obișnuite pentru o serie de operații simple.



# Funcții anonime

- pot fi declarate oriunde în cod și pot fi atribuite variabilelor
- valoarea de retur a funcției este implicită expresiei efectuate

```
sum2 = lambda x, y : x + y
```

Definirea unei funcții anonime

Parametrii

Expresia efectuată

```
print(sum2(2, 3))      # 5
```

# Predicat

Se referă la o funcție ce întoarce valoarea de adevăr a unei expresii.

```
is_even = lambda x : x % 2 == 0
```

Definirea unei funcții anonime

Parametru

Expresia efectuată

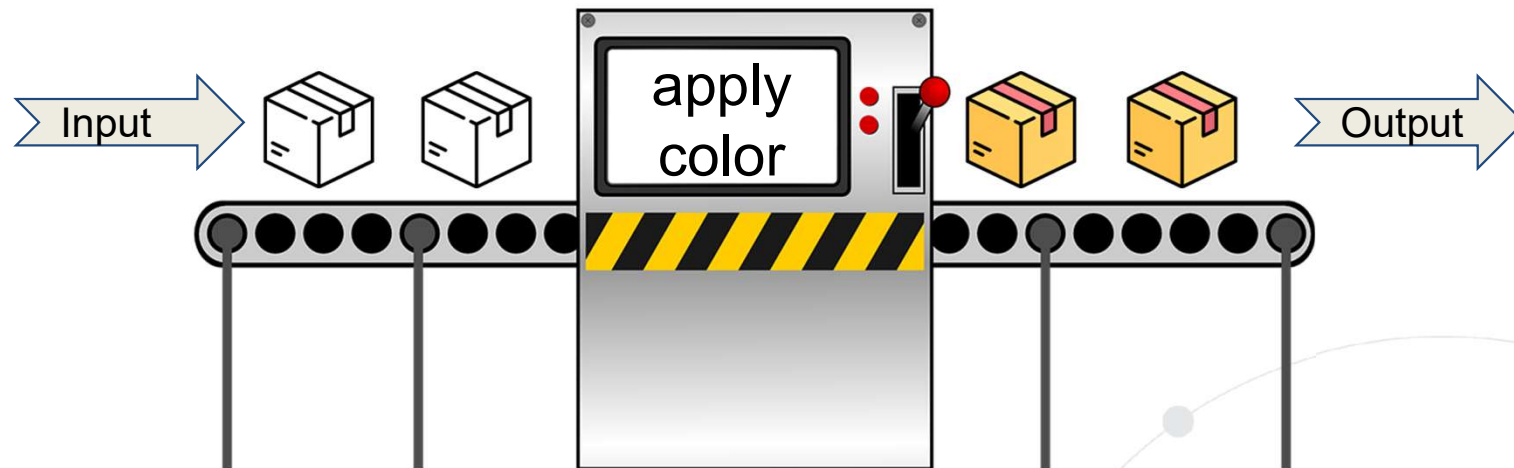
```
print(is_even(2))           # True
```

# Funcționale

- este abordarea funcțională a unor proceduri
- scriem cod mai puțin și mai expresiv.
- funcționale des utilizate:
  - Map
  - Filter
- mai există:
  - Reduce
  - Zip

# Funcționala map

- poate fi asemănată cu benzi de asamblare
- aplică o operație tuturor obiectelor puse pe bandă



# Funcționala map

Primește o funcție și o colecție și întoarce colecția formată prin aplicarea funcției asupra tuturor elementelor colecției.

```
values = [1, 2, 3]
```

```
triple = lambda x : x * 3
```

```
triple_values = list(map(triple, values))
```

```
print(triple_values)      # [3, 6, 9]
```



# Funcționala map

Operația efectuată poate fi definită și ca o funcție normală.

```
values = [1, 2, 3]
```

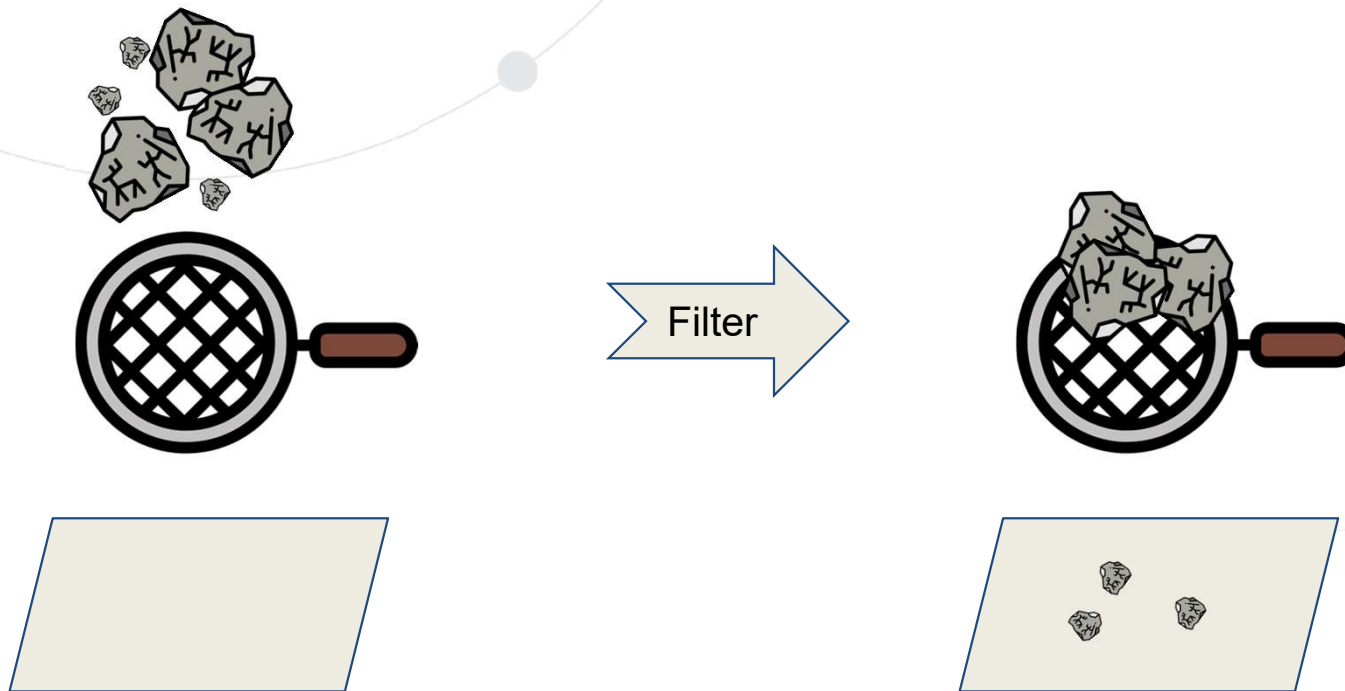
```
def triple(x):  
    return x * 3
```

```
triple_values = list(map(triple, values))
```

```
print(triple_values)      # [3, 6, 9]
```

# Funcționala filter

Poate fi asemănată cu o sită. Anumite obiecte pot trece prin sită, altele nu.



# Funcționala filter

Primește un predicat și o colecție și întoarce colecția formată din elementele care respectă predicatul.

```
values = [1, 2, 3]
```

```
is_odd = lambda x : x % 2 != 0
```

```
odd_values = list(filter(is_odd, values))
```

```
print(odd_values)    # [1, 3]
```

# Funcționala filter

Operația efectuată poate fi definită și ca o funcție normală.

```
values = [1, 2, 3]
```

```
def is_odd(x):  
    return x % 2 != 0
```

```
odd_values = list(filter(is_odd, values))
```

```
print(odd_values)    # [1, 3]
```

# Procedural vs funcțional

Afișați pătratul elementelor unei liste care sunt divizibile cu 7.

```
my_list = [7, 8, 14]
result = []

for e in my_list:
    if e % 7 == 0:
        result.append(e ** 2)
```

# Procedural vs funcțional

Afișați pătratul elementelor unei liste care sunt divizibile cu 7.

```
my_list = [7, 8, 14]
```

```
result = [i ** 2 for i in my_list if i % 7 == 0]
```

# Procedural vs funcțional

Afișați pătratul elementelor unei liste care sunt divizibile cu 7.

```
my_list = [7, 8, 14]
```

```
square = lambda x : x ** 2  
div7 = lambda x : x % 7 == 0
```

```
result = list(map(square, filter(div7, my_list)))
```

The background features several decorative elements: a large blue shape on the left, a thin grey arc connecting two points on the left and a point on the right, and a blue semi-circle at the bottom right.

# Întrebări?





**Pauză**

The background features several decorative elements: a large blue shape on the left, a blue semi-circle at the bottom right, and thin white curved lines with small grey dots that sweep across the slide.

# Iteratori și tipuri de evaluare

# Reprezentarea obiectelor în memorie

- python, fiind un limbaj orientat obiect, are deviza “**totul** este un obiect”
- obiectele sunt stocate în memorie și accesate prin referință

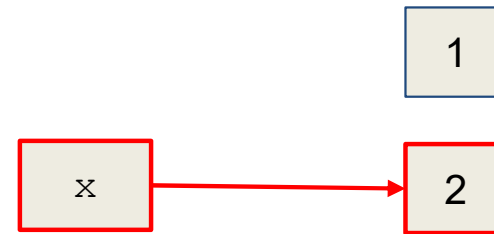
```
x = 3  
type(x)          #<class 'int'>
```

# Reprezentarea obiectelor în memorie

```
x = 1
```



```
x = 2
```



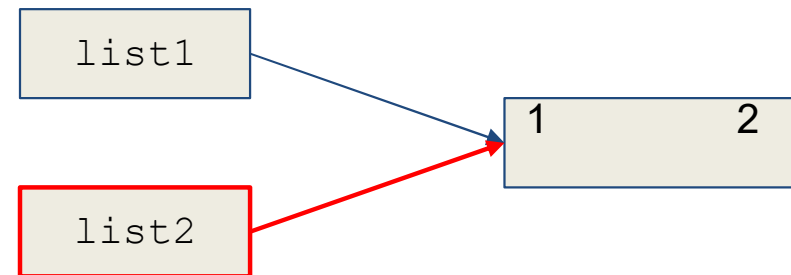
```
print(x)      #2
```

# Reprezentarea obiectelor în memorie

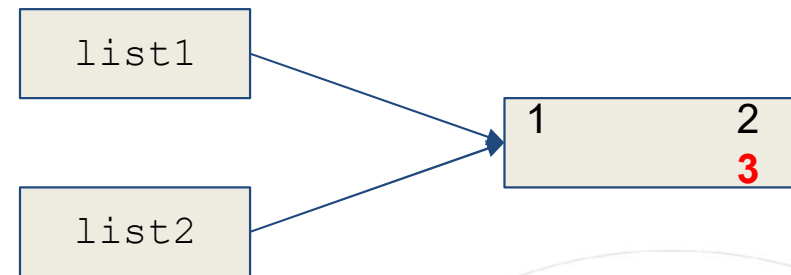
```
list1 = [1, 2]
```



```
list2 = list1
```



```
list2.append(3)
```

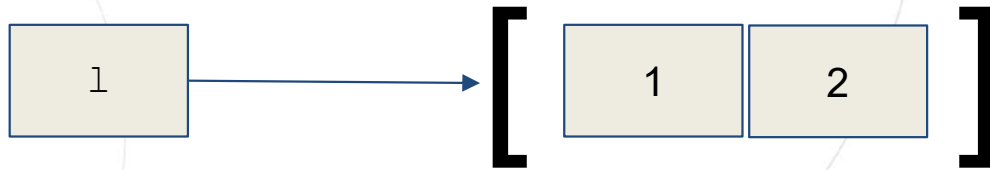


```
print(list1) # [1, 2, 3]
```

# Iterator

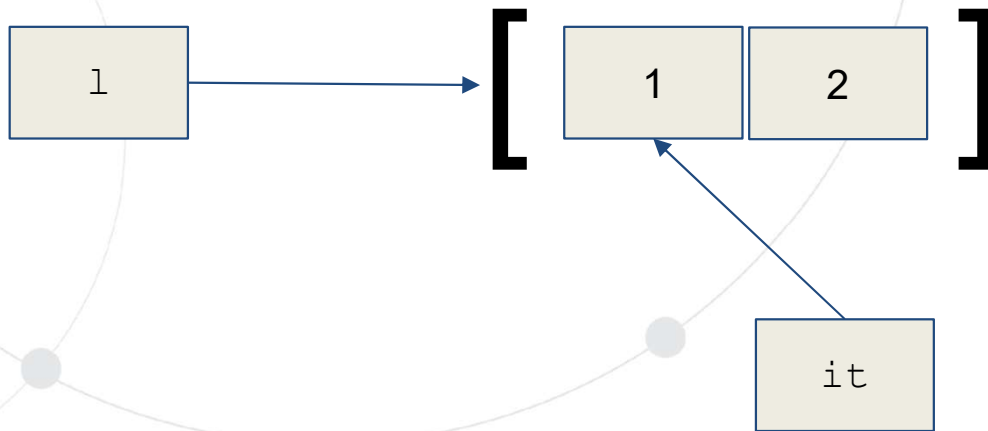
- este obiectul folosit la parcurgere unei colecții
- ne arată care este elementul următor
- un iterator peste o colecție se creează cu **`iter(colecție)`**
- putem extrage un element și avansa la următorul cu **`next(colecție)`**

# Iterator



`1 = [1, 2]`

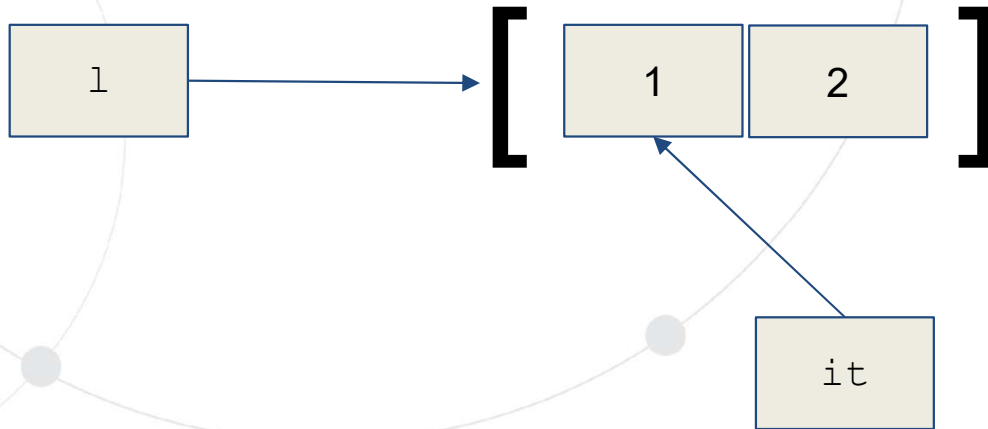
# Iterator



```
1 = [1, 2]  
it = iter(1)
```

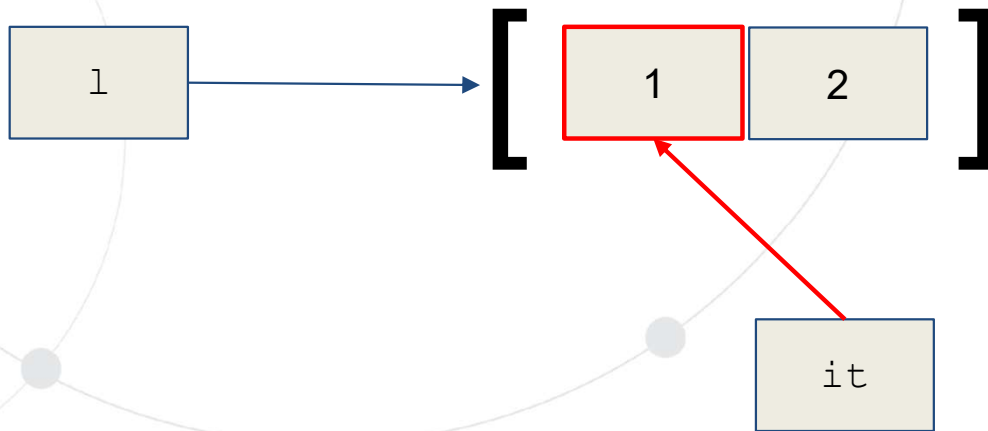


# Iterator



```
1 = [1, 2]
it = iter(1)
print(next(it))
```

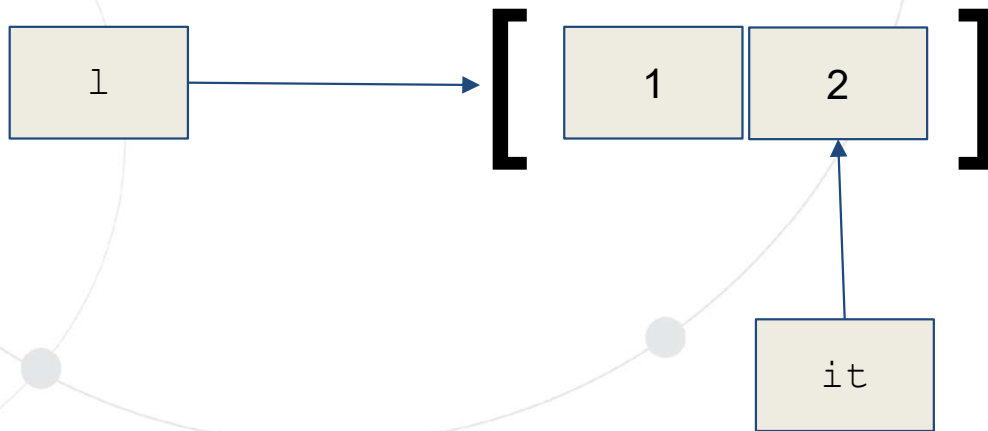
# Iterator



```
1 = [1, 2]  
it = iter(1)  
print(next(it))
```

# 1

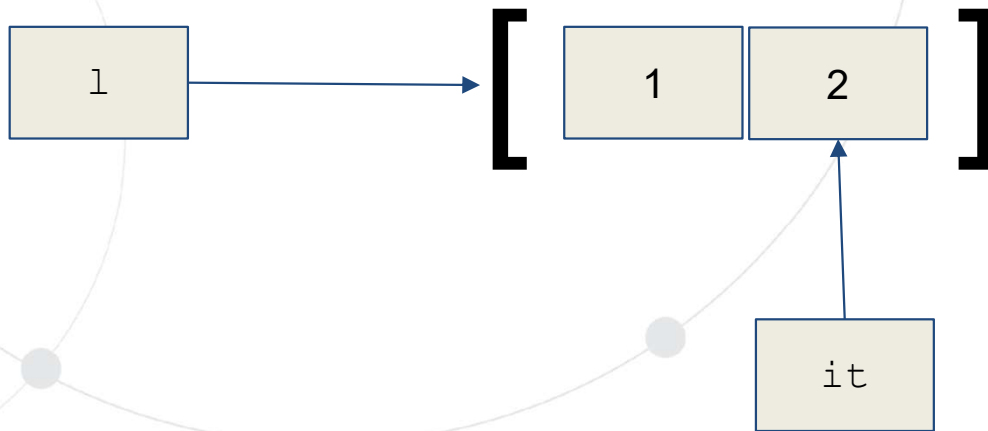
# Iterator



```
1 = [1, 2]  
it = iter(1)  
print(next(it))
```

# 1

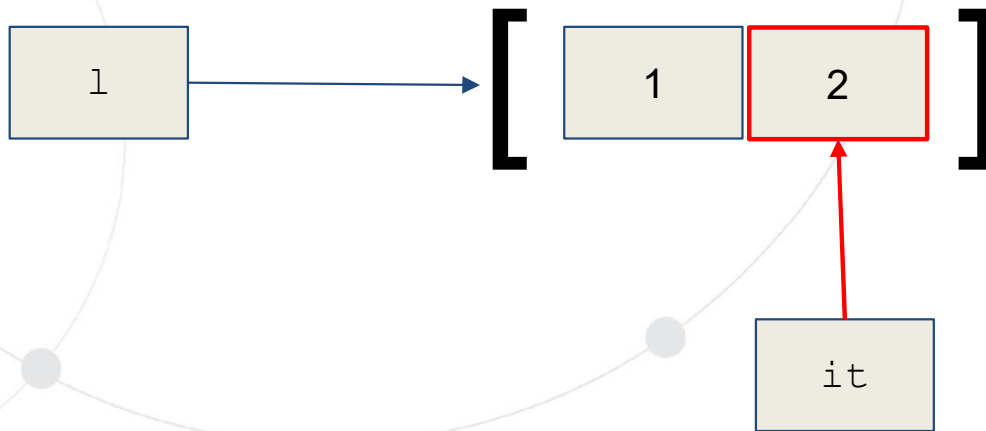
# Iterator



```
l = [1, 2]
it = iter(l)
print(next(it))
print(next(it))
```

# 1

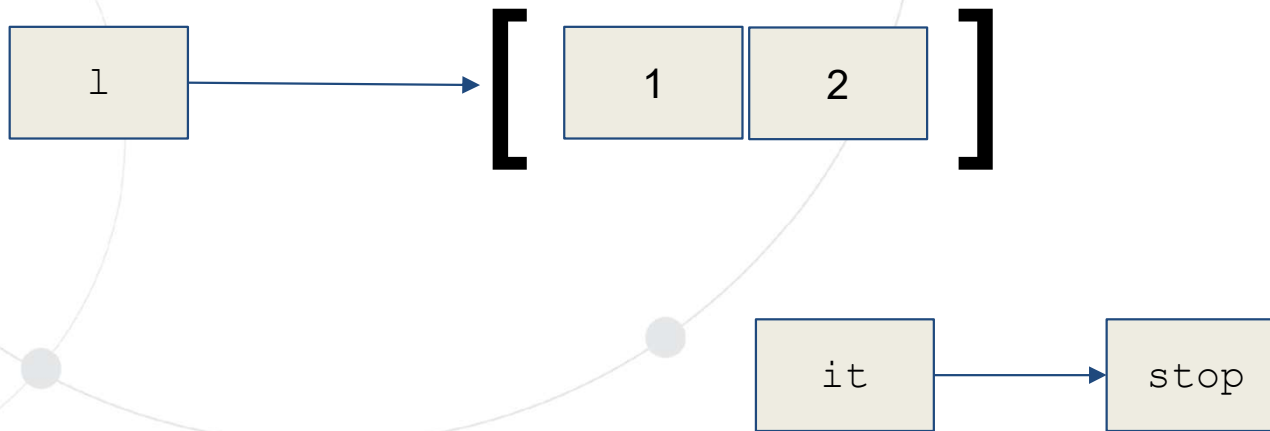
# Iterator



```
1 = [1, 2]
it = iter(1)
print(next(it))
print(next(it))
```

```
# 1
# 2
```

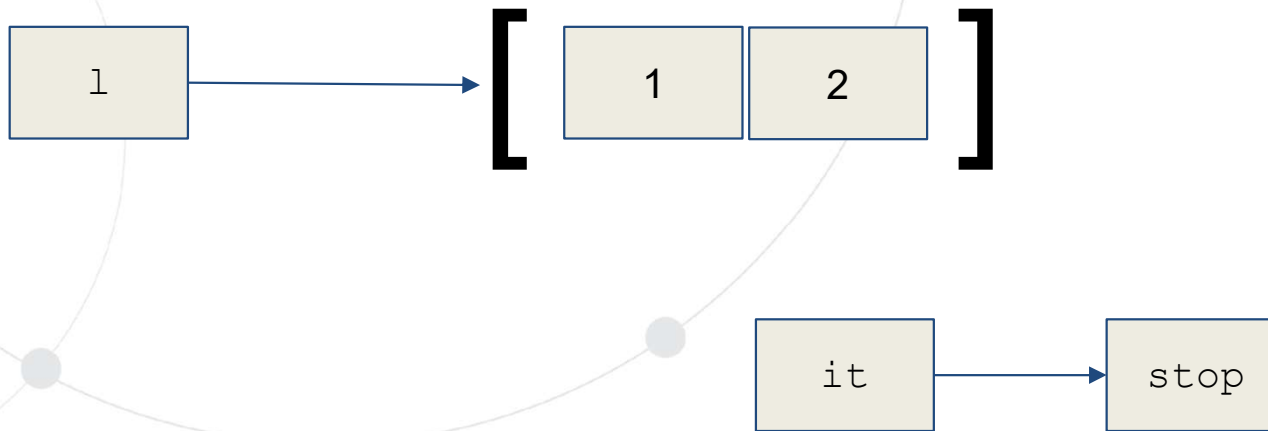
# Iterator



```
l = [1, 2]
it = iter(l)
print(next(it))
print(next(it))
```

```
# 1
# 2
```

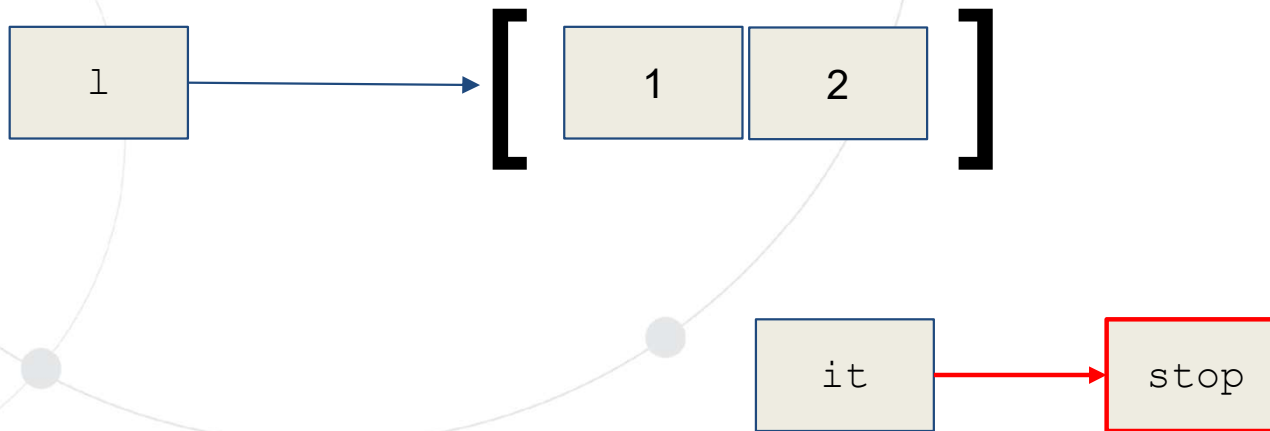
# Iterator



```
l = [1, 2]
it = iter(l)
print(next(it))
print(next(it))
print(next(it))
```

```
# 1
# 2
```

# Iterator



```
l = [1, 2]
it = iter(l)
print(next(it))
print(next(it))
print(next(it))
stop
```

```
# 1
# 2
# Exceptie de
```



# Iterator

- când folosim o buclă, colecția este parcursă **automat** cu un iterator

```
l = [1, 2]  
it = iter(l)
```

```
for x in it:  
    print(x) #1 #2
```

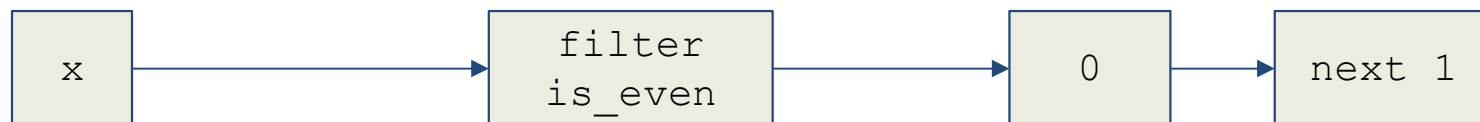
# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))  
  
for x in l:  
    print(x)
```

# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

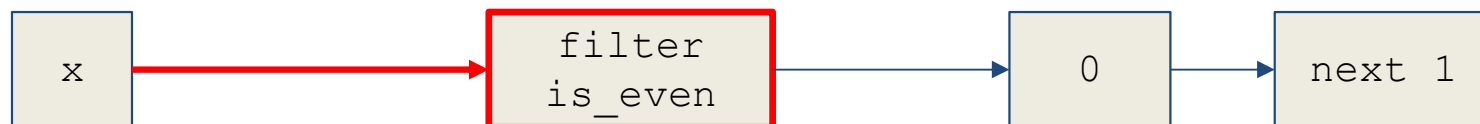
```
for x in l:  
    print(x)
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

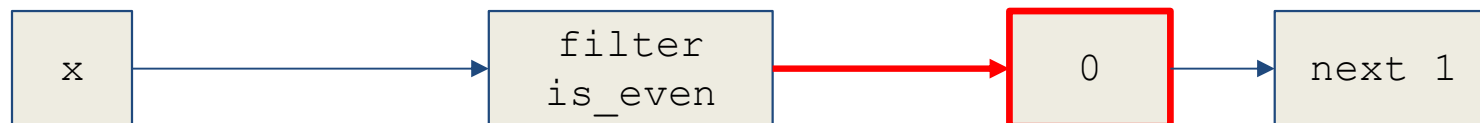
```
for x in l:  
    print(x)          #
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

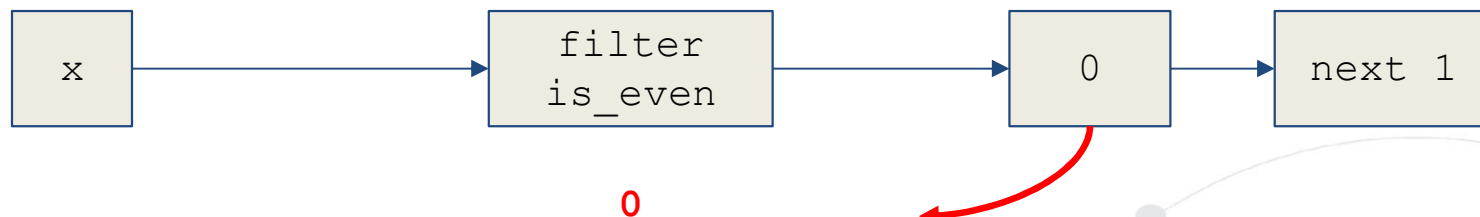
```
for x in l:  
    print(x)          #
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

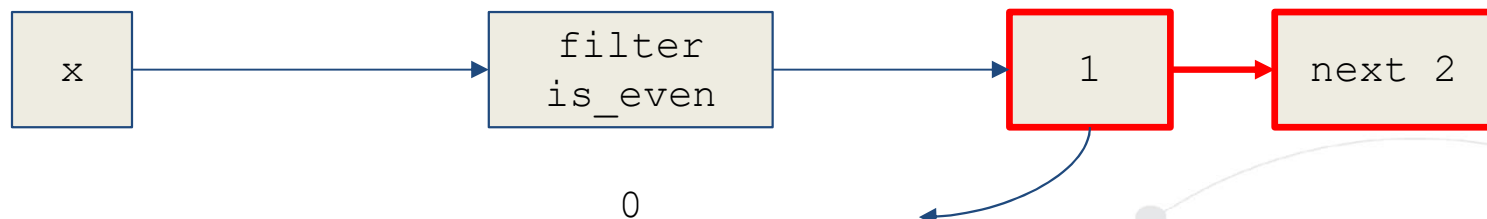
```
for x in l:  
    print(x)           #
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

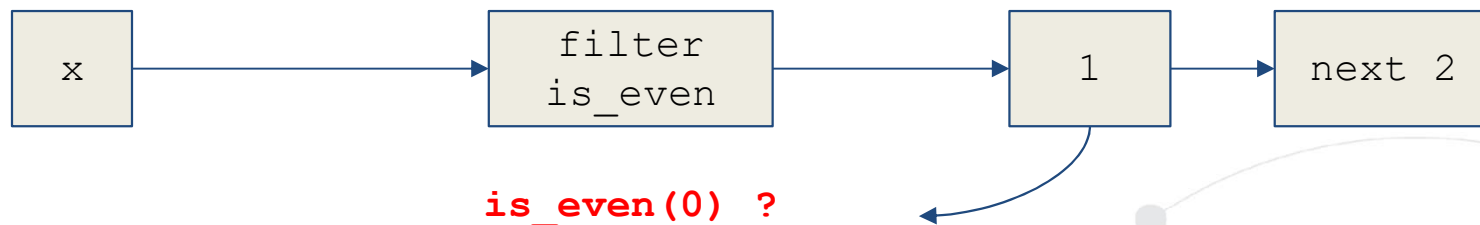
```
for x in l:  
    print(x)          #
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

```
for x in l:  
    print(x)          #
```

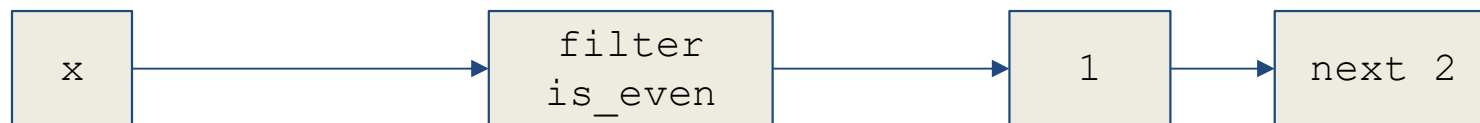




# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

```
for x in l:  
    print(x)          #
```

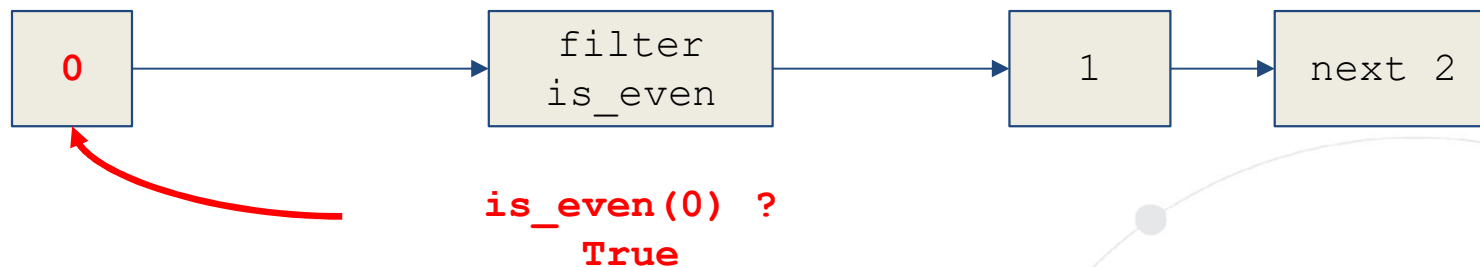


`is_even(0) ?`  
`True`

# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

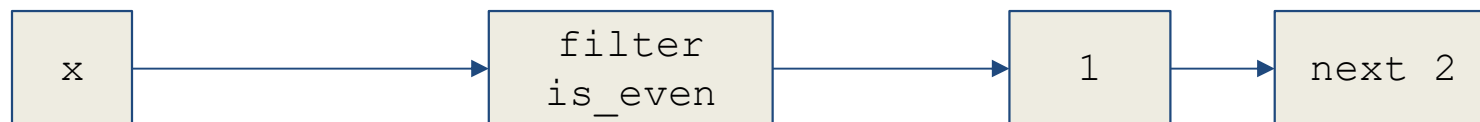
```
for x in l:  
    print(x)           # 0
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

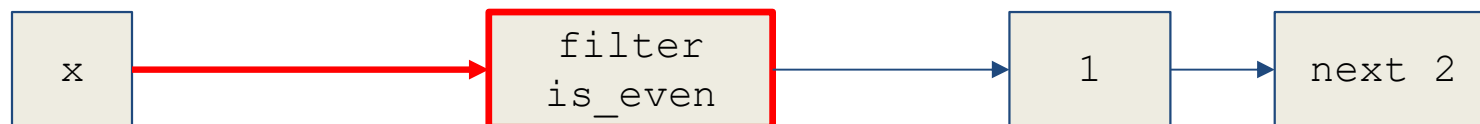
```
for x in l:  
    print(x)          # 0
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

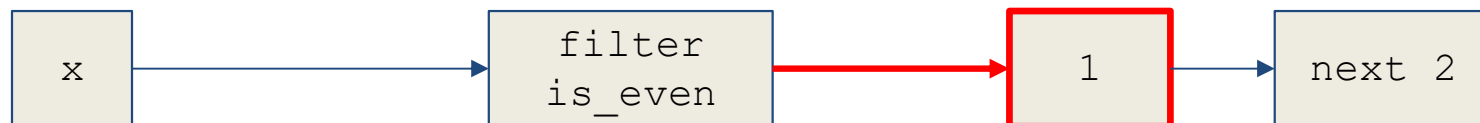
```
for x in l:  
    print(x)          #
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

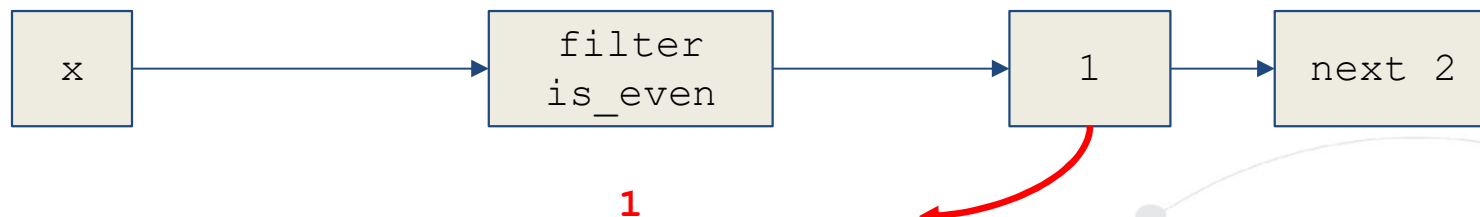
```
for x in l:  
    print(x)          #
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

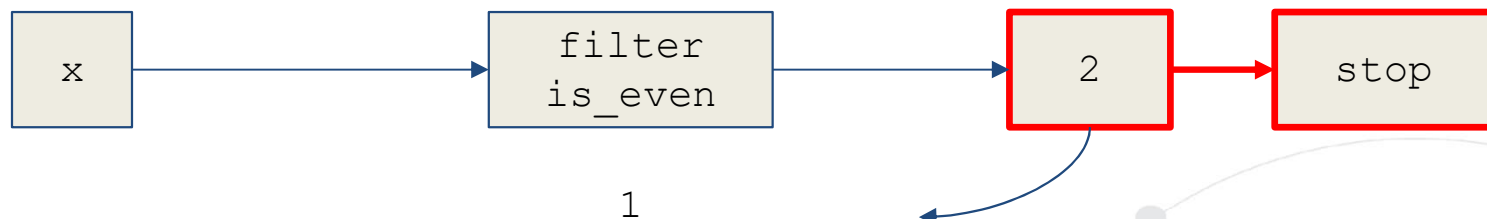
```
for x in l:  
    print(x)           #
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

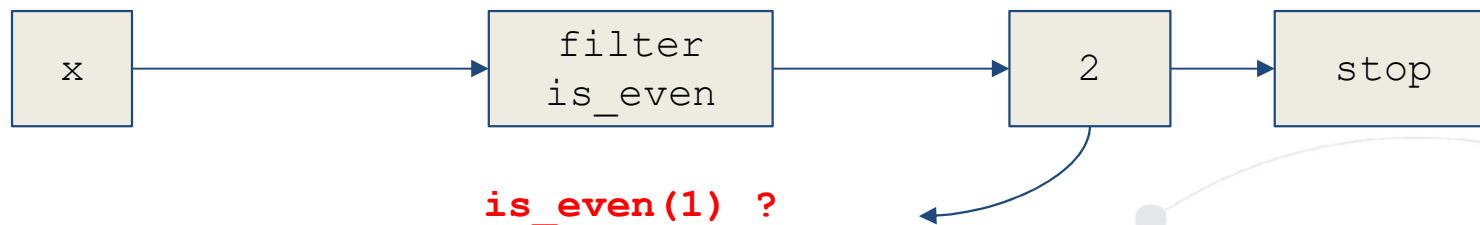
```
for x in l:  
    print(x)          #
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

```
for x in l:  
    print(x)          #
```

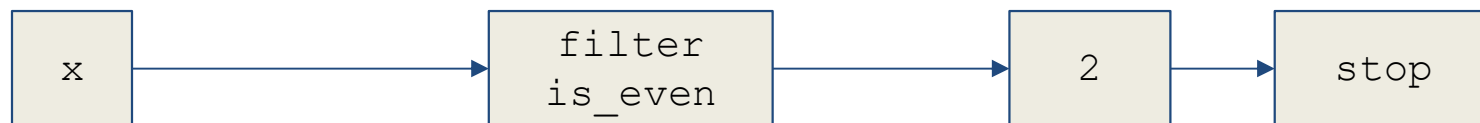




# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

```
for x in l:  
    print(x)          #
```

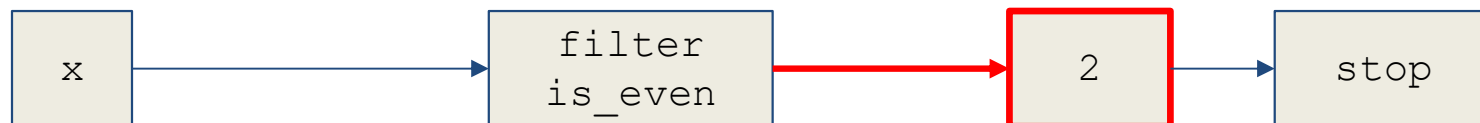


**is\_even(1) ?**  
**False**

# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

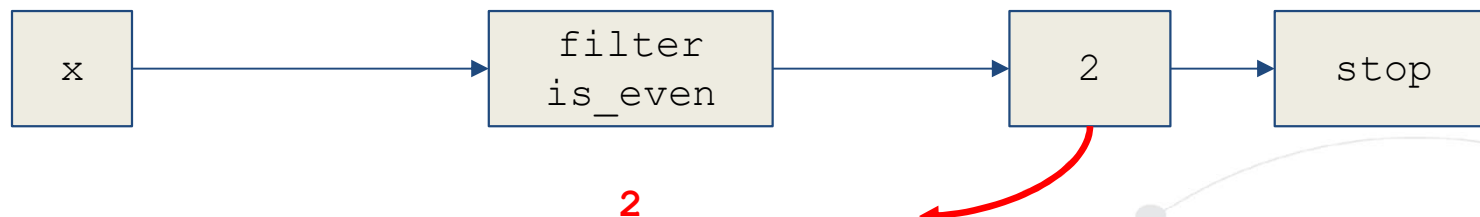
```
for x in l:  
    print(x)          #
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

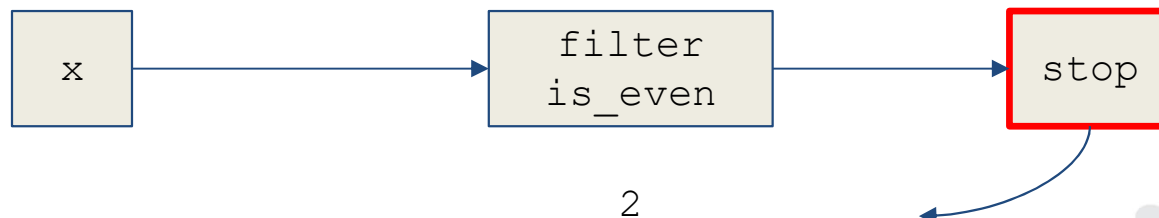
```
for x in l:  
    print(x)          #
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

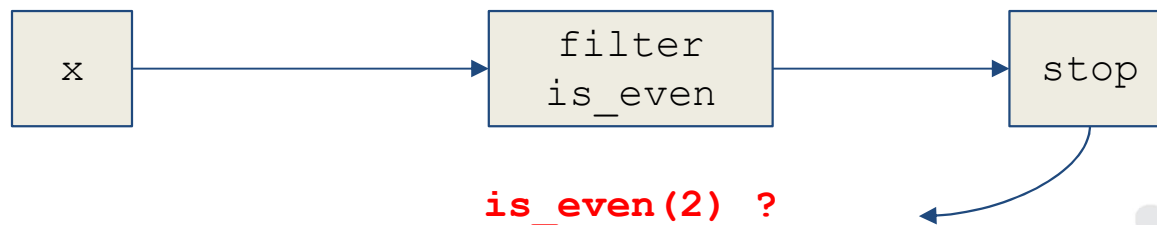
```
for x in l:  
    print(x)          #
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

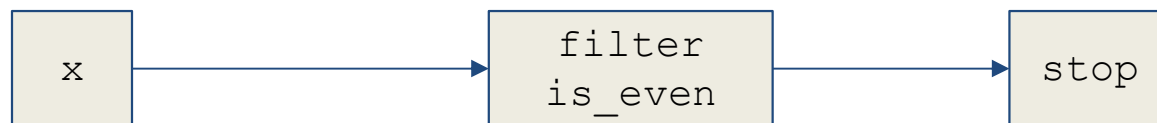
```
for x in l:  
    print(x)          #
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

```
for x in l:  
    print(x)          #
```

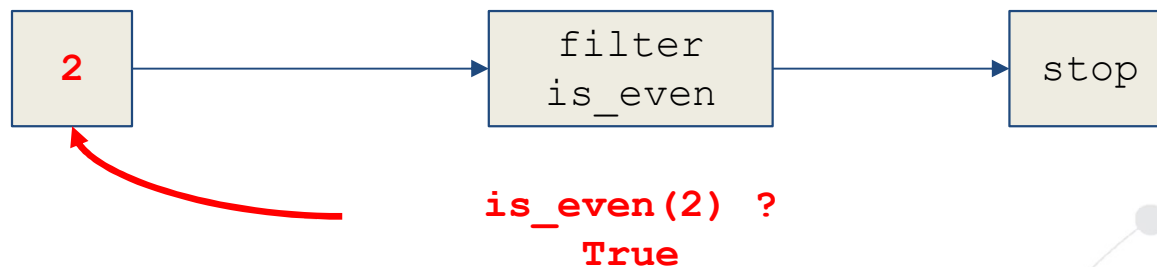


**is\_even(2) ?**  
**True**

# Funzionale

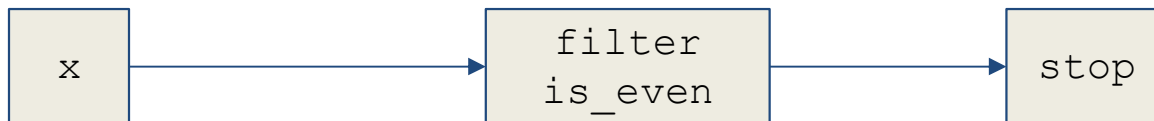
```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

```
for x in l:  
    print(x)           # 2
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))  
  
for x in l:  
    print(x)           # 2
```

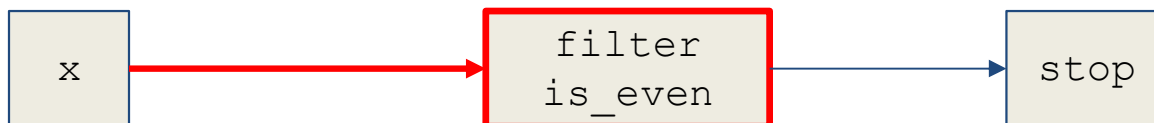




# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

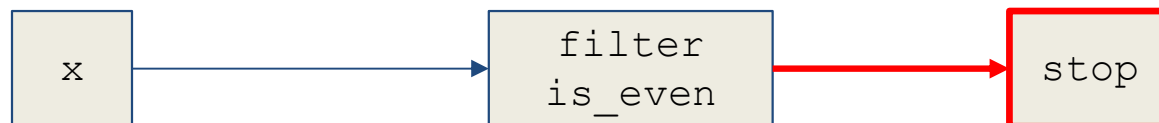
```
for x in l:  
    print(x)          #
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

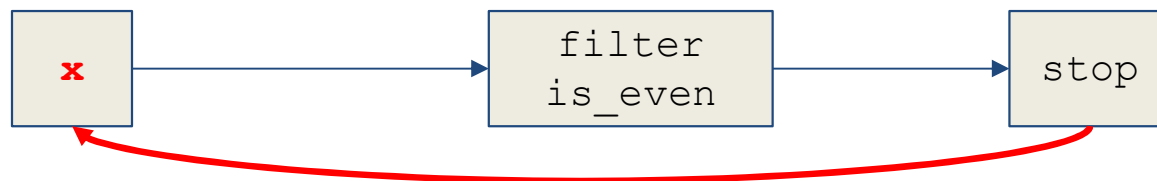
```
for x in l:  
    print(x)          #
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

```
for x in l:  
    print(x)          #
```



# Funzionale

```
is_even = lambda x : x % 2 == 0  
l = filter(is_even, range(3))
```

```
for x in l:  
    print(x)
```

```
# 0
```

```
# 2
```

# Întrebări?

**Nu uitati de feedback: [aici](#)**