Estructura/organización del código

Tema 5. Apuntes

[**Definición de funciones 2**](#_6e8hnwcl7l0k)

[**Diseño modular 3**](#_aeazzeq9sfgf)

[**Parámetros 3**](#_4jfzurorsm2m)

[**Concepto de scope (ámbito) 4**](#_ytu495gfjq7l)

[**Variables locales y globales 4**](#_8kc7c4ua6kg4)

[**Argument packaging 5**](#_clzrxlxc6mzn)

[**Argumentos de clave y de posicionamiento 5**](#_eq6gd0dde830)

[**Documentación interna (importante) 6**](#_k95xwcnl8azv)

[**Mal uso de los comentarios 6**](#_5ontvi8rbfw6)

[**Buen uso de los comentarios 7**](#_2ouavkyfo01y)

[**Docstring 7**](#_cc4d9j6f8v1c)

[**Módulos 8**](#_ihm5wnxoc896)

[**Paquetes 8**](#_xq73tchnf2lr)

[**Estilo del código 8**](#_7yqbz5natd2q)

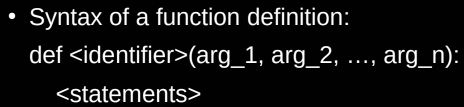
# Definición de funciones

**¿Qué es y función?**

Un pequeño trozo de código que lleva a cabo algo concreto. Se puede invocar/reutilizar/llamar desde cualquier lugar. Sirve para reducir nuestras líneas de código y estructurar el código.

Un programador puede crear sus propias funciones y reutilizarlas.

**Sintaxis:**



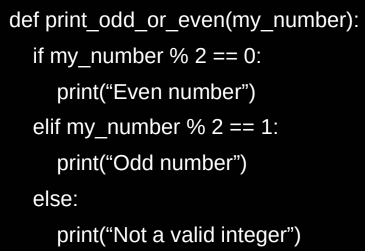
* Identificador: nombre que sigue las mismas reglas de nombramiento que una variable.
* Argumentos: como los inputs, son los datos que necesita la función para hacer los cálculos. Se pueden poner todos los que quieras. Recomendable no más de 5/6
* Statements: contenido de la función

**Importante: tamaño de las funciones**

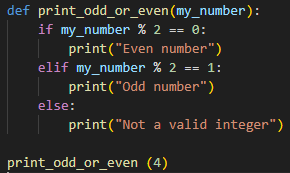
Hay que procurar que sean cortas, si son demasiado largas mata si objetivo. Por ejemplo no más de 30 líneas.

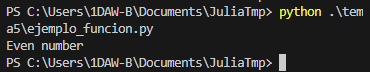
**Ejemplo:**

Es importante que los nombres de las funciones y de los parámetros sean descriptivos y claros.



Ejecución:





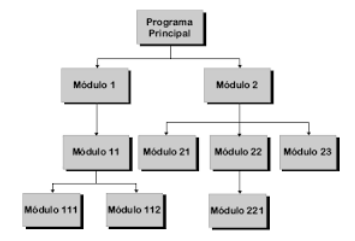
**Procedimientos**: funciones que no devuelven ningún valor. El ejemplo anterior es un procedimiento, ya que no usa return (aunque haga un print)

Para devolver un valor se usa la palabra reservada **return**. Se puede devolver cualquier valor.

# Diseño modular

Partir un programa/código en módulos o partes para que sea más manejable (más fácil de depurar, mantener o corregir).

Los módulos pueden ser funciones, librerías, clases/objetos…



# Parámetros

Siempre van a tener un identificador que puede llegar a coincidir con otras variables o funciones (no es para nada recomendable).

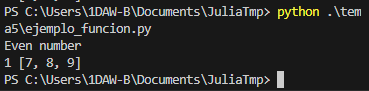
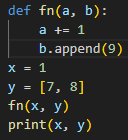
Se va a comportar siempre como una variable pero NO lo es. Se le puede asignar valor, meterlo en una expresión, pasarselo a otra función…

Hay que procurar no meter demasiados parámetros en una función (no más de 4/5 en general).

**¿Se puede modificar el contenido/valor?¿Qué pasa cuando termina la función si se modifica el contenido?**

Con el siguiente ejemplo se puede ver que depende. Se siguen ciertos criterios a la hora de modificar los valores.

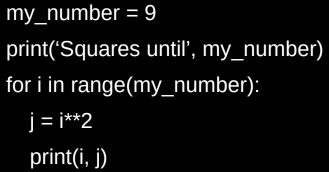
Los valores más complejos (arrays/listas) sí se cambian. (Investigar un poco cuales si y cuales no)



# Concepto de scope (ámbito)

Conjunto de instrucciones consecutivas que conservan el mismo nivel de tabulación, llamados **bloques**.

Este ejemplo tiene 2 bloques:



El **ámbito** de una variables es el bloque, función o módulo en el que es posible usarla

Un **bloque interno** es un bloque que se encuentra dentro de otro, con una tabulación. Para estos es posible acceder a las variables del bloque que lo contiene pero no al contrario.

Por eso las variables declaradas dentro de una función no se pueden usar fuera de esta.

# Variables locales y globales

Una **variable local** es declarada y usada dentro de un bloque, por lo que no es accesible desde cualquier parte.

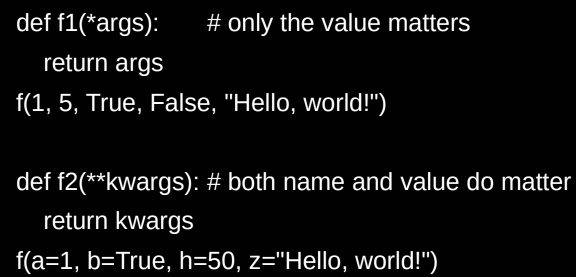
Una **variable global** es accesible desde cualquier parte del programa. Lo mejor es evitarlas, solo usarlas en casos muy concretos. Su valor puede cambiarse en cualquier parte y eso puede llevar a muchos problemas. Es importante no modificar su contenido.

# Argument packaging

Habrá ocasiones que hagamos funciones que admitan cualquier número de parámetros. Esto se consigue con el empaquetamiento de argumentos.

**Empaquetamiento de argumentos sencillo →** La función f1() lo que hace es aceptar un número de argumentos no definido, se puede usar con tantos como quiera el usuario. Importante: se usa un \*

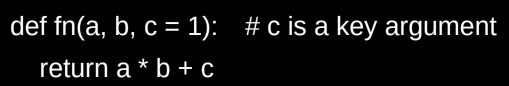
**Empaquetamiento de argumentos complejo(? →** En la función f2() también se aceptan el número de argumentos no definido, aquí es importante el nombre de los parámetros.



## Argumentos de clave y de posicionamiento

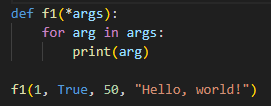
**Argumentos posicionales →** Son los que necesitan identificador.

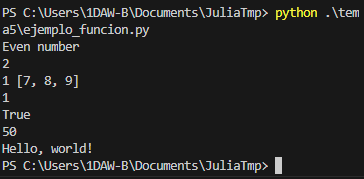
**Argumentos de clave →** Parámetros declarados con un valor por defecto.



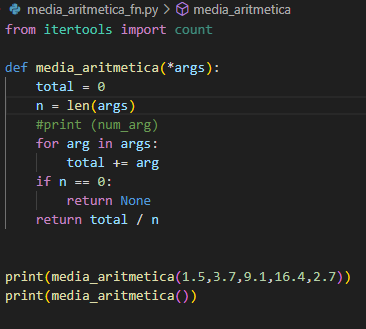
En ese ejemplo si se introduce fn(1,2,3) el resultado será 5 porque se le asigna a c el valor 3, pero si se ejecuta fn(1,2) el valor de c será 1 ya que es el valor que tendrá por defecto si no se introduce un valor nuevo.

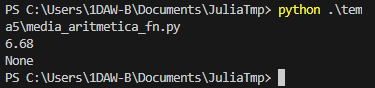
Ejemplo:





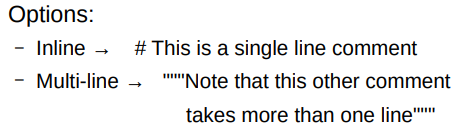
Ejercicio de clase:





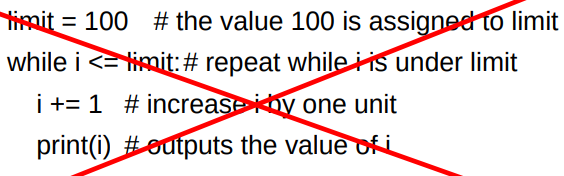
# Documentación interna (importante)

Los comentarios ayudan a entender el código



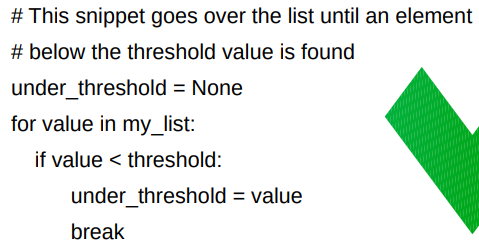
## Mal uso de los comentarios

Comentarios que no aclaran nada, solo dicen lo mismo que el código pero con palabras. No ayade ningún valor y solo ensucia el código



## Buen uso de los comentarios

Comentario general que describe el bloque/trozo de código que viene a continuación



## Docstring

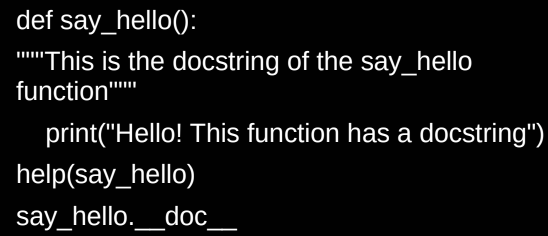
Docstring es un estándar de documentación de código

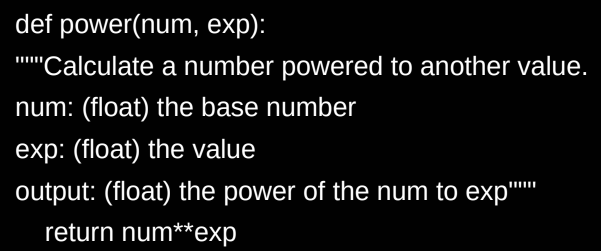
Los comentarios que asociamos a las funciones clases y objetos automáticamente documentan esa parte del código, estos se llaman comentarios **comentarios de prólogo o de declaración**. **Indican el propósito, los parámetros (tipos, detalles, etc) y la salida(return/output).**

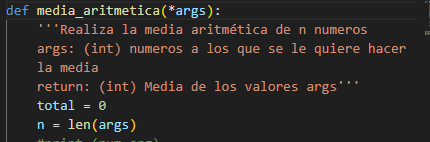
**¿Cómo se escriben los comentarios de prólogo?**

Escribiendo un comentario multilínea (“”” ”””) justo debajo de la declaración de la función/clase/objeto

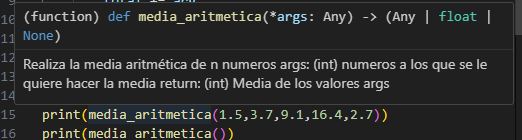
Ejemplos:





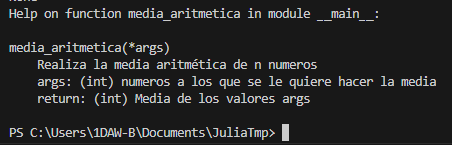


Ahora si mantenemos el ratón sobre el nombre de la función nos mostrará un cuadro informativo con la documentación de la función:



Si ponemos help(media\_aritmetica) al ejecutar saldrá la documentación de la función:





Se puede conseguir la documentación de las funciones con .\_doc\_, por ejemplo print.\_doc\_

# Módulos

Son librerías internas. Su propósito es es organizar el código de forma apropiada, así los programas salen limpios, más entendibles y además parte del código podrá ser reutilizado en algún momento.

Es posible llevar una importación selecia del contenido de un módulo

En Python sería creando un fichero .py y metiendo las funciones a reutilizar. Al usarlo en otro programa se hará un import del fichero y se podrán usar las funciones del módulo.

# Paquetes

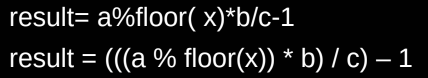
Agrupación de módulos.

Será un directorio con los ficheros de los módulos con las funciones. Contendrá un fichero llamado \_ \_init\_ \_.py (sin los espacios) que en un principio estará vacío.

Entre otras cosas sirven para crear bibliotecas más amplias.

# Estilo del código

La segunda opción es más comprensible, los espacios y los paréntesis separan y agrupan las operaciones y facilitan su comprensión al usuario.



Estandar a seguir: <https://pep8.org/> (leer un poco)