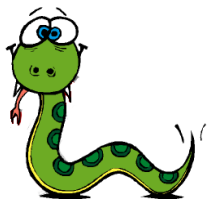
Instituto Tecnológico de Costa Rica
Escuela de Computación
Introducción a la Programación / Taller de programación.
Profesor: Jeff Schmidt Peralta



EJERCICIOS INICIALES SOBRE RECURSIVIDAD.

1 Revisión de conceptos.

1. Defina los siguientes conceptos:
   - Recursividad
   - Condición de terminación o caso base
2. Describa las razones que pueden provocar una recursividad infinita.
3. Explique las razones que justifiquen utilizar una función principal y una auxiliar en la solución de un problema por medio de la recursividad.

2 Ejercicios prácticos.

1. Escriba una función `forme1` que reciba un número (que debe ser entero) y retorne un nuevo número formado por los dígitos que sean 1.

```
>>> forme1(482401)
1
>>> forme1(4)
0
```

```
>>> forme1(182101)
111
```

2. Escribir una función num_append(num1, num2) que encadene los dígitos del segundo número con el primer número, de acuerdo al comportamiento siguiente:

```
>>> num_append(124, 56)
12456
>>> num_append(2340, 0)
23400
```

3. La operación de multiplicar dos números `a * b`, cuando `b` es un número entero, puede calcularse recursivamente por medio de `b - 1` sumas sucesivas del número `a`. Por ejemplo la multiplicación 8 * 4, podría verse como:
   `8 + 8 + 8 + 8 (tres sumas sucesivas de 8)`
   Escriba una función recursiva `multi(num1, num2)` que reciba dos números e implemente la multiplicación por medio de sumas sucesivas. La ejecución de la función debe mostrar resultados como los siguientes:

```
>>> multi(8, 4)
32
>>> multi(6.3, 3)
18.899999999999999
>>> multi(6.3, 3.4)
'Error: segundo argumento debe ser entero'
```

4. Escriba una función booleana suma10(num) que recibe un número entero y verifica si la suma de sus dígitos es mayor o igual a 10. La función debe comportarse de la siguiente forma:

```
>>> suma10(80642)
True
>>> suma10(200412)
False
```

5. Escriba una función `sume_parimpar` que reciba un número (que debe ser entero) y retorne una tupla que tenga la siguiente forma:
   (suma-dígitos-pares, suma-dígitos-impares)
   SUGERENCIA: usar funciones lambda.
   La función debe retornar su resultado en forma similar al siguiente ejemplo:

```
>>> sume_parimpar( 482401)
(18, 1)
>>> sume_parimpar(4)
(4, 0)
```

6. Escriba una función Python `clasifique(num)` que determine si el número dado es deficiente, perfecto o abundante. Se dice que un número entero positivo es perfecto si la suma de sus divisores exactos (excepto él mismo) es igual a dicho número, si la suma de sus divisores exactos es menor se dice que es deficiente y si esa suma de divisores es mayor entonces es abundante.

```
>>> clasifique(28)          >>> clasifique(30)
'Perfecto'                  'Abundante'
```

7. Escriba una función booleana `dig_ab(num1, num2)` que recibe 2 números enteros A, B y retorna True si todos los dígitos del primer número están contenidos en el segundo número y False en otro caso. La función debe retornar resultados como los ejemplos siguientes:

```
>>> dig_ab(24, 42)
True
>>> dig_ab(333, 3)
True
>>> dig_ab(3, 3333)
True
>>> dig_ab(123, 632)
False
```