차량 출고 시 유지비 계산기

학번: 2018068

이름: 손승현

Github address: SonseungHyun1

1. 계산기의 목적

a. 이 계산기의 목적은 차량을 구매할 경우 고려해야 할 여러 지출 요인에 대해 계산해줄 수 있는 계산기로 차량 자체의 가격, 차량의 용도, 종류 등 여러 항목을 종합하여 차량의 유지비 계산을 목적으로 하며, 사용자는 차량의 구매 조건과 입력자의 정보 등을 입력하면 연간 차량의 유지비와 월간 차량의 유지비를 각각 산출해내는 계산기이다.

b. 계산기 활용 대상:
차량 출고 계획을 세우면서 월 유지비에 대한 계산이 필요하거나 유지비를 통해 차량 구매 계획을 세워 차량별로 비교 분석이 필요한 사람들을 대상으로 한다.

2. 계산기의 네이밍의 의미

a. 제목 그대로 차량을 출고하고, 앞으로 지출되는 차량의 유지비를 계산해준다는 의미를 가지고 있다.

3. 계산기 개발 계획

a. 입력 변수

car_price: 차량의 가격을 의미
driver_age: 운전자의 나이를 의미
fuel_type: 차량의 연료 유형을 의미
annual_mileage: 연간 주행 예상 거리를 의미
fuel_efficiency: 차량의 평균 연비를 의미
down_payment: 차량 구매 시 선납금을 의미
loan_duration: 할부 개월을 의미

b. 개발한 함수는 무엇을 연산하는 함수인지

get_car_price(self)
역할: 사용자로부터 차량의 가격을 입력 받는다.
연산: 입력 값이 숫자인지 확인하고, 숫자일 경우에만 차량 가격을 설정한다.

get_driver_age(self)
역할: 사용자로부터 운전자의 나이를 입력 받는다.
연산: 입력 값을 정수형으로 변환하여 운전자의 나이를 설정한다.

get_fuel_expenses(self)
역할: 사용자로부터 차량의 연료 유형, 연간 주행 거리, 차량의 연비를 입력 받는다.
연산: 입력 값이 숫자인지 확인하고, 숫자일 경우에만 해당 값을 설정한다.

get_installment_payment(self)
역할: 사용자로부터 선납금과 할부 기간을 입력 받는다.
연산: 입력 값이 숫자인지 확인하고, 숫자일 경우에만 해당 값을 설정한다.

calculate_insurance_premium(self)
역할: 차량의 보험료를 계산한다.
연산: 차량 가격과 운전자의 나이를 고려하여 보험료를 계산하고, 연간 및 월간 보험료를 반환한다.

calculate_fuel_expenses(self)
역할: 차량의 연료비를 계산한다.
연산: 연료 유형에 따라 연료 소비량을 계산하고, 연간 및 월간 연료비를 반환한다.

calculate_installment_payment(self)
역할: 할부금 상환액을 계산한다.
연산: 차량 가격과 선납금, 연이자율, 할부 기간을 고려하여 월 할부금을 계산하고 반환한다.

calculate_taxes(self)
역할: 차량에 대한 세금을 계산한다.
연산: 차량 가격을 기반으로 소비세, 교육세, 등록세를 계산하고 총 세금을 반환한다.

calculate_total_expenses(self)
역할: 전체 유지비를 계산한다.
연산: 보험료, 연료비, 할부금, 세금을 고려하여 월간 및 연간 유지비를 계산하고 반환한다.

c. 연산 과정은 어떻게?, 조건문은 왜 필요하며, 왜 이렇게 설계했는지 등

연산과정

차량 가격 입력 받기
운전자의 나이 입력 받기
연료 비용 관련 정보 입력 받기
할부금 및 기간 입력 받기
보험료 계산
연료 비용 계산
할부금 계산
세금 및 등록세 계산
총 비용 계산
결과 출력 순으로 연산이 진행된다.

조건문

get_car_price(self)

```
if not car_price_str.replace('.', '').isdigit():
        print("입력값은 숫자여야 합니다.")
        return False
    self.car_price = float(car_price_str) * 1000000
    return True
```

조건문: 입력 값이 숫자가 아닌 경우를 확인한다.
사용자가 가격을 입력할 때, 숫자가 아닌 문자를 입력하는 상황을 방지하기 위해 사용했다.
숫자가 아닌 입력 값을 받으면 숫자로 입력할 때까지 계속 입력하게 한다.

get_fuel_expenses(self)

```
if not annual_mileage_str.isdigit() or not fuel_efficiency_str.replace('.', '').isdigit():
        print("입력값은 숫자여야 합니다.")
        return False
```

조건문: 입력 값이 숫자가 아니거나 연비 값이 숫자가 아닌 경우를 확인한다.
사용자에게 연료 유형, 연간 주행 거리, 연비를 입력 받을 때, 숫자가 아닌 값을 입력하거나 연비에 숫자가 아닌 값이 들어가면 계산이 제대로 이루어지지 않을 수 있어 올바른 입력을 받기 위해 조건문을 사용했다

get_installment_payment(self)

```
if not down_payment_str.replace('.', '').isdigit() or not loan_duration_str.isdigit():
            print("입력값은 숫자여야 합니다.")
            return False

        self.down_payment = float(down_payment_str) * 1000000
        self.loan_duration = int(loan_duration_str)
        return True
```

조건문: 입력 값이 숫자가 아니거나 할부 기간이 숫자가 아닌 경우를 확인한다.
선납금과 할부 기간을 입력 받을 때, 숫자가 아닌 값을 입력하면 할부금 계산이 올바르게 이루어지지 않을 수 있기 때문에 올바른 입력을 받기 위해 조건문이 사용했다.

def get_installment_payment(self):

```
if not down_payment_str.replace('.', '').isdigit() or not loan_duration_str.isdigit():
            print("입력값은 숫자여야 합니다.")
            return False

        self.down_payment = float(down_payment_str) * 1000000
        self.loan_duration = int(loan_duration_str)
        return True
```

조건문: 입력된 선납금이 소수점을 포함한 숫자로 이루어져 있지 않은 경우를 체크하고 입력된 할부 개월 수가 숫자로 이루어져 있지 않은 경우를 체크한다.
올바른 유형은 할부 개월 수와 선납금을 입력 받지 않으면 할부 금액 계산이 올바르게 이루어지지 않을 수 있기 때문에 올바른 입력을 받기 위해 조건문이 사용했다.

calculate_insurance_premium(self)

```
if self.car_price > 50000000:
            additional_fee += 0.1

        if self.driver_age < 25:
            additional_fee += 0.5
```

```
        elif 25 <= self.driver_age < 35:
            additional_fee += 0.4
        elif 35 <= self.driver_age < 45:
            additional_fee += 0.3
        elif self.driver_age > 60:
            additional_fee += 0.1
```

조건문: 차량 가격 및 운전자의 나이에 따라 추가 보험료를 계산할 때 조건문을 사용한다.
차량 가격이 특정 금액 이상이거나 운전자의 나이가 특정 범위에 속할 경우 추가 보험료를 부과하기 위해 조건문을 사용했다.

calculate_fuel_expenses(self)

```
if self.fuel_type == "가솔린":
            fuel_consumption = self.annual_mileage / self.fuel_efficiency
            fuel_cost = fuel_consumption * petrol_price
        elif self.fuel_type == "디젤":
            fuel_consumption = self.annual_mileage / self.fuel_efficiency
            fuel_cost = fuel_consumption * diesel_price
        else:
            print("유효하지 않은 연료 유형입니다.")
            fuel_cost = 0
```

조건문: 입력된 연료 유형이 "가솔린" 또는 "디젤"이 아닌 경우를 확인한다.
올바른 연료 유형을 입력 받지 않으면 연료비 계산이 올바르게 이루어지지 않을 수 있기 때문에 올바른 입력을 받기 위해 조건문이 사용했다.

설계 목적: 차량의 가격, 연료 유형, 주행 예상 거리, 평균 연비, 선납금, 운전자의 나이, 할부 개월 수를 입력 받아 연간, 월간 유지비에 대한 정확한 결과값을 도출해 내기 위해 각 함수들에 필요한 조건문을 사용하여 코드를 진행시켰다.

4. 계산기 개발 과정

a. 계획 후 실제 개발 과정을 기록

해당 계산기를 계획한 후 가장 먼저 자동차를 유지하는 데에 필요한 요소들에 대해 조사를 진행했다.
이후 각 요소들을 계산하기 위해 필요한 입력 값, 즉 입력 변수에 대해 고민 후 설계를 하게 되었다.

자동차 유지비 계산기
1. 보험료, 차량가액, 세금, 기타 유지비에 관한 계산을 각각의 함수를 설정해서 최종 결과값으로는 각각의 요소로 구해진 금액을 월간과 연간 유지비로 합산해서 도출
- 2. 보험료는 운전자의 나이와 차량의 가격, 연간 주행 거리를 입력받아 계산한다.
- 3. 차량 가액은 차량의 가격과 선납금, 할부 개월 수를 입력 받아 계산한다.
- 4. 기타 유지비 중 유류비는 1년 키로수를 입력 받아 12로 나눠 현재 가솔린 또는 디젤의 가격을 곱해 월 유류비를 구한다.

5. 마지막으로 세금은 차량의 가격을 입력 받아 *0.05를 해서 개별 소비세를 계산하고 개별소비세에 *0.3을 계산해서 교육세를 구해주고, 차량가격의 7%인 취등록세를 구해주고 모두 더해준다.

이렇게 구해진 개별소비세 교육세 자동차세를 모두 더해서 월간과 연간 세금에 대한 결과값을 추출한다.

설계 후 각 요소들을 각각 계산할 수 있는 계산기를 먼저 코드를 짜고
각각의 계산기의 요소들을 가져와 총 비용을 계산 할 수 있는 자동차 유지비 계산기를 완성 시켰다.

b. 각 함수는 어떻게 동작하는 지 구체적으로 설명

get_car_price(self): 사용자로부터 차량 가격을 입력받고, 숫자가 아니라면 에러 메시지를 출력한다. 입력값을 숫자로 변환하여 클래스 속성에 저장한다.

get_driver_age(self): 사용자로부터 운전자의 나이를 입력받아 클래스 속성에 저장한다.

get_fuel_expenses(self): 연료 유형, 연간 주행 거리, 연비를 사용자로부터 입력받고, 숫자가 아니라면 에러 메시지를 출력한다. 입력 값을 숫자로 변환하여 클래스 속성에 저장한다.

get_installment_payment(self): 선납금과 할부 기간을 사용자로부터 입력받고, 숫자가 아니라면 에러 메시지를 출력한다. 입력값을 숫자로 변환하여 클래스 속성에 저장한다.

calculate_insurance_premium(self): 보험료를 계산하는 메서드로, 차량 가격, 운전자 나이에 따라 기본 보험료와 추가 요금을 계산한다.

calculate_fuel_expenses(self): 연료 비용을 계산하는 메서드로, 연료 유형에 따라 연간 및 월간 연료 소비량과 비용을 계산한다.

calculate_installment_payment(self): 할부 이자 및 월간 상환액을 계산하는 메서드로, 대출액, 연이자율 등을 고려하여 월간 상환액을 계산한다.

calculate_taxes(self): 차량에 대한 세금을 계산하는 메서드로, 소비세, 교육세, 등록세를 고려하여 총 세금을 계산한다.

calculate_total_expenses(self): 총 유지비를 계산하는 메서드로, 보험료, 연료비, 할부 이자, 세금 등을 고려하여 월간 및 연간 유지비를 계산한다.

c. 에러 발생 지점

1. 숫자 대신 문자열을 입력했을 때 발생한 에러
2. 숫자를 입력할 부분에서 문자열을 입력했을 때 발생한 에러

d. 에러 발생에 대한 해결책

위 에러 발생의 경우 입력 값이 올바른 유형인지 확인하는 조건문을 삽입하여
올바른 유형이 아닌 경우 "입력 값은 숫자여야 합니다", "유효하지 않은 연료입니다" 등
안내문구를 출력해주는 해결책을 사용하였다.

e. 해결책 적용 시 어떻게 변화

1. 입력 값을 잘 못 넣었을 경우 결과값을 출력하는 과정에서 에러가 뜨면서 코드 자체의 동작이 멈추게 되었지만 해결책을 사용한 결과 다시 입력 값을 입력하게 되면서 계산기의 작동이 반복되게 되었다.

f. 동작 결과 캡쳐

```python
# 자동차 유지비 계산기 (신차 출고 기준)
class CarExpensesCalculator:
    def __init__(self):
        self.car_price = 0
        self.driver_age = 0
        self.fuel_type = ""
        self.annual_mileage = 0
        self.fuel_efficiency = 0
        self.down_payment = 0
        self.loan_duration = 0
```

```
01.Homework × 연습장2 × 01.Homework × 보험료 계산 ×
차량의 가격을 입력하세요 (백만원 단위로 예: 5000만원 -> 50 ): 83
운전자의 나이를 입력하세요: 32
차량의 연료 유형을 입력하세요 (가솔린 또는 디젤): 가솔린
연간 주행 예상 거리를 입력하세요 (km 단위로): 15000
차량의 평균 연비를 입력하세요 (km/l 단위로): 21
선납금을 입력하세요 (백만원 단위로): 23
할부 개월을 입력하세요: 36

=== 최종 결과 ===
월간 유지비: 2004629.46 원
연간 유지비: 24055553.50 원
```

입력 사항

차량의 가격 :8300 만원

운전자의 나이: 32 세

차량 연료 유형: 가솔린

연간 예상 주행 거리: 15,000km

평균 연비: 21km/L

선납금: 2300 만원

할부 개월: 36 개월

결과값

월간 유지비: 200 만 4629 원

연간 유지비: 2405 만 5553 원

5. 계산기 개발 후기

a. 계산기 개발 후 느낀 점 설명

사실 계산기를 설계하면서 차를 좋아한다는 이유로 자동차 유지비를 계산하게 되었는데 고려할 사항이 많아 쉽지 않게 느껴졌다.

그럼에도 불구하고, 자동차의 각각의 요소들을 계산할 수 있는 계산기를 하나씩 먼저 만들어보면서 잘못된 부분들에 대한 수정을 통해 새롭게 배우게 된 점도 많았으며, 차량 유지비에 대해서도 한 층 더 알게 되었다.

또한 쉽지 않은 코딩이라고 느껴졌지만 하나씩 해결해 나가면서 희열을 느낄 수 있었다.