PSET1

Problema A: Ahorrando para una Casa

En este problema simulamos la situación de un recién graduado de MIT que quiere comprar una casa en el área de la Bahía de San Francisco. Como las casas son caras, necesitamos calcular cuánto tiempo nos tomará ahorrar para el enganche (down payment).

Conceptos clave:

- **Interés compuesto**: Nuestros ahorros generan un 5% anual de retorno
- **Ahorro mensual**: Guardamos un porcentaje fijo de nuestro salario cada mes
- **Cálculo del enganche**: Necesitamos el 25% del costo total de la casa

Ejemplo práctico:

Si ganas $112,000 al año, ahorras el 17% de tu salario, y quieres una casa de $750,000, el programa calculará que necesitas 97 meses para juntar los $187,500 del enganche.

Mi aproximación

```
################################################################################
## Get user input for the three variables below ##
################################################################################

yearly_salary = float(input("Enter your yearly salary: "))
portion_saved = float(input("Enter the percent of your salary to save, as 
cost_of_dream_home = float(input("Enter the cost of your dream home: "))

################################################################################
## Initialize other variables you need (if any) for your program below ##
################################################################################

# Constants
portion_down_payment = 0.25
r = 0.05  # annual return

# Variables
monthly_salary = yearly_salary / 12
amount_saved = 0.0
```

```
months = 0
down_payment = cost_of_dream_home * portion_down_payment

################################################################################
## Determine how long it takes to save for a down payment ##
################################################################################

# Loop until enough savings
while amount_saved < down_payment:
    amount_saved += amount_saved * (r / 12) + monthly_salary * portion_sav
    months += 1

# Output
print("Number of months:", months)
```

Problema B: Ahorrando con Aumentos de Sueldo

Este problema extiende el Problema A agregando un elemento más realista: ¡los aumentos de sueldo! Como graduado de MIT, tu valor en el mercado laboral aumenta con el tiempo, por lo que recibes aumentos cada 6 meses.

Conceptos clave:

- **Aumentos semianuales**: Cada 6 meses tu salario aumenta por un porcentaje fijo
- **Timing de los aumentos**: Los aumentos ocurren exactamente al final de los meses 6, 12, 18, etc.
- **Efecto acumulativo**: Los aumentos se aplican sobre el salario ya aumentado

Ejemplo práctico:

Con un salario inicial de $110,000, ahorrando 15%, una casa de $750,000, y aumentos del 3% cada 6 meses, solo necesitas 92 meses en lugar de los 97 del Problema A.

Mi aproximación

```
################################################################################

```

```python
## Get user input for the three variables below ##
################################################################################

yearly_salary = float(input("Enter your starting yearly salary: "))
portion_saved = float(input("Enter the percent of your salary to save, as a
decimal: "))
cost_of_dream_home = float(input("Enter the cost of your dream home: "))
semi_annual_raise = float(input("Enter the semi-annual raise, as a decimal
"))

################################################################################
## Initialize other variables you need (if any) for your program below ##
################################################################################

# Constants
portion_down_payment = 0.25
r = 0.05  # annual return

# Variables
monthly_salary = yearly_salary / 12
amount_saved = 0.0
months = 0
down_payment = cost_of_dream_home * portion_down_payment

################################################################################
## Determine how long it takes to save for a down payment ##
################################################################################

# Loop
while amount_saved < down_payment:
    amount_saved += amount_saved * (r / 12) + monthly_salary * portion_sav
    months += 1
    if months % 6 == 0:
        yearly_salary *= (1 + semi_annual_raise)
        monthly_salary = yearly_salary / 12

# Output
print("Number of months:", months)
```

Problema C: Eligiendo una Tasa de Interés

En este problema invertimos la ecuación: en lugar de calcular cuánto tiempo toma ahorrar, calculamos qué tasa de retorno necesitamos para alcanzar nuestro objetivo en exactamente 3 años (36 meses).

Conceptos clave:

- **Búsqueda binaria**: Algoritmo eficiente para encontrar la tasa óptima

- **Objetivo fijo**: Siempre queremos una casa de $800,000 en exactamente 36 meses
- **Tolerancia**: Aceptamos estar dentro de $100 del objetivo
- **Casos límite**: Algunas situaciones son imposibles o no requieren interés

Ejemplo práctico:

Si tienes $65,000 de depósito inicial, necesitas una tasa de retorno de aproximadamente 38.06% anual para alcanzar los $200,000 necesarios en 36 meses. El programa usa búsqueda binaria para encontrar esta tasa en 12 pasos.

Casos especiales:

1. **Depósito suficiente**: Si ya tienes $199,900 o más, no necesitas interés (r = 0.0)
2. **Objetivo imposible**: Si ni siquiera con 100% de retorno anual puedes alcanzar el objetivo, el resultado es None
3. **Búsqueda normal**: Para la mayoría de casos, usamos búsqueda binaria entre 0% y 100%

Mi aproximación

```
################################################
## Get user input for the three variables below ##
################################################

initial_deposit = float(input("Enter the initial deposit: "))

###############################################################################
## Initialize other variables you need (if any) for your program below ##
###############################################################################

# Constants
cost = 800000
down_payment = 0.25 * cost  # $200,000
months = 36
epsilon = 100

# Variables for bisection search
low = 0.0
high = 1.0  # 100% annual rate
steps = 0
r = None

```

```python
##############################################################################
## Determine how long it takes to save for a down payment ##
##############################################################################

# Check if initial deposit is already sufficient (within $100 of target)
if initial_deposit >= down_payment - epsilon:
    r = 0.0
else:
    # Check if it's impossible even with 100% return rate
    max_amount = initial_deposit * ((1 + high / 12) ** months)
    if max_amount < down_payment - epsilon:
        r = None
    else:
        # Bisection search
        while low <= high:
            guess = (low + high) / 2.0
            amount_saved = initial_deposit * ((1 + guess / 12) ** months)
            steps += 1

            # Check if we're within the acceptable range
            if abs(amount_saved - down_payment) <= epsilon:
                r = guess
                break
            elif amount_saved < down_payment:
                low = guess
            else:
                high = guess

            # Stop if the range is small enough
            if high - low < 1e-10:
                break

# Output
if r is None:
    print("Best savings rate: None")
else:
    print("Best savings rate:", r)
print("Steps in bisection search:", steps)
```