ITCR – Escuela de Computación
Carrera: Ingeniería en Computación
Curso: Taller de Programación, I semestre 2015, Grupo 4
Tarea 2: Prácticas de programación

Indicaciones

- Siga la metodología de solución de problemas: entender el problema, diseñar un algoritmo, codificar y probar programa.
- Los productos entregables son los programas codificados en Python 3.x.
- Recuerde usar buenas prácticas de programación como nombres significativos, reutilización de código, documentación interna, eficiencia, etc.
- Considere que además de la función principal puede necesitar otras funciones.
- Son algoritmos numéricos, no se pueden usar funciones de: secuencias (str, list, etc), diccionarios, conjuntos.
- Valide las restricciones: tipo de datos y valores de los datos.
- Los procesos repetitivos usan la técnica de iteración; cada ejercicio indica la estructura a usar en caso de ser necesaria: while o for. Sino esta la indicación decida Usted la que es más apropiada.
- Las funciones se van a probar desde el modo comando.
- Todas las funciones deben ser definidas en el programa fuente **tarea2.py** el cual será enviado por medio del tecDigital en la sección de Evaluaciones. Note que esta tarea se convierte a la vez en un módulo ya que solo hay definiciones.
- Fecha máxima de entrega: 15 de marzo (24 horas).

1. Haga la función **al_reves** que recibe un entero y lo retorna en otra variable pero al revés (while).

   Ejemplos del funcionamiento:

   al_reves(357)
   753

   al_reves(987.1)
   ERROR: LA ENTRADA DEBE SER UN ENTERO

   al_reves(123456789)
   987654321

   al_reves(22322)
   22322

2. Haga la función **palindromo** que recibe un entero y retorna True si ese número es palíndromo o False de lo contrario. Un número es palíndromo si lo leemos igual de izquierda a derecha o de derecha a izquierda. Para reutilizar código use la función anterior.

   Ejemplos del funcionamiento:

   palindromo(22322)
   True

   palindromo(1800081)

True

palindromo(123456)
False

palindromo(-3443)
True

palindromo(0)
True

3. Haga la función **rombo** que reciba un entero (>=2) que representa el tamaño de cada uno de sus cuatro lados e imprima un rombo de "*". Una posible solución es imprimir caracteres blanco (" ") para alinear la figura. Ejemplos del funcionamiento:

rombo(5)

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

rombo(-10)
ERROR: ENTRADA DEBE SER UN ENTERO >= 2

rombo("abc")
ERROR: ENTRADA DEBE SER UN ENTERO >= 2

4. Haga la función **imprime_tabla** para imprimir (no retorna valores) la tabla de multiplicar de un número, iniciando y terminando donde indique el usuario. La función recibe tres números enteros mayores o iguales a uno. (for). Ejemplo del funcionamiento:

imprime_tabla(3, 20, 23)
Tabla de multiplicar del: 3
Iniciando en: 20
Terminando en: 23
3 x 20 = 60
3 x 21 = 63
3 x 22 = 66
3 x 23 = 69

5. Haga la función **serie_adelante** para obtener un solo número que contenga todos los números enteros sucesivos (de 1 en 1) que se encuentren en un rango. Recibe 2 entradas: el inicio del rango y el fin del rango. Retorna un número conteniendo todos los números enteros en el rango indicado. Los números deben ponerse en el resultado secuencialmente de izquierda a derecha.

Proceso:
Debe generar la serie de números enteros sucesivos (de 1 en 1) en el rango indicado y formar un solo número con esa serie. El primer número de la serie será el número de más a la izquierda en el número resultante, luego se le pega a la derecha el segundo número de la serie y así sucesivamente hasta el último número en el rango. (for)

Ejemplos del funcionamiento:
serie(98, 101) → 9899100101
serie(8, 11) → 891011
serie(997,1005) → 997998999100010011002100310041005

6. Haga la función **serie_atras** para obtener un solo número que contenga todos los números enteros sucesivos (de 1 en 1) que se encuentren en un rango. Recibe 2 entradas: el inicio del rango y el fin del rango. Retorna un número conteniendo todos los números enteros en el rango indicado. Los números deben ponerse en el resultado secuencialmente de derecha a izquierda.
   Proceso:
   Debe generar la serie de números enteros sucesivos (de 1 en 1) en el rango indicado y formar un solo número con esa serie. El primer número de la serie será el número de más a la derecha en el número resultante, luego se le pega a la izquierda el segundo número de la serie y así sucesivamente hasta el último número en el rango. (for)

   Ejemplos del funcionamiento:
   serie(98, 101) → 1011009998
   serie(8, 11) → 111098
   serie(997,1005) → 100510041003100210011000999998997

7. Haga la función **intersecta** que reciba dos números enteros (>=0) y retorne los dígitos que están presentes en ambos números de entrada. (while)
   El programa recibe 2 entradas:
   un número sin dígitos repetidos
   un número sin dígitos repetidos
   Retorna (salidas):
   un número conteniendo solo los dígitos que estén presentes en ambos números de entrada. El orden de los dígitos en la salida debe seguir el orden que tengan en el primer número de entrada de izquierda a derecha. Retorna False sino hay dígitos en común.
   Proceso:
   Analizar los dos números de entrada y formar un nuevo número que va a contener los dígitos que estén presentes en ambos números de entrada.
   Sino hay dígitos en común lo que debe retornar es el valor booleano False.

   Ejemplos del funcionamiento:
   intersecta(1426, 20813) → 12
   intersecta(123, 0) → False
   intersecta(0, 1234) → False
   intersecta(2498, 135) → False
   intersecta(450, 0) → 0

8. Haga la función **imprime_p_i_w** que recibe dos enteros, el inicio y el fin de un rango, y un string que va a tener el valor “par” o “impar”. La función debe imprimir todos los números, pares o impares dependiendo del tercer parámetro recibido, en el rango recibido. (while). Para imprimir en una misma línea use la opción end en el print.

   Ejemplos del funcionamiento:
   imprime_p_i_w(10, 15, “par”)
   10, 12, 14

   imprime_p_i_w(10, 15, “impar”)
   11, 13, 15

9. Haga la función **imprime_p_i_f** que recibe dos enteros, el inicio y el fin de un rango, y un string que va a tener el valor “par” o “impar”. La función debe imprimir todos los números, pares o impares dependiendo del tercer parámetro recibido, en el rango recibido. (for).

   Ejemplos del funcionamiento:
   imprime_p_i_f(10, 15, “par”)
   10, 12, 14

   imprime_p_i_f(10, 15, “impar”)
   11, 13, 15

10. Desarrolle la función **doble_factorial** que reciba un número n que va a ser entero mayor o igual a 0 y retorne su doble factorial, el cual se denota así: n!! (solo un for)

    El doble factorial para estos números lo definimos así:

    Si n es 0 el doble factorial es: 1

    Si n es impar el doble factorial es: 1 x 3 x 5 x ... x (n- 2) x n, es decir, se multiplican todos los impares en el rango de 1 hasta n.)

    Si n es par el doble factorial es: 2 x 4 x 6 x ... x (n – 2) x n, es decir, se multiplican todos los pares en el rango de 2 hasta n

    Ejemplos del funcionamiento:
    doble_factorial(9) → salida: 945
    (el resultado se obtuvo multiplicando los impares desde 1 hasta 9, así: 1 * 3 * 5 * 7 * 9)

    doble_factorial(8) → salida: 384
    (el resultado se obtuvo multiplicando los pares desde 2 hasta 8, así: 2 * 4 * 6 * 8)

    doble_factorial(0) → salida: 1

11. Haga la función **sumatoria_w** que reciba dos enteros: el inicio y el fin de la sumatoria (inicio <= fin). Debe retornar la suma de los números en ese rango. (while).

    Ejemplo del funcionamiento:
    sumatoria_w(10, 12) → 33 (resultado de 10+11+12)

sumatoria_w(50, 25) → ERROR: EL INICIO DEBE SER <= QUE EL FINAL

12. Haga la función **sumatoria_f** que reciba dos enteros: el inicio y el fin de la sumatoria (inicio <= fin). Debe retornar la suma de los números en ese rango. (for).

    Ejemplo del funcionamiento:
    sumatoria_f(10, 12) → 33 (resultado de 10+11+12)

13. En matemáticas la sucesión de Finonacci es una sucesión infinita de números naturales que empiezan con los términos 0 y 1, luego cada siguiente término se calcula sumando los dos términos anteriores:

    0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

    Haga la función **fibo** que recibe un entero n (>0) e imprima los primeros n términos de la sucesión de Fibonacci (while). Ejemplos del funcionamiento:

    fibo(1)
    0

    fibo(2)
    0, 1

    fibo(3)
    0, 1, 1

    fibo(4)
    0, 1, 1, 2

    fibo(5)
    0, 1, 1, 2, 3

    fibo(0)
    ERROR: CANTIDAD DE TÉRMINOS DEBE SER > 0

    fibo(8)
    0 1 1 2 3 5 8 13

14. Haga la función **nombre_dia** que recibe una fecha como un entero de 8 dígitos estructurados así ddmmaaaa: los dos primeros dígitos de la izquierda representan el día, los siguientes dos dígitos son el mes y los cuatro dígitos de la derecha son el año. Retorna el nombre del día respectivo. Si la fecha es incorrecta retorne False. Sugerencia: estudie algoritmo de Zeller.

    Ejemplos del funcionamiento:
    nombre_dia(13032015) → Viernes
    nombre_dia(11132015) → False

15. Haga la función **divisores** que reciba un número entero de 1 en adelante e imprima sus divisores. (for)

    Ejemplos del funcionamiento
    divisores(1) → 1
    divisores(2) → 1, 2
    divisores(3) → 1, 3
    divisores(15) → 1, 3, 5, 15

16. Haga la función **suma_divisores** que reciba un número entero de 1 en adelante y retorne la suma de sus divisores. (while)

    Ejemplos del funcionamiento:
    suma_divisores(1) → 1 (los divisores son: 1)
    suma_divisores(2) → 3 (los divisores son: 1, 2)
    suma_divisores(3) → 4 (los divisores son: 1, 3)
    suma_divisores(15) → 24 (los divisores son: 1, 3, 5, 15)

17. Haga la función **primo** que determine si un número (entero > 0) es primo (retorna True) o no (retorna False). (for). Use el método visto en clase.

18. Haga la función **primo2** que determine si un número (entero > 0) es primo (retorna True) o no (retorna False). Para determinar si un número es primo busque un método diferente al visto en clase. (use la estructura que considere conveniente).

19. Haga la función **suma_primos** que reciba dos enteros (>=1), el inicio y el fin de un rango, y haga la sumatoria de los números primos encontrados en ese rango. (for)

    Ejemplo del funcionamiento:

    suma_primos(10, 15)
    24 (primos en el rango: 11, 13)

    suma_primos(1, 7)
    17 (primos en el rango: 2, 3, 5, 7)

20. Un par de números *m* y *n* es llamado par amigable (o números amistosos), si la suma de todos los divisores de *m* (excluyendo a *m*) es igual al número *n*, y la suma de todos los divisores del número *n* (excluyendo a *n*) es igual a *m* (donde $m \neq n$).
    Ejemplo: los números 220 y 284 son un par amigable. La explicación es la siguiente:
    Los únicos números que dividen de forma exacta a 220 son 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 y 110, y la suma de ellos es: 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284
    Para el otro número los únicos números que dividen de forma exacta a 284 son 1, 2, 4, 71 y 142, y la suma de ellos es: 1 + 2 + 4 + 71 + 142 = 220

    Haga la función **par_amigable** que reciba dos números naturales (>= 2) y retorne el string “PAR AMIGABLE” si cumplen esa propiedad, de lo contrario retorne “No es par amigable”.

    Ejemplos del funcionamiento:

    par_amigable(8, 7) → No es par amigable
    par_amigable(220, 284) → PAR AMIGABLE
    par_amigable(1184, 1210) → PAR AMIGABLE
    par_amigable(2, 3) → No es par amigable

    Evaluación: 5 puntos cada ejercicio.

    Última línea.