

Python

Esempi di codice

Materiale di supporto al volume Python: Programmazione, Data Science e Intelligenza artificiale

Carlo A. Mazzone

Prefazione al volume di Paolo Benanti

Questa raccolta riunisce i blocchi di codice, i comandi da terminale, i frammenti di markup e le sessioni interattive presenti nel volume. Gli esempi seguono lo stesso ordine del libro e sono organizzati per capitolo e sezione.

Non sono riportati i numeri di pagina del volume: in questo modo il documento resta utilizzabile anche in caso di future revisioni dell'impaginazione. Gli output isolati non sono inclusi; sono mantenuti soltanto quando fanno parte di una sessione interattiva utile a comprendere l'esempio.

Pagina di supporto: www.tesseract.it/python

Versione 1.0 - 2026

Indice dei capitoli con esempi

I capitoli che non contengono blocchi autonomi di codice non compaiono in questa raccolta.

Capitolo 1 - Premesse e motivazioni

Capitolo 2 - 3, 2, 1: Python!

Capitolo 3 - Python e i dati

Capitolo 4 - È tempo di scelte: il costrutto if

Capitolo 5 - Aggregati di dati e iterazioni

Capitolo 6 - Funzioni, sviluppo procedurale e moduli

Capitolo 7 - Classi, oggetti e atti creativi

Capitolo 8 - Files, dati e controllo degli errori

Capitolo 9 - NumPy e Data Science

Capitolo 10 - Pandas

Capitolo 12 - Apprendimento Supervisionato

Capitolo 13 - Matematica, statistica e regressione

Capitolo 15 - Dentro una rete neurale

Capitolo 16 - Le reti neurali in pratica con PyTorch e TensorFlow

Capitolo 17 - LLM: i Modelli Linguistici di Grandi Dimensioni

Capitolo 1 - Premesse e motivazioni

Python e la filosofia

CODICE

```
import this
```

Qualche cenno sulla teoria dei linguaggi di programmazione

PSEUDOCODICE

```
READ 9
```

Capitolo 2 - 3, 2, 1: Python!

Esempi

COMANDO

```
sudo apt install python3
```

COMANDO

```
python3 --version
```

COMANDO

```
whereis python
```

COMANDO

```
python3
```

SESSIONE INTERATTIVA

```
>>> print("Salve mondo da Python")  
Salve mondo da Python
```

CODICE

```
print("Salve mondo da Python")
```

COMANDO

```
python3 salve.py
```

COMANDO

```
./salve.py
```

CODICE

```
#!/usr/bin/python3  
print("Salve mondo da Python")
```

COMANDO

```
chmod u+x salve.py
```

COMANDO

```
cat miofile.gif|more
```

CODICE

```
#!/usr/bin/env python3
```

CODICE

```
#!/usr/bin/env bash
```

CODICE

```
#!/usr/bin/python3
```

CODICE

```
#!/usr/bin/env python3
```

Python Enhancement Proposals

CODICE

```
a = 1  
b = 2  
print(a)  
print(b)
```

CODICE

```
a = 1; b = 2; print(a); print(b)
```

Il linguaggio Markdown

CODICE

```
# Intestazione Livello 1
## Intestazione Livello 2
### Intestazione Livello 3
#### Intestazione Livello 4
##### Intestazione Livello 5
##### Intestazione Livello 6
```

Immagini

CODICE

```
![testo alternativo](URL dell'immagine)
```

CODICE

```
![Logo]( https://example.com/logo.png)
```

Elenchi

CODICE

```
Un elenco
* Romolo (753 AC - 716 AC)
* Numa Pompilio (715 AC - 674 AC)
* Tullo Ostilio (673 AC - 642 AC)
* Anco Marzio (642 AC - 617 AC)
* Tarquinio Prisco (616 AC - 579 AC)
* Servio Tullio (578 AC - 535 AC)
* Tarquinio il Superbo (535 AC - 510 AC)
```

CODICE

```
Un elenco numerato:
1. Romolo (753 AC - 716 AC)
2. Numa Pompilio (715 AC - 674 AC)
3. Tullo Ostilio (673 AC - 642 AC)
4. Anco Marzio (642 AC - 617 AC)
5. Tarquinio Prisco (616 AC - 579 AC)
6. Servio Tullio (578 AC - 535 AC)
7. Tarquinio il Superbo (535 AC - 510 AC)
```

CODICE

```
1. Primo elemento
1. Secondo elemento
1. Terzo elemento
```

CODICE

```
5. Primo
6. Secondo
7. Terzo
```

Tabelle

CODICE

```
| Intestazione 1 | Intestazione 2 |
| ----- | ----- |
| Cella 1 | Cella 2 |
| Cella 3 | Cella 4 |
```

Capitolo 3 - Python e i dati

Esempi

SESSIONE INTERATTIVA

```
>>> x="Salve, mondo"
>>> print(x)
Salve, mondo
```

CODICE

```
int x;
```

SESSIONE INTERATTIVA

```
>>> x = 10
>>> print(x)
10
>>> x = "Carlo"
>>> print(x)
Carlo
>>>
```

SESSIONE INTERATTIVA

```
>>> x = 'Carlo'
>>> print(x)
Carlo
>>>
```

SESSIONE INTERATTIVA

```
>>> 10 + "2"
```

Il casting dei tipi

SESSIONE INTERATTIVA

```
>>> int("66")
66
```

SESSIONE INTERATTIVA

```
>>> str(12)
'12'
```

CODICE

```
int ("66")
```

SESSIONE INTERATTIVA

```
>>> int("Carlo")
```

Gli assegnamenti e le stringhe immutabili

CODICE

```
x = 10
```

SESSIONE INTERATTIVA

```
>>> x = 10
>>> x = x + 2
>>> print(x)
12
```

CODICE

```
x = x + 2
```

CODICE

```
x <- x + 2
```

CODICE

```
x = 10  
x = 12
```

CODICE

```
x = 10;
```

CODICE

```
x = 12;
```

SESSIONE INTERATTIVA

```
>>> nome = "Carlo"  
>>> lettera = nome[0]  
>>> print(lettera)  
C
```

SESSIONE INTERATTIVA

```
>>> lettera = nome[1]  
>>> print(lettera)  
a
```

CODICE

```
nome[0] = "c"
```

SESSIONE INTERATTIVA

```
>>> nome[0] = "c"  
Traceback (most recent call last):  
  File "<pyshell#7>", line 1, in <module>  
    nome[0] = "c"  
TypeError: 'str' object does not support item assignment
```

Commentando Python

CODICE

```
# Questo è un commento singolo in Python  
  
# Calcolo della somma di due numeri  
numero1 = 10  
numero2 = 12  
somma = numero1 + numero2 # Questo è un commento a fine riga  
  
print("La somma è:", somma) # Questo stampa il risultato
```

Gli array di Python

CODICE

```
import array  
  
# Creazione di un array di interi  
arr = array.array('i', [1, 2, 3, 4, 5])  
print (arr[0])
```

CODICE

```
import array
```

CODICE

```
arr = array.array('i', [1, 2, 3, 4, 5])
```

CODICE

```
print(arr[0])
```

CODICE

```
arr[0]=66
```

Append

CODICE

```
arr.append(6)
```

CODICE

```
import array
arr = array.array('i', [1, 2, 3, 4, 5])
arr.append(6)
print(arr)
```

Extend

CODICE

```
# Creiamo un altro array da aggiungere all'array 'arr'
arr_to_extend = array.array('i', [7, 8, 9])

# Estendiamo 'arr' con 'arr_to_extend'
arr.extend(arr_to_extend)
```

Insert

CODICE

```
arr.insert(3, 10)
```

Remove

CODICE

```
arr.remove(5)
```

Pop

CODICE

```
elemento = arr.pop()
```

Ancora sulle stringhe

CODICE

```
len(nome)
```

CODICE

```
print(nome[4])
```

CODICE

```
print(nome[len(nome) - 1])
```

CODICE

```
print(nome[1:5])
```

SESSIONE INTERATTIVA

```
>>> Nome = 'c' + nome[1:5]
>>> print (Nome)
carlo
>>>
```

CODICE

```
Nome = 'c' + nome[1:]
```

CODICE

```
nome = 'Carlo'
nome = 'carlo'
```

Input di dati

CODICE

```
Variabile = input("Messaggio di testo")
```

CODICE

```
nome = input("Come ti chiami?")  
print("Il tuo nome è ", nome)
```

CODICE

```
nome = input("Come ti chiami? ")  
print("Il tuo nome è", nome, "La variabile è di tipo:", type(nome))
```

CODICE

```
print("Una calcolatrice davvero minimale")  
x = input("Inserisci il primo numero: ")  
y = input("Inserisci il secondo numero: ")  
s=x+y  
print("La somma dei due numeri è: ", s)
```

CODICE

```
s=x+y
```

CODICE

```
print("Una calcolatrice davvero minimale")  
x = int(input("Inserisci il primo numero: "))  
y = int(input("Inserisci il secondo numero: "))  
s=x+y  
print("La somma dei due numeri è: ", s)
```

CODICE

```
print("Una calcolatrice davvero minimale")  
x = input("Inserisci il primo numero: ")  
y = input("Inserisci il secondo numero: ")  
s = int(x) + int(y)  
print("La somma dei due numeri è: ", s)
```

CODICE

```
print("Una calcolatrice davvero minimale")  
x = input("Inserisci il primo numero: ")  
y = input("Inserisci il secondo numero: ")  
s = float(x) + float(y)  
print("La somma dei due numeri è: ", s)
```

Funzioni di conversione

CODICE

```
nominativo = "carlo MAZZONE"  
print(str.lower(nominativo))  
print(str.upper(nominativo))  
print(str.capitalize(nominativo))
```

CODICE

```
nominativo = "carlo MAZZONE"  
print(nominativo.lower())  
print(nominativo.upper())  
print(nominativo.capitalize())
```

Metodo find

CODICE

```
frase = "Salve mondo da Python!"  
posizione = frase.find("mondo")  
print("La parola 'mondo' si trova alla posizione:", posizione)
```

CODICE

```
posizione = frase.find("S")
```

CODICE

```
posizione = frase.find("mondo", 7)
```

Metodo replace

CODICE

```
frase = "Salve mondo da Python!"  
frase_modificata=frase.replace("Salve", "Ciao")  
print(frase_modificata)
```

CODICE

```
frase_modificata=frase.replace("Salve", "Ciao", 1)
```

Il metodo format

CODICE

```
nome = "Carlo"  
città = "Ceppaloni"  
  
messaggio = "Ciao, mi chiamo {0} e vivo a {1}.".format(nome, città)  
  
print(messaggio)
```

Formatted String Literals

CODICE

```
nome = "Carlo"  
città = "Ceppaloni"  
  
messaggio = f"Ciao, mi chiamo {nome} e vivo a {città}."  
  
print(messaggio)
```

CODICE

```
messaggio = F"Ciao, mi chiamo {nome} e vivo a {città}."
```

Il metodo di divisione split: spoiler di una lista

CODICE

```
frase = "Salve mondo da Python!"  
parole = frase.split()  
print(parole)
```

CODICE

```
frase = "Salve mondo, da Python!"  
parole = frase.split(',')  
print(parole)
```

Il metodo di unione join

CODICE

```
parole = ["Salve", "mondo", "da", "Python!"]  
frase = ' '.join(parole)  
print(frase)
```

CODICE

```
frase = '_'.join(parole)
```

I metodi strip, lstrip e rstrip: ripuliamo le stringhe

CODICE

```
frase = " Salve mondo, da Python! "
```

CODICE

```
# Rimuoviamo gli spazi all'inizio e alla fine della stringa
frase_strip = frase.strip()

# Rimuoviamo gli spazi all'inizio (lato sinistro) della stringa
frase_lstrip = frase.lstrip()

# Rimuoviamo gli spazi alla fine (lato destro) della stringa
frase_rstrip = frase.rstrip()

print("Originale:", "|" + frase + "|")
print("strip():", "|" + frase_strip + "|")
print("lstrip():", "|" + frase_lstrip + "|")
print("rstrip():", "|" + frase_rstrip + "|")
```

Capitolo 4 - È tempo di scelte: il costrutto if

Esempi

CODICE

```
a=5
b=3
if a>b:
    print("Il primo numero è maggiore del secondo.")
```

CODICE

```
print("Il costrutto if")
a=5
b=3
if a>b:
    print("Il primo numero è maggiore del secondo.")
    print("Io sono la seconda riga")
```

CODICE

```
print("Il costrutto if")
a=1
b=3
if a>b:
    print("Il primo numero è maggiore del secondo.")
print("Io sono la seconda riga")
```

CODICE

```
a=5
b=3
if a>b:
    print("a è maggiore di b.")
elif a<b:
    print("b è maggiore di a.")
```

CODICE

```
a=3
b=3
if a>b:
    print("a è maggiore di b.")
elif a<b:
    print("b è maggiore di a.")
else:
    print("a e b sono uguali.")
```

Short-circuit evaluation: il comportamento speciale degli operatori logici di Python

SESSIONE INTERATTIVA

```
>>> 0 and 5
```

SESSIONE INTERATTIVA

```
>>> 3 and 5
```

SESSIONE INTERATTIVA

```
>>> 3 or 5
```

SESSIONE INTERATTIVA

```
>>> 0 or 5
```

CODICE

```
nome = input("Nome: ") or "Anonimo"
```

AND binario

CODICE

```
a = 12 # binario: 1100
b = 10 # binario: 1010

# AND bit a bit
risultato = a & b

print("Decimale: " + str(risultato))
print("Binario: " + bin(risultato))
```

OR binario

CODICE

```
a = 12 # binario: 1100
b = 10 # binario: 1010

# OR bit a bit
risultato = a | b

print("Decimale: " + str(risultato))
print("Binario: " + bin(risultato))
```

NOT, la negazione logica

CODICE

```
a = 12 # binario: 1100

# NOT bit a bit
risultato = ~a

print("Decimale: " + str(risultato))
print("Binario: " + bin(risultato))
```

XOR: l'OR esclusivo

CODICE

```
a = 12 # binario: 1100
b = 10 # binario: 1010

# OR esclusivo bit a bit
risultato = a ^ b

print("Decimale: " + str(risultato))
print("Binario: " + bin(risultato))
```

Shift a sinistra

CODICE

```
numero = 4 # In binario: 100

# Shift a sinistra di 1 posizione
risultato = numero << 1
print("Binario: " + bin(risultato))
print("Decimale: " + str(risultato))
```

Shift a destra

CODICE

```
numero = 8 # In binario: 1000

# Shift a destra di 2 posizioni
risultato = numero >> 2 # Equivale a dividere numero per 2^2, ovvero per 4

print("Numero originale:", numero)
print("Dopo lo shift a destra:", risultato)
```

Funzioni predicato

CODICE

```
myvar=input("Inserisci un input e ne valuterò la tipologia: ")
if myvar.islower():
    print("La stringa è tutta minuscola.")
elif myvar.isupper():
    print("La stringa è tutta maiuscola.")
elif myvar.isspace():
    print("La stringa è composta solo da spazi.")
elif myvar.isalpha():
    print("Stringa generica di tipo alfabetico.")
elif myvar.isdigit():
    print("La stringa è tutta numerica.")
else:
    print("Stringa non classificata.")
```

Operatori di appartenenza

CODICE

```
import array

# Crea un array di interi (tipo 'i' per interi)
arr = array.array('i', [1, 2, 3, 4, 5])

# Verifica la presenza di un elemento
if 3 in arr:
    print("Il numero 3 è presente nell'array.")
if 6 not in arr:
    print("Il numero 6 non è presente nell'array.")
```

CODICE

```
frase = "Benvenuto nel mondo della programmazione in Python!"

# Verifica la presenza di una sottostringa
parola = "Python"
if parola in frase:
    print(f"La parola '{parola}' è presente nella frase.")
else:
    print(f"La parola '{parola}' non è presente nella frase.")

# Verifica l'assenza di una sottostringa
parola_assente = "Java"
if parola_assente not in frase:
    print(f"La parola '{parola_assente}' non è presente nella frase.")
else:
    print(f"La parola '{parola_assente}' è presente nella frase.")
```

Capitolo 5 - Aggregati di dati e iterazioni

Liste

CODICE

```
ingredienti=["latte","zucchero","farina","uova"]
```

CODICE

```
print(ingredienti[0])
```

CODICE

```
potenze=[1,2,4,8,16,32,64,128]
```

CODICE

```
personaggi=["Carlo",1966,"Mickey",1928,"Donald",1934]
```

CODICE

```
print(len(personaggi))
```

CODICE

```
print(type(personaggi))
```

CODICE

```
personaggi = list(("Carlo",1966,"Mickey",1928,"Donald",1934))
```

CODICE

```
print(personaggi[-1])
```

CODICE

```
print(personaggi[-2])
```

CODICE

```
print(personaggi[1:3])
```

CODICE

```
print(personaggi[:])
```

CODICE

```
if 1966 in personaggi:  
    print("Elemento presente!")
```

CODICE

```
if "Carlo" in personaggi:  
    print("Elemento presente!")
```

Le liste sono modificabili

CODICE

```
personaggi = list(("Carlo",1966,"Mickey",1928,"Donald",1934))  
personaggi[0]="Carlo Mazzone"  
print(personaggi)
```

CODICE

```
personaggi = list(("Carlo",1966,"Mickey",1928,"Donald",1934))  
personaggi[0:2]=["Goofy",1932]  
print(personaggi)
```

CODICE

```
personaggi = list(("Carlo","Mickey","Donald"))  
personaggi.insert(1,"Goofy")  
print(personaggi)
```

CODICE

```
personaggi = list(("Carlo", "Mickey", "Donald"))
personaggi.append("Goofy")
print(personaggi)
```

CODICE

```
ingredienti = list(("farina", "latte", "zucchero"))
ingredienti_2 = list(("uova", "burro", "limone"))
ingredienti.extend(ingredienti_2)
print(ingredienti)
```

Rimozione di elementi da una lista

CODICE

```
ingredienti = list(("farina", "latte", "zucchero", "uova", "burro", "limone"))
ingredienti.remove("burro")
print(ingredienti)
```

CODICE

```
ingredienti = list(("farina", "latte", "zucchero", "uova", "burro", "limone", "burro"))
print(ingredienti)
ingredienti.remove("burro")
print(ingredienti)
ingredienti.remove("burro")
print(ingredienti)
```

CODICE

```
ingredienti = list(("farina", "latte", "zucchero", "uova", "burro", "limone"))
ingredienti.pop(4)
print(ingredienti)
```

CODICE

```
ingredienti = list(("farina", "latte", "zucchero", "uova", "burro", "limone"))
ingredienti.pop()
print(ingredienti)
```

CODICE

```
x = ingredienti.pop()
```

CODICE

```
ingredienti = list(("farina", "latte", "zucchero", "uova", "burro", "limone"))
del ingredienti[4]
print(ingredienti)
```

CODICE

```
ingredienti = list(("farina", "latte", "zucchero", "uova", "burro", "limone"))
del ingredienti
```

CODICE

```
print(ingredienti)
```

CODICE

```
ingredienti = list(("farina", "latte", "zucchero", "uova", "burro", "limone"))
ingredienti.clear()
```

CODICE

```
print(ingredienti)
```

Il ciclo for

CODICE

```
ingredienti = ["farina", "latte", "zucchero", "uova", "burro", "limone"]
for x in ingredienti:
    print(x)
```

CODICE

```
ingredienti = ["farina", "latte", "zucchero", "uova", "burro", "limone"]
i=0
for x in ingredienti:
    i=i+1
    print("Ingrediente numero ", i, " - ", x)
```

CODICE

```
ingredienti = ["farina", "latte", "zucchero", "uova", "burro", "limone"]
i=0
for x in ingredienti:
    i=i+1
    print("Ingrediente numero ", i, " - ", x)
    if x == "burro":
        break
```

CODICE

```
ingredienti = ["farina", "latte", "zucchero", "uova", "burro", "limone"]
ricetta_OK=True
for x in ingredienti:
    if x == "burro":
        ricetta_OK=False
        break
if ricetta_OK == True:
    print("Ingredienti corretti!")
else:
    print("Ingredienti errati, il burro non è previsto nella ricetta!")
```

CODICE

```
ingredienti = ["farina", "latte", "zucchero", "uova", "cacao", "limone"]
```

Stop and go con continue

CODICE

```
ingredienti = ["farina", "latte", "zucchero", "uova", "burro", "limone"]
for x in ingredienti:
    if x == "burro":
        continue
    print(x)
```

CODICE

```
ingredienti = ["farina", "latte", "zucchero", "uova", "burro", "limone"]
for x in ingredienti:
    if x == "burro":
        continue
    print(x)
```

Un for più tradizionale con range

CODICE

```
for ($i = 1; $i <= 10; $i++)
{
    echo "<br>" . $i ;
}
```

CODICE

```
for i in range(1, 11):
    print(i)
```

CODICE

```
for i in range(6):
    print(i)
```

CODICE

```
for i in range(2,11,2):
    print(i)
```

CODICE

```
for i in range(2,11,2):
    print(i)
else:
    print("Ciclo terminato!")
```

CODICE

```
for i in range(2,11,2):
    if i==8:
        break
    print(i)
else:
    print("Ciclo terminato!")
```

CODICE

```
for i in range(1,11):
```

CODICE

```
for i in range(1,11):
    pass
```

Il ciclo while

CODICE

```
i=1
while i < 11:
    if i == 6:
        break
    print(i)
    i=i+1
```

CODICE

```
i=1
while i < 11:
    print(i)
    i=i+1
else:
    print("i vale: ", i)
```

CODICE

```
i=1
while i < 11:
    if i == 6:
        break
    print(i)
    i=i+1
else:
    print("i vale: ", i)
```

Tuple

CODICE

```
ingredienti = ("farina","latte","zucchero","uova","burro","limone")
```

CODICE

```
print(type(ingredienti))
```

CODICE

```
ingredienti = ("farina",)
```

CODICE

```
print(type(ingredienti))
```

CODICE

```
ingredienti = ("farina")
```

CODICE

```
print(type(ingredienti))
```

CODICE

```
ingredienti = tuple(("farina", "latte", "zucchero", "uova", "burro", "limone"))
```

CODICE

```
ingredienti = tuple(("farina", "latte", "zucchero", "uova", "burro", "limone", "burro"))
```

CODICE

```
print(len(ingredienti))
```

CODICE

```
ingredienti = tuple(("farina", "latte", "zucchero", "uova", "burro", "limone"))  
primo_ingrediente = ingredienti[0]  
print("Primo ingrediente:", primo_ingrediente)
```

CODICE

```
ingredienti[4]="olio"
```

Ricerca di elementi

CODICE

```
ingredienti = tuple(("farina", "latte", "zucchero", "uova", "burro", "limone", "burro"))  
print(ingredienti.count("burro"))
```

CODICE

```
print(ingredienti.index("burro"))
```

CODICE

```
tuple.index(x[, inizio[, fine]])
```

CODICE

```
ingredienti = tuple(("farina", "latte", "zucchero", "uova", "burro", "limone", "burro"))  
print(ingredienti.index("burro", 5))
```

Concatenazione

CODICE

```
ingredienti_1 = tuple(("farina", "latte", "zucchero"))  
ingredienti_2 = tuple(("uova", "burro", "limone"))  
ingredienti_ricetta = ingredienti_1 + ingredienti_2  
print("Ingredienti ricetta:", ingredienti_ricetta)
```

CODICE

```
tupla1 = (1, 2, 3)  
tupla2 = (4, 5, 6)  
tupla_concatenata = tupla1 + tupla2  
print("Tupla concatenata:", tupla_concatenata)
```

Join di tuple in salsa di codice Pythonico

CODICE

```
elementi = ("Questo", "metodo", "è", "più", "efficiente", "e", "pythonico!")  
frase = " ".join(elementi)  
print(frase)
```

CODICE

```
print(type(frase))
```

Ripetizione

CODICE

```
tupla = (1, 2, 3)
tupla_risultato = 5 * tupla
print("Tupla risultato:", tupla_risultato)
```

Slicing

CODICE

```
ingredienti = tuple(("farina", "latte", "zucchero", "uova", "burro", "limone"))
```

CODICE

```
primi_tre = ingredienti[:3]
ultimi_tre = ingredienti[-3:]
senza_primo_ultimo = ingredienti[1:-1]
ogni_due = ingredienti[::2]
invertita = ingredienti[::-1]
```

CODICE

```
print(primi_tre)
print(ultimi_tre)
print(senza_primo_ultimo)
print(ogni_due)
print(invertita)
```

Unpacking: spacchettiamo gli elementi di una tupla

CODICE

```
tupla = (1, 2, 3, 4, 5)
```

CODICE

```
print(a)
print(c)
print(resto)
```

CODICE

```
ingredienti = tuple(("farina", "latte", "zucchero", "uova", "burro", "limone"))
base, liquido, dolcificante, *aromi = ingredienti
print(base)
print(liquido)
print(dolcificante)
print(aromi)
```

Iterazione e ricerca di elementi in una tupla

CODICE

```
ingredienti = tuple(("farina", "latte", "zucchero", "uova", "burro", "limone"))

# Iterazione tupla
print("Elenco ingredienti:")
for ingrediente in ingredienti:
    print(ingrediente)

# Verifica dell'appartenenza di un elemento specifico alla tupla
elemento_da_verificare = "burro"
if elemento_da_verificare in ingredienti:
    print(f"\nL'elemento '{elemento_da_verificare}' è presente nella tupla.")
else:
    print(f"\nL'elemento '{elemento_da_verificare}' NON è presente nella tupla.")
```

Insiemi - sets

CODICE

```
ingredienti = {"farina", "latte", "zucchero", "uova", "burro", "limone"}
```

CODICE

```
print(type(ingredienti))
```

CODICE

```
ingredienti = set(("farina", "latte", "zucchero", "uova", "burro", "limone"))
```

CODICE

```
ingredienti = {"farina", "latte", "zucchero", "uova", "burro", "limone", "burro"}
```

CODICE

```
print(ingredienti)
```

CODICE

```
ingredienti = {"farina", "latte", "zucchero", "uova", "burro", "limone", "burro"}  
print(len(ingredienti))
```

Aggiunta di elementi

CODICE

```
ingredienti = {"farina", "latte", "zucchero", "uova", "burro", "limone"}  
print(ingredienti)  
ingredienti.add("vaniglia")  
print(ingredienti)
```

CODICE

```
print("vaniglia" in ingredienti)
```

Operazioni insiemistiche sugli insiemi

CODICE

```
A = {1, 2, 3, 4}  
B = {3, 4, 5, 6}
```

CODICE

```
C = A.union(B) # Equivalente a C = A | B  
print(C)
```

CODICE

```
C = A.intersection(B) # Equivalente a C = A & B  
print(C)
```

CODICE

```
C = A.difference(B) # Equivalente a C = A - B  
print(C)
```

CODICE

```
C = A.symmetric_difference(B) # Equivalente a C = A ^ B  
print(C)
```

CODICE

```
A.update(B) # Equivalente a A |= B  
print(A)
```

Dizionari

CODICE

```
utente = {  
    "nome": "Carlo",  
    "cognome": "Mazzone",  
    "nascita": 1966  
}
```

CODICE

```
print(type(utente))
```

CODICE

```
print(utente["cognome"])
```

CODICE

```
utente = {  
    "nome": "Carlo",  
    "cognome": "Mazzone",  
    "nascita": 1966  
}  
  
print(utente["nome"])  
utente["nome"]="Nightfile"  
print(utente["nome"])
```

CODICE

```
utente = {  
    "nome": ["Carlo", "Andrew Stuart", "Linus"],  
    "cognome": ["Mazzone", "Tanenbaum", "Torvalds"],  
    "nascita": [1966, 1944, 1960]  
}
```

In somma sintesi: differenze tra liste, tuple, insiemi e dizionari

CODICE

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

CODICE

```
tupla = (1, 2, 3)
```

CODICE

```
insieme = {1, 2, 3}
```

CODICE

```
dizionario = {'chiave1': 'valore1', 'chiave2': 'valore2'}
```

Capitolo 6 - Funzioni, sviluppo procedurale e moduli

Esempi

CODICE

```
def mia_funzione():  
    print("Io sono una funzione.")  
  
mia_funzione()
```

CODICE

```
# Attenzione: codice non funzionante  
mia_funzione()  
def mia_funzione():  
    print("Io sono una funzione.")
```

Input per le funzioni

CODICE

```
def salve(x):  
    print("Salve mondo da:", x)  
  
salve("Carlo")
```

CODICE

```
def salve(x):  
    print("Salve mondo da:", x)  
  
salve()
```

CODICE

```
def salve(nome, cognome):  
    print("Salve mondo da:", nome, cognome)  
  
salve("Carlo", "Mazzone")
```

CODICE

```
def salve(nome, cognome):  
    print("Salve mondo da:", nome, cognome)  
  
salve("Carlo")
```

CODICE

```
def salve(*nome):  
    print("Salve mondo da:", nome)  
  
salve("Carlo", "Mazzone")
```

CODICE

```
def salve(*nome):  
    print("Salve mondo da:", nome[0])  
  
salve("Carlo", "Mazzone")
```

CODICE

```
def salve(nome, cognome):  
    print("Salve mondo da:", nome, cognome)  
  
salve(cognome="Mazzone", nome="Carlo")
```

CODICE

```
def salve(**nominativo):
    print("Salve mondo da:", nominativo["cognome"])

salve(cognome="Mazzone", nome="Carlo")
```

Valori predefiniti

CODICE

```
def salve(nominativo="Ulisse"):
    print("Salve mondo da:", nominativo)

salve("Carlo")
salve("Mazzone")
salve()
```

CODICE

```
def ricetta(ingredienti):
    for x in ingredienti:
        print(x)

ingredienti = ["farina", "latte", "zucchero", "uova", "burro", "limone"]
ricetta(ingredienti)
```

Una funzione non è una procedura: valori di ritorno

CODICE

```
def media(a, b):
    m = (a + b) / 2
    return m

print(media(8, 3))
print(media(8, 4))
```

La visibilità delle variabili

CODICE

```
def media(a, b):
    m = (a + b) / 2
    return m

m = 5
print(media(8, 3))
print(media(8, 4))
print("m vale:", m)
```

CODICE

```
def media(a, b):
    m = (a + b) / 2
    return m

print("m vale:", m)
```

Puntatori? No, grazie.

CODICE

```
def ricetta(ingredienti):
    ingredienti.remove("burro")

ingredienti2 = ["farina", "latte", "zucchero", "uova", "burro", "limone"]
ricetta(ingredienti2)
print (ingredienti2)
```

CODICE

```
def ricetta(ingredienti):
    ingredienti = ingredienti + " latte"
    return ingredienti

ingredienti = "farina"
ricetta_2 = ricetta(ingredienti)

print("Stringa originale:", ingredienti)
print("Nuova ricetta:", ricetta_2)
```

Moduli

CODICE

```
#Modulo geometria
def rettangolo_area(a, b):
    return a * b
```

COMANDO

```
%%writefile mio_modulo.py
def mia_funzione():
    print("Questo è un modulo!")
```

CODICE

```
import geometria
print("Area del rettangolo: ", geometria.rettangolo_area(4, 3))
```

CODICE

```
import geometria
```

CODICE

```
geometria.rettangolo_area(4, 3)
```

CODICE

```
import geometria as geo
print("Area del rettangolo: ", geo.rettangolo_area(4, 3))
```

CODICE

```
import platform
print("Sistema: ", platform.system())
print("Piattaforma:", platform.platform())
print("Processore: ", platform.processor())
print('Compilatore Python:', platform.python_compiler())
print('Versione Python:', platform.python_version())
```

CODICE

```
import platform
info = dir(platform)
print(info)
```

CODICE

```
#Modulo geometria
def rettangolo_area(a,b):
    return a*b

def parallelepipedo_volume(a,b,h):
    return a*b*h

rettangolo1 = {
    "base": 12,
    "altezza": 6
}

rettangolo2 = {
    "base": 20,
    "altezza": 10
}
```

CODICE

```
from geometria import rettangolo1
print(rettangolo1["base"])
```

CODICE

```
from geometria import rettangolo1
```

CODICE

```
from geometria import rettangolo_area, parallelepipedo_volume, rettangolo1

area = rettangolo_area(rettangolo1["base"], rettangolo1["altezza"])
volume = parallelepipedo_volume(12, 6, 10)
print("Area: ", area, "Volume: ", volume)
```

Pacchetti e moduli

COMANDO

```
pip --version
```

COMANDO

```
python -m pip --version
```

CODICE

```
pip3 --version
```

COMANDO

```
python3 -m pip --version
```

COMANDO

```
sudo apt install python3-pip
```

COMANDO

```
pip list
```

COMANDO

```
python -m pip list
```

COMANDO

```
pip install nome_pacchetto
```

COMANDO

```
python -m pip install nome_pacchetto
```

COMANDO

```
pip install --upgrade nome_pacchetto
```

COMANDO

```
pip install -U nome_pacchetto
```

COMANDO

```
python -m pip install --upgrade nome_pacchetto
```

COMANDO

```
pip uninstall nome_pacchetto
```

COMANDO

```
pip show nome_pacchetto
```

COMANDO

```
pip -h
```

COMANDO

```
python -m pip -h
```

Capitolo 7 - Classi, oggetti e atti creativi

Oggetti e classi

CODICE

```
class Punto:
    x=0
    y=0
    color=0
```

CODICE

```
p1 = Punto()
```

CODICE

```
print("Ascissa: ", p1.x)
print("Ordinata: ", p1.y)
print("Colore: ", p1.color)
```

Il costruttore, ovvero come ti creo gli oggetti

CODICE

```
class Punto:
    def __init__(self, x, y, color):
        self.x = x
        self.y = y
        self.color = color
```

CODICE

```
p1 = Punto(0, 0, 0)
```

CODICE

```
print("Ascissa: ", p1.x)
print("Ordinata: ", p1.y)
print("Colore: ", p1.color)
```

CODICE

```
p2 = Punto(1, 1, 255)
```

CODICE

```
print("Ascissa: ", p2.x)
print("Ordinata: ", p2.y)
print("Colore: ", p2.color)
```

CODICE

```
p1.x = 1
```

CODICE

```
class Punto:
    def __init__(self, x, y, color):
        self.x = x
        self.y = y
        self.color = color

p1 = Punto(0, 0, 0)
print("Ascissa: ", p1.x)
print("Ordinata: ", p1.y)
print("Colore: ", p1.color)

p1.x=1
p1.y=1
p1.color=128

print("Nuova ascissa: ", p1.x)
print("Nuova ordinata: ", p1.y)
print("Nuovo colore: ", p1.color)
```

CODICE

```
class Punto:  
    pass
```

CODICE

```
class Punto:  
    x = 0  
    y = 0  
  
p = Punto()  
print(p.x, p.y)  
  
del p  
print(p.x, p.y)
```

Ereditarietà e programmazione

CODICE

```
class Punto:  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y  
  
    def print_coord(self):  
        print(self.x, self.y)
```

CODICE

```
class Cerchio(Punto):  
    pass
```

CODICE

```
c1 = Cerchio(2,2)
```

CODICE

```
class Punto:  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y  
  
    def print_coord(self):  
        print(self.x, self.y)  
  
class Cerchio(Punto):  
    pass  
  
p1 = Punto(1,1)  
c1 = Cerchio(2,2)  
  
p1.print_coord()  
c1.print_coord()
```

CODICE

```
super().__init__(x, y)
```

CODICE

```
class Punto:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def print_coord(self):
        print(self.x, self.y)

class Cerchio(Punto):
    def __init__(self, x, y, r):
        super().__init__(x, y)
        self.r = r

    def area(self):
        print(self.r * self.r * 3.14)

p1 = Punto(0, 0)
c1 = Cerchio(1, 1, 2)

p1.print_coord()
c1.print_coord()
c1.area()
```

Dal polimorfismo agli argomenti opzionali

CODICE

```
p1 = Punto()
```

CODICE

```
class Punto:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

class Cerchio(Punto):
    def __init__(self, x=0, y=0, r=1): # r ha un valore predefinito uguale a 1
        super().__init__(x, y)
        self.r = r
```

CODICE

```
p1 = Punto()
c1 = Cerchio()
```

CODICE

```
p1 = Punto(1, 1)
c1 = Cerchio(1, 1, 2)
```

Capitolo 8 - Files, dati e controllo degli errori

Esempi

CODICE

```
f = open("carlo.txt")
```

CODICE

```
try:
    f = open("carlo.txt")
except:
    print("Attenzione: si è verificata un'eccezione!")
```

CODICE

```
try:
    f = open("carlo.txt")
except FileNotFoundError:
    print("Attenzione: impossibile individuare il file richiesto!")
```

Modalità di apertura dei file

CODICE

```
f = open("carlo.txt")
```

CODICE

```
f = open("carlo.txt", "r")
```

CODICE

```
f = open("carlo.txt", "r")
print(f.read())
```

CODICE

```
f = open("carlo.txt", "r", encoding="utf-8")
```

CODICE

```
print(f.read(7))
```

CODICE

```
print(f.readline())
```

CODICE

```
print(f.readline())
print(f.readline())
```

CODICE

```
print(f.readline().strip())
```

CODICE

```
for line in f:
    print(line)
```

CODICE

```
for line in f:
    print(line.strip())
```

Ascolta Totò e chiudi il file

CODICE

```
f = open("carlo.txt", "r", encoding="utf-8")
for line in f:
    print(line.strip())
f.close()
```

CODICE

```
try:
    with open("carlo.txt", "r", encoding="utf-8") as f:
        for line in f:
            print(line.strip())
except FileNotFoundError:
    print("Attenzione: si è verificata un'eccezione!")
```

Scriviamo nei file

CODICE

```
f = open("carlo.txt", "w", encoding="utf-8")
f.write("Un nuovo file.\n")
f.write("Seconda riga.")
f.close()
```

CODICE

```
f = open("carlo.txt", "x", encoding="utf-8")
```

CODICE

```
filename = "carlo.txt"

f = open(filename, "w", encoding="utf-8")
f.write("Un nuovo file.")
f.close()

f = open(filename, "r", encoding="utf-8")
print(f.read())
f.close()
```

CODICE

```
f = open(filename, "w+")
```

CODICE

```
f = open(filename, "r+")
```

CODICE

```
filename = "carlo.txt"

f = open(filename, "w+", encoding="utf-8")
f.write("Un nuovo file.")

f.seek(0)
print(f.read())

f.close()
```

CODICE

```
filename = "carlo.txt"

with open(filename, "w+", encoding="utf-8") as f:
    f.write("Un nuovo file.")
    f.seek(0)
    print(f.read())
```

Spostarsi all'interno dei file

CODICE

```
f.seek(offset, whence)
```

Testo o binario: questo è il problema

CODICE

```
binario = b"\x00\x41\xff"

f = open("carlo.bin", "wb")
f.write(binario)
f.close()

f = open("carlo.bin", "rb")
print("Valori esadecimali:")
print(f.read(1))
print(f.read(1))
print(f.read(1))

print("Valori decimali:")
f.seek(0)

decimale = int.from_bytes(f.read(1), byteorder='big')
print(decimale)
decimale = int.from_bytes(f.read(1), byteorder='big')
print(decimale)
decimale = int.from_bytes(f.read(1), byteorder='big')
print(decimale)

f.close()
```

CODICE

```
binario = b"\x00\x41\xff"
```

CODICE

```
f = open("carlo.bin", "wb")
f.write(binario)
f.close()
```

CODICE

```
f = open("carlo.bin", "rb")
print("Valori esadecimali:")
print(f.read(1))
print(f.read(1))
print(f.read(1))
```

CODICE

```
decimale = int.from_bytes(f.read(1), byteorder='big')
print(decimale)
```

CODICE

```
int.from_bytes(bytes, byteorder, *, signed=False)
```

Capitolo 9 - NumPy e Data Science

L'installazione e gli strumenti necessari

COMANDO

```
pip install numpy
```

COMANDO

```
python -m pip install -U numpy
```

COMANDO

```
python -m pip install -U pip
```

COMANDO

```
conda install numpy
```

Come sopravvivere ai conflitti creando un ambiente virtuale Python

COMANDO

```
python -m venv nome_ambiente
```

COMANDO

```
python -m venv numpy_env
```

pip freeze

COMANDO

```
python -m pip freeze
```

COMANDO

```
python -m pip freeze > requirements.txt
```

COMANDO

```
python -m pip install -r requirements.txt
```

Un array NumPy non è una lista

CODICE

```
import numpy
arr = numpy.array([1, 2, 4, 8, 16, 32, 64, 128])
print(arr)
```

CODICE

```
import numpy as np
```

CODICE

```
arr = np.array([1, 2, 4, 8, 16, 32, 64, 128])
```

CODICE

```
arr = [1, 2, 4, 8, 16, 32, 64, 128]
print(arr)
```

CODICE

```
import numpy as np
arr = np.array([1, 2, 4, 8, 16, 32, 64, 128])
arr = arr * 2
print(arr)
```

CODICE

```
lista = [1, 2, 4, 8, 16, 32, 64, 128]
lista = lista * 2
print(lista)
```

CODICE

```
import numpy as np
a = np.array([1, 2, 4, 8, 16, 32, 64, 128])
b = np.array([1, 2, 4, 8, 16, 32, 64, 128])
c = a + b
print(c)
```

CODICE

```
import numpy as np
a = np.array([128, 64, 32, 16, 8, 4, 2, 1])
b = np.array([1, 1, 1, 1, 1, 1, 1, 1])
c = a * b
print(c)
print (sum(c))
```

Accesso e modifica

CODICE

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])
arr[0]=5
print(arr[0])
```

CODICE

```
import numpy as np

arr = np.zeros((4, 4), dtype="int32")
print(arr[0, 0])

arr[0, 0] = 5
print(arr[0, 0])
```

Un array 2D

CODICE

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr)
print(arr.shape)
```

CODICE

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
```

CODICE

```
print(arr.ndim)
```

CODICE

```
import numpy as np
arr = np.array([ [1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr.dtype)
```

CODICE

```
import numpy as np
arr = np.array([ [1.0, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr.dtype)
```

CODICE

```
arr = np.array([ [1., 2, 3, 4, 5], [6, 7, 8, 9, 10]])
```

CODICE

```
import numpy as np
arr = np.array([ [1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print("Tipo di dati dell'array:", arr.dtype)
print(arr)

# Sovrascriviamo l'array originale con il nuovo tipo di dati
arr = arr.astype(np.float64)

print("Array con tipo di dati float64:")
print(arr)
print("Tipo di dati dell'array:", arr.dtype)
```

Inizializzazione “binaria”

CODICE

```
import numpy as np
arr = np.zeros(10)
print(arr)
```

CODICE

```
import numpy as np
arr = np.zeros(10, dtype='int32')
print(arr)
```

CODICE

```
import numpy as np
arr = np.zeros((4, 8))
print(arr)
```

CODICE

```
arr = np.zeros((4, 8), dtype="int64")
```

CODICE

```
arr = np.zeros((4, 8), dtype=int)
```

CODICE

```
arr = np.zeros((4, 8), dtype=np.int32)
```

CODICE

```
import numpy as np
arr = np.ones((4, 8), dtype=np.int32)
print(arr)
```

Una terza dimensione

CODICE

```
import numpy as np

array_3d = np.array([
    [[ 1,  2,  3], [ 4,  5,  6], [ 7,  8,  9]],
    [[10, 11, 12], [13, 14, 15], [16, 17, 18]],
    [[19, 20, 21], [22, 23, 24], [25, 26, 27]]
])

print(array_3d)
```

Facciamo slicing negli array

CODICE

```
import numpy as np

arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

# Estraiamo gli elementi dall'indice 2 all'indice 5 (escluso)
subset = arr[2:5]
print(subset) # Output: [2 3 4]

# Estraiamo gli elementi dall'indice 1 al termine dell'array
subset2 = arr[1:]
print(subset2) # Output: [1 2 3 4 5 6 7 8 9]

# Estraiamo tutti gli elementi eccetto gli ultimi tre
subset3 = arr[:-3]
print(subset3) # Output: [0 1 2 3 4 5 6]

# Estraiamo gli elementi a partire da quello con indice 1 fino ad indice 9 (escluso) con incremento di 2
subset4 = arr[1:9:2]
print(subset4) # Output: [1 3 5 7]
```

Slicing 2D

CODICE

```
import numpy as np
arr_2d = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
```

CODICE

```
subset_col = arr_2d[:, 0]
print(subset_col)
```

CODICE

```
subset_row = arr_2d[1, :]
print(subset_row)
```

Slicing 3D

CODICE

```
import numpy as np
array_3d = np.array([
    [1, 2, 3], [4, 5, 6], [7, 8, 9]],
    [[10, 11, 12], [13, 14, 15], [16, 17, 18]],
    [[19, 20, 21], [22, 23, 24], [25, 26, 27]]
])

subset_3d = array_3d[1, :, :]
print(subset_3d)
```

CODICE

```
subset_3d = array_3d[1, :, 1:2]
```

View e copy

CODICE

```
import numpy as np

# Creiamo un array NumPy
array_originale = np.array([1, 2, 3, 4, 5])

print("Array originale:", array_originale)

# Creiamo una view dell'array originale
view_array = array_originale.view()

print("View array:", view_array)

# Modifichiamo la view
view_array[2] = 10 # Questo modifica anche l'array originale

# Stampa finale
print("View array dopo modifica:", view_array)
print("Array originale dopo modifica:", array_originale)
```

CODICE

```
import numpy as np

# Creiamo un array NumPy
array_originale = np.array([1, 2, 3, 4, 5])

print("Array originale:", array_originale)

# Creiamo una copia dell'array originale
copy_array = array_originale.copy()

print("Copy array:", copy_array)

# Modifichiamo la copia
copy_array[2] = 10 # Questo NON modifica l'array originale

# Stampa finale
print("Copy array dopo modifica:", copy_array)
print("Array originale dopo modifica:", array_originale)
```

Reshape

CODICE

```
np.arange(start, stop, step)
```

CODICE

```
import numpy as np

# Creiamo un array NumPy con i numeri da 1 a 27
array_numeri = np.arange(1, 28)
print(array_numeri.shape)
```

CODICE

```
print(array_numeri)
```

CODICE

```
# Reshape dell'array in una forma 3x3x3
array_reshaped = array_numeri.reshape((3, 3, 3))

print(array_reshaped)
```

Iterazione su un array 1D

CODICE

```
import numpy as np

arr_1d = np.array([1, 2, 3, 4, 5])
for element in arr_1d:
    print(element)
```

Iterazione su un array 2D

CODICE

```
import numpy as np
arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
for row in arr_2d:
    print(row)
```

CODICE

```
import numpy as np
arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

for row in arr_2d:
    for element in row:
        print(element)
```

Iterazione con np.nditer

CODICE

```
import numpy as np

arr_2d = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])

for element in np.nditer(arr_2d):
    print(element)
```

CODICE

```
import numpy as np

array_3d = np.arange(1, 28).reshape((3, 3, 3))

for element in np.nditer(array_3d):
    print(element)
```

CODICE

```
import numpy as np

array_3d = np.arange(1, 28).reshape((3, 3, 3))

for idx, element in np.ndenumerate(array_3d):
    print(idx, element)
```

Iterazione contemporanea su differenti array

CODICE

```
import numpy as np

arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([6, 7, 8, 9, 10])

for a, b in np.nditer([arr1, arr2]):
    print(a, b)
```

Concatenazione e stacking

CODICE

```
import numpy as np

array1 = np.array([1, 2, 3, 4, 5])
array2 = np.array([6, 7, 8, 9, 10])

# Uniamo gli array lungo l'unico asse disponibile, l'asse 0
array_unito = np.concatenate((array1, array2))
print(array_unito)
```

CODICE

```
import numpy as np

array1 = np.array([1, 2, 3, 4, 5])
array2 = np.array([6, 7, 8, 9, 10])

array_append = np.append(array1, array2)
print(array_append)
```

CODICE

```
import numpy as np

array1 = np.array([1, 2, 3, 4, 5])
array2 = np.array([6, 7, 8, 9, 10])

# Stack verticale
vstack_unito = np.vstack((array1, array2))
print("Verticale:\n", vstack_unito)

# Stack orizzontale
hstack_unito = np.hstack((array1, array2))
print("Orizzontale:\n", hstack_unito)
```

Splitting di array

CODICE

```
import numpy as np

array = np.arange(1, 13) # Creazione array con valori da 1 a 12
print("Array originale:", array)

# Dividiamo l'array in 3 parti uguali
split_arrays = np.split(array, 3)
print("Array divisi:")

print(type(split_arrays))

print("Parte 1: ", split_arrays[0])
print("Parte 2: ", split_arrays[1])
print("Parte 3: ", split_arrays[2])
```

CODICE

```
for i, arr in enumerate(split_arrays):
    print(f"Parte {i + 1}: {arr}")
```

CODICE

```
array_2D = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print("Array 2D originale:\n", array_2D)

# Dividiamo l'array 2D in 2 parti lungo l'asse delle colonne (asse 1)
split_arrays_2D_col = np.split(array_2D, 2, axis=1)
print("\nArray diviso lungo l'asse delle colonne:")
for i, arr in enumerate(split_arrays_2D_col):
    print(f"Parte {i + 1}:\n{arr}")
```

CODICE

```
split_arrays_2D = np.split(array_2D, 2, axis=0)
```

CODICE

```
import numpy as np

array = np.arange(1, 11) # Creazione array con valori da 1 a 10

# Usiamo array_split per dividere l'array in 3 parti
split_arrays = np.array_split(array, 3)

# Stampiamo i risultati
print("Array divisi:")
for i, arr in enumerate(split_arrays):
    print(f"Parte {i + 1}: {arr}")
```

CODICE

```
import numpy as np

array_2D = np.arange(1, 21).reshape(4, 5) # Crea un array 2D 4x5
print(array_2D)

# Usiamo array_split per dividere l'array lungo l'asse delle colonne (asse 1) in 2 parti
split_arrays_2D = np.array_split(array_2D, 2, axis=1)

# Stampiamo i risultati
print("Array 2D diviso lungo l'asse delle colonne:")
for i, arr in enumerate(split_arrays_2D):
    print(f"Parte {i + 1}:\n{arr}\n")
```

CODICE

```
split_arrays_2D = np.array_split(array_2D, 2, axis=1)
```

where

CODICE

```
import numpy as np
arr = np.array([1, 2, 2, 3, 5, 9, 6, 2])
x = np.where(arr == 2)
print(x)
```

CODICE

```
x = np.where(arr % 2 == 0)
```

CODICE

```
x = np.where(arr % 2 == 1)
```

CODICE

```
import numpy as np

arr_2D = np.array([[1, 2, 6],
                   [4, 5, 6],
                   [7, 8, 9]])

indici = np.where(arr_2D > 5)

print("Indici degli elementi maggiori di 5:")
print(indici)
```

searchsorted

CODICE

```
import numpy as np

# Array ordinato
arr = np.array([1, 2, 4, 5])

# Elemento da inserire
elemento = 3

indice = np.searchsorted(arr, elemento)
print(f"Indice di inserimento per mantenere l'array ordinato: {indice}")
```

CODICE

```
import numpy as np

arr = np.array([1, 2, 4, 5])

elementi = [3, 6]

indici = np.searchsorted(arr, elementi)
print(f"Indici di inserimento: {indici}")
```

CODICE

```
import numpy as np

arr = np.array([1, 2, 4, 5])

# Elemento da inserire
elemento = 3

# Individuiamo l'indice di inserimento
indice = np.searchsorted(arr, elemento)

# Inseriamo l'elemento nell'array
arr_nuovo = np.insert(arr, indice, elemento)

print(f"Array originale: {arr}")
print(f"Array con l'elemento inserito: {arr_nuovo}")
```

Ordinamento, sì grazie

CODICE

```
import numpy as np
arr = np.array([2, 1, 4, 3, 5])
arr.sort()
print(f"Array ordinato: {arr}")
```

Ordinamento di base

CODICE

```
import numpy as np

arr = np.array([2, 1, 4, 3, 5])

arr_ordinato = np.sort(arr)

print(f"Array originale: {arr}")
print(f"Array ordinato: {arr_ordinato}")
```

Ordinamento di un array multidimensionale lungo un asse

CODICE

```
arr_2d = np.array([[5, 4, 7], [2, 3, 6], [1, 9, 8]])

# Ordinamento lungo l'asse delle righe (asse 1)
ordinato_asse1 = np.sort(arr_2d, axis=1)

# Ordinamento lungo l'asse delle colonne (asse 0)
ordinato_asse0 = np.sort(arr_2d, axis=0)

print("Array originale:\n", arr_2d)
print("\nOrdinato lungo l'asse delle righe:\n", ordinato_asse1)
print("\nOrdinato lungo l'asse delle colonne:\n", ordinato_asse0)
```

Ordinamento discendente

CODICE

```
import numpy as np

arr = np.array([2, 1, 4, 3, 5])

arr.sort()

print(f"Array ordinato: {arr}")

arr_ordinato_discendente = np.sort(arr)[::-1]
print(f"Array ordinato discendente: {arr_ordinato_discendente}")
```

CODICE

```
import numpy as np

arr = np.array(["Jacopo", "Carlo", "Ludovica"])

print(np.sort(arr))
```

CODICE

```
import numpy as np

arr = np.array([10, 59, 12, 66, 45])

filtro = [True, False, True, True, False]

arr_filtrato = arr[filtro]

print(arr_filtrato)
```

CODICE

```
import numpy as np

# Creazione di un array NumPy
array = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])

# Creazione di una maschera booleana
maschera = array > 5

# Applicazione della maschera per filtrare l'array
array_filtrato = array[maschera]

print(array_filtrato)

print(type(maschera))
```

Capitolo 10 - Pandas

Installazione e primo uso

COMANDO

```
python -m venv pandas_env
```

COMANDO

```
pip install pandas
```

COMANDO

```
python -m pip install pandas
```

CODICE

```
import pandas as pd
```

CODICE

```
print(pd.__version__)
```

DataFrame

CODICE

```
import pandas as pd

# Creazione di un DataFrame
data = {
    'Nome': ['Alice', 'Dorothy', 'Peter'],
    'Età': [7, 12, 13],
    'Stato': ['Paese delle Meraviglie', 'Stati Uniti', 'Inghilterra']
}

df = pd.DataFrame(data)
```

CODICE

```
print(df)
```

CODICE

```
display(df)
```

CODICE

```
from IPython.display import display
```

CODICE

```
print(df["Nome"])
```

CODICE

```
older_than_10 = df[df["Età"] > 10]
print(older_than_10)
```

CODICE

```
df["Adulto"] = df["Età"] >= 18
print(df)
```

CODICE

```
print(df.loc[1])
```

CODICE

```
df.at[1, "Stato"] = "USA"
print(df)
```

CODICE

```
df.rename(columns={"Età": "Anni"}, inplace=True)
print(df)
```

CODICE

```
# Calcolo della media dell'età
media_eta = df["Anni"].mean() # Usa la colonna rinominata "Anni"
print(f"Età media: {media_eta}")
```

CODICE

```
df.to_csv("individui.csv", index=False)
```

Importazione da un file CSV

CODICE

```
import pandas as pd

# Carichiamo il file CSV in un DataFrame
df = pd.read_csv("disney.csv")

# Visualizziamo il DataFrame
print(df)
```

CODICE

```
# Conteggio dei personaggi per specie
conteggio_specie = df["Specie"].value_counts()

print(conteggio_specie)
```

Importazione da un file JSON

CODICE

```
import pandas as pd

df = pd.read_json("popolazione.json")
print(df)
```

Analisi di un DataFrame

CODICE

```
pd.read_csv(
    "nome_file.csv",
    sep=";",
    encoding="utf-8",
    dtype={0: str},
    low_memory=False
)
```

CODICE

```
df = pd.read_csv("istat_popolazione_comuni.csv", sep=";", dtype={0: str})
```

CODICE

```
print(df.head())
```

CODICE

```
print(df.shape)
```

CODICE

```
print(df.columns)
```

CODICE

```
df.info()
```

CODICE

```
import pandas as pd

# Carichiamo il file CSV in un DataFrame
df = pd.read_csv("istat_popolazione_comuni.csv", dtype={0: str})

# Filtriamo il DataFrame per includere solo le righe
# in cui la colonna "Territorio" contiene il valore "Benevento"
df_filtered = df[df["Territorio"] == "Benevento"]

# Visualizziamo il DataFrame filtrato
print(df_filtered)
```

Filtraggio basato su condizioni

CODICE

```
df.loc[df["colonna"] > 10]
```

Applicazione di funzioni

COMANDO

```
python -m pip install matplotlib
```

CODICE

```
import pandas as pd
import matplotlib.pyplot as plt

# Dati di esempio
sales = [200, 220, 250, 275, 300, 320, 350, 370, 355, 340, 320, 310]
months = [
    "Gennaio", "Febbraio", "Marzo", "Aprile", "Maggio", "Giugno",
    "Luglio", "Agosto", "Settembre", "Ottobre", "Novembre", "Dicembre"
]

# Creazione del DataFrame
df = pd.DataFrame({"Vendite": sales, index=months})

print(df)
```

CODICE

```
df.plot(kind="line")

plt.title("Vendite mensili")
plt.xlabel("Mese")
plt.ylabel("Vendite")
plt.show()
```

CODICE

```
ax = df.plot(
    kind="bar",
    figsize=(10, 6)
)

ax.set_title("Vendite mensili")
ax.set_xlabel("Mese")
ax.set_ylabel("Vendite (€)")

plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```

Le Pandas Series

CODICE

```
import pandas as pd

# Creiamo una Series
s = pd.Series([1, 3, 5, 2, 8], index=["a", "b", "c", "d", "e"])

# Operazioni sulla Series
print(s.mean())          # Calcolo della media
print(s[s > s.median()]) # Selezione degli elementi maggiori della mediana
```

CODICE

```
pd.Series(data, index=index)
```

CODICE

```
index=["a", "b", "c", "d", "e"]
```

CODICE

```
s[s > s.median()]
```

CODICE

```
import matplotlib.pyplot as plt

s.plot()

plt.title("Esempio di Series")
plt.xlabel("Indice")
plt.ylabel("Valore")
plt.show()
```

Capitolo 12 - Apprendimento Supervisionato

Librerie e codice

COMANDO

```
python -m venv ml_env
```

COMANDO

```
python -m pip install scikit-learn numpy
```

CODICE

```
import sklearn
print(sklearn.__version__)
```

CODICE

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

# Dati di addestramento: peso e altezza degli animali
X_train = np.array([
    [22, 55], # Cane
    [8, 40],  # Cane
    [4, 25],  # Gatto
    [5, 30],  # Gatto
    [30, 60], # Cane
    [6, 35]   # Gatto
])

# Label associate ai dati di addestramento
y_train = np.array([
    "Cane",
    "Cane",
    "Gatto",
    "Gatto",
    "Cane",
    "Gatto"
])

# Creazione del modello KNN con K=3
knn = KNeighborsClassifier(n_neighbors=3)

# Addestramento del modello
knn.fit(X_train, y_train)

# Nuovo esempio da classificare: peso e altezza
X_new = np.array([[7, 35]])

# Previsione della classe per il nuovo esempio
prediction = knn.predict(X_new)

print(prediction)
```

CODICE

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
```

CODICE

```
X_train = np.array([
    [22, 55], # Cane
    [8, 40],  # Cane
    [4, 25],  # Gatto
    [5, 30],  # Gatto
    [30, 60], # Cane
    [6, 35]   # Gatto
])
```

CODICE

```
y_train = np.array([
    "Cane",
    "Cane",
    "Gatto",
    "Gatto",
    "Cane",
    "Gatto"
])
```

CODICE

```
knn = KNeighborsClassifier(n_neighbors=3)
```

CODICE

```
knn.fit(X_train, y_train)
```

CODICE

```
X_new = np.array([[7, 35]])
```

CODICE

```
prediction = knn.predict(X_new)
```

CODICE

```
print(prediction)
```

Capitolo 13 - Matematica, statistica e regressione

Funzioni matematiche

CODICE

```
y = f(x)
```

Funzioni lineari

CODICE

```
y = mx + b
```

CODICE

```
y = 2x + 3
```

Modello di Regressione Lineare

CODICE

```
y = mx + b
```

COMANDO

```
python -m pip install numpy matplotlib scikit-learn
```

CODICE

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Dati di esempio: ore di studio (X) e voti (y)
X = np.array([1, 2, 3, 4, 5, 6]).reshape(-1, 1) # Variabile indipendente
y = np.array([2, 3, 5, 4, 6, 6])             # Variabile dipendente

# Creazione del modello di regressione lineare
model = LinearRegression()

# Addestramento del modello
model.fit(X, y)

# Otteniamo i parametri m e b della retta
m = model.coef_[0]
b = model.intercept_

# Stampiamo la retta appresa
print(f"Retta appresa: y = {m:.2f}x + {b:.2f}")

# Esempio di previsione su un nuovo dato
X_new = np.array([[7]]) # 7 ore di studio
y_new_pred = model.predict(X_new)

print(f"Previsione per 7 ore di studio: {y_new_pred[0]:.2f}")

# Grafico
plt.figure(figsize=(10, 6))
plt.scatter(X, y, label='Dati di esempio')

# Dati per il plotting della retta di regressione
X_grafico = np.array([1, 9]).reshape(-1, 1)
y_grafico = m * X_grafico + b
plt.plot(X_grafico, y_grafico, label='Retta di regressione')

plt.scatter(X_new, y_new_pred, marker='x', s=100, label='Nuova previsione')
plt.xlabel('Ore di studio')
plt.ylabel('Voti ottenuti')
plt.title('Esempio di Regressione Lineare')
plt.legend()
plt.grid()
plt.show()
```

CODICE

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

CODICE

```
from sklearn.linear_model import LinearRegression
```

CODICE

```
X = np.array([1, 2, 3, 4, 5, 6]).reshape(-1, 1)
y = np.array([2, 3, 5, 4, 6, 6])
```

CODICE

```
model = LinearRegression()
```

CODICE

```
model.fit(X, y)
```

CODICE

```
m = model.coef_[0]
b = model.intercept_
```

CODICE

```
print(f"Retta appresa: y = {m:.2f}x + {b:.2f}")
```

CODICE

```
print(f"Previsione per 7 ore di studio: {y_new_pred[0]:.2f}")
```

CODICE

```
plt.figure(figsize=(10, 6))
```

CODICE

```
plt.figure(figsize=(10, 6), dpi=200)
```

CODICE

```
plt.scatter(X, y, label='Dati di esempio')
```

CODICE

```
X_grafico = np.array([1, 9]).reshape(-1, 1)
y_grafico = m * X_grafico + b
plt.plot(X_grafico, y_grafico, label='Retta di regressione')
```

CODICE

```
plt.scatter(X_new, y_new_pred, marker='x', s=100, label='Nuova previsione')
```

CODICE

```
plt.xlabel('Ore di studio')
plt.ylabel('Voti ottenuti')
plt.title('Esempio di Regressione Lineare')
```

CODICE

```
plt.legend()
```

CODICE

```
plt.grid()
plt.show()
```

Capitolo 15 - Dentro una rete neurale

Un neurone in codice: il prodotto scalare con NumPy

CODICE

```
import numpy as np

# Dati di input (un array con un solo valore)
input_data = np.array([0.5])

# Peso del neurone
weights = np.array([1.5])

# Bias (aggiunto al risultato ponderato)
bias = -0.2

# Calcolo del valore z del neurone
z = np.dot(input_data, weights) + bias

# Funzione di attivazione (step function)
activated_output = 1 if z > 0 else 0

print("Valore z:", z)
print("Output attivato:", activated_output)
```

CODICE

```
a = [a1, a2, ..., an]
b = [b1, b2, ..., bn]
```

CODICE

```
input_data = np.array([0.5, 1.0, -0.5]) # Tre input
weights = np.array([1.5, -0.5, 2.0])    # Tre pesi

# Calcolo della somma ponderata
result = np.dot(input_data, weights)

print(result)
```

Una piccola rete neurale costruita da zero con NumPy

CODICE

```
import numpy as np

# 1 - Creiamo il dataset (X, y) come array numpy
X = np.array([[1.0],
              [1.5],
              [2.0],
              [3.0],
              [3.5],
              [4.0]])

y = np.array([[0],
              [0],
              [0],
              [1],
              [1],
              [1]])

# 2 - Inizializzazione casuale di pesi e bias
np.random.seed(42) # per riproducibilità
weights = np.random.rand(1, 1) # un unico peso
bias = np.random.rand(1) # un unico bias

# 3 - Parametri di training
learning_rate = 0.1
epochs = 3000

# 4 - Definizione della sigmoide e della sua derivata
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def sigmoid_derivative(z):
    s = sigmoid(z)
    return s * (1 - s)

# 5 - Training (Backpropagation)
for epoch in range(epochs):
    # Forward pass
    z = np.dot(X, weights) + bias # dimensioni: (6,1)
    predictions = sigmoid(z) # stesse dimensioni di Y

    # Errore
    error = predictions - y # (6,1)

    # Calcolo del gradiente
    dZ = error * sigmoid_derivative(z) # (6,1)
    gradient = np.dot(X.T, dZ) # (1,6) * (6,1) -> (1,1)

    # Aggiorniamo i pesi (weights) e il bias
    weights = weights - learning_rate * gradient
    bias = bias - learning_rate * np.sum(dZ)

# 6 - Test del modello
test_inputs = np.array([[2.25], [2.5], [4.5]])
test_outputs = sigmoid(np.dot(test_inputs, weights) + bias)

print("Pesi (weights) appresi:", weights)
print("Bias appreso:", bias)
print()

for x_val, pred in zip(test_inputs, test_outputs):
    print(f"Input = {x_val[0]:.1f}, Predizione sigmoide = {pred[0]:.3f}, Classe = {int(pred[0] >= 0.5)}")
```

1 - Creazione del dataset

CODICE

```
X = np.array([[1.0],
              [1.5],
              [2.0],
              [3.0],
              [3.5],
              [4.0]])
```

CODICE

```
y = np.array([[0],
              [0],
              [0],
              [1],
              [1],
              [1]])
```

2 - Inizializzazione dei parametri

CODICE

```
np.random.seed(42)
```

CODICE

```
np.random.seed(int(time.time()))
```

CODICE

```
weights = np.random.rand(1, 1)
```

CODICE

```
numpy.random.rand(d0, d1, ..., dn)
```

CODICE

```
bias = np.random.rand(1)
```

CODICE

```
bias = 0.5
```

3 - Parametri di training della rete: learning rate ed epochs

CODICE

```
learning_rate = 0.1
```

CODICE

```
parametro = parametro - (learning rate × gradiente)
```

Epochs

CODICE

```
epochs = 3000
```

Funzioni di attivazione: definizione della sigmoide e della sua derivata

CODICE

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

CODICE

```
np.exp(-z)
```

CODICE

```
def sigmoid_derivative(z):
    s = sigmoid(z)
    return s * (1 - s)
```

CODICE

```
s = sigmoid(z)
```

CODICE

```
return s * (1 - s)
```

Ciclo di addestramento, o training loop

CODICE

```
# Forward pass
z = np.dot(X, weights) + bias # dimensioni: (6, 1)
predictions = sigmoid(z)     # stesse dimensioni di y
```

CODICE

```
error = predictions - y # dimensioni: (6, 1)
```

CODICE

```
dZ = error * sigmoid_derivative(z) # dimensioni: (6, 1)
gradient = np.dot(X.T, dZ)          # (1, 6) * (6, 1) -> (1, 1)
```

CODICE

```
gradient = 1.0*dZ1 + 1.5*dZ2 + 2.0*dZ3 + 3.0*dZ4 + 3.5*dZ5 + 4.0*dZ6
```

CODICE

```
np.dot(X.T, dZ)
```

CODICE

```
weights = weights - learning_rate * gradient
bias = bias - learning_rate * np.sum(dZ)
```

CODICE

```
weights -= learning_rate * gradient
bias -= learning_rate * np.sum(dZ)
```

Test del modello

CODICE

```
test_inputs = np.array([[2.25], [2.5], [4.5]])
```

CODICE

```
test_outputs = sigmoid(np.dot(test_inputs, weights) + bias)
```

CODICE

```
print("Pesi (weights) appresi:", weights)
print("Bias appreso:", bias)
print()

for x_val, pred in zip(test_inputs, test_outputs):
    print(f"Input = {x_val[0]:.2f}, Predizione sigmoide = {pred[0]:.3f}, Classe = {int(pred[0] >= 0.5)}")
```

CODICE

```
int(pred[0] >= 0.5)
```

Una verifica grafica

CODICE

```
import numpy as np
import matplotlib.pyplot as plt

# Dataset originale
X = np.array([[1.0], [1.5], [2.0], [3.0], [3.5], [4.0]])
y = np.array([0, 0, 0, 1, 1, 1])

# Pesi e bias appresi nell'esempio precedente
weights = np.array([[3.31827382]])
bias = np.array([-8.17592369])

# Funzione sigmoide
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Generazione di valori per disegnare la curva sigmoide
x_range = np.linspace(0.5, 4.5, 200).reshape(-1, 1)
z = np.dot(x_range, weights) + bias
predictions = sigmoid(z)

# Plot
plt.figure(figsize=(8, 5))
plt.plot(x_range, predictions, label="Output della rete (sigmoide)")
plt.scatter(X, y, label="Dati reali", zorder=5)
plt.axhline(0.5, linestyle="--", label="Soglia 0.5")
plt.xlabel("Input")
plt.ylabel("Output / Classe")
plt.title("Classificazione binaria con un neurone sigmoide")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Capitolo 16 - Le reti neurali in pratica con PyTorch e TensorFlow

PyTorch

CODICE

```
X = np.array([[1.0], [1.5], [2.0], [3.0], [3.5], [4.0]])
y = np.array([[0], [0], [0], [1], [1], [1]])
```

I tensori

COMANDO

```
pip install torch numpy
```

CODICE

```
import torch

# Tensore 1D
a = torch.tensor([1.0, 2.0, 3.0])
```

CODICE

```
print(a.shape)
```

CODICE

```
X = torch.tensor([[1.0, 2.0], [3.0, 4.0]])
print(X)
```

CODICE

```
b = torch.tensor([[1.0], [2.0], [3.0]])
```

CODICE

```
print(b.shape)
```

CODICE

```
a = torch.tensor([[1.0], [2.0], [3.0]])
```

CODICE

```
print(a.dtype)
```

CODICE

```
a = torch.tensor([[1.0], [2.0], [3.0]], dtype=torch.float32)
```

CODICE

```
import numpy as np
import torch

np_array = np.array([[1.0, 2.0]])
torch_tensor = torch.from_numpy(np_array)
back_to_numpy = torch_tensor.numpy()
```

CODICE

```
torch_tensor = torch.from_numpy(np_array).clone()
```

PyTorch all'opera

CODICE

```
import torch
import torch.nn as nn
import torch.optim as optim

# Per rendere l'esempio riproducibile
torch.manual_seed(42)

# 1 - Dataset in torch.Tensor
X_torch = torch.tensor([[1.0],
                        [1.5],
                        [2.0],
                        [3.0],
                        [3.5],
                        [4.0]], dtype=torch.float32)

y_torch = torch.tensor([[0.0],
                        [0.0],
                        [0.0],
                        [1.0],
                        [1.0],
                        [1.0]], dtype=torch.float32)

# 2 - Definizione del modello
class SimpleNet(nn.Module):
    def __init__(self):
        super(SimpleNet, self).__init__()
        self.linear = nn.Linear(1, 1) # 1 input, 1 output

    def forward(self, x):
        return torch.sigmoid(self.linear(x))

model = SimpleNet()

# 3 - Loss e ottimizzatore
criterion = nn.BCELoss() # Binary Cross Entropy
optimizer = optim.SGD(model.parameters(), lr=0.1)

# 4 - Training loop
for epoch in range(3000):
    optimizer.zero_grad()

    predictions = model(X_torch)
    loss = criterion(predictions, y_torch)

    loss.backward()
    optimizer.step()

# 5 - Test su dati non presenti nel training set
with torch.no_grad():
    test_inputs = torch.tensor([[2.25],
                                [2.50],
                                [4.50]], dtype=torch.float32)

    test_predictions = model(test_inputs)

    print("Predizioni PyTorch:")

    for x_val, pred in zip(test_inputs, test_predictions):
        print(
            f"Input: {x_val.item():.2f} -> "
            f"Classe: {int(pred.item() >= 0.5)} "
            f"(Prob: {pred.item():.3f})"
        )
```

Importazione delle librerie

CODICE

```
import torch
```

CODICE

```
import torch.nn as nn
```

CODICE

```
torch.nn.Linear(...)
```

CODICE

```
nn.Linear(...)
```

CODICE

```
import torch.optim as optim
```

Il seme del generatore casuale di PyTorch

CODICE

```
torch.manual_seed(42)
```

Creazione del dataset

CODICE

```
# 1 - Dataset in torch.Tensor
X_torch = torch.tensor([[1.0],
                        [1.5],
                        [2.0],
                        [3.0],
                        [3.5],
                        [4.0]], dtype=torch.float32)

y_torch = torch.tensor([[0.0],
                        [0.0],
                        [0.0],
                        [1.0],
                        [1.0],
                        [1.0]], dtype=torch.float32)
```

CODICE

```
print(X_torch.shape)
print(y_torch.shape)
```

La definizione del modello

CODICE

```
class SimpleNet(nn.Module):
    def __init__(self):
        super(SimpleNet, self).__init__()
        self.linear = nn.Linear(1, 1)

    def forward(self, x):
        return torch.sigmoid(self.linear(x))

model = SimpleNet()
```

CODICE

```
def __init__(self):
    super(SimpleNet, self).__init__()
    self.linear = nn.Linear(1, 1)
```

CODICE

```
super(SimpleNet, self).__init__()
```

CODICE

```
self.linear = nn.Linear(1, 1)
```

CODICE

```
def forward(self, x):
    return torch.sigmoid(self.linear(x))
```

CODICE

```
model(X_torch)
```

CODICE

```
oggetto(...)
```

CODICE

```
model(X_torch)
```

CODICE

```
model.forward(X_torch)
```

CODICE

```
model = SimpleNet()
```

Loss e ottimizzatore

CODICE

```
criterion = nn.BCELoss()
```

CODICE

```
loss = criterion(predictions, y_torch)
```

CODICE

```
optimizer = optim.SGD(model.parameters(), lr=0.1)
```

CODICE

```
model.parameters()
```

CODICE

```
lr=0.1
```

Training loop: il cuore dell'addestramento del modello

CODICE

```
for epoch in range(3000):
    optimizer.zero_grad()

    predictions = model(X_torch)
    loss = criterion(predictions, y_torch)

    loss.backward()
    optimizer.step()
```

CODICE

```
for epoch in range(3000):
```

CODICE

```
optimizer.zero_grad()
```

CODICE

```
predictions = model(X_torch)
```

CODICE

```
model(X_torch)
```

CODICE

```
def forward(self, x):
    return torch.sigmoid(self.linear(x))
```

CODICE

```
self.linear(x)
```

CODICE

```
torch.sigmoid(z)
```

CODICE

```
predictions = model(X_torch)
```

CODICE

```
loss = criterion(predictions, y_torch)
```

CODICE

```
criterion = nn.BCELoss()
```

CODICE

```
loss = criterion(predictions, y_torch)
```

CODICE

```
loss.backward()
```

CODICE

```
optimizer.step()
```

La fase di test

CODICE

```
with torch.no_grad():
```

CODICE

```
test_inputs = torch.tensor([[2.25],  
                             [2.50],  
                             [4.50]], dtype=torch.float32)
```

CODICE

```
test_predictions = model(test_inputs)
```

CODICE

```
for x_val, pred in zip(test_inputs, test_predictions):  
    print(  
        f"Input: {x_val.item():.2f} -> "  
        f"Classe: {int(pred.item() >= 0.5)} "  
        f"(Prob: {pred.item():.3f})"  
    )
```

Il codice completo

CODICE

```
import tensorflow as tf
import numpy as np

# Per rendere l'esempio riproducibile
tf.keras.utils.set_random_seed(42)

# 1 - Dataset
X_tf = np.array([[1.0],
                 [1.5],
                 [2.0],
                 [3.0],
                 [3.5],
                 [4.0]], dtype=np.float32)

y_tf = np.array([[0.0],
                 [0.0],
                 [0.0],
                 [1.0],
                 [1.0],
                 [1.0]], dtype=np.float32)

# 2 - Definizione del modello
model = tf.keras.Sequential([
    tf.keras.Input(shape=(1,)),
    tf.keras.layers.Dense(1, activation="sigmoid")
])

# 3 - Compilazione
model.compile(
    optimizer=tf.keras.optimizers.SGD(learning_rate=0.1),
    loss="binary_crossentropy"
)

# 4 - Addestramento
model.fit(X_tf, y_tf, epochs=3000, verbose=0)

# 5 - Test su dati non presenti nel training set
test_inputs = np.array([[2.25],
                        [2.50],
                        [4.50]], dtype=np.float32)

predictions = model.predict(test_inputs, verbose=0)

print("Predizioni TensorFlow:")

for x_val, pred in zip(test_inputs, predictions):
    print(
        f"Input: {x_val[0]:.2f} -> "
        f"Classe: {int(pred[0] >= 0.5)} "
        f"(Prob: {pred[0]:.3f})"
    )
```

Le librerie

CODICE

```
import tensorflow as tf
```

COMANDO

```
python -m pip install tensorflow
```

Riproducibilità dell'esempio

CODICE

```
tf.keras.utils.set_random_seed(42)
```

I dati di addestramento

CODICE

```
X_tf = np.array([[1.0],
                 [1.5],
                 [2.0],
                 [3.0],
                 [3.5],
                 [4.0]], dtype=np.float32)

y_tf = np.array([[0.0],
                 [0.0],
                 [0.0],
                 [1.0],
                 [1.0],
                 [1.0]], dtype=np.float32)
```

Definizione del modello neurale

CODICE

```
model = tf.keras.Sequential([
    tf.keras.Input(shape=(1,)),
    tf.keras.layers.Dense(1, activation="sigmoid")
])
```

CODICE

```
tf.keras.Input(shape=(1,))
```

CODICE

```
tf.keras.layers.Dense(1, activation="sigmoid")
```

Compilazione del modello

CODICE

```
model.compile(
    optimizer=tf.keras.optimizers.SGD(learning_rate=0.1),
    loss="binary_crossentropy"
)
```

CODICE

```
optimizer=tf.keras.optimizers.SGD(learning_rate=0.1)
```

CODICE

```
loss="binary_crossentropy"
```

Fase di Addestramento (Training)

CODICE

```
model.fit(X_tf, y_tf, epochs=3000, verbose=0)
```

Fase di Test: valutiamo il modello su nuovi dati

CODICE

```
test_inputs = np.array([[2.25],
                        [2.50],
                        [4.50]], dtype=np.float32)
```

CODICE

```
predictions = model.predict(test_inputs, verbose=0)
```

CODICE

```
print("Predizioni TensorFlow:")

for x_val, pred in zip(test_inputs, predictions):
    print(
        f"Input: {x_val[0]:.2f} -> "
        f"Classe: {int(pred[0] >= 0.5)} "
        f"(Prob: {pred[0]:.3f})"
    )
```

Capitolo 17 - LLM: i Modelli Linguistici di Grandi Dimensioni

Installazione della libreria

COMANDO

```
python -m pip install -U google-genai
```

Il codice completo

CODICE

```
from google import genai

# Inserire qui la propria API key
api_key = "INSERISCI QUI LA TUA CHIAVE_API"

client = genai.Client(api_key=api_key)

response = client.models.generate_content(
    model="gemini-3.6-flash",
    contents="Cosa pensi, in poche parole, della singolarità tecnologica?"
)

print(response.text)
```

CODICE

```
from google import genai
```

CODICE

```
api_key = "INSERISCI QUI LA TUA CHIAVE_API"
```

CODICE

```
client = genai.Client(api_key=api_key)
```

CODICE

```
response = client.models.generate_content(
    model="gemini-3.6-flash",
    contents="Cosa pensi, in poche parole, della singolarità tecnologica?"
)
```

CODICE

```
print(response.text)
```

Un esempio più guidato: istruzioni di sistema e livello di ragionamento

CODICE

```
from google import genai
from google.genai import types

# Inserire qui la propria API key
api_key = "INSERISCI QUI LA TUA CHIAVE_API"

client = genai.Client(api_key=api_key)

response = client.models.generate_content(
    model="gemini-3.6-flash",
    contents="Spiegami in poche parole la differenza tra intelligenza e coscienza.",
    config=types.GenerateContentConfig(
        system_instruction=(
            "Rispondi in italiano, con tono chiaro e didattico. "
            "Organizza la risposta in tre punti: intelligenza, coscienza, "
            "differenza principale."
        ),
        thinking_config=types.ThinkingConfig(
            thinking_level="low"
        )
    )
)

print(response.text)
```

CODICE

```
from google.genai import types
```

CODICE

```
contents="Spiegami in poche parole la differenza tra intelligenza e coscienza."
```

CODICE

```
config=types.GenerateContentConfig(
    system_instruction=(
        "Rispondi in italiano, con tono chiaro e didattico. "
        "Organizza la risposta in tre punti: intelligenza, coscienza, "
        "differenza principale."
    ),
    thinking_config=types.ThinkingConfig(
        thinking_level="low"
    )
)
```

CODICE

```
thinking_config=types.ThinkingConfig(
    thinking_level="low"
)
```

Un minimo di controllo degli errori

CODICE

```
from google import genai
from google.genai import types

# Inserire qui la propria API key
api_key = "INSERISCI QUI LA TUA CHIAVE_API"

try:
    client = genai.Client(api_key=api_key)

    response = client.models.generate_content(
        model="gemini-3.6-flash",
        contents="Spiegami in poche parole la differenza tra intelligenza e coscienza.",
        config=types.GenerateContentConfig(
            system_instruction=(
                "Rispondi in italiano, con tono chiaro e didattico. "
                "Organizza la risposta in tre punti: intelligenza, coscienza, "
                "differenza principale."
            ),
            thinking_config=types.ThinkingConfig(
                thinking_level="low"
            )
        )
    )

    try:
        print(response.text)
    except ValueError:
        print("La richiesta non ha prodotto una risposta testuale valida.")

    if response.candidates:
        print("Dettagli:")

        for candidate in response.candidates:
            finish_reason = candidate.finish_reason

            if hasattr(finish_reason, "name"):
                finish_reason = finish_reason.name

            print(f"Motivo di conclusione: {finish_reason}")

            if candidate.safety_ratings:
                print("Valutazioni di sicurezza:")

                for rating in candidate.safety_ratings:
                    category = rating.category
                    probability = rating.probability

                    if hasattr(category, "name"):
                        category = category.name

                    if hasattr(probability, "name"):
                        probability = probability.name

                    print(f"- Categoria: {category}, probabilità: {probability}")

            else:
                print("Nessun dettaglio disponibile.")

except Exception as error:
    print("Si è verificato un errore durante la chiamata a Gemini.")
    print("Dettaglio dell'errore:")
    print(error)
```

CODICE

```
print(response.text)
```