ELEMENTOS DE COMPUTACIÓN – GRUPO 2 – PRÁCTICA 2
PRÁCTICA USANDO ESTRUCTURAS DE SECUENCIA Y CONDICIONAL.

OBJETIVO DE LA PRÁCTICA: **aprender la estructura condicional (if) mediante su aplicación intensiva.**

Indicaciones

- Desarrolle en Python3 en el archivo fuente: **práctica2_su_nombre.py.**
- Con material estudiado a la fecha. No se permite el uso de otros tipos de datos que no se han estudiado: secuencias –strings, listas, tuplas-, diccionarios y conjuntos. Tampoco se permite el uso de estructuras de repetición debido a que el objetivo es la aplicación intensiva del if.
- 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.
- Las funciones deben tener documentación interna (COMENTARIOS EN EL PROGRAMA FUENTE), al menos: lo que hace la función, entradas, salidas.
- Los datos de entrada siempre van a ser numéricos de acuerdo a la especificación del ejercicio (por ejemplo un numérico entero) pero deben cumplir con las restricciones que se indican (por ejemplo que el entero esté entre 0 y 100). En caso de errores en los datos de entrada debe dar los mensajes respectivos para que el usuario se entere de lo que está sucediendo.
- Pruebe la ejecución de cada función en el modo comando.
- Terminarla para hoy.
- Enviarla al tecDigital / DOCUMENTOS / EJEMPLOS DE PROGRAMAS.

Ejercicios:

1) Se requiere la función **nota** para determinar el estado de aprobación de un estudiante en un curso. La función va a recibir una nota que va a ser un número entero y va a retornar una letra según ese valor. La función debe validar que esa nota este en el rango de 0 a 100, de lo contrario retorna "Error: Nota debe estar entre 0 y 100".

   Debe retornar los siguientes valores:
   "A": si la nota es de 100
   "B": si la nota está en el rango de 90 a 99
   "C": si la nota está en el rango de 80 a 89
   "D": si la nota está en el rango de 70 a 79
   "E": si la nota está en el rango de 60 a 69
   "F": si la nota está en el rango de 0 a 59.
   "Error: Nota debe estar entre 0 y 100": si la nota es menor a 0 o mayor a 100.

   Ejemplos del funcionamiento:
   >>> nota(95)
   B
   >>> nota(-25)
   Error: Nota debe estar entre 0 y 100

2) La paridad de un número entero se refiere a su atributo de ser par o impar. La paridad par se refiere a que el número es divisible por 2 (la división entera del número entre 2 da un residuo de 0), y la paridad impar se refiere a que no es divisible por 2.
   Haga la función **paridad** que reciba un entero y retorne "Par" si tiene paridad para o "Impar" si tiene paridad impar.

   Ejemplos del funcionamiento:
   >>> paridad(10)
   Par

   >>> paridad(15)
   Impar

   >>> paridad(-38)
   Par

   En este ejercicio siempre va a venir un número entero, no hay que validar las restricciones de la entrada.

3) Comparativamente dos números pueden tener una misma paridad. Haga la función **paridad_igual** que reciba dos números enteros y retorne:
   - “Tienen paridad par” si ambos tienen paridad par,
   - “Tienen paridad impar” si ambos tienen paridad impar,
   - “Paridad diferente” si tienen paridad diferente.

   Reutilice la función anterior para determinar la paridad de cada número y luego en esta función compare esas paridades.

   Ejemplos del funcionamiento:

```
>>> paridad_igual(2, 4)
Tienen paridad par

>>> paridad_igual(11, 9)
Tienen paridad impar

>>> paridad_igual(15, 20)
Tienen paridad diferente

>>> paridad_igual(11, -21)
Tienen paridad impar

>>> paridad_igual(2, 11)
Tienen paridad diferente
```

   En este ejercicio siempre va a venir un número entero, no hay que validar las restricciones de la entrada.

4) Haga la función **contar_pares_impares** que reciba un número entero positivo entre 0 y 9999 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(123)
(1, 2)

>>> 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
```

5) 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. 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
```

6) Haga la función **minicalculadora** que reciba tres datos, los dos primeros son números y el tercero va a ser un string conteniendo el código de operación matemática (puede ser uno de estos caracteres: “+”, “-”, “/”, “*”) 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 +, -, / o *
```

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

```
>>> minicalculadora(5, 0, “/”)
Error: el divisor debe ser diferente de 0
```

7) Haga la función **bisiesto** que reciba un año como argumento (número entero de 4 dígitos >= 2000) 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. Un año es bisiesto si al dividirlo por 4 su residuo es 0. En caso de que no se cumpla con la restricción del año (tiene un valor < 2000), retornar el valor "Error: año debe ser >= 2000". Ejemplos del funcionamiento:

```
>>> bisiesto(2020)
True

>>> bisiesto(2013)
False

>>> bisiesto(1850)
Error: año debe ser >= 2000
```

8) Haga la función **nombre_mes** que reciba un entero y retorne el nombre del mes respectivo, donde 1 es enero, 2 es febrero, etc. Si el número no corresponde a algún mes retorne el valor booleano False.

Ejemplos del funcionamiento:

```
>>> nombre_mes(2)
febrero

>>> nombre_mes(15)
False
```

9) Haga la función **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 >= 2000
- 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. Un año es bisiesto si al dividirlo entre 4 el residuo es 0. Ejemplos: 2020 es bisiesto, 2008 es bisiesto, 2010 no 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.

Ejemplos del funcionamiento:

```
>>> valida_fecha(30032015)
True

>>> valida_fecha(31092006)
False

>>> valida_fecha(29022015)
False
```

10) Haga la función **fecha** que reciba un entero positivo de 8 dígitos en el formato ddmmaaaa. Debe retornar tres valores según se muestra en los ejemplos del funcionamiento. Reutilice funciones ya desarrolladas: use la función valida_fecha para validar que la fecha este correcta, use la función nombre_mes para obtener el nombre del mes. Si la fecha está incorrecta retorne el valor booleano False.

```
>>> fecha(10012015)
(10, “enero”,  2015)

>>> fecha(25122018)
(25, “diciembre”, 2018)

>>> fecha(29022019)
False
```

11) Haga la función **dígitos_en_comun** que reciba dos números (entre 1 y 999) y retorne un número con los dígitos que tienen en común esas entradas. El valor retornado no debe tener dígitos repetidos. Sino tienen dígitos en común retorne False. Valide restricciones y si hay errores retorne “Error”. Ejemplos del funcionamiento:

```
>>> digitos_en_comun(23, 329)
23
>>> digitos_en_comun(123, 456)
False
>>> digitos_en_comun(638, 68)
68
>>> digitos_en_comun(233, 322)
23
```

12) Haga una función llamada **dobleDeImpar** que reciba un número entero y retorne True si éste es el doble de un número impar, de lo contrario retorne False. Ejemplos del funcionamiento:

```
>>> dobleDeImpar(21)
False
>>> dobleDeImpar(14)
True
>>> dobleDeImpar(8)
False
>>> dobleDeImpar(15)
False
```

NOTA: los números pares o impares son números enteros.

13) Hay que reforestar un bosque con diferentes tipos de árbol. Si la superficie del bosque es menor a 100000 metros la reforestación se indica a continuación:

Superficie del bosque	Tipo de árbol
50 %	pino
30 %	eucalipto
20 %	cedro

Si la superficie es de 100000 metros o más la reforestación es así:

Superficie del bosque	Tipo de árbol
40 %	pino
25 %	eucalipto
35 %	cedro

Haga la función **reforestar** que calcule y retorne el número de pinos, eucaliptos y cedros que se tendrán que sembrar en un bosque. El valor de entrada es un entero (>=0) que indica la cantidad de hectáreas que se van a reforestar. Una hectárea equivale a 10 mil metros cuadrados.
Considere que en 10 metros cuadrados caben 8 pinos, en 15 metros cuadrados caben 15 eucaliptos y en 18 metros cuadrados caben 10 cedros.
Validar restricciones.

14) Haga la función **desglose** que reciba una cantidad de colones e imprima su desglose de billetes. Hay billetes de 50000, 20000, 10000 y 5000 colones. En el desglose se debe dar la mínima cantidad de billetes. Las denominaciones en cero no se imprimen. Validar que la cantidad sea positiva y múltiplo de 5000.

Ejemplo del funcionamiento:

```
>>> desglose(125000)
Desglose de billetes:
2 de 50000          100000
1 de 20000           20000
1 de 5000             5000
                   ----------
Total desglosado    125000
```

Los billetes que no se ocupan no se deben imprimir, este ejemplo no imprimió billetes de 10000 porque no se ocuparon.

15) Haga la función **pago_celular** para calcular y retornar el monto a pagar por servicios de telefonía celular. Recibe los siguientes argumentos para calcular el monto:

- la cantidad de minutos consumidos en llamadas (entero >=0),
- la cantidad de mensajes enviados (entero >=0) y
- el tipo de plan de uso de Internet (1 dígito). Use la siguiente tabla escalonada para calcular el monto a pagar.

a. Tarifa básica de 2750 colones, dando derecho a 60 minutos de consumo. Si usa menos minutos debe pagar esta tarifa mínima.
b. Si consume más de 60 minutos y menos de 121, paga la base más 50 colones adicionales por cada minuto en ese rango.
c. Si consume más de 120, paga la base mas 60 minutos a 50 colones más 35 colones adicionales por cada uno de los minutos adicionales a 120
d. Al costo de las llamadas se agrega el costo de cada mensaje: 3 colones.
e. También se agregar el costo de uso de Internet de la siguiente manera:
    Si el tipo de plan es 0 no hay uso de Internet.
    Si el tipo de plan es 1 se cobra 12000 colones.
    SI el tipo de plan es 2 se cobra 15000 colones.
    Si el tipo de plan es 3 se cobra 25000 colones.
    Otros valores en el tipo de plan no son permitidos.

Adicionalmente debe agregarle al monto a pagar el IVA (13%). Luego agregue una colaboración de 200 colones para el Servicio de la Cruz Roja.

Haga las validaciones de las restricciones y si encuentra algún error retorne el mensaje respectivo.

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

```
>>> orden_ascendente(10, 4, 20)
(4, 10, 20)
>>> orden_ascendente(4, 8, -3)
(-3, 4, 8)
```

17) 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 ordenamientos. Ejemplos del funcionamiento:

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

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

18) Haga 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 la fecha con el nombre del día según ejemplo. Sugerencia: use el algoritmo Congruencia de Zeller. Valide que la fecha sea correcta. Reutilice funciones.

Ejemplo del funcionamiento:

```
>>> nombre_dia(20082019)
Martes 20 de agosto de 2019
```

RECUERDE: Los programas deben ser generalistas: funcionar con cualquier grupo de datos que cumplan con las restricciones especificadas. Probar cada función con diferentes valores.
Última línea.