Mètodes màgics en Python

Els noms que comencen i acaben amb doble subratllat (`__`) en Python tenen un significat especial i són coneguts com a “dunder methods” o mètodes màgics. Aquests noms amb doble subratllat estan associats a funcions i comportaments específics de Python que s'executen de manera automàtica o interna.

1. Mètodes Màgics (Dunder Methods):

- Els mètodes màgics són funcions especials que tenen el seu nom envoltat per dobles subratllats (`__nom__`). Aquests mètodes defineixen comportaments especials de les classes en Python i solen ser cridats automàticament pel propi llenguatge en certs contextos.

- Exemple clàssic és el mètode `__init__()`, que s'executa quan es crea una nova instància d'una classe.

Exemples de mètodes màgics:

__init__(self): Constructor d'una classe. Es crida quan s'instancia un objecte.

__str__(self): Defineix la representació de l'objecte com a cadena. Es crida quan fas servir `print()` amb l'objecte.

__repr__(self): Retorna una representació de l'objecte per a depuració.

__len__(self): Defineix el comportament quan crides `len()` sobre un objecte.

Exemple:

```
class Persona:
    def __init__(self, nom, edat):
        self.nom = nom
        self.edat = edat

    def __str__(self):
        return f"Persona: {self.nom}, {self.edat} anys"

p = Persona("Anna", 30)
print(p)  # Això crida el mètode __str__ automàticament
```

Sortida:

Persona: Anna, 30 anys

2. Noms de Variables Privades "Mangled" (Amb un Subratllat al Principi):

- Els noms que comencen amb doble subratllat (però sense acabar amb doble subratllat, per exemple `__variable`) tenen un comportament especial conegut com a “name mangling”. Això es fa per evitar col·lisions de noms en subclasses.

- Aquestes variables són essencialment "privades" dins de la classe, i el seu nom es modifica internament afegint el nom de la classe.

Exemple:

```
class Exemple:
    def __init__(self):
        self.__privat = "Això és privat"

    def mostrar(self):
        print(self.__privat)

e = Exemple()
e.mostrar()
# Això no funcionarà: print(e.__privat) perquè el nom ha estat modificat
```

Tot i que `__privat` sembla "privat", el nom real de l'atribut es converteix en `_Exemple__privat`, fent-lo inaccessible directament.

3. Dunder Methods per Sobrecàrrega d'Operadors:

- Python permet sobrecarregar operadors com `+`, `-`, `*`, etc., mitjançant mètodes màgics amb doble subratllat. Per exemple, el mètode `__add__` permet definir el comportament de l'operador `+`.

Exemple de sobrecàrrega d'operadors:

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

    def __add__(self, altre):
        return Vector(self.x + altre.x, self.y + altre.y)

    def __str__(self):
        return f"Vector({self.x}, {self.y})"

v1 = Vector(1, 2)
v2 = Vector(3, 4)
v3 = v1 + v2  # Això crida __add__ automàticament
print(v3)  # Sortida: Vector(4, 6)
```

Quan i Per Què S'Utilitzen?

- Mètodes Màgics (`__init__`, `__str__`, etc.): Són essencials per definir el comportament intern de les classes i objectes en Python.

- Privadesa amb `__variable: Utilitzat per evitar col·lisions de noms en subclasses. No és veritablement privat, però es fa servir per evitar accés accidental a atributs sensibles.

- Sobrecàrrega d'operadors (`__add__`, `__sub__`, etc.): Permet que les classes personalitzades es comportin com els tipus integrats en Python.

En general, no es recomana que els programadors utilitzin noms amb dobles subratllats al principi i al final (com `__nom__`) fora dels casos específics que defineix Python, com els mètodes màgics. Aquests noms estan reservats per al llenguatge i per a l'ús intern de Python, així que utilitzar noms personalitzats amb aquest format podria causar confusió o problemes inesperats.

Per què no s'haurien de fer servir aquests noms en altres casos?

1. Confusió amb els mètodes màgics

- Els noms amb doble subratllat tenen una connotació molt clara: són mètodes especials o "màgics" definits pel llenguatge Python. Si els programadors els utilitzen per a altres propòsits, podria ser confús per a altres desenvolupadors que revisin el codi, ja que podrien pensar que estan treballant amb un comportament intern del llenguatge.

2. Evitació de conflictes

- Python reserva els noms amb doble subratllat al principi i al final per a les seves pròpies funcions especials. Si un programador defineix mètodes o variables amb aquests noms, hi ha risc de conflicte amb les funcionalitats internes de Python.

Què poden utilitzar els programadors?

- Subratllat simple (`_`)

- Els noms que comencen amb un sol subratllat (`_nom`) són una convenció que indica que una variable o mètode és "privat" i no hauria de ser accedit fora de la classe o mòdul. Aquest és l'ús correcte per indicar privadesa o ocultar detalls d'implementació.

Exemple

```
class Exemple:
    def __init__(self):
        self._atribut_intern = "Privat"
```

- Doble subratllat al principi (`__nom`) sense el subratllat al final

- Aquesta convenció s'utilitza per evitar conflictes de noms en subclasses mitjançant un procés conegut com :name mangling: (canvi del nom intern). Això no s'ha de confondre amb els noms com `__init__`.

Exemple

```
class Exemple:
    def __atribut_privadet(self):
        print("Això és intern")
```

Aquests noms es transformen internament per incloure el nom de la classe, evitant que es sobrescriguin fàcilment en subclasses.

Conclusió

- Noms amb doble subratllat al principi i al final (`__nom__`) no haurien de ser utilitzats fora dels mètodes màgics o funcionalitats reservades per Python.

- És millor utilitzar el subratllat simple (`_`) per indicar que una variable o mètode és privat o intern.

- El doble subratllat només al principi (`__nom`) s'ha d'utilitzar en casos específics per evitar conflictes de noms en subclasses.

D'aquesta manera, el codi serà més clar i es mantindrà alineat amb les convencions de Python, evitant possibles problemes.