Let's go pour un concept très **libre, souple et 100% Pythonic** :

**🦆 Concept #15 — Duck Typing**

*"If it walks like a duck and quacks like a duck, it’s a duck."*

**🧠 1. Définition**

En Python, **on ne se soucie pas du type exact d’un objet**, tant qu’il **se comporte** comme on veut.

🟢 Ce qui compte : **le comportement**, **pas la classe d’origine**.

**📘 Exemple simple :**

python

CopyEdit

class Canard:

def parler(self):

print("Coin coin")

class Personne:

def parler(self):

print("Bonjour !")

def faire\_parler(objet):

objet.parler() # Peu importe que ce soit un Canard ou une Personne

faire\_parler(Canard())

faire\_parler(Personne())

✅ Résultat :

nginx

CopyEdit

Coin coin

Bonjour !

**➕ Aucun isinstance(), aucun héritage nécessaire.**

Juste : "A-t-il une méthode .parler() ?"

**🎯 Avantages du Duck Typing :**

* ✨ **Flexible**
* 🔧 Pas besoin d’une structure complexe
* 💡 Encourage le **code simple et lisible**
* 🧪 Idéal pour le **polymorphisme implicite**