LABORATORIUM 3

TREŚCI KSZTAŁCENIA:

- PROGRAMOWANIE OBIEKTOWE: DZIEDZICZENIE I HERMETYZACJA WE PYTHONIE

Spis treści

Programowanie obiektowe

Programowanie obiektowe (OOP - Object-Oriented Programming) jest paradygmatem programowania, który skupia się na obiektach i ich wzajemnych relacjach. W Pythonie OOP jest szeroko stosowane do modelowania rzeczywistych problemów i strukturyzacji kodu w sposób czytelny i zorganizowany.

Podstawowe Koncepcje Programowania Obiektowego w Pythonie

1. Klasa i Obiekt
   - Klasa: Szablon definiujący strukturę danych i zachowanie. Klasy są jak plany, które określają, jak powinny wyglądać obiekty.
   - Obiekt: Instancja klasy, czyli konkretny egzemplarz utworzony na podstawie klasy.
2. Atrybuty i Metody
   - Atrybuty: Zmienne zdefiniowane w klasie, które przechowują dane związane z obiektem.
   - Metody: Funkcje zdefiniowane w klasie, które opisują zachowanie obiektu. Metody mogą manipulować atrybutami klasy i wykonywać inne operacje.
3. Konstruktor (__init__())
   - Specjalna metoda wywoływana automatycznie przy tworzeniu nowego obiektu. Używana do inicjalizacji atrybutów obiektu.
4. Dziedziczenie
   - Mechanizm pozwalający na tworzenie nowych klas na podstawie istniejących, co ułatwia ponowne wykorzystanie kodu. Dziedziczenie pozwala na rozszerzanie funkcjonalności bez modyfikacji oryginalnej klasy.
5. Polimorfizm
   - Zdolność obiektów różnych klas do używania tych samych metod w różny sposób. Pozwala to na pisanie elastycznego i skalowalnego kodu.
6. Hermetyzacja (Enkapsulacja)
   - Ukrywanie szczegółów implementacji i udostępnianie tylko niezbędnego interfejsu. To podejście chroni dane i ułatwia utrzymanie kodu.
7. Abstrakcja
   - Upraszczanie złożonych systemów poprzez modelowanie ich w formie uproszczonych reprezentacji (klas), które skupiają się na kluczowych aspektach problemu.

Przykład Klasy i Obiektu

Poniżej przedstawiono prosty przykład klasy Car, która modeluje samochód:

```
class Car:
    # Konstruktor - definiowanie atrybutów
    def __init__(self, make, model, year):
        self.make = make       # Marka samochodu
        self.model = model     # Model samochodu
        self.year = year       # Rok produkcji
        self.odometer = 0      # Początkowy stan licznika

    # Metoda opisująca samochód
    def describe_car(self):
        print(f"{self.year} {self.make} {self.model}")

    # Metoda do aktualizacji licznika
    def update_odometer(self, mileage):
        if mileage >= self.odometer:
            self.odometer = mileage
        else:
```

```
            print("Nie można cofnąć licznika!")

    # Metoda do zwiększenia przebiegu
    def increment_odometer(self, miles):
        self.odometer += miles

# Tworzenie obiektu klasy Car
my_car = Car('Toyota', 'Corolla', 2020)
my_car.describe_car()  # Wywołanie metody
my_car.update_odometer(5000)  # Aktualizacja licznika
my_car.increment_odometer(200)
print(f"Przebieg: {my_car.odometer}")
```

Kluczowe Funkcjonalności OOP w Pythonie

1. Dziedziczenie - Rozszerzanie Funkcjonalności Klas.

Dziedziczenie pozwala na tworzenie nowych klas na podstawie istniejących. Przykład:

```
from Lab03.Car import Car

class ElectricCar(Car):  # ElectricCar dziedziczy po Car
    def __init__(self, make, model, year, battery_size=75):
        super().__init__(make, model, year)  # Wywołanie konstruktora klasy
nadrzędnej
        self.battery_size = battery_size

    def describe_battery(self):
        print(f"Ten samochód ma baterię o pojemności {self.battery_size} kWh.")

my_electric_car = ElectricCar('Tesla', 'Model S', 2022)
my_electric_car.describe_car()
my_electric_car.describe_battery()
```

2. Polimorfizm - Różne Implementacje Metod.

Dzięki polimorfizmowi można zdefiniować metody o tej samej nazwie w różnych klasach, co pozwala na elastyczne korzystanie z kodu.

```
class Dog:
    def make_sound(self):
        return "Woof!"

class Cat:
    def make_sound(self):
        return "Meow!"

animals = [Dog(), Cat()]

for animal in animals:
    print(animal.make_sound())  # Różne wyniki w zależności od typu obiektu
```

3. Hermetyzacja - Ukrywanie Danych

Atrybuty klasy mogą być chronione przed bezpośrednim dostępem, np. przez użycie podwójnego podkreślenia (__), które ukrywa zmienne:

```
class BankAccount:
    def __init__(self, balance):
        self.__balance = balance  # Ukryty atrybut

    def deposit(self, amount):
        self.__balance += amount

    def withdraw(self, amount):
```

```
        if amount <= self.__balance:
            self.__balance -= amount
        else:
            print("Niewystarczające środki")

    def get_balance(self):
        return self.__balance

account = BankAccount(1000)
account.deposit(500)
print(f"Stan konta: {account.get_balance()}")
```

Zalety Programowania Obiektowego

- Modularność: Kod jest zorganizowany w klasy i obiekty, co ułatwia jego utrzymanie i rozwijanie.
- Reużywalność: Możliwość ponownego wykorzystania kodu dzięki dziedziczeniu.
- Łatwość w zarządzaniu złożonością: Obiekty pozwalają na modelowanie rzeczywistości w sposób naturalny i czytelny.
- Zabezpieczenie danych: Hermetyzacja chroni dane przed nieuprawnionym dostępem.

Wady Programowania Obiektowego

- Złożoność: Projektowanie systemów OOP wymaga przemyślenia struktury klas i relacji między nimi.
- Wydajność: Operacje na obiektach mogą być mniej wydajne niż na strukturach prostszych, takich jak tablice czy słowniki.
- Przekombinowanie: W małych projektach może prowadzić do nadmiernej komplikacji kodu.

Programowanie obiektowe w Pythonie jest potężnym narzędziem, które umożliwia modelowanie złożonych systemów w przejrzysty i rozszerzalny sposób. Dzięki mechanizmom dziedziczenia, polimorfizmu, i hermetyzacji, OOP pozwala na tworzenie solidnych, elastycznych i łatwych do utrzymania aplikacji.

Zadania do samodzielnego rozwiązania

Rozwiązania w postaci niezbędnych plików źródłowych należy przesłać do utworzonego zadania na platformie e-learningowej zgodnie ze zdefiniowanymi instrukcjami oraz w nieprzekraczalnym wyznaczonym terminie.

Zadanie 1. Z wykorzystaniem OOP zaproponuj implementację Employees System Project.

Employees System Project ma być prostym projektem oparty na języku Python, który prezentuje wykorzystanie zasad programowania obiektowego (OOP). Obejmować ma trzy główne klasy: Employee, EmployeesManager i FrontendManager. System ma za zadanie zarządzać danymi i interakcją pracowników przy użyciu koncepcji OOP. Employees System Project powinien objemować trzy podstawowe klasy, z których każda służy odrębnemu celowi:

1. Employee - reprezentuje indywidualnego pracownika z następującymi atrybutami:
   - name: Imię i nazwisko pracownika.
   - age: Wiek pracownika.
   - salary: Wynagrodzenie pracownika.

Klasa ma udostępniać metody pozwalające na reprezentację informacji o pracownikach, które zostały wprowadzone jako dane wejściowe.

2. EmployeesManager – klasa która jest odpowiedzialna za zarządzanie listą pracowników. Posiada nastepujące funkcjonalności:
   - dodanie nowego pracownika do listy,
   - wyświetlenie listy wszystkich istniejących pracowników,
   - usunięcie pracowników w określonym przedziale wiekowym,
   - wyszukanie pracownika według jego imienia i nazwiska,
   - aktualizacja wynagrodzenia pracownika według jego imienia i nazwiska.
3. FrontendManager - klasa zapewnia interfejs użytkownika do interakcji z EmployeesManager. Użytkownicy mogą wykonywać akcje takie jak:
   - Dodawanie nowych pracowników.
   - Wyświetlenie listy istniejących pracowników.
   - Usuwanie pracowników na podstawie przedziału wiekowego.
   - Aktualizacja wynagrodzeń pracowników według nazwiska.

Zadanie 2. W oparciu o system z zadania 1 zmodyfikuj/rozszerz go o poniższe funkcjonalności:

- logowanie do systemu – poprzez konto admin, hasło admin,
- dane o pracownikach mają być przechowywane w plikach, format pliku dowolny txt, json, zewnętrzna baza,
- dane maja być zapisywane do pliku,
- wyświetlane dane mają być odczytywane z pliku,
- każde zmiany/modyfikacje mają być aktualizowane w plikach
- walidację danych.