TEC – Escuela de Computación – Elementos de computación
TAREA 3 PRÁCTICA PROGRAMACIÓN ESTRUCTURADA

PROGRAMACIÓN ESTRUCTURADA
ESTRUCTURAS SECUENCIAL, CONDICIONAL, ITERACIÓN

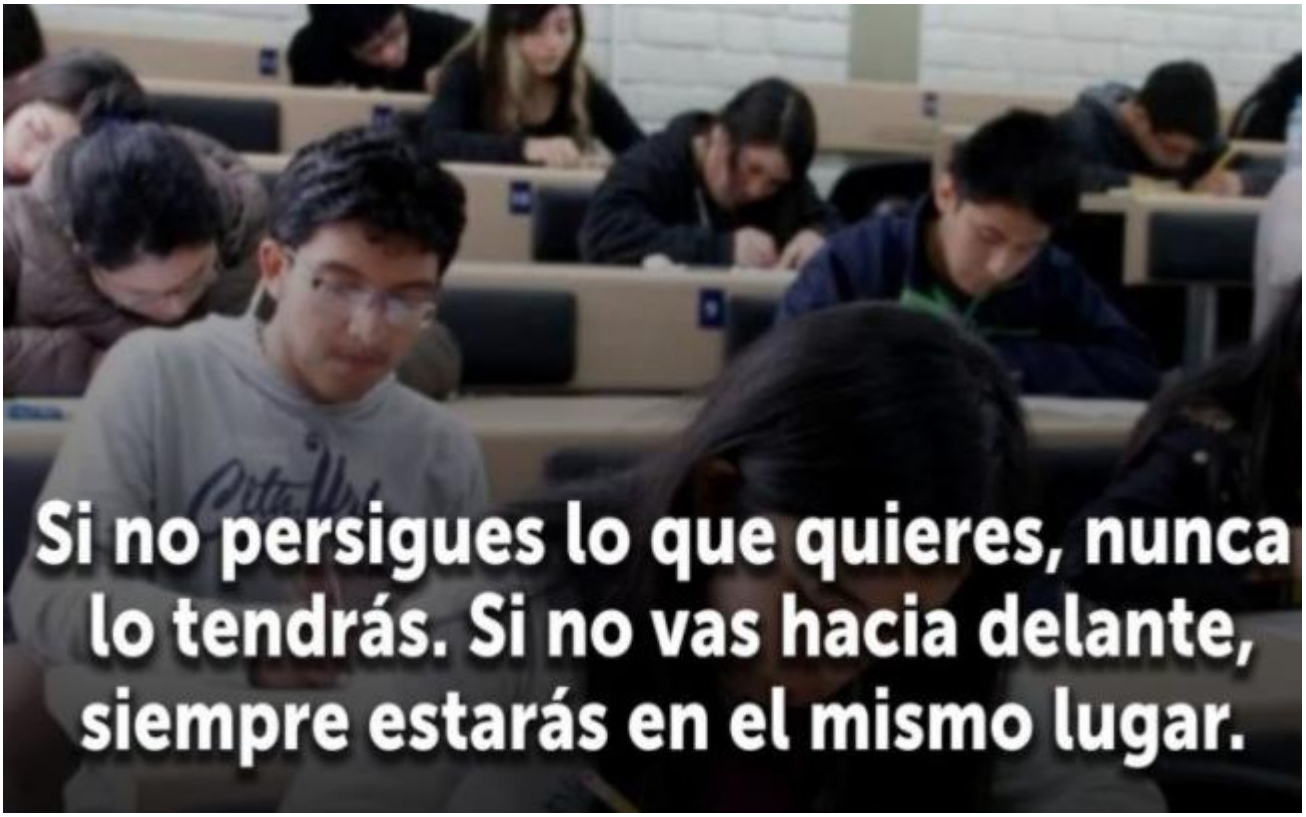
INSTRUCCIONES:

- Desarrollar en Python 3 en un archivo de nombre: tarea3_su_nombre.py.
- Use el material estudiado a la fecha de la evaluación. Solo se permiten usar los tipos de datos numéricos y booleanos. Valide las restricciones solo cuando se pida explícitamente.
- Enviar la solución al tecDigital EVALUACIONES / TAREAS. Revise que el trabajo enviado sea el correcto y a este destino.
- Requisitos para revisar la evaluación:
    - Las funciones deben tener: los nombres indicados en cada ejercicio, el número de cada ejercicio y deben estar en la solución ordenados de acuerdo a su número de ejercicio.
    - Buenas prácticas de programación:
        - Las funciones deben tener documentación interna (COMENTARIOS EN EL PROGRAMA FUENTE), al menos: lo que hace, entradas y sus restricciones, salidas. También ponga documentación interna en aquellas partes que ameriten una explicación adicional para un mejor entendimiento del programa fuente.
        - Nombres significativos de variables y funciones.
        - Reutilizar software: usar funciones que han sido desarrolladas previamente en esta misma evaluación.
- Ejercicios 4 y 7 valen 20 puntos, los otros valen 10 puntos.
- Fecha de entrega: 19 de abril, 11pm.
- Se coordinará día y hora para revisar la evaluación junto con el estudiante quien siendo el autor mostrará el dominio de la solución implementada desde el punto de vista técnico (uso de conceptos de programación y del lenguaje) así como de la funcionalidad (lo que hace la solución). La revisión puede constar de las siguientes actividades:
    - Revisar esta solución particular
    - Revisar conceptos incluidos en la evaluación
    - Aplicar otras actividades con una complejidad igual o menor a la evaluación.

RECOMENDACIÓN. Siga la metodología de solución de problemas: entender el problema, diseñar un algoritmo, codificar y probar programa. En la etapa de diseño del algoritmo haga un esquema para determinar el comportamiento o patrón del algoritmo, luego proceda con su desarrollo.
NO SE VAYA DIRECTAMENTE A PROGRAMAR EN LA COMPUTADORA, PRIMERO PLANIFIQUE LA SOLUCIÓN HACIENDO ESE ESQUEMA Y LUEGO HAGA EL PROGRAMA.

RECOMENDACIÓN. La universidad le brinda diferentes recursos en este proceso de enseñanza-aprendizaje que usted puede aprovechar: consultas presenciales (vía Zoom), tutorías, asistentes, correos.

Por favor compartan conocimientos e ideas. Para ello pueden trabajar en grupos de 3 personas, pero la presentación y evaluación del trabajo es individual. Es decir que cada miembro del grupo debe enviar el trabajo y su nota será de acuerdo a la revisión particular. Si trabaja en grupos escriba en el fuente su nombre como autor del trabajo y los nombres de los compañeros como coautores.



1) Siguiendo la metodología de desarrollo de programas desarrolle la función **factorial.** El factorial de un número n (denotado como n!) está definido solo para números naturales: es el producto de todos los números enteros desde 1 hasta n:

   n! = 1 * 2 * 3 * … * (n – 1) * n

   Por definición: 0! = 1

   Ejemplos del funcionamiento:

   >>> factorial(0) → 1 (por definición es 1)
   >>> factorial(1) → 1 (1*1)
   >>> factorial(4) → 24 (1*2*3*4)
   >>> factorial(6) → 720 (1*2*3*4*5*6)

   La función recibe un número natural y retorna su factorial.
   Validar restricciones (número entero >= 0), sino se cumplen retorna 0.

2) Siguiendo la metodología de desarrollo de programas construya la función **tabla_multiplicar**. La función lee tres números enteros (>= 0) e imprime la tabla de multiplicar del primer valor empezando en el segundo valor y terminando en el tercer valor. El segundo valor es <= tercer valor. Ejemplo del funcionamiento:

   ```
   >>> tabla_multiplicar()
   Tabla de multiplicar del número: 12
   Empieza en el número: 3
   Termina en el número: 6
   12 x 3 = 36
   12 x 4 = 48
   12 x 5 = 60
   12 x 6 = 72
   ```

3) Desarrolle la función **primo** que reciba un entero n >= 2 y retorne True si ese número es primo, de lo contrario retorne False. Use el método de divisiones sucesivas desde 2 hasta $\sqrt{n}$ sino hay divisores en ese rango el número es primo. Ejemplos del funcionamiento:

   primo(23)
   True

   Se probó desde 2 hasta $\sqrt{23}$ = 4: 23/2, 23/3, 23/4 → alto: no se encontraron divisores por tanto es primo

   primo(65)
   False

   Se probaría desde 2 hasta $\sqrt{65}$ = 8: 65/2, 65/3, 65/4, 65/5 → alto: se encontró el divisor 5 en este rango por tanto no es primo

4) Haga la función **rango_primos** que reciba dos enteros n y m (>= 2, n <= m) e imprima los números primos en ese rango. Ejemplos del funcionamiento:

   >>> rango_primos(9, 17)
   11
   13
   17

   >>> rango_primos(8, 10)
   No hay primos

   Implemente dos soluciones:
   a) Para determinar si un número es primo no use funciones adicionales: **USE CICLOS ANIDADOS. El primer ciclo es para ir por cada número del rango, un segundo ciclo anidado es para determinar si el número es primo.**
   b) Para determinar si un número es primo reutilice software: llame a la función ***primo***. Usa un ciclo para ir por cada número del rango.

5) Desarrolle la función **sumatoria** que reciba dos enteros: el inicio (m) y el fin (n) de la sumatoria, donde inicio <= fin. Debe retornar la suma de los números en ese rango. Valide las restricciones y retorne los mensajes respectivos en caso de errores.

$$\sum_{i=m}^{n} i = i + (i+1) + (i+2) + \ldots + (i+n)$$

   Ejemplos del funcionamiento:

   >>> sumatoria(10, 15)
   75 ← el resultado de 10+11+12+13+14+15

   >>> sumatoria(8.5, 15)
   ERROR: LAS ENTRADAS DEBEN SER NÚMEROS ENTEROS

   >>> sumatoria(250, 8)
   ERROR: EL INICIO DEL RANGO DEBE SER <= AL FIN DEL RANGO

6) Haga la función **contar_pares_impares** que reciba un número natural y retorne dos valores: el primero va a contener la cantidad de todos los dígitos pares y el segundo la cantidad de todos los dígitos impares que aparecen en el número de entrada. Validar que la entrada cumpla las restricciones, de lo contrario retornar el mensaje respectivo.

   Ejemplos del funcionamiento:
   >>> contar_pares_impares(123)
   (1, 2)

   >>> contar_pares_impares(2426)
   (4, 0)

   >>> contar_pares_impares(3557)
   (0, 4)

   >>> contar_pares_impares("78.97")
   ERROR: LA ENTRADA DEBE SER UN NÚMERO NATURAL

   >>> contar_pares_impares(-100)
   ERROR: LA ENTRADA DEBE SER UN NÚMERO NATURAL

7) En matemáticas la fórmula de la combinatoria sin repetición es:

$$C_{n,x} = \binom{n}{x} = \frac{n!}{x!\,(n-x)!}$$

   donde ***n*** es la cantidad de elementos en un conjunto y ***x*** es el tamaño de cada subconjunto que se puede formar. Cada subconjunto debe ser diferente y el orden de los elementos no interesa. Ejemplo: en un grupo de 6 estudiantes se quieren hacer subgrupos de 2 estudiantes, ¿cuántos subgrupos diferentes se podrían formar?

$$\frac{6!}{2!\,(6-2)!} = 15$$

   Desarrolle la función **combinatoria** que reciba ***n*** y ***x***, retorna el cálculo de la fórmula.

   Implemente dos soluciones:
   a) Para calcular los factoriales -n!, x!, (n – x)!- ponga la lógica respectiva.
   b) Para calcular los factoriales reutilice software: llame a la función **factoria**l.

8) En matemáticas la sucesión de Fibonacci es la sucesión infinita de números naturales:

   0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

   (Número de término: 1 2 3 4 5 6 7 8 9 10 11, …)

   Por definición el término número 1 de la sucesión es 0 y el término número 2 es 1. A partir de estos dos términos calculamos el término n-ésimo de la sucesión sumando los dos términos anteriores: término $_{n-1}$ + término $_{n-2}$

   Desarrolle la función ***fibonacci*** que calcule y retorne el n-ésimo término de esta sucesión, donde n es un entero >= 1. Ejemplos del funcionamiento:
   >>> fibonacci(10) ← calcular el término número 10
   34
   >>> fibonacci(8) ← calcular el término número 8
   13
   Última línea