**Popište OOP obecně a zdůvodněte proč tento koncept využíváme v programování. Popište základní charakteristiky OOP. Na příkladu objektů Geometricky\_utvar -> Ctverec (Viz příloha 2) vysvětlete pojem dědičnost aplikovanou v jazyku Python.**

### **Objektově orientované programování (OOP)**

OOP je programovací paradigma, které využívá koncept **objektů** – tedy entit, které mají **vlastnosti** (atributy) a **chování** (metody). Tento přístup přináší lepší strukturu kódu, opakovanou použitelnost a usnadňuje údržbu aplikací.

#### **Základní charakteristiky OOP**

1. **Zapouzdření (Encapsulation)** – Skrytí interních detailů objektu a jejich zpřístupnění pouze prostřednictvím metod.
2. **Dědičnost (Inheritance)** – Možnost vytvářet nové třídy na základě již existujících tříd, což podporuje opětovné použití kódu.
3. **Polymorfismus (Polymorphism)** – Možnost používat stejnou metodu pro různé typy objektů.
4. **Abstrakce (Abstraction)** – Skrytí nepotřebných detailů a zpřístupnění pouze podstatných funkcí.

### **Dědičnost na příkladu tříd v Pythonu**

Dědičnost umožňuje odvozovat nové třídy z existujících a **přejímat jejich vlastnosti**. V příkladu máme **nadřazenou třídu Geometricky\_utvar** a z ní odvozené třídy **Čtverec, Obdélník a Kruh**.

import math  
class Geometricky\_utvar:  
 def \_\_init\_\_(self, nazev=None):  
 self.nazev = nazev  
 self.obvod = self.vypocti\_obvod()  
 self.obsah = self.vypocti\_obsah()  
 self.params = self.params()  
   
 def vypocti\_obvod(self):  
 return None  
   
 def vypocti\_obsah(self):  
 return None  
   
 def params(self):  
 return None

Tato třída slouží jako **základní šablona** pro geometrické útvary. Definuje tři metody, které budou přepsány v potomcích:

* **vypocti\_obvod()** – vrací None, protože v této obecné třídě neznáme konkrétní vzorec.
* **vypocti\_obsah()** – obdobně vrací None.
* **params()** – slouží k uložení parametrů tvaru.

Třída **Ctverec** dědí od **Geometricky\_utvar** a definuje konkrétní výpočty pro obvod a obsah:

class Ctverec(Geometricky\_utvar):  
 def \_\_init\_\_(self, strana):  
 self.strana = strana  
 super().\_\_init\_\_('Ctverec')  
  
 def vypocti\_obvod(self):  
 return 4 \* self.strana  
  
 def vypocti\_obsah(self):  
 return self.strana \*\* 2  
  
 def params(self):  
 return 'a = ' + str(self.strana)

**Klíčové prvky:**

* **super().init('Ctverec')** – Volá konstruktor nadřazené třídy a nastavuje název objektu.
* **vypocti\_obvod a vypocti\_obsah** – Přepisují metody z nadřazené třídy a implementují konkrétní výpočty.

Obdobně fungují i třídy **Obdelnik** a **Kruh**:

class Obdelnik(Geometricky\_utvar):  
 def \_\_init\_\_(self, strana\_a, strana\_b):  
 self.strana\_a = strana\_a  
 self.strana\_b = strana\_b  
 super().\_\_init\_\_('Obdelnik')  
  
 def vypocti\_obvod(self):  
 return 2 \* self.strana\_a + 2 \* self.strana\_b

def vypocti\_obsah(self):  
 return self.strana\_a \* self.strana\_b  
  
 def params(self):  
 return 'a = ' + str(self.strana\_a) + ', b = ' + str(self.strana\_b)  
 class Kruh(Geometricky\_utvar):  
 def \_\_init\_\_(self, polomer):  
 self.polomer = polomer  
 super().\_\_init\_\_('Kruh')  
  
 def vypocti\_obvod(self):  
 return 2 \* math.pi \* self.polomer  
  
 def vypocti\_obsah(self):  
 return math.pi \* (self.polomer \*\* 2)  
  
 def params(self):  
 return 'r = ' + str(self.polomer)

Program lze spustit následovně:

def main():  
 utvary = [Ctverec(5), Kruh(4), Obdelnik(5,2)]  
 for u in utvary:  
 print(u.nazev, 's parametry', u.params, 'ma obvod', u.obvod, 'a obsah', u.obsah, '.')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()