



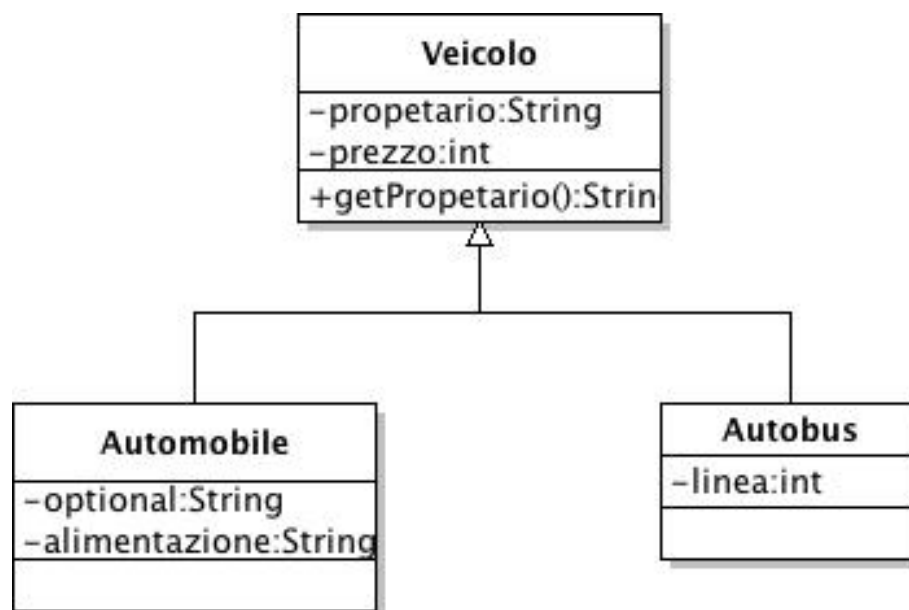
# ereditarietà e polimorfismo

Python

## *ereditarietà*

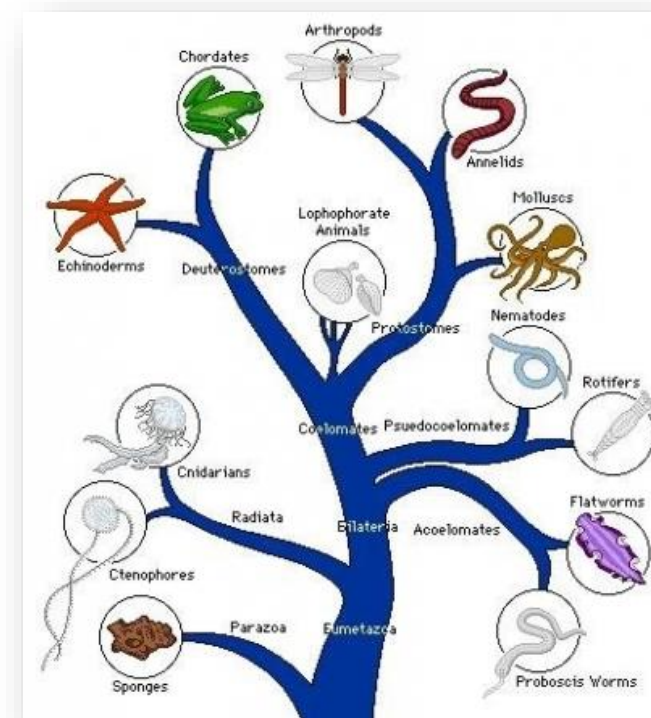
- l'ereditarietà permette di **definire nuove classi** partendo da classi sviluppate in precedenza
- la nuova classe viene definita esprimendo solamente le **differenze** che essa possiede rispetto alla classe di partenza
- l'ereditarietà permette di specificare “il punto di partenza”, cioè la **classe base**, e le **differenze** rispetto a questa

*un esempio di ereditarietà*



## relazione is-a

- relazione **is-a**
  - **classificazione**, es. in biologia
    - vertebrati **sottoclasse** di animali
    - mammiferi **sottoclasse** di vertebrati
    - felini **sottoclasse** di mammiferi
    - gatti **sottoclasse** di felini
- relazione **is-a tra classi**: ogni sottoclasse...
  - **eredita** le caratteristiche della classe base
  - ma introduce delle **specializzazioni**



*esempio: fattoria parlante*

- definiremo una **classe base** come **interfaccia astratta**
  - es. Animale:
- tutti gli animali fanno un verso (**interfaccia**)
- ogni animale fa un verso diverso (**polimorfismo**)

```
class Animale:  
    def parla(self):  
        raise NotImplementedError("metodo astratto")
```

## *definizione di sottoclassi*

```
class Sottoclasse(Superclasse) :  
    ...
```

## *classi concrete*

```
class Cane(Animale):  
    def __init__(self, nome):  
        self._nome = nome  
    def parla(self):  
        print("sono", self._nome, ", un cane ... bau")  
  
class Gatto(Animale):  
    def __init__(self, nome):  
        self._nome = nome  
    def parla(self):  
        print("sono", self._nome, ", un gatto ... miao")
```



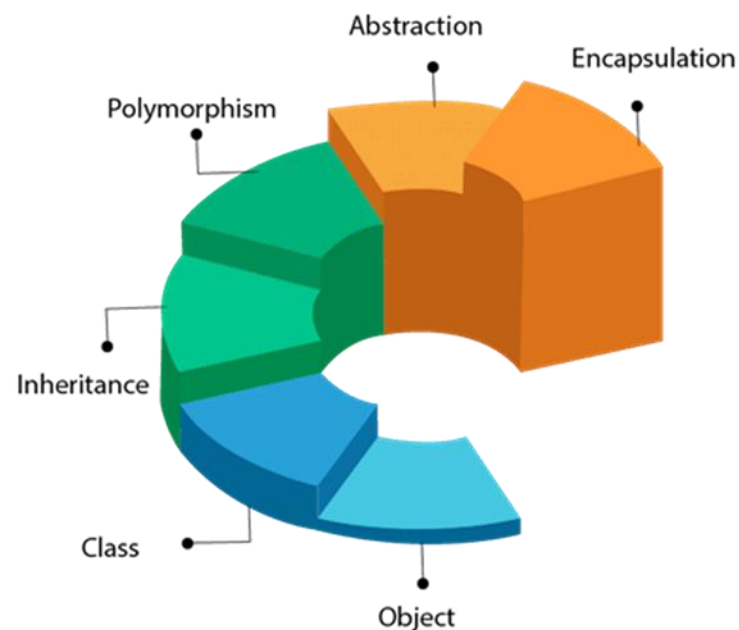
## *sottoclasse e superclasse*

- la nuova classe (nell'esempio Cane o Gatto) viene definita **sottoclasse** (o **classe derivata**)
- la classe di provenienza (nell'esempio Animale) viene definita **superclasse** (o **classe base**)
- la sottoclasse ***eredita*** tutte le caratteristiche (attributi e metodi) della superclasse e si differenzia da questa:
  - per l'***aggiunta*** di nuovi attributi e/o metodi
  - per la ***ridefinizione*** di alcuni metodi della superclasse



## *polimorfismo*

- «*In generale, l'assumere forme, aspetti, modi di essere diversi secondo le varie circostanze*»  
(Treccani)
- il ***polimorfismo*** è un concetto molto importante nella programmazione
  - permette di utilizzare un singolo tipo di entità (metodo, operatore, oggetto) per rappresentare diversi tipi in scenari differenti



## *esempio: polimorfismo dell'operatore di addizione*

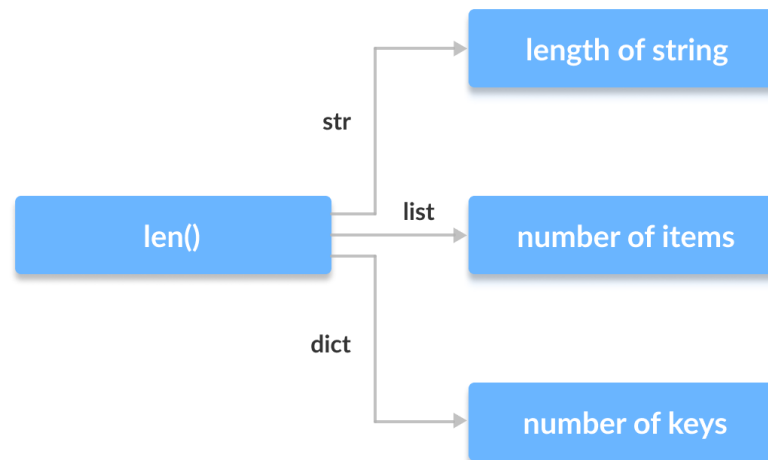
```
a = 3
b = 5
c = a + b
print(c)      # 8

a = 'intelligenza '
b = 'artificiale'
c = a + b
print(c)      # 'intelligenza artificiale'

a = [1,3,5]
b = [2,4,6]
c = a + b
print(c)      # [1,3,5,2,4,6]
```

## *esempio: polimorfismo della funzione len()*

```
print(len("Linguaggi"))           # 9
print(len(["Python", "Java", "C"])) # 3
print(len({"Nome": "Giuseppe", "Cognome": "Verdi"})) # 2
```



## *esempio: polimorfismo nelle classi*

```
class Animale:
    def parla(self):
        raise NotImplementedError("metodo astratto")

class Cane(Animale):
    def __init__(self, nome):
        self._nome = nome
    def parla(self):
        print("sono", self._nome, ", un cane ... bau")

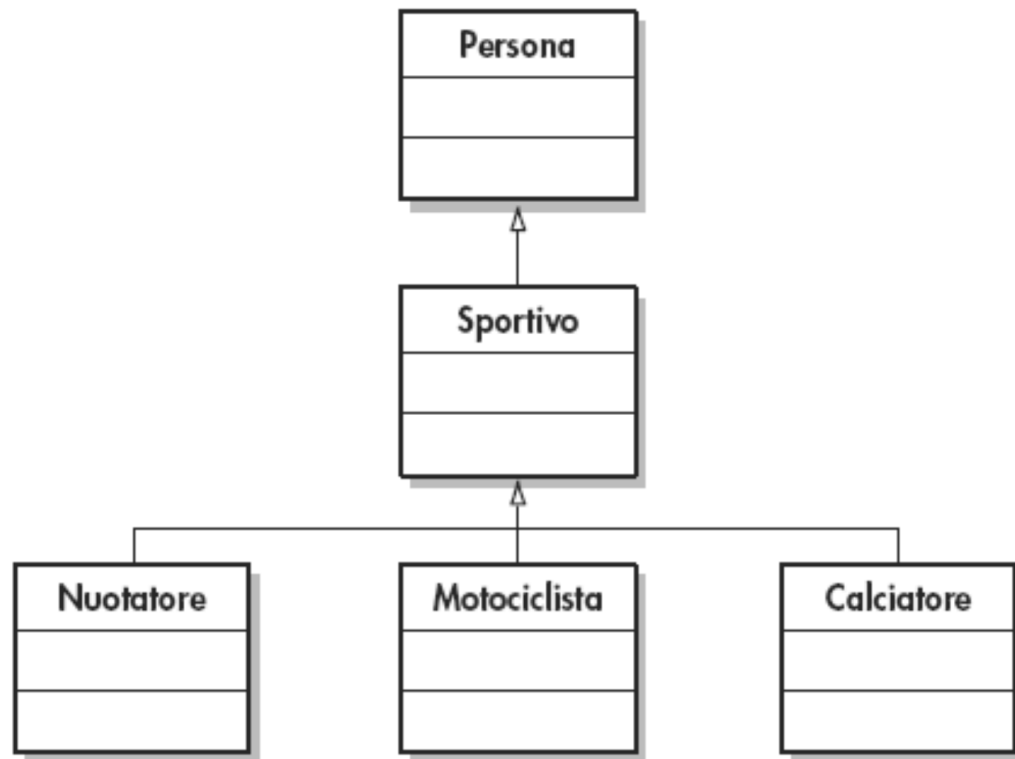
class Gatto(Animale):
    def __init__(self, nome):
        self._nome = nome
    def parla(self):
        print("sono", self._nome, ", un gatto ... miao")
```

```
# lista di animali
c = Cane("Belva")
g = Gatto("Fufi")

animali = [c, g]

for a in animali:
    a.parla()
```

## *gerarchia di classi*



- l'ereditarietà può estendersi a più livelli generando una **gerarchia di classi**
- una classe derivata può, a sua volta, essere base di nuove sottoclassi
- nell'esempio: Sportivo è sottoclasse di Persona ed è superclasse di Nuotatore, Motociclista e Calciatore
- nella parte alta della gerarchia troviamo le **classi generiche**, scendendo aumenta il **livello di specializzazione**

## *un esempio*

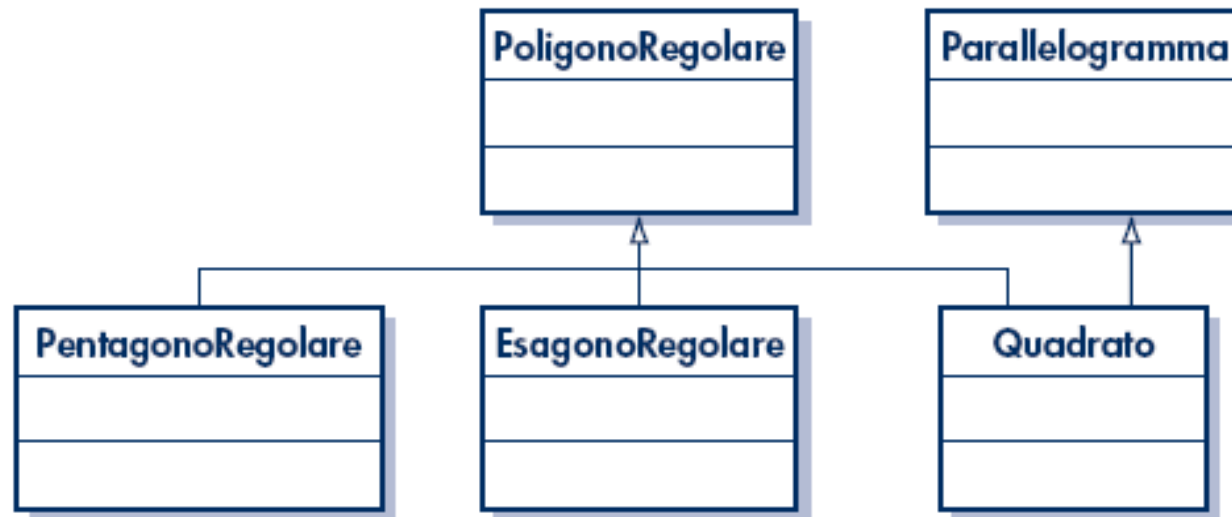
```
class Persona:  
    ...  
class Sportivo(Persona):  
    ...  
class Nuotatore(Sportivo):  
    ...  
class Motociclista(Sportivo):  
    ...  
class Calciatore(Sportivo):  
    ...
```

## *ereditarietà singola e multipla*

- sono possibili due tipi di ereditarietà:
  - ereditarietà singola
  - ereditarietà multipla
- **l'ereditarietà singola** impone ad una sottoclasse di derivare da **una sola superclasse**
- l'esempio presentato precedentemente è un caso di ereditarietà singola: ogni sottoclasse ha una sola classe base, mentre è possibile da una superclasse avere più classi derivate
- vari linguaggi ad oggetti pongono il vincolo dell'ereditarietà singola (Java) per problemi di chiarezza e semplicità d'implementazione (in Python è ammessa l'ereditarietà multipla)

## *ereditarietà multipla*

- l'ereditarietà multipla si ha quando una sottoclasse deriva da ***più*** superclassi
- la classe **Quadrato** ha due superclassi: **PoligonoRegolare** e **Parallelogramma**





## ***estensione***

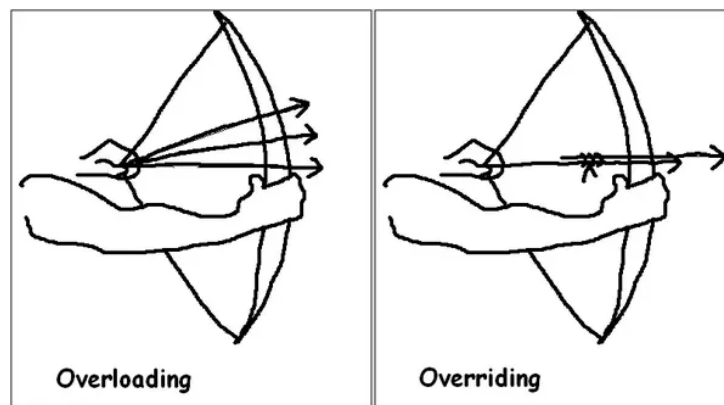
- una classe derivata può differenziarsi dalla classe base aggiungendo nuove caratteristiche:
  - nuovi attributi
  - e/o nuovi metodi
- in questo caso si parla di ***estensione***

## ***ridefinizione***

- la classe derivata potrebbe però fornire le stesse caratteristiche della classe base differenziandosi invece per il **comportamento**
- si definisce **ereditarietà per ridefinizione (overriding)** la situazione in cui uno o più **metodi** della classe base siano **ridefiniti** nella classe derivata
- i metodi avranno quindi la **stessa firma** (nome e lista di tipi dei parametri) ma **differenti corpo**

## *overriding e overloading*

- attenzione a non confondere
- il **sovraccarico** dei metodi (**overloading**)  
situazione in cui oltre al corpo del metodo è differente anche la sua firma
- con la **ridefinizione** (**overriding**)  
situazione in cui la firma del metodo è identica ma è differente il corpo



## *vantaggi dell'ereditarietà*

- l'ereditarietà facilita il **riutilizzo** di software estendendone o ridefinendone caratteristiche e comportamenti; è possibile adattare una classe preesistente alle nuove esigenze
- specificare le differenze da una classe simile piuttosto che ridefinire completamente la classe facilita enormemente lo sviluppo di nuovi progetti poiché **elimina ridondanza di codice**
- l'ereditarietà non è un meccanismo di inclusione del codice di una classe base in una derivata.
  - **non c'è copia di codice**, ogni modifica della struttura di una classe base si ripercuote automaticamente nelle sue classi derivate

## *l'interfaccia verso il mondo esterno*

- *l'interfaccia* di un oggetto è l'insieme delle firme dei suoi metodi
- se non si vuole dare accesso diretto agli attributi, e se si vuole nascondere l'implementazione di una classe, l'unica cosa che deve conoscere chi utilizza la nostra classe, è l'interfaccia
- conoscere l'interfaccia significa sapere quali sono le operazioni (metodi) che si possono invocare su un oggetto