**OOP a dědičnost – podrobný výklad**

a) **Co je to OOP (Objektově orientované programování)**

* **OOP je způsob psaní programů**, který je založen na **modelech reálného světa**. V reálném světě máme objekty – třeba "auto", "člověk", "kniha". V OOP pracujeme se **softwarovými objekty**, které mají **vlastnosti (atributy)** a **chování (metody)**.
* Příklad:
  + Objekt **Auto**:
    - Atributy: barva, značka, rychlost.
    - Metody: jezdi(), zastav().

b) **Základní principy OOP (vysvětlit každou vlastnost s příkladem)**

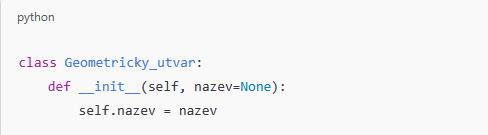
1. **Třída (class):**
   1. **Šablona, předloha pro objekt.**
   2. Např.: class Auto:, kde definujeme, jaké vlastnosti a metody auto má.
2. **Objekt (instance):**
   1. **Konkrétní auto** vytvořené podle třídy.
   2. Např.: moje\_auto = Auto()
3. **Zapouzdření (Encapsulation):**
   1. **Schováváme vnitřní stav objektu** a přistupujeme k němu přes metody.
   2. Zamezuje neoprávněné manipulaci.
   3. Např.: atributy s podtržítkem \_rychlost, metody get\_rychlost().
4. **Dědičnost (Inheritance):**
   1. **Potomek přebírá vlastnosti a chování rodiče.**
   2. Např.: class ElektrickeAuto(Auto): – dědí vše z Auto, ale přidá si třeba "baterie".
5. **Polymorfismus:**
   1. Stejná metoda se chová různě podle třídy.
   2. Např.: metoda zvuk() u Pes a Kočka.

c) **Proč OOP využíváme v praxi?**

* **Přehlednost a modularita** – rozdělení na logické celky.
* **Znovupoužitelnost kódu** – jednou napíšu třídu a znovu používám.
* **Údržba a rozšiřitelnost** – snadná změna/rozšíření.
* **Bezpečnost (zapouzdření)** – omezení přístupu k citlivým datům.

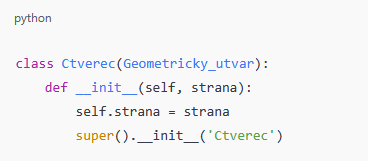
d) **Dědičnost na konkrétním příkladu (Geometrický útvar → Čtverec)**

1. **Geometrický útvar – obecná třída:**



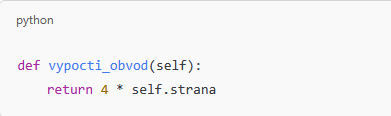
* Obecná třída, o útvaru víme jen, jak se jmenuje.
* Nemá konkrétní výpočet obvodu/obsahu.

1. **Čtverec dědí Geometrický útvar:**



* Dědí atribut nazev, přidává strana.
* **super()** volá rodiče a nastaví název "Čtverec".

1. **Přepsání metod (polymorfismus):**



* Specifické pro čtverec, jiné než např. u kruhu.

1. **Další třídy (Obdélník, Kruh) dědí Geometricky\_utvar** – ukázka polymorfismu.

e) **Jak by vypadala práce bez dědičnosti?**

* Museli bychom **znovu psát výpočty pro každý útvar**, opakování kódu.
* S dědičností **zdědíme metody, atributy**, měníme jen potřebné části.

**2. Linux, terminál, shell – podrobný výklad**

a) **Co je to terminál?**

* **Historicky:** fyzické zařízení (obrazovka, klávesnice) pro komunikaci s UNIX.
* **Dnes:** Software (emulátor terminálu), např. **GNOME Terminal**, **Konsole**.

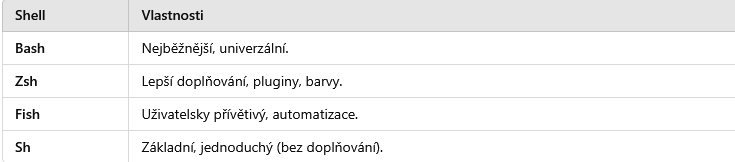
b) **Proč emulátor terminálu v Linuxu?**

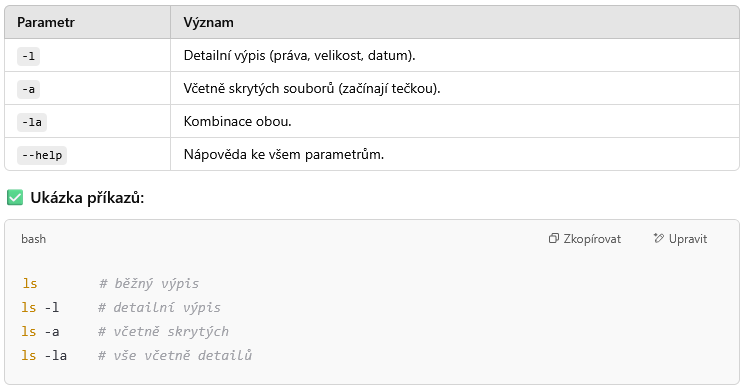
* **Ovládání systému pomocí příkazů.**
* Automatizace (skripty).
* Přístup k hlubokým částem systému.

c) **Shell – co to je?**

* **Program, co zpracovává příkazy.**
* Komunikuje mezi uživatelem a jádrem OS.
* Např.: bash, zsh, fish.

d) **Typy shellů a rozdíly**

* e) **Příkaz ls a parametry – detailně**
* **ls** – výpis obsahu adresáře.
* **Parametry:**

**3. SQL databáze – SQLite vs. MySQL**

a) **Co je SQL?**

* **Structured Query Language** – jazyk pro komunikaci s databázemi.
* Umožňuje **vytvářet, upravovat a mazat** data.

b) **SQLite – podrobně**

* **Souborová databáze** (např. data.db).
* **Jednoduchá, lehká, nevyžaduje server**.
* Vhodná pro:
  + **Mobilní aplikace.**
  + **Malé weby, testování.**

**Výhody:**

* Rychlá, jednoduchá, snadno přenositelná (1 soubor).

**Nevýhody:**

* **Nevhodná pro více uživatelů současně.**
* Omezené funkce.

c) **MySQL – podrobně**

* **Relační databáze na serveru.**
* Umožňuje:
  + Více uživatelů.
  + Vysokou bezpečnost a zálohy.
* Vhodná pro:
  + **Webové aplikace (e-shopy, portály).**
  + **Velké systémy.**

**Výhody:**

* Stabilní, výkonná, bezpečná.

**Nevýhody:**

* Nutnost serveru.
* Složitější na správu.

