TEC – Escuela de Computación – ELEMENTOS DE COMPUTACIÓN – TAREA 2

PROGRAMACIÓN ESTRUCTURADA
PRÁCTICA USANDO ESTRUCTURAS DE SECUENCIA Y CONDICIONAL (if)

OBJETIVO DE LA PRÁCTICA: aplicación intensiva del if.

INSTRUCCIONES:

- Desarrollar en Python 3 en un archivo de nombre: tarea2_su_nombre.py.
- Use el material estudiado a la fecha. Solo se permite usar el tipo de datos numérico (int o float). Puede usar strings para interfaz con usuario. Tampoco se permite el uso de estructuras de repetición, no son necesarias y por otro lado el objetivo es practicar la estructura condicional.
- Los datos de entrada siempre serán numéricos.
- Probar la ejecución de cada función desde el modo comando con diferentes valores. Los programas deben ser generalistas: funcionar con cualquier grupo de datos que cumplan con las especificaciones.
- Enviar la solución al tecDigital EVALUACIONES / TAREAS. Revise que el trabajo enviado sea el correcto y a este destino.
- Requisitos para revisar el programa:
    - Las funciones deben tener: los nombres indicados en cada ejercicio, el número de cada ejercicio y deben estar en la solución ordenados de acuerdo a su número de ejercicio.
    - Buenas prácticas de programación:
        - Las funciones deben tener documentación interna (COMENTARIOS EN EL PROGRAMA FUENTE), al menos: lo que hace, entradas y sus restricciones, salidas. También ponga documentación interna en aquellas partes que ameriten una explicación adicional para un mejor entendimiento del fuente.
        - Nombres significativos de las variables.
- Cada ejercicio vale 10 puntos.
- Fecha de entrega: 29 de marzo, 11 pm.
- Se coordinará día y hora para revisar la evaluación junto con el estudiante quien siendo el autor mostrará el dominio de la solución implementada desde el punto de vista técnico (uso de conceptos de programación y del lenguaje) así como de la funcionalidad (lo que hace la solución). La revisión puede constar de las siguientes actividades:
    - Revisar esta solución particular
    - Revisar conceptos incluidos en la evaluación
    - Aplicar otras actividades con una complejidad igual o menor a la evaluación.

En esta práctica se va a introducir la reutilización de software propio, es decir el desarrollo de funciones utilizando otras funciones que usted ha desarrollado previamente.

SUGERENCIA: Siga la metodología de solución de problemas: entender el problema, diseñar un algoritmo, codificar y probar programa. En la etapa de diseño del algoritmo haga un esquema para determinar el comportamiento o patrón del algoritmo, luego proceda con su desarrollo.
NO SE VAYA DIRECTAMENTE A PROGRAMAR EN LA COMPUTADORA, PRIMERO PLANIFIQUE LA SOLUCIÓN HACIENDO ESE ESQUEMA Y LUEGO HAGA EL PROGRAMA.

Por favor compartan conocimientos e ideas. Para ello pueden trabajar en grupos de 3 personas, pero la presentación y evaluación del trabajo es individual. Es decir que cada miembro del grupo debe enviar el trabajo y su nota será de acuerdo a la revisión particular. Si trabaja en grupos escriba en el fuente su nombre como autor del trabajo y los nombres de los compañeros como coautores.

Ejercicios:

1) Desarrolle la función **par_impar** que reciba un número natural y retorne el valor “par” cuando sea par y el valor “impar” cuando sea impar. Un número es par cuando el residuo de su división entera entre 2 es 0. El valor recibido va a ser un entero pero debe validar que sea un número natural (entero >= 0), sino cumple esta restricción retorne el mensaje “ERROR: NÚMERO DEBE SER NATURAL”.
Ejemplos del funcionamiento:

```
>>> par_impar(200)
par

>>> par_impar(25)
impar

>>> par_impar(-1)
ERROR: NÚMERO DEBE SER NATURAL
```

2) Desarrolle la función **minicalculadora** que reciba tres datos: los dos primeros son números y el tercero un string conteniendo un código de operación matemática (“+”, “-”, “/”, “*”) que se les va a aplicar. Retorne el resultado. Ejemplos del funcionamiento:

```
>>> minicalculadora(5, 9, “*”)
    45
```

Si el código de operación no es válido retorne un mensaje de error:

```
>>> minicalculadora(5, 9, “x”)
    ERROR: CÓDIGO DE OPERACIÓN DEBE SER: +, -, /, *
```

En las divisiones el divisor debe ser diferente de cero, de lo contrario retorne un mensaje de error:

```
>>> minicalculadora(5, 0, “/”)
    ERROR: DIVISOR DEBE SER DIFERENTE DE CERO
```

3) Desarrolle la función **mayor** que reciba tres números y retorne cuál es el mayor. Para obtener este resultado no se permiten usar funciones preconstruidas como max, etc. Ejemplos del funcionamiento:

```
>>> mayor(38, 2000, 1)
2000

>>> mayor(6789, 236, -8888)
6789

>>> mayor(40, 50, 100)
100
```

4) Desarrolle la función **desglose_billetes** que lea una cantidad de colones e imprima su desglose de billetes, es decir, la cantidad mínima de billetes que se deben dar por cada denominación para completar esa cantidad indicada. Hay billetes de 50000, 20000, 10000 y 5000 colones. Los cálculos se hacen por cada denominación de billetes, y para obtener la cantidad mínima de billetes se empieza por el billete más alto hasta llegar al menor. No debe imprimir las denominaciones innecesarias. El dato de entrada es un entero positivo pero debe validar que sea un múltiplo de 5000, de lo contrario imprima el mensaje respectivo. Ejemplos del funcionamiento:

```
>>> desglose_billetes()
Cantidad de colones: 215000
Desglose de billetes:
    4 de 50000          200000
    1 de 10000           10000
    1 de 5000             5000
Cantidad total de billetes: 6

>>> desglose_billetes()
Cantidad de colones: 75200
ERROR: CANTIDAD DEBE SER UN MÚLTIPLO DE 5000
```

5) Desarrolle la función **dígitos_en_comun** que reciba dos números naturales de 3 dígitos y retorne un número con los dígitos que tienen en común esas entradas (intersección de dígitos). El resultado no debe tener dígitos repetidos. Sino tienen dígitos en común retorne -1. Los valores recibidos son números naturales pero valide que sean de 3 dígitos, sino se cumplen estas restricciones de los datos de entradas retorne “Error: las entradas deben ser de 3 dígitos”. Ejemplos del funcionamiento:

```
>>> digitos_en_comun(203, 329)
23

>>> digitos_en_comun(123, 456)
-1

>>> digitos_en_comun(638, 168)
68

>>> digitos_en_comun(233, 322)
23

>>> digitos_en_comun(55, 322)
Error: las entradas deben ser de 3 dígitos
```

6) Desarrolle la función **calificación** para pasar de una escala numérica a alfabética. La función va a recibir una calificación que es un número natural entre 0 y 100, de lo contrario retorna el string “ERROR: NOTA DEBE ESTAR ENTRE 0 Y 100”.

Debe retornar los siguientes valores:
“A – Excelente (Aprobado)”: notas de 90 a 100
“B – Bien (Aprobado)”: notas de 80 a 89
“C – Suficiente (Aprobado)”: notas de 70 a 79
“D – Deficiente (Reprobado)”: notas de 50 a 69
“F - Muy deficiente (Reprobado)”: notas de 0 a 49

Ejemplos del funcionamiento:

```
>>> calificación(95)
A – Excelente (Aprobado)

>>> calificación(65)
D – Deficiente (Reprobado)

>>> calificación(-25)
ERROR: NOTA DEBE ESTAR ENTRE 0 Y 100
```

7) Desarrolle la función **contar_pares_impares** que reciba un número natural de 4 dígitos y retorne dos valores: el primero va a contener la cantidad de todos los dígitos pares y el segundo la cantidad de todos los dígitos impares que aparecen en el número de entrada. Validar que el número de entrada esté en el rango indicado, de lo contrario retornar el mensaje “ERROR: DEBE SER UN NÚMERO NATURAL DE 4 DÍGITOS”. Ejemplos del funcionamiento:

```
>>> contar_pares_impares(1235)
(1, 3)

>>> contar_pares_impares(2426)
(4, 0)

>>> contar_pares_impares(3557)
(0, 4)

>>> contar_pares_impares(219999)
ERROR: DEBE SER UN NÚMERO NATURAL DE 4 DÍGITOS
```

8) Haga la función **pares_impares** que reciba un número natural de 4 dígitos y retorne dos valores: el primero va a contener todos los dígitos pares y el segundo todos los dígitos impares que aparecen en el número de entrada. Cuando no hayan dígitos pares o impares imprimir “no hay” en la parte respectiva del resultado. El orden de los dígitos en los resultados debe ser acorde al orden en el número de entrada. Validar que el número de entrada esté en el rango indicado, de lo contrario retornar el mensaje “ERROR: DEBE SER UN NÚMERO NATURAL DE 4 DÍGITOS”. Ejemplos del funcionamiento:

```
>>> pares_impares(7851)
(8, 751)

>>> pares_impares(1234)
(24, 13)

>>> pares_impares(2426)
(2426, “no hay”)

>>> pares_impares(3557)
(“no hay”, 3557)

>>> pares_impares(219999)
ERROR: DEBE SER UN NÚMERO NATURAL DE 4 DÍGITOS
```

9) Haga la función **orden_descendente** que reciba tres valores numéricos. Debe retornar los tres valores recibidos en orden descendente, es decir de mayor a menor. No se permiten usar funciones preconstruídas de Python para ordenamientos. Ejemplos del funcionamiento:

```
>>> orden_descendente(10, 4, 20)
(20, 10, 4)

>>> orden_descendente(4, 8, -3)
(8, 4, -3)
```

10) Una función boolena es la que retorna solamente 2 valores: True o False. Desarrolle la función booleana **bisiesto** que reciba un año como parámetro (número natural de 4 dígitos >= 1800) y retorne el valor booleano de verdadero (True) si el año es bisiesto o el valor booleano de falso (False) si el año no es bisiesto. Investigue las condiciones para que un año sea bisiesto. En caso de que no se cumpla con la restricción del año retornar el valor False. Ejemplos del funcionamiento:

```
>>> bisiesto(2016)
True

>>> bisiesto(2000)
True

>>> bisiesto(2100)
False

>>> bisiesto(2013)
False

>>> bisiesto(1650)
False
```

11) Desarrolle función booleana **valida_fecha** para determinar si una fecha esta correcta o incorrecta. La función recibe un entero positivo de 8 dígitos en el formato ddmmaaaa: dd son los días, mm son los meses y aaaa es el año. Una fecha se considera correcta si cumple con estas condiciones:

- el año debe ser >= 1800
- el mes debe estar entre 1 y 12
- el día debe ser según el mes y el año. Febrero tiene 29 días cuando el año es bisiesto, si el año no es bisiesto febrero tiene 28 días. Meses con 31 días: enero, marzo, mayo, julio, agosto, octubre y diciembre. Meses con 30 días: abril, junio, setiembre y noviembre

Si la fecha esta correcta retorna True de lo contrario retorna False.

Reutilización de software: una ventaja de las funciones es que podemos reutilizarlas, es decir podemos usar funciones que han sido desarrolladas previamente. En esta función reutilice la función **bisiesto** desarrollada anteriormente.

Ejemplos del funcionamiento:

```
>>> valida_fecha(30032015)
True

>>> valida_fecha(31092006)
False

>>> valida_fecha(29022015)
False
```

12) Desarrolle la función **nombre_dia** que reciba 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. Debe retornar el nombre del día correspondiente a esa fecha. Sugerencia: estudie algoritmo de Zeller. Valide que la fecha sea correcta, de lo contrario retorne False. Para validar la fecha reutilice la función **valida_fecha**. Ejemplos del funcionamiento:

```
>>> nombre_dia(24022016)
Miércoles
```

```
>>> nombre_dia(31092006)
False

>>> nombre_dia(29022015)
False

Última línea.
```