

## RETO 41

### En `utilNotas.py`:

1. Agregar una función `calcularTotal`, que servirá para calcular la nota total.

La función recibirá los siguientes parámetros:

**n1**: el valor de la nota 1 (float)

**n2**: el valor de la nota 2 (float)

**n3**: el valor de la nota 3 (float)

**faltas**: el número de inasistencias a clases

Calcula el valor total, en base a la siguiente lógica:

El número de faltas determinará el valor de una cuarta nota(n4), así:

Si tiene 0 faltas, n4 es 10

Si tiene [1,4[ n4 es 9 (como se indica, el rango es 1 incluido, 4 excluido)

Si tiene 4 o 5 faltas n4 es 8

Si tiene más de 5 faltas n4 es 7

Una vez que obtiene el valor de n4, calcula el promedio de las 4 notas y lo retorna, redondeado a 2 decimales.

2. Ejecutar el archivo `TestCalcularTotal.py`, este archivo no puede ser modificado.

Resultados esperados:

```
FALTAS: 1 ---> Nota 4: 7.25
FALTAS: 2 ---> Nota 4: 7.25
FALTAS: 3 ---> Nota 4: 7.25
FALTAS: 4 ---> Nota 4: 7.0
FALTAS: 5 ---> Nota 4: 7.0
FALTAS: 6 ---> Nota 4: 6.75
FALTAS: 7 ---> Nota 4: 6.75
```





Escuela de Programación

3. En **notasCompletoV3.py**:

- 3.1 Crear una función **fnObtenerEntero**, con una estructura igual a **fnObtenerFloat**, pero permite obtener un valor entero dentro de un rango, recibe los parámetros **txtInfo**, **lblError**, **desde** y **hasta**. El parámetro **hasta** es opcional, su valor por defecto es **None**. Realizar las mismas validaciones que se hace en **fnObtenerFloat**, pero debe convertir a **int** en lugar de convertir a **float**.  
Además considerando que el parámetro **hasta** es opcional, validar que si llega un valor, haga la validación de rango que sea mayor o igual a **desde** y menor o igual que **hasta**. Pero si el usuario no manda ningún valor, solamente validar que sea mayor o igual a **desde**.
- 3.2 Desde **fnGuardar**, invocar a **fnObtenerEntero**, para recuperar el valor de las inasistencias de la caja de texto **txtInasistencias** con los valores correspondientes, pedir que valide que el número sea mayor o igual a 0.
- 3.4 Antes de ejecutar el guardado en la lista, agregar la condición para que si falló la validación, no guarde.
- 3.5 Calcular el valor del total, invocando a la función **calcularTotal**, mostrar en el label **lblTotal**. Agregar el valor de las inasistencias y de total, luego de la **nota3** en el String que tiene toda la info del estudiante, por ejemplo quedaría así:  

```
1614616123#Maritza#Ramos#8.3#2.6#7.2#0#7.5
```
- 3.6 Ejecutar y probar que cuando la nota sea incorrecta, muestre el mensaje y no deje guardar y que cuando todo esté ok, se agregue la info al arreglo.
- 3.8 Modificar la función limpiar, para que también limpie **lblTotal** y **txtInasistencias**



Email  
[info@krakedev.com](mailto:info@krakedev.com)



Phone  
+593 98 412 767



Web  
[www.krakedev.com](http://www.krakedev.com)

#### CAMBIOS EN BUSCAR

1. En los datos iniciales de la lista, agregar al final de cada String las inasistencias y la nota total, así por ejemplo

```
1614616123#Maritza#Ramos#8.3#2.6#7.2#0#7.5
```

2. Completar el código necesario en la búsqueda para que cuando se busque el estudiante, también muestre en pantalla las inasistencias y el total recuperado.
3. Probar con la búsqueda de uno de los datos iniciales de la lista y con un dato ingresado.

#### En `utilNotas.py`:

Agregar una función `calcularPromedioCurso`, que recibe como parámetro la lista con la info de los estudiantes y realiza la siguiente lógica:

Crear una variable llamada `totalSuma` con valor inicial 0.

Itera toda la lista y de cada cadena extrae la nota total, usando `Split` y la posición correcta de esta nota.

Convierte la nota a float

En cada iteración va sumando el valor de la nota a la variable `totalSuma`.

Al terminar la iteración, divide el valor de `totalSuma`, para el número de elementos de la lista y retorna este resultado, redondeado a 4 decimales.

Ejecutar `TestPromedioCurso.py`, el test no puede sufrir ningún cambio.

Resultado Esperado:

```
Promedio1: 10.0  
Promedio2: 8.75  
Promedio2: 8.25
```

