Laboratorium 7 (Programowanie 2 – 410-KS1-2PRO3)
Kierunek kognitywistyka i komunikacja
Programowanie obiektowe (OOP) w Pythonie - Klasy.

Klasy

1. **(przypomnienie: klasa, pola, metody) Przeanalizuj następujący kod**

```
class Person:
    name = ''
    age = None
    sex = ''

    # Konstruktor obiektu klasy
    def __init__(self, name, age, sex):
        self.name = name
        self.age = age
        self.sex = sex

    # Funkcje w klasie
    def introduce_yourself(self):
        print(f'Cześć. Mam na imię {self.name}.')

    def Move(self):
        if self.sex == 'female':
            print('Ruszyłam w drogę.')
        else:
            print('Ryszyłem w drogę.')

    def Think(self):
        if self.age < 2:
            print('Dopiero się uczę.')
        else:
            print('Nie ma problemu.')

mike = Person('Mike', 45, 'male')

print(f'Obiekt - {mike.name}, {mike.age}, {mike.sex}.\n')
mike.introduce_yourself()
mike.Move()
mike.Think()
```

Wskaż pola i metody.

2. **Nie uruchomiając poniższego kodu powiedz co zostanie wyświetlone na konsoli.**

```
print(mike)
```

Sprawdź, czy się nie myliłaś/myliłeś!

3. **(metody specjalne: __repr__() i __str__()) Dołącz do klasy *Person* następującą metodę specjalną**

```
def __repr__(self):
    return f'Person({self.name}, {self.age}, {self.sex})'
```

i ponownie wywołaj polecenie z punktu 2 oraz plecenie

```
print([mike, Person('Jan', 50, 'male')])
```

Co zaobserwowałaś/eś? Dołącz teraz do klasy kolejną metodę

```
def __str__(self):
     return f'Instancja klasy Person o następujących polach: name =
{self.name}, age = {self.age}, sex = {self.sex}'
```

i ponownie wywołaj polecenie z punktu 2 oraz poprzednie polecenie służące do wyświetlenia na konsoli. Co zaobserwowałaś/eś?

4. **Zdefiniuj klasę *book* zawierającej pola *title, author, year (rok wydania), pages (liczba stron), owner (właściciel)* oraz metody *__str__* (do ładnego wyświetlania na konsoli danych instancji) i metodę, która przypisuje nowego właściciela.**
5. **(metoda instancja vs metoda klasy vs. metoda statystyczna)**

```
from datetime import date
class Person:
    species = 'homo_sapiens'  # This is class variable

    def __init__(self, name, age, sex):
        self.name = name
        self.age = age
        self.sex = sex

    # A instance method
    def __str__(self):
        return f'Instancja klasy Person o następujących polach:\n\
        name = {self.name}\n\
        age = {self.age}\n\
        sex = {self.sex}.'

    def introduce_yourself(self):
        print(f'Cześć. Mam na imię {self.name}.')

    def Move(self):
        if self.sex == 'female':
            print('Ruszyłam w drogę.')
        else:
            print('Ryszyłem w drogę.')

    def Think(self):
        if self.age < 2:
            print('Dopiero się uczę.')
        else:
            print('Nie ma problemu.')

    # A class method to access to class variable
    @classmethod
    def print_species(cls):
        print('species: {}'.format(cls.species))

    # A class method to create a Person object by birth year.
    @classmethod
    def create_with_birth_year(cls, name, birth_year, sex):
        return cls(name, date.today().year - birth_year, sex)

    # A static method to check if a Person is adult or not.
    @staticmethod
    def check_is_adult(age):
        return 'Osoba pełnoletnia.' if age > 18 else 'Osoba
niepełnoletnia.'

    # A static method to get the Person birth year.
    @staticmethod
    def get_birth_year(age):
        return date.today().year - age
```

- **Metoda instancji (*introduce_yourself, Move, Think*) wymaga instancji i musi używać *self* jako pierwszego parametru. Może uzyskać dostęp do instancji poprzez siebie i wpływać na stan instancji.**
- **Metoda klasy (*create_with_birth_year, print_species*) nie wymaga instancji i używa *cls*, aby uzyskać dostęp do klasy i wpływać na stan klasy. Możemy użyć *@classmethod* do stworzenia fabryki.**
- **Metoda statyczna (*get_birth_year, check_is_adult*) nie wymaga specjalnego parametru (*self* lub *cls*) i zmieni dowolny stan klasy lub instancji. Może udostępniać jakąś funkcję pomocniczą dotyczącą klasy.**

Wykonaj i przeanalizuj następujący kod:

```
person1 = Person('Tom', 95, 'male')
person2 = Person.create_with_birth_year('Tom', 1926, 'male')
print(Person.check_is_adult(22))
Person.print_species()
```

6. **Dane wydanie książki można wywieźć za granicę bez pozwolenia celników jeżeli książka jest wydana po roku 1945. Zmodyfikuj definicję klasę *book* dodając metodę statyczną, która po podaniu roku wydania poda informację, czy można książkę wywieźć za granicę bez zgody Urzędu Celnego.**
7. **(__slots__: pola dynamiczne vs. pola z góry ustalone, czyli jak mieć szybszy dostęp do atrybutów i oszczędzić miejsca w pamięci.) Uruchom następujący kod**

```
tom = Person('Tom', 95, 'male')
tom.przydomek ='Voldemort'
print(tom.__dict__)
```

Czy jest on wykonywany bez błędów. Zmodyfikuj teraz definicję klasy (pola) w sposób następujący

```
__slots__ = ['name', 'age', 'sex']
```

Teraz spróbuj uruchomić trzy linijki kodu tego zadania. Jaki masz wnioski? Wypróbuj teraz polecenie

```
print(tom.__slots__)
```

8. **Zmodyfikuj definicję klasę *book* zmieniając pola z góry ustalone.**
9. **(hermetyzacja: pola prywatne, chronione i publiczne) Przeanalizuj następujący kod**

```
class Person:
    __slots__ = ["name", '_family', '__sex']

    def __init__(self):
        self.name, self._family,  self.__sex = 'Tom', 'Riddle',
'male'

    def __str__(self):
        return f'Pole publiczne = {self.name}, Pole chronione =
{self._family},  Pole prywatne = {self.__sex}'
```

Następnie utwórz instancje *tom* i spróbuj wyświetlić na konsoli po kolei instancję *tom* oraz pola *name, _family, __sex* dla instancji *tom*.

10. **Przeanalizuj następujący kod dotyczący działania pojazdu:**

```
class vehicle:
    def __init__(self):
        self.engine = False
        self.gear = 0
        self.speed = 0

    def start_vehicle(self):
        self.engine = True

    def turn_off_vehicle(self):
        self.engine = False

    def engage_next_gear(self):
        if self.gear <= 5:
            self.gear += 1
        print(self.gear)

    def engage_previous_gear(self):
        if self.gear >= 0:
            self.gear -= 1
        print(self.gear)

    def speed_up_vehicle(self):
        if self.engine == True and self.gear > 0:
            self.speed += 10
        print(self.speed)

    def brake_vehicle(self):
        if self.speed >= 10:
            self.speed -= 10
        else:
            self.speed = 0
        print(self.speed)
```

11. **Stwórz instancje klasy *vehicle* będącą samochodem marki fiat, a następnie uruchom samochód zwiększ dwukrotnie prędkość, zmień dwukrotnie biegi, zwolnij i zatrzymaj samochód.**
12. **(hermetyzacja cd.) Zmodyfikuj definicję klasy *vehicle* w taki sposób, aby nie można było ustawiać wartości biegu, prędkości i silnika przez użytkownika klasy.**
13. **(hermetyzacja cd. - metody prywatne) Zmodyfikuj definicję klasy *vehicle* w taki sposób, aby symulować automatyczną skrzynię biegów.**
14. **(dziedziczenie) Przeanalizuj następujący kod:**

```
class bird:
    def __init__(self, species, speed):
        self.species = species
        self.speed = speed

    def fly(self):
        print(f'Tu {self.species}. Startuje, i zaraz osiągne prędkość
{self.speed}.')

    def make a sound(self):
        pass


class eagle(bird):
    def   init  (self, speed):
        super().__init__('orzeł', speed)
```

```
    def hunt(self):
        print(f'Tu {self.species}. Rozpocząłem polowanie.')


class penguin(bird):
    def __init__(self, speed):
        super().__init__('pingwin', speed)

    def slide(self):
        print(f'Tu {self.species}. Rozpocząłem ślizg.')

    def fly(self):
        print('Tu ', self.species, '. Przykro mi, ale nie latam.',
sep='')
```

15. (metody abstrakcyjne z pakietu abc) Przeanalizuj następujący kod

```
from abc import ABC, abstractmethod
class bird(ABC):
    def __init__(self, species, speed):
        self.species = species
        self.speed = speed

    def fly(self):
        print(f'Tu {self.species}. Startuję, i zaraz osiągnę prędkość
{self.speed}.')

    @abstractmethod
    def make_a_sound(self):
        pass


class eagle(bird):
    def __init__(self, speed):
        super().__init__('orzeł', speed)

    def hunt(self):
        print(f'Tu {self.species}. Rozpocząłem polowanie.')

    def make_a_sound(self):
        print('wrrrr.')


class penguin(bird):
    def __init__(self, speed):
        super().__init__('pingwin', speed)

    def slide(self):
        print(f'Tu {self.species}. Rozpocząłem ślizg.')

    def fly(self):
        print(f'Tu {self.species}. Przykro mi, ale nie latam.')

    def make_a_sound(self):
        print('kwiiiii.')
```