INSTITUTO TECNOLÓGICO DE COSTA RICA
Escuela de Computación - Bachillerato en Ingeniería en Computación
IC-1802 Introducción a la Programación - Examen Parcial
Prof. Mauricio Avilés

Total de puntos: 32 Tiempo: 2 horas

Instrucciones:
Lea cuidadosamente los casos proporcionados, y basándose en la teoría y práctica vista en clases, aplique su criterio para responder las preguntas indicadas de manera individual.
Este examen debe resolverse mediante el uso de la computadora.
Cualquier comportamiento deshonesto durante la prueba será sancionado con la nota mínima.
Debe entregarse un único archivo .ZIP con las soluciones de los problemas propuestos. La entrega se hacer por medio del TEC-Digital, en las evaluaciones del curso.

Nombre: _____

Carné: _____

Las respuestas de este examen serán resultado de mis decisiones individuales. No usaré, recibiré, ni ofreceré ayuda no autorizada. No copiaré de otros exámenes, ni permitiré que nadie copie parte alguna de este examen. No realizaré ninguna trampa ni procedimiento deshonesto. Juro por mi honor que todo lo anterior es cierto.

Firma: _____

Parte I – Problemas

Para cada uno de los problemas presentados debe cumplirse lo siguiente:

- Entregarse documentación interna para cada subrutina (descripción, entradas, restricciones y salidas) y la solución al problema en código Python.
- Las subrutinas que utilicen recursión deben dividirse en función principal y auxiliar.
- No escriba código adicional al solicitado.
- No lea entradas o realice impresiones a menos que el enunciado se lo solicite explícitamente.
- Las subrutinas utilizadas deben lanzar excepciones en caso de que las restricciones no se cumplan.
- Los programas principales de cada problema deben utilizar el manejo de errores adecuado para que nunca terminen con un error, deben atraparse todos los errores que puedan suceder y darse un mensaje amigable para el usuario.

1. Suma Estacionaria

   La suma estacionaria de un número N es la suma de todos sus dígitos, pero si este resultado es un número de más de un dígito, se repite el mismo procedimiento sobre el resultado hasta que se obtenga un número de un dígito. Ejemplo: 16542 → 18 → 9.
   Escriba un programa que lea un número entero no negativo e imprima el resultado de su suma estacionaria.
   El programa debe invocar una función recursiva (sin ciclos) llamada **sumaEstacionaria(n)** que se encargue de hacer el cálculo e imprimir su resultado.

   Ejemplos de pruebas con la función recursiva:

```
>>> sumaEstacionaria(51954)
6
>>> sumaEstacionaria(123215)
5
>>> sumaEstacionaria(3)
3
>>> sumaEstacionaria(9**90)
9
```

   Evaluación:

Documentación (ESR):	**1 punto**
Restricciones y manejo de errores:	**2 puntos**
Resto del algoritmo:	**8 puntos**
Total:	**11 puntos**

2. Familias de números enteros

Escriba un programa que lea una serie de números enteros para analizar y una serie de factores (enteros positivos). La lectura la puede realizar de dos formas: (1) pidiéndole al usuario que indique cuántos serán los números a leer y luego repitiendo un input para cada uno, o (2) pidiéndole que escriba directamente la lista de números enteros y utilizar la función **eval** para convertirlo a lista. Estas dos listas deben enviarse a una función llamada **familias(enteros, factores)** que reciba como entradas la lista de números enteros a procesar y la lista de números enteros positivos que son los factores. La función debe retornar un diccionario donde cada elemento de la lista de factores es una llave y como valor asociado se encuentra una lista con todos los enteros que son divisibles por la llave. La función debe lanzar excepción en caso de que no se cumplan las restricciones. Puede utilizarse iteración y/o listas por comprensión.
El programa principal debe imprimir los contenidos del diccionario retornado por la función.

Ejemplos de pruebas de la función familias:

```
>>> familias([45, 23, 91, 76, 25, 77, 13, 66, 48], [2, 3])
{2: [76, 66, 48], 3: [45, 66, 48]}
>>> familias([45, 23, 91, 76, 25, 77, 13, 66, 48], [7, 5])
{7: [91, 77], 5: [45, 25]}
>>> familias([45, 23, 91, 76, 25, 77, 13, 66, 48], [9, 10, 11, 13])
{9: [45], 10: [], 11: [77, 66], 13: [91, 13]}
>>> familias([1, 2, 3, 4, 5, 6, 7, 8], [2, 2, 2, 2])
{2: [2, 4, 6, 8]}
```

Evaluación:

Documentación (ESR):	**1 punto**
Restricciones y manejo de errores:	**2 puntos**
Resto del algoritmo:	**8 puntos**
Total:	**11 puntos**

3. Borrado recursivo

Escriba una función recursiva llamada **borradoAnidado(L, e)** que reciba por parámetro una lista L de elementos de cualquier tipo que puede contener otras listas con cualquier cantidad de anidamientos y un elemento e de cualquier tipo que va a ser eliminado de la estructura. La función debe retornar la misma estructura de listas anidadas sin ninguna de las ocurrencias del elemento e. La función debe eliminar todas las ocurrencias del elemento e, no sólo la primera. Los elementos en la lista y el buscado pueden ser de cualquier tipo. Utilice recursión para resolver este problema, se permite combinarla con ciclos.

En este problema no hace falta hacer un programa principal, sólo la función.

```
>>> borradoAnidado([[1, 2], [3], [5, 6, [7, 1, 2, [3]], 8]], 8)
[[1, 2], [3], [5, 6, [7, 1, 2, [3]]]]
>>> borradoAnidado([[1, 2], [3], [5, 6, [7, 1, 2, [3]], 8]], 1)
[[2], [3], [5, 6, [7, 2, [3]], 8]]
>>> borradoAnidado([[1, 2], [3], [5, 6, [7, 1, 2, [3]], 8]], 3)
[[1, 2], [], [5, 6, [7, 1, 2, []], 8]]
>>> borradoAnidado([[1, 2], [3], [5, 6, [7, 1, 2, [3]], 8]], 10)
[[1, 2], [3], [5, 6, [7, 1, 2, [3]], 8]]
>>> borradoAnidado([1, 2, 3], 0)
[1, 2, 3]
```

Evaluación:

Documentación (ESR):	**1 punto**
Chequeo de restricciones:	**1 punto**
Resto del algoritmo:	**8 puntos**
Total:	**10 puntos**