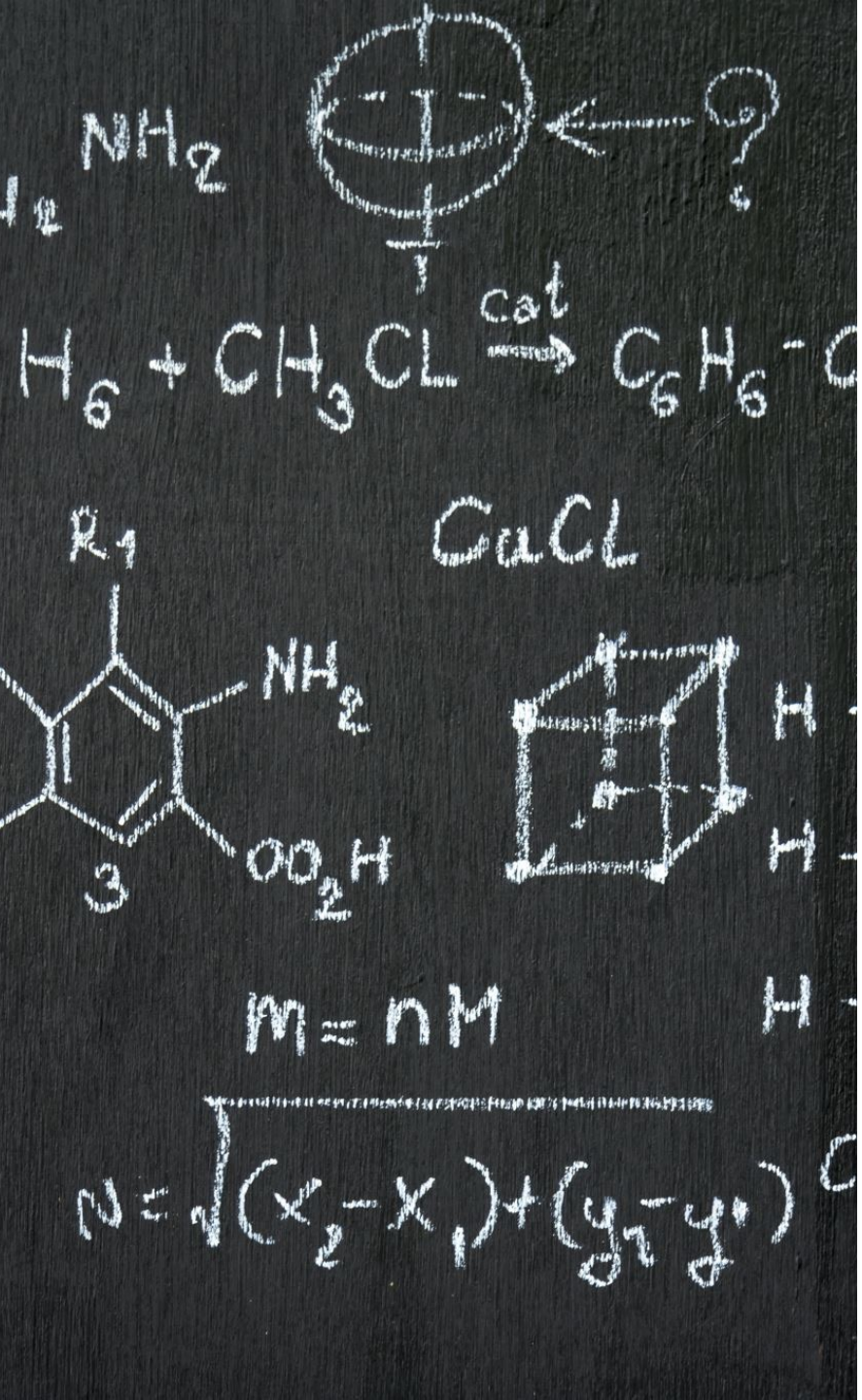
What does the script do?

Given a loan amount, repayment frequency, loan term range and interest rate range declared by the user, the script will output how much the repayments would be for the given ranges.

Once printed it will ask if the user would also like a csv version of the output and export this to the users desktop.



How the script structure is built

- The script is broken into 5 files which will individually be covered.
- There is a 6th script (validation.py) which exists for testing purposes in the 2 test scenarios
- The final 2 scripts are the tests:
  - test_calculations.py which asserts the mathematical formula is working based off the file calculation.py
  - test_input.py which assets the input validation is working based where incorrect inputs are declared.

src > calculations.py > RepaymentCalculator > calculate_loan_repayments

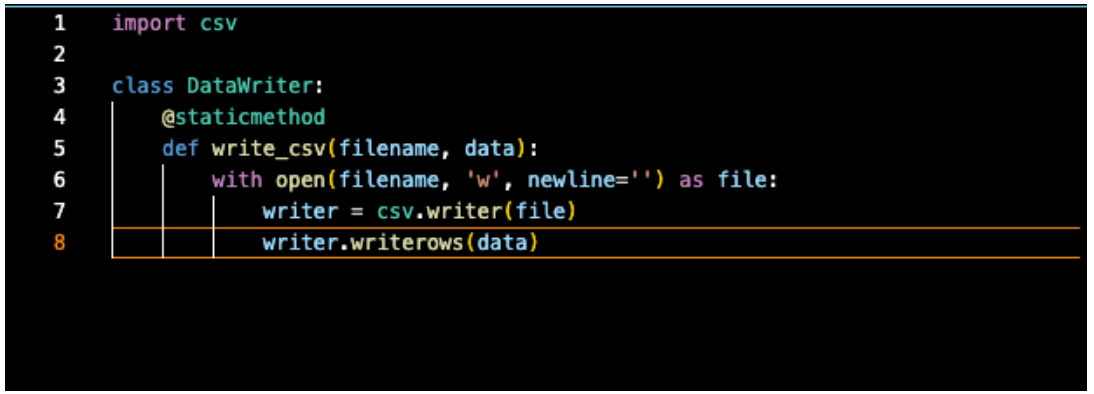
```python
class RepaymentCalculator:
    # Initialise loan with loan details
    def __init__(self, loan_details):
        self.loan_details = loan_details

    #Function to calculate the repayments for a loan.
    def calculate_loan_repayments(self):
        #Calculate the interest rate per repayment period (Done by dividing annual interest rate by the number of payments per ye
        rate = self.loan_details.interest_rate / (100 * self.loan_details.num_payments)
        #Total repayments for the loan term.
        repayments = self.loan_details.loan_term * self.loan_details.num_payments
        # Calculate the repayment amount for the loan.
        # The formula takes into account the interest rate per repayment period
        # and the total number of repayments over the loan term.
        return (rate * self.loan_details.loan_amount) / (1 - (1 + rate) ** -repayments)
```

Calculations.py

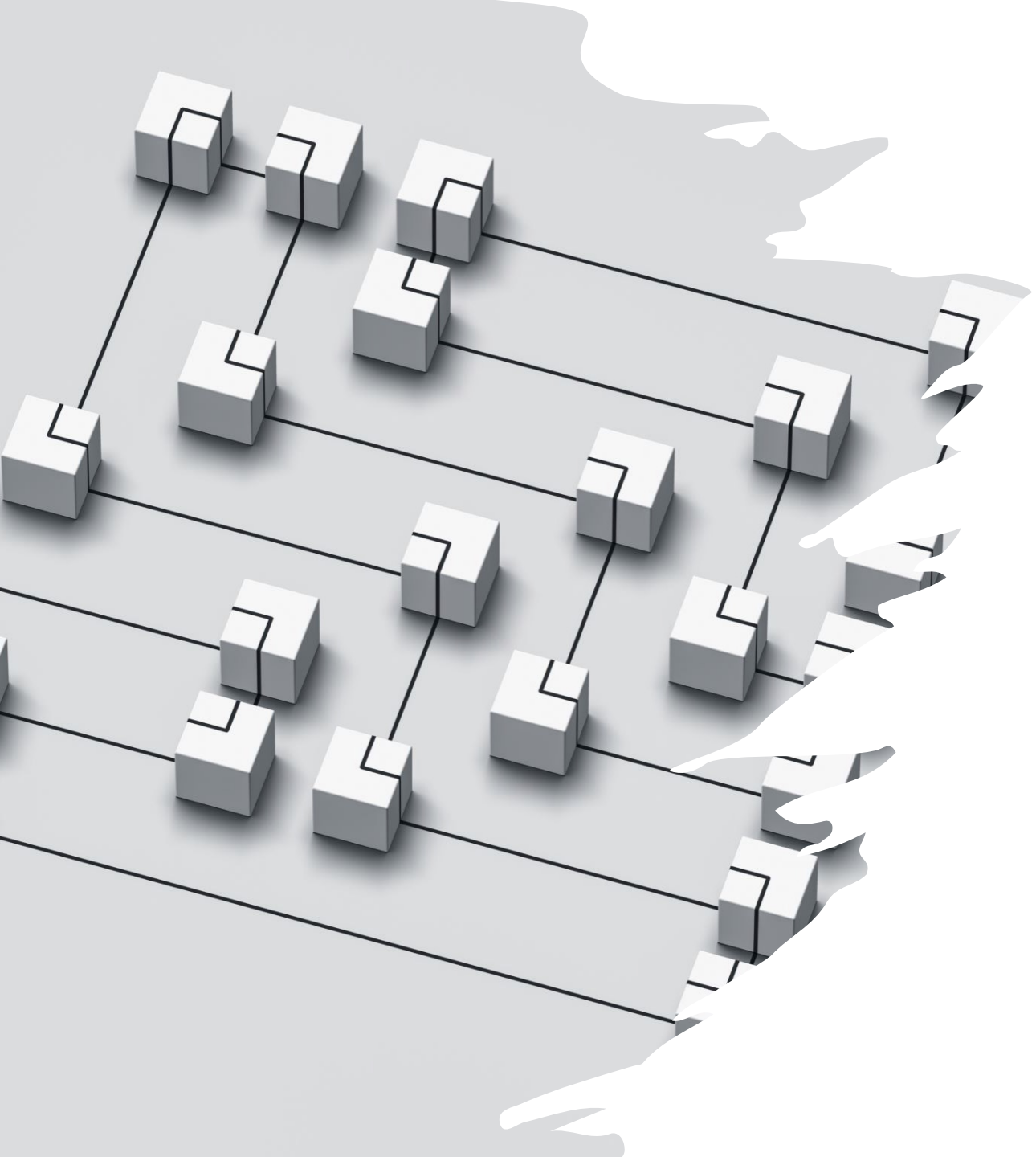
- First part of script developed, as it was necessary to first confirm that the calculation for loan repayments was correct
- We define 2 functions:
  - Init – to initialize the loan details to be used (the loan_details are ‘stored’ in a separate class and are not used directly in this script, but are imported into the main)
  - Calculate_loan_repayments – to perform the actual calculation by parsing the information in loan_details

Data_writer.py

- This was a ‘stretch’ goal, where it wasn’t necessary for the code to run, however if time allowed was an added feature
- This code imports the standard library csv for python and uses the @staticmethod to call a class without creating an instance of it.
- The code the defines how to write a csv file (for later use)



```python
import csv

class DataWriter:
    @staticmethod
    def write_csv(filename, data):
        with open(filename, 'w', newline='') as file:
            writer = csv.writer(file)
            writer.writerows(data)
```



Input.py

- The largest code block, however in some ways the most simple.
- This stores all of the user inputs as variables.
- After building the basic necessity of the code I added some additional features:
  - Applying the DRY principal I defined a function to clear the terminal, that can then be used in each other function with a single line ```self.clear_terminal()```
  - A confirm input step to display back what the user had declared and allowed for the user to start over if information was wrong – this came about when testing and being irritated that I had to go through the entire code or close the terminal if something went wrong.

Input.py

- Cont
  - An exit (and confirmation) function that allows a user to type exit at any input to leave the program early
  - Error handling to ensure that correct inputs are being declared, and if not providing feedback to the user, and allow them to try again.
- This part of the code took the shortest time to build the ‘basics’ (inputs and variables) but also the longest in refinement to handle the added features that make it more user friendly

Input.py

```python
    def get_loan_amount(self):
        self.clear_terminal()
        while True:
            try:
                loan_amount = self.get_input("Please enter the loan amount (or type 'exit' to stop the program): $")
                loan_amount = float(loan_amount)
                if loan_amount > 0:
                    return loan_amount
                else:
                    print("Loan amount must be a positive number. Please try again.")
            except ValueError:
                print("Invalid input. Please enter a number.")

    # Function to get the repayment frequency from the user
    def get_repayment_frequency(self):
        self.clear_terminal()
        while True:
            print("Select your repayment frequency:")
            print("[1] - Weekly repayments")
            print("[2] - Fortnightly repayments")
            print("[3] - Monthly repayments")
            print("Type 'exit' to quit")
            try:
                frequency = self.get_input("Enter repayment frequency (1, 2, or 3): ")
                frequency = int(frequency)
                if frequency == 1:
                    return "weekly"
                elif frequency == 2:
                    return "fortnightly"
                elif frequency == 3:
                    return "monthly"
                else:
                    print("Invalid selection. Please enter either 1, 2, or 3.")
            except ValueError:
```

```python

class UserInput:
    # Function to clear terminal
    def clear_terminal(self):
        os.system('cls' if os.name == 'nt' else 'clear')

    # Function to confirm loan details with the user
    def confirm_inputs(self, loan_amount, frequency, term_start, term_end, rate_start, rate_end):
        self.clear_terminal()
        print("Please confirm the following details:")
        print(f"Loan Amount: ${format(loan_amount, ',.2f')}")
        print(f"Repayment frequency: {frequency.capitalize()}")
        print(f"Loan term range: {term_start} years to {term_end} years")
        print(f"Interest rate range: {format(rate_start, '.2f')}% to {format(rate_end, '.2f')}%")
        while True:
            # Get confirmation from the user
            confirmation = self.get_input("Is this information correct (y/n): ")
            if confirmation.lower() == "y":
                return True
            elif confirmation.lower() == "n":
                return False
            else:
                print("Invalid input. Please enter only y or n.")

    # Function to get any user input with the option to exit program
    def get_input(self, prompt):
        while True:
            user_input = input(prompt)
            if user_input.lower() == 'exit':
                if self.confirm_exit("Are you sure you wish to exit?"):
```

Input.py

```
    #Function to get the loan term range from user
    def get_term_range(self):
        self.clear_terminal()
        while True:
            try:
                print("Input the loan term range you are testing, type 'exit' to quit")
                term_start = self.get_input("Enter the start of the loan term you would like to test (in years): ")
                term_end = self.get_input("Enter the end of the loan term you would like to test (in years): ")
                term_start = int(term_start)
                term_end = int(term_end)
                if term_start > 0 and term_end > 0 and term_end >= term_start:
                    return term_start, term_end
                else:
                    print("Invalid range. Both terms must be positive, and term end must be larger or equal to term start.")
            except ValueError:
                print("Invalid input. Please enter a number.")

    # Function to get the interest rate range from the user
    def get_interest_rate_range(self):
        self.clear_terminal()
        while True:
            try:
                print("Input the interest rate range you are testing, type 'exit' to quit")
                rate_start = self.get_input("Enter the start of the interest rate you would like to test: ")
                rate_end = self.get_input("Enter the end of the interest rate you would like to test: ")
                rate_start = float(rate_start)
                rate_end = float(rate_end)
                if rate_start > 0 and rate_end > 0 and rate_end >= rate_start:
                    return rate_start, rate_end
                else:
                    print("Invalid range. Both terms must be positive, and end rate must be larger or equal to start rate.")
            except ValueError:
                print("Invalid input. Please enter a number.")
```

```
    # Function to confirm user's request to exit the program
    def confirm_exit(self, prompt):
        while True:
            confirmation = self.get_input(prompt + " (y/n): ")
            if confirmation.lower() == "y":
                return True
            elif confirmation.lower() == "n":
                return False
            else:
                print("Invalid input. Please enter 'y' or 'n'.")
```

```python
class LoanDetails:
    def __init__(self, loan_amount, num_payments, loan_term, interest_rate):
        self.loan_amount = loan_amount
        self.num_payments = num_payments
        self.loan_term = loan_term
        self.interest_rate = interest_rate
```

Loan_details.py

- The Class creates an object that represents the loan and all the details.
- Originally I had not had a class for this function and housed it directly in ‘main.py’ however for Organisation, reuse and potential future extension I separated this into its own file and class.

Main.py

‘Where the magic happens’

Imports all of the classes from the other scripts, and some python standard libraries (pathlib and os)

This is the builder and brings everything together.

It first creates an instance or the UserInput to get all the user inputs.

It maps the repayment frequencies to numbers (to be used to calculate the number of repayments)

It also prepares the data for csv export creating an overall header and then column headers (separated by commas)

Then it performs the main function:

Main.py

- The script then loops all of the interest rates in the range declared by the user (stepping by 0.25% each time) inside a loop of each of the loan term years (stepping 1 year at a time)
- The loop performs the repayment calculator function then appends the csv row with the data declared and prints the result to terminal.
    - This uses f string to make the text visually appealing and understandable by the user instead of simply providing the output with no context.
- After the loop completes the script will ask if the user wishes to export the results. If yes it will then export a csv to the desktop and if no will close the program.

Main.py

```
    data = [
        ["For a loan of :", f"${loan_amount}"],
        ["Loan term", "Interest Rate", "Repayments", "Frequency"]
    ]

    for term in range(term_start, term_end + 1):
        for rate in range(int(rate_start * 100), int(rate_end * 100) + 1, 25):
            loan_details = LoanDetails(loan_amount, frequencies[frequency], term, rate / 100)
            repayment_calculator = RepaymentCalculator(loan_details)
            repayments = repayment_calculator.calculate_loan_repayments()
            row = [f"{term}y", f"{format(rate / 100, '.2f')}%", f"${format(repayments, ',.2f')}", frequency.capitalize()]
            data.append(row)
            print(f"For a loan term of {term} years and an interest rate of {format(rate / 100, '.2f')}%, your repayments would be ${format(repa

    while True:
        download_output = input("Would you like to export the results to your desktop (y/n)? ")
        if download_output.lower() == "n":
            print("Thank you for using this calculator")
            print("Goodbye")
            exit()
        elif download_output.lower() == "y":
            # Set the export file path to the desktop
            desktop_path = str(Path.home() / "Desktop")
            file_path = os.path.join(desktop_path, "loan_data.csv")
            # Call the write_csv method and pass the file path
            DataWriter.write_csv(file_path, data)
            print("Thank you for using this calculator")
            print("Goodbye")
            exit()
        else:
            print("Inavlid input, please list either y or n")
# Call the main function
if __name__ == "__main__":
    main()
```

```
from validation import Validation
from loan_details import LoanDetails
from data_writer import DataWriter
from pathlib import Path
import os

def main():
    # Use the UserInput class to handle all user inputs.
    user_input = UserInput()

    while True:
        # Get loan amount from user
        loan_amount = user_input.get_loan_amount()
        # Get repayment frequency from user
        frequency = user_input.get_repayment_frequency()
        # Get loan term range from user
        term_start, term_end = user_input.get_term_range()
        # Get interest rate range from user
        rate_start, rate_end = user_input.get_interest_rate_range()

        # Confirm all user inputs
        if user_input.confirm_inputs(loan_amount, frequency, term_start, term_end, rate_start, rate_end):
            break

    user_input.clear_terminal()

    # Map the frequencies to the number of repayments per year
```

Tests

- There are 2 test scripts and an additional validation script that is used in conjunction.
- These are tested using pytest
- Test_calculations works on the ‘calculations’ function to ensure the mathematic operation works as intended by parsing known loan details and repayments.
- Had to investigate the ‘abs’ function, as the output was not always 100% due to loan term, interest rates and rounding.
- Test_input uses the validation script to make sure that errors are handled correctly.

```python
from calculations import RepaymentCalculator
from loan_details import LoanDetails

def test_calculate_loan_repayments():
    # Testing with a known loan amount, term and interest rate.

    # Test 1: 5 year term with 5% interest
    loan_details_1 = LoanDetails(10000, 12, 5, 5)
    loan_1 = RepaymentCalculator(loan_details_1)
    assert abs(loan_1.calculate_loan_repayments() - 188.71) < 0.01

    # Test 2: 10 year term with 5% interest
    loan_details_2 = LoanDetails(10000, 12, 10, 5)
    loan_2 = RepaymentCalculator(loan_details_2)
    assert abs(loan_2.calculate_loan_repayments() - 106.07) < 0.01
```

```python
from validation import Validation

def test_loan_amount_validation():
    # Testing the loan amount validation
    validation = Validation()

    # Test case 1: Valid loan amount
    valid, value = validation.validate_loan_amount("10000")
    assert valid and value == 10000  # Expect True and 10000

    # Test case 2: Invalid loan amount (negative)
    valid, value = validation.validate_loan_amount("-10000")
    assert not valid and value is None  # Expect False and None

    # Test case 3: Invalid loan amount (non-numeric)
    valid, value = validation.validate_loan_amount("abc")
    assert not valid and value is None  # Expect False and None
```