**Manual de uso modulo Datafab**

**Autores**: Arkantos1000

**Versión**: 1.0

Con **Datafab** podrás generar archivos para usarlos en procesos de Testing.

El módulo **Datafab** tiene por objetivo general, generar datos ficticios siguiendo las reglas que el usuario especifique.

Una de las características del modulo es su personalización; De tal modo que puedes crear nuevos componentes e integrarlos a la fábrica.

**Demostración general**

En esta sección descompilaremos un caso usual al trabajar con el módulo Datafab



Explicaremos cada parte del código:

La clase Datafab por defecto tiene cargados 5 “módulos”, cada uno asignados con un número, los números del 1 al 5 mostraran lo siguiente:

1. Nombres
2. Genero
3. Apellidos
4. Fecha de sistema
5. Valor autoincremental

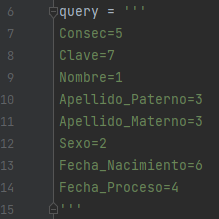
Tras exportar el módulo datafab, agregamos dos nuevos “módulos” al atributo “modules” de nuestra clase Datafab; **random\_date**() y **range\_code**(start, end), **random\_date** creara una fecha aleatoria entre los años 1980 y los dos años siguientes al año actual del sistema y **range\_code** creara códigos numéricos entre los rangos indicados.

En nuestro código estamos indicando que este par de nuevos módulos se identificaran con las llaves 6 y 7, respectivamente. Es recomendable manejar la llave de los módulos en int.



Para iniciar un objeto de tipo **Datafab** requerimos de un “query” o dicho de otra forma, una serie de columnas con el tipo de valor que imprimirán.

La sintaxis seria*: NombreColumna=Valor*



Nótese que aquí estamos asignando cada uno de los módulos como valor, tanto los predefinidos como los configurados por el usuario.

Tras tener nuestro “query”, podremos inicializar un objeto de tipo **Datafab**



Una vez hechas las configuraciones y cargado nuestro objeto de tipo **Datafab** con su respectivo query, podremos disponer de una función en nuestro objeto, llamada **generate\_file**, la cual requiere dos parámetros:

1. Cantidad de registros
2. Ruta del archivo a crear



**Clases y funciones explicadas**

**Datafab** (Class)

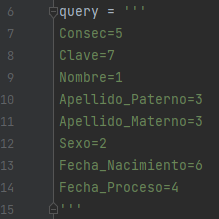
La clase **Datafab** es la maquinaria principal del módulo, dentro de ella podremos modificar los componentes que conforman la fábrica para crear información ficticia.

**Parámetros de inicialización**

**query (str):** Para cargar nuestro query tendremos que iniciar un string multilínea (‘’’str’’’’), donde cada línea representa una columna de nuestros datos, la sintaxis para agregar nuevas columnas y asignarle un módulo, es la siguiente:

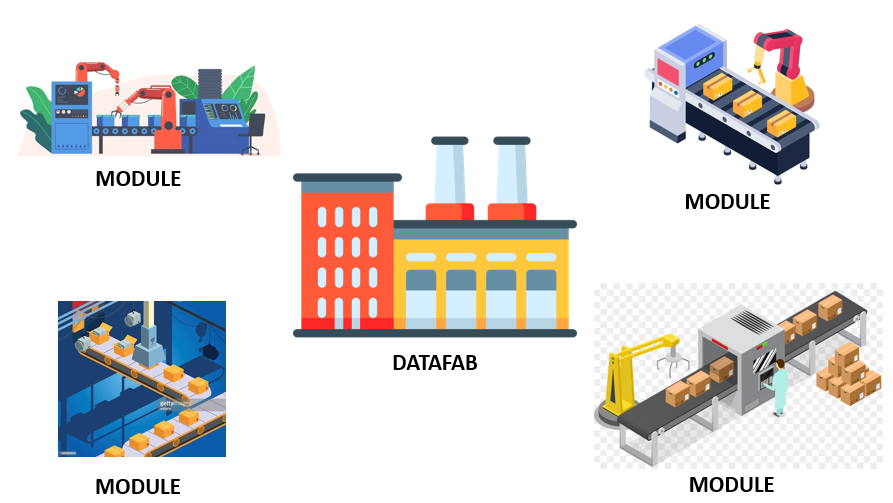
***NOMBRE=MODULO***

Siguiendo este criterio, podremos crear las columnas asignándole el “módulo” a usar para crear sus propios datos. Para más información acerca de los “módulos”, ir al apartado de Flujo de trabajo o Person.modules.



**modules** (Attribute: dict)

Un “modulo” en Datafab hará referencia a una sección de la fábrica, por la cual puede o no pasar la información para ser procesada, así como una fábrica tiene distintas secciones donde se elaboran distintos productos, en Datafab usaremos los **módulos** para elaborar distintos *tipos de datos*.



El atributo **modules** es probablemente una de las cosas que más brillaran en tus programas usando el módulo **Datafab**, pues no solo se limita a trabajar con los “módulos” pre-hechos por el módulo, ¡si no que puedes implementar tus propios “módulos” personalizados!

Existen 4 tipos de datos que pueden ser considerados como módulos:

* Clases
* Atributo de una Clase
* Generador
* Strings

Cualquiera que no sea uno de estos tipos de datos, será convertido automáticamente a un String.

Aquí viene la pregunta elemental ¿Cómo es que puedo insertar estos valores para que sean reconocidos por la Fabrica? Todos estos módulos están integrados en un diccionario, donde la clave será su identificador en el query y el valor el tipo de dato que puede ser integrado, a continuación, se mostrará una sintaxis de inserción en el diccionario:

**Clases**

***1: Class***

Es importante saber que lo que se devolverá por la clase es lo que está definido en la función especial \_\_str\_\_.

**Atributo de una Clase**

***1: [Class, ‘attribute’]***

Si usted ha declarado una clase en algún otro modulo y se pregunta si devolverá el atributo de ese objeto, la respuesta es: sí. Es obligatorio que ese atributo ya tenga un valor.

**Generador**

***1: Generator()***

Si usted intenta hacer uso de una función, necesitara cambiar su enfoque a un generador, con esta opción podrá generar sus datos.

**String**

***1: ‘str’***

Cualquier cosa que no sea una de las anteriores, se transformara a un string.

Por defecto podrá encontrar que las llaves de la 1-5 están ocupadas por los siguientes elementos, podrás encontrar la respectiva documentación a lo largo del Manual:

1. **Person**

Retornara un nombre de la Clase Person.

1. **[Person, ‘genre’]**

Retornara el género de la persona seleccionada, de la Clase Person.

1. **LastName**

Seleccionara un apellido de la Clase LastName.

1. **date.today()**

¡Qué las apariencias no te engañen! Esta es una función que retornará la fecha del sistema, un valor estático, será tratado como un string.

1. **consec()**

Se trata de un generador que genera un valor autoincremental.

**lay\_out** (Attribute: list)

Es una secuencia con los módulos a ejecutar.

**generate\_rows** (Function)

Esta función devolverá una *lista* con los registros generados.

**Parámetros obligatorios**

**row\_amount (int):** La cantidad de registros a generar.

**Parámetros opcionales**

**header (bool)**: Cuando esta opción está en True, pondrá encabezado a los registros, por defecto está en True.

**generate\_rows** (Function)

Esta función generara un archivo con los registros indicados.

**Parámetros obligatorios**

**row\_amount (int):** La cantidad de registros a generar.

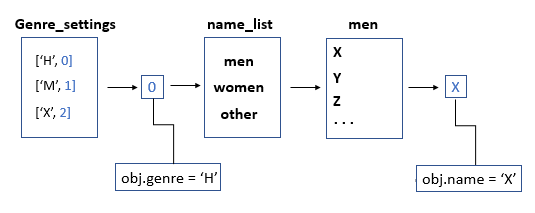
**path (str):** La ruta completa del archivo a generar.

**Parámetros opcionales**

**delimiter (str)**: El delimitador del archivo, por defecto es la “**,**”.

**header (bool)**: Cuando esta opción está en True, pondrá encabezado a los registros, por defecto está en True.

**Person** (Class)

La clase **Person** se utiliza para invocar un “nombre”. A continuación, se presenta un flujo que describe como es que se utilizan el resto de atributos para llegar al correspondiente resultado.

Tras inicializarse un objeto de clase **Person** “aleatoriamente” tomara uno de los valores de *genre\_settings*, tras tomar uno de los valores, ira a su índice en *name\_list* y finalmente seleccionare uno de los nombres aleatorios de esa lista.

Predeterminadamente esta alojada en *Datafab.Modules* en la clave 1.

**men** (Attribute: list)

Esta lista contiene una serie de nombres masculinos, es usada en el atributo *name\_list* , para más información, ir a *name\_list* .

**women** (Attribute: list)

Esta lista contiene una serie de nombres femeninos, es usada en el atributo *name\_list* . Para más información, ir a la sección de *name\_list* .

**name\_list** (Attribute: list)

Por defecto esta lista contiene a los atributos *men* y *women*, cualquier lista que este dentro de **name\_list** estará participando por ser seleccionada.

**genre\_settings** (Attribute: list)

Para configurar **genre\_settings**, deberemos tener en cuenta que es una lista con listas dentro, cada lista con dos valores, el primero hará referencia a que “Genero” seleccionara y el segundo, al índice que tiene que ir en *name\_list* para seleccionar un nombre de su lista asignada.

**name** (Attribute: str)

Es el nombre que fue seleccionado. Esta asignado al método especial *\_\_str\_\_*.

**genre** (Attribute: str)

Es el género que fue seleccionado. Predeterminadamente esta alojada en *Datafab.Modules* en la clave 2.

**LastName** (Class)

La clase **LastName** se utiliza para invocar un “apellido” aleatorio de una lista dada, esta lista puede ser modificada desde el Atributo *last\_name\_list* .

Predeterminadamente esta alojada en *Datafab.Modules* en la clave 3.

**last\_name\_list** (Attribute: *list*)

Desde esta lista la clase **LastName** define a sus objetos con un nombre. Puedes modificar los elementos de la lista para que se elija un elemento que será utilizado por la función especial *\_\_str\_\_*.

**range\_code** (Generator)

Devuelve un número entre *start* y *end* .

**Parámetros obligatorios**

**start (int):** El número más bajo a poder ser seleccionado.

**end (int):** El número más alto a poder ser seleccionado.

**consec** (Generator)

Devuelve un número autoincremental.

**Parámetros opcionales**

**start\_with (int)**: El número desde el que iniciara, por defecto inicia en 1.

**incremented\_by (int)**: El número de incrementos que sufrirá tras iteración, por defecto inicia en 1.

**random\_date** (Generator)

Elige una fecha aleatoria entre los años 1980 y dos años más a la fecha del sistema.

**Parámetros opcionales**

**form (str)**: El formato con el que se generara la fecha, usa las reglas de formato del módulo datetime.strftime, por defecto su formato es ‘%d-%m-%Y’

**regex\_code** (Generator)

Devuelve una cadena basándose en un patrón, con la posibilidad de usar expresiones regulares.

**Diccionario de expresiones regulares.**

\d – Cualquier digito del 0 al 9.

\l – Cualquier letra del alfabeto ASCII en minúscula.

\L - Cualquier letra del alfabeto ASCII en mayúscula.

{n} – Seguido a la expresión regular, debe ir la cantidad de veces que se repetirá entre llaves.

**Parámetros obligatorios**

**pattern (str)**: El patrón a tomar como base. A continuación, un ejemplo.



Lo que queremos indicar es que se repetirán dos dígitos, la palabra manzana, dos dígitos, la palabra pera y tres letras en mayúscula.

