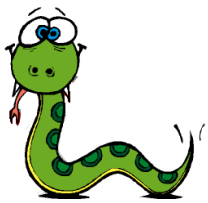
Instituto Tecnológico de Costa Rica
Escuela de Computación
Introducción a la Programación / Taller de programación.
Profesor: Jeff Schmidt Peralta



EJERCICIOS INICIALES SOBRE RECURSIVIDAD. II SEMESTRE 2019

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. (debe realizar la prueba manual de cada ejercicio)

1. Escriba una función booleana `largo(num)` que recibe un número entero y cuenta los dígitos en el número. La función debe comportarse:

```
>>> largo(93235)
5
>>> largo(735)
3
```

2. Escriba una función `cuente_dig(num, dig)` que recibe un número entero y un dígito válido y cuenta las veces que aparece el dígito en el número. La función debe comportarse de la siguiente forma:

```
>>> cuente_dig(93235, 3)
2
>>> cuente_dig(93735, 2)
0
```

3. Escriba una función recursiva de pila `cuente_par(num)` que reciba un número entero y cuente los dígitos que sean pares.

```
>>> cuente_par(482401)
5
```

```
>>> cuente_par(3579)
0
```

4. Escriba una función booleana `iguales(num)` que recibe un número entero y verifica si el primer y el último dígito del número son iguales. La función debe comportarse de la siguiente forma:

```
>>> iguales( 80642)
False
>>> iguales(351733)
True
```

5. Escriba una función recursiva de pila `suma_impar(num)` que reciba un número entero y sume los dígitos que sean impares.

```
>>> suma_impar(482401)
1
```

```
>>> suma_impar(3579)
24
```

6. Encuentre los errores de sintaxis en la siguiente función e indique las e60ntradas, salidas, restricciones y que hace la función.

```
def q(n):
    if isinstance(n, int)
         return q_aux(abs(n)
    else:
        return "Error"

def q_aux(n):
    if n == 0:
        return 0:
    elif (n % 10 == 1):
        return q_aux(n // 10)
    else:
        return n % 10 + q1_aux(n // 10)
```

7. Escriba una función booleana `todos_pares(num)` que recibe un número entero y verifica si el número dado tiene todos sus dígitos pares. La función debe comportarse de la siguiente forma:

```
>>> todos_pares(80642)
True
>>> todos_pares(201462)
False
```

8. Escriba una función booleana `hay_par(num)` que recibe un número entero y verifica si el número dado tiene al menos un dígito par. La función debe comportarse de la siguiente forma:

```
>>> hay_par(80642)
True
>>> hay_par(351733)
False
```

9. Escriba una función booleana `tiene_nueve(num)` que recibe un número entero y verifica si el número dado tiene al menos un dígito nueve. La función debe comportarse de la siguiente forma:

```
>>> tiene_nueve(900035)
True
>>> tiene_nueve(83735)
False
```

10. 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
```

11. 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
```

12. Escriba una función `revise_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)

    La función debe retornar su resultado en forma similar al siguiente ejemplo:

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

13. 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)

    La función debe retornar su resultado en forma similar al siguiente ejemplo:

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