Klasy

April 16, 2023

[ ]:
```
Lab04 - Klasy
```

[ ]:
```
1. Przykład klasy

"""
Klasy
"""


class Zwierz:
    """Pierwsza klasa"""

    def podaj_gatunek(self):
        print("lis")


a = Zwierz()
print(a)

b = Zwierz()
print(b)

# Sprawdzenie czy mamy do czynienia z tym samym obiektem
print(a == b)

# wywołanie metody obiektu
a.podaj_gatunek()
b.podaj_gatunek()
```

[ ]:
```
2. Inicializowanie klas i atrybuty klas

"""
Inicializowanie klas i atrybuty klas
"""


class Zwierz:
```

```
    """Pierwsza klasa"""
    rodzaj = "zwierzę"

    def __init__(self, gatunek, wiek):
        self.gatunek = gatunek
        self.wiek = wiek

    def podaj_gatunek(self):
        print("lis")


# instancje klas
a = Zwierz("lis", 5)
b = Zwierz("python", 2)

# wyświetlenie wartości atrybutów
print(a.gatunek, a.wiek)
print(b.gatunek, b.wiek)

# definiowanie atrybutów instancji poza klasą
b.dlugosc = 10
print(b.dlugosc)
# print(a.dlugosc)

# wyświetlenie atrybutów wspólnych dla obiektów klas
print(a.rodzaj, b.rodzaj)

# zmiana wartości atrybutów
b.dlugosc = 11
print(b.dlugosc)
a.wiek = 6
print(a.wiek)

# atrybuty specialne - nie wymagają deklaracji

# atrybuty klasy w formie słownika
print(a.__dict__)
print(b.__dict__)

# opis klasy
print(Zwierz.__doc__)
print(a.__doc__)
```

[ ]:
```
3. Metody klas

"""
Metody klas
```

```
"""


class Zwierz:
    """Pierwsza klasa"""
    rodzaj = "zwierzę"
    zwierzeta = {}

    def __init__(self, gatunek, wiek, predkosc):
        self.gatunek = gatunek
        self.wiek = wiek
        self.max_predkosc = predkosc
        if gatunek in Zwierz.zwierzeta:
            Zwierz.zwierzeta[gatunek] += 1
        else:
            Zwierz.zwierzeta[gatunek] = 1

    def oblicz_odleglosc(self, czas):
        print(czas * self.max_predkosc)

    @staticmethod
    def wypisz_zwierzeta():
        print(Zwierz.zwierzeta)


Zwierz.wypisz_zwierzeta()

# instancje klas (inicializacja obiektów)
a = Zwierz("lis", 5, 10)
b = Zwierz("python", 2, 5)
c = Zwierz("lis", 3, 10)

a.oblicz_odleglosc(2)
b.oblicz_odleglosc(2)

# definiowenie metod na poziomie klas
Zwierz.wypisz_zwierzeta()
```

[ ]:
```
4. Metody specialne klas

"""
Metody specialne klas
"""


class Zwierz:
    """Pierwsza klasa"""
```

```
    rodzaj = "zwierzę"
    zwierzeta = {}

    def __init__(self, gatunek, wiek, predkosc):
        self.gatunek = gatunek
        self.wiek = wiek
        self.max_predkosc = predkosc
        if gatunek in Zwierz.zwierzeta:
            Zwierz.zwierzeta[gatunek] += 1
        else:
            Zwierz.zwierzeta[gatunek] = 1

    def oblicz_odleglosc(self, czas):
        print(czas * self.max_predkosc)

    @staticmethod
    def wypisz_zwierzeta():
        print(Zwierz.zwierzeta)

    # nadpisuje zmienną specialną (zmiana działania polecenia print)
    def __str__(self):
        return self.gatunek + " ma " + str(self.wiek) + \
               " lat i osiaga predkosc " + str(self.max_predkosc) + " km/h."


a = Zwierz("lis", 3, 10)
print(a)
```

[ ]:
```
5. Dziedziczenie

"""
Dziedziczenie
"""


class Zwierz:
    """Pierwsza klasa"""
    rodzaj = "zwierzę"
    zwierzeta = {}

    def __init__(self, gatunek, wiek, predkosc):
        self.gatunek = gatunek
        self.wiek = wiek
        self.max_predkosc = predkosc
        if gatunek in Zwierz.zwierzeta:
            Zwierz.zwierzeta[gatunek] += 1
        else:
```

```
            Zwierz.zwierzeta[gatunek] = 1

    def oblicz_odleglosc(self, czas):
        print(czas * self.max_predkosc)

    @staticmethod
    def wypisz_zwierzeta():
        print(Zwierz.zwierzeta)

    # nadpisuje zmienną specialną (zmiana działania polecenia print)
    def __str__(self):
        return self.gatunek + " ma " + str(self.wiek) + \
               " lat i osiaga predkosc " + str(self.max_predkosc) + " km/h."


class Ptak(Zwierz):
    def __init__(self, gatunek, wiek, predkosc, max_predkosc_lotu, miejsce):
        # funkcja super() zwraca klasę Zwierz
        super().__init__(gatunek, wiek, predkosc)
        self.predkosc_lotu = max_predkosc_lotu
        self.miejsce = miejsce

    def przenies(self):
        if self.miejsce == "klatka":
            self.miejsce = "otwarty"
        else:
            self.miejsce = "klatka"


# deklaracja instancji klasy
p = Ptak("pingwin", 2, 3, 0, "otwarty")
print(p)

p.przenies()
print(p.miejsce)

p.przenies()
print(p.miejsce)
```

[ ]:
```
6. Polimorfizm

"""
Polimorfizm - przesłanianie metod klasy nadrzędnej. Metody prywatne
"""


class Zwierz:
```

```
    """Pierwsza klasa"""
    rodzaj = "zwierzę"
    zwierzeta = {}

    def __init__(self, gatunek, wiek, predkosc, stan_zdrowia):
        self.gatunek = gatunek
        self.wiek = wiek
        self.max_predkosc = predkosc
        self.stan_zdrowia = stan_zdrowia
        if gatunek in Zwierz.zwierzeta:
            Zwierz.zwierzeta[gatunek] += 1
        else:
            Zwierz.zwierzeta[gatunek] = 1

    def _sprawdz_stan_zdrowia(self):
        if self.stan_zdrowia == 1:
            return 1
        else:
            return 0

    def oblicz_odleglosc(self, czas):
        print(czas * self.max_predkosc)

    @staticmethod
    def wypisz_zwierzeta():
        print(Zwierz.zwierzeta)

    # nadpisuje zmienną specialną (zmiana działania polecenia print)
    def __str__(self):
        return self.gatunek + " ma " + str(self.wiek) + \
               " lat i osiaga predkosc " + str(self.max_predkosc) + " km/h."


class Ptak(Zwierz):
    def __init__(self, gatunek, wiek, predkosc, stan_zdrowia,␣
 ↪max_predkosc_lotu, miejsce):
        # funkcja super() zwraca klasę Zwierz
        super().__init__(gatunek, wiek, predkosc, stan_zdrowia)
        self.predkosc_lotu = max_predkosc_lotu
        self.miejsce = miejsce

    def przenies(self):
        if self.miejsce == "klatka" and self._sprawdz_stan_zdrowia() == 1:
            self.miejsce = "otwarty"
        else:
            self.miejsce = "klatka"
```

```
    def oblicz_odleglosc(self, czas):
        if self.predkosc_lotu == 0:
            print(czas * self.max_predkosc)
        else:
            print(czas * self.predkosc_lotu)


p = Ptak("pingwin", 2, 3, 1, 0, "otwarty")
p1 = Ptak("kos", 2, 2, 0, 15, "klatka")

# odległość w h
p.oblicz_odleglosc(2)
p1.oblicz_odleglosc(2)

# metody prywatne
p.przenies()
print(p.miejsce)

p1.przenies()
print(p1.miejsce)
```

[ ]:
```
7. Zadania

Do każdego z podanych niżej zadań dołącz skrypt prezentujacy działanie kodu

Zadanie 1
-----------------------------------------------------------

Napisz skrypt, który będzie działał jak prosta baza produktów spożywczych.␣
 ↪Skrypt ma korzytać z klas.
Zmodyfikuj kod z zadania wykonanego na jednych z poprzednich zajęć.


Zadanie 2
-----------------------------------------------------------
Napisz skrypt z implementacją algorytmu Dijkstra. Skrypt ma korzystać z klas.
Zmodyfikuj kod z zadania wykonanego na jednych z poprzednich zajęć.


Zadanie 3
-----------------------------------------------------------

Utwórz klasę Gracz, która będzie przechowywać informacje o użytkowniku gry.␣
 ↪Klasa Gracz będzie dziedziczyć po klasie Osoba.
Klasa Osoba powinna zawierać pola: Imie, Nazwisko, Płeć, Data_urodzenia oraz␣
 ↪metody: wyswietl_info(), wyswietl_osoby().
```

```
Metoda wyswietl_info() powinna umożliwić wyswietlenie informacji o osobie w␣
 ↪postaci listy, krotki lub słownika.
Metoda wyswietl_osoby() powinna zwrócić informacje o osobach w postaci listy␣
 ↪słowników: [Osoba1: {}, Osoba2: {}]
Klasa Gracz powinna zawierać pola: Nick, Typ:{NPC, Human}, Email oraz metody:␣
 ↪wyswietl_info(), wyswietl_graczy().
Metoda wyswietl_info() powinna umożliwić wyświetlenie informacji o graczu w␣
 ↪postaci listy, krotki lub słownika.
Metoda wyswietl_graczy() powinna zwrócić informacje o graczach w postaci listy␣
 ↪słowników: [Gracz1: {}, Gracz2: {}]
```