Principy objektového programování, agregace a kompozice objektů

Objektové programování
Je to způsob psaní programů, který se soustředí na objekty - části kódu, které spojují atributy a metody. Dřívější postupy (např procedurální programování) se zaměřovaly na posloupnost instrukcí a oddělené funkce, OOP nám nabízí způsob hezkého uspořádaného kódu podle toho, jak vnímáme svět. Soubor objektů, které mají vlastnosti a mohou spolu komunikovat, je to přirozenější pro nás pro lidi a docela to pomáhá řešit problémy.

+ rozdělení kódu
+ opakovatelné využití
+ jednoduché úpravy (měnění/rozšiřování)
+ přehlednost

- většinou pomalejší
- vyšší znalost jazyka

Hlavní pojmy
Třída - návod/plán pro vytváření objektů, určuje jaké vlastnosti a funkce bude objekt mít
Objekt - instance třídy, obsahuje konkrétní hodnoty a vykonává funkce
Atributy - vlastnosti uvnitř objektu
Metody - funkce definované v třídě, říkají co vlastně objekt má dělat
Konstruktor - speciální funkce, spouští se při vytvoření objektu, nastavuje počáteční hodnoty vlastností

4 základní pojmy OOP
Zapouzdření - skrýváme vnitřní detaily objektu, řízený přístup k datům pomocí metod, chráníme tím data před nechtěnými změnami
public, private, protected
V Pythonu
veřejný přístup - přístupné odkudkoli (self.vlastnost)
chráněný (protected) přístup - naznačují, že by měly být použity jen uvnitř třídy (self.vlastnost)
privátní přístup - skryté před přístupem zvenčí (self __ vlastnost)

Dědičnost - způsob jak jedna třída může dědit vlastnosti a funkce od jiné třídy, nadtřídy a podtřídy, podtřídy mohou rozšiřovat nebo upravovat zděděné vlastnosti a chování, šetří nám spoustu kódu atd…
V pythonu máme možnost vícenásobné dědičnosti, narozdíl třeba od C# kde můžeme dědit jenom jednou

Polymorfismus - Polymorfismus umožňuje používat jednotné rozhraní pro práci s různými typy objektů, hodně souvisí s dědičností

```python
class Zvire:
    def vydej_zvuk(self):
        pass


class Pes(Zvire):  # Dědí od Zvire
    def vydej_zvuk(self):
        return "Haf haf!"


class Kocka(Zvire):  # Dědí od Zvire
    def vydej_zvuk(self):
        return "Mňau!"


# Tady je polymorfismus - stejná funkce pracuje s různými typy
def prehraj_zvuk(zvire):
    print(zvire.vydej_zvuk())


pes = Pes()
kocka = Kocka()
prehraj_zvuk(pes)  # Vypíše: Haf haf!
prehraj_zvuk(kocka)  # Vypíše: Mňau!
```

Abstrakce - skrývá složitost implementace a poskytuje jednoduché rozhraní, zaměřuje se spíše na to co objekt dělá než jak to dělá, inteface/abstraktní třídy/abstraktní metody

Příklad ze života: Auto - řidič ho ovládá přes jasné rozhraní (volant, pedály), ale nemusí rozumět tomu, jak funguje motor.

Agregace
Vztah mezi objekty, který vyjadřuje, že jeden objekt obsahuje reference na jiné objekty, přičemž tyto části mohou existovat nezávisle na celku. Používá se tam, kde chcem flexibilní struktury, které se mohou dynamicky měnit

Například vztah mezi počítačem a tiskárnou je vztah typu agregace, kdy počítač s tiskárnou tvoří jeden celek, ale tiskárna může existovat i tehdy, pokud není k žádnému počítači připojena

Kompozice
Silnější forma vztahu mezi objekty, kde je jeden objekt (celek) je složen z jiných objektů a tyto části nemohou existovat bez celku. Je vhodná pro situace, kdy části objektu tvoří jeho podstatu a nemají smysl mimo něj

Příkladem kompozice je vztah mezi fakturou a jejími položkami. Každá položka musí být součástí právě jedné faktury, faktura má jednu a více položek. Jestliže fakturu zahodíme, nezbudou nám po ni ani žádné položky.