TEC | Tecnológico de Costa Rica

Instituto Tecnológico de Costa Rica **Área Académica de Ingeniería en Computadores** **Programa de Licenciatura en Ingeniería en Computadores** **Curso:** CE-1101 Introducción a la programación **Profesor:** Lic. Ing. Fabián Zamora Ramírez **Semestre:** I, 2018	**Práctica de temas:** • Programación recursiva con Python. Listas.

INDICACIONES:

Para todos los siguientes ejercicios muestre su solución en lenguaje de programación Python. Toda función debe estar documentada con una descripción breve de lo que realiza y mostrar sus entradas (E), salidas (S) y restricciones (R). Si no se especifican restricciones, debe realizar las restricciones lógicas según el problema. Recuerde utilizar nombres significativos para sus funciones y variables. En caso de no cumplir con lo anterior se calificará que nota de cero.

1. Escriba una función a llamar **iota(num)** que reciba un número natural y obtenga una lista en orden ascendente de números naturales menores al número dado.

>>> iota(5)
[0, 1, 2, 3, 4]
>>> iota(1)
[0]

2. Escriba una función **cuales(num)** que reciba un número (que debe ser entero) y retorne una lista con los dígitos entre 0 y 4.

>>> cuales(482401)
([4,2,4,0,1])
>>> cuales(0)
([0])
>>> cuales(97)
([])

3. Escriba una función a llamar **eliminar(lista, ele)** que reciba una lista y un elemento y elimine la primera aparición de ese elemento en la lista.

>>> eliminar([0, 5, 1, 2, 5, 3, 4], 5)

[0, 1, 2, 5, 3, 4]

>>> eliminar([0, 5, 1, 2, 5, 3, 4], 8)

[0, 5, 1, 2, 5, 3, 4]

4. Escriba una función a llamar **eliminar_todos(lista, ele)** que reciba una lista y un elemento y elimine todas las apariciones de ese elemento en la lista.

>>> eliminar_todos([0, 5, 1, 2, 5, 3, 4], 5)

[0, 1, 2, 3, 4]

>>> eliminar_todos ([0, 5, 1, 2, 5, 3, 4], 8)

[0, 5, 1, 2, 5, 3, 4]

5. Escriba una función recursiva a llamar **frente2(lista)**, que reciba una lista de números y devuelva la lista de todos sus elementos exceptuando los dos últimos. Si la lista tiene dos o menos elementos, debe retornar la lista nula.

>>> frente2([0, 1, 2, 3, 4])

[0, 1, 2]

6. Hacer una función llamada **invierta(lista)**, que reciba una lista e invierta el orden de sus elementos (sin utilizar la función reverse de Python):

>>> invierta([0, 1, 2, 3, 4])

[4, 3, 2, 1, 0]

7. La sumatoria de cocientes tiene la siguiente fórmula:

$$\sum_{i=1}^{n} i/(i*(i+1))$$

Escriba una función **suma_coc(n)** que reciba el límite superior de la sumatoria y calcule el resultado hasta ese número.

8. La siguiente sucesión numérica recibe únicamente valores de n enteros mayores o iguales a cero. Programe esta sucesión numérica.

f(0) = 0

f(1) = 1

*f(n) = 2 * f(n-1) + f(n-2)*

9. Escriba una función recursiva llamada **elimine_todos(ele, lista)** que reciba un elemento y una lista y elimine todas las apariciones del elemento en la lista, aún si el elemento se encuentra en una sublista (a cualquier nivel).

>>> elimine_todos(4, [4, 5, [[[3, 4], 5], [3]], 4, 8])

[5, [[[3], 5], [3]], 8]

10. Se define el doble factorial de un número n como:

$$n!! = \begin{cases} 1, & \text{si } n = 0 \text{ o } n = 1; \\ n \times (n-2)!! & \text{si } n \geq 2. \end{cases}$$

Escriba una función llamada **doble_fact(n)** que calcule n!!. La función debe comportarse de la forma siguiente:

>>> doble_fact(8)

384

11. Analice el siguiente código de una función Python:

```
def foo(x, l):
    if l == []:
        return []
    elif x == l[0]:
        return ['x'] + foo(x, l[1:])
    else:
        return [l[0]] + foo(x, l[1:])
```

Realice la siguiente ejecución manual de la función e indique que problema resuelve:

```
>>> foo('a', ['a', 'b', 'c', 'a', 'c', 'c'])
```

12. En una lista se reciben las notas de n estudiantes de un curso. Escriba una función recursiva notas(lista) que considerando que la nota de aprobación sea 70 y obtenga:

a) Cuántos estudiantes aprobaron el curso.

b) Cuantos estudiantes reprobaron.

c) El promedio de notas.

13. Escriba una función **ultimo_par(lista)** que recibe una lista no nula y retorna el último número par de la lista dada. Asuma que la lista está compuesta solo por números. La función debe mostrar un comportamiento similar a los siguientes ejemplos:

```
>>> ultimo_par([23, 72, 149, 34])
34
>>> ultimo_par([12, 5, 21, 3])
12
>>> ultimo_par([1, 3, 5, 7])
None
```

14. La conjetura de Ulam genera una sucesión de números basado en la siguiente fórmula:

a. Inicia con cualquier número positivo.

b. Si el número es par, se divide entre 2, si es impar se multiplica por 3 y se le suma 1.

c. Se obtienen enteros sucesivamente hasta llegar a obtener el número 1.

Escriba una función **ulam(n)** que reciba un número, que va a ser el inicial y obtenga la Sucesión de Ulam.

15. Analice la siguiente serie numérica:

2, 5, 7, 10, 12, 15, 17, ...

Determine su fórmula y construya una función recursiva llamada **serie(n)** que reciba como argumento un número y genere la serie desde 2 hasta el número dado.

16. Analice el siguiente código escrito en Python:

```
def i(l1, l2):
    if l1 == []:   # condición de terminación
        return l2
    elif l2 == []:          # condición de terminación
        return l1
    else:
        return [l1[0]] + [l2[0]] + i(l1[1:], l2[1:])
```

Realice la siguiente ejecución manual de la función e indique que problema resuelve:

>>> i([1, 3, 5], [2, 4, 6])

17. Hacer una función recursiva llamada **invertir(lista),** que reciba una lista e invierta el orden de sus elementos:

a. Usando sublistas
b. Sin utilizar sublistas

18. Escriba una función **cuente_pares_lista(lista)** que cuente los elementos pares en listas anidadas.

>>> cuente_pares_lista([1, 4, [5, 2, [2,2,1]]])

4

19. Escriba una función **coincide(lista)** que recibe una lista de números enteros e indique si algún elemento de la lista coincide con la suma de todos los que le preceden.

>>> coincide([2, 4, 3, 9, 14]) # 2 + 4 + 3 = 9

True

>>> coincide([5, 6, 14, 18, 20, 41])

False

20. El siguiente código en Python resuelve un determinado problema:

```
def misterio(l):
    if l[2:] == []:
        return l[0] == l[1]
    else:
        return misterio(l[1:])
```

Realizar la prueba manual de las siguientes ejecuciones:

>>> misterio([8, 6, 4, 8, 1, 1])

>>> misterio([5, 6, 14, 8, 10, 1])

Indique cuál es el problema que resuelve la función misterio.

21. Un cuadrado semi-mágico es aquel en que todas las sub-listas (filas) que lo componen suman lo mismo. Este concepto se aplica en listas formadas por sublistas del mismo tamaño. Por ejemplo, la lista: *[[6, 2, 1], [5, 4, 0], [4, 4, 1]]* sería un cuadrado semi-mágico. Escriba una función **semi_magico(lista)** que reciba una lista de listas, asumiendo que vienen en forma correcta y verifique si el argumento dado es o no un cuadrado semi-mágico.

22. Escriba una función amigos(num) que reciba un número entero y encuentre todas las parejas de números amigos que son inferiores al número dado. Se dice que dos números son amigos si cada uno de ellos es igual a la suma de los divisores del otro. Por ejemplo 220 y 284 son dos números amigos, ya que los divisores de 220 son:

1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284.

Los divisores de 284 son:

1 + 2 + 4 + 71 + 142 = 220.

23. Se desea recibir una lista no nula y devolver otra compuesta por dos sublistas, la primera tendrá a los elementos que ocupan un lugar impar, mientras que la segunda contendrá a los que ocupan un lugar par. Escriba la función **separa(lista)** utilizando recursividad, de forma que se obtengan resultados como los siguientes:

```
>>> separar(['a', 'b', 'c', 'd', 'e'])
[['b', 'd'], ['a', 'c', 'e']]
```

24. Escribir una función llamada concordar que reciba como argumentos 3 listas: una lista patrón, una lista sustitución y una lista original. Se debe sustituir la aparición de los elementos de la lista patrón en la lista original por el respectivo elemento en la lista sustitución. Note que el largo de la lista patrón y la lista sustitución deben ser iguales.

```
>>> concordar([], [], [1, 2, 3, 4, 5])
[1, 2, 3, 4, 5]
>>> concordar([1, 2, 3], ['a', 'b', 'c'], [2, 4, 1, 5, 6,  3])
['b', 4, 'a', 5, 6, 'c']
```

25. Escriba una función llamada **natural(lista)** que reciba una lista de números desordenada y produce una lista de listas, donde cada sublista mantiene el orden natural de la lista original. Por ejemplo:

```
>>> natural([3, 4, 5, 1, 2, 3, 4, 7, 3, 2, 1])
[[3, 4, 5], [1, 2, 3, 4, 7], [3], [2], [1]]
```

26. Escriba una función recursiva **elimine(lista, num)** que reciba una lista y un número y elimine de la lista los números que coincidan con el número dado, a excepción de la primera aparición.

```
>>> elimine([4,8,2,4,0,1], 4)
[4,8,2,0,1]
>>> elimine([5,2,3], 6)
[5,2,3]
```

27. Escriba una función **revise_num(num)** que reciba un número (que debe ser entero) y retorne una tupla que tenga la siguiente forma:

(cantidad-dígitos-entre 0 y 4, cantidad-dígitos-entre 5 y 9)

```
>>> revise_num(482401)
(5, 1)
>>> revise_num(0)
(1, 0)
>>> revise_num(4)
(1, 0)
```

28. Escriba una función **intersec(lista1, lista2)** que recibe 2 listas y forma una nueva lista con los números de la primera lista que están contenidos en la segunda lista.

```
>>> intersec([2,4], [4,2])
[2,4]
>>> intersec([3,3,3], [3])
[3,3,3]
>>> intersec([1,2,3], [6,9,2])
[2]
>>> intersec([4,3],[2,0])
[]
```

29. Escriba una función recursiva **salarios(lista)** que reciba n salarios de los empleados de una empresa. Debe obtener (en una lista) cuántos salarios hay menores a 999,999, cuántos mayores a un millón y el promedio de salarios.

```
>>> salarios([1000000,470000,2500000])
[1, 2, 1323333]
```