

Programmazione avanzata a.a. 2020-21

A. De Bonis

## Introduzione a Python (III e IV parte)

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

1

1

## Funzioni in Python

- Le funzioni sono definite usando la keyword **def**
- Viene introdotto un nuovo identificatore (il nome della funzione)
- Devono essere specificati
  - Il **nome** e la lista dei **parametri**
  - La funzione può avere un numero di parametri variabile
- L'istruzione **return** (opzionale) restituisce un valore ed interrompe l'esecuzione della funzione

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

2

2

## Esempi

```
def contains(data, target):
    for item in data:
        if item == target:
            return True
    return False
```

```
def count(data, target):
    n = 0
    for item in data:
        if item == target:
            n += 1
    return n
```

```
def sum(values):
    total = 0
    for v in values:
        total = total + v
    return total
```

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

3

3

## Esempi

```
def bubble_sort(a):
    n=len(a)
    while(n>0):
        for i in range(0,n-1):
            if(a[i]>a[i+1]):
                a[i], a[i+1] = a[i+1], a[i]
        n -= 1
    return a
```

Assegnamento multiplo  
swap in un rigo

```
a = [5, 3, 1, 7, 8, 2]
print(a)
bubble_sort(a)
print(a)
```

Il parametro a è passato  
per riferimento

```
[5, 3, 1, 7, 8, 2]
[1, 2, 3, 5, 7, 8]
```

```
a = [5, 3, 1, 7, 8, 2]
print('a =', a)
b = bubble_sort(a[:])
print('b =', b)
print('a =', a)
```

```
a = [5, 3, 1, 7, 8, 2]
b = [1, 2, 3, 5, 7, 8]
a = [5, 3, 1, 7, 8, 2]
```

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

4

4

## Stringa di documentazione

- La prima riga di codice nella definizione di una funzione dovrebbe essere una breve spiegazione di quello che fa la funzione
  - docstring

```
def my_function():
    """Do nothing, but document it. ...

    No, really, it doesn't do anything.
    """
    pass # Istruzione che non fa niente
```

```
print(my_function.__doc__)
```



```
Do nothing, but document it. ...

No, really, it doesn't do anything.
```

5

## Variabili globali

- Nel corpo di una funzione si può far riferimento a variabili definite nell'ambiente (scope) esterno alla funzione, ma tali variabili non possono essere modificate
- Per poterle modificare bisogna dichiararle **global** nella funzione
  - Se si prova ad accedere ad esse senza dichiararle global viene generato un errore

6

## Esempi

```
n = 111
def f():
    print('nella funzione n =', n)

f()
print('fuori la funzione n =', n)
```

→

```
nella funzione n = 111
fuori la funzione n = 111
```

```
m = 999
def f1():
    m = 1
    print('nella funzione m =', m)

f1()
print('fuori la funzione m =', m)
```

→

```
nella funzione m = 1
fuori la funzione m = 999
```

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

7

7

## Esempi

```
m=999
def f3():
    print('nella funzione m =', m)
    m = 1

f3()
print('fuori la funzione m =', m)
```

↘

```
UnboundLocalError: local variable 'm' referenced before assignment
```

```
n = 777
def varGlobaliQuattro():
    global n
    print('nella funzione n =', n)
    n=3

print('fuori la funzione n =', n)
varGlobaliQuattro()
print('fuori la funzione n =', n)
```

→

```
fuori la funzione n = 777
nella funzione n = 777
fuori la funzione n = 3
```

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

8

8

## Parametri di una funzione

- Parametri **formali** di una funzione
  - Identificatori usati per descrivere i parametri di una funzione nella sua definizione
- Parametri **attuali** di una funzione
  - Valori passati alla funzione all'atto della chiamata
  - Argomenti di una funzione
- Argomento **keyword**
  - Argomento preceduto da un identificatore in una chiamata a funzione
- Argomento **posizionale**
  - Argomento che non è un argomento keyword

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

9

9

## Passaggio dei parametri

- Il passaggio dei parametri avviene tramite un riferimento ad oggetti
  - Per valore, dove il valore è il riferimento (puntatore) dell'oggetto passato

```
lst = [1, 'due']

def modifica(lista):
    lista.append('nuovo')

print('lista =', lst)
modifica(lst)
print('lista =', lst)
```



```
lista = [1, 'due']
lista = [1, 'due', 'nuovo']
```

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

10

10

## Parametri di default

- Nella definizione della funzione, ad ogni parametro formale può essere assegnato un valore di default
  - a partire da quello più a destra
- La funzione può essere invocata con un numero di parametri inferiori rispetto a quello con cui è stata definita

```
def default(a, b=3):
    print('a =', a, 'b =', b)

default(2)
default(1,1)
```

→

```
a = 2 b = 3
a = 1 b = 1
```

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

11

11

## Parametri di default

- Gli argomenti di default devono sempre seguire quelli non di default.
  - la funzione `f` nel riquadro è definita in modo sbagliato

```
>>> def f(a=1,b):
...     print(a,b)
...
File "<stdin>", line 1
SyntaxError: non-default argument follows default argument
```

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

12

12

## Attenzione

- I parametri di default sono valutati nello scope in cui è definita la funzione

```
d = 666
def default_due(a, b=d):
    print('a =', a, 'b =', b)
```

```
d = 0
default_due(11)
default_due(22,33)
```

```
a = 11 b = 666
a = 22 b = 33
```

13

## Attenzione

- I parametri di default sono valutati solo una volta (quando si definisce la funzione)
  - **Attenzione a quando il parametro di default è un oggetto mutable**

```
def f(a, L=[]):
    L.append(a)
    return L
```

```
print(f(1))
print(f(2))
print(f(3))
```

La lista L conserva il proprio valore tra chiamate successive, non è inizializzata ad ogni chiamata

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

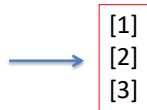
14

## Attenzione

- Se non si vuole che il parametro di default sia condiviso tra chiamate successive si può adottare la seguente tecnica (lo si inizializza nel corpo della funzione)

```
def f(a, L=None):
    if L is None:
        L = []
    L.append(a)
    return L

print(f(1))
print(f(2))
print(f(3))
```



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

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

15

15

## Numero variabile di argomenti

- In Python si possono definire funzioni con un numero variabile di parametri
- L'ultimo parametro è preceduto da \*
- Dopo ci possono essere solo parametri keyword (dettagli in seguito)
- Il parametro formale preceduto da \* indica la sequenza in cui sono contenuti un numero variabile di parametri
  - Nel corpo della funzione possiamo accedere al valore di questi parametri tramite la posizione

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

16

16



## Esempio

```
def variabili(v1, v2=4, *arg):
    print('primo parametro =', v1)
    print('secondo parametro =', v2)
    print('# argomenti passati', len(arg) + 2)
    if arg:
        print('# argomenti variabili', len(arg))
        print('arg =', arg)
        print('primo argomento variabile =', arg[0])
    else:
        print('nessun argomento in più')
```

variabili(1, 'a', 4, 5, 7)

```
primo parametro = 1
secondo parametro = a
# argomenti passati 5
# argomenti variabili 3
arg = (4, 5, 7)
primo argomento variabile = 4
```

variabili(3, 'b')

```
primo parametro = 3
secondo parametro = b
# argomenti passati 2
nessun argomento in più
```

17

## L'operatore \*

- Ogni tipo iterabile può essere spaccettato usando l'operatore \* (unpacking operator).
- Se in un assegnamento con due o più variabili a sinistra dell'assegnamento, una di queste variabili è preceduta da \* allora i valori a destra sono assegnati uno ad uno alle variabili (senza \*) e i restanti valori vengono assegnati alla variabile preceduta da \*.
- Possiamo passare come argomento ad una funzione che ha k parametri posizionali una collezione iterabile di k elementi preceduta da \*
  - Questo è diverso dal caso in cui utilizziamo \* davanti ad un parametro formale

18

## Esempi di uso di \*

```
>>> primo, secondo, *rimanenti = [1,2,3,4,5,6]
>>> primo
1
>>> secondo
2
>>> rimanenti
[3, 4, 5, 6]
```

```
>>> primo, *rimanenti, sesto, = [1,2,3,4,5,6]
>>> primo
1
>>> sesto
6
>>> rimanenti
[2, 3, 4, 5]
```

19

## Esempi di uso di \*

```
def variabili(v1, v2=4, *arg):
    print('primo parametro =', v1)
    print('secondo parametro =', v2)
    print('# argomenti passati', len(arg) + 2)
    if arg:
        print('# argomenti variabili', len(arg))
        print('arg =', arg)
        print('primo argomento variabile =', arg[0])
    else:
        print('nessun argomento in più')
```

```
variabili(1, 'a', 4, 5, 7)
```

```
L=[4,5,7]
variabili(1,'a',*L)
```

```
primo parametro = 1
secondo parametro = a
# argomenti passati 5
# argomenti variabili 3
arg = (4, 5, 7)
primo argomento variabile = 4
```

20

## Esempi di uso di \*

```
def somma(addendo1, addendo2, addendo3):
    return addendo1+addendo2+addendo3

addendi=[56,2,4]

print("somma =",somma(*addendi))
```

**Attenzione:**  
**addendi** deve  
 contenere  
 esattamente 3  
 elementi

```
somma = 62
```

21

## Unpacking

- Quando a sinistra di un assegnamento ci sono due o più variabili e a sinistra c'è una sequenza, la collezione viene spaccata e gli elementi assegnati alle variabili a sinistra
  - Lo abbiamo già visto per le tuple
- Esempio:

```
>>> l=[1,2,3,4]
>>> a,b,c,d = l
>>> a
1
>>> b
2
>>> c
3
>>> d
4
```

22

## Parametri keyword

- Sono argomenti di una funzione preceduti da un identificatore oppure passati come dizionario (**dict**) preceduto da **\*\***
- Un argomento keyword può essere specificato anche assegnando esplicitamente, attraverso il nome, un parametro attuale ad un parametro formale
- Nella definizione di una funzione i parametri keyword possono essere rappresentati dall'ultimo parametro della funzione preceduto da **\*\***
  - Il parametro è considerato un dizionario (**dict**)

23

## L'operatore **\*\***

- L'operatore **\*\*** è il mapping unpacking operator e può essere applicato ai tipi mapping (collezione di coppie chiave-valore) quali i dizionari per produrre una lista di coppie chiave-valore adatta ad essere passata come argomento ad una funzione.

24

## Esempio

Qui `cmd` è un dizionario

```
def esempio_kw(arg1, arg2, arg3, **cmd):
    if cmd.get('operatore') == '+':
        print('La somma degli argomenti è: ', arg1 + arg2 + arg3)
    elif cmd.get('operatore') == '*':
        print('Il prodotto degli argomenti è: ', arg1 * arg2 * arg3)
    else:
        print('operatore non supportato')

    if cmd.get('azione') == "stampa":
        print('arg1 =', arg1, 'arg2 =', arg2, 'arg3 =', arg3)
```

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

25

25

## Esempio

```
esempio_kw(2, 3, 4, operatore='+')
```

La somma degli argomenti è: 9

```
esempio_kw(2, 3, 4, operatore='*')
```

Il prodotto degli argomenti è: 24

```
esempio_kw(2, 3, 4, operatore='/')
```

operatore non supportato

```
esempio_kw(2, 3, 4, operatore='+', azione='stampa')
```

La somma degli argomenti è: 9  
arg1 = 2 arg2 = 3 arg3 = 4

```
esempio_kw(2, 3, 4, **{'operatore': '+', 'azione': 'stampa'})
```

La somma degli argomenti è: 9  
arg1 = 2 arg2 = 3 arg3 = 4

```
diz= {'operatore': '+', 'azione': 'stampa'}
esempio_kw(2, 3, 4, **diz)
```

La somma degli argomenti è: 9  
arg1 = 2 arg2 = 3 arg3 = 4

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

26

26

## Esempio

Parametri variabili
Parametro keyword

```
def concat(*args, sep="/"):
    return sep.join(args)

print(concat('ciao','a','tutti', sep='/'))
print(concat('ciao','a','tutti', sep='.'))
```

ciao/a/tutti  
 ciao.a.tutti

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

27

## Riassumendo

- Ci sono due modi per assegnare valori ai parametri formali di una funzione
- Secondo la **posizione**
  - Parametri *tradizionali*
  - Parametri *di default*
  - Gli argomenti posizionali non hanno keyword e devono essere assegnati per primi
  - La posizione è importante
- Secondo la **keyword**
  - Gli argomenti keyword hanno keyword e sono assegnati in seguito, dopo i parametri posizionali
  - La posizione non è importante
    - def f(x, a, b): ...
    - f('casa', a=3, b=7) è la stessa cosa di f('casa', b=7, a=3)

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

28

## Riassumendo

- Una funzione può anche essere definita con tutti e tre i tipi di parametri
  - Parametri posizionali
    - Non inizializzati e di default
  - Parametro variabile
  - Parametri keyword

```
def tutti(arg1, arg2=222, *args, **kwargs):
    #Corpo della funzione
```

29

## Esempio

```
def tutti(arg1, arg2=222, *args, **kwargs):
    print('arg1    =', arg1)
    print('arg2    =', arg2)
    print('*args   =', args)
    print('**kwargs =', kwargs)
```

```
tutti('prova', 999, 'uno', 2, 'tre', a=1, b='sette')
```

```
arg1    = prova
arg2    = 999
*args   = ('uno', 2, 'tre')
**kwargs = {'a': 1, 'b': 'sette'}
```

```
tutti('seconda prova')
```

```
arg1    = seconda prova
arg2    = 222
*args   = ()
**kwargs = {}
```

30

## Annotazioni

- Le annotazioni sono dei metadati associati alle funzioni definite dal programmatore
- Sono memorizzate come un dizionario nell'attributo `__annotation__` della funzione
- Non hanno nessun effetto sulla funzione
- Servono ad indicare il tipo dei parametri e del valore eventualmente restituito

31

## Annotazioni

- L'**annotazione di parametri** è definita da `:` dopo il nome del parametro seguito da un'espressione che, una volta valutata, indica il tipo del valore dell'annotazione
- Le **annotazioni di ritorno** sono definite da `->` seguita da un'espressione e sono poste tra la lista dei parametri e i due punti che indicano la fine dell'istruzione `def`

32



## Esempio

```
def saluta(nome: str, età: int = 23) -> str:
    print('Ciao ', nome, 'hai ', età, ' anni')
    return nome + ' ' + str(età)
```

```
s=saluta('mario')
print(s)
s=saluta('luisa', 21)
print(s)
```

```
Ciao mario hai 23 anni
mario 23
Ciao luisa hai 21 anni
luisa 21
```

```
print(saluta.__annotations__)
```

```
{'età': <class 'int'>, 'nome': <class 'str'>, 'return': <class 'str'>}
```

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

33

33

## A cosa servono?

- Potrebbero essere utilizzate come help della funzione

```
def saluta(nome: 'rappresenta il nome dell\'utente ', età: int = 23) -> str:
    print('Ciao ', nome, 'hai ', età, ' anni')
    return nome + ' ' + str(età)
```

```
print(saluta.__annotations__)
```

```
{'età': <class 'int'>, 'nome': "rappresenta il nome dell'utente ", 'return': <class 'str'>}
```

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

34

34

## Funzioni come parametro di funzioni

- È possibile passare l'identificatore di una funzione **a** come parametro di un'altra funzione **b**
  - Si passa il riferimento alla funzione **a**
- Nel corpo della funzione **b**, si può invocare **a**
  - Come nome della funzione si usa il parametro formale specificato nella definizione della funzione **b**

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

35

35

```
def insertion_sort(a):
    for i in range(1,len(a)):
        val=a[i]
        j=i-1
        while (j>=0 and a[j]>val):
            a[j+1]=a[j]
            j=j-1
            a[j+1]=val
    return a
```

riferimento a funzione

## Esempio

```
def ordina(lista, metodo, copia=True):
    if copia == True:
        #si ordina una copia della lista
        return metodo(lista[:])
    else:
        return metodo(lista)
```

```
a = [5, 3, 1, 7, 8, 2]
print('a =', a)
b = ordina(a, insertion_sort)
print('a =', a)
print('b =', b)
print('-----')
a = [5, 3, 1, 7, 8, 2]
print('a =', a)
b = ordina(a, bubble_sort, copia=False)
print('a =', a)
print('b =', b)
```

```
a = [5, 3, 1, 7, 8, 2]
a = [5, 3, 1, 7, 8, 2]
b = [1, 2, 3, 5, 7, 8]
-----
a = [5, 3, 1, 7, 8, 2]
a = [1, 2, 3, 5, 7, 8]
b = [1, 2, 3, 5, 7, 8]
```

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

36

36

## Espressioni **lambda**

- Funzioni anonime create usando la keyword **lambda**
- **lambda** *a,b,c* : *a + b + c*
  - Restituiscono la valutazione dell'espressione presente dopo i due punti
    - Può essere presente solo un'istruzione
  - Possono far riferimento a variabili presenti nello scope (ambiente) in cui sono definite
  - Possono essere restituite da funzioni
    - Una funzione che restituisce una funzione
  - Possono essere assegnate ad un identificatore
- Maggiori dettagli in seguito

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

37

37

## Esempi

```
def f(x): return x**2
g = lambda x: x**2
```

```
print(g(3))
print(f(3))
```

f e g sono equivalenti, nel senso che producono lo stesso risultato

```
9
9
```

```
dati = [1, -4, 2, 7, -10, -3]
print(dati)
```

```
dati.sort(key=lambda x: abs(x))
print(dati)
```

Ordina la lista **dati** considerando il valore assoluto degli elementi

```
[1, -4, 2, 7, -10, -3]
```

```
[1, 2, -3, -4, 7, -10]
```

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

38

38

## Funzioni Python built-in

<code>abs()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>all()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>any()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>ascii()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bin()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>bool()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	
<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>	

39

## Output: funzione `print`

- Riceve un numero variabile di parametri da stampare e due parametri keyword (`end` e `sep`)
- Aggiunge automaticamente `\n` alla fine dell'output
- Parametri keyword (opzionali)
  - `sep` - stringa di separazione dell'output (default spazio)
  - `end` - stringa finale dell'output (default `\n`)
- Gli argomenti ricevuti sono convertiti in stringhe, separati da `sep` e seguiti da `end`

40

## Esempi

```
dati = [1, -4, 2, 7, -10, -3]
a = 1
b = 'a'
c = 'casa'
```

```
print(a, b, c, dati)
print(a, b, c)
print(dati)
print(a, b, c, dati, sep=':')
```

```
for v in dati:
    print(v)
```

```
for v in dati:
    print(v, end=' ')
```

```
1 a casa [1, -4, 2, 7, -10, -3]
1 a casa
[1, -4, 2, 7, -10, -3]
1:a:cas:[1, -4, 2, 7, -10, -3]
```

```
-4
2
7
-10
-3
```

```
1 -4 2 7 -10 -3 1
```

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

41

## Output formattato

```
print('{} {}'.format('primo', 'secondo'))
print('{0} {1}'.format('primo', 'secondo'))
print('{1} {0}'.format('primo', 'secondo'))
print('{2} {0}'.format('primo', 'secondo', 'terzo'))
```

```
primo secondo
primo secondo
secondo primo
terzo primo
```

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

42

## Output formattato

- Esempio di uso di format con parametri keywords

```
>>> d={"parola1": "ciao", "parola2": "?"}
>>> s="{parola1} Laura, come va {parola2}".format(**d)
>>> s
'ciao Laura, come va ?'
```

```
>>> s="{parola1} Laura, come va {parola2}".format(parola1="ciao", parola2="?")
>>> s
'ciao Laura, come va ?'
```

```
>>> s="{parola1} Laura, come va {parola2}".format(parola2="?", parola1="ciao")
>>> s
'ciao Laura, come va ?'
```

43

## Output formattato

- Consultare
  - <https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files>
- Oppure consultate il tutorial più immediato presso
  - <https://pyformat.info/>

44

## Input: funzione `input`

- Riceve input da tastiera
- Può mostrare un cursore opzionale specificato come stringa
- Quello che viene letto è considerato stringa
  - Potrebbe dover essere convertito al tipo richiesto
- L'input termina con la pressione di invio (`\n`) che non viene inserito nella stringa letta

45

## Esempi

```
a = input('Inserisci un valore: ')
print(a, type(a))
```



```
Inserisci un valore: e
e <class 'str'>
```

```
a = input('Inserisci un valore: ')
print(a, type(a))
```



```
Inserisci un valore: 12
12 <class 'str'>
```

```
a = int(input('Inserisci un valore: '))
print(a, type(a))
```



```
Inserisci un valore: 14
14 <class 'int'>
```

46

## Letture e scrittura di file

- La funzione built-in `open()` restituisce un file object che ci permette di agire sui file
- Comunemente `open()` è invocato con due argomenti:
  - `open(filename,mode)`
  - Esempio: `p=open("file.txt","w")`
- Il primo argomento `filename` è la stringa contenente il nome del file
- Il secondo argomento `mode` è una piccola stringa che indica in quale modalità deve essere aperto il file
  - `'r'` : modalità di sola lettura
  - `'w'` : modalità di sola scrittura; se il file non esiste lo crea; se il file già esiste il suo contenuto viene cancellato
  - `'a'` : modalità di append; se il file non esiste lo crea; se il file già esiste il suo contenuto viene non cancellato
  - `'r+'` : modalità di lettura e scrittura; il contenuto del file non viene cancellato
  - Se il secondo argomento non è specificato viene utilizzato il valore di default che è `'r'`

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

47

47

## Letture e scrittura di file

Esempio: file.txt inizialmente vuoto

```
>>> fp=open("file.txt",'r+')
>>> fp.write("cominciamo a scrivere nel file")
30
>>> fp.write("\nvado al prossimo rigo")
22
```

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

48

48



## Lettura e scrittura di file

- Possiamo usare `close()` per chiudere il file e liberare immediatamente qualsiasi risorsa di sistema usata per tenerlo aperto.
- Se il file non venisse chiuso esplicitamente, il garbage collector di Python ad un certo punto distruggerebbe il file object e chiuderebbe il file.
  - Ciò potrebbe avvenire però dopo molto tempo.
    - Dipende dall'implementazione di Python che stiamo utilizzando
- Dopo aver chiuso il file non è possibile accedere in lettura o scrittura al file

## Lettura e scrittura di file

Esempio (stesso file di prima)

```
>>> fp.close()
```

```
>>> fp.readline()
```

Traceback (most recent call last):

```
File "<stdin>", line 1, in <module>
```

```
ValueError: I/O operation on closed file.
```

## Funzioni sui file

Calling Syntax	Description
<code>fp.read()</code>	Return the (remaining) contents of a readable file as a string.
<code>fp.read(k)</code>	Return the next $k$ bytes of a readable file as a string.
<code>fp.readline()</code>	Return (remainder of) the current line of a readable file as a string.
<code>fp.readlines()</code>	Return all (remaining) lines of a readable file as a list of strings.
<code>for line in fp:</code>	Iterate all (remaining) lines of a readable file.
<code>fp.seek(k)</code>	Change the current position to be at the $k^{\text{th}}$ byte of the file.
<code>fp.tell()</code>	Return the current position, measured as byte-offset from the start.
<code>fp.write(string)</code>	Write given string at current position of the writable file.
<code>fp.writelines(seq)</code>	Write each of the strings of the given sequence at the current position of the writable file. This command does <i>not</i> insert any newlines, beyond those that are embedded in the strings.
<code>print(..., file=fp)</code>	Redirect output of print function to the file.

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

51

51

## Lettura e scrittura di file

```
Esempio:
>>> f=open("newfile",'w')
>>> f.write("prima linea\n")
12
>>> f.write("seconda linea\n")
14
>>> f.write("terza linea\n")
12
>>> f.write("quarta linea\n")
13
>>> f.close()
>>> f=open("newfile",'r')
>>> for line in f:
...     print(line)
...
prima linea

seconda linea

terza linea

quarta linea
```

Contenuto di newfile

```
prima linea
seconda linea
terza linea
quarta linea
```

Programmazione Avanzata a.a. 2020-21  
A. De Bonis

52

52

## Letture e scrittura di file

Esempio: continua dalla slide precedente

```
>>> f.seek(0)
0
>>> f.readline()
'prima linea\n'
>>> for linea in f:
...     print(linea)
...
seconda linea

terza linea

quarta linea
```

Contenuto di newfile

```
prima linea
seconda linea
terza linea
quarta linea
```

53

## Gestione dei file

- Maggiori dettagli in
  - <https://docs.python.org/3/library/filesys.html>

54