Pensamiento Computacional - 1er Semestre 2022 **Tutor:** Darío Creado F.
Fecha: 09/06/2021

PRUEBA # 3 (PARTE I)

Librerías + Recursión y Orden + Data Science + ML + DL

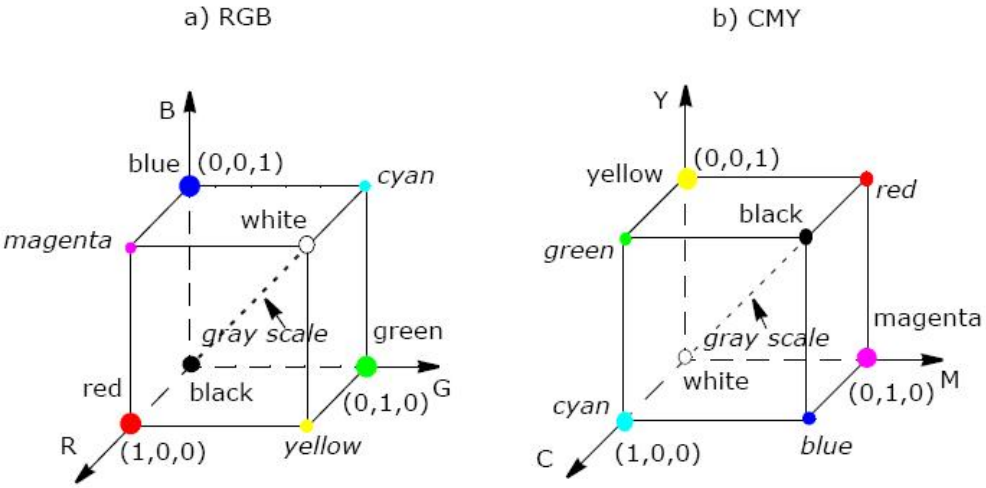
Instrucciones

1. La evaluación tiene una duración de **70** minutos (ó bloque completo de clases). Se considera el tiempo necesario para la repartición de enunciados, toma de asistencia y acomodo en los puestos.
2. La evaluación puede realizarse en forma **individual, pareja o grupo de tres estudiantes**. Puedes utilizar apuntes de elaboración propia escritos en el cuaderno. No se permite el uso de ningún artefacto electrónico durante la prueba (celular, computador, calculadora programable, etc.)
3. Solo se aceptan consultas a voz alzada. Por lo mismo, solo se aceptarán preguntan en forma excepcional (por ejemplo, posible error en el enunciado o clarificar supuestos). En caso de que persistan dudas, considera un supuesto razonable y trabaja sobre él.
4. Para las respuestas asociadas a preguntas con alternativas se pide justificar la elección. Respuestas correctas sin justificación, **no se considerarán en el puntaje**.

Preguntas

1) En computación es usual la utilización del sistema de colores **RGB**, acrónimo que describe al sistema de colores que tiene como base al *red = rojo*, *green = verde* y *blue = azul*. Los niveles para cada uno de estos colores oscilan discretamente entre [0-255] o de forma normalizada entre [0-1] (*estos rangos son equivalentes y dependen de quién los esté considerando*). De esta forma el color verde (*green*) equivale en **RGB** al vector [0,255,0] o equivalentemente [0,1,0]. Alternativamente, existe el modelo de color **CMY** el cuál funciona bajo la misma lógica, con la diferencia de que los *colores base* corresponden a *cyan, magenta, yellow* (existe la versión extendida **CMYK** en donde **K** hace referencia a *black*). Expresa la siguiente lista de colores por medio de sus vectores en los códigos **RGB** y **CMY**.

```
LISTA = [AZUL, CYAN, VERDE, AMARILLO, ROJO, MAGENTA, BLANCO, NEGRO]
```



2) Debes completar el programa que se muestra más abajo. Este programa modela la interacción entre los siguientes objetos: dos cocineros, un mesón, y dos mozos del restorán Don Yadran. Los cocineros ponen platos en el mesón para que sean retirados por los mozos. El mesón tiene un espacio limitado y los cocineros no pueden poner más platos cuando el mesón está lleno. Don Yadran es muy famoso y se especializa en solo dos tipos de platos: "lomitos" y "completos". El cocinero1 solo hace "lomitos" y el cocinero2 sólo hace "completos". Los mozos sacan platos del mesón (si es que los hay) y los ponen en una bandeja distribuidora que los llevan a los hambrientos clientes. El mozo1 sólo saca del mesón platos "lomitos" y el mozo2 sólo saca platos "completos". El siguiente programa ejecuta muchas iteraciones de la interacción entre cocineros,

mesón y mozos. En cada iteración, cada cocinero prepara y pone en el mesón una cantidad aleatoria de (entre cero y P) platos. En el programa, P vale 6 ("lomitos") para cocinero1 y 8 ("completos") para cocinero2. Si un cocinero debe poner x platos en el mesón, pero en éste sólo caben k, con $k < x$, sólo pone k platos.

En cada iteración, cada mozo saca del mesón una cantidad aleatoria de (entre cero y S) platos. En el programa, S vale 5 ("lomitos") para mozo1 y 4 ("completos") para mozo2. Si un mozo debe sacar x platos del mesón, pero en éste sólo quedan k, con $k < x$, sólo saca k platos. Debes completar este programa escribiendo las clases Meson, Cocinero, y Mozo, asegurando que funcione a cabalidad. No puedes modificar el código presentado. Debes definir las clases, atributos y métodos necesarios. El método lleno() retorna `True` si el mesón está lleno y `False` en caso contrario. El atributo faltan indica la cantidad de platos que no estaban disponibles en el mesón para ser retirados por un mozo (no se acumulan de una iteración a otra). El atributo tipo indica si es "lomito" o "completo". El atributo `lom` indica la cantidad de "lomitos" en el mesón. El atributo indica la cantidad de "completos" en el mesón.

```
### programa principal ###
meson=Meson(20) # crea un mesón de capacidad 20 platos
cocinero1 = Cocinero("lomito",6,meson) # crea cocinero con máx. 6 lomitos
cocinero2 = Cocinero("completo",8,meson) # crea cocinero con máx. 8 completos
mozo1=Mozo("lomito",5,meson) # crea mozo que retira máx. 5 lomitos
mozo2=Mozo("completo",4,meson) # crea mozo que retira máx. 4 completos

t=0 # ejecuta 50 iteraciones
while t < 50:
    cocinero1.agregaPlatos()
    cocinero2.agregaPlatos()
    print(" Mesón lleno : ", meson.lleno())
    mozo1.retiraPlatos()
    print(" Faltan : ", mozo1.faltan," ",mozo1.tipo)
    mozo2.retiraPlatos()
    print(" Faltan : ", mozo2.faltan," ",mozo2.tipo)
    t = t + 1
    print ("t = "+str(t) + " Meson: " + str(meson.lom) + ", " + str(meson.comp))
# fin de la iteración
```

3) El producto punto de dos vectores de n dimensiones $\vec{a}$ y $\vec{b}$ se define a continuación:

$$\vec{a} \cdot \vec{b} = \sum_{i=0}^{n} a_i \cdot b_i = C \quad C \in \mathcal{R}$$

donde C es un número escalar (real). Utilizando la librería `numpy` escriba una función que reciba dos arreglos de tres elementos cada uno y retorne su producto punto.

4) ¿Cuál de las siguientes líneas de código **dibujará una línea** de (0, 0) a (100, 100)?

```
    # a)
    pygame.draw.line(screen, GREEN, [0,0,100,100], 5)
    # b)
    pygame.draw.line(screen, GREEN, 0, 0, 100, 100, 5)
    # c)
    pygame.draw.line(screen, GREEN, [0, 0], [100, 100], 5)
    # d)
    pygame.draw.line(GREEN, screen, 0, 0, 100, 100, 5)
    # e)
    pygame.draw.line(5, GREEN, [0, 0], [100, 100], screen)
```

5) Analice el código de la siguiente funci´on que suma dos enteros positivos:

```
def suma ( x , y ) :
    if y == 0 :
        return x
    return suma (x+1, -y1)
```

a) ¿Cuál es el caso base?

b) ¿Qué pasa si se ejecuta con un x negativo?

c) ¿Qué pasa si se ejecuta con un y negativo?

d) ¿Cómo la cambiaría para que funcione con números enteros positivos y negativos?

6) Explica brevemente cuales son las diferencias entre los algoritmos de ordenamiento *quick sort* y *merge sort*.

7) Explica a qué corresponde el parámetro k dentro del algoritmo *k-Means* y cómo se escoge un valor adecuado para este parámetro. Por otro lado, explica brevemente en qué consiste este algoritmo.

8) Sea el conjunto de puntos $\{p_1, p_2, \cdots, p_N\}$ que representan el grado de molestia debido a la cantidad de asaltos observados durante el último tiempo en la avenida principal de la comuna percibido por cada persona encuestada en una campaña realizada por la municipalidad hace un par de años. De esta forma p_j corresponde al grado de molestia que presenta la persona j debido a la situación delictual descrita en una escala discreta [0-100] donde 100 corresponde al máximo grado de molestia. El alcalde de la comuna decide contratarlo debido a su excelente manejo en el lenguaje de programación *Python* y le encarga clasificar a las personas según categorías asociadas a lo realizado en la encuesta (el grado de molestia p podría ser de gran ayuda en esta tarea). Por otro lado, se espera que esta encuesta se repita de forma periódica para llevar

un mejor seguimiento de la situación. Por este motivo será útil generar una lista *ordenada* de todos los *grados de molestia* que presenta cada persona que participe de la encuesta. Explica y escribe algún código (no es necesario que sea 100 % funcional) que permita atender a las tareas solicitadas por la alcaldía. Utiliza lo visto en los temas de este apartado (Data Science, K-Means, ML, DL).

Respuesta Problema 1

Respuesta Problema 2

Respuesta Problema 3

Respuesta Problema 4 y 5

Respuesta Problema 6

Respuesta Problema 7

Respuesta Problema 8