Actividad 03 - Diez mil (simplificado)

Pensamiento Computacional

Segundo Cuatrimestre 2024

Pasos para resolver los ejercicios

1. **Entender qué hay que hacer**
   a) **Leer en detalle** el texto de la descripción de la función e interpretar qué se espera (por ej., qué parámetros tiene, qué hace con la entrada y cómo genera la salida).
   b) Escribir en el cuaderno distintos ejemplos de **entradas** y qué **salidas** se esperan para cada una.
2. **Armar el esqueleto de la función**
   a) Escribir exactamente como se pidió el **encabezado de la función**, incluyendo los parámetros.
   b) Definir la **variable de la salida** y ponerle un valor inicial válido (por ej., 0 para un contador)
   c) **Agregar el** `return` donde se devuelve la variable anterior.
   d) Probar **ejecutar la función** con diferentes valores, aunque todavía no hace lo que se pide.
3. **Implementar la función**
   a) **Escribir de a poco** el contenido de la función.
   b) **Usar otras funciones** para resolver el problema, ya sea que estén de antes (y las reutilizan), o que crean otras nuevas (porque nos conviene partir el problema en otros más pequeños y manejables).
4. **Probar la función**
   a) **Probar la función solamente por la consola**, usando distintos valores para los parámetros, y ver que funciona bien. No hacer las pruebas desde el editor de texto de Spyder ni usar prints.
   b) **Ver qué pasa con casos especiales**, como cuando le dan una lista vacía u otro caso donde se usa una entrada válida, pero que pone a prueba los límites de lo que se hizo.
5. **Preguntar al equipo docente** por cualquier duda o consulta.

El objetivo de esta actividad es realizar un programa en `Python` que *simule* varios individuos jugando con dados.
Vamos a escribir un programa en `Python` que *juegue* a los dados. Con suerte, en algún momento, llegaremos programar el `Diez mil S` (S de simplificado), un entretenimiento basado en el famoso *Diez mil* pero con reglas simplificadas, que tiene como objetivo es llegar a los 10000 puntos.

A lo largo de esta guía utilizaremos dados equilibrados (no hay trampas), de forma tal que las caras caen con igual probabilidad.

Otras herramientas útiles de Python

Para que estén disponibles más funciones de `Python`, tenemos que utilizar el comando `import`. En esta actividad vamos a usar funciones implementadas en `random`, para lo que vamos a ejecutar `import random`.

> **Nota**: en caso de que existan funciones de `Python` que resuelvan alguno de los ítems pedidos total o parcialmente, **no es posible** utilizarlas (salvo que esté explícitamente mencionado).

Diez mil Simplificado

Vamos a introducir las reglas: en cada **jugada** se lanzan 5 dados y se calcula el puntaje obtenido de la siguiente manera: por cada uno obtenido, se suman 100 puntos, mientras que 3 unos suman 1000 puntos, 4 unos suman 1100, y 5 unos suman 10000 puntos. Además, por cada cinco obtenido se suman 50, mientras que 3 cincos suman 500, 4 cincos suman 550, y 5 cincos suman 600 puntos. Los dados dos, tres, cuatro y seis no suman puntos. Se juega por rondas. En cada ronda cada participante realiza una **jugada**, como se acaba de describir. Al finalizar la ronda se suman los puntos obtenidos por cada jugador a los que ya tenía. Cada participante comienza con 0 puntos.

El juego termina cuando al cabo de una ronda algún jugador alcanza (o supera) los 10000 puntos, o saca 5 unos. En este caso ganan todos aquellos que superen dicho puntaje, o hayan sacado 5 unos.

Una típica duda de domingo por la tarde es decidir si jugar o no al 10000. Surge, en muchos casos, la siguiente pregunta: En promedio, ¿cuántas rondas tiene una partida con 10 jugadores?

Para dar respuesta a esta duda existencial de las familias argentinas, comencemos implementando las siguientes funciones:

1. `tirar_cubilete()`: devuelve una lista con los valores de los 5 dados tirados. **Cabe notar que esta función no recibe parámetros**.

   Nota: El cubilete es el vasito que se usa para juntar y arrojar los dados :)

   > **Para probar en consola:**
   >
   > Correr la función `tirar_cubilete()` varias veces para asegurarnos que devuelva lo que esperamos.

2. `cuantos_hay(elemento, lista)`: toma un *elemento* y una *lista* y devuelve la cantidad de veces que *elemento* aparece en *lista*.

3. `puntos_por_unos(lista_dados)`: toma una lista de valores obtenidos y devuelve el puntaje obtenido **por la cantidad de unos** en `lista_dados` según las reglas actuales.

   > **Para probar en consola:**
   >
   > Probá tu función dandole como argumento estas listas de dados:
   > `[1, 1, 3, 5, 1]` (1000 puntos)
   > `[6, 5, 5, 1, 5]` (100 puntos)

4. `puntos_por_cincos(lista_dados)`: toma una lista de valores obtenidos y devuelve el puntaje obtenido **por la cantidad de cincos** en `lista_dados` según las reglas actuales.

5. `total_puntos(lista_dados)`: toma una lista de valores obtenidos y devuelve el puntaje obtenido según las reglas actuales.

6. `jugar_ronda(cant_jugadores)`: simula una ronda de jugadas y devuelve una lista con los puntos obtenidos por cada uno de los `cant_jugadores` jugadores.

7. `acumular_puntos(puntajes_acumulados, puntajes_ronda_actual)`: recibe como parámetro una lista con los `puntajes_acumulados` de cada jugador y los resultados de la ronda actual (`puntajes_ronda_actual`). La función devuelve la lista de puntajes actualizados.

> **Para probar en consola:**
>
> Creá dos variables llamadas `lista_prueba1` y `lista_prueba2` usando la función `jugar_ronda()` Luego pasaselas como parámetros a `acumular_puntos(lista_prueba1, lista_prueba2)` para asegurarte de que funciona bien. (Siempre en la consola)

8. `hay_10mil(puntajes_acumulados)`: toma como parámetro una lista con los `puntajes_acumulados` por los jugadores, y devuelve el valor lógico `True` si algún puntaje es mayor o igual a 10000 o `False` si no.

> **Para probar en consola:**
>
> Probá tu función dandole como argumento estas listas de puntajes:
> `[100, 10000, 50, 0]`
> `[11000, 9000, 550, 1200, 3450]`
> `[50, 50, 100]`

9. `partida_completa(cant_jugadores)`: Simula una partida completa entre una dada cantidad de jugadores (`cant_jugadores`), incluyendo el o los pasos necesarios para comenzar la partida. La partida termina cuando algun jugador llega a sumar 10000 puntos o más. La función debe devolver la cantidad de rondas jugadas.

> **Para pensar**
>
> Esta función solo te devuelve la cantidad de rondas que fueron necesarias. ¿Cómo te podes asegurar que el juego evolucionó correctamente?

10. Vamos ahora a utilizar las funciones implementadas para responder las siguientes preguntas:

    a) En promedio, ¿cuántas rondas tiene una partida con 10 jugadores?
    Crear la función `cant_rondas_promedio(cant_jugadores, cant_partidas)`, que recibe la cantidad de jugadores que van a participar (en este caso, 10) y la cantidad de partidas a simular (aquí, 10000), y guarda la cantidad de rondas de cada partida, devolviendo el promedio de rondas (se divide la suma de la cant. de rondas de cada partida por la cant. total de partidas).

    b) ¿Qué chances hay de terminar una partida con 10 jugadores (que algún jugador alcance los diez mil puntos) si sólo tenemos tiempo para jugar a lo sumo 18 rondas?
    Crear la función `chance_de_terminar(cant_jugadores, max_rondas, cant_partidas)`, que recibe la cantidad de jugadores que van a jugar (aquí, son 10), el máximo de rondas que vamos a evaluar (acá, 18) y la cantidad de partidas a jugar (aquí, serán nuevamente 1000), y se debe devolver el porcentaje de partidas donde se terminó de jugar en hasta 18 rondas con respecto al total de partidas jugadas (para esto, se debe tomar la cantidad de partidas que terminaron en 18 o menos rondas, y y dividirla por la cantidad de partidas jugadas totales (en este caso, 10000).

    c) ¿Qué va a pasar con 20 jugadores?
    Primero se deberá pensar qué esperamos que pase, y luego realizar la simulación anterior pero con 20 jugadores. ¿Los resultados obtenidos coinciden con los anticipados?