Kavish Shah: Dal ID Number: B00961429

Jay Raval: Dal ID Number: B00966367

ENGM 4620 – Python for Engineers

Python Project – 1

Title: Canadian Financial Toolkit

- **INTRODUCTION**

  - The project aims to inform users about how money/taxes work in Canada through Canadian banking accounts. It provides information on Tax-Free Savings Accounts (TFSA), First Home Savings Accounts (FHSA), Registered Retirement Savings Plan (RRSP), calculating taxes for the year 2023, and estimating the time required to earn a specific amount of money.

  - **main.py**
    - Acts as the main file for the project.
    - Handles user interactions and imports necessary modules.

  - **seperator() function**
  - This function enhances readability by printing a separator line.

```python
def seperator():
    """Prints a separator line to enhance readability."""
    print("-" * 110)
```

  - **General Information Display**
    - Upon execution, the program displays general information about the project.

```
--------------------------------------------------------------------------------------------------------------
                                Canadian Financial Toolkit
This project aims to inform users about how money/taxes works in Canada through Canadian banking accounts.
It provides information on Tax-Free Savings Accounts (TFSA), First Home Savings Accounts (FHSA),
Registered Retirement Savings Plan (RRSP), calculating taxes for the year 2023,
and estimating the time required to earn a specific amount of money.
--------------------------------------------------------------------------------------------------------------
```

  - **User Interaction**
    - Users are prompted to select an option from 1 to 7.

```
--------------------------------------------------------------------------------------------------------------
1 - Check your Tax Free Savings Account (TFSA) status
2 - Calculate the tax you will pay for 2023
3 - Calculate how many years it will take for you to earn a millionaire/any amount
4 - Know about First Home Saving Account (FHSA) account and the amount of tax you will save by investing in it
5 - Know about Registered Retirement Savings Plan (RRSP)
6 - Detailed information on various Canadian Bank accounts
7 - To exit the program
--------------------------------------------------------------------------------------------------------------
```

- **Option Validation**
  - The program ensures that the user enters a valid option within the range of 1 to 7.
  - It utilizes a while loop, error handling, and nested if-else statements for robust validation. Input outside this range is discarded and the user is asked to renter.

```
    while True:
        print("\n\n\n")
        for x in range(3):
            print("...", end=" ")
            time.sleep(0.6)
        print()
        seperator()
        # Displays the options menu
        for key, value in options.items():
            print("{} - {}".format( *args: key, value))
        seperator()

        try:
            # Prompt user for choice
            user_choice = int(input("\nPlease select an option from the above list: "))

            if user_choice == 7:
                print()
                seperator()
                print("Bye Bye. See you next time!")
                seperator()
                break  # Exit the loop if the user chooses to exit

            elif user_choice in options:

                # Perform action based on user's choice
                if user_choice == 1:
                    print()
                    tfsa_func()  # Call function to check TFSA status
                elif user_choice == 2:
                    print()
                    taxes_func()  # Call function to calculate taxes for 2023
                elif user_choice == 3:
                    print()
                    millionaire_func()  # Call function to calculate time to become a millionaire
                elif user_choice == 4:
                    print()
                    eligible = is_eligible_for_fhsa()
                    calculate_fhsa_investment(eligible)
                elif user_choice == 5:
                    print()
                    rrsp_func()  # Call function to calculate time to become a millionaire
                elif user_choice == 6:
                    print()
                    info_func()  # Call function to calculate time to become a millionaire
                else:
                    print("Nothing")

            else:
                print("Invalid option. Please select a valid number.\n")
        except ValueError:
            print("Invalid input. Please enter a number.")

```

- **TAX FREE SAVINGS ACCOUNT (TFSA)**

Upon selecting option ***1***, the program initiates the functionality related to the Tax-Free Savings Account (TFSA).

- **Data Structures Utilized**
  - status Dictionary
    - ❖ This dictionary contains information about living status in Canada.
    - ❖ It helps determine the user's residency status, which is crucial for TFSA contributions and withdrawals.

```
# Dictionary containing the status in Canada
status = {
    1: "Canadian Citizen",
    2: "Permanent Resident",
    3: "Temporary Resident (international student, foreign worker)",
    4: "Refugee or Protected Person",
    5: "Visitor (on visitors visa)"
}
```

  - contribution Dictionary
    - ❖ The contribution dictionary maps years to their corresponding contribution limits set by the Canadian Government for TFSA accounts.

```
contribution = {
    2009: 5000, 2010: 5000, 2011: 5000, 2012: 5000,
    2013: 5500, 2014: 5500, 2015: 10000, 2016: 5500,
    2017: 5500, 2018: 5500, 2019: 6000, 2020: 6000,
    2021: 6000, 2022: 6000, 2023: 6500, 2024: 7000,
}
```

- **delay() Function.**
  - The main aim of the **printing_dict()** function is to delay the output by 2.25 seconds, allowing the user time to identify the last output of the code and creating a sense of suspense.

- **printing_dict() Function**
  - The printing_dict() function serves a vital purpose in managing user-specific data associated with TFSA contributions. Its primary objective is to print the key (year) value (contribution) associated with the individual.
    - ❖

```
def printing_dict():
    """
    The function aims to print the year, its respective contribution and the gross total of TFSA
    depending on individual's status in Canada.
    """
    global total_contr_limit, contribution
    seperator()
    print("\nFetching your TFSA contribution data", end=" ")
    delay()
    print("\n")
    print("Year, Contribution")
    for keys, values in contribution.items():
        print("{0}:  $ {1:,}".format( *args: keys, values))
    values = contribution.values()
    for val in list(values):
        total_contr_limit += val

    print("\nYour maximum/total contribution limit is: $ {0:,}".format(total_contr_limit))
    print("You can enjoy the TAX FREE dividend/interest gains by investing $ {0:,} in your TFSA account"
          .format(total_contr_limit))
```

- **Implementation Details**
  - The function iterates through the status dictionary to identify the user's residency status.
  - Based on the residency status, it filters out irrelevant contribution limit data from the contribution dictionary.

```
if status_input == 1:

    try:
        age = int(input("Enter your age: "))
    except ValueError:
        print("\nFollow instructions properly. See ya next time.")

    if age < 18:
        print()
        seperator()
        print("Come back when you are 18 or above to open the TFSA account.")
        seperator()

    elif 18 <= age <= 120:
        for year in range((2024 + (17 - age)), 2008, -1):
            contribution.pop(year)
        printing_dict()
    else:
        print("\n\n")
        seperator()
        print("Kindly enter a valid age next time. See ya!")
        seperator()
```

```
elif status_input == 2 or status_input == 3 or status_input == 4:
    try:
        year_landed = int(input("Enter the year in which you landed in Canada: "))
    except ValueError:
        print("\nFollow instructions properly. See ya next time.")

    if 2009 <= year_landed <= 2024:
        for year in range(year_landed - 1, 2008, -1):
            contribution.pop(year)
        printing_dict()
    elif 1900 < year_landed < 2009:
        printing_dict()
    else:
        print()
        seperator()
        print("Do not try to be over smart! Enter appropriate year")
        seperator()
```

- This ensures that the user is presented with accurate and relevant information tailored to their residency status in Canada.

```
------------------------------------------------------------------------------------------------------------
Choose from the following to check your eligibility status for TFSA:

1 : Canadian Citizen
2 : Permanent Resident
3 : Temporary Resident (international student, foreign worker)
4 : Refugee or Protected Person
5 : Visitor (on visitors visa)
------------------------------------------------------------------------------------------------------------

Enter the number that associates with your living condition in Canada (between 1 and 5): 1
Enter your age: 23
------------------------------------------------------------------------------------------------------------

Fetching your TFSA contribution data ...

Year, Contribution
2019:  $ 6,000
2020:  $ 6,000
2021:  $ 6,000
2022:  $ 6,000
2023:  $ 6,500
2024:  $ 7,000

Your maximum/total contribution limit is: $ 37,500
You can enjoy the TAX FREE dividend/interest gains by investing $ 37,500 in your TFSA account
------------------------------------------------------------------------------------------------------------
```

- **TAXES**

Data Structures Utilized

1. federal_tax Dictionary
    - This dictionary contains tax percentage information for the federal level.
    - Each key represents a tax percentage, with a corresponding value as a list of two numbers indicating the lower and higher income levels.

```
# Define tax information dictionaries
federal_tax = {
    0: [0, 15000],
    15.0: [15001, 53359],
    20.5: [53360, 106717],
    26.0: [106718, 165430],
    29.0: [165431, 235675],
    33.0: [235676, float('inf')]  # Assume no upper limit for the highest bracket
}
```

2. provincial_tax Dictionary
    - The provincial_tax dictionary contains tax percentage information for provincial levels.
    - It is structured as a dictionary of dictionaries, where each province maps tax percentages to income brackets for provincial taxation.

```
provincial_taxes = {
    "Newfoundland and Labrador": {
        0: [0, 10382],
        8.7: [10382, 41457],
        14.5: [41458, 82913],
        15.8: [82914, 148027],
        17.8: [148028, 207239],
        19.8: [207240, 264750],
        20.8: [264751, 529500],
        21.3: [529501, 1059000],
        21.8: [1059001, float('inf')]
    },
    "Prince Edward Island": {
        0: [0, 11250],
        9.8: [11250, 31984],
        13.8: [31985, 63969],
        16.7: [63970, float('inf')]
    },
    "Quebec": {
        0: [0, 17183],
        15: [17183, 49276],
        20: [49277, 98542],
        24: [98543, 119910],
        25.75: [119910, float('inf')]
    },
    "Nova Scotia": {
        0: [0, 11481],
        8.79: [11481, 29590],
        14.95: [29590, 59180],
        16.67: [59180, 93000],
        17.5: [93000, 150000],
        21: [150001, float('inf')]
    }
```

Refer the code to see the complete dictionary

3. province_list() Function

- This function prints the list of provinces in Canada, enabling users to select their province for tax calculation.

```
def province_list():
    """ Printing the list of provinces in Canada """
    province_names = list(provincial_taxes.keys())

    seperator()
    print("Province Names: \n")
    for province_ in province_names:
        print("- {}".format(province_))
    seperator()
```

Person Class

The Person class encapsulates the logic for tax calculations based on the provided province and net income.

Class Functions

1. federal_tax_calc()
    - Calculates the federal tax based on the provided net income and federal tax rates.

```
def federal_tax_calc(self, print_=True):
    if print_:
        seperator()
        print("Calculating Federal Taxes: ", end=" ")
        print("\n\n")

    federal_tax_amount = 0
    for tax_rate, income_range in federal_tax.items():
        if self.income > income_range[1]:
            federal_tax_amount += (income_range[1] - income_range[0]) * (tax_rate / 100)
            individual_fed_tax = (income_range[1] - income_range[0]) * (tax_rate / 100)
            if print_:
                print("Tax : {0:5} %, Income Range: {1:16}, Federal Tax: {2:<11,.2f}".
                      format( *args: tax_rate, str(income_range), individual_fed_tax))
        else:
            federal_tax_amount += (self.income - income_range[0]) * (tax_rate / 100)
            individual_fed_tax_2 = (self.income - income_range[0]) * (tax_rate / 100)
            if print_:
                print("Tax : {0:5} %, Income Range: {1:16}, Federal Tax: {2:<11,.2f}".
                      format( *args: tax_rate, str(income_range), individual_fed_tax_2))
                print("\nFederal tax for 2023 at income of ${:,.2f} is: ${:,.2f}"
                      .format( *args: self.income, federal_tax_amount))
            return federal_tax_amount
```

2. provincial_tax_calc()

- Calculates the provincial tax based on the provided net income and provincial tax rates specific to the province selected.

```
def provincial_tax_calc(self, print_=True):
    if print_:
        seperator()
        print("Calculating Provincial Taxes: ", end=" ")
        print("\n\n")

    provincial_tax_amount = 0
    if self.province in provincial_taxes:
        for tax_rate, income_range in provincial_taxes[self.province].items():
            if self.income > income_range[1]:
                provincial_tax_amount += (income_range[1] - income_range[0]) * (tax_rate / 100)
                individual_prov_tax = (income_range[1] - income_range[0]) * (tax_rate / 100)
                if print_:
                    print("Tax : {0:5} %, Income Range: {1:16}, Provincial Tax: {2:<11,.2f}".
                          format( *args: tax_rate, str(income_range), individual_prov_tax))
            else:
                provincial_tax_amount += (self.income - income_range[0]) * (tax_rate / 100)
                individual_prov_tax_2 = (self.income - income_range[0]) * (tax_rate / 100)
                if print_:
                    print("Tax : {0:5} %, Income Range: {1:16}, Provincial Tax: {2:<11,.2f}".
                          format( *args: tax_rate, str(income_range), individual_prov_tax_2))
                    print("\nProvincial tax for 2023 at income of ${:,.2f} is: ${:,.2f}"
                          .format( *args: self.income, provincial_tax_amount))
                    seperator()
                return provincial_tax_amount
```

3. comparison()

- Compares the taxes of different provinces based on the same salary, enabling users to make informed decisions regarding their tax liabilities.

```
def comparison(self):

    province_comparison = {}
    feds_tax = self.federal_tax_calc(print_=False)

    for province_ in provincial_taxes:
        provincial_tax_amount_ = 0
        for tax_rate_, income_range_ in provincial_taxes[province_].items():
            if income > income_range_[1]:
                provincial_tax_amount_ += (income_range_[1] - income_range_[0]) * (tax_rate_ / 100)
            else:
                provincial_tax_amount_ += (income - income_range_[0]) * (tax_rate_ / 100)
                break
        province_comparison[province_] = income - (provincial_tax_amount_ + feds_tax)

    # Sort the dictionary in descending order based on values
    sorted_province_comparison = dict(
        sorted(province_comparison.items(), key=lambda item: item[1], reverse=True))

    # Print the sorted dictionary
    seperator()
    print("With annual income of: {:,.2f}, "
          "you will be left with the following money with respect to the province: \n".
          format(income))
    print("Province                 |  Money left after taxes:")
    for province__, money_left in sorted_province_comparison.items():
        print("{0:25} |  {1:,.2f}".format( *args: province__, money_left))
    seperator()
```

taxes_func() Function

The taxes_func() function orchestrates the tax calculation process by soliciting user input for income and province. Subsequently, it invokes the relevant functions within the Person class to calculate federal and provincial taxes and facilitates tax comparisons across provinces.

Additional Feature

At the end of the code execution, the program prompts the user whether they want to watch a YouTube video explaining how taxes are calculated. If the user agrees, the video opens in the default browser, providing further insights into tax calculation processes.

```
-----------------------------------------------------------------------------------------------------------
Calculating Federal Taxes:


Tax :     0 %, Income Range: [0, 15000]      , Federal Tax: 0.00
Tax :  15.0 %, Income Range: [15001, 53359]  , Federal Tax: 5,753.70
Tax :  20.5 %, Income Range: [53360, 106717] , Federal Tax: 1,361.20

Federal tax for 2023 at income of $60,000.00 is: $7,114.90
...


-----------------------------------------------------------------------------------------------------------
Calculating Provincial Taxes:


Tax :     0 %, Income Range: [0, 11481]      , Provincial Tax: 0.00
Tax :  8.79 %, Income Range: [11481, 29590]  , Provincial Tax: 1,591.78
Tax : 14.95 %, Income Range: [29590, 59180]  , Provincial Tax: 4,423.70
Tax : 16.67 %, Income Range: [59180, 93000]  , Provincial Tax: 136.69

Provincial tax for 2023 at income of $60,000.00 is: $6,152.18
-----------------------------------------------------------------------------------------------------------
...


Income                : $ 60,000.00
Total Tax paid        : $ 13,267.08
Money left after taxes : $ 46,732.92

Please note that the above calculations does not include CPP (Canada Pension Plan) calculations.
-----------------------------------------------------------------------------------------------------------
```



```
Would you like to see how much tax people pay with your salary in other provinces? [Y/N]: Y
-----------------------------------------------------------------------------------------------------------
With annual income of: 60,000.00, you will be left with the following money with respect to the province:

Province                  |  Money left after taxes:
Nunavut                   |  50,928.41
British Columbia          |  50,076.55
Ontario                   |  50,012.75
Northwest Territories     |  50,008.95
Yukon                     |  49,832.50
Alberta                   |  48,938.20
New Brunswick             |  47,770.30
Newfoundland and Labrador |  47,492.99
Manitoba                  |  47,125.86
Prince Edward Island      |  46,987.10
Nova Scotia               |  46,732.92
Saskatchewan              |  46,379.50
Quebec                    |  45,926.55
-----------------------------------------------------------------------------------------------------------
```

- **MILLIONAIRE**

Upon selecting option 3, the program initiates the functionality related to calculating the timeframe required to achieve a specific amount of money.

Functionality Overview

The core functionality is encapsulated within the millionaire_func() function, which guides users through the process of determining the timeframe needed to reach their financial goal.

millionaire_func() Function

The millionaire_func() function executes the following steps:

1. User Input Collection
    - Prompts the user to input their annual investment and the desired target amount of money.

```
while True:
    try:
        annual_investment = int(input("\nEnter annual investment amount: "))
        break  # Exit the loop if input is valid
    except ValueError:
        print("Invalid input. Please enter an integer.")

while True:
    try:
        goal = int(input("Enter the goal that you want to achieve. E.g: 1000000: "))
        break  # Exit the loop if input is valid
    except ValueError:
        print("Invalid input. Please enter an integer.")
```

```
Enter annual investment amount: 36000
Enter the goal that you want to achieve. E.g: 1000000: 1000000
```

2. CSV File Generation
    - Creates a CSV file named investment_details.csv.
    - Writes columns including 'Years', 'Year', 'Investment/year', 'Interest Earned', 'Fund', 'Inflation Adjusted Fund', and populates them with the respective data.
3. Data Display

- Reads the generated CSV file and displays the investment details on the console for user review and analysis.

```
ann_inv = annual_investment

# Initialize variables
fund = annual_investment
serial_number = 1
years = 0

# Create and open CSV file for writing
with open('investment_details.csv', mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow([' Years', 'Year', 'Investment/year', 'Interest Earned', 'Fund', 'Inflation Adjusted Fund'])

    while fund < goal:
        interest_earned = fund * interest_rate
        fund += interest_earned

        # Adjust for inflation
        inflation_adjusted_fund = fund / (1 + inflation_rate) ** (current_year - 2023)

        writer.writerow([serial_number, current_year, '{:.2f}'.format(initial_investment + annual_investment),
                         '{:.2f}'.format(interest_earned), '{:.2f}'.format(fund),
                         '{:.2f}'.format(inflation_adjusted_fund)])

        if (current_year - 2024) % 5 == 0:
            annual_investment += extra_investment

        fund += annual_investment

        initial_investment = 0
        current_year += 1
        years += 1
        serial_number += 1
```

```
Enter annual investment amount: 36000
Enter the goal that you want to achieve. E.g: 1000000: 1000000
-----------------------------------------------------------------------------------------------
It will take 15 years to make $ 1,000,000.00 by investing 36,000.00 annually at 8.0%. ...
-----------------------------------------------------------------------------------------------
  Years    Year   Investment/year  Interest Earned        Fund       Inflation Adjusted Fund
   1       2024      36,000.00         2,880.00        38,880.00            37,703.65
   2       2025      36,500.00         6,030.40        81,410.40            76,558.62
   3       2026      36,500.00         9,432.83       127,343.23           116,130.73
   4       2027      36,500.00        13,107.46       176,950.69           156,487.86
   5       2028      36,500.00        17,076.06       230,526.75           197,700.06
   6       2029      36,500.00        21,362.14       288,388.89           239,839.71
   7       2030      37,000.00        26,031.11       351,420.00           283,417.16
   8       2031      37,000.00        31,073.60       419,493.60           328,081.77
   9       2032      37,000.00        36,519.49       493,013.08           373,914.50
   10      2033      37,000.00        42,401.05       572,414.13           420,999.23
   11      2034      37,000.00        48,753.13       658,167.26           469,422.96
   12      2035      37,500.00        55,653.38       751,320.64           519,649.45
   13      2036      37,500.00        63,105.65       851,926.29           571,405.30
   14      2037      37,500.00        71,154.10       960,580.40           624,788.54
   15      2038      37,500.00        79,846.43     1,077,926.83           679,901.06

PS: After every 5 years, an extra $500 amount is added to the investment to compensate inflation
The above data is added in the CSV file for you!
-----------------------------------------------------------------------------------------------
```

First Home Savings Account (FHSA)

Upon selecting option 4, the program initiates the functionality related to the First Home Savings Account (FHSA).

Functionality Overview

The FHSA functionality consists of two main components:

1. **is_eligible_for_fhsa() Function.**
    - This function determines the eligibility of an individual to open an FHSA account based on specific criteria.
    - Criteria include age, possession of a valid SIN number, and ownership of a qualifying home within the past 4 years.

```
while True:
    try:
        age = int(input("\n1) How old are you?: "))
        if age < 18:
            break
        else:
            break
    except ValueError:
        print("Invalid input. Please enter a valid age (integer number).")

# Validate SIN number format
while True:
    sin_number = input("2) Do you have a valid SIN number (Y/N)? ").upper()
    if sin_number not in ("Y", "N"):
        print("\nInvalid input. Please answer 'Y' for yes or 'N' for no.")
    else:
        break

# Case-insensitive conversion for qualifying ownership of home
while True:
    has_qualifying_home = input(
        "3) Have you or your spouse/common-law partner owned a qualifying home in the past 4 years (Y/N)? ").upper()
    if has_qualifying_home not in ("Y", "N"):
        print("Invalid input. Please answer 'Y' for yes or 'N' for no.")
    else:
        break

# Eligibility check based on age and SIN requirements
if age >= 18 and sin_number == "Y" and has_qualifying_home == "N":
    return True
else:
    return False
```

2. **calculate_fhsa_investment() Function.**
    - Calculates the amount of time required to fill the maximum contribution limit of $40,000 for the FHSA account.
    - Determines the amount of tax saved by investing in the FHSA account, leveraging the tax calculation functionality.

```python
def calculate_fhsa_investment(eligible_or_not):

    """Calculates recommended annual FHSA investment based on eligibility.
    Args:
        eligible_or_not (bool): Whether the user is eligible for FHSA.
    Returns:
        None if not eligible, otherwise prints message with recommendation.
    """

    if not eligible_or_not:
        seperator()
        print("We are sorry, you are not eligible to open an FHSA account.\n")
        print("To be eligible to open FHSA account, you must be:")
        print("1) 18 or above")
        print("2) holding a valid SIN number")
        print("3) not have owned a qualifying home in the past 4 years")
        seperator()
        return

    seperator()
    print("Congrats, you are eligible to open FHSA account.")
    seperator()

    # Recommend maximum contribution (case-insensitive for user input)
    print("Maximum contribution limit per year to invest in FHSA account is $8,000")
    value_ranges = {
        1: 4000,
        2: 5000,
        3: 8000
    }
    for key, value in value_ranges.items():
        print("{} - $ {}".format( *args: key, value))

    while True:
        index = input("\nPick number from 1, 2 or 3 that associates annual investment: ").upper()
        if index.isdigit() and index in ["1", "2", "3"]:  # Check if the input consists only of digits
            index = int(index)
            x = value_ranges[index]
            break  # Exit the loop if the input is successfully converted to an integer
        else:
            print("Invalid input. Please enter an integer.")

    seperator()
    print(f"Investment amount: ${value_ranges[index]} per year.\n")
    years = 1
    count = 0
    inv_per_year = value_ranges[index]
    while value_ranges[index] <= 40000:
        print("Year {0:2}, Cumulative Invested: {1:,.1f}".
              format( *args: years, value_ranges[index], 40000 % value_ranges[index]))
        value_ranges[index] += inv_per_year
        years += 1
        count += 1
```

```
143
144     seperator()
145     print("\nNow, we will calculate how much taxes you would save "
146           "while investing in FHSA account ... ...", end=" ")
147     delay()
148     print()
149
150     province_list()
151
152     province = input("\nFrom above list, enter exact spelling of your province of residence: ")
153
154     while province not in provincial_taxes:
155         print("\nKindly enter the exact spelling of Province. First letters should be capital")
156         province = input("From above list, enter exact spelling of your province of residence: ")
157
158     try:
159         income = int(input("Enter your total annual income: "))
160     except ValueError:
161         print("Please enter a valid income next time.")
162         exit()
163
164     person1 = Person(province, income)
165     fd_tax = person1.federal_tax_calc(print_=False)
166     pv_tax = person1.provincial_tax_calc(print_=False)
167     total_tax = pv_tax + fd_tax
168     seperator()
169     print("Tax paid WITHOUT FHSA investment for {0} years : $ {1:,.2f}".format( *args: count, total_tax * count))
```

```
170
171     if value_ranges[index] < person1.income:
172         person1.income -= x
173         fd_tax_2 = person1.federal_tax_calc(print_=False)
174         pv_tax_2 = person1.provincial_tax_calc(print_=False)
175         total_tax_2 = pv_tax_2 + fd_tax_2
176         print("Tax paid WITH FHSA investment for {0} years    : $ {1:,.2f}".format( *args: count, total_tax_2 * count))
177         seperator()
178         print("So, by investing $ {0:,.2f} for {1} years, you will save $ {2:,.2f} in taxes".
179               format( *args: value_ranges[index], count, (total_tax - total_tax_2) * count))
180         seperator()
181         print()
182     else:
183         print("Not possible to invest more than your income ... ...")
184
```

is_eligible_for_fhsa() Function

The **is_eligible_for_fhsa()** function assesses the eligibility of an individual to open an FHSA account by evaluating three criteria:

1. **Age**: Ensures the individual meets the minimum age requirement.
2. **Possession of Valid SIN**: Validates whether the individual holds a valid Social Insurance Number (SIN).
3. **Qualifying Home Ownership**: Determines if the individual or their spouse/common-law partner owned a qualifying home within the past 4 years.

calculate_fhsa_investment() Function

The **calculate_fhsa_investment()** function performs the following tasks:

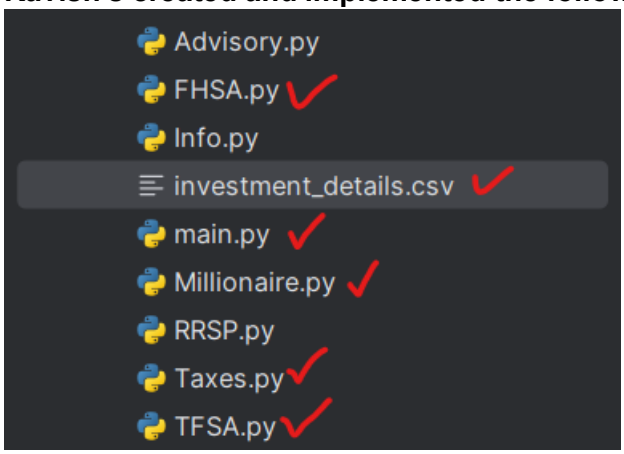
1. **Timeframe Calculation**: Determines the amount of time required to reach the maximum contribution limit of $40,000 for the FHSA account.
2. **Tax Savings Calculation**: Utilizes the tax calculation functionality to compute the amount of tax saved by investing in the FHSA account.

```
--------------------------------------------------------------------------------------------------------------
Congrats, you are eligible to open FHSA account.
--------------------------------------------------------------------------------------------------------------
Maximum contribution limit per year to invest in FHSA account is $8,000
1 - $ 4000
2 - $ 5000
3 - $ 8000

Pick number from 1, 2 or 3 that associates annual investment: 3
--------------------------------------------------------------------------------------------------------------
Investment amount: $8000 per year.

Year  1, Cumulative Invested: 8,000.0
Year  2, Cumulative Invested: 16,000.0
Year  3, Cumulative Invested: 24,000.0
Year  4, Cumulative Invested: 32,000.0
Year  5, Cumulative Invested: 40,000.0
--------------------------------------------------------------------------------------------------------------

Now, we will calculate how much taxes you would save while investing in FHSA account ... ... ...
--------------------------------------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------------------------------------

From above list, enter exact spelling of your province of residence: Nova Scotia
Enter your total annual income: 60000
--------------------------------------------------------------------------------------------------------------
Tax paid WITHOUT FHSA investment for 5 years : $ 66,335.40
Tax paid WITH FHSA investment for 5 years    : $ 52,459.63
--------------------------------------------------------------------------------------------------------------
So, by investing $ 48,000.00 for 5 years, you will save $ 13,875.77 in taxes
--------------------------------------------------------------------------------------------------------------
```

Till the above code is created by Kavish Shah

Kavish's created and implemented the following files:



All the below code is created by Jay Raval

- **Registered Retirement Savings Plan (RRSP)**

Upon selecting option ***5***, the program initiates the functionality related to Registered Retirement Savings Plan.

- **Import Libraries**
  - The code begins by importing necessary libraries
  - ***random*****:** Used for generating random investment return rates.
  - ***get_close_matches*** from ***difflib:*** Utilized for suggesting similar province names when a user input does not match exactly.

```
import random
from difflib import get_close_matches
```

- **Defining Tax Brackets and Rates:**
  - Federal and provincial tax brackets and rates are defined as **lists** and **dictionaries**, respectively.
  - Federal tax brackets are defined with corresponding income thresholds and tax rates.
  - Provincial tax rates are stored in a dictionary with provinces as keys and lists of **tuples** containing income thresholds and tax rates as values.

```
# Federal tax brackets
federal_tax_brackets = [
    (55867, 0.15),
    (111733, 0.205),
    (173205, 0.26),
    (246752, 0.29),
    (float('inf'), 0.33)
]
```

```
# Provincial tax rates
provincial_tax_rates = {
    "alberta": [(142292, 0.1), (170751, 0.12), (227668, 0.13), (34150
    "british columbia": [(45654, 0.0506), (91310, 0.077), (104835, 0.
    "manitoba": [(36842, 0.108), (79625, 0.1275), (float('inf'), 0.17
    "new brunswick": [(47715, 0.094), (95431, 0.14), (176756, 0.16),
    "newfoundland and labrador": [(41457, 0.087), (82913, 0.145), (14
    "northwest territories": [(48326, 0.059), (96655, 0.086), (157139
    "nova scotia": [(29590, 0.0879), (59180, 0.1495), (93000, 0.1667)
    "nunavut": [(50877, 0.04), (101754, 0.07), (165429, 0.09), (float
    "ontario": [(49231, 0.0505), (98463, 0.0915), (150000, 0.1116), (
    "prince edward island": [(31984, 0.098), (63969, 0.138), (float('
    "quebec": [(49275, 0.14), (98540, 0.19), (119910, 0.24), (float('
    "saskatchewan": [(49720, 0.105), (142058, 0.125), (float('inf'),
    "yukon": [(53359, 0.064), (106717, 0.09), (165430, 0.109), (50000
}
```

- ***Calculate_taxable_income* <u>Function:</u>**
  - The *calculate_taxable_income* function takes a taxable income and a list of tax brackets as input. It iterates through each tax bracket, calculating the tax based on the income within each bracket. It accumulates the total tax amount as it progresses through the brackets. If the income is fully taxed within a bracket, it breaks the loop. Otherwise, it continues to the next bracket, updating the remaining income accordingly. Finally, it returns the total tax calculated.

```python
def calculate_taxable_income(taxable_income, brackets):
    total_tax = 0
    remaining_income = taxable_income
    for bracket in brackets:
        if remaining_income <= 0:
            break
        bracket_amount, tax_rate = bracket
        if remaining_income <= bracket_amount:
            total_tax += remaining_income * tax_rate
            break
        else:
            total_tax += bracket_amount * tax_rate
            remaining_income -= bracket_amount
    return total_tax
```

- ***Calculate_rrsp_contribution* <u>Function:</u>**
  - The *calculate_rrsp_contribution* function is integral to projecting the growth of a Registered Retirement Savings Plan (RRSP) over multiple years.
  - Taking inputs such as the user's yearly income, the start and retirement years, and the expected salary growth rate, the function iterates through each year, calculating both employee and employer contributions based on preset rates.

```python
def calculate_rrsp_contribution(yearly_income, start_year, retirement_year, salary_growth_rate):
    table = []
    salary = yearly_income
    contribution_rate = 0.06
    employer_contribution_rate = 0.04
    total_contributions = 0
    total_returns = 0
    rrsp_account = 0

    for year in range(start_year, retirement_year + 1):
        employee_contribution = salary * contribution_rate
        employer_contribution = salary * employer_contribution_rate
        total_contribution = employee_contribution + employer_contribution
        total_contributions += total_contribution
        rate_of_return = random.uniform( a: 0.03,  b: 0.05) * total_contribution
        total_returns += rate_of_return
        rrsp_account += total_contribution + rate_of_return
        table.append([year, salary, employee_contribution, employer_contribution, total_contribution, rate_of_return, rrsp_account])
        salary *= (1 + min(salary_growth_rate, 5) / 100)

    return table, total_contributions, total_returns, rrsp_account
```

  - It then computes the total contributions made annually, as well as the investment returns generated, factoring in random rates of return between 3% and 5%.
  - The function updates the RRSP account balance accordingly and adjusts the user's salary for the following year based on the provided growth rate.
  - Finally, it returns a detailed breakdown (table) of each year's contributions, returns, and RRSP balance, alongside totals for contributions, returns, and the final account balance at the retirement year.
  - This comprehensive projection aids users in understanding how their RRSP will grow over time, facilitating informed retirement planning decisions.

- ***Calculate_tfsa_contribution* <u>Function:</u>**
  - Identically to RRSP, The *calculate_tfsa_contribution* function projects the growth of a Tax-Free Savings Account (TFSA) over multiple years based on yearly income, contribution rate, and investment returns.

```
def calculate_tfsa_contribution(yearly_income, start_year, retirement_year, salary_growth_rate):
    table = []
    salary = yearly_income
    contribution_rate = 0.08
    total_contributions = 0
    total_returns = 0
    tfsa_account = 0

    for year in range(start_year, retirement_year + 1):
        employee_contribution = salary * contribution_rate
        total_contributions += employee_contribution
        rate_of_return = random.uniform( a: 0.03,  b: 0.05) * employee_contribution
        total_returns += rate_of_return
        tfsa_account = employee_contribution + rate_of_return
        table.append([year, salary, employee_contribution, rate_of_return, tfsa_account])
        salary *= (1 + min(salary_growth_rate, 5) / 100)

    return table, total_contributions, total_returns, tfsa_account
```

  - It iterates through each year, calculating contributions, returns, and updating the account balance. It returns a summary of contributions, returns, and the final account balance, aiding users in understanding their TFSA's growth trajectory for retirement planning.

- **<u>User Input and RRSP Calculation:</u>**
  - The function begins by prompting the user to input their yearly income, start year of work, expected retirement year (with a maximum of 40 years after the start year), and salary growth rate.
  - It then calls the *calculate_rrsp_contribution* function to calculate RRSP details based on the provided inputs, including yearly contributions, returns, and final account balance.
  - The RRSP details are printed using the print_table function, displaying yearly contributions, returns, and the RRSP account balance.

```
def main():
    yearly_income = float(input("Enter your yearly income: "))
    start_year = int(input("Enter the year you started working: "))
    max_retirement_year = start_year + 40
    retirement_year = int(input(f"Enter your expected year of retirement (max {max_retirement_year}): "))
    retirement_year = min(retirement_year, max_retirement_year)

    salary_growth_rate = float(input("Enter the rate at which your salary will grow annually (max 5%) (%): "))

    rrsp_table, rrsp_total_contributions, rrsp_total_returns, rrsp_final_amount = calculate_rrsp_contribution(yearly_i
    print_table(rrsp_table,  title: "RRSP Details")

    print(f"\nYour yearly salary started in {start_year} and grew at an annual rate of {salary_growth_rate}%, until th
    print(f"At an average annual return rate of 3% to 5%, you can expect to earn approximately ${rrsp_total_returns:.2
    print(f"The total amount in your RRSP Account at the end of {retirement_year} would be ${rrsp_final_amount:.2f}.")
```

- **Tax Calculations and Printing Tax Details for RRSP:**
  - The function prompts the user to input their province.
  - It suggests a similar province name if the input does not match exactly.
  - The function retrieves the provincial tax brackets based on the user's input province.
  - If the province is not found, it prints a message stating that the province was not found, and tax calculation cannot proceed.
  - Otherwise, it calculates the federal and provincial taxes based on the RRSP returns using the *calculate_taxable_income* function.
  - It calculates the total tax deduction by adding federal and provincial taxes.
  - It computes the RRSP balance after tax deduction.
  - The function prints the federal tax deduction, provincial tax deduction, total tax deduction, and the RRSP balance after tax deduction.

```
province_input = input("Enter your province: ").lower()
suggested_province = get_close_matches(province_input, provincial_tax_rates.keys(), n=1, cutoff=0.7)
if suggested_province and suggested_province[0] != province_input:
    confirm = input(f"Did you mean {suggested_province[0]}? Enter 'y' or 'n': ")
    if confirm.lower() == 'y':
        province_input = suggested_province[0]

provincial_tax_brackets = provincial_tax_rates.get(province_input)

if provincial_tax_brackets is None:
    print("Province not found. Cannot calculate taxes.")
    return

federal_tax = calculate_taxable_income(rrsp_total_returns, federal_tax_brackets)
provincial_tax = calculate_taxable_income(rrsp_tota
total_tax_deduction = federal_tax + provincial_tax

rrsp_after_tax = rrsp_final_amount - total_tax_deduction

print(f"\nFederal Tax Deduction: \033[90m${federal_tax:.2f}\033[0m")
print(f"Provincial Tax Deduction ({province_input.title()}): \033[90m${provincial_tax:.2f}\033[0m")
print(f"Total Tax Deduction: \033[90m${total_tax_deduction:.2f}\033[0m")
print(f"\nAmount left after tax deduction for RRSP: \033[90m${rrsp_after_tax:.2f}\033[0m")
```

```
rrsp_total_returns: int = (calculate_rrsp_contribution(yearly_income,
```

- **Information Module**

  - Upon selecting option ***6***, the program initiates the functionality related to Information module.
  - It provide users with information about various Canadian investment account types, such as the Tax-Free Savings Account (TFSA), First Home Savings Account (FHSA), and Registered Retirement Savings Plan (RRSP).

- **Importing a Function from a Module**

```
from Advisory import advisory_func
```

  - The line from Advisory import *advisory_func* is an import statement in Python.
  - It imports a specific function named *advisory_func* from the module named Advisory.
  - This allows the *advisory_func* function to be used in the current Python script or program.
  - The Advisory.py file must be present in the same directory or in a directory listed in the Python module search path.
  - The Advisory.py file should contain the definition of the *advisory_func* function.

- **Investment Account Information Functions**

  - Three functions (*show_tfsa_info*, *show_fhsa_info*, *show_rrsp_info*) provide brief information on TFSA, FHSA, and RRSP respectively.
  - Each function prints essential details about the account type, including its purpose, tax treatment, contribution rules, and government regulations.

```
def show_tfsa_info():
    print("Tax-Free Savings Account (TFSA) Information
    print("- A TFSA is a registered account that allow
    print("- Contributions to a TFSA are made with aft
```

```
def show_fhsa_info():
    print("First Home Savings Account (FHSA) Infor
    print("- The FHSA is designed to help Canadian
    print("- Contributions to an FHSA are not tax-
```

```
def show_rrsp_info():
    print("Registered Retirement Savings Plan (RRSP) Informa
    print("- An RRSP is a tax-advantaged account designed to
    print("- Contributions to an RRSP are tax-deductible, me
```

- **Investment Account Information Program (Menu Display)**
  - The main function displays a menu for users to choose options.
  - It prompts the user to select an option (1-6) to view information about different types of investment accounts or exit the program.

```
def main():
    print("Welcome to the Canadian Investment Account Information Program!\n")
    while True:
        print("Choose an option to view information:")
        print("1. TFSA (Tax-Free Savings Account)")
        print("2. FHSA (First Home Savings Account)")
        print("3. RRSP (Registered Retirement Savings Plan)")
        print("4. Financial Advisory")
        print("5. References")
        print("6. Exit")
        choice = input("Enter your choice (1/2/3/4/5/6): ")
        print()
```

- **Investment Account Information Display (Response)**
  - The code checks the user's choice and displays detailed information about different investment account types based on the selected option.
  - For each option (1, 2, 3), it calls respective functions *(show_tfsa_info(), show_fhsa_info(), show_rrsp_info()*) to print information about the chosen account type.
  - After displaying basic information, it prompts the user to enter 'm' for more details.
  - If the user enters 'm', it prints additional information about the selected investment account type.

```
elif choice == '3':
    show_rrsp_info()
    more_info = input("Enter 'm' for more information, or press Enter
    if more_info.lower() == 'm':
        print("- RRSP contribution room is based on your earned income
        print("- Common RRSP investment options include mutual funds,
        print("- The rate of return for RRSP investments can vary depe
```

- **Financial Advisory**
  - If the user selects option 4, the code calls the *advisory_func()* from the Advisory module.
  - This function provides financial advice or guidance to the user regarding investment decisions or related matters.

```
elif choice == '4':
    advisory_func()
```

- **References**
  - If the user selects option 5, the code displays a list of references related to different investment accounts.
  - Each reference includes the name of the account and a corresponding link to a relevant website or document providing additional information.
  - The references cover topics such as TFSA (Tax-Free Savings Account), FHSA (First Home Savings Account), RRSP (Registered Retirement Savings Plan), and differences between TFSA, RRSP, and FHSA.
  - These references serve as supplementary resources for users who want to explore further details about the mentioned investment accounts and their differences.

```
elif choice == '5':
    print("References:")
    print("1. TFSA - Tax-Free Savings Account: https://www.canada.ca/en/revenue-agency/services/forms-pub
    print("2. FHSA - First Home Savings Account: https://www.canada.ca/en/revenue-agency/services/tax/ind
    print("3. RRSP - Registered Retirement Savings Plan: https://www.canada.ca/en/revenue-agency/services
    print("4. Difference Between TFSA, RRSP, and FHSA: https://protectyourwealth.ca/difference-between-tf
    print()
```

- **Program Termination and User Interaction**
  - If the user selects option 6, indicating they want to exit the program, the code prints a farewell message.
  - The program then breaks out of the loop, effectively ending the program execution.
  - If the user enters an invalid choice (neither 1, 2, 3, 4, 5, nor 6), the code prints an error message prompting the user to enter a valid option.
  - After displaying the error message, the code prompts the user if they want to select another option.
  - If the user enters 'y' (yes), the loop continues, allowing them to choose another option.
  - If the user enters any other input or presses Enter, indicating they do not want to select another option, the code prints a farewell message and breaks out of the loop, terminating the program.

```
elif choice == '6':
    print("Thank you for using the program. Goodbye!")
    break


else:
    print("Invalid choice. Please enter a valid option (1-5).\n")

another_choice = input("\nWould you like to select another option? (y/n): ")
if another_choice.lower() != 'y':
    print("Thank you for using the program. Goodbye!")
    break
print()
```

- **Financial Advisory for RRSP and TFSA Investment Decision**
  - The Python script titled Advisory.py provides guidance on whether to invest in a Registered Retirement Savings Plan (RRSP) or a Tax-Free Savings Account (TFSA), tailored to individual financial circumstances.
  - It offers insights based on factors such as employer RRSP matching, existing debt, emergency fund status, current income level, and future income expectations.
- **Key Features of the Script:**
  - User Input Handling:
    - The script begins by asking the user if their employer offers RRSP matching.
    - It then guides users through a series of questions regarding their financial situation, including debt status, emergency fund availability, current income, and future income expectations.
    - Based on user responses, the script recommends whether to prioritize RRSP or TFSA contributions.
  - Decision-Making Logic:
    - The advisory logic considers factors such as employer RRSP matching, debt status, emergency fund availability, and income levels to provide personalized recommendations.
    - It assesses whether the user's income is likely to increase or decrease in the future to determine the optimal investment strategy.
  - Clear Recommendations:
    - The script offers clear and actionable recommendations based on the user's financial circumstances.
    - Recommendations include prioritizing TFSA contributions if income is below a certain threshold, anticipating future income changes, and maximizing tax advantages associated with RRSP contributions.
  - Scenario Analysis:
    - The script accounts for various scenarios, such as high- or low-income levels and expectations for future income changes, to provide tailored recommendations.
  - Interactive Execution:
    - Users engage with the script through a series of interactive prompts, providing input that influences the advisory process.
    - The script guides users step-by-step through the decision-making process, ensuring clarity and understanding at each stage.

```python
def advisory_func():
    print("WEALTHSIMPLE EXPLAINS: SHOULD I INVEST IN AN RRSP OR A TFSA?")
    employer_matching = input("Does your employer offer RRSP matching? (y/n): ").lower()

    if employer_matching == "y":
        print("Take the maximum free money every year. It's part of your salary!")
        ask_debt()
    else:
        ask_debt()
```

```
def ask_debt():
    debt = input("Before you go any further, do you have debt? (y/n): ").lower()

    if debt == "n":
        ask_emergency_fund()
    else:
        print(
            "For example, high-interest rate debt could be credit cards, while low-int
        debt_type = input("Enter 1 if you have high-interest rate debt, or 2 if you ha
        if debt_type == "1":
            print("STOP! Use any spare cash to eliminate your debt.")
        else:
            ask_emergency_fund()
```

```
def ask_emergency_fund():
    emergency_fund = input("Do you have an emergency fund? (y/n): ").lower()

    if emergency_fund == "n":
        print("STOP! Use any spare cash to start an emergency fund.")
    else:
        print("Great, you have some money to invest.")
        ask_income()
```

```
def ask_income():
    income = input("Now, do you earn more than $50,000 per year? (y/n): ").lower()

    if income == "n":
        print("Start contributing to TFSA.")
    else:
        ask_high_income()
```

```
def anticipate_high_income():
    anticipate = input(
        "Do you anticipate that your annual income will be more than $100,0

    if anticipate == "y":
        print("Contribute to your TFSA first until it's maxed out.")
    else:
        contribute_rrsp()
```

```
def anticipate_low_income():
    anticipate = input(
        "Do you anticipate that your annual income will be more than $100,000 pe

    if anticipate == "y":
        contribute_rrsp()
    else:
        print("Contribute to your TFSA until it's maxed out.")
def contribute_rrsp():
    withdrawal_income = input(
        "When the time comes to withdraw funds, do you think your income wi

    if withdrawal_income == "y":
        print(
            "Contribute to an RRSP. You'll save money on your taxes now, ar
    else:
        print("Contribute to your TFSA until it's maxed out. Then start mak
```

- **Important Features and functions of code:**

  - *advisory_func():*

    Initiates the advisory process by asking users if their employer offers RRSP matching and redirects to subsequent functions based on user input.

  - *ask_debt():*

    Determines if the user has outstanding debt and directs them to prioritize debt repayment if applicable and guides users to assess their debt type and make informed decisions about debt management.

  - *ask_emergency_fund():*

    Asks users if they have an emergency fund and advises them to establish one if not. Additionally, It does provide guidance on the importance of having an emergency fund before considering investments.

  - *ask_income():*

    Assesses user income levels to determine the appropriate investment strategy and advises users on whether to contribute to a TFSA based on their income threshold.

  - *ask_high_income() and anticipate_high_income():*

    Determines if the user's income exceeds a certain threshold and anticipates future income changes and provides recommendations on TFSA and RRSP contributions based on income expectations.

  - *contribute_rrsp():*

    Guides users on RRSP contributions based on future income expectations and tax implications and recommends RRSP contributions for tax advantages if income is expected to decrease.

  - The script employs nested if-else statements to manage various decision paths based on user input. These nested structures enable the evaluation of multiple conditions and the execution of corresponding actions.

  - This hierarchical approach allows the script to provide personalized financial guidance tailored to the user's circumstances.

Conclusion and Future Scope

In conclusion, our Python project aimed to enlighten users about the financial landscape in Canada by providing insight into various banking accounts and taxation systems. Through diligent research and programming, we developed a comprehensive tool with several key features.

Our project allows users to calculate their maximum Tax-Free Savings Account (TFSA) contribution limits based on their residency status in Canada. Additionally, users can estimate the time required to achieve specific financial goals and explore the benefits of Registered Retirement Savings Plans (RRSP) through our RRSP calculator. Furthermore, our program provides insights into taxation, offering users a clearer understanding of their financial obligations.

Throughout the development process, we encountered the challenge of acquiring detailed knowledge about Canadian banking accounts, economics, and taxation. However, with perseverance and rigorous research, we overcame these obstacles to create a robust and informative tool.

Looking ahead, we recognize the potential for future enhancements. While our current version covers a range of banking accounts and financial concepts, there is room to expand and include additional features. Future updates may incorporate more banking accounts available in Canada, ensuring that users have access to a comprehensive resource for financial planning and management.

In summary, our Python project serves as a valuable resource for individuals seeking to navigate the complexities of personal finance in Canada. By providing accurate calculations, informative insights, and the potential for future expansion, we aim to empower users on their financial journey.