ITCR
CARRERA: INGENIERÍA EN COMPUTACIÓN
CURSO: TALLER DE PROGRAMACIÓN
GRUPO: 4

TAREA 1: PRÁCTICAS USANDO ESTRUCTURA CONDICIONAL (if)

Indicaciones:

- Para cada ejercicio defina: entradas, salidas, proceso y restricciones. Ponga esto como comentarios al inicio de cada función.
- Asuma que los datos son numéricos pero deben cumplir las restricciones que se indican en cada ejercicio.
- Son algoritmos numéricos: no se permite el uso de funciones como str o de secuencias.
- Use nombres significativos.
- Con material estudiado a la fecha. Objetivo: aplicación intensiva del if.
- Haga solo un archivo fuente **tarea1.py** que contenga todas las funciones las cuales se van a probar desde el modo Shell (o modo comando) con los nombres indicados.
- Enviar el archivo a través del tecDigital.
- Fecha máxima de entrega: 9 de marzo 2015.

Ejercicios:

1) Haga la función **minicalculadora** que reciba tres datos, los dos primeros son números y el tercero va a ser 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 +, -, / 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 cero

2) 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 retornar el valor “ERROR EN EL AÑO”. Ejemplos del funcionamiento:

   >>> bisiesto(2000)
   True
   >>> bisiesto(2013)
   False

>>> bisiesto(1850)
ERROR EN EL AÑO

3) 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 >= 1900
- 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: 2000 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

4) Haga la función **fecha** que reciba un entero positivo de 8 dígitos en el formato ddmmaaaa. Debe retornar la fecha según se muestra en los ejemplos del funcionamiento. Validar que la fecha este correcta, de lo contrario retorne False.

>>> fecha(10012015)
10 de enero 2015

>>> fecha(25122010)
25 de diciembre 2010

>>> fecha(29022015)
False

5) Haga la función **elimina_digito** que reciba dos argumentos, un dígito (entre 0 y 9) y un número entero (positivo de cuatro dígitos exactos), y elimine de este número los dígitos que sean iguales al dígito del primer argumento. Validar restricciones: cuando no se cumpla con la primera restricción retornar el valor “ERROR EN DIGITO”, en caso de no cumplir con la segunda restricción retornar el valor “ERROR EN NÚMERO”, y en

caso de no cumplir ambas restricciones retornar “ERROR EN DÍGITO Y EN NÚMERO”. Ejemplos del funcionamiento:

```
>>> elimina_digito(2, 1234)
134
>>> elimina_digito(8, 8108)
10
>>> elimina_digito(4, 1256)
1256
```

6) Haga la función **convertir_segundos** que tome dos argumentos como entrada: un entero positivo en segundos como primer argumento y un string indicando el tipo al que se quiere transformar: “segundos”, “minutos”, “ horas”, “días”, “semanas”, “meses”.

   Valide las restricciones en los dos valores de entrada y si hay error envíe el mensaje “ERROR”. Ejemplos del funcionamiento:

```
>>> convertir_segundos(930, “minutos”)
15.5
>>> convertir_segundos(45, “segundos”)
45
>>> convertir_segundos(120000, “dias”)
1.388...
```

7) 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
```

8) 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:

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

```
>>> dobleDeImpar(15)
False
```

9) 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.

10) Manejo de una cuenta bancaria de ahorros con la función **cuenta_bancaria**. La función va a tener 3 argumentos: el saldo actual de la cuenta (>=0), el tipo de operación que el usuario quiere hacer (el tipo puede ser el valor 1 que significa una operación de depósito, o un 2 que significa una operación de retiro), la cantidad de la operación (>0). Debe devolver el nuevo saldo. Tanto los depósitos como los retiros deben ser múltiplos de 5000. Hay que validar los datos (restricciones) según se indica y revisar en los retiros que estos se puedan hacer en caso de que el saldo lo permita, de lo contrario retornar el valor booleano False.

11) 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.
Por ejemplo, si se desea conocer el desglose de 125000 colones (dato de entrada), el programa debe mostrar en la pantalla los siguientes resultados:

```
Desglose de billetes:
2 de 50000      100000
1 de 20000       20000
1 de 5000         5000
```

Total desglosado 125000

No imprimió billetes de 10000 porque no se ocuparon.

Validar que la cantidad sea positiva y múltiplo de 5000.

12) Haga la función **cajero** que reciba 3 argumentos: el saldo actual de una cuenta (>=0), el tipo de operación que el usuario quiere hacer (el tipo puede ser el valor 1 que significa una operación de depósito, o un 2 que significa una operación de retiro), la cantidad de la operación (>0 y múltiplo de 5000). Validar los datos. Debe imprimir:

- el nuevo saldo
- si es un retiro también el desglose de la moneda.

13) Haga la función **pago_celular** para calcular y retornar el monto a pagar por servicios de telefonía celular. Recibe estos argumentos para calcular este 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:

- Tarifa básica de 2750 colones, dando derecho a 60 minutos de consumo. Si usa menos minutos debe pagar esta tarifa mínima.
- Si consume mas de 60 minutos y menos de 121, paga la base mas 50 colones adicionales por cada minuto en ese rango.
- Si consume mas de 120, paga la base mas 60 minutos a 50 colones mas 35 colones adicionales por cada uno de los minutos adicionales a 120
- Al costo de las llamadas se agrega el costo de cada mensaje: 3 colones.
- 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 un impuesto de ventas del 13% y 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.

14) Haga la función **pares_impares** que reciba un número entero positivo entre 0 y 9999 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”. Validar que el número de entrada esté en el rango indicado, de

lo contrario retornar el mensaje “Error: debe estar en el rango de 0 a 9999”. Ejemplos del funcionamiento:

```
>>> pares_impares(123)
(2, 13)

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

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

>>> pares_impares(219999)
Error: debe estar en el rango de 0 a 9999
```

15) 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)
```

16) 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)
```

Cada ejercicio vale 6.25
Última línea.