```
import numpy as np
import pandas as pd


# Una series è un dataframe composto da una sola colonna. Abbiamo
quindi una colonna di indici (da 0 a n-1) e la nostra colonna, che può
essere
# di qualsiasi tipo. ESEMPIO


prezzi = [125,322,9,44]

prezzi_series = pd.Series(prezzi)
prezzi_series
```

```
0    125
1    322
2      9
3     44
dtype: int64
```

```
type(prezzi_series)
```

```
pandas.core.series.Series
```

```
# Abbiamo quindi la nostra variabile prezzi e l'indice associato parte
da 0 e termina a n-1. Possiamo posticipare l'indice


prezzi_index = ["productA","productB","productC","productD"]
prezzi_index
```

```
['productA', 'productB', 'productC', 'productD']
```

```
prices = pd.Series(data=prezzi,index=prezzi_index)
prices
```

```
productA    125
productB    322
productC      9
productD     44
dtype: int64
```

```
# OPPURE

prices = pd.Series(prezzi,prezzi_index)
prices
```

```
productA    125
productB    322
productC      9
productD     44
dtype: int64
```

```
# Accedere agli elementi della Series. Possiamo accedervi sia tramite indice, sia tramite label


prices[1]
```

```
C:\Users\jgian\AppData\Local\Temp\ipykernel_15332\3717727957.py:1: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  prices[1]
```

```
322
```

```
prices.iloc[1]
```

```
322
```

```
prices["productB"]
```

```
322
```

```
prices.loc["productB"]
```

```
322
```

```
# Abbiamo costruito una series partendo da una lista. Possiamo crearne una partendo da un dizionario


prezzo_cani = {"SHIBAINO":135,"BASSOTTO":200,"ALANO":190,"HUSKY":450}
prezzo_cani
```

```
{'SHIBAINO': 135, 'BASSOTTO': 200, 'ALANO': 190, 'HUSKY': 450}
```

```
prezzo_cani_series = pd.Series(prezzo_cani)
prezzo_cani_series
```

```
SHIBAINO    135
BASSOTTO    200
ALANO       190
HUSKY       450
dtype: int64
```

```
# Gli indici, in automatico, sono i label
```

```
# Partiamo ora da due dizionari aventi stesse chiavi:
```

```
student1 = {
    "name": "John Doe",
    "age": 20,
    "major": "Computer Science"
}

student2 = {
    "name": "Jack Bauer",
    "age": 44,
    "major": "Machine learning"
}

student1
```

```
{'name': 'John Doe', 'age': 20, 'major': 'Computer Science'}
```

```
student2
```

```
{'name': 'Jack Bauer', 'age': 44, 'major': 'Machine learning'}
```

```
# Trasformiamo ognuno in una pandas series

student1_series = pd.Series(student1)
student2_series = pd.Series(student2)

student1_series
```

```
name             John Doe
age                    20
```

```
major    Computer Science
dtype: object
```

```
student2_series
```

```
name           Jack Bauer
age                    44
major    Machine learning
dtype: object
```

```
# Possiamo accedere ad un elemento della Series tramite loc:
```

```
student2_series.loc["age"]
```

```
44
```

```
# Oppure possiamo accedere ad un elemento tramite posizione
dell'indice utilizzando iloc
```

```
student2_series.iloc[1]
```

```
44
```

```
# Solitamente si accede ad un elemento tramite loc["Label
dell'indice"].
```

```
# Per estrarre una lista di tutte le chiavi di una Series si utilizza
il metodo keys:
```

```
student2_series.keys()
```

```
Index(['name', 'age', 'major'], dtype='object')
```

```
type(student2_series.keys())
```

```
pandas.core.indexes.base.Index
```

```
# OPERAZIONI TRA SERIES
```

```
student2_series*3
```

```
name                        Jack BauerJack BauerJack Bauer
age                                                    132
```

```
major    Machine learningMachine learningMachine learning
dtype: object
```

```
# Moltiplicando una Series per uno scalare, avremo che i valori di
tipo string verranno concatenati per un numero di volte pari al valore
dello scalare,
# mentre i valori di tipo numerico verranno moltiplicati per esso.
```

```
student1_series + student2_series
```

```
name                   John DoeJack Bauer
age                                    64
major    Computer ScienceMachine learning
dtype: object
```

```
# Sommando due Series aventi stesse chiavi, avremo che le stringhe
vengono concatenate mentre i valori numerici vengono tra loro sommati.
```

```
#student1_series/3
# Nel caso di name e major abbiamo una stringa che divide lo scalare 3
==> Otterremo un errore
```

```
# CONSIDERIAMO ORA DUE SERIES AVENTI CHIAVI DIVERSE E SOMMIAMOLE
```

```
student1 = {
    "name": "John Doe",
    "age": 20,
    "major": "Computer Science"
}

student2 = {
    "name": "Jack Bauer",
    "age": 44,
    "major": "Machine learning",
    "stip_mensile":0,
    "lavoro_prob": 8
}

student1_series = pd.Series(student1)
student2_series = pd.Series(student2)

student1_series
```

```
name              John Doe
age                     20
major    Computer Science
dtype: object
```

```
student2_series
```

```
name                    Jack Bauer
age                             44
major           Machine learning
stip_mensile                     0
lavoro_prob                      8
dtype: object
```

```
student1_series + student2_series
```

```
age                                                64
lavoro_prob                                       NaN
major           Computer ScienceMachine learning
name                           John DoeJack Bauer
stip_mensile                                      NaN
dtype: object
```

```
# Vengono sommati solo i valori rispettivi alle chiavi presenti sia in
student1_series sia in student2_series.
# Le chiavi che non si trovano in entrambe le liste avranno un valori
corrispondente pari a NaN (Not a Number)

# Se abbiamo un NaN, quindi, la chiave corrispondende NON si trova in
entrambe le Series.



# Se i NaN non ci piacciono, possiamo sostituirli con un valore a
piacere. Per farlo, dobbiamo sommare le due Series tramite il metodo
add, e NON
# tramite il simbolo +. Vediamo quindi come fare



student1_series.add(student2_series,fill_value=3) # Non possiamo
inserire una stringa come valore di fill_value perché alcuni argomenti
# sono numeri ==> Un numero sommato ad una stringa genera un errore.
```

```
age                                                64
lavoro_prob                                        11
major           Computer ScienceMachine learning
name                           John DoeJack Bauer
stip_mensile                                        3
dtype: object
```

```
# Questo significa che, tramite l'aggiunta del parametro fill_value,
viene considerato il valore delle chiavi che si trovano in una sola
series,
# e tale valore viene sommato al valore specificato nel "fill_value".


# CONSIDERIAMO ORA DUE SERIES NUMERICHE, FORMATE DA TUTTI INT.
FACCIAMO SI CHE IN d2 CI SIA UNA COPPIA CHIAVE-VALORE NON PRESENTE IN
d1


d1 = {"A":12,"B":2,"C":44}
d1
```

```
{'A': 12, 'B': 2, 'C': 44}
```

```
d2 = {"A":7,"B":10,"C":1,"D":16}
d2
```

```
{'A': 7, 'B': 10, 'C': 1, 'D': 16}
```

```
d1_series = pd.Series(d1)
d1_series
```

```
A    12
B     2
C    44
dtype: int64
```

```
d2_series = pd.Series(d2)
d2_series
```

```
A     7
B    10
C     1
D    16
dtype: int64
```

```
d1_series.dtype
```

```
dtype('int64')
```

```
d2_series.dtype
```

```
dtype('int64')
```

```
sumd1d2 = d1_series+d2_series
sumd1d2
```

```
A    19.0
B    12.0
C    45.0
D     NaN
dtype: float64
```

```
sumd1d2.dtype
```

```
dtype('float64')
```

```
# ATTENZIONE!! NONOSTANTE LE DUE SERIES DI PARTENZA FOSSERO DI TIPO
INT64, LA LORO SOMMA E' COMPOSTA DA ELEMENTI DI TIPO FLOAT64!

# COSA SUCCEDE SE SOMMIAMO ENTRAMBE LE SERIES TRAMITE LA FUNZIONE ADD?



sommaConAdd = d1_series.add(d2_series, fill_value=2)
sommaConAdd
```

```
A    19.0
B    12.0
C    45.0
D    18.0
dtype: float64
```

```
sommaConAdd.dtype
```

```
dtype('float64')
```

```
# STESSO RISULTATO DI PRIMA. LA TIPOLOGIA DELLA SOMMA DEI VALORI E'
FLOAT64, NONOSTANTE I DUE VALORI DI PARTENZA FOSSERO INT64.
```