APARTADO 4

# ¿Por qué Python?

La razón es la sencillez y capacidad de este lenguaje para el análisis y tratamiento de los datos gracias a librerías como Pandas o Numpy. Base gran parte de la metodología de análisis usada en este proyecto en el libro: LIBRO.

Las distintas posibilidades que se barajaron antes de empezar fueron Java, Android, R y Python. Al final me decanté por Python por diferentes motivos, el primero en ser descartado fue Java, tras un estudio inicial sobre lo que quería hacer y el como hacerlo, me di cuenta que necesitaba un lenguaje potente que me dejase tratar los datos con claridad para su análisis, y después de leer varios artículos como: REF\_ARTICULOS, me percaté de que Java no era la mejor opción, estos artículos siempre te orientaban hacia Python y R. Entonces, ¿Por qué Android?, por la sencilla razón de que este trabajo está desarrollado para unir el mundo de la dietoterapia, y las ciencias de la Salud con la informática para hacer que el usuario tenga un fácil aprendizaje de dicha metodología. A mi parecer la forma más clara y rápida de llevar al usuario dicha tecnología es a través de su SmartPhone, pero se acabó descartando debido a la falta de conocimientos sobre sistemas Android. A estas alturas ya solo me quedaba elegir entre Python y R, tras indagar superficialmente sobre ambos lenguajes para el análisis y el tratamiento de datos, se llego a la conclusión que ambos lenguajes tienen una forma de trabajar muy similares, y entonces la decisión fue clara, debido a que he trabajado en numerosas ocasiones con Python y la sintaxis de R para mi era desconocida, al final, me decante por Python.

# Metodología

## Introducción

En este subapartado explicaremos el cómo y porque se ha tratado los datos en este TFG además de los diferentes cálculos internos que se realizan para el sistema de recomendaciones, cálculos, etc.

## Excel y Pandas

Se usará la herramienta de Microsoft, Excel, para trabajar como una base de datos, y poder tratarlos en forma de DataFrame, esta particularidad, nos la da la librería de Pandas. Hay un Excel para la base de datos genérica, y otro para la base de históricos. Se leen los datos automáticamente en cuanto el usuario entra en la aplicación, pero solo se guarda si el usuario así lo desea.

## DataFrame

Se usan los DataFrame, para llevar un registro de todos los datos que el programa necesita, se tiene en cuenta tanto las bases de datos de los alimentos, usuarios y comidas como lo que el usuario lleva en el día.

Los alimentos se tratan en el momento en el que se registra el usuario, de manera que, se separan las comidas de la lista principal, creando 5 listas (Desayuno, Merienda, Comida, Almuerzo y Cena), se tratan por separado y se filtran por el umbral, el umbral es un medidor de calidad de la comida, para qué solo te muestre lo más sano en un principio, y se van actualizando sensibles a los cambios de la aplicación. Si se refresca la página aumenta el umbral para que sea mas amplia las posibilidades de comida, aunque esto signifique una peor alimentación. Con cada elección se actualizan el resto de las comidas, para que se ajuste al máximo la dieta a sus necesidades, se ordena por la formula que se hablará mas adelante, se crea la variable “dif” que sale de esta fórmula, y este es el eje principal de las recomendaciones, cada vez que se elige una comida, la comida se almacena para poder tratarla mas tarde, en cuanto se selecciona los botones se desactivan, y se cambia el botón seleccionar a editar, para que se pueda editar en cualquier momento. Cada vez que se elige una comida se reparte las calorías sobrantes/que faltan, y se recalculan las posibilidades.

# Estructura del programa

## Introducción

En un principio se pensó en hace una estructura MVC clásica pero conforme el proyecto fue creciendo, y prácticamente de manera imperceptible se convirtió en una versión un tanto modificada de este modelo.

La estructura se basa en cuatro módulos principales, además de las librerías pertinentes, obviamente.

1. Main – Tronco del programa
2. Vistas – Relacionado con la parte visual
3. CalculosDieta – Todo calculo usado en el programa
4. AdminBase – Todo proceso relacionado con la base de datos

En los apartados siguientes se hablará de manera extensa sobre las librerías utilizadas y los módulos creados para el desarrollo de esta aplicación, por el momento, este apartado será explicando en que deriva y se basa dicha estructura, conforme el MVC clásico y cuales son las consecuencias de esto.

## Cálculos, Administración y Vista

Podríamos llamar propiamente a este modelo el CAV, y se basa en separar toda función que se vaya a crear en el programa en estos tres pilares.

Todo surgió de la necesidad de separar la administración de la base de datos con cualquier otro tipo de función en el programa, esto sin darme cuenta creaba una cuarta pata en el trípode que supone el MVC. Este fue uno de los motivos, pero el motivo principal fue menos precipitado, se iba creando un módulo cada vez que la lógica de una función necesitará un hueco donde encajar y ninguno de los módulos creados lo sostenía. Debido a mi forma de trabajar, me sentí cómodo de dividirlo así, creando inicialmente una estructura básica y modulada, pero no en exceso. Por un lado todo elemento relacionado con la base de datos, por otro lado todo cálculo matemático realizado en el programa y por ultimo toda función que intervenga en la interacción usuario-aplicación.

Esto tuvo en su momento sus inconvenientes, surgían funciones que encajaban en diferentes módulos debido a que igual era puramente visual, pero interactuaba con la base de datos, o llevaba a cabo unos cálculos para dibujártelos visualmente, por ello se llevo a cabo una serie de reglas sencillas:

* Si su finalidad es puramente visual – Vista
* Si realiza cálculos u operaciones matemáticas, indistintamente de si sea sobre la dieta -CalculosDieta
* Si interacciona y guarda datos en la base de datos -AdminBase
* Si mezcla alguna de estas funcionalidad, por ejemplo, guardar en la base de datos y mostrar por pantalla algo, se tendrá en cuenta cual es su principal finalidad, por ejemplo, se guardan estilos, que se cargaran automáticamente para el diseño del programa, obviamente su finalidad es puramente visual, y guardar en la base de datos una transacción necesaria.
* Cualquier Objeto, ventana, tabulación, etc. Que siempre vaya estar ahí al Main, pero sus modificaciones se repartirán entre los diferentes modulos.

Estas reglas, a día de hoy, han cumplido con todas las funciones que se han creado en el programa dando el resultado previsto.

# Librerías

## Pandas

La librería desarrollada por Wes McKinney, Pandas, es una librería usada para el tratamiento de los datos como estructuras. No fue la primera opción para este proyecto, debido a que este proyecto se enfoca en el análisis de datos, y no en el tratamiento de bases de datos, se decidió crea una base de datos local en un archivo Excel, la primera opción fue la librería “openpyxl”, la cual, es una librería de código abierto, que permite la carga y manejo de datos XLS, el problema vino que te devolvía objetos desestructurados difíciles de tratar, y complicaba él objetivo principal del proyecto, acto seguido basándome en él libro: LIBRO, decidí probar con Pandas, esta librería me permitía leer los datos de los archivos .XLS, en forma de dataframes. Esto simplificaba el análisis de los datos, permitiendo tratarles tanto como dataFrame como en forma de numpy.array, dicha librería será analizada a continuación.

## Numpy

Como ya he nombrado en varias ocasiones en este memoria, el libro: LIBROO, se basa principalmente en esta librería, como añadido, es la librería sobre la que he trabajado a lo largo de la carrera en cuanto a análisis de datos en Python, y la mas recomendada por los usuarios en la web, sobre esta librería se sustenta principalmente este proyecto, siendo la encargada, del tratamiento, procesamiento y calculo, de todos los datos que internamente realiza el programa.

## Interfaz gráfica

La librería utilizada para realizar y diseñar la interfaz gráfica fue Tkinter, tras buscar en diferentes páginas que encontraremos en la bibliografía, me decante por Tkinter debido a la falta de conocimientos en interfaces gráficas con Python, me decante por la mas fácil de aprender. Entre las opciones que baraje se encontraban: Tkinter, WxPython, PyQt y PyGTK.

Tkinter traía una serie de ventajas, entre ellas que viene preinstalada con Python, era fácil de aprender, y hay una documentación amplia y extensa. Pero también tiene sus desventajas, incluye pocos elementos gráficos, tiene un limitado control y la navegabilidad mas sencilla se hace complicada, y sobre todo lo que mas noto es su lentitud, cuanto mas elementos añadas, mas lento va la interfaz, no tiene alguna especie de cache o memoria que guarde lo que has “dibujado”, sino que los dibuja cada vez que salen en pantalla, lo hace sobre cada botón etiqueta, etc.

La segunda opción que se me paso por la cabeza, fue WxPython, tenia grandes ventajas, como la rapidez, la flexibilidad que este ofrecía y sobre todo su mayor ventaja son todas las opciones que tiene para crear una interfaz gráfica compleja y “profesional”, tras barajar, estas opciones y compararlas con las de Tkinter, me di cuenta que todo lo que te ofrecía WxPython, era innecesario para la interfaz tan simplista que necesitaba para mi proyecto, y el aprendizaje era más complejo, además de ser mas complicado encontrar información o documentación sobre esta librería. El mayor problema de esta librería era que tiene una comunidad muy activa, la cual, esta constantemente insertando cambios, esto era un problema para proyectos largos, pues creaba problemas de compatibilidad, pero para este proyecto, no era algo que me resultará un inconveniente.

El resto de librerías fueron descartadas, al poco de buscar información sobre ellas debido a que daban las mismas ventajas o similares que la WxPython, pero tenían mas inconvenientes, al menos para el tema que aborda este proyecto.

### Diseño

Se decidió realizar un diseño simple, sin complicaciones, ni laberintos internos de pura navegabilidad que haga del manual de usuario un mapa para guiarse a través del programa. Esta decisión trivial se llevo a cabo debido al principal objetivo del proyecto, el aprendizaje de la dietoterapia, esta aplicación esta, básicamente, orientada a todos los públicos, por lo que si se construye una aplicación tediosa, solo complicaría el uso y aprendizaje del usuario.

Se basa en un menú principal ramificado en tres vertientes: Usuario, Dieta y Registro ( en el programa aparecen con otros nombres, pues aquí se explica la lógica del diseño). En la vertiente usuario, esta la información del usuario y la posibilidad de cambiar dichos datos para un avance del programa. En la rama de la dieta, se encuentra el tronco principal de la aplicación, se muestra las recomendaciones alimenticias, a parte de darte libertad a la hora de escoger y decidir que deseas comer en el día de hoy. Por ultimo estaría el registro o historial, es decir todo aquello que necesites para llevar un registro de tu progreso y concienciar de esta manera al usuario.

Respecto a los colores, se decido un tema básico que no agote la vista del usuario, ni tenga múltiples colores deslumbrantes, da la posibilidad al usuario de elegir entre una serie de estilos predefinidos para que escoja el que mejor se ajuste a sus gustos y necesidades.

# Módulos

## CalculosDieta

### Introducción

Es el modulo principal sobre el que se sustenta el programa, hace todos los cálculos de recomendación, kilocalorías, repartos, etc.

En el manual del programador se extenderá mas sobre cada función, a continuación, se hará un breve comentario sobre cada función en base a la metodología.

### CalculoTMB, rapartoDeKcal y distribuciónDeMacronutrientes

Estas tres funciones trabajan de manera paralela, el **calculoTMB**, lo que hace es calcular el gasto calórico basal, de aquí se saca el objetivo diario, una vez calculada la cantidad de Kilocalorías diarias, se reparten las kcal (**repartoDeKcal** )según sea desayuno, almuerzo, merienda o cena, se reparten en base a unos porcentajes calculados en base a distintas fuentes de información encontrados a lo largo del estudio sobre el que se sustenta este proyecto, no son porcentajes exactos, Se distribuyen de la siguiente manera:

* Desayuno: 24,75% -- Alta carga calórica, es una de las tres comidas principales y la carga en la cual el cuerpo absorbe mejor los nutrientes.
* Almuerzo: 13,5% --Baja carga calórica, pero mayor que la merienda debido a la disposición del metabolismo por la mañana.
* Comida: 30,5% --Mayor carga calórica, por ser el eje central del día, y pese a los mitos que corren sobre el desayuno, es posiblemente la comida más importante del día.
* Merienda: 11,5% --Baja carga calórica, por ser una comida de paso y prescindible.
* Cena: 19,75% --Alta carga calórica pero con moderación, es una de las tres comidas principales, pero ocurre cuando el metabolismo esta mas ralentizado, por ello es la mas baja de las tres principales

A la par se calcula el número de Kilocalorías que se tiene que tomar de hidratos, grasas y proteínas, en base al tipo de dieta recomendable para tu patología en la función **distribuciónDeMacronutrientes**, Esta función toma como parámetros las kilocalorías diarias calculadas previamente por la función calculoTMB, y el tipo de dieta del usuario en base a su patología, y te devuelve una lista de los macronutrientes diarios, repartidos en gramos y las kilocalorías totales en el siguiente orden: ListMacDiarios [Kilocalorías, Hidratos, Proteínas, Grasas ], es importante saber que se dividen hidratos, proteínas y grasas, por 4,4 y 8 (respectivamente), para calcular los gramos, y luego de nuevo entre otro número que es la proporción que van a tener (una proporción normal, de una dieta normal sería: Hidrados:50%,proteínas:25%, grasas:25%.

## OrdMinimaDiferencia, formulDif, repartoKcalSobrantes

Son tres funciones que coexisten de alguna manera turnándose las interacciones con el programa. En verdad OrdMinimaDiferencia y formulDif, se usan juntas y se separaron para mayor claridad en caso de problemas, o posibles mejoras. Y repartoKcalSobrantes funciona independientemente, pero siempre de manera que cuando se llama a OrdMinimaDiferencia se llama a repartoKcalSobrantes.

**rapartoKcalSobrantes,** Recoge como parámetros la lista de kilocalorías por comida, la comida en la que nos encontramos y las kilocalorías totales. Se calcula de tal manera que según la comida en la que te encuentres, compara las kilocalorías que has comido con las que deberías haber comida y distribuye lo que sobre/falte entre el resto de las comidas.

Mientras que **OrdMinimaDiferencia** y **formulDif**, son el pilar base de esta aplicación, en cuanto al calculo de comidas se refiere, OrdMinimaDiferencia, coge como parámetros la lista de las comidas, el objetivo, el tipo de comida, lo que ya llevo comido y las kilocalorías diarias. En base a esto hace un calculo de lo que debería llevar de cada macronutriente concreto en cada comida concreta en el total del día(Siendo esto el sumatorio de las comidas anteriores), lo que debería llevar de cada macronutriente específico en esa comida específica, luego se recorre todas las comidas de las bases de datos calculando cual es la que mas se ajusta a lo que necesitamos, una vez cogemos la comida le pasamos todo lo calculado anteriormente, y se lo pasamos a **formulDif**, para llevar a cabo la fórmula, que consiste en:

INSERTAR FORMULA

Si los macronutrientes que debería llevar son mas de lo que en realidad llevo, sacamos la diferencia en base al macronutriente concreto, resto lo que debería llevar a lo que llevo, y dividiéndolo, entre los carbohidratos de la comida en la que nos encontramos, menos los datos del alimento que analizamos y más la resta de las kilocalorías que llevo menos las kilocalorías del alimento que se analiza.

De esta manera el resultado será mayor cuando la ausencia de ese macronutriente sea mayor, y por el lado del denominador, si las kilocalorías y macronutrientes del menú se ajustan a las necesidades del usuario, menos será este y mayor el resultado final. De esta manera se crea el mayor ajuste posible para el usuario.

Se suma el resultado de los 3 macronutrientes y se divide entre tres, creando así un valor por el que posteriormente se ordenará la lista provocando que las primeras recomendaciones sean las más adecuadas.

Cada vez que se elige una comida se hace el reparto de kilocalorías sobrantes y se recalcula el nuevo valor “diferencia”, en base a lo que el usuario a elegido.

## Gráficos

Para comenzar, he de transmitir me no del todo conformidad, respecto que al calculo de los gráficos este dentro de este módulo, pero debido al estado avanzado del proyecto, y la existencia de tres módulos, que siguen un patrón, base de datos, vista y cálculos, se decidió no romper esta estructura, y añadir estos cálculos a calculo dieta, y dividir los grafos en su parte visual (Que veremos a posteriori en el modulo VISTA) y su parte de cálculo1.

Hay dos tipos de gráficos por lo general, que buscan mostrar al usuario de manera clara y transparente su progreso a lo largo del uso del programa, dicho progreso se verá a través de la calidad, por ello, dividirlo en dos tipos de gráficos

Gráfico total, que hace una media de la calidad, es decir del como has comido en todo el día, del ultimo mes, y te lo muestra por pantalla, permitiéndote ver si hay mejora o no.

Los otros tipos de gráficos son por comida, una vez el usuario, vea su progreso general, este puedo mirar los gráficos del desayuno, comida, merienda… Así poder ver que comida le cuesta mas o menos de manera sencilla y clara.

## AdminBase y estructura de la base de datos

### Introducción

En este apartado hablaremos de como esta estructurada la base de datos, y de como se cargan, se guardan y se trabajan con los datos desde el módulo AdminBase, el cual, es un modulo bastante simple, necesario para el trato de datos.

### CargaBaseDeDatos y guardaDatos

Estas dos funciones cargan y guardan los datos de/en la base de datos respectivamente, como se decidió simular una base de datos con un Excel, se usa la librería Pandas, mencionada antes en este apartado para cargar las bases de datos en los respectivos arrays, nada más arrancar el programa, te guarda las tres bases de datos principales, Alimentos, usuarios, patologías e historial en cuatro vectores independientes, que se trabajaran con cada uno de ellos sin tener en cuenta al resto y se guarda el resultado de los cambios producidos durante la ejecución del programa.

Recordemos que estos cuatro valores se sacan de tres Excel diferentes, un Excel nombrado BaseDeDatosDeAlimentos que contiene la lista de alimentos y patologías, otro llamado BaseDeDatosUsuarios, en un principio estos dos estaban juntos, pero separarlos en dos Excel diferentes nos permite la mejor trata de datos a la hora del almacenamiento, y el historial, el cual lleva consigo un listado de las fechas y los usuarios como clave primaria, junto a todo lo que han comido ese día.

La base de datos se estructura en dos archivos Excel independientes, donde cada hoja seria lo equivalente a una colección de una base de datos, y cada fila seria un objeto dentro de dicha colección. A distinguir las siguientes hojas:

#### Alimentos

La hoja de alimentos, en verdad, sería una hoja de menús ya construidos los cuales tienen 10 campos, sobre los que actualmente se trabajan 8,pero se mantiene su estructura para posibles ampliaciones.

La clave primaria de esta colección sería el nombre, en un futuro esta pensado añadir un id, o depende la metodología de desarrollo de la base de datos, ese id se generaría de manera automática teniendo en cuenta que no sigue ningún tipo de normal en relación con el alimento.

Los campos son:



Nombre (PK), Calorías, Grasa, Saturadas (Sin uso), Hidratos, Azucares (Sin uso), Proteína, Tipo, LRE, Calidad

De los primeros siete campos no hay mucho mas que añadir, que lo que el propio nombre indica, no obstante, se hará una pequeña aclaración de los últimos tres.

El **tipo**, marca el tipo de comida que es, se pensó en que metodología usar, teniendo en cuanta que un mismo menú puede ser a la vez, desayuno, almuerzo y merienda, cena y comida, y un largo etc. Por ello se pensaron dos posibilidades, la primera poner la inicial, letra o nombre de la comida para la que valga ese menú, tratarlo como un array y descomponerlo en su debido momento, pero en el futuro caso de que se diera la posibilidad al usuario de editar un alimento de manera que no le apareciese entre una comida específica se complicaba algo más, y la segunda opción era tratarlo como una cadena de bits: desayuno-almuerzo-comida-merienda-cena, donde el valor 1 sería si es valido para esa comida y 0 si no lo es. Por ejemplo:

1-1-0-1-0

Significa que es desayuno, almuerzo y merienda, pero no comida o cena, luego esto se traduce en el número decimal, que en este caso sería 26.

El **LRE**, recibe ese nombre por un juego de palabras con la gestión de procesos de un sistema operativo LRU (last reciently used), pues lo que hace es llevar una métrica de la frecuencia con la que el usuario toma esa opción, literalmente significa “last reciently eat”.

La **Calidad**, como el propio nombre indica, es la calidad del alimento, es la variable de indicar la diferencia entre 100 kilocalorías de ensalada y 100 kilocalorías de azúcar, tiene un rango de valores entre el uno y el cuatro, siendo cuatro la peor calidad de todos, y uno por el contrario la mejor, la calidad va estrechamente relacionada con el umbral del momento y la comida dentro del programa, esta variable, empieza con el valor uno (siendo este el mas bajo posible) y según vas refrescando va aumentando para que las posibilidades de menú aumenten con el número de veces que refresques la comida. El umbral sirve para cribar del DataFrame de la comida que estemos trabajando, todos los alimentos cuya calidad sea superior al valor del umbral en el momento de selección, de esta manera se le complicará al usuario la posibilidad de hacer una mala elección a la hora de organizar su menú, además que se verá reflejado en el momento en la barra de progresión.

##### Añadir nuevos Alimentos

En el programa se incluirá la opción de poder añadir nuevos menús a la base de datos, cómodamente por parte del usuario, de esta manera, el usuario tendrá una mayor libertad a la hora de poner Menús típicos que se obvien hasta la fecha en el programa. Se le recordará la importancia de ser realista con la comida, para que el programa pueda seguir haciendo su función.

#### Usuarios

La hoja de usuarios alberga los datos de todos los usuarios que usan la aplicación la estructura de esta colección en si es muy básica y contiene toda la información necesaria para el calculo de la dieta, del usuario.

La clave primaria sería el DNI del usuario, el cual, se encuentra dentro del campo ID. La hoja se estructura de la siguiente manera:

Id (PK)-nombre-apellido-password-sexo-edad-altura-peso-actividad-patología (FK de la tabla Patologías) y tipo.

Tanto **id**, como **nombre**, **apellido** y **password**, son los datos privados del usuario, los cuales, son simplemente informativos y no tienen ningún valor adicional en el calculo de la dieta y los resultados. En cambio, el **sexo**, la **edad**, **altura**, **peso**, **actividad**, **patología** y **tipo**, influyen de manera directa con el calculo de la dieta, como pasaba en la hoja de alimentos en los primeros campos su nombre, explica su significado. A tener en cuenta, que el valor del campo **patología** es numérico para mayor privacidad y conexión con la base de datos de Patologías, pues es el id de cada patología y en base a su id se busca la información sobre dicha fila. En el caso de valer -1, significa que el usuario no tiene ningún tipo de patología y su uso es exclusivamente para el aprendizaje y seguimiento de la dietoterapia adecuada. La **actividad** tiene valores entre 0 y 4, donde cero significa el máximo nivel de sedentarismo y 4 el máximo nivel de actividad. Por ultimo el **tipo**, hace referencia, a los objetivos que el usuario tiene respecto a si mismo, si quiere mantenerse, subir de peso, o bajar.

#### Patologías

En esta hoja, la cual se encuentra dentro de las BaseDeDatosDeAlimentos, se encuentra la información básica de las patologías nombre, id, y el tipo de dieta que lleva alto o bajo en carbohidratos, proteínas o grasas. Esto se carga al inicio del programa, se checkea si el usuario padece alguna patología y se selecciona el tipo de dieta correspondiente.

### Guardar Todo

Función que se encarga de juntar en una única función el almacenamiento de todos los datos con los que se trabaja en esta herramienta. Dentro de esta función encontramos:

* guardarHistorial
* guardarUsuario
* guardarDatos

A continuación, daremos una breve explicación del ¿Por qué? en cada una.

#### guardarHistorial

Como el propio nombre indica, se encarga de almacenar las selecciones que hicimos en el mismo día en la base de datos. A través de la librería dateTime de Python sacamos el día de hoy, con la fecha en forma de cadena de caracteres y el DNI del usuario creamos la clave primaria, no se puede repetir dos veces esta combinación, si a la hora de guardar ya existe, se actualiza pero no se añade; se crea un objeto tipo DataFrame segregado del array menuDeHoy, y se escribe en el Excel “historial”.

#### guardarDatos

Mencionada en el apartado anterior, se encarga de guardar el groso de la aplicación, que es todos los cambios que hayas hechos en la aplicación respecto a la comida permitiendo que el programa poco a poco se vaya adecuando a los gustos y lo que mas come el usuario.

#### guardarUsuario

Guarda los datos del usuario. Se llama en dos ocasiones, cuando se guarda todo y cuando se edita la información del usuario. La primera vez, en la ocasión que se guarda todo, es redundante, se hace para tener un respaldo en caso de que al editar usuario se de algún fallo y no se guarde, en un principio, en el momento que edites la información del usuario y se pulse en “Aceptar y guardar”, se guardarían los datos.

### getXXXX

En este apartado se respaldaran las tres funciones que se encargan de dar la información de una fila concreta de una de las colecciones de la base de datos, estas tres funciones son:

* getFilaPatologia(patologíaID,hojaPatologia)
* getFilaUsuario(userId,HojaUsuarios)
* getFilaAlimento(NombreAlimento,HojaAlimentos)

## VISTA

### Introducción

Para empezar, recordar que se ha seguido una adaptación de la estructura típico MVC, donde se sustituye el modelo y el controlador propio y se tiene una vista adaptada, esto ya fue explicado al inicio de este apartado, a continuación me centrare en la vista

En el módulo vista se muestran, o llevan a cabo toda función que tendrá influencia en lo que el usuario ve, es decir, actualización de pantallas, muestra de datos, transacciones entre las dos ventanas principales, el cambio de los gráficos, etc.

El modulo vista contiene tres variables calves globales, que son el DNI del usuario (usr), la contraseña del usuario (contraseña) y la bandera. De los dos primeros poco hay que explicar, pero del ultimo sí, esto se hará en el siguiente apartado, junto a la debida explicación de las funciones que hacen uso de ello.

## Cambio y get bandera

Empezando por la función “cambio”, que se nombro de esta manera porque indica un cambio en la funcionalidad del programa, explicamos esto:

El programa sigue su curso, cuando se cierra la ventana de log-in, o acceso, según como se haya procesado la información en cambio cuando la ventana se cierra, el programa continúa con su normal ejecución o se cierra completamente.

Cambio comprueba que la combinación usuario y contraseña sea válida en caso de ser así cambia el valor de bandera a verdadero (True), indicando así que el acceso a sido un éxito, si no, la bandera seguirá a False, y se mostrara por pantalla un mensaje de error.

Cuando la tupla usuario y contraseña es validad cierra la pantalla de “log-in”, lo que hace que en el “main” se siga con su normal ejecución, lo cual lleva, de manera directa a una condición que comprueba que la bandera es verdadera, si lo es relanza la aplicación principal, sino cierra el programa, y se termina toda ejecución.

Como el propio nombre indica getBandera devuelve este valior al main.

### Intercambio entre frames

Como se mencionó en la introducción del módulo vista, dicho módulo no solo se encarga de la parte visual, sino de las transacciones conexas a estas. Para ello existe una función principal: **show\_frame**, la cual coge el frame que se quiere mostrar y la clase que instancia dicho frame (explicado extensamente en el módulo Main), y lo lanza a través de la propiedad de Tkinter, ‘tkraise’

### Seleccionar y refrescar

Sin duda alguna esto presento uno de los mayores retos de la aplicación debido a las deficiencias que presenta Python para construir interfaces gráficas, y dentro de dichas deficiencias generales, se añadía la dificultad que presentaba el simplismo de la librería tkinter para dichas interfaces.

Todo surgió de la necesidad de actualizar lo que se mostraba por pantalla cada vez que hacía una elección, pues si se quería sacar de este proyecto la mayor precisión posible del sistema de recomendación, cada elección, cada cambio de variable, debía suponer nuevas recomendaciones, esto en un principio parecía sencillo, pero Tkinter no permite la recarga de los datos, sino que al inicio del programa lo “dibuja”, todo de manera simultánea, impidiendo una actualización del frame cada vez que haya un cambio, por ello se crearon una serie de funciones, necesarias las cuales se engloban en una mayor, para que de una llamada realice todas las funciones necesarias. **seleccionarYActualizarResto**, esta función se encarga principalmente de actualizar todos los datos en pantalla, el menú que hemos comido hoy, ademñas de llamar a la función  **seleccionar**, la cual guarda la opción escogida y bloquea las posibles opciones, y acto seguido recorre las otros cuatro frames recalculando todas las listas, todas los resultados y sus respectivos LRE, etc.

Se ha decidido que para evitar errores de coherencia el programa solo actualizará aquellos frames no bloqueados por el método seleccionar, sacrificando un pequeño porcentaje de la precisión, para conseguir un funcionamiento mas fluido sin errores de compatibilidad o múltiples selecciones.

## Graficos

Hago una pequeña mención, pues esta parte del módulo de la vista es la continuación de los gráficos del módulo de los cálculos de la dieta, se basa en una función básica en la cual se le pasa el gráfico que se quiere usar con sus respectivos parámetros y se modifica al que ya estuviera siendo mostrado, esto permite mejor navegabilidad y una mejora en el rendimiento del programa.

### Editar usuario y estilo

Como pasaba con anterioridad con el calculo de los gráficos y el módulo calculosDieta, me encontré con la incertidumbre de en que parte de la estructura de programación desarrollada durante este programa podría meter el editar usuario y el estilo, pero siguiendo la lógica de la estructura usada en el programa se decidió meterlo en este módulo, pues la posibilidad de editar la información del usuario es otro frame diferente, en el cual la única parte que implica a otra cosa que no sea la vista, es en la persistencia de los datos, y dicha parte se delega al administrador de la base de datos, haciendo uso de las funciones pertinentes del programa.

En el caso de los estilos, si que se mezcla ligeramente la administración de la base de datos, con el diseño, es un punto intermedio, que fue difícil de categorizar, pero finalmente se decanto por la vista debido a que su finalidad es puramente visual, y en verdad interacciona mas con el usuario que con la propia base de datos. Se crea un pequeño archivo que contiene los colores bi-temáticos de la aplicación

## Main

Es la columna vertebral del programa, el cual, se encarga de cargar y procesar toda la información relevante que más adelante se va a ir editando en el programa, el main tiene una única función y el resto se divide en clases que se instancian en esta función.

La función principal hace de flujo de entrada y salida, donde un if condicional hace la función de interruptor permitiendo que la aplicación original, distribuida en clases se lance, o se cierre, como se ha explicado anteriormente.

Todo se basa en una especie de jerarquía de objetos, en el que el objeto principal (Menú principal), contiene otros tres objetos que estos a su vez contienen otros tantos, creando una simple estructura de árbol, entendible por cualquier programador.

De entre las tres subclases, instanciadas dentro del Menú principal (Usuario, Dieta, Historial), cabe destacar la de Dieta, pues esta a su ver tiene una clase por cada comida, estas clases son actualizadas constantemente en cuanto se hace algún cambio en el programa, esto es una abstracción del **patrón de diseño observador**, en el que las clases observan esperando el cambio para actualizar sus recomendaciones.

Se quiso evitar un main lioso con cosas que no son propias del dicho, por ello se separo en lo que es el main, y las clases las cuales, se encargan del funcionamiento principal del programa. De alguna forma, podríamos ver el main como los cimientos del programa completo, y los diferentes módulos, son los encargados de terminar de construir el programa.

# Herramientas

## Introducción

Una vez explicadas el por que de cada decisión que se ha tomado a lo largo del desarrollo del programa, y expuesto cada decisión meticulosamente, procurando que cualquier duda del lector respecto a como se ha trabajado quede resuelta; Nos centraremos en mencionar de manera breve, dando una ligera explicación del por qué, se han usado las herramientas que se han usado para el desarrollo del programa.

## Anaconda

Distribución de código abierto, la cual, esta indicada para el análisis, procesamiento y computo de datos, de Python y R, trae consigo una serie de programas y características entre los que destacaremos: Spyder y VisualCode

Sobre la razón, por la cual, se decidió usar anaconda es básica, la propia definición lo dice, a fin de cuentas, es una distribución para el análisis de datos con Python y R, lo cual, es de manera encubierta, el principal trabajo en este proyecto.

## Spyder

Entorno de desarrollo multiplataforma para Python, se incluye dentro del paquete de Anaconda, el cual, desde su propia prompt, te permite la descarga de librerías, para un rápido desarrollo siendo una herramienta potente e intuitiva para el trabajo diario. Debido a su sistema de reconocimiento de clases, y su capacidad de ejecutar desde el propio programa el proyecto, se convierto en el entorno de desarrollo principal de todo el programa.

## VisualCode

Al igual que Spyder, es un entorno de desarrollo incluido en el paquete de anaconda, con la diferencia de que VisualCode, es ampliamente utilizado por la comunidad de programadores para múltiples lenguajes, y frameworks, como angular2, javascript, c3, Python, ect.

Se pensó largo y tendido sobre que entorno utilizar entre Spyder y VisualCode, y durante los primeros periodos de la aplicación, se turnaron ambos sin ninguna razón aparante, con el tiempo, me sentí más cómodo trabajando con Spyder, y lo fui adoptando como entorno principal, delegando en VisualCode la tarea de hacer debug, debido a la notable mejora que este presenta sobre Spyder.

Indicar que antes de empezar se probaron varios entornos como NoteBook, PyCharm, y eclipse añadiendo el API de Python, pero se acabo decantando por estas dos herramientas nombradas anteriormente.

## GitHub

Proyección de la metodología GIT para el almacenamiento y control de versiones, sin duda GIT es actualmente la estructura más usada en el mundo de la programación, y GitHub, una de las herramientas más conocidas y valoradas que hay. Con el tiempo, fue mi mayor aliado, permitiéndome recuperar antiguas versiones con el programa era corrompido por algún error lógico, y dando una seguridad de soporte sobre el desarrollo del proyecto.

## GitKraken

Potente interfaz gráfica multiplataforma desarrollada por Electron, que nos permite controlar de forma sencilla y cómoda nuestro repositorio GitHub, permitiendo que el uso y control del GitHub fuera más ameno y rápido, sin duda es una de las mejores aplicaciones que he probado, y de las que mas me han ayudado a lo largo del proyecto.