

# La Programmazione ad Oggetti in Python

**Docente:** Ambra Demontis

**Anno Accademico:** 2021 - 2022

Corso di Laurea in Ingegneria Elettrica, Elettronica e Informatica



University of Cagliari,  
Italy

Department of Electrical and  
Electronic Engineering



# La Programmazione ad Oggetti in Python

In queste slide vedremo come estendere classi built-in.

# Classi Built-in

Python mette a disposizione diverse classi predefinite.

Ad esempio, le strutture dati che abbiamo visto (tuple, liste e dizionari) sono tutte classi.

Se sono degli oggetti perchè non vengono istanziati con la stessa sintassi degli altri oggetti?

Semplicemente per far si che il programmatore debba scrivere meno codice.

# Classi Built-in

Ad esempio, possiamo creare un oggetto lista vuoto sia con la sintassi.

```
oggetto_lista_vuota = []
```

Sia con la sintassi:

```
oggetto_lista_vuota = list()
```

In entrambi i casi, ad esempio:

```
print(oggetto_lista)
```

Stamperà: []

# Estendere Classi Built-in

Come per tutti gli altri oggetti, è possibile utilizzare gli oggetti built-in come classe padre.

Ad esempio nel caso in cui vogliamo avere un oggetto che abbia tutti i metodi dell'oggetto built-in ma ne abbia anche dei nuovi, implementati da noi o vogliamo modificare il funzionamento di alcuni metodi.

# Estendere Classi Built-in

Supponiamo di voler creare una classe chiamata CNuovaLista che eredita dalla classe lista.

```
class CNuovaLista(list):  
    pass
```

Questa classe erediterà attributi e metodi della classe delle liste.

Potremo creare un oggetto appartenente a questa classe con la solita sintassi, es:

```
oggetto_lista = CNuovaLista()
```

# Estendere Classi Built-in

Per poter aggiungere elementi all'oggetto possiamo sfruttare la funzione `append` della classe `lista`.

Questa funzione aggiunge un elemento in coda alla lista. Es:

```
oggetto_lista.append(6)
```

```
oggetto_lista.append(4)
```

```
print(oggetto_lista) # la funzione print può essere utilizzata per visualizzare il  
contenuto di una struttura dati (come una lista, una tupla, un dizionario).
```

Stamperà:

```
[6, 4]
```

# Estendere Classi Built-in

Supponiamo ora di voler modificare la funzione `append` in modo che, prima di appendere un elemento stampi a schermo la stringa:

appendo l'elemento `x`

Dove `x` dovrà essere l'elemento che andrà ad aggiungere.



# Estendere Classi Built-in

```
class CNuovaLista(list):  
  
    def append(self, x):  
        print("appendo l'elemento ", x)  
        super().append(x)
```

# Estendere Classi Built-in

```
oggetto_lista = CNuovaLista()  
oggetto_lista.append(6)  
oggetto_lista.append(4)  
print(len(oggetto_lista))
```

Stamperà:

```
appendo l'elemento 6  
appendo l'elemento 4  
2
```

## Esercizio sull' Estendere Classi Built-in

Creare la classe CListaElementiUnici che eredita dalla classe built-in lista.

Effettuare l'override della funzione append in modo che prima di appendere un nuovo elemento controlli se è già presente nella lista e, se già presente, non lo inserisca e stampi a schermo il messaggio: l'elemento è già presente nella lista.

# Esercizio sull' Estendere Classi Built-in

```
class CListaElementiUnici(list):
```

```
    def append(self, x):
```

```
        # cerca l'elemento x nella lista
```

```
        trovato = False
```

```
        for elem in self:
```

```
            if elem == x:
```

```
                trovato = True
```

```
        if trovato is True:
```

```
            print("L'elemento è già presente nella lista.")
```

```
        else:
```

```
            print("Appendo l'elemento ", x)
```

```
            super().append(x)
```

# Esercizio sull' Estendere Classi Built-in

```
oggetto_lista = CListaElementiUnici()
oggetto_lista.append(6)
oggetto_lista.append(4)
oggetto_lista.append(6)
print(oggetto_lista)
print(len(oggetto_lista))
```

Stamperà:

Appendo l'elemento 6

Appendo l'elemento 4

L'elemento è già presente nella lista.

[6, 4]

# Esercizio sull' Estendere Classi Built-in

Nota: L'istanza dell'oggetto (che la funzione `append` riceve come primo argomento) è una lista, pertanto la possiamo usarla come tale.

Nella soluzione la scorriamo con un'istruzione iterativa `for`.

Tuttavia potremmo anche, ad esempio, utilizzare l'operatore di selezione per estrarre alcuni elementi, e.g., `self[1:3]`.