TTPS Opción Ruby

Práctica 2: Ruby

Repasando la sintaxis

Nota: No usar while, for ni repeat.

1 - Si listamos todos los números naturales menores que 10 que son múltiplos de 3 o 5 obtenemos 3, 5, 6 y 9. La suma de todos estos números es 23. Encontrá la suma de todos los múltiplos de 3 o 5 menores que 1000.

2 - Cada nuevo término en la secuencia de Fibonacci es generado sumando los 2 términos anteriores. Empezando con 1 y 2; los primeros 10 términos son: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89. Considerando los términos en la secuencia de Fibonacci cuyos valores no exceden los 4 millones, encontrá la suma de los términos pares.

3 - Un número palíndromo se lee igual al derecho y al revés. El número palíndromo más grande obtenido de la multiplicación de dos números de 2 dígitos es 9009 (91 x 99 = 9009). Encontrá el palíndromo más grande obtenido a través de la multiplicación de dos números de 3 dígitos.

4 - 2520 es el número más chico que puede ser dividido por cada uno de los números del 1 al 10 sin obtener resto. ¿Cual es el número más chico que puede ser dividido por cada uno de los números del 1 al 20?

5 - La suma de los cuadrados de los primeros 10 números naturales es 385 ($1^2 + 2^2 + \ldots + 10^2 = 385$). El cuadrado de la suma de los primeros 10 números naturales es 3025 ($(1 + 2 + \ldots + 10)^2 = 55^2 = 3025$). Por lo tanto, la diferencia entre el cuadrado de la suma y la suma de los cuadrados de los primeros 10 números naturales es 2640 (3025 - 385 = 2640). Encontrá la diferencia entre el cuadrado de la suma y la suma de los cuadrados de los primeros 100 números naturales.

6 - La lista de los primeros 6 números primos es 2, 3, 5, 7, 11 y 13. Se puede observar que el 6to número primo es 13. ¿Cual es el número primo nro 10001?

7 - La suma de los primos menores que 10 es 17 (2 + 3 + 5 + 7 = 17). Encontrá la suma de todos los primos menores que 2 millones.

8 - Dado un arreglo de strings cualquiera, es necesario escribir un método que devuelva un arreglo con la longitud de dichos strings. Ejemplo: dado [‘Ruby’, ‘is’, ‘awesome’] debe retornar [4, 2, 7]

9 - Dado un color expresado como una combinación RGB calcular su representación entera. Consideramos que un color rgb se expresa como un hash con las claves [:red, :green, :blue], y para cada una toma valores en el rango (0..255). Por ejemplo:

- { red: 0, green: 0, blue: 255 },
- { red: 128, green: 128, blue: 255 },

La representación entera se calcula como: red + green*256 + blue*256²

9.b - Realizar el mismo cálculo obteniendo los coeficientes para cada componente del color de otro hash coefficients = { red: 256^0, green: 256^1, blue: 256^2 }

Clases y objetos

10 - Se quiere administrar una agenda electrónica. De cada contacto necesita guardarse:

- Nombre
- Fecha de nacimiento
- Email
- Teléfono
- Dirección

Se quiere hacer una aplicación de línea de comandos que permita:

- Ver todos los contactos
- Agregar un contacto
- Editar un contacto
- Buscar un contacto

Los datos deben ser guardados en un archivo CSV.

11 - Se quiere hacer un conversor de medidas:

- De pies a metros
- de metros a pies

Donde m = ft / 3.2808 (m = meters, ft = feets)

¿Que alternativas tenemos para implementar la solución? ¿Cuál considerás mejor? ¡Implementala!

12 -Crear un archivo de texto plano que tenga el siguiente contenido:

```
'001','Caja de sorpresas',52.50
'002','Viaje de ida al infinito y mas allá',120
'003','Historias de chillar',75
```

```
'003','Pegamento de personas',80
```

12.a - Crear una clase ProductsList que lo parsee y almacene la información en una colección de productos.

12.b - Teniendo en cuenta la lista de productos anterior, crear una clase Purchase que procese órdenes de compra. Dicha clase deberá:

- Implementar un método #add(producto) que agregue un producto a la orden. Si el producto no está en la lista, no lo agregará a la compra.
- Implementar un método #total que calcule el total de los elementos agregados.

12.c - Incorporar lógica de procesamiento de descuentos al sistema anterior de manera que si el total supera los $200, se aplique un descuento del 10%

12.d - Extender la funcionalidad para que le procesamiento de descuentos permita aplicar los siguientes descuentos:

- Si la compra totaliza más de $200 se aplica un 10% de descuento
- Si se llevan 2 unidades de "Historias de chillar" el precio del producto baja a $65

12.e - Extender las solución anterior de manera que se puedan agregar y modificar reglas descuentos y promociones

13 - En un juego de combate marcial participan 2 jugadores. Cada jugador posee una energía vital, la cual varía durante el combate. El combate se organiza en una serie de turnos, en cada turno el jugador decide qué movimiento realizar (golpear o bloquear). El combate dura como máximo 10 turnos, y cada jugador comienza el mismo con 100 puntos de energía. Al final del combate puede suceder que:

- Un jugador se quede sin energía, por lo tanto pierde (y gana su oponente).
- Ambos jugadores se quedan sin energía en el mismo turno, en este caso empatan.
- Ambos jugadores terminan el combate con energía mayor que 0. En este caso gana aquel con más energía. En cada turno los jugadores pueden bloquear o golpear. Por lo tanto de se pueden dar 3 casos:
    - El jugador A golpea y el jugador B bloquea: en este caso el que golpea (A) pierde 10 puntos de energía.
    - Ambos jugadores bloquean: en este caso la energía de ambos se mantiene igual.
    - Ambos golpean: el jugador A resta de su energía el 20% de la energía que tiene B, y el jugador B resta el 20% de la energía que tiene A.

A modo de ejemplo considere la siguiente secuencia de turnos:

Turno	Mov Jug A	Mov Jug A	Energía A	Energía B

0	-	-	100	100
1	Bloqueo	Bloqueo	100	100
2	Bloqueo	Golpe	100	90
3	Golpe	Golpe	82	70

Usted debe implementar un sistema (clase Juego) donde, dados 2 jugadores que pueden responder con un movimiento para cada turno, determine quién gana o si hubo empate. Su implementación debe respetar las siguiente indicaciones:

- La clase Jugador debe entender el mensaje "#jugada(nro_turno)" (donde nro_turno es un entero entre 1 y 10) que retorna la jugada para ese turno.
- La clase Jugador debe entender el mensaje "#nombre" que retorna el nombre del mismo.
- La clase Juego debe implementar el mensaje: "#determinar_ganador(jugador_a, jugador_b)"

13.a - Implemente.

13.b - Instancie los jugadores con sus correspondientes movimientos y verique quien gana.