Klasa - abstrakcyjny opis obiektu/rzeczywistości, np. samochód, który można opisać za pomocą marki, roku produkcji, parametrów silnika, aktualnej prędkości, etc.
Obiekt - konkretny egzemplarz samochodu (oraz jego stan), np. Kia Sportage 2022 1.6 T-DGI, pruszająca się z prędkością 75km/h na 6 biegu.
Metoda - funkcja powiązana z daną klasą, która może być wywołana na rzecz obiektu danej klasy.
Pole / *Parametr* - dane dotyczące obiektu.

Zad.1. Budowanie klasy.

```
class Car:
    def __init__(self, make):
        self.make = make
        self.motor_run = False
        self.gear = 0
        self.speed = 0

    def motor_start(self):
        self.motor_run = True

    def motor_stop(self):
        self.motor_run = False

    def gear_change_up(self):
        if self.gear <= 7:
            self.gear+=1
        print(self.gear)

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

my_car = Car("Kia Sportage")
my_car.motor_start()
my_car.gear_change_down()
my_car.gear
```

Dodaj do powyższego kodu metody `speed_up` i `brake`.

self - w Pythone metody przyjmują jako pierwszy parametr obiekt, na rzecz którego są wywoływane. Przy wywoływaniu metody argument *self* należy pominąć.

Zad.2. Hermetyzacja.

Hermetyzacja / Enkapsulacja - ograniczenie dostępu do wybranych składowych klasy. Zwyczajowo:

- *publiczne* / *public* - dostępne dla każdego;
- *chronione* / *protected* - dostęp dla klas dziedziczących (patrz zad.5);
- *prywatne* / *private* - tylko wewnętrz klasy.

```
class Encapsulation:
    def __init__(self):
        self.public, self._protected, self.__private = 1, 2, 3


def main():
    encapsulation = Encapsulation()
    print(encapsulation.public)
    print(encapsulation._protected)
    print(encapsulation._Encapsulation__private)
# a teraz będzie błąd
    print(encapsulation.__private)


main()
```

W przypadku klasy *car*, z zad.1:

```
class Car:
    def __init__(self, make):
        self.make = make
        self.__motor_run = False
        self.__gear = 0
        self.__speed = 0

    def motor_start(self):
        self.__motor_run = True
```

```
    def motor_stop(self):
        self.__motor_run = False

    def gear_change_up(self):
        if self.__gear <= 7:
            self.__gear+=1
        print(self.gear)

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

my_car = Car("Kia")
my_car.motor_start()
my_car.gear_change_down()
my_car.gear
```

Zaimplementuj automatyczną zmianę biegów.

```
def __gear_change_up(self):
     if self.__gear <= 7:
     self.__gear+=1
     print(self.gear)
```

Zad.3. Pola statyczne. Metody statyczne.

Pola statyczne(*static*) są współdzielone przez wszystkie obiekty danej klasy. Przynależą one do klasy, a nie do obiektu. Metody statyczne nie mogą korzystać z pól i metod instancyjnych (niestatycznych) - nie są wywoływane na rzecz danego obiektu.

```
class Car:
    how_many = 0

    def __init__(self):
        Car.how_many += 1
        self.car_number = Car.how_many
        print(f"Numer of cars is equal to {Car.how_many}")

    def __del__(self):
```

```
        Car.how_many -= 1
        print(f"Number of cars is equal to {Car.how_many}")

    @staticmethod
    def count_cars():
        return Car.how_many

car_1 = Car()
car_2 = Car()
car_3 = Car()
print(f"Total number of cars {Car.count_cars()}")
car_2 = None
print(f"Total number of cars {Car.count_cars()}")
```

Zad.4. Stwórz klasę *Q_function*. Klasa powinna pozwalać na zdefiniowane funkcji kwadratowej (a*x^2 + b*x + c) i zawierać medodę *Solve()* zwracająca miejsca zerowe.

Zad.5. Dziedziczenie.

Mechanizm pozwalający na przejęcie pewnych cech (pola i metody) z klasy bazowej przez klasę pochodną/potomną. Klasa pochodna może dodać własne składowe, a także może zmieniać działanie metod klasy bazowej.

```
class Person:
    def __init__(self,name,surname,age):
        self.name = name
        self.surname = surname
        self.age = age

    def hasName(self):
        print("Has name")

class Student(Person):
    def __init__(self,name,surname,age,field_of_study):
        super().__init__(name,surname,age)
        self.field_of_study = field_of_study

    def isStudent(self):
        print("Is student")

person_1 = Person("Tom","Nowak",25)
```

```
student_1 = Student("Tom","Nowak",25,"Informatics")

student_1.hasName()
```

Zad.4. Przygotowanie środowiska pracy.

Sugerowanym środowiskiem pracy jest PyCharm oraz Python 3.10/3.11. Na zajęciach można jednak używać dowolnego środowiska pozwalającego na uruchamianie skryptów Python - Jupyter Notebook, Thonny IDE, Microsoft Visual Studio Code, etc. Potrzebne moduły: *opencv-python*, *mediapipe*, *numpy*. W środowisku PyCharm można je dodać korzystając z:

File -> Settings -> Project: python -> Python interpreter

Po dodaniu modułów należy podłączyć kamerę do portu USB komputera oraz uruchomić poniższy skrypt.

```
import cv2

wCam, hCam = 800,600
cap = cv2.VideoCapture(0)
cap.set(3,wCam)
cap.set(4,hCam)

while True:
   success, img = cap.read()

   cv2.imshow("Image", img)
   cv2.waitKey()
```

Zad.5. Utworzyć i uruchomić w wybranym IDE poniższy skrypt.

```
import cv2
import mediapipe as mp
import time

c_time = 0
p_time = 0
```

```
        #webcam
        cap = cv2.VideoCapture(0)
        mpHands = mp.solutions.hands
        hands = mpHands.Hands()
        mpDraw = mp.solutions.drawing_utils

        while True:
           sucess, img = cap.read()
           imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
           results = hands.process(imgRGB)
           print(results.multi_hand_landmarks)
           if results.multi_hand_landmarks:
               for hand_landmarks in results.multi_hand_landmarks:
                  mpDraw.draw_landmarks(img,hand_landmarks,
                                        mpHands.HAND_CONNECTIONS)
                  for id, lm in enumerate(hand_landmarks.landmark):
                       print(id,lm)
                       height,width,c = img.shape
                       cx, cy = int(lm.x*width), int(lm.y*height)
                       print(id, cx, cy)
                       if id == 0:
                           cv2.circle(img,(cx,cy),15,(255,255,0),cv2.FILLED)

           #FPS calculation
           c_time = time.time()
           fps = int(1/(c_time - p_time))
           p_time = c_time
           cv2.putText(img,str(fps),(15,80),cv2.FONT_HERSHEY_SIMPLEX,
                       3,(0,0,255),2)

           cv2.imshow("Image",img)
           cv2.waitKey(1)
```

Zad.6. Zbudować klasę *HandTracking*.

```
import cv2
import mediapipe as mp

class HandTracking():
   def __init__(self, mode = False, maxHands = 2, modelComplexity = 1,
                detectonConfidence = 0.5, trackingConfidence = 0.5):
       self.mode = mode
       self.maxHands = maxHands
       self.modelComplexity = modelComplexity
       self.detectonConfidence = detectonConfidence
       self.trackingConfidence = trackingConfidence
       self.mpHands = mp.solutions.hands
       self.hands = self.mpHands.Hands(self.mode,self.maxHands,self.modelComplexity,
                                       self.detectonConfidence,self.trackingConfidence)
```

```
        self.mpDraw = mp.solutions.drawing_utils
        # fingers: index, middle, ring, pinky
        self.fingerTipsId = [8, 12, 16, 20]

    def findHand(self, img, draw = True):
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imgRGB)
        if self.results.multi_hand_landmarks:
            for hand_landmarks in self.results.multi_hand_landmarks:
                if draw:
                    self.mpDraw.draw_landmarks(img, hand_landmarks,
                                               self.mpHands.HAND_CONNECTIONS)
        return img

    def findPostition(self, img, handNumber = 0, draw = True):
        self.landmarkList = []
        if self.results.multi_hand_landmarks:
            processedHand = self.results.multi_hand_landmarks[handNumber]
            for id, lm in enumerate(processedHand.landmark):
                height, width, c = img.shape
                cx, cy = int(lm.x * width), int(lm.y * height)
                self.landmarkList.append([id, cx, cy])
                if draw:
                    cv2.circle(img, (cx, cy), 10, (255, 255, 0), cv2.FILLED)
        return self.landmarkList
```

Przykład:

```
import cv2
import mediapipe as mp
import time
import HandTrackingClass as htc

c_time = 0
p_time = 0

#webcam
cap = cv2.VideoCapture(0)
detector = htc.handDetector()

while True:
    sucess, img = cap.read()
    img = detector.findHand(img,True)
    landmarkList = detector.findPostition(img,draw = True)

    if len(landmarkList) != 0:
        print(landmarkList[2])

    c_time = time.time()
    fps = int(1 / (c_time - p_time))
    p_time = c_time
    cv2.putText(img, str(fps), (15, 80), cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 0, 255), 2)

    cv2.imshow("Image", img)
```

```
   cv2.waitKey(1)
```

Zad.7. Wykorzystać klasę z zadania 6 do zliczania ilości palców.

```
import cv2
import os
import HandTracking as htc

wCam, hCam = 1280, 720
cap = cv2.VideoCapture(0)
cap.set(3,wCam)
cap.set(4,hCam)

fingerPath = "FingerImages"
myList = os.listdir(fingerPath)
#print(myList)
overlayList = []
for imPath in myList:
   image = cv2.imread(f'{fingerPath}/{imPath}')
   overlayList.append(image)

detector = htc.handDetector(detectonConfidence=0.8)

# fingers: index, middle, ring, pinkey
fingertipsId = [8, 12, 16, 20]

while True:
   success, img = cap.read()
   img = detector.findHand(img,draw = True)
   landmarkList = detector.findPostition(img,draw = False)
#    print(landmarkList)

   if len(landmarkList) != 0:
       fingers = []
       #thumb
       if landmarkList[4][1] > landmarkList[3][1]:
           fingers.append(1)
       else:
           fingers.append(0)
       # fingers: index, middle, ring, pinky
       for id in range(0,4):
            if landmarkList[fingertipsId[id]][2] < landmarkList[fingertipsId[id]-2][2]:
               fingers.append(1)
           else:
               fingers.append(0)
       #print(fingers)
       countFingers = fingers.count(1)
       #print(countFingers)
```

```
        h,w,c = overlayList[countFingers].shape
        img[0:h, 0:w] = overlayList[countFingers]
         cv2.putText(img,str(countFingers),(20,280),cv2.FONT_HERSHEY_COMPLEX,
               2,(255,170,0),3)

    cv2.imshow("Image", img)
    cv2.waitKey(1)
```

Zad.8. Rozbudować klasę z zadania 6 o metodę finUp zwracającą ilość wyprostowanych palców.

Zad.9. Wykorzystać klasę z zadania 6 do sterowania poziomem jasności monitora.