PODSTAWY PROGRAMOWANIA W PYTHON

PO 12 ZAJĘCIACH:

1. Omawiane zagadnienia:
    a. dziedziczenie diamentowe:
    w Python dozwolone jest dziedziczenie z wielu klas. Musimy w takiej sytuacja pamiętać o kolejności wyszukiwania atrybutów – Python będzie je wyszukiwał w kolejności w jakiej były zdefiniowane w sygnaturze klasy. Jeśli chcemy użyć metodę/pole z klasy innej musimy się do niej odwołać.
    b. pola klasy (zmienne klasy) – zmienne umieszczone na poziomie klasy, ich zawartość jest widoczna przez wszystkie instancje. Najczęściej używamy je do trzymania tzw. stałych, lub domyślnych wartości. Przy ich **definiowaniu nie używamy** słowa self
    c. metody klasy – oznaczamy dekoratorem **@classmethod**, definiujemy jak metodę instancji, ale zamiast słowa self używamy **cls**. cls oznacza, że jako pierwszy argument, do metody jest przekazywana klasa.
    Metody te używamy w celu manipulowania polami klasy, lub jako alternatywne konstruktory. Jeśli korzystamy z nich jak z konstruktorów to musimy pamiętać o kolejności – tworzymy instancję, zmieniamy dane wg. argumentów i na końcu zwracamy gotowy obiekt.
    d. metody statyczne – używamy dekoratora **@staticmethod**– metody, które można użyć bez przekazywania klasy lub instancji. Metody te wykorzystujemy w sytuacji gdy jakaś funkcjonalność jest związana z naszym modułem, ale nie jest konieczne tworzenie instancji. Np. wyobraźmy sobie moduł zarządzania pracownikami, możemy mieć w nim metodę statyczną, która będzie sprawdzała, czy numer PESEL jest poprawny, lub w module płatności sprawdzamy czy numer karty jest poprawny – w tym celu nie musimy tworzyć instancji pracownika, ani instancji płatności.

e. pola i metody pseudo-prywatne – zmienne i metody klasy możemy ukrywać przed dostępem poza klasą. W tym celu nazwy metod i zmiennych zaczynamy od dwóch podkreślników, np.: **__imie**, **__sprawdz_numer()** – gdy tak zrobimy to Python zmieni i ukryje te nazwy – IDE nie podpowie nam ich. Pamiętajmy, że to jest tylko ukrycie – w łatwy sposób możemy sprawdzić jak te nazwy są zmienione – w tym celu możemy wyświetlić namespace (przestrzeń nazw – czyli wszystkie atrybuty do których instancja ma dostęp) – np.: **pracownik.__dict__**
   Python jest językiem otwartym, jak widzimy nawet pseudoprywatność nie ochroni naszych zmiennych – nie musimy się tym jednak przejmować – dobrą praktyką jest zawsze korzystać z udostępnionych przez programistę metod i właściwości do manipulowania obiektami.
f. właściwości – properties – są to metody, które definiujemy w celu kontrolowania dostępu do zmiennych – metody te „udają" zmienne – korzystamy z nich bez nawiasów.
   i. getter – dekorator: **@property** – służy do zwracania wartości, kontrolujemy co i jak użytkownik naszego kodu może odczytać
   ii. seter – dekorator **@nazwa_property.setter** – aby utworzyć seter musimy najpierw zdefiniować getter. Settery służą do walidacji i odpowiedniej obróbki przekazanych przez nie wartości, zanim je zapiszemy do zmiennej
   iii. deleter - **@nazwa_property.deleter** – służy do wyczyszczenia w zdefiniowany przez nas sposób property.

2. Poznaliśmy również metodę specjalną destruktor **__del__(self)** – ta metoda jest wykonywana w momencie niszczenia obiektu – przez użycie del.
   Musimy pamiętać, że zostanie ona wywołana również dla wszystkich obiektów w momencie zakończenia naszego programu! Zobacz plik z kodem pole_klasy.py
3. Zadanie domowe – umiemy już pisać kod obiektowy, zachowując przy tym hermetyczność (in. enkapsulację) – czyli teraz nasza baza będzie odporna na nieuprawnione zmiany – dajemy możliwość manipulowania danymi we wskazany przez nas sposób. Wszystkie metody i klasy są opisane, potrafimy walidować dane. Możemy wprowadzić pola klasy i metody klasy, które będą zmieniać informacje wspólne dla wszystkich instancji. Zadanie do wykonania do wtorku g.10.00