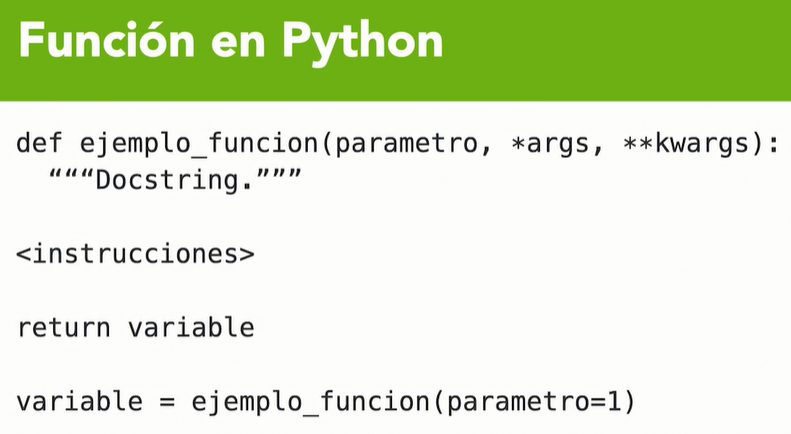
**Función:**

Es un bloque de código que se encarga de realizar una tarea específica, cada función es independiente ya que contiene instrucciones relacionadas entre sí, que se encargan de realizar paso a paso la tarea para la cual han sido creadas, las funciones nos permiten tener el código organizado y separado en pequeñas partes, estas evitan que se repitan las mismas instrucciones en diferentes partes del código, ya que para hacer una misma tarea invocamos a la función en vez de escribir las mismas líneas repetidas veces, estas permiten la legibilidad ya que son bloques de código que se encargan de realizar una sola tarea y además van acompañadas de una documentación necesaria para entenderlo fácilmente, en Python, escribimos una función de la siguiente manera:

**Conceptos:**

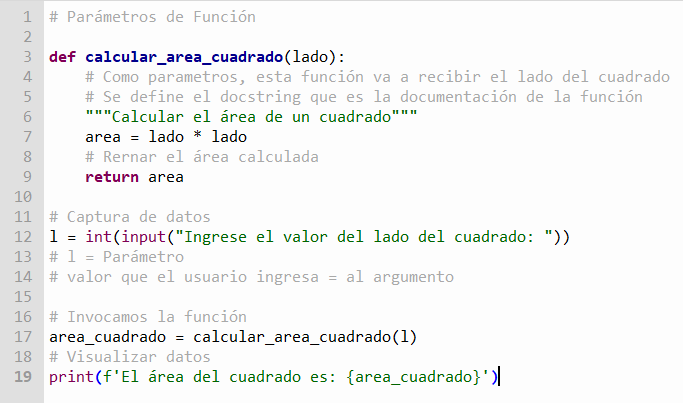
**Parámetro:** Variables que referencian a los datos que ingresan a la función, el parámetro, es el nombre de la variable.

**Argumento:** Son los valores asignados a cada uno de los parámetros de la función cuando esta es invocada, parámetro y argumento, hacen referencia al valor que entra a la función, parámetro es el nombre y argumento es el valor asignado.

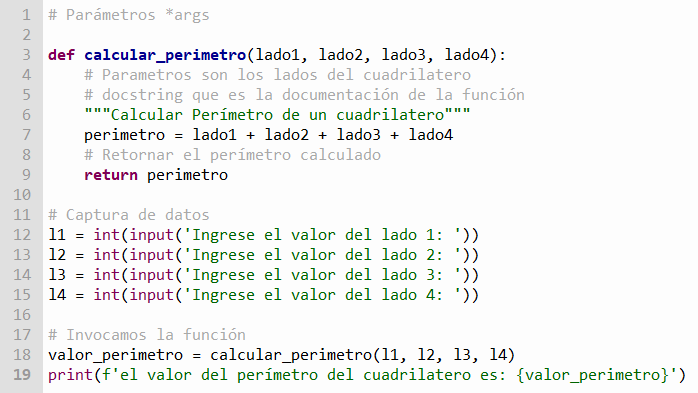
**\*args:** Hace referencia a Nom Key Word Arguments, son parámetros especiales usados para pasar parámetros opcionales en forma de lista, cada posición en la lista de ese parámetro, será usado en la función, se declara usando un asterisco antes del nombre args (\*args).

**\*\*kwargs:** significa key Word arguments, son parámetros especiales, usados en las funciones para pasar parámetros opcionales en forma de diccionario, ya que los kwargs, llegan en forma de diccionario, cada parámetro, tiene un nombre asignado que sería la llave del diccionario y un valor asociado, se declara usando dos asteriscos antes del nombre kwargs (\*\*kwargs).

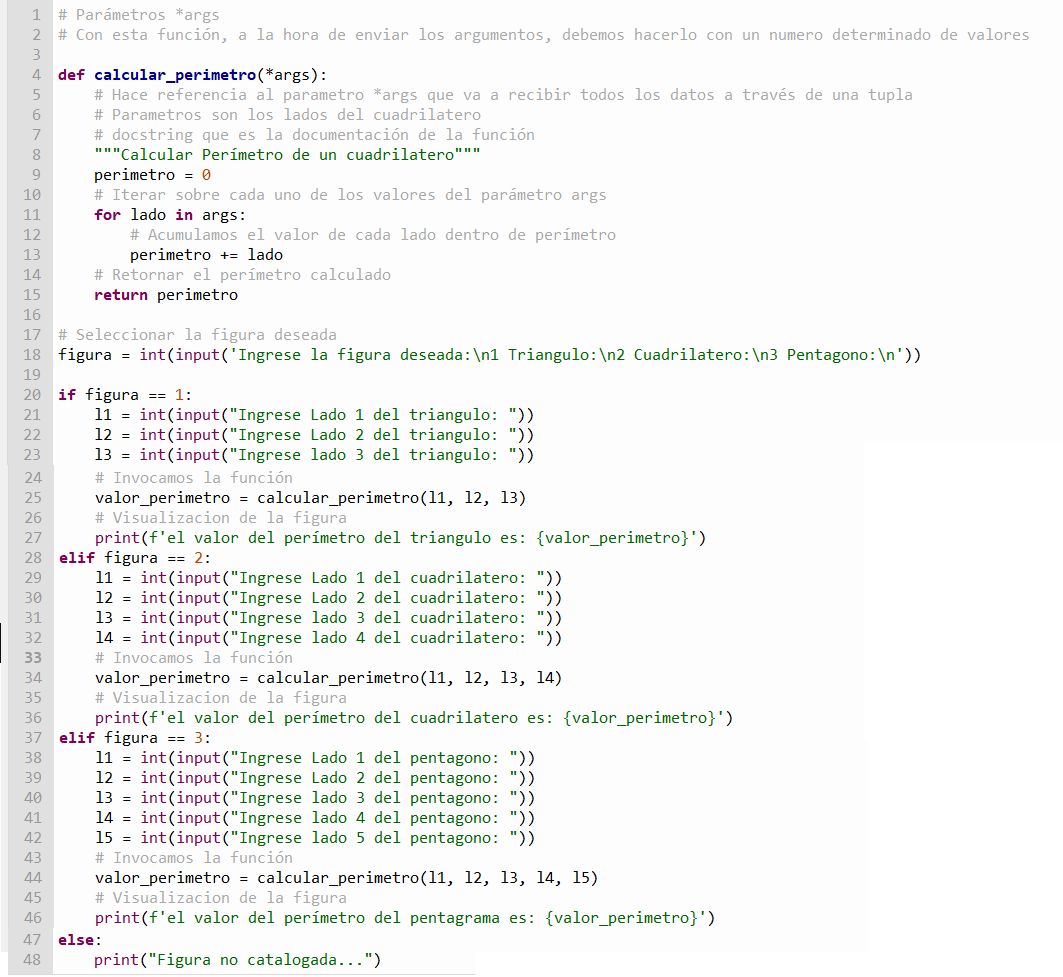
Retorno: Es el elemento que regresa la función como resultado del proceso o tarea que esta realiza.

Ejemplo:

Los parámetros args, son parámetros especiales que permiten pasar a la función múltiples datos en forma de tupla, este parámetro es opcional, por convenció se usa el nombre de args que hace referencia a arguments, estos se identifican cuando vemos un \* seguido de un parámetro, en algunas ocasiones, los usamos cuando no sabemos cuántos parámetros debemos pasar a la función o no queremos definir una gran cantidad de parámetros que pueden ser recibidos en forma de tupla.

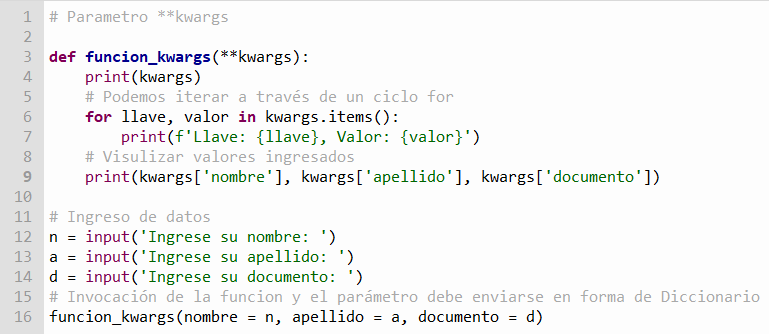
Forma habitual:

Si quisiéramos ingresar más argumentos, colocamos el parámetro \*args y hacemos uso del ciclo for para calcular el perímetro de la figura:



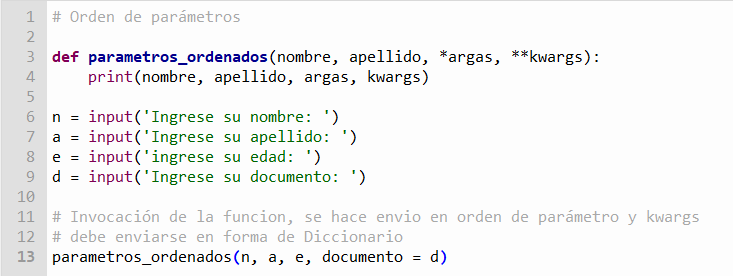
Los parámetros kwargs, son parámetros que nos permiten enviar datos no declarados a la función en forma de llave y valor, es decir en un diccionario, significa argumentos de palabra clave en español, por convención se llaman \*args y \*\*kwargs, pero se nombran de cualquier manera, son los asteriscos los que en realidad los diferencian.

Ejemplo:

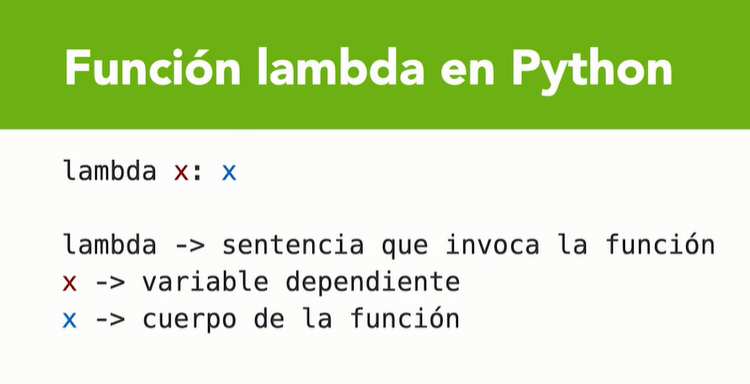


Orden de los parámetros: podemos usar en una misma función los tipos de parámetros combinados, primero van los parámetros normales, segundo lugar los \*args y por último los \*\*kwargs

Ejemplo:

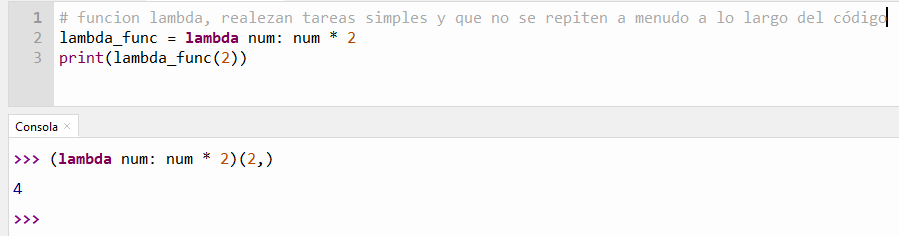


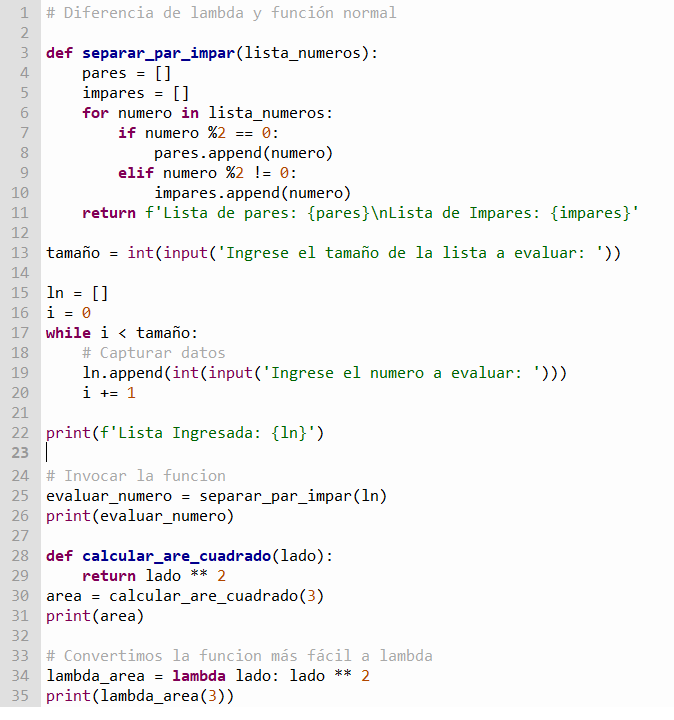
Calculo lambda:

Es una expresión sencilla de una sola línea, esta es un tipo de programación funcional, es decir basada en funciones, las cuales se componen de una sola expresión, una sola línea, al construir una función lambda, esta puede tomar uno o más argumentos.

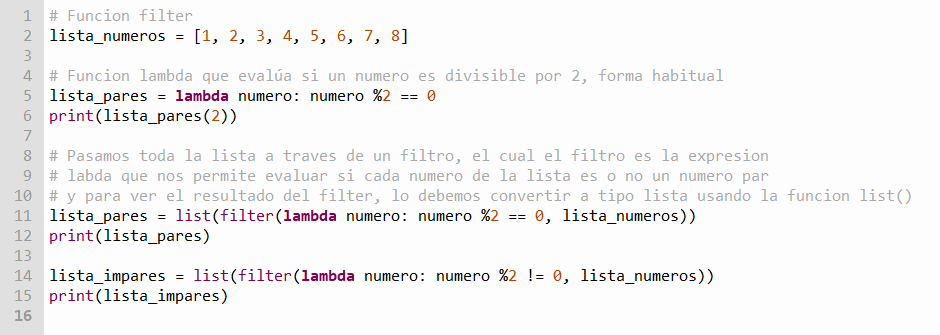
En la programación, se conoce como función anónima a una función sin nombre, se declara únicamente a través de una expresión que nos retorna un resultado, en Python una función anónima, se define como una función que retorna un valor, pero no se define usando la sentencia def, estas tienen diferentes nombres, de acuerdo al lenguaje de programación pueden ser funciones anónimas, expresiones lambda, funciones lambda, entre otros, en pyithon las funciones anónimas son las funciones lambda y se declaran a través de la sentencia lambda, este tipo de funciones se usa en programación funcional, su estructura es:

Lambda argumentos: expresión a evaluar



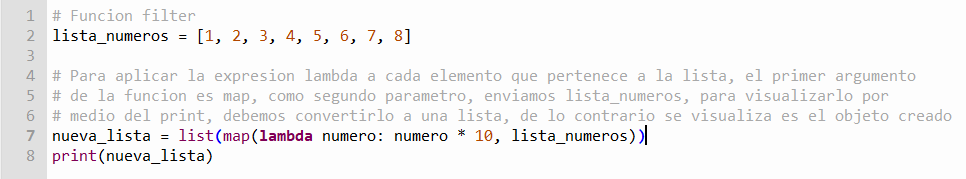


Muchos integrantes de la comunidad de Python, sugieren no hacer mucho uso de las funciones lambda, ya que pueden llegar a ser difíciles de leer, sin embargo estas funciones siguen existiendo y se usan para hacer filtros con ayuda de las función filter o para aplicar una función a los elementos de una lista usando la función map.

Ejemplo función filter:

Ejemplo función map:

Esta función recibe dos argumentos, el primero es una función o una expresión lambda, y l segundo un objeto iterable al cual le aplicaremos una función.



Iterables e iteradores:

Iterable: es el nombre que recibe un conjunto de elementos y este debe permitir que se retornen sus elementos, los objetos iterables se pueden recorrer usando ciclos y obtener cada uno de sus elementos:

Los iterables pueden ser, cadenas de texto, estructuras de datos como listas, tuplas o diccionarios.

Los iteradores son objetos, estos nos permiten recorrer un objeto iterable, son el objeto que nos permite obtener de un elemento iterable cada uno de los elementos que lo componen.

En Python existen dos funciones que nos permiten trabajar con iterables sin tener que hacer uso de los ciclos.

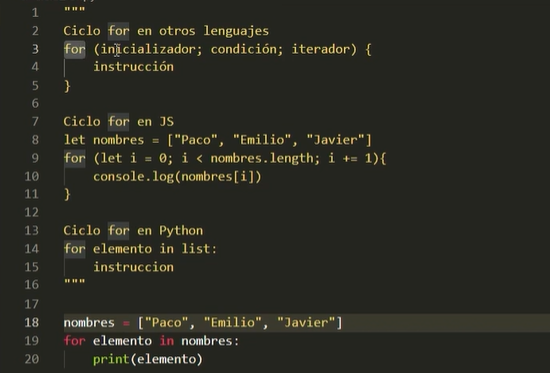
La función iter(): Crea un iterador y de esta manera permite recorrer el conjunto de elementos u objeto iterable.

La función nex(): Recibe el iterador creado y cada vez que se invoca, retorna el elemento siguiente del objeto iterable.

En Python, no es común encontrar la función iter para crear objetos iteradores, sin embargo, esta función es usada como base de otras funciones.

Ciclo for:

Es la manera tradicional de escribir, listas tuplas y diccionarios, iteramos sobre cada uno de los elementos que hacen parte del objeto iterable sin necesidad de usar un índice que indique la posición de cada elemento:



La sentencia break, detiene totalmente el ciclo y pasa a evaluar la siguiente sentencia.

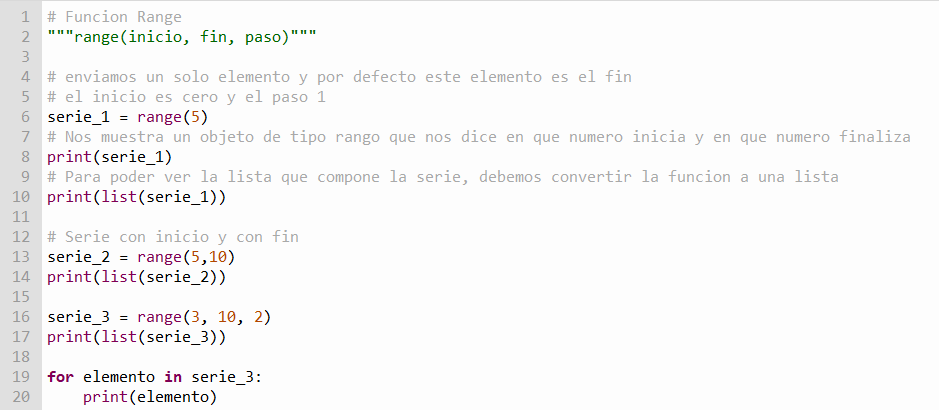
La sentencia Break, apenas encuentra la condición que se cumple para el ciclo.

La sentencia continue, apenas encuentra la sentencia no la ejecuta y sigue con las siguientes sentencias.

Estas dos instrucciones, se usan para evaluar si una condición se cumple,

Función range()

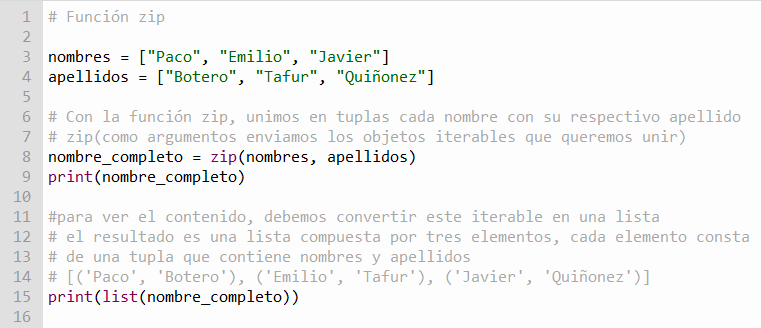
Crea una serie de números consecutivos, esta se inicia en cero y termina un numero antes del elemento que se entrega a la función, cada número de la serie, se incrementa en 1.



Función zip

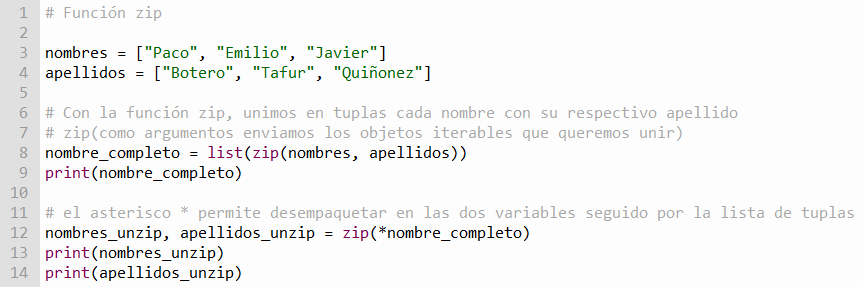
Crea un objeto iterable a partir de otros objetos iterables, cuando usamos la función zip, estamos uniendo uno o más iterables y cada unión de elementos resulta en una tupla.

Si queremos unir dos iterables de diferentes tamaños, este va a contener una lista de menos elementos y esta cantidad de elementos, siempre será igual al iterable de menos elementos.

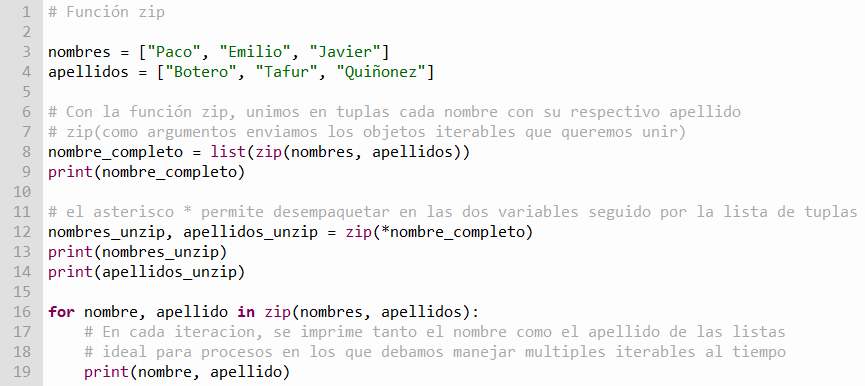


La función unzip

Sirve para desempaquetar una lista o iterables de tuplas en nuevas tuplas con con los valores originales que teníamos con nuevos iterables, este se desempaqueta en la cantidad de elementos que contiene cada tupla.

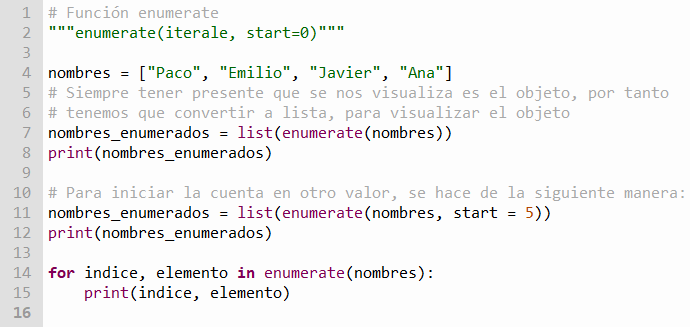


La función zip, puede ser usada para iterar sobre dos objetos iterables al mismo tiempo:



Función enumerate

Esta recibe un objeto iterable y retorna tupla en las que cada una contiene un elemento del objeto que recibe y un índice que indica su posición, por defecto esta función retorna el índice iniciando en cero, aunque este puede ser modificado, muy importante a la hora de hacer uso del elemento y su posición en el objeto iterable.



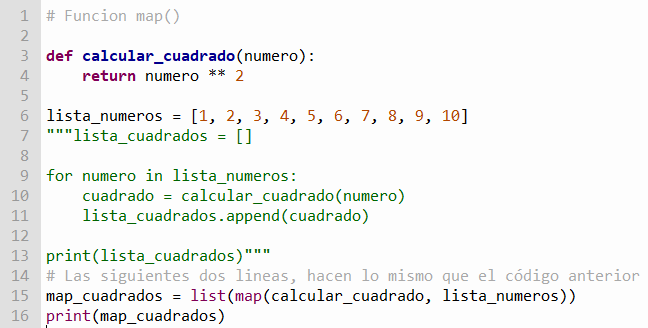
Sentencia else en el ciclo for

Esta sentencia, permite ejecutar el código, cuando este ha terminado de iterar sobre todos los elementos:



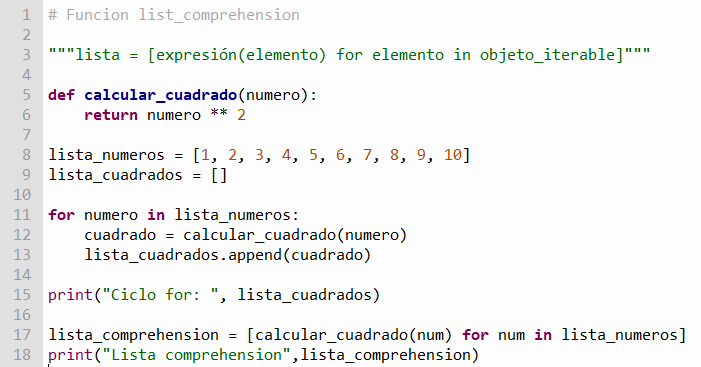
Función map()

La función map(), retorna un objeto de tipo map que entrega el resultado de acabar de aplicar una función a cada uno de los elementos de la lista que recibe, map es un iterador, esta función nos permite reemplazar lo que podemos hacer con un ciclo para aplicar una función a todos los elementos de una lista.

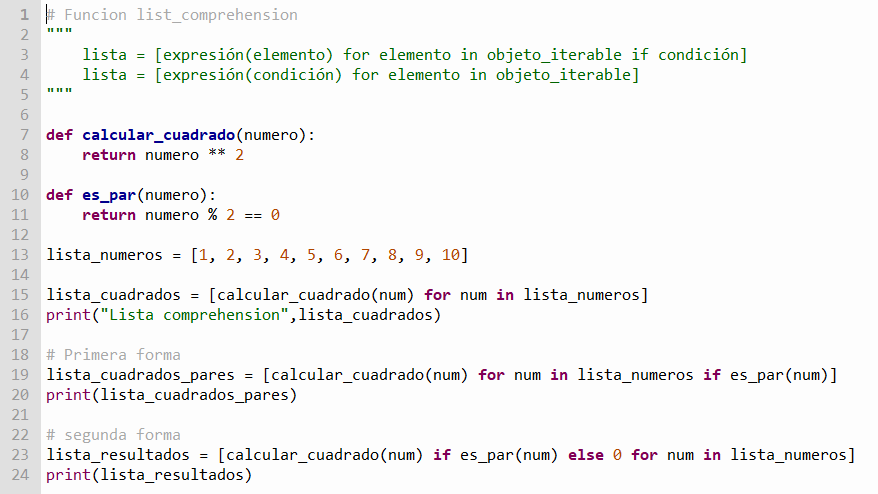


Función list\_comprehension

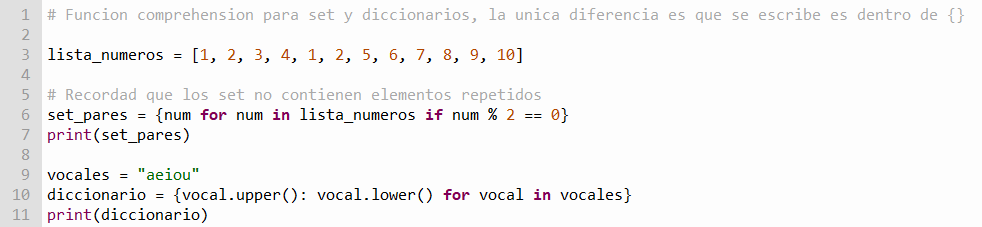
Es una manera de construir objetos iterables, usando una sola línea de código, este método es más elegante que escribir un ciclo for para crear una lista ya que estamos simplificando el código.



Lis\_comprehension, también nos permite aplicar condiciones para que las expresiones sean o no calculadas en cada iteración del objeto iterable.

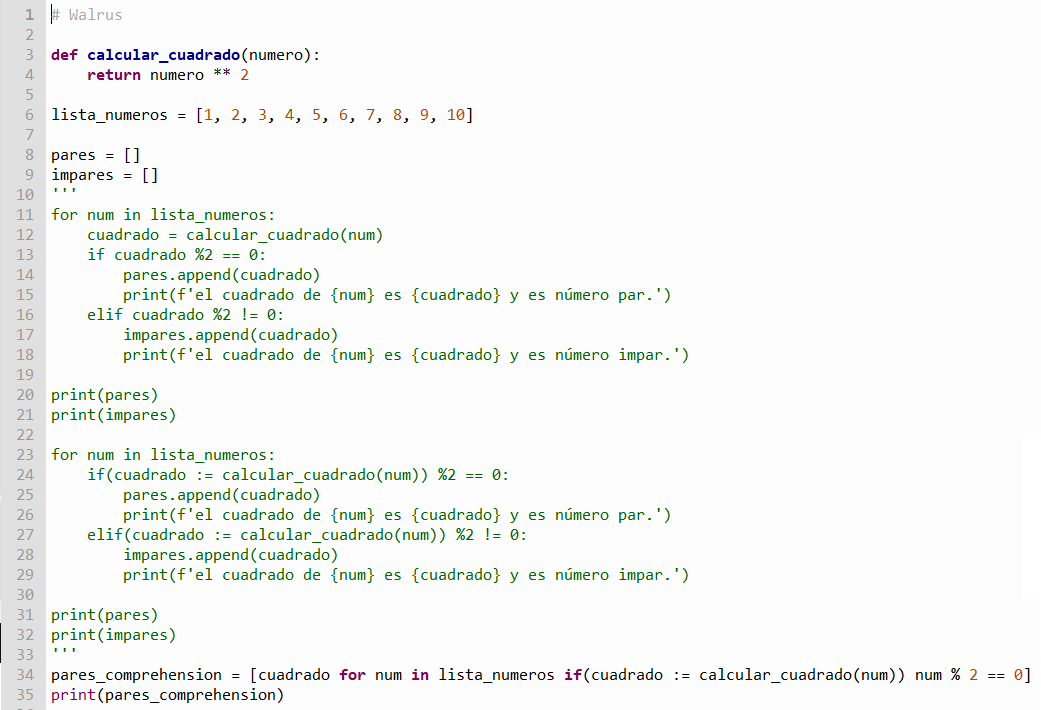


También puede ser usado en la creación de set o conjuntos y diccionarios.

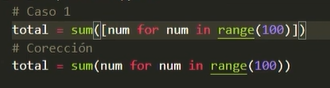


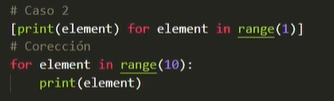
Operador de walrus

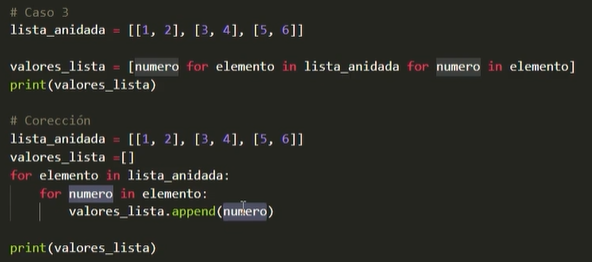
Es un operador que permite asignar un valor a una variable mientras se evalúa como expresión, se escribe como :=, fue introducido después de Python 3.8, es una expresión de asignación.



List\_comprehension, no se debe usar en expresiones largas, ya que da lugar a que la expresión sea difícil de leer y no se puede entender so usar:

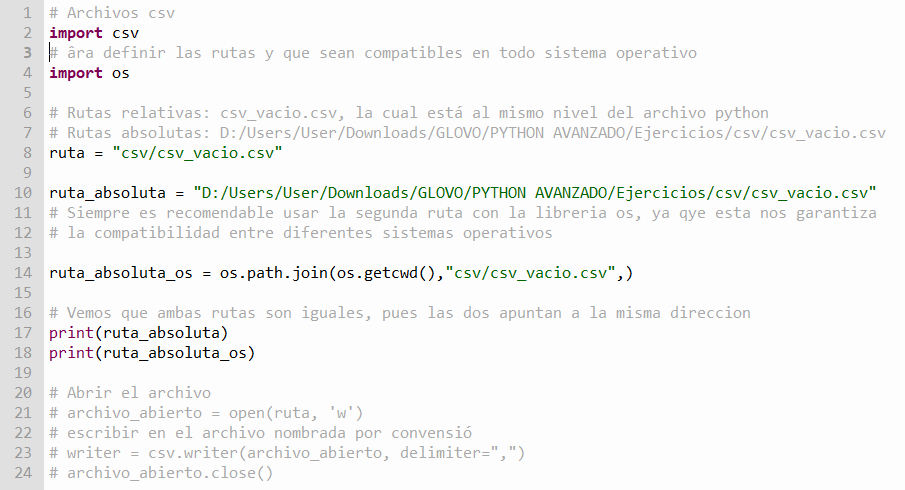
Cuando no necesitamos realmente el uso de una lista para obtener un resultado, ya que la creación de la lista está ocupando espacio en la memoria para algo que no se usará de nuevo.

Para imprimir variables de objetos iterables.

Evitar el uso de ciclos anidados, ejemplo, un for dentro de mas for o if etc.

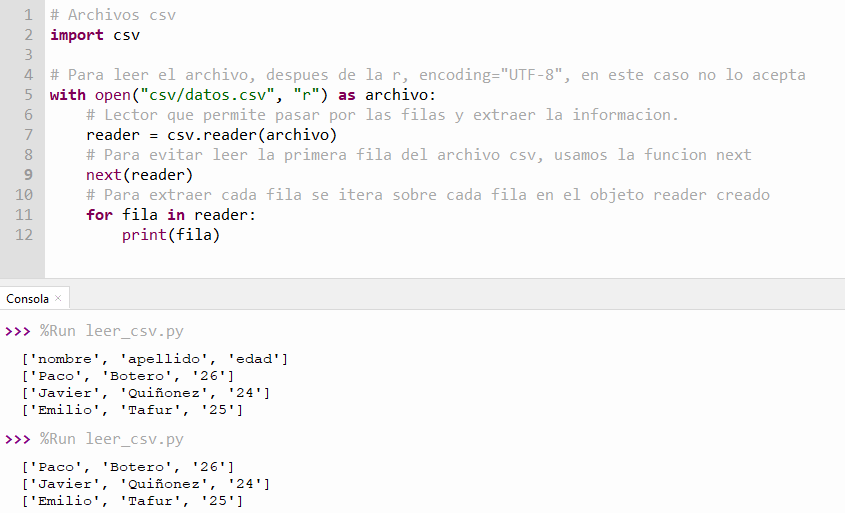
CSV

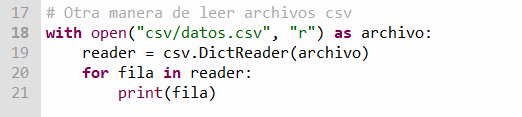
Es un formato que nos permite guardar datos de manera sencilla, los datos almacenados en estos archivos representan tablas, en las cuales cada valor separado por una coma, pertenece a una columna y cada línea del archivo es una fila.



Ejemplo:

Abrir Archivos csv





Logging:

Es el proceso para obtener los logs de los programas, son registros que nos ayudan a entender los comportamientos de un programa, al definir un log, indicamos la ocurrencia de algún evento, estos logs o registros, permiten detectar cuando se está produciendo un error, Python, tiene una librería propia para hacer este proceso, dicha librería es built-in, es usada por otras librerías y nos permite entender que pasa cuando las usamos, niveles de seguridad que permiten indicar si los registros que estamos obteniendo al correr el programa son informátivos, de desarrollo, advertencia o errores.

Existen 5 niveles de seguridad:

Debug: es una información detallada, usualmente se usan mientras se están desarrollando en partes del código que debemos revisar.

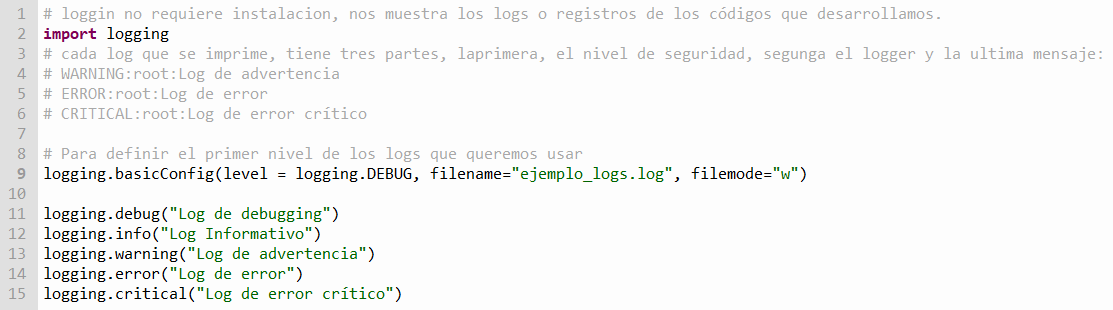
Info: es el reporte de eventos de los cuales se requiere información.

Warnin: es un reporte de algo inesperado que ha sucedido o va a suceder en el código, sin embargo, eso ni genera ningún error.

Error: es un reporte de algo que no se pudo ejecutar porque ocurrió un error en el código.

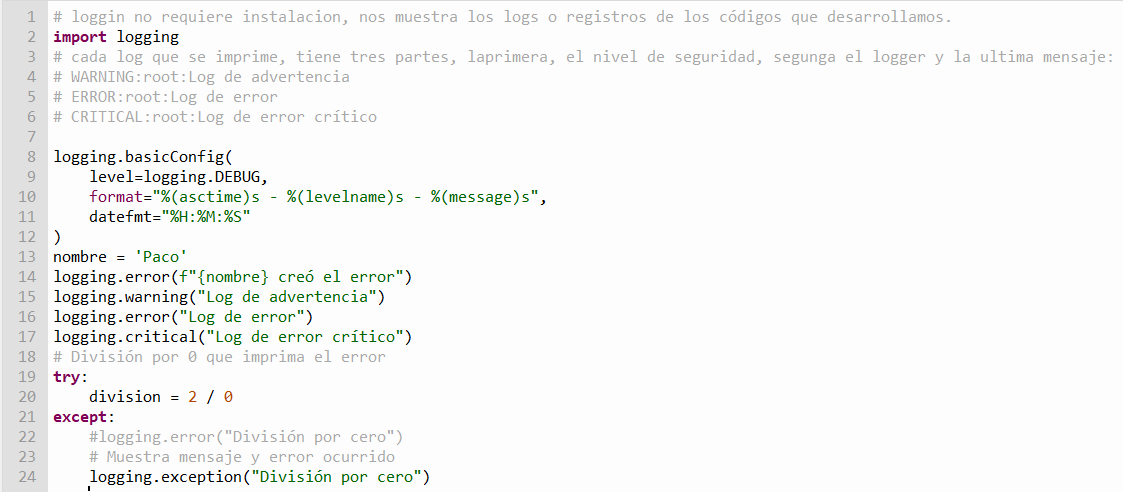
Critical: es un reporte de un error grave, puede indicar que el programa se detendrá por este error.

Por defecto, la librería loggin, genera warnings de primer nivel, para usar debug e info, es necesario definirlo en el código.



Cuando creamos diferentes tipos de logs, lo que hacemos es invocar la función de cada uno de los niveles y pasar como argumento el mensaje o texto a mostrar.

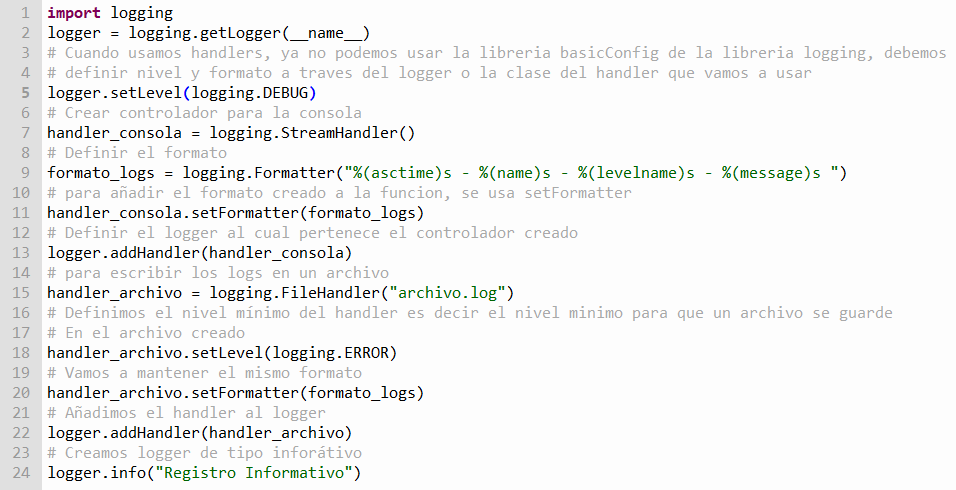
La manera en que se muestran los logs la podemos cambiar desde la función basicConfig



Log handlers

Controlador de logs o de registros, en Python el log handler es un elemento que nos permite configurar y personalizar los loggers, cada tipo de handler permite enviar los registros de diferente manera, String Handler para consola y file handler para archivos, también existe el smtp handler que permite envío de logs a través de email, estos se componen de dos elementos:

Un formato que permite organizar el mensaje para dar contexto al registro y el nivel mínimo que nos indica la severidad del log o del registro.



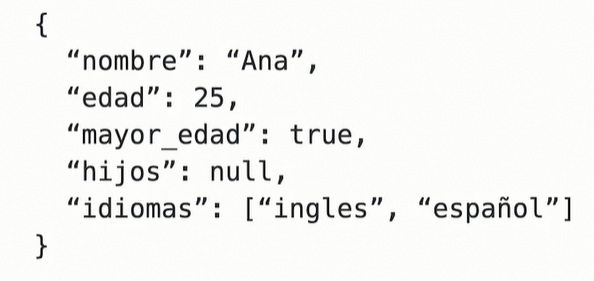
JSON

Es el acrónimo de javascript Object Notation, es un formato de texto simple que permite el intercambio de datos entre servicios y aplicaciones, no depende de un lenguaje de programación, entre las ventajas encontradas con el uso de este formato:

* Sencillo.
* Legible y fácil de entender, tanto para el usuario como para los servicios que lo usan como comunicación.
* No requiere ser codificado para ser enviado o recibido entre servicios.
* Su contenido representa uno o varios objetos

Podemos manejar diferentes tipos de datos.

* Cadenas de texto.
* Datos numéricos, enteros y decimales
* Booleanos, valores de verdad False y True
* Datos nulos
* Arreglos o listas
* Objetos

Estructura:

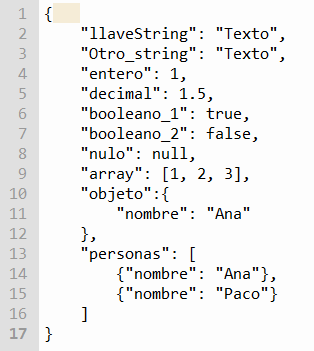
Usado para:

* Envío de datos entre servicios como aplicaciones, servicios web, servidores entre otros.
* Almacenar datos de manera temporal.
* Almacenar datos persistentes.

Formato de texto plano muy similar a los diccionarios en pyothon y lo que se conoce en otros lenguajes de programación como hashmaps, se puede guardar en archivos de texto plano con extensión .json, los elementos que creamos en un json, son conocidos como objeto, cada objeto definido puede tener ninguno, uno o varios atributos.

camelCase: “llaveString”

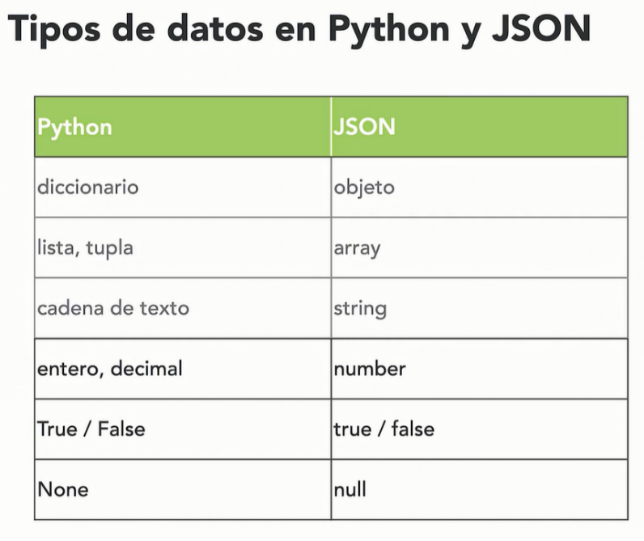
snakeCase: “llave\_string”

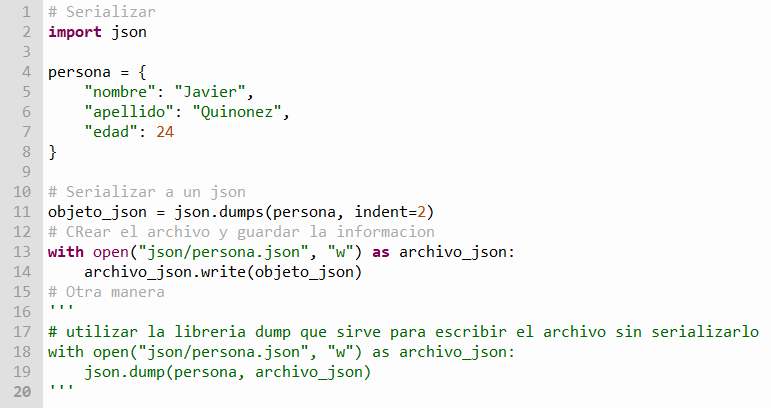


Serializar: Consiste en codificar el JSON, significa que los datos que contiene este formato, los transformamos en bites ya que pueden ser transformados o enviados a través de un servicio.

En Python, el proceso de serializar se hace con la librería nativa json.dumps().

Deserializar: es el proceso contrario y consiste en decodificar los datos, se reciben los datos y son transformados al lenguale en el cual estamos trabajando, en Python, también se hace este proceso con la librería nativa json.loads().







Peticiones HTTP

Protocolo de transferencia de datos que permite la comunicación entre servicios, funciona con el modelo cliente servidor, en el cual se crea una conexión y el cliente hace una petición al servidor, esperando su respuesta, el cliente es quien realiza la solicitud de información, mientras que el servidor es quien la provee.

Características:

* Simple: diseñado para ser interpretado fácilmente por las personas.
* Extensible: permite que se pueda desarrollar sobre el para agregar nuevas funcionalidades.
* No guarda estados: no se guardan datos entre dos o más peticiones,
* Guarda sesiones: almacena datos respecto a una sesión iniciada.

Una petición HTTP, es un mensaje que se envía entre los servicios que establecen una conexión para comunicarse, las peticiones inician acciones en el servidor, esto permite el movimiento de información entre los servicios, estas peticiones están compuestas por tres elementos principales:

* Línea de inicio: es donde se define el tipo de petición que se va a hacer.
* Cabecera o headers: metadatos que se envían para proveer información sobre la petición.
* Cuerpo: contienen los datos que serán enviados a través de la petición si esta implica un cambio en el servidor.

Las respuestas de una petición http, también tienen 3 elementos:

Línea de estado: contiene el protocolo http usado, codigo de estado que indica si la petición falló o fue exitosa, Texto de estado: indica si es una petición de lo que ocurrio en la petición.

Cabecera o headers: son los datos que se envían para proveer la petición.

El cuerpo: que contiene información solicitada por la petición.

Métodos http, indican las acciones que se pueden realizar por el servicio con el cual se hace la conexión.

* GET: Permite la entrega de datos desde el servidor, hacia el cliente.
* POST: Permite el guardado de datos desde el cliente hacia el servidor.
* PUT: Permite modificar datos desde el cliente hacia el servidor.
* DELETE: Permite borrar datos en el servidor.

Cuando hacemos la petición http, esta nos entrega una respuesta la cual contiene un código de estado y cada código indica el estado de la petición.

Códigos:

* 1XX: respuestas informativas.
* 2XX: respuestas satisfactorias, las más usadas son 200 ok, es decir que tuvo éxito y 201 created que quiere decir que el recurso fue creado, lo usamos para peticiones post.
* 3XX: nos indican redirecciones.
* 4XX: son errores del cliente, es decir de quien hace la petición, son muy usados el 400 bad request que quiere decir que algo salió mal en una petición 401 unautorizad, que quiere decir que no está autenticado, por lo tanto, no se entrega información y 403 forbidem, es decir que no tiene los permisos necesarios.
* 5XX: son los errores del servidor, es decir de quien entrega la información, el más común es 500 internal server error, que indica que algo falló en la petición al servidor.

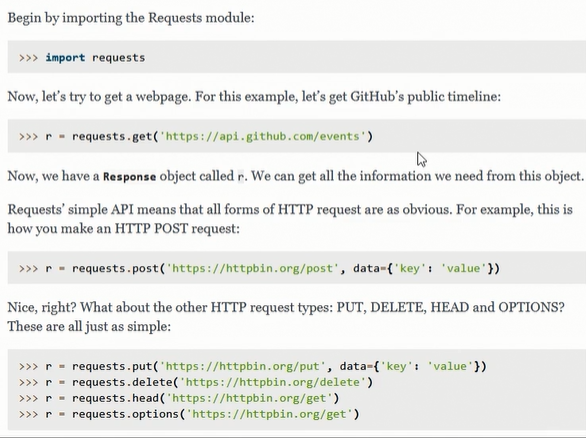
HTTPS

Corresponde a las siglas de HyperTexst Transfer Protocol Secure, es un protocolo basado en HTTP con una capa de seguridad, es decir la versión segura de HTTP.

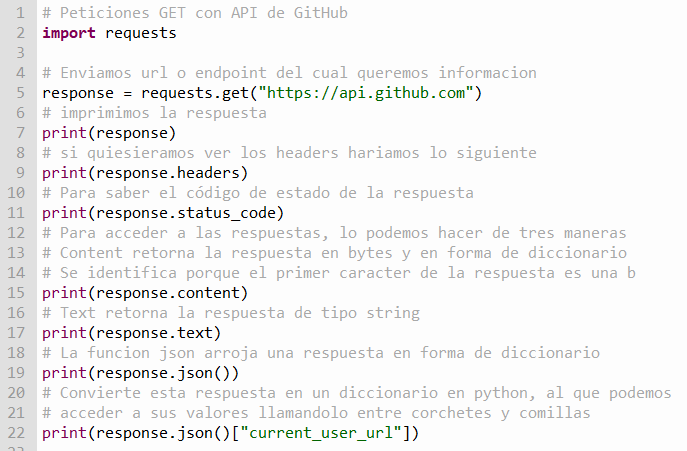
API

Corresponde a las siglas de interfaz de programación de aplicaciones, es un conjunto de funciones y protocolos que permiten la comunicación entre servicios usando reglas la comunicación con una API, se hace a través de HTTP usando los métodos o acciones mencionados anteriormente, las API y API REST, usan endpoints que permiten entregar la información o recibir las solicitudes hacia el servicio, el endpoint es una url que corresponde a la solicitud realizada y tiene asignado un método especifico que permite realizar una tarea.

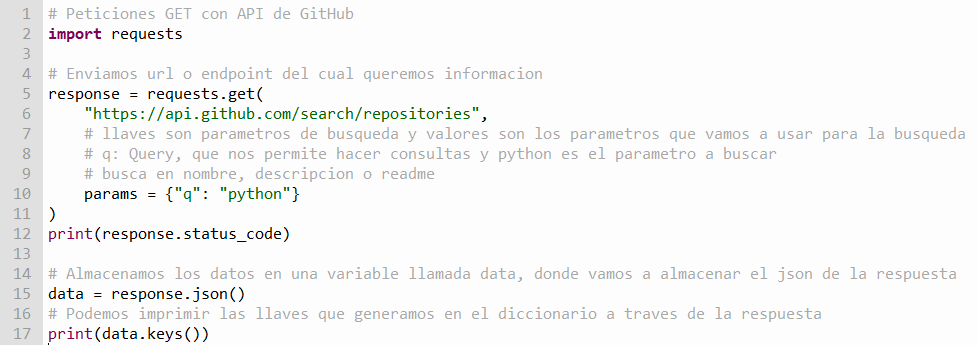
LIBRERÍA REQUEST

Es una librería de Python que permite hacer peticiones HTTP, esta librería la podemos encontrar en pypi.org, donde están las versiones disponibles, generalidades de su funcionamiento, la documentación y su repositorio.

Petición GET

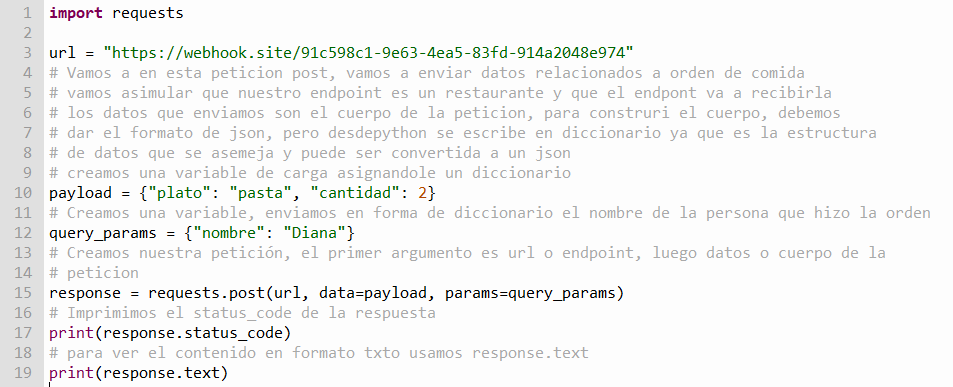
Una de las peticiones HTTP más usadas ya que permiten la entrega de información sin hacer cambio en el servidor al cual se le hace la solicitud, sirven para recibir información de un endpoint al cual estamos llamando y como usar query params para filtrar la petición, cada petición GET que se haga, dependerá de la API o servicio que se está consumiendo.

Endpoint con parámetros especiales:

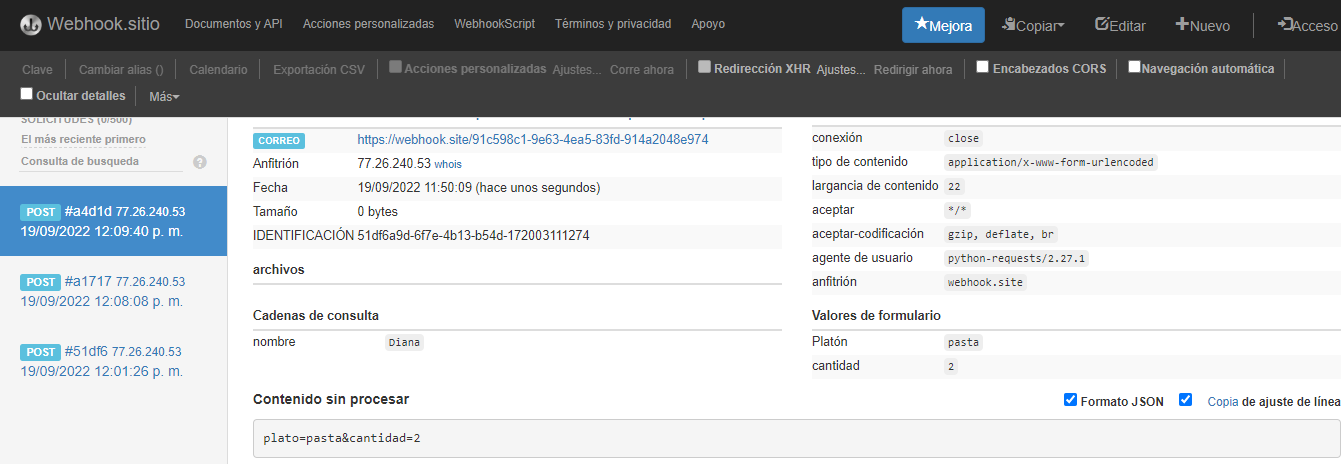


Petición POST

Es muy usada, ya que nos permite enviar información hacia un servicio, generalmente la información que se envía de esta manera, se almacena en una base de daos que está ligada al servidor al cual le hacemos la solicitud, en este caso usaremos una página llamada webhook.site, que nos genera una url hacia la cual podemos enviar peticiones sin hacer ningún cambio en servicios o servidores.

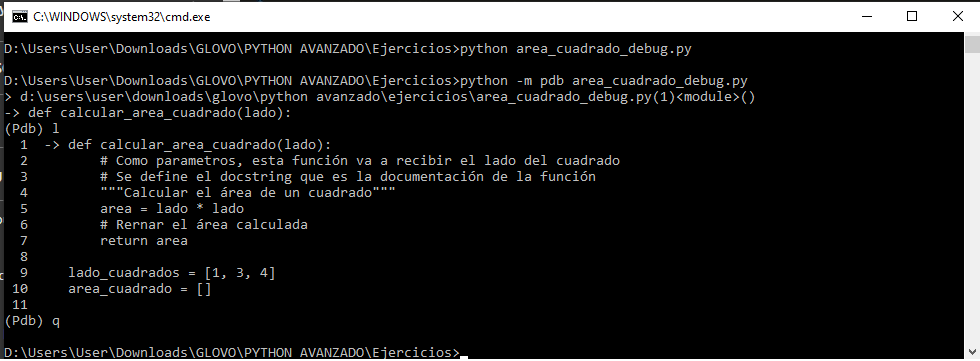


En la página de la url:



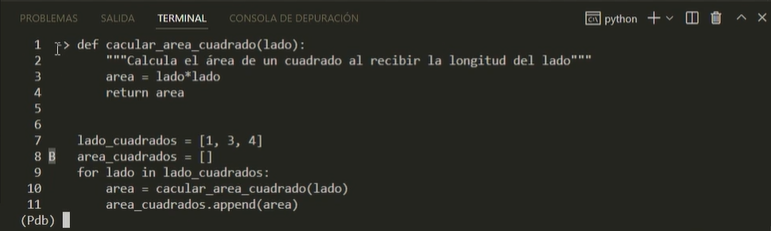
DEBUGGING

Es un error o una falla en un sistema o en un programa, el termino bug viene de la palabra bicho en inglés, este término se usa en temas de hardware y software, desde que Grace Murray Hopper, una pionera de la computación reportó que una falla en el hardware del computador estaba siendo causado por un bug, un bicho que estaba en uno de los componentes, el proceso debugging, en español depuración, es un proceso que permite identificar errores o fallas en el funcionamiento de un sistema informático, es un proceso manual, en el que se evalúa paso a paso el funcionamiento del código y se identifica dónde y porqué se genera el error, Python, cuenta con una librería que no requiere ser instalada para hacer debugging, o depuración del código, la librería pdb: Python debugger, esta librería, nos permite depurar el código a través de la línea de comandos, este proceso usualmente se hace mientras estamos en procesos de desarrollo.

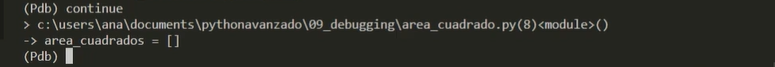


Break points:

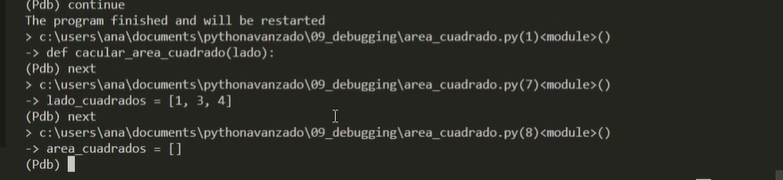
Son puntos o marcas en los que detenemos el código para saber y entender su funcionamiento.



El comando continue significa que la ejecución va hasta la línea del break point y se detiene.



Con el comando next, podemos ejecutar la siguiente línea de código sin necesidad de definir breakpoints.



Como inspeccionar una variable, es decir ver su contenido desde la terminal debugger sin necesidad de usar la función print.

