

06. OOP in Python

Corso di Algoritmi e Linguaggi di Programmazione Python/C

Outline

- Classi in Python
- Modificatori di accesso
- Metodi
- Metodi di classe
- Metodi statici
- Metodi astratti
- Proprietà

Classi in Python

- Le classi in Python si dichiarano usando la parola chiave **class**:

```
class NomeClasse(ClasseBase):  
    # Attributi e metodi di classe...
```

- Python non prevede un costruttore, ma un metodo `__init__` per inizializzare i valori degli attributi:

```
class NomeClasse(ClasseBase):  
    def __init__(self, *args, **kwargs):  
        # ...  
        self.arg_1 = arg_1  
        # ...
```

Classi in Python

- I parametri **args** e **kwargs** rappresentano il concetto di **unpacking** (traducibile maccaronicamente con 'spacchettamento' di una lista e di un dizionario, rispettivamente).
- Ad esempio, possiamo creare la classe **Persona**:

```
class Persona(object):  
    def __init__(self, nome, cognome, eta=18):  
        self.nome = nome  
        self._cognome = cognome  
        self.__eta = eta
```

Modificatori di accesso

- Gli underscore prima del nome di un attributo vanno a definire (per convenzione) un modificatore di accesso **protected** o **private**.
- Permettono di perseguire il principio dell'**incapsulamento**.
 - In pratica, non ci interessa il come si definiscono il cognome e l'età di una persona, ma soltanto gli attributi in quanto tali!

```
self.nome = nome           # Membro "public"  
self._cognome = cognome    # Membro "protected"  
self.__eta = eta           # Membro "private"
```

Metodi

- Sono sintatticamente quasi identici alle classiche funzioni Python.
- Accettano come primo parametro la parola chiave **self**, che indica che si riferiscono all'istanza attuale della classe.

```
def metodo(self, *args, **kwargs):  
    pass
```

- Il riferimento a **self** non va indicato quando si chiama il metodo dall'esterno della classe, ma ci si limita a richiamarlo dall'istanza stessa.

```
p = Persona(nome, cognome) # p è un'istanza di Persona  
p.metodo(parametro)        # richiamo un metodo parametrizzato  
                             # sulla specifica istanza
```

Metodi di classe

- I **metodi di classe** sono contraddistinti dal *decorator* `@classmethod`.
- Sono metodi richiamabili sull'intera classe.
- Non hanno il riferimento all'istanza (`self`), ma all'intera classe (`cls`).
- Sono utilizzati per definire particolari tipi di costruttori/builder.

```
@classmethod
def builder_stringa(cls, stringa: str):
    nome, cognome, eta = stringa.split(' ')
    return Persona(nome, cognome, eta)
```

Metodi statici

- I **metodi statici** sono contraddistinti dal *decorator* `@staticmethod`.
- Sono metodi che non si applicano solo su un'istanza o sulla classe, ma risultano essere di applicabilità più generica.
- Di solito, sono all'interno di una classe per coerenza logica e funzionale.

```
@staticmethod
def nome_valido(nome):
    if len(nome) < 2:
        return False
    else:
        return True
```


Metodi astratti

- L'ereditarietà si ottiene **specializzando** una classe madre.
- Ad esempio, uno studente è un caso **speciale** di persona, ma il contrario non è vero.
- Si dice che lo studente **eredita** attributi e metodi della persona, definendone di altri.
- Le classi ed i metodi **astratti** sono dei 'prototipi' di metodi, definiti in una classe madre, che saranno implementati nelle classi figlie.
- Sono contraddistinti dalla parola chiave **ABC** e dal decorator **@abstractmethod**.
- *Vedremo un esempio tra gli esercizi.*

Proprietà

- Python offre un modo alternativo per implementare gli attributi di una classe.
- In particolare, usando le **proprietà**, contraddistinte dal *decorator* **@property**, possiamo fare in modo da definire metodi **impliciti** di accesso e modifica dei dati.
- Risultano essere estremamente efficaci nel caso di implementazioni complesse.

Esercizio

- **Esercizio 1:** scrivere una classe **Persona** utilizzando i concetti visti in precedenza.
- **Esercizio 2:** creiamo due classi. La prima è la classe **Quadrato**, che modella tutti i quadrati, la seconda è la classe **Cerchio**, che modella tutti i cerchi.

Domande?

42