## Anexo 3. Unidad 4. Ejemplo de documentación de programas

**Descripción del problema:**

Escribir un programa que permita gestionar los datos de clientes de una empresa. Los clientes se guardarán en un diccionario en el que la clave de cada cliente será su NIF.

El valor asociado a cada cliente será otro diccionario o una lista, con los datos del cliente (apellidos y nombre, dirección, teléfono, correo electrónico, cliente habitual, fecha de la operación), donde “cliente habitual” tendrá el valor True si se trata de un cliente no esporádico. Se valorará que se realicen dos versiones: con campo de valor diccionario y lista.

El carácter de cliente habitual lo introducirá el empleado de la empresa que utilizar el programa

(si/no). La fecha se introducirá entre comillas en la forma: ‘12/01/2020’.

El programa debe preguntar al usuario por una opción del siguiente menú:

(1) Añadir cliente.

(2) Eliminar cliente.

(3) Mostrar cliente.

(4) Listar todos los clientes.

(5) Listar clientes habituales.

(6) Salir.

En función de la opción elegida el programa tendrá que hacer lo siguiente:

1. Preguntar los datos del cliente, crear un diccionario (o lista) con los datos y añadirlo al contenido del campo valor del cliente.
2. Preguntar por el NIF del cliente y eliminar sus datos de la variable que contiene los datos.
3. Preguntar por el NIF del cliente y mostrar todos sus datos.
4. Mostrar lista de todos los clientes de la base datos con su NIF y nombre.
5. Mostrar la lista de clientes habituales de la base de datos con su NIF y nombre.
6. Terminar la ejecución del programa.

En su resolución se deberá programar el menú de forma que cada opción se realice implementando una función. Caso de introducir una opción fuera del rango de 1 a 6, no debe terminar el programa, sino que debe advertirlo para que se vuelva a seleccionar otro punto del menú. El programa sólo terminará cuando se seleccione la opción 6. Se comprobará que el NIF contiene 9 caracteres, los 8 primeros numéricos. A su vez, el teléfono debe contener 9 dígitos. También se puede comprobar la fecha. Estas comprobaciones no se deben gestionar mediante excepciones.

**DESCRIPCIÓN DE LA SOLUCIÓN ADOPTADA:**

**Estructura de datos**

La estructura de datos adoptada para este caso es la de un diccionario cuyas claves se corresponden al valor NIF del cliente y cuyo valor será un diccionario con los datos del cliente correspondientes a dicho NIF, el repositorio de datos se ve parecido al siguiente:

```
clients = {
      '82037269A': {
            'nombre': 'Mauricio',
            'apellido': 'Contreras',
            'direccion': 'Barcelona',
            'telefono': '999666333',
            'correo': 'mauricio@email.com',
            'habitual': True,
            'fecha': '14/11/2020'
      }
}
```

Se entrega una versión del programa donde se implementa una estructura de tipo `lista` para almacenar los datos del usuario.

La estructura de esta lista es la siguiente:

```
clients = [
      {
            'nif': '82037269A',
            'nombre': 'Mauricio',
            'apellido': 'Contreras',
            'direccion': 'Barcelona',
            'telefono': '999666333',
            'correo': 'mauricio@email.com',
            'habitual': True,
            'fecha': '14/11/2020',
      }
]
```

**Ciclo del programa**

Para ejecutar el programa se debe abrir una consola interactiva del entorno operativo, que tenga acceso al ejecutable del intérprete de Python. En dicha consola se debe escribir lo siguiente:

```
python ejercicio_ feedback _ diccion ario.py
```

Lo cual ejecutará la versión diccionario del programa, si se desea ejecutar la versión de lista, se debe escribir lo siguiente en la consola:

```
python ejercicio_ feedback _ lista.py
```

Ambos programas han sido probados en la versión Python 3.6.9.

Al iniciar el programa se llama a la función `menu()`, la cual ejecuta un bucle infinito para mostrar las opciones al usuario.

El bucle se detiene sólo cuando el usuario selecciona la opción de salida del programa (o cuando el usuario presione ctrl + c ).

Cada opción del menú ejecuta una función, la cual devuelve un valor booleano para indicar si se debe seguir ejecutando el bucle, todas las funciones, (excepto la función de salida del programa), devuelven un valor de **True**.

No se hace casting a tipo entero a la opción elegida por el usuario, ya que no se considera realmente necesario. Se opta por crear un diccionario que crea un mapeo entre las distintas opciones y la función asociada. De esta forma se reduce la cantidad de código necesario en una estructura `if elif … else` y se opta por usar el método `get` del tipo Diccionario.

Cada función del menú hace uso de funciones utilitarias que evitan la repetición de código y hacen el mismo más limpio. Las funciones utilitarias forman parte de las validaciones necesarias para evitar errores en la introducción de los datos.

En cada llamado a `input()` se hace uso del método `strip()` del tipo String, esto con la finalidad de eliminar posibles espacios sobrantes antes o después del valor introducido por el usuario. Muchos usuarios que han tomado algún curso de mecanografía, tienden a escribir siempre un espacio en blanco al finalizar cada palabra, es por eso que usar `strip()` siempre es una buena idea para sanitizar la entrada de datos.

Las validaciones han sido pensadas usando el estilo first fail donde se verifica primero siempre una posible falla, se realizan en forma de cascada, asegurando de esta manera que el test se puede realizar.

Por ejemplo, para validar la fecha introducida primero se verifica que el tamaño de la cadena sea de 10 dígitos, esto nos permitirá acceder luego a las posiciones 2 y 5 que corresponden a los separadores de dia, mes y año. Luego si ambas validaciones se superan se procede a extraer la parte correspondiente a los valores para dia, mes y año para que sean evaluados individualmente.

**Funciones utilitarias**

Se decide escribir una serie de funciones utilitarias que nos permitan reutilizar el código de manera eficiente. Estas funciones tienen el propósito de realizar las validaciones necesarias para que no se introduzcan datos erróneos al listado de clientes.

```
is_ on ly_ digits(valu e: strin g) → boolean
```

Esta función nos permite determinar si la cadena `value` contiene sólo dígitos numéricos. Devuelve True en caso de que los caracteres de la cadena sean todos valores numéricos y False en caso contrario.

Se recorre la cadena carácter por carácter y se verifica si su valor decimal se corresponde con el valor decimal ASCII para los caracteres numéricos, esto se realiza mediante la siguiente comprobación:

```
if ord(valu e[i]) not in range(4 8 ,5 8 ):
```

Los valores decimales para caracteres numéricos van desde el 48 al 57 en la tabla ASCII. Se convierte el carácter a su valor decimal usando la función `ord()`.

Uso:

1.- Validación de teléfono

3.- Validación de día, mes y año

4.- Validación de la parte correspondiente al host en un correo electrónico

5.- Validación de los primeros 8 caracteres del número NIF

```
is_ valid_ date(date: strin g) → boolean
```

Esta función nos permite validar si el valor de fecha representado por la cadena `date` tiene el formato requerido: **dd/mm/yyyy**

Para verificar el tamaño de la cadena se usa el siguiente condicional:

```
len (date) != 1 0
```

Para determinar si los caracteres de separación se corresponden con el carácter permitido se usan las siguiente condiciones:

```
date[2 ] != '/ '
```

```
date[5 ] != '/ '
```

Las validaciones incluyen:

1.- Tamaño de la cadena: 10 dígitos.

2.- Carácter de separación: /.

3.- Mes debe ser un valor entre 1 y 12 (se realiza cast de string a entero).

4.- Día debe ser un valor entre 1 y 28 para el mes de febrero y entre 1 y 29 para febrero en años bisiestos.

5.- Día debe ser un valor entre 1 y 30 para los meses 04, 06, 09 y 11.

6.- Día debe ser un valor entre 1 y 31 para el resto de los meses.

7.- Año no puede ser inferior a 1900.

```
is_ valid_ n am e(n am e: strin g, tipo= ” n om bre” : strin g) → boolean
```

Esta función valida que un nombre o apellido tengan al menos 2 caracteres. Se establece una lista de caracteres admitidos de acuerdo al alfabeto castellano.

Recibe 2 parámetros:

name: valor de cadena a analizar

tipo: tipo de valor que estamos analizando, para mostrar un mensaje de acuerdo al tipo, su valor por defecto es “nombre”.

Se realizan validaciones creando una lista de caracteres permitidos (alfabeto castellano) para verificar que cada carácter de la cadena se encuentra en dicha lista y se analiza el tamaño de la cadena recibida (al menos 2 caracteres).

Devuelve True si el valor para nombre es válido, y False en caso contrario

```
is_ valid_ n if(n if: string) → boolean
```

Esta función realiza la validación de un número de NIF. Se toma en cuenta sólo aspectos de formato, no de validez legal de un número NIF.

Recibe un argumento de tipo cadena y se verifican los siguientes aspectos:

1.- Tamaño de la cadena: 9 caracteres

2.- Los primero 8 caracteres deben ser numéricos

3.- El último carácter debe ser una letra mayúscula del alfabeto Latino (código ASCII decimal desde 65 hasta 91)

4.- También verifica si el nif introducido existe en la lista de clientes del programa. Devuelve Tru e si el nif es válido o False en caso contrario

```
is_ valid_ phon e(phone: strin g) → boolean
```

Función que verifica la validez de un número telefónico, basado en los siguientes aspectos:

1.- Tamaño de la cadena: 9 caracteres

2.- Valores admitidos: sólo caracteres numéricos (se usa la función is_only_digits)

3.- Los números de teléfono admitidos deben empezar por 9 o por 6

Devuelve un valor True si el teléfono es válido y False en caso contrario

```
is_ valid_ em ail(em ail: strin g) → boolean
```

Esta función valida el formato de un correo electrónico, se basa en parte del estándar definido en el RFC5322: https://tools.ietf.org/html/rfc5322

Los aspectos de validación se ajustan, en lo posible, a la sección 3.4.1 de dicho documento.

```
prin t_ clien t(n if: string, clien t: diction ary) → void
```

Esta función imprime una fila que contiene los datos de un cliente asociado al **nif** introducido. Recibe como parámetros el número de NIF y el diccionario que contiene los datos.

Usando esta función podemos recorrer el diccionario de clientes e imprimir uno a uno todos los

clientes.

Usando la llamada siguiente:

```
print_client("82037269A",clientes["82037269A"])
```

La salida sería parecida a la siguiente:

**82037269A | Mauricio | Contreras | Barcelona | 999666333 | SI | 14/11/2020 | mauricio@email.com**

```
prin t_ header(title: strin g, pad_ ch ar: strin g) → void
```

Esta función permite imprimir el encabezado de la tabla de clientes, asignando un título a la misma. Recibe 2 parámetros de tipo cadena:

title: El título de la tabla

pad_char: Un carácter que se usará para decorar el título de la tabla.

La función calcula el centrado del título de forma dinámica, para sin importar el tamaño de la cadena de título, la misma quede lo más centrada posible en la tabla.

Para calcular el centrado del título se usa un valor fijo de 114 caracteres. Se usa la siguiente lógica para calcular la cantidad de caracteres que se van a insertar antes y después del título:

```
righ t_ pad = 5 7 + (len (title) / / 2 )
```

Lo cual calcula el padding derecho mediante la suma de la división entera del tamaño del título entre 2 con 57 caracteres (57 es la mitad de 114).

Este valor se usará en el método rjust() del tipo String, que añade un padding por la derecha usando un carácter pasado como segundo argumento.

Para el padding izquierdo se usa el método ljust() el cual recibirá el tamaño total de la linea (114)

y el carácter usado para padding.

`title.rju st(righ t_ pad, pad_ ch ar).lju st(1 1 4 , pad_ char)`

De esta forma se logra centrar (aproximadamente) el título en una linea de 114 caracteres.

También imprime las cabeceras de cada columna de la tabla. Esta función está pensada para llamarse 1 vez al momento de imprimir el menú, la lista de todos los clientes o la lista de clientes habituales y cuando se añade o se elimina un cliente.

Usando la siguiente llamada:

`print_header(" Listado de clientes ", "-")`

La salida sería la siguiente:

---------- Listado de clientes ----------
--- NIF | Nombre | Apellido | Dirección | Teléfono | Habitual | Fecha | Correo

---

-

`in it_ clien t() → dictionary`

Esta función permite iniciar un diccionario con los campos del cliente vacíos (establecidos a None ) Devuelve dicho diccionario para que sea utilizado por la función encargada de crear un nuevo

cliente. El diccionario devuelto tiene la siguiente estructura:

{

'nombre': None,

'apellido': None,

'telefono': None,

'direccion': None,

'correo': None,

'habitual': None,

'fecha': None

}

`ex ist_ clien t(n if: string) → boolean (Sólo en la versión de estructura de datos tipo lista)`

Esta función se encarga de buscar un cliente en la lista de clientes. Sólo se implementa en la versión del programa que usa una estructura de repositorio de tipo lista.

Devuelve un valor booleano correspondiente a la existencia o no del cliente cuyo NIF ha sido pasado como parámetro.

La búsqueda se realiza recorriendo la lista de clientes y comparando el valor NIF con el campo nif de cada documento de la lista.

**Funciones del menú**

Las funciones del menú son las que se ejecutan al momento que un usuario selecciona una opción del mismo. En total se definen 7 funciones, 6 de las cuales son parte de una opción del menú y la séptima es una función que se ejecuta cuando el usuario no ingresa una opción válida.

Las funciones son:

```
add_ clien t() → boolean
```

```
delete_ clien t() → boolean sh ow _ clien t() → boolean
list_ of_ clien ts() → boolean list_ of_ frecu en t_
clients() → boolean ex it_ program () → boolean
default() → boolean
```

Se crea un diccionario que relaciona las diferentes opciones del menú con una función. De esta forma si la opción seleccionada se encuentra dentro de las opciones del diccionario se ejecutará la función adecuada.

La estructura del diccionario de funciones creadas es la siguiente:

**cases = {**

**“1”: add_client, “2”: delete_client, “3”: show_client, “4”: list_of_clients,**

**“5”: list_of_frecuent_clients,**

**“6”: exit_program**

**}**

La forma de escoger la función es usando el método get del tipo diccionario. Se usa de esta forma:

```
ex ec_ selected = cases.get(op, defau lt)
```

Se usa la función default como valor por defecto en caso que el método get no encuentre la opción introducida por el usuario.

Así exec_selected será una función que podremos llamar usando:

```
ex ec_ selected()
```

La misma se corresponderá con la función adecuada de acuerdo a la selección del usuario.

Se muestran los mensajes adecuados en caso de que el usuario elija una opción no válida según la lista de opciones mostradas.

Todo el código está comentado, usando comentarios sencillos y docstring.

Se adjuntan las 2 versiones del programa.