CAR_DEALERSHIP.PY

```
from datetime import date

car_register = {
    "toyotaBZ4X": {
            "brand": "Toyota",
            "model": "Corolla",
            "price": 96_000,
            "year": 2012,
            "month": 8,
            "new": False,
            "km": 163_000
    },
    "pugeot408": {
            "brand": "Pugeot",
            "model": "408",
            "price": 330_000,
            "year": 2019,
            "month": 1,
            "new": False,
            "km": 40_000
    },
    "audiRS3": {
            "brand": "Audi",
            "model": "RS3",
            "price": 473_000,
            "year": 2022,
            "month": 2,
            "new": True,
            "km": 0
    },
}

NEW_CAR_REGISTRATION_FEE = 8745
RENT_CAR_PERCENTAGE = 0.4
RENT_NEW_CAR__FEE = 1000


def print_car_information(car):
    # Oppgave 3.1

def create_car(car_register, brand, model, price, year, month, new, km):
    # Oppgave 3.2

def get_car_age(car):
    # Oppgave 3.3

def next_eu_control(car):
    # Oppgave 3.4

def rent_car_monthly_price(car):
    # Oppgave 3.5

def calculate_total_price(car):
    # Oppgave 3.6



def is_new(car):
    return car['new']
```

```
if __name__ == '__main__':
    create_car(car_register, "Volvo", "V90", 850_000, 2021, 12, True, 0)


    toyota = car_register['toyotaBZ4X']

    print_car_information(toyota)

    print(f"\nThe total price for this {toyota['brand']} {toyota['model']} is
          {calculate_total_price(toyota)}kr.")
    print(f"Next EU-control for the {toyota['brand']} {toyota['model']} is
          {next_eu_control(toyota)}")
    print(f"If you want to rent the {toyota['brand']} {toyota['model']} the
          monthly fee will be {rent_car_monthly_price(toyota)}.")


    audi = car_register['audiRS3']

    print_car_information(audi)

    print(f"\nThe total price for this {audi['brand']} {audi['model']} is
          {calculate_total_price(audi)}kr.")
    print(f"Next EU-control for the {audi['brand']} {audi['model']} is
          {next_eu_control(audi)}")
    print(f"If you want to rent the {audi['brand']} {audi['model']} the monthly
          fee will be {rent_car_monthly_price(audi)}kr.")
```

CAR_DEALERSHIP.PY - OUTPUT

```
Brand: Toyota
Model: Corolla
Price: 96000,-
Manufactured: 2012-8
Condition: Used
Km: 163000
The total price for this Toyota Corolla is 100034kr.
Next EU-control for the Toyota Corolla is 2022-08-01
If you want to rent the Toyota Corolla the monthly fee will be 3200.0.

Brand: Audi
Model: RS3
Price: 473000,-
Manufactured: 2022-2
Condition: New
Km: 0
The total price for this Audi RS3 is 481745kr.
Next EU-control for the Audi RS3 is 2024-02-01
If you want to rent the Audi RS3 the monthly fee will be 16766.67kr.
```

DATE – UTDRAG FRA DOKUMENTASJON

date Objects

A date object represents a date (year, month and day) in an idealized calendar, the current Gregorian calendar indefinitely extended in both directions.

January 1 of year 1 is called day number 1, January 2 of year 1 is called day number 2, and so on. 2

class `datetime.`**`date`**(*`year`*, *`month`*, *`day`*)

All arguments are required. Arguments must be integers, in the following ranges:

- `MINYEAR <= year <= MAXYEAR`
- `1 <= month <= 12`
- `1 <= day <= number of days in the given month and year`

If an argument outside those ranges is given, ValueError is raised.

Example:

```
>>> a_date = date(2022, 1, 28)
>>> print(a_date)
2022-01-28
```

classmethod `date.`**`today`**()

Return the current local date.

```
Example:
  >>> todays_date = date.today()
  >>> print(todays_date)
  2022-02-22
```

Instance attributes (read-only):

`date.`**`year`**

Between MINYEAR and MAXYEAR inclusive.

`date.`**`month`**

Between 1 and 12 inclusive.

`date.`**`day`**

Between 1 and the number of days in the given month of the given year.

Example:

```
>>> a_date = date(2022, 1, 28)
>>> print(a_date.year)
>>> print(a_date.month)
>>> print(a_date.day)
2022
1
28
```

DEL 3 – OPPGAVETEKST – REPETERT

Vi utvikler et system for en oppdragsgiver, BilligBil A/S. Vi har nettopp begynt utviklingen og vi jobber med å få på plass kjernefunksjonalitet. Det skal selges og leies brukte og nye biler, og du skal implementere noen av funksjonene det vil bli behov for å ha.

Vedlagt oppgavene ligger det informasjon fra:

- *car_dealership.py* - inneholder:
  - funksjonsdefinisjonene og annen kode du skal ta utgangspunkt i
  - Implementasjon av main som illustrerer bruken av funksjonene du skal implementere
- *car_dealership output* - gir outputen du ville fått ved kjøring av main-metoden
- *date dokumentasjon* - utdrag av dokumentasjonen for date-klassen du potensielt vil ha nytte av i noen av funksjonene
- *del 3 - repetert oppgavetekst* - denne teksten er repetert i det vedlagte dokumentet

Del 3 består av 7 oppgaver som er er basert på å fylle ut en rekke funksjonsdefinisjoner. Disse funksjonene har ferdigdefinerte parametere du skal ta utgangspunkt i. Du kan likevel modifisere disse parameterne og/eller legge til flere. Du kan også anta at verdiene som blir sendt inn med disse parameterne samsvarer med det som er å forvente. For eksempel, hvis en parameter skal ta et heltall for å representere kilometerstand, kan du anta at denne verdien alltid vil være et heltall og aldri negativt.

Ta utgangspunkt i at ***car***-parameteren, som er satt opp i en del av funksjonene i *car_dealershop.py*, er en direkte referense til en gitt bil. Altså en av de indre dictionary-elementene i car_register. Et eksempel på dette kan være:

```
{
        "brand": "Toyota",
        "model": "Corolla",
        "price": 96_000,
        "year": 2012,
        "month": 8,
        "new": False,
        "km": 163_000
}
```

Du står fritt til å utvide koden med egne funksjoner om du ser et behov for det. I motsetning til funksjoner fra andre oppgaver; Hvis du velger å utvide med egne funksjoner, skal du inkludere definisjonene av disse i alle oppgaver du benytter dem.