Ejercicios Extra
Programación
2020 - 2021
Grado en Ingeniería Informática

Observaciones: hay muchas maneras de resolver estos problemas; algunos de ellos se resuelven mejor usando programación orientada a objetos, mientras que en otros es más adecuado el uso de programación estructurada. En ambos casos, hay muchas maneras de dividir el problema en diferentes métodos/funciones. Las soluciones que se proporcionarán se deben tomar como una de las muchas formas de resolver estos ejercicios. Cada estudiante debería intentar la suya propia.

Ejercicio 1. Crear un programa que permita al usuario trabajar con conjuntos de números de tipo float. El programa ofrecerá al usuario las siguientes opciones:

1) Llenar una lista aleatoria: el usuario seleccionará el número de elementos y el límite superior e inferior.
2) Llenar una lista aleatoria donde el número de elementos y los límites inferior y superior estarán en el rango [1,1000)
3) Introducir manualmente una lista: el usuario seleccionará primero el número de elementos y luego irá introduciendo cada uno de los valores.
4) Reducir la lista: el programa creará una nueva con tamaño igual a la mitad de la lista original y que será el resultado de sumar los elementos de la lista original de dos en dos (el primer elemento de la lista resultante será el resultado de sumar el primer y segundo elemento de la original; el segundo será la suma de la tercera y la cuarta posición original y así sucesivamente). Se debe considerar el caso en que la lista original tiene un número impar de elementos.
5) Invertir la lista (**sin usar el método reverse**): el programa devolverá una lista con los elementos en orden inverso (el primero será el último y así sucesivamente).
6) Imprimir la lista: el programa imprimirá la lista.

Ejemplo de ejecución:

```
¿Cómo quieres rellenar la lista?
1) Aleatoriamente con valores límite
2) De forma totalmente aleatoria
3) Manualmente
4
¡Por favor introduce una opción correcta!
¿Cómo quieres rellenar la lista?
1) Aleatoriamente con valores límite
2) De forma totalmente aleatoria
3) Manualmente
1
Introduce el número de elementos
10
Introduce límite superior
20
Introduce límite inferior
8
La lista es:
11 19 11 19 8 8 13 12 11 8
```

```
Elige una opción:
A) Reducir la lista
B) Invertir la lista
C) Salir
B
8 11 12 13 8 8 19 11 19 11
Elige una opción:
A) Reducir la lista
B) Invertir la lista
C) Salir
A
19 25 16 30 30
Elige una opción:
A) Reducir la lista
B) Invertir la lista
C) Salir
F
¡Por favor introduce una opción correcta!
Elige una opción:
A) Reducir la lista
B) Invertir la lista
C) Salir
C
¡Gracias!
```

Ejercicio 2. Las páginas para trabajar con fechas son bastante habituales en internet. Su objetivo es permitir al usuario realizar diversas tareas como calcular el día siguiente a uno dado, el número de días entre dos fechas, etc. En este ejercicio crearemos un programa capaz de realizar estos cálculos. En concreto el programa será capaz de realizar las siguientes operaciones:

- Seleccionar el formato de fecha a usar:
  i. dd/mm/aaaa
  ii. mm/dd/aaaa
  iii. aaaa/mm/dd
  iv. <Día de mes de año> (como en 22 de Enero de 2019)
- Comprobar que la fecha es correcta (teniendo en cuenta bisiestos)
- Convertir la fecha desde uno de los formatos anteriores a cualquiera de ellos (en los siguientes apartados el usuario introducirá la fecha en el formato elegido previamente)
- Calcular el día siguiente a uno dado, teniendo en cuenta bisiestos.
- Comprobar si una fecha es anterior a otra.
- Calcular el número de años entre dos fechas.
- Calcular el número de días entre dos fechas.
- Imprimir todos los días entre dos fechas.
- Comprobar si una fecha está entre otras dos.
- Ordenar una lista de fechas en orden ascendente o descendente (a elegir por el usuario)
- Crear una clase Persona, con campos nombre y cumpleaños y los siguientes métodos:

i. Un método para calcular la edad de la persona, teniendo en cuenta su fecha de nacimiento y la fecha actual, que se recibirá como parámetro.

ii. Un método que recibe otro objeto Persona y devuelve 1 si esta persona es mayor que la otra, 0 si tienen la misma edad y -1 si la otra persona es mayor.

En el programa principal, hay que pedir al usuario que rellene una lista de personas, ordenarla de más joven a más viejo e imprimir la diferencia de años entre cada uno de ellos como en el siguiente ejemplo: "Pepe: 2 años, tiene la misma edad que Juana: 2 años, es 1 año(s) más joven que Mario: 3 años". El programa debe funcionar para cualquier número de personas.

Ejercicio 3. En este ejercicio simularemos una estación meteorológica que es capaz de medir temperatura y presión atmosférica. Para hacerlo serán útiles las siguientes definiciones:

- Una marca de tiempo está formada por horas, minutos y segundos. El programa debe comprobar que sus valores tienen sentido.
- Una medida está compuesta por una marca de tiempo y la cantidad medida (por ejemplo 32 grados a las 12:23:48). La temperatura se medirá en grados centígrados y la presión en milibares. En nuestro caso tanto la marca de tiempo como la cantidad se generarán aleatoriamente cuando se cree la medida (elegir valores adecuados para cada tipo). Se deben crear métodos `__str__` que devuelvan cadenas del estilo de "`1013 mb a las 15:12:12`" o "`32 grados a las 08:21:13`".
- Una estación meteorológica tiene un identificador, un sitio en el que está emplazada (representado por su nombre), dos listas que contienen todas las medidas de presión y temperatura de las últimas 24 horas, el número de medidas que toma cada hora y la temperatura y presión media.
- Las estaciones meteorológicas pueden tomar medidas. El que la medida sea de temperatura o de presión se recibirá como parámetro. Cada vez que se pida tomar una medida, se generará aleatoriamente una nueva y se incluirá en la lista de medidas de temperatura o presión, según corresponda. Cada vez que se genere una nueva medida la lista deberá ordenarse de forma que las más recientes (según la marca de tiempo) estén primero. Una vez que la lista de medidas de algún tipo esté llena (se hayan tomado todas las posibles medidas en 24 horas), las nuevas medidas generadas se perderán.
- Se pueden leer las medidas de las siguientes maneras:

  i. Recibiendo una marca temporal: se devolverá la medida más cercana a esa hora y se borrará de la lista de medidas.

  ii. Recibiendo una posición: al igual que en el caso anterior se extraerá la medida de la lista, reorganizando las restantes.

  iii. Leyendo la lista completa: se devolverá toda la lista y a continuación se vaciará.

En un programa principal se le pedirá al usuario el identificador y nombre de la estación meteorológica, además del número de medidas tomadas cada hora. El usuario podrá generar nuevas medidas (especificando el tipo y cuántas quiere crear), leer una o varias medidas (usando los métodos anteriores), o leer los valores medios de temperatura o presión.

Ejercicio 4. Se debe programar el juego del MultiPóker, que es un programa que simula la extracción aleatoria de un conjunto de cartas de tamaño N y descubre las combinaciones de cartas que son útiles para ganar el juego. Las reglas básicas son:

- Las cartas se numeran del 1 al 10 (sota, caballo y rey se numeran como 8, 9 y 10). No se considera el palo de las cartas.
- En todas las rondas del juego se debe jugar con el mismo número de cartas.
- Las combinaciones posibles son: pareja (2 cartas repetidas), trío (tres cartas repetidas) y escalera (del grupo de cartas al menos 4 deben ser de números consecutivos). Ej. en el juego de seis cartas (5 2 3 3 8 4) hay una pareja y una escalera (2 al 5).
- Cuando no hay ninguna combinación ganadora se dice que es una "mala ronda".

Hacer un programa usando las funciones que se consideren necesarias para reproducir el siguiente comportamiento para un solo jugador.

1. Pedir al usuario el número de cartas utilizado para las rondas.
2. Pedir al usuario el número de malas rondas que pueden pasar antes de terminar la partida.
3. Para cada ronda se debe:
    - Generar el conjunto N de cartas de forma aleatoria.
    - Imprimir las cartas de la ronda.
    - Imprimir el número de parejas, de tríos y si hay o no escalera en la ronda. Si no hay nada se debe imprimir "mala ronda".
    - Terminar la partida si se ha producido el número de malas rondas establecido.

Ejercicio 5. Programar un sistema hipotético de control de efectivo en bancos y cajeros automáticos que tenga las siguientes características.

- Una cuenta corriente abierta en un banco pertenece a un usuario, tiene un número de cuenta, un número de tarjeta y mantiene el balance de dinero.
- Un usuario tiene un nombre, DNI y una dirección. Además puede tener como máximo 5 cuentas bancarias.
- Un cajero automático pertenece a un banco, tiene asociada una dirección y mantiene el saldo del efectivo disponible.

Se pide lo siguiente:

1. Diseñar las clases necesarias para representar los elementos descritos anteriormente.
2. Programar los métodos __init__ y las propiedades necesarias en cada clase.
3. Implementar en la clase más adecuada el método `sacarDineroCajero` que dado un usuario y una cantidad a retirar, haga lo siguiente.
    - Determine si el usuario tiene una cuenta en el banco al que pertenece el cajero. De no tener se debe avisar al usuario para no continuar el proceso.
    - Determine si la cuenta tiene saldo disponible para la cantidad solicitada. Si no tiene saldo se debe avisar al usuario para no continuar con el proceso.
    - Determine si el cajero tiene efectivo disponible para satisfacer la operación.
    - Reste la cantidad solicitada de la cuenta del usuario y del saldo disponible del cajero.

4. Implementar en la clase más adecuada el método `reponerCajero` que dada una cantidad de efectivo actualice el saldo disponible del cajero.
5. Hacer un programa principal que:
    - Cree 3 cajeros, cada uno de bancos distintos
    - Cree 2 usuarios. Para el usuario 1 se deben crear 2 cuentas de bancos distintos, y para el usuario 2 una cuenta del banco restante.
    - Simule que un usuario intenta sacar dinero de un cajero que no es de su banco, y luego que saca dinero correctamente.
    - Imprima el saldo final de los cajeros.

Ejercicio 6. El objetivo de este ejercicio es crear mediante programación orientada a objetos un juego de persecución y evasión de acuerdo a las reglas siguientes:

- El juego se desarrolla en un tablero de dimensiones MxN, elegidas por el usuario.
- Dentro del tablero hay un cazador y varias presas. El número de presas lo elegirá el usuario y debe ser mayor que 0, en caso contrario se generarán 5 presas.
- Tanto el cazador como las presas comenzarán en posiciones aleatorias y distintas del tablero: en cada posición del tablero solo puede haber un cazador o una presa.
- Las presas se mueven aleatoriamente según el siguiente procedimiento:
    - En el estado inicial elegirán aleatoriamente una dirección y se moverán una casilla. Posteriormente, en cada turno tendrán una probabilidad equivalente de moverse una casilla a la derecha, una casilla a la izquierda, una casilla hacia delante o quedarse quietas.
    - Si al moverse chocan con los bordes del tablero, rebotarán y se moverán en la dirección contraria. Lo mismo ocurrirá si no pueden moverse a la izquierda o la derecha porque han alcanzado los límites del tablero.
- El cazador intentará dar caza a las presas de la siguiente manera:
    - Al inicio del juego calculará la distancia a cada presa según el número de casillas que debe moverse para alcanzarlas, teniendo en cuenta que no se puede mover en diagonal. Una vez calculadas las distancias para todas las presas, las ordenará por cercanía y se moverá una casilla en la dirección de la presa más cercana.
    - Cuando el cazador alcance a una presa, la capturará y la hará desaparecer del juego. En ese momento volverá a calcular las distancias al resto de presas, pasando a perseguir a la más cercana.
- El juego termina cuando se han cazado todas las presas o una vez transcurridos 200 turnos.
- En cada turno se deben realizar las siguientes tareas.
    - Cada presa se moverá a su siguiente posición.
    - El cazador se moverá una casilla en dirección de la presa que esté persiguiendo en ese momento.
    - El cazador comprobará si ha cazado a la presa. En caso positivo la eliminará del tablero y elegirá a la que va a perseguir el turno siguiente. Si no hay presas vivas el juego terminará.
    - Se imprimirá el estado del juego como en el ejemplo, en el que las casillas se representan con ‘S’, las presas por su número id y el cazador por una ‘C’.

- Crear una estructura de clases para representar el juego: al menos debe tener clases para las presas, el cazador y el tablero. Definir para cada una de ellas los campos más adecuados, al igual que las propiedades y métodos init. Crear como mínimo métodos para: crear el tablero inicial, calcular la distancia a una presa, mover una presa, mover el cazador, ordenar las presas por distancia y determinar si se ha cazado una presa. Crear también un método `str` para pintar el tablero. Colocar cada método dentro de la clase que se considere más adecuada.

Ejemplo de estado inicial:

```
Introducir el tamaño del tablero y el número de presas
5
5
3
2SSSS
SSSS0
SSSSS
SSCSS
SS1SS
```