

## Python 3 y Django

### Introducción

En éste curso se van a ver temas como variables, ciclos, decisiones, funciones y también las muy importantes listas en Python así como OOP.

### **¿Que es Python?**

Es un lenguaje sencillo de programación de propósito general, o sea, se puede usar para programar casi cualquier cosa.

Tiene un intérprete de linea de comando.

### Descargando e instalando Python

[www.python.org](http://www.python.org) es el sitio de donde se descarga y la documentación.

Se va a la pestaña de descarga, corre en windows y linux.

Luego de descargado, en windows se instala el ejecutable.

Cuando se ejecuta, se instala con las variables por default

### Hola Mundo

```
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("Hola mundo")
Hola mundo
>>>
```

### Configurando IDE

En windows, en IDE puede configurarse la apariencia del python shell options → configure IDLE  
Ahí aparece un menú donde se puede editar la fuente, el tamaño de letra, los colores de la ventana, en fin, todo. (Yo como estoy en linux configuro la ventana del shell y ya está).

### PIP

Pip es un manejador de paquetes que utiliza python para instalar librerías y programas hechos en python.

Para ejecutarlo, hay que agregarlo a las variables de entorno en windows.

En linux: **sudo apt install python3-pip**.

Y para ejecutar pip: **pip3 install <nombre\_de\_lo\_que\_se\_quiera\_instalar>**

### Números y operaciones aritméticas en Python

En Python se comportan igual que en el mundo real: suma, resta, multiplicación, división, exponentiación y resto.

// redondea hacia abajo

\*\* potenciación

La precedencia es igual que en el mundo real.

### Variables

Las variables en el mundo real, se usan en aritmética en funciones por ejemplo vemos usualmente expresiones como  $x = 10$   $y=20$   $z = 14$ . En estos casos x y z todas estas son variables. La variable viene siendo la X y el valor es el valor que le asignamos a esta variable.

Se les llama variables porque su valor puede cambiar dependiendo de pues el contexto la velocidad de un vehículo puede cambiar el valor de x puede cambiar si le sumas el valor de y o el valor de z etc.

Pues bien en los lenguajes de programación y en Python en particular las variables son muy similares. Para declarar una variable en Python escribimos `x = 2` y lo que va a hacer Python es asignar a la variable `x` el valor de 2 utilizando el operador de igual.

Para conocer el valor de una variable lo único que tenemos que hacer es escribir `X` e inmediatamente se imprimirá el valor de dos.

Lo bueno de las variables es que ese valor se queda almacenado.

Vamos a utilizar otra variable `j` es Igual a `y`. Podemos ponerle el resultado de alguna operación. Por ejemplo ponemos seis potencia 3 que ya sabemos que era 216. Si ponemos `y`, nos imprime el valor 216.

Qué pasa si ponemos `X`, sigue siendo 2 porque se almacena su valor `x + 2` es igual a 4.

¿Por qué? Porque el valor de `X` era 2 si le sumas dos es igual a 4.

Pero qué sucede si ponemos de nuevo `X`, se imprimirá 2 por qué nosotros no cambiamos el valor de `x` simplemente le sumamos un 2 y vimos cuál sería ese valor pero no se lo asignamos.

Para asignar un nuevo valor a `X` lo que tienen que hacer es `x = x + 2`.

Ahora si se pone `X` se imprimirá 4.

De esta forma pueden cambiar los valores de sus variables `x = x * 2`.

Por otra parte hay tipos de datos en su mayoría de lo visto son enteros es decir números que no tienen punto decimal. Pero siempre puede haber un número que tenga por ejemplo 7.0, 9.5 etc.

Vamos a utilizar `z = 19/2`. A estos números que tienen un punto decimal se les llama números de punto flotante porque simplemente tienen un punto decimal.

También hay otros tipos de datos, por ejemplo las cadenas por ejemplo “Hola mundo”

Vamos a poner otra variable `a = “Hola mundo”`. A estas variables que son de texto se les conoce en Python y en muchos lenguajes de programación como strings o traducidas al español como cadenas.

Para recapitular un poco tenemos tipos de variables que son

- enteros o int
- float
- strings

Entre otras

Las variables en python no requieren declaración para usarla, con asignarle un valor ya se puede usar.

Otra cosa que se puede hacer es asignar variables a variables Esto simplemente se refiere a que podemos utilizar el valor de `x = x + y` para calcular una suma y asignarse a otra variable, en este caso estamos utilizando la variable `x` y sumándole el valor de `y`. Ésto podemos utilizar variables también para hacer sumas y restas y multiplicaciones y asignárselas a otras variables.

Lo último es que el tipo de las variables puede cambiar sin ningún aviso, por ejemplo en este caso nuestra variable x está almacenando un tipo de dato int, es decir está almacenando un número entero, pero también puede almacenar otra cosa sin ningún previo aviso aunque sea de un tipo diferente.

Por ejemplo podemos ponerle X = “sándwich” y python no se queja ni manda ningún error de tipo de dato ni nada por el estilo.

Esto pudo haber causado un error en otros lenguajes de programación pero a Python no le parece mal.

Aunque antes guardaba un número entero cuando le asignamos una cadena y simplemente lo acepta y ahora x es igual a “sándwich”. Por lo tanto Python no tiene ningún problema en cambiar de tipos de datos en las variables.

Esto puede ser muy bueno y muy malo a veces simplemente es bueno tener mucho cuidado cuando se asignan datos a una variable.

### **Strings**

Se emplean para infinidades de cosas.

Para generarlas, hay que poner las cadenas entre comillas, si no, da error.

Python no distingue entre comillas y comillas dobles, pero para poder poner f strings llevan comillas, conviene abrir y cerrar el string con un tipo de comillas y, el entrecomillado de la frase con otro tipo de comillas.

Ejemplo

```
frase = "Ella me dijo: 'Hola'" o frase = 'Ella me dijo: "Hola"'
```

Para python es indistinto, siempre que se siga esa regla

Otra forma es usar el carácter de escape “\”

En el ejemplo

```
frase = "Ella me dijo: \"Hola\""
```

La diagonal invertida, también se usa para caracteres especiales, por ejemplo, si se encuentra “\n”, lo que hace es poner todo lo que sigue en una nueva linea.

Ejemplo

```
>>>frase = "Ella me dijo: \n \"Hola\""  
>>>print (frase)  
Ella me dijo:  
"Hola"
```

También se puede usar el signo de suma para concatenar cadenas:

```
>>>nombre = "Jenifer"  
>>>apellido = "Barneche"  
>>> print (nombre+ " "+apellido)  
Jenifer Barneche
```

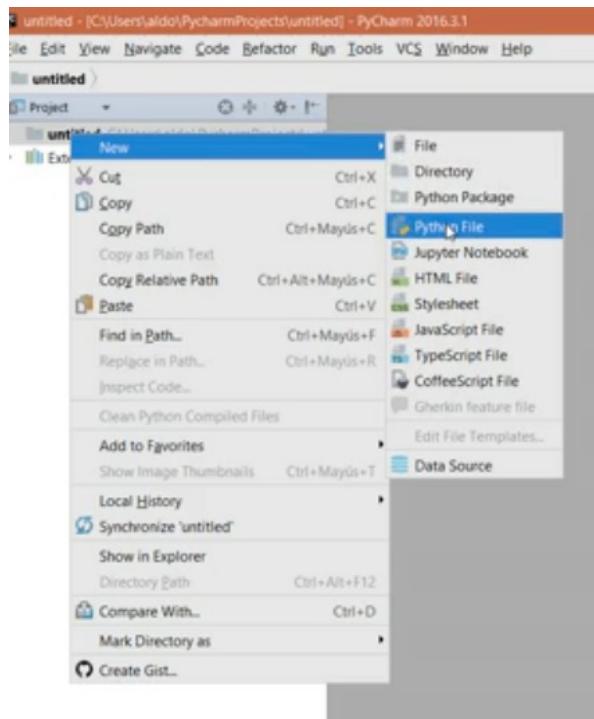
Otra cosa que se puede hacer es repetir el string un número determinado de veces

```
>>>print (nombre * 3)  
JeniferJeniferJenifer
```

## Pycharm

Es un IDE para escribir programas de Python

Luego de instalado, pide que crees una nueva carpeta donde va a guardar todos los proyectos. Una vez creado el proyecto, se paran sobre la carpeta, click derecho, menú New → Python File, se le da un nombre y ésto genera un nuevo archivo.

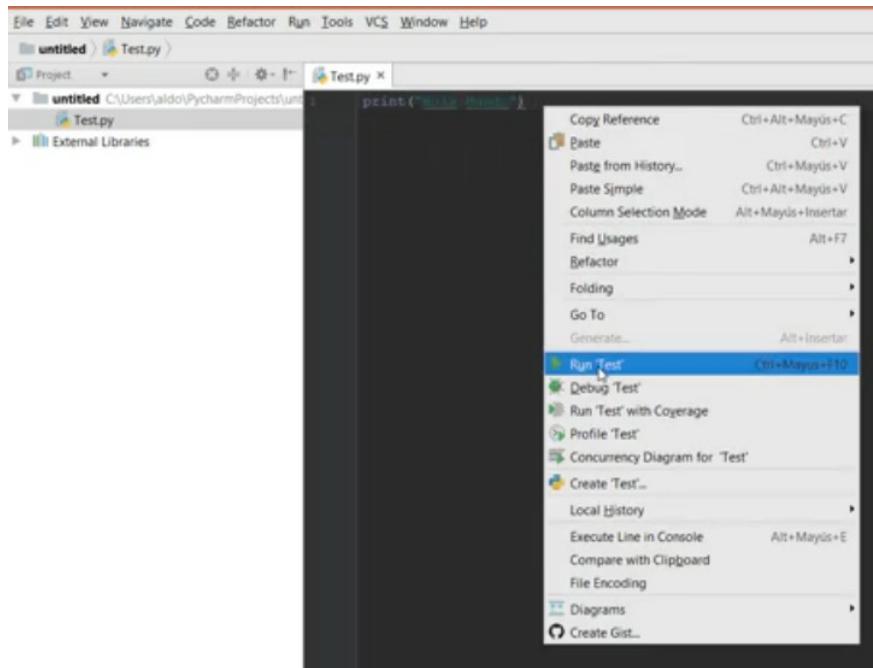


Los archivos de python tienen extensión .py

## Hola mundo

En la parte derecha que es donde está el código se escribe `print ("Hola mundo")`

La barra que podemos utilizar para correr nuestro código automáticamente está desactivada, para activarla en esta primera corrida damos clic derecho en nuestro archivo y elijen text aquí está la consola y como pueden ver escribe el famosísimo Hola mundo.



Posteriormente cuando Pycharm ya sabe que este es el archivo que va a correr simplemente da clic en Run test y correrá el archivo automáticamente.

### Cómo ejecutar código en una consola

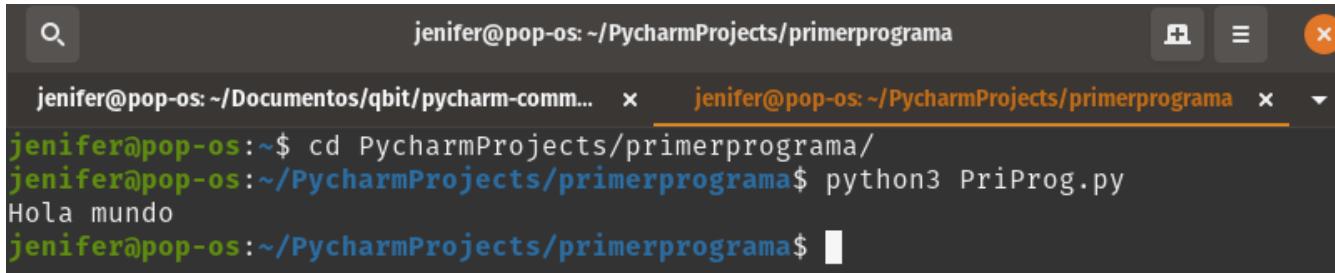
En windows se hace usando cmd, ese programa se busca en el menú desplegable de windows.  
En linux, simplemente se abre una terminal.

Una vez que la ventana está abierta y se tiene el archivo guardado de “Hola mundo”, lo que se escribe en la consola es python3 <nombre\_del\_archivo>.py

Notese que si en la consola, no se está parado sobre el directorio que contiene el archivo, hay que especificar la ruta

A screenshot of a terminal window titled 'jenifer@pop-os: ~'. It contains two tabs: 'jenifer@pop-os: ~/Documentos/qbit/pycharm-comm...' and 'jenifer@pop-os: ~'. The second tab is active. The user has run the command 'python3 PycharmProjects/primerprograma/PriProg.py'. The output of the script, 'Hola mundo', is displayed in green text in the terminal.

Otra forma, para no poner toda la ruta, es ir a donde está el archivo usando el comando cd:



A screenshot of a terminal window titled 'jenifer@pop-os: ~/PycharmProjects/primerprograma'. It shows two tabs open: one for 'jenifer@pop-os: ~/Documentos/qbit/pycharm-comm...' and another for 'jenifer@pop-os: ~/PycharmProjects/primerprograma'. The second tab is active. The command 'cd PycharmProjects/primerprograma/' is run, followed by 'python3 PriProg.py'. The output 'Hola mundo' is displayed. The prompt 'jenifer@pop-os: ~/PycharmProjects/primerprogramma\$' is shown at the bottom.

## Condicionales en Python

En general nos sirven para verificar si un valor o de una variable se cumple o no se cumple.

Esto puede ser una variable de tipo Boolean o de un valor numérico de tipo Float o INT o incluso pueden ser cadenas.

### **IF**

Supongamos que ustedes quieren saber si pueden entrar a un bar y beber en un bar.

La edad en la mayoría de los países para poder beber es 21.

Por lo tanto creamos una variable llamada edad=21.

Si tienes 21 años puedes entrar y beber en ese bar, para checkear que esa condición se cumpla lo que hacemos es utilizar la palabra reservada IF y quiere decir que en inglés "sí" y lo que hacemos después es testear el valor de una variable.

```
if edad == 21:  
    print ("Puedes entrar")
```

Las condiciones en Python se escriben utilizando la palabra reservada If.

Después escribimos lo que queremos testear y lo comparamos con algún signo de comparación en este caso utilizamos el doble igual.

Si la edad es igual a 21 imprime puedes entrar.

Qué pasa si ponemos que nuestra edad es que nuestra edad es 15 años, pues no se va a ejecutar este bloque de código porque nuestra condición es que tiene que ser igual a 21 para poder entrar.

Notar que las expresiones "=" e "==" son diferentes pues la primera asigna un valor a una variable y la segunda lo que hace es hacer comparaciones entre los valores de distinta variables o con valores.

El problema aquí es que deja afuera no solo a los menores de 21, sino que a los mayores de 21 también. Para arreglar ésto, se cambia la condición.

```
if edad >= 21:  
    print ("Puedes entrar")  
    print ("y tambien puedes beber")
```

Hay que prestar atención a que el bloque de código que se ejecutará es el que tiene un tabulador todo lo que tenga un Tab antes esa tecla Tab es de su teclado y es el estándar de Python genera un bloque de código todo lo que tenga un tab después de este IF se ejecutará solo si entra dentro del bloque IF.

Si olvidamos poner el tab o los cuatro espacios como también lo puede admitir Python

```
if edad >= 21:  
    print ("Puedes entrar")  
print ("y tambien puedes beber")
```

En ese caso va a imprimir “y también puedes beber” porque esta línea ya no es parte de este IF porque no le pusimos el TAB entonces no es parte de este bloque de código.

Python hace mucho hincapié en este aspecto porque con las tabulaciones y con los espacios se da cuenta de que bloques de código pertenecen a condición o a qué función etc etc etc..

## ELIF

Se usa para añadir más de una condición, por ejemplo:

Si tienes menos de 18 años no puedes ni siquiera entrar pero si tienes más de 18 años puedes entrar aunque todavía no puedas beber por qué no tienes veintiún años.

Qué es lo que podemos hacer en esta situación.

Esto lo hacemos con la palabra reservada Elif.

```
edad = 16  
if edad < 18:  
    print ("No puedes entrar")  
  
elif edad >= 21:  
    print ("Puedes entrar y tambien puedes beber")  
  
else:  
    print ("Puedes entrar pero no puedes beber")
```

La sentencia else cubre todas las demás posibles situaciones o condiciones lo que pueden hacer es agregar un bloque de else.

## Respecto a la indentación

La indentación no los espacios que especifican los bloques de código. Lo que hay que tener en cuenta es que, si bien python acepta tabuladores o espacios, una vez que se elige el número de tabs o de espacios que se van a usar, se debe usar ese tipo de indentación en todo el programa. Es decir, si yo a mi primer condición o función le doy como indent 2 espacios, el resto del código debe tener 2 espacios para que considere que la linea está indentada, no puedo usar primero espacios y después tabs.

## Funciones

Las funciones son un concepto que están prácticamente presentes en cualquier lenguaje de programación pues nos ayudan a encapsular nuestro código en pequeños bloques que podemos reutilizar en varias otras partes de nuestro código sin tener que volver a escribirlo una y otra vez.

Además esto permite que nuestro código sea mucho más mantenable. O sea que es muy difícil entender un código que simplemente tiene un montón de elif nombre apellido la legalidad y muchas y miles y tal vez cientos de miles de líneas de código es mucho más fácil entender lo que hace el código de otra persona.

Si todo está encapsulado en pequeñas funciones que tienen un propósito específico por ejemplo la función de ver si puedes entrar a un bar, ver si tienes el nombre adecuado para poder entrar a una fiesta etcétera.

Eso lo podemos encapsular en una función para que la otra persona sepa exactamente qué es lo que estamos haciendo.

Además, si queremos utilizar esa función en otra en otra parte de nuestro código pues también lo vamos a poder hacer simplemente utilizando el nombre.

Por ejemplo la función print para desplegar texto en nuestro en la consola de nuestro usuario. En este caso somos nosotros mismos.

Otra cosa importante es que, cuando se utiliza una función en el código hay que llamarla.

Para llamar a una función, lo que se tiene que hacer es escribir el nombre de la función y después los paréntesis y entre ellos, colocar los parámetros correspondientes en el caso que corresponda.

Algunas funciones tienen parámetros o argumentos que tenemos que poner pero algunas no pero, por ejemplo la función print necesita un parámetro que es la cadena que queremos imprimir.

Por lo tanto aquí podemos poner el parámetro que será una cadena como “Mi nombre es Aldo” por lo tanto nuestro código se va a imprimir “Mi nombre es Aldo”.

Sin embargo las funciones pueden no tener ningún parámetro puedes hacer una función que se llame algo así como “pedir\_pizza” que lo único que haga sea pedir una pizza.

No tienes por qué poner ningún parámetro.

Sin embargo también la puedes personalizar tal vez quieras tu pizza con peperoni con doble queso etcétera.

Para definir nuestras propias funciones en Python lo que tenemos que hacer es escribir la palabra “def”, después el nombre de la función y después , entre paréntesis, el o los parámetros que quieras ingresar a la función, y finalmente “:”

Hay que poner dos puntos y seguido para que Python sepa que todo lo que sigue es un bloque de código que se va a ejecutar en la función y al igual que con las condiciones tiene que haber un tab de espacio para que Python reconozca que todo lo que tenga espacio es parte del bloque de nuestra función.

Luego para llamarla lo que tenemos que hacer es poner abajo el nombre de la función y después los dos paréntesis. Si no se coloca nada dentro es porque no estamos pasando ningún parámetro.

Ejemplo:

```
def pedir_pizza():
    print ("Pedir pizza")

pedir_pizza()
```

¿Qué pasa si queremos definir una función que lleve parámetros?

Digamos que lo que queremos es una función que calcule si puedes entrar a un bar.

```
def chequear_entrada(edad):
    if edad < 18:
        print ("No puedes entrar")

    elif edad >= 21:
        print ("Puedes entrar y tambien puedes beber")

    else:
        print ("Puedes entrar pero no puedes beber")
edad = 18
edad_2 = 16
chequear_entrada (edad)
chequear_entrada (edad_2)
```

La ventaja de las funciones es que puedes llamarlas más de una vez en tu código.

Los guiones sencillos no están permitidos al definir nombres de variables ni de funciones, tienen que utilizar guiones bajos. Para que se entienda, como no acepta espacios tampoco, usualmente lo que se hace es separar los nombres con guión bajo. Hay otras convenciones que python también las entiende, pero son menos comunes. Por ejemplo, puede tomar como nombre válido ParaElisa, paraElisa, Para\_Elisa, pero no acepta “Para Elisa”.

## Objetos en Python

Se dice en general que todo en Python es un objeto.

Pensemos que es un objeto en el mundo real, pro ejemplo un coche un coche es un objeto y tiene una serie de atributos: el número de placa el color el número de puertas el tipo de transmisión si es manual automática y también tiene funciones: arrancar, manejar, tocar la bocina, encenderse, apagarse, etc.

Los objetos en el mundo de la programación son algo muy similar se encargan de representar una forma abstracta.

En esta implementación que representa algo que puede tener varios atributos y varias funciones.

Por ejemplo podemos tener un objeto que se llame jugador que tenga de atributos el número de vidas, el número de armas, el tipo de armas, y tiene funciones como caminar, correr, atacar, defenderse.

Para acceder a los métodos de un objeto se utiliza la notación de punto.

Ejemplo, el objeto lista, es de tipo list y la clase list tiene un método llamado append que lo que hace es agregar valores a la lista

```
>>>lista= []
```

```
>>> lista.append(4) ← append es un método de python y accedo con "."
>>> print (lista)
[4]
```

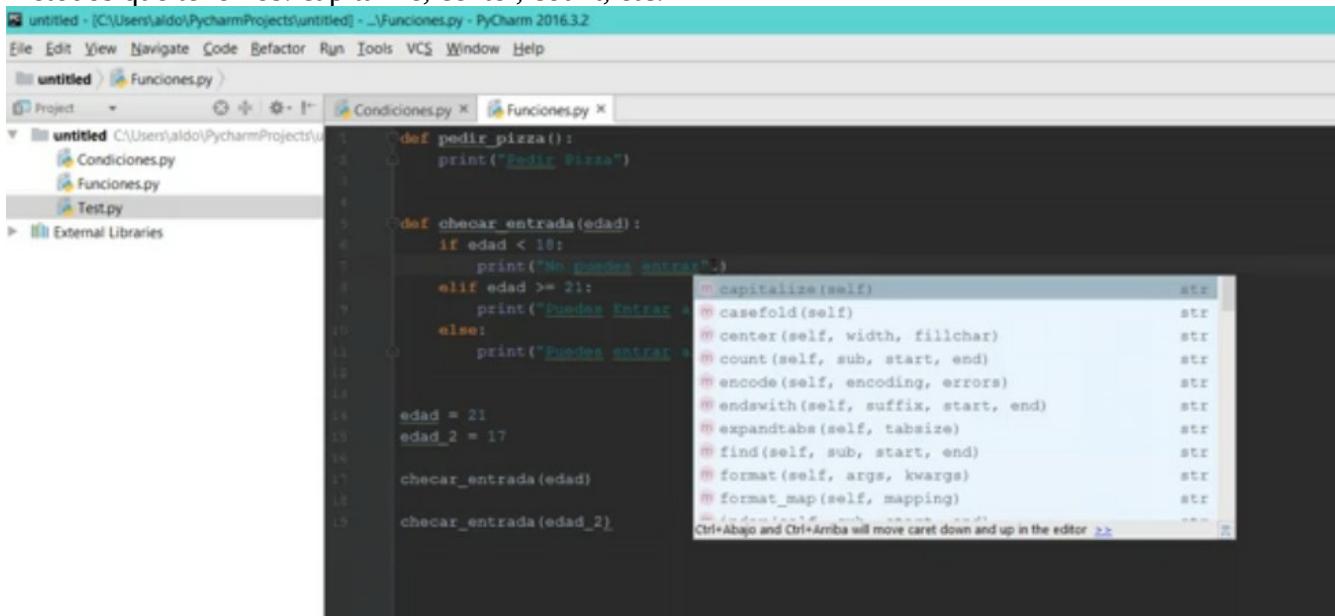
```
jenifer@pop-os:~$ python3
Python 3.7.3 (default, Apr 3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> lista=[ ]
>>> lista.append(4)
>>> print(lista)
[4]
>>>
```

Cabe mencionar que un método y una función son nombres muy similares.

Una función es cualquier función, valga la redundancia, que se define en nuestro código de Python. Sin embargo esta función pasa a llamarse método si está dentro de un objeto.

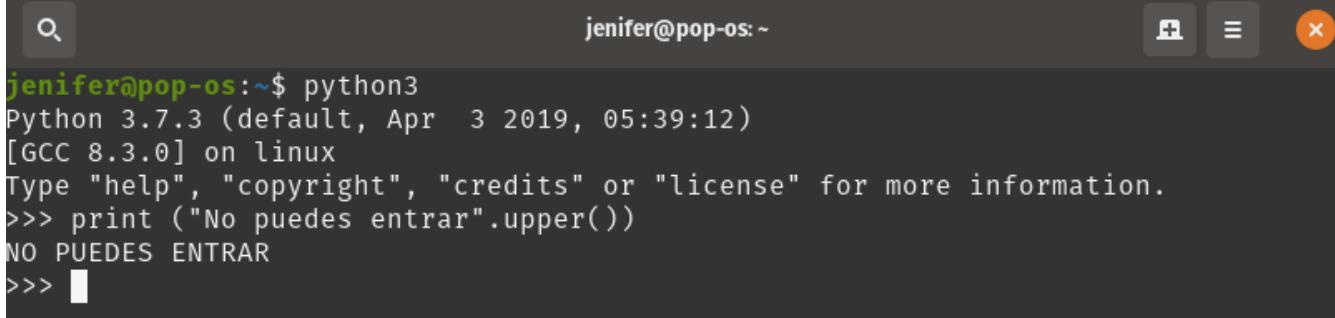
Por lo tanto un método es una función pero una función no es necesariamente un método.

Pues esto aplica para las cadenas: si queremos acceder a los métodos que tiene un objeto de la clase cadena ponemos punto y cómo se puede ver nuestro pycharm convenientemente nos va a decir los métodos que tenemos: capitalize, center, count, etc.



Como ejemplo vamos a utilizar uno que se llama upper y este método pertenece a la clase string que lo que hace es pasar toda la cadena que le damos a mayúscula.

## Ejemplo



A screenshot of a terminal window titled "jenifer@pop-os: ~". The window shows the following Python session:

```
jenifer@pop-os:~$ python3
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("No puedes entrar".upper())
NO PUEDES ENTRAR
>>> 
```

En este caso transforma “No puedes entrar” en mayúsculas.

En éste caso, accedimos a los métodos de la clase String con el operador “.” y utilizamos el método upper que no acepta ningún parámetro, siendo el objeto, la cadena de caracteres.

## Listas

En Python se usan para agrupar cosas. Las listas representan un conjunto de objetos así como en el mundo real tendremos por ejemplo, una bolsa de juguetes o una caja de zapatos, en Python las listas representan un agrupamiento de objetos. Son similares a lo que son los arreglos en otros lenguajes de programación.

La diferencia de los arreglos en otros lenguajes de programación las listas en Python es que pueden cambiar de tamaño como se les antoje y pueden guardar cualquier tipo de objetos y de datos y de todo lo que quieran sin ningún problema, a las listas no les importa cómo utilizamos listas en Python.

Pues bien para aprender sobre listas en Python basta con saber que las listas en Python se definen poniendo entre corchetes lo que va a llevar nuestra lista.

Por ejemplo si queremos hacer una lista de los números del 1 al 5 escribimos

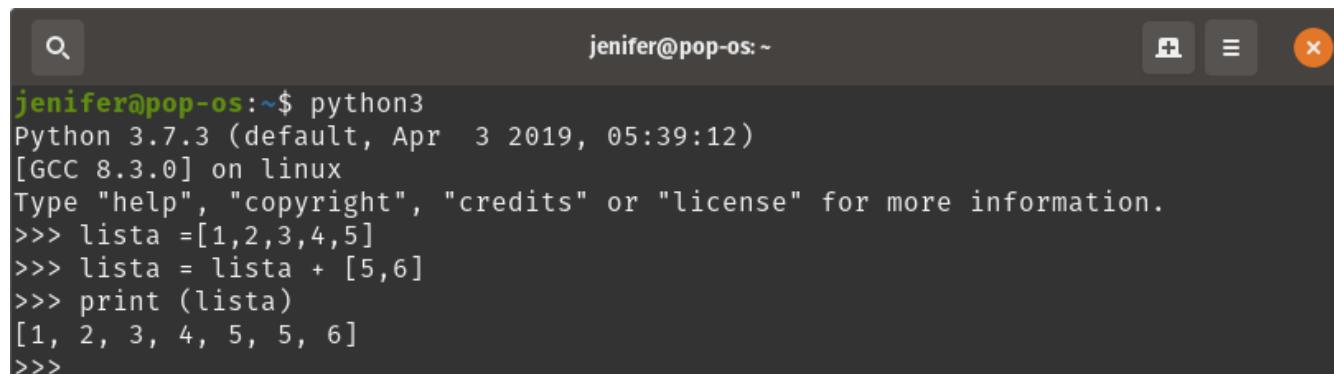
```
lista=[1,2,3,4,5]
```

Me refiero con esto a que podemos guardar toda esta lista en una sola variable que es igual a 1 2 3 4 5.

Ahora si queremos agregar un elemento o una serie de elementos de una lista a otra tendremos que escribir lista más y otra lista

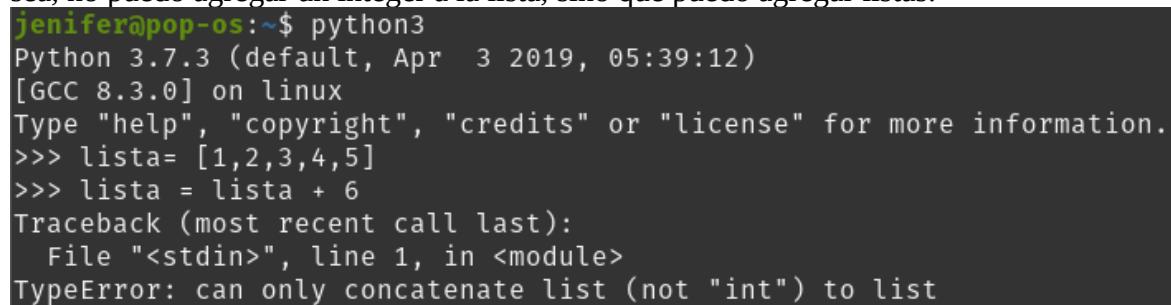
```
>>>lista + [5, 6]
```

Esto sería igual a una lista de 1 2 3 4 5 5 6 porque ya teníamos un 5 y después le agregamos 5 6.



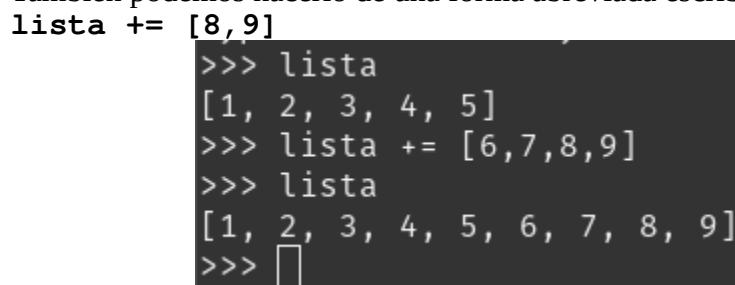
```
jenifer@pop-os:~$ python3
Python 3.7.3 (default, Apr 3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> lista =[1,2,3,4,5]
>>> lista = lista + [5,6]
>>> print (lista)
[1, 2, 3, 4, 5, 5, 6]
>>>
```

Con ésto hay que tener cuidado, porque las operaciones se tienen que hacer con datos del mismo tipo, o sea, no puedo agregar un integer a la lista, sino que puedo agregar listas.



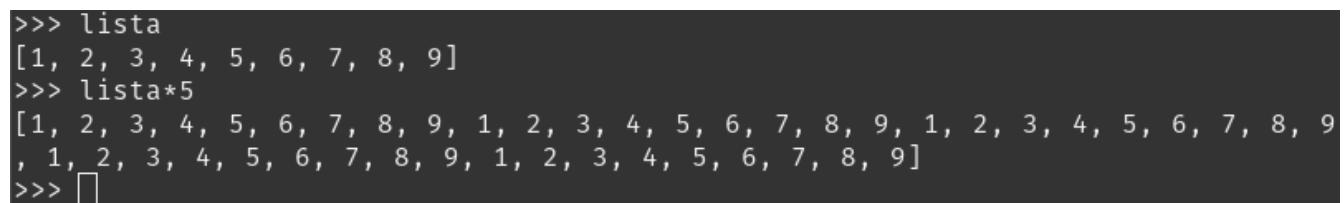
```
jenifer@pop-os:~$ python3
Python 3.7.3 (default, Apr 3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> lista= [1,2,3,4,5]
>>> lista = lista + 6
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "int") to list
```

También podemos hacerlo de una forma abreviada escribiendo por ejemplo



```
lista += [8,9]
>>> lista
[1, 2, 3, 4, 5]
>>> lista += [6,7,8,9]
>>> lista
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> 
```

En cuanto a operaciones, comparte las mismas operaciones que las cadenas, por ejemplo lista\*5 repite la lista 5 veces



```
>>> lista
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> lista*5
[1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9
, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> 
```

Sin embargo el signo de más con una con una lista no es la única forma de agregar elementos, también podemos utilizar un método de la clase lista.

### Append

Con ésto podemos agregar un elemento que queremos ponerle a nuestra lista al final de la misma.

```
>>>lista.append(10)
>>>lista
[1,2,3,4,5,6,7,8,9,10]
```

En realidad, el tipo de datos que hay dentro de la lista no importa, por lo cual, si, queremos hacer un append, o agregar, por ejemplo, una cadena, se puede hacer:

```
>>>lista.append("Juana")
>>>lista
[1,2,3,4,5,6,7,8,9,10, 'Juana']
```

Y también podemos poner otra lista como elemento de la lista.

```
>>> lista.append("Juana")
>>> lista
[1, 2, 3, 4, 5, 6, 7, 8, 9, 'Juana']
>>> lista.append([1,2,3])
>>> lista
[1, 2, 3, 4, 5, 6, 7, 8, 9, 'Juana', [1, 2, 3]]
```

### Extend

Se utiliza para agregarle a la lista más de un elemento.

La diferencia con append, es que si realizo:

```
>>>lista.extend([11,12])
>>>lista
[1,2,3,4,5,6,7,8,9, 'Juana', 11, 12]
```

y no [1,2,3,4,5,6,7,8,9, 'Juana', [11, 12]] que es lo que resulta al hacer append.

### Remove

Si se quiere remover un elemento particular de la lista, se usa el método remove.

```
>>>lista.remove(8)
>>>lista
[1,2,3,4,5,6,7,9, 'Juana', 11, 12]
```

Hay que tener en cuenta que ésta operación solo remueve la primer ocurrencia del objeto a borrar, por ejemplo si la lista fuera [1,2,3,4,5,6,7,9, 'Juana', 11, 12, 8], el resultado de `lista.remove(8)` será [1,2,3,4,5,6,7,9, 'Juana', 11, 12, 8]; y si no encuentra nada, da error.

### list

Se emplea para crear listas nuevas.

`lista=list([1,2,3,4,5,6,7,8,9])` ← los corchetes es porque list toma un único elemento como argumento

```
>>> lista= list("Juana de Arco")
>>> lista
['J', 'u', 'a', 'n', 'a', ' ', 'd', 'e', ' ', 'A', 'r', 'c', 'o']
```

O sea, toma los espacios como objetos también.

### Split y Join

Éstas son operaciones que se pueden hacer con cadenas y listas.

#### Split

Split lo que hace es poner toda una cadena de caracteres como elemento de una lista y esa es su diferencia con list es que no separa carácter a carácter, sino que, por ejemplo:

```
>>> "Juana de Arco".split()
['Juana', 'de', 'Arco']
```

En sí, lo que hace es armar una lista donde cada elemento es separado según donde está el espacio en la frase porque toma al carácter espacio como separador de campos.

Si se quiere usar otra cosa como delimitador, se pone `<objeto>.split("<caracter delimitador>")`

```
jenifer@pop-os:~$ python3
Python 3.7.3 (default, Apr 3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> comida="Hamburguesas,Pizza,Helado"
>>> comida
'Hamburguesas,Pizza,Helado'
>>> comida.split(",")
['Hamburguesas', 'Pizza', 'Helado']
>>>
```

#### Join

Sería algo así como el opuesto al split. Lo que hace es tomar una lista y juntarla en una sola cadena dependiendo del separador que le hayan puesto.

Join no es un método de la clase lista sino de la clase cadena, por lo tanto:

```
>>> lista_comida=['Hamburguesas', 'Pizza', 'Helado']
>>> ", ".join(lista_comida)
'Hamburguesas, Pizza, Helado'
>>>
```

La sintaxis es, primero va el campo delimitador, luego “join” y entre paréntesis la lista.

También se puede combinar con otras operaciones de cadena

```
>>> "Mi comida favorita es: "+", ".join(lista_comida)
'Mi comida favorita es: Hamburguesas, Pizza, Helado'
>>>
```

### Índices

Los índices son números que indican la posición de cada elemento dentro de una lista o una cadena. A diferencia de lo que se hace en la realidad, Python comienza a contar las posiciones desde 0.

por ejemplo:

```
>>> alfabeto = "abcdefghijklmnñopqrstuvwxyz"
```

```
>>> lista_alfabeto=list(alfabeto)
>>> lista_alfabeto
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
'n', 'ñ', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
>>>lista_alfabeto[0]
'a'
```

El primer elemento en la lista es A.

En general, para saber qué elemento está en una posición determinada, se pone

**<nombre\_de\_la\_lista>[<posición>]**

ejemplo:

```
>>>lista_alfabeto[10]
'k'
```

### Método index

Por otra parte, para saber qué índice tiene un elemento en la lista, se usa el método index.

```
>>>lista_alfabeto.index('a')
0
```

Al igual que otras funciones, sólo devuelve el índice de la primera ocurrencia y si no está, da error.

En el caso de cadenas, si se busca una subcadena y la encuentra, devuelve la posición donde se encuentra el primer elemento de la subcadena.

```
>>> alfabeto.index("bc")
1
```

### Índices negativos

A diferencia de otros lenguajes de programación, Python maneja índices negativos, contando a partir de -1 desde el último lugar de la lista hasta el primero

```
>>>lista_alfabeto[-1]
'z'
>>>lista_alfabeto[-2]
'y'
```

### Del

Del es una palabra reservada para borrar variables enteras o elementos de listas.

Eso si...una vez ejecutado ésto, no hay forma de recuperar lo que se borró, con lo cual, hay que tener cuidado.

```
>>> del <nombre de la variable>
```

Una vez borrada la variable, si se la quiere invocar de nuevo da error diciendo que no está declarada

```
jenifer@pop-os:~$ python3
Python 3.7.3 (default, Apr 3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> vocales="aeiou"
>>> lista_vocales=list(vocales)
>>> vocales
'aeiou'
>>> lista_vocales
['a', 'e', 'i', 'o', 'u']
>>> del vocales
>>> vocales
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'vocales' is not defined
```

Para borrar elementos de una lista, se usan los índices.

>>> **del <nombre\_de\_la\_lista>[<índice>]**  
del toma tanto índices negativos como positivos.

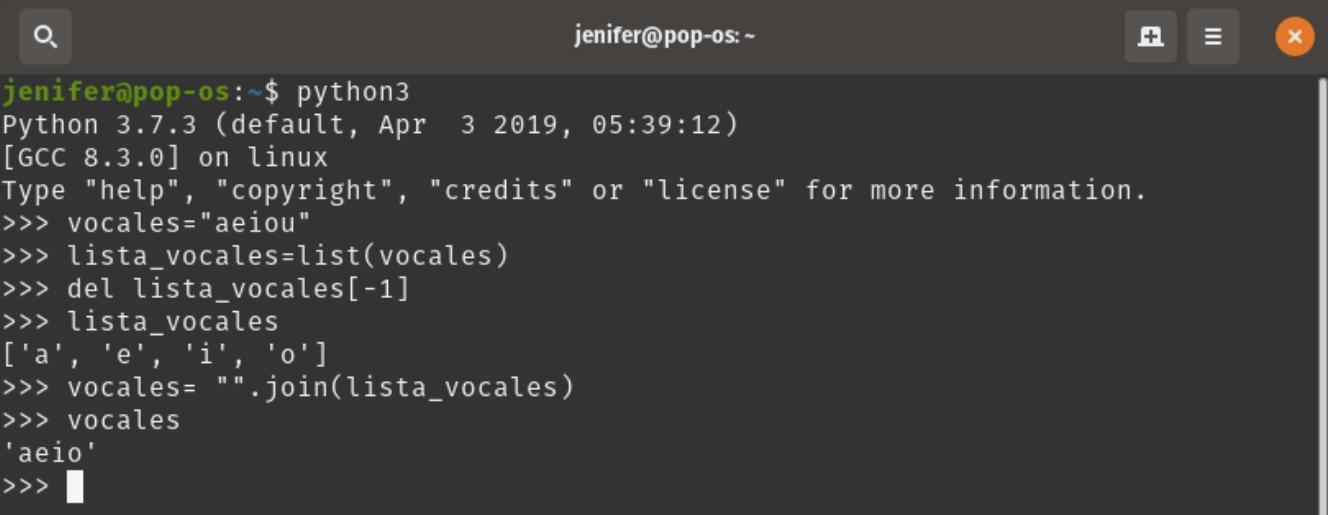
```
>>> del lista_vocales[0]
>>> lista_vocales
['e', 'i', 'o', 'u']
>>> del lista_vocales[-1]
>>> lista_vocales
['e', 'i', 'o']
>>>
```

Ésto no funciona para strings porque son de tipo immutable

Ejemplo:

```
jenifer@pop-os:~$ python3
Python 3.7.3 (default, Apr 3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> vocales="aeiou"
>>> vocales
'aeiou'
>>> del vocales[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object doesn't support item deletion
>>> 
```

Para alterar un string, lo que se hace es convertirlo en lista, alterar la lista y después reconvertirlo a string:



A screenshot of a terminal window titled "jenifer@pop-os: ~". The window shows a session of the Python 3 interpreter. The user has defined a string "vocales" containing the vowels "aeiou", converted it into a list, removed the last element ('o'), joined the remaining elements into a new string, and then printed the result "aeio". The terminal interface includes a search icon, a user icon, and standard window control buttons.

```
jenifer@pop-os:~$ python3
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> vocales="aeiou"
>>> lista_vocales=list(vocales)
>>> del lista_vocales[-1]
>>> lista_vocales
['a', 'e', 'i', 'o']
>>> vocales= "".join(lista_vocales)
>>> vocales
'aeio'
>>> 
```

## In

Es otra palabra reservada en Python y se utiliza para nos a verificar si un algo está dentro de otro algo, por ejemplo si un elemento está o no dentro de una lista. El valor que retorna ésta palabra es un boolean (verdadero o falso) y por ende, se puede usar dentro de comandos como if.

Por ejemplo:

```
>>>vocales="aeiou"
>>>lista_vocales= list(vocales)
>>>"a" in vocales
True
>>>"z" in vocales
False
>>>if "a" in vocales:
    print ("está en vocales") ← para hacer ésto en IDE es con ctrl+j y hay que
                                indentar porque si no, da error
>>>if "z" not in vocales:
    print ("no está en vocales")
```

## Ciclos

Se emplean para ejecutar código de forma cíclica, o sea nos ayuda a repetir un bloque de código un cierto número de veces.

En Python se usan 2 tipos de ciclos: For y While. While ejecuta un bloque de código mientras la condición que determinemos sea cierta mientras que for, recorre elementos dentro de una colección de cosas por ejemplo un rango de números, los caracteres de una cadena, o los elementos de una lista, etc, y se ejecuta para todos los elementos de esa colección.

## While

Éstos repiten los ciclos dada una condición que de True, si da false en algún momento en la ejecución del código, en la proxima interacción, no se ejecuta.

Ejemplo:

```
>>>manzanas =10
>>> while manzanas > 0:
    manzanas -=1
```

```
print("me estoy comiendo una manzana, me quedan"+str(manzana)+  
"manzanas")
```

Ésto va a ejecutar el código 10 veces, porque en un momento, la variable manzanas va a tener un valor de 0, y como manzanas > 0 no se cumple, no va a entrar al while.

El str(manzana) es porque manzana es un integer y para concatenarlo, necesito convertir la variable a tipo string para poder concatenarla

```
>>> manzanas =10  
>>> while manzanas > 0:  
...     manzanas -=1  
...     print("Me estoy comiendo 1 manzana, me quedan "+ str(manzanas)+" manzanas"  
)  
...  
Me estoy comiendo 1 manzana, me quedan 9 manzanas  
Me estoy comiendo 1 manzana, me quedan 8 manzanas  
Me estoy comiendo 1 manzana, me quedan 7 manzanas  
Me estoy comiendo 1 manzana, me quedan 6 manzanas  
Me estoy comiendo 1 manzana, me quedan 5 manzanas  
Me estoy comiendo 1 manzana, me quedan 4 manzanas  
Me estoy comiendo 1 manzana, me quedan 3 manzanas  
Me estoy comiendo 1 manzana, me quedan 2 manzanas  
Me estoy comiendo 1 manzana, me quedan 1 manzanas  
Me estoy comiendo 1 manzana, me quedan 0 manzanas  
>>> █
```

**Tip:**

Al igual que los ifs, while acepta else, lo cual es un truco que se usa cuando se quiere ejecutar una parte del código antes de que se salga del while

**For**

Los ciclos for sirven para recorrer una colección.

La estructura es

```
for <variable_que_se_emplea_para_recorrer> in <conjunto_a_recorrer>
```

Ejemplo:

```
>>>lista_numeros= [1,2,3,4,5,6,7,8,9,10]  
>>>for numero in lista_numeros:  
...     print (numero)  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

La variable número va tomando cada uno de los valores dentro de la lista.

**Break**

Es una palabra reservada de Python que se utiliza para romper el ciclo. Se suele usar conjuntamente con condicionales.

Ejemplo:

```
>>>lista_numeros= [1,2,3,4,5,6,7,8,9,10]
>>>for numero in lista_numeros:
...     if numero > 5:
...         break
...     print (numero)
1
2
3
4
5
```

Como se ejecuta el break, que es cuando numero vale 6, no sigue ejecutando el for y por eso no se imprime.

### Continue

Es otra palabra reservada que python utiliza para que, cuando se cumple con determinada condición, se saltee todo el código que le sigue dentro del ciclo, la variable de recorrida toma el siguiente valor en la colección y prosigue con lo que debe ejecutar.

Ejemplo

```
>>>lista_numeros= [1,2,3,4,5,6,7,8,9,10]
>>>for numero in lista_numeros:
...     if numero == 5:
...         continue
...     print (numero)
1
2
3
4
6
7
8
9
10
```

### Reto Ciclos

Crear con un for o un while un ciclo que lo que haga, sea imprimir en mayúsculas las vocales.

Solución con while:

```
vocales = "aeiou"
contador = 0
while contador < len(vocales): ← La función len da el tamaño del string
    letra = vocales[contador]
    print(letra.upper())
    contador +=1
```

Como el índice de elementos en python arranca desde 0, se inicia el contador desde allí, y entonces el contador tiene que ser el largo del string menos 1, o sea, debe imprimir en mayúsculas (para eso el método upper), y vocales[contador] son, en cada ciclo, el valor que tiene en el índice en la cadena. En el ciclo contador va a tomar los valores del 0 al 5 (**contador = 0 y contador +=1**).

El último valor que toma la variable contador en el ciclo, es el número 5, entonces, cuando va a verificar la condición en el siguiente ciclo, no entra porque `len(vocales)` vale 5. Si se hubiera puesto `contador <= len(vocales)` entraría al bucle pero daría error porque `vocales[5]` no existe (pues va de 0 a 4)

Solución con for:

```
vocales = "aeiou"  
for letra in vocales:  
    print(letra.upper())
```

En el for la variable letra va tomando cada uno de los valores del conjunto vocales y los va imprimiendo.

Éstas no son las únicas soluciones posibles (en general, en programación, siempre hay más de una forma de resolver las cosas)

### Entrada de información del usuario desde el prompt

El prompt vendría a ser como el espacio en los formularios en linea que un usuario debe llenar. Python tiene funciones para preguntar al usuario, que él ingrese los datos y luego de que los ingresa, sigue ejecutando lo que siga.

#### **Input**

La función input lo que hace es aceptar un parámetro que es una cadena que le vamos a enviar al usuario como mensaje para decirle qué es lo que queremos.

Ejemplo:

```
>>>nombre= input ("¿Cuál es tu nombre? ")  
¿Cuál es tu nombre?
```

Ésto lo que va a hacer es imprimir la frase, en pantalla y luego de que el usuario ponga una cadena, la va a guardar en la variable nombre.

```
>>> a=input("¿Cuál es tu nombre? ")  
¿Cuál es tu nombre? Jenifer  
>>> a  
'Jenifer'  
>>>
```

La función input siempre va a devolver variables de tipo string, por lo cual, si lo que se le ingresa es un número y después se quiere operar con él, se debe convertir a integer o float antes de usarla

Ejemplo:

```
>>> a =float(input("Dime un número del 1 al 10: "))  
Dime un número del 1 al 10: 7.5  
>>> a  
7.5  
>>> a+2  
9.5
```

Si no se convierte a float, da error al hacer las operaciones, o hace operaciones de tipo string

```
>>> a = input("Dime un número del 1 al 10: ")
Dime un número del 1 al 10: 7.5
>>> a
'7.5'
>>> a*2
'7.57.5'
>>> a+2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

### Excepciones y manejo de errores en Python

Empezamos con un ejemplo:

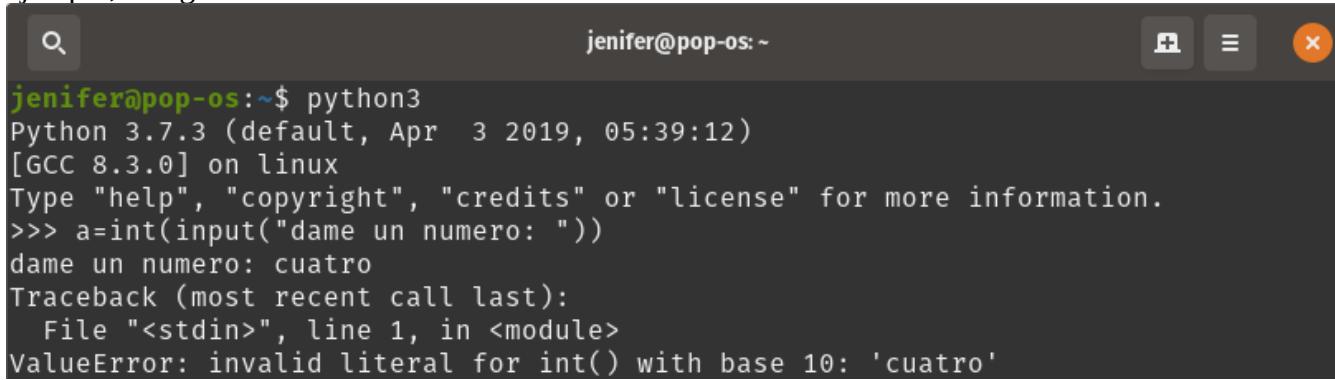
```
>>> a=int(input("dame un numero: "))
dame un numero: 10
>>> b =int (input("Dame otro numero: "))
Dame otro numero: 11
>>> suma = a+b
>>> print ("La suma es: "+suma)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>> █
```

En éste caso, da un error porque estamos tratando de concatenar un string con una variable int y eso Python no lo permite.

La forma correcta de hacerlo sería

```
>>>print ("La suma es "+str(suma))
'21'
```

Otra cosa que nos daría error en el código es si el usuario pone otra cosa que no sean números, por ejemplo, si ingresa el numero 4 como “cuatro”.



```
jenifer@pop-os:~$ python3
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a=int(input("dame un numero: "))
dame un numero: cuatro
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'cuatro'
```

### Variar mensajes de error para presentarle al usuario (uso de try)

No es común que un usuario que utiliza programas escritos en python sepan del lenguaje, por lo cual, no tienen por qué saber los tipos de errores y pueden no entender nada cuando Python da error.

Lo que se puede hacer para que no pase lo del error y se termine de ejecutar el programa.

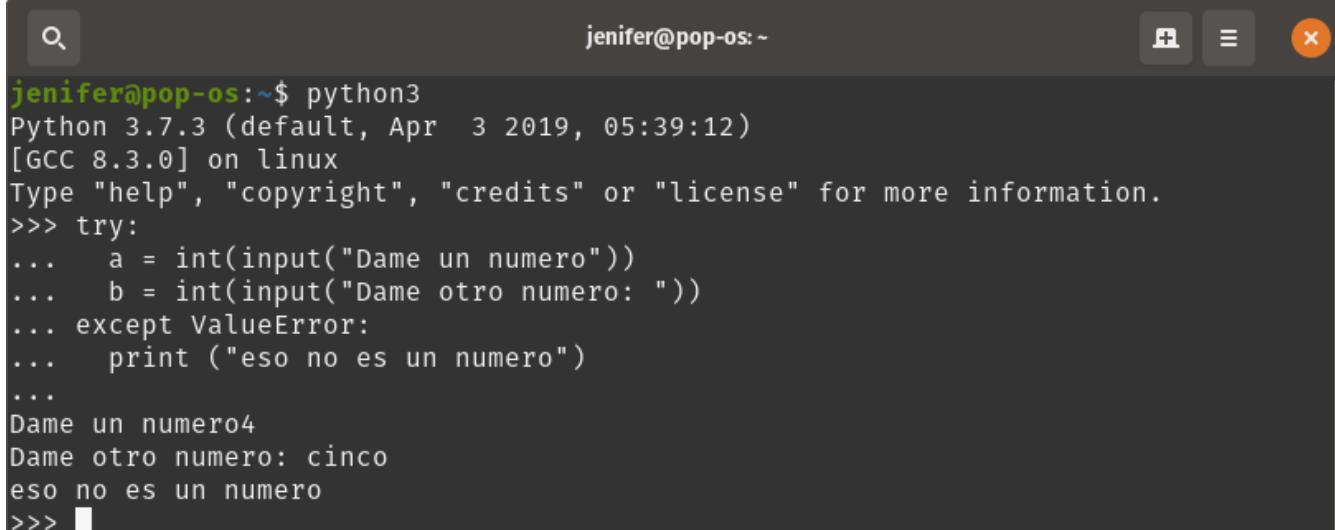
Para eso se usa **try**, que lo que hace es probar condiciones que pueden dar error.

La sintaxis es

**try:**

```
<codigo que puede dar error>
except <nombre del error>:
    <comandos a ejecutar si da éste error>
else:
    <todo el código que se quiera ejecutar si no da error>
Ejemplo
try:
    a=int(input("Dame un numero: "))
except ValueError:
    print("ese no es un número")
else:
    print ("la suma es "+str(a+b))
```

En el código del ejemplo nos queda así:



A screenshot of a terminal window titled "jenifer@pop-os:~". The window shows the following Python session:

```
jenifer@pop-os:~$ python3
Python 3.7.3 (default, Apr 3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> try:
...     a = int(input("Dame un numero"))
...     b = int(input("Dame otro numero: "))
... except ValueError:
...     print ("eso no es un numero")
...
Dame un numero4
Dame otro numero: cinco
eso no es un numero
>>>
```

Para que el programa sepa que hacer en caso de que NO de error, se usa el comando `else`



A screenshot of a terminal window titled "jenifer@pop-os:~". The window shows the following Python session, which includes an `else` block:

```
jenifer@pop-os:~$ python3
Python 3.7.3 (default, Apr 3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> try:
...     a = int(input("Dame un numero: "))
...     b = int(input("Dame otro numero: "))
... except ValueError:
...     print ("eso no es un numero")
... else:
...     print("la suma da: "+ str(a+b))
...
Dame un numero: 1
Dame otro numero: 2
la suma da: 3
>>>
```

**Ejercicio: Hacer una pequeña calculadora**

La calculadora debe presentarse de la siguiente manera:

```
Bienvenido a la calculadora
Estas son las operaciones que pueden hacer
1 - suma
2 - resta
3 - multiplicación
4 - División
Introduce el numero de operación que quieras realizar: 1
inserta el primer número: 8 ← los datos sin negrita son los que introduce el usuario
inserta el segundo número: 2
La suma es 10
¿Deseas continuar? (si/no)
```

La solución que se me ocurrió a mi fue:

```
#####Definición de funciones que hacen las operaciones#####
def suma(valor1,valor2):
    suma= valor1 + valor2
    return str(suma)

def resta(valor1,valor2):
    resta= valor1 - valor2
    return str(resta)

def multiplicacion (valor1,valor2):
    multip = valor1 * valor2
    return str(multip)

def division (valor1,valor2):
    div = valor1 / valor2
    return str(div)
#####

#####Funcion que despliega el menú#####
def impr_op_posibles():
    print("Estas son las operaciones que se pueden hacer")
    print("1 - suma")
    print("2 - resta")
    print("3 - multiplicacion")
    print("4 - division")
#####

print("Bienvenidos a la calculadora")

continuar = "si"

while continuar == "si":
    impr_op_posibles()
    try:
        operacion= float(input("Introduce el numero de operacion que
quieres realizar: "))
```

```
    except ValueError:  
#####Si el usuario no pone una opcion correcta, se le pide que ingrese  
#####un valor correcto#####  
        print("no se reconoce esa operacion")  
        continue  
    else:  
  
#####Acá chequea que el usuario ponga numeros#####  
    try:  
        value1= float(input("inserta el primer numero: "))  
        value2= float(input("inserta el segundo numero: "))  
    except ValueError:  
        print("Lo ingresado no son numeros")  
  
#####Acá entra a hacer la operación que corresponda#####  
    else:  
        if operacion == 1:  
            resultado = suma(value1, value2)  
            print ("la suma es: "+resultado )  
        elif operacion == 2:  
            resultado = resta(value1,value2)  
            print ("la resta es: "+resultado )  
        elif operacion == 3:  
            resultado = multiplicacion(value1,value2)  
            print ("la multiplicacion es: "+resultado )  
        elif operacion == 4:  
            if value2 == 0: #acá chequea que el usuario no divida entre 0  
                print("El numero divisor no puede ser igual a 0")  
            else:  
                resultado = division(value1,value2)  
                print ("la division es: "+resultado )  
  
        else:  
            print ("lo ingresado no son numeros válidos")  
  
#####Acá le pregunta al usuario si quiere continuar#####  
    continuar = input("Desea continuar(SI/no): ")  
    continuar = continuar.lower()  
  
#####Esta parte evita que el programa colapse si el#####  
#####usuario pone otra cosa que no sea si o no#####  
  
    while (continuar != "si") and (continuar != "no"):  
        continuar = input("La opcion ingresada no es valida. Desea  
continuar(SI/no): ")  
        continuar = continuar.lower()  
  
#####Cuando llega acá es porque el usuario no quiere seguir#####  
print ("Gracias por usar nuestra calculadora")
```

```
exit()
```

Apreciaciones: **return** es una palabra reservada en Python que se usa para devolver valores que puedan ser usados en el código principal por ejemplo **return(str(suma))**, lo que hace es transformar la variable suma en formato cadena y retorna esa variable, lo cual se hace al invocarla como **resultado = suma(valor1, valor2)**, va a guardar como un string el resultado de la suma en la variable resultado.

Como ya se mencionó, en programación siempre hay más de una forma de resolver las cosas, el docente del curso lo resolvió así:

```
def realizar_operación(operacion, numero1, numero2):
    if operación == 1:
        return numero1 + numero2
    elif operación == 2:
        return numero1 - numero2
    elif operación == 3:
        return numero1 * numero2
    else:
        return numero1 / numero2

print("Bienvenidos a la calculadora")
while True:
    print("Estas son las operaciones que se pueden hacer")
    print("1 - suma")
    print("2 - resta")
    print("3 - multiplicacion")
    print("4 - division")

    try:
        operación = int(input("Introduce el numero de operacion que
quieres realizar:"))
        numero1=int(input("Introduce el primer numero: "))
        numero2=int(input("Introduce el segundo numero: "))
    except ValueError:
        print ("Por favor, introduce solo numeros")
    else:
        if operación<1 or operación>4:
            print("Ese no es un numero de operación valido")
            continue
        resultado = realizar_operación(operacion,numero1,numero2)
        print("El resultado es" + str(resultado))
        continuar = input(¿Deseas continuar?)
        if continuar.lower() != "si":
            break

print ("Gracias por usar nuestra calculadora")
```

Estructura de datos en Python

Además de las variables comunes que existen en Python (int, float, strings o cadenas, boolean), en particular, dicho lenguaje maneja las siguientes estructuras para guardar datos:

- Listas
- Slices (o rebanadas)
- Diccionarios (Hashes)
- Tuplas
- Conjuntos

### Slices

Se emplean para obtener partes de una lista o cadena cuando no se quiere manejar de a uno los elementos y no se quieren todos los elementos.

Ejemplo:

Si se tiene una lista con los números del 1 al 10

```
numeros=[0,1,2,3,4,5,6,7,8,9,10]
```

>>>numeros [1] va a devolver el elemento que está en la posición 1

```
[1]
```

```
>>>numeros [0:6]
```

[0,1,2,3,4,5] devuelve una lista con los números que estan en la pocisión 0 hasta la posición 5.

Ésto es porque el punto de partida es inclusivo mientras que el de llegada es exclusivo (o sea, excluye a ese índice). La razón de ésto es que, si no se sabe el largo de la liata a priori dado que puede variar porque el tamaño de las listas es dinámico, se puede usar por ejemplo

```
>>>numeros[5:len(numeros)]
```

```
[5,6,7,8,9,10]
```

>>> numeros[4:] devuelve desde la posición 4 hasta el final

```
[4,5,6,7,8,9,10]
```

```
>>>numeros[:5]
```

[0,1,2,3,4] devuelve desde el principio hasta el elemento anterior al indice marcado

```
>>>numeros[:10:2]
```

[0,2,4,6,8] recorre desde el indice 0 al 9 pero devuelve de 2 en 2, o sea, en este caso devuelve lo que está en el indice 2, 4, 6 y 8.

Osea el número de salto es el número de pasos entre cada uno de los elementos del slice.

```
>>>numeros[::-3]
```

```
[0,3,6,9]
```

```
>>> numeros[2:8:2]
```

```
[2, 4, 6]
```

```
>>> numeros[2:8:3]
```

```
[2, 5]
```

En general la nomenclatura es <lista>[<indice\_de\_partida>:<indice\_de\_llegada+1>:<opcional, número\_de\_salto>]

### **Índices negativos**

Por otra parte, los slices aceptan índices negativos

```
>>>numeros[-6:-3]
```

```
[5, 6, 7]
```

Siempre el orden en que van los indices de partida y llegada es de menor a mayor, si no, da conjunto vacío.

Si en el salto se ponen números negativos, va a recorrer de atrás hacia adelante.

```
>>>numeros[::-1]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
>>>numeros[:5:-2]
[10, 8, 6]
```

### Borrar elementos usando slices

Para borrar elementos de una lista, se utiliza la combinación del comando del.

```
>>>del numeros[0]
>>>numeros
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10] ← ahora el elemento con indice 0 de la lista es el valor 1
>>>del numeros[:4]
>>>numeros
[5, 6, 7, 8, 9, 10] ← Ahora el indice 0 tiene valor 5
>>> del numeros[2:4]
>>> numeros
[5, 6, 9, 10]
```

### Reemplazar elementos usando slices

Para reemplazar elementos, se hace citando el índice del elemento que queremos reemplazar

Ejemplo

```
>>> numeros
[5, 6, 9, 10]
>>> numeros[1:2] =[6, 7, 8] ← ésto reemplaza al 6 pues está en la posición 1
>>> numeros
[5, 6, 7, 8, 9, 10]
>>numeros [4:6] = [80, 80]
>>numeros
[5, 6, 7, 8, 80, 80]
>>>numeros [4:6] = [9, 10]
>>>numeros
[5, 6, 7, 8, 9, 10]
```

### Diccionarios (Hashs)

En el caso de las listas, que tienen como características que cumplen con su propósito y nos dejan almacenar cosas de forma ordenada en índices y aparte son mutables, es decir, se puede cambiar su tamaño y contenido. Ésta es una característica de Python pues en otros lenguajes como C, las listas deben tener un tamaño y contenido especificados a priori, y una vez que se define el tamaño y el tipo de datos que contiene, ya no se puede variar (pueden cambiar los valores dentro de la lista, pero no el tipo de datos)

En cuanto a los hashs o diccionarios, se diferencian de las listas en que, en el caso de las listas, para acceder a un elemento de ella, se requiere saber el índice mientras que en el caso de los Hashs, lo que se usa es una estructura de clave → referencia (key → value) permitiendo asociar llaves o identificadores a los valores correspondientes. Además las listas están ordenadas por indices mientras

que los diccionarios, los valores de las keys no están almacenadas en orden, es decir, si agrego una clave y valor nuevos, no se sabe en qué parte del diccionario está, sólo se conoce que esos dos valores estan asociados

Por ejemplo si se tienen estudiantes y sus calificaciones dentro de un curso, se puede usar como clave los nombres, y las notas como la referencia.

```
>>>notas = { 'Pedro' : 7, 'Juan' : 6, 'Maria' : 10, 'Laura' : 5}
>>>notas["Pedro"]
7
```

Para crear un diccionario, se usan los símbolos “{}” para indicar que es un hash y cada uno de los valores van asociados a su llave, va con “:” y cada entrada se divide con comas.

A su vez, si se quiere acceder al valor de una llave, se usan paréntesis rectos con la llave  
**(notas[“Pedro”])**

Por otra parte, las llaves son *case sensitive*, o sea que, por ejemplo Kevin y kevin serían dos llaves diferentes y que además, no se admite llaves repetidas. Si se ingresan 2 llaves llamadas igual, va a tomar un solo valor y el otro lo va a ignorar

Ejemplo:

```
>>> notas = { 'Pedro' : 7, 'Juan' : 6, 'Maria': 10, 'Laura' : 5}
>>> notas
{'Pedro': 7, 'Juan': 6, 'Maria': 10, 'Laura': 5}
>>> notas["Pedro"] = 6
>>> notas
{'Pedro': 6, 'Juan': 6, 'Maria': 10, 'Laura': 5}
>>> notas["Andrea"] = "diez"
>>> notas
{'Pedro': 6, 'Juan': 6, 'Maria': 10, 'Laura': 5, 'Andrea': 'diez'}
>>> notas["pedro"] = 9
>>> notas
{'Pedro': 6, 'Juan': 6, 'Maria': 10, 'Laura': 5, 'Andrea': 'diez', 'pedro': 9}
```

## Dict

Para crear diccionarios también se puede usar la función dict, que es una función que recibe una lista de listas donde en cada sublista se encuentra la llave y el valor y retorna un diccionario con las llaves así indicadas:

```
<nombre_del_diccionario>= dict([[<clave1>,<valor1>],...,
[<claveN>,<valorN>]])
```

Ejemplo:

```
notas = dict([["Pedro",6] , ["Juan","seis"] , ["Maria",10],
["Laura",5]])
```

## Update

Para hacer un update de más de un elemento el el diccionario se invoca la función update y se le pasan las llaves (keys) y sus valores nuevos(value) como un diccionario nuevo, o sea, usa las llaves “{}” separados entre los pares por comas y cada key con su value, separados por dos puntos:

```
<nombre_de_la_lista>.update({<key1>:<value1>, ... , <keyN>:<valueN> })
```

La función update agrega nuevos pares key, value y si, una key ya estaba en el mismo, cambia su valor correspondiente por el value pasado a la función.

## Remover elementos de un diccionario

Para remover cualquier variable en python, se usa la función del. En el caso de los diccionarios, para borrar un elemento se hace utilizando la llave de lo que se quiere borrar:

```
del <nombre_del_diccionario>[<key>]  
del <nombre_del_diccionario> ← esto borra el diccionario entero
```

### Resumiendo con un ejemplo

```
>>> notas=dict([["Juan",9],["Maria", 6]])  
>>> notas  
{'Juan': 9, 'Maria': 6}  
>>> notas.update({"Juan" : 8, "Pedro" :7})  
>>> notas  
{'Juan': 8, 'Maria': 6, 'Pedro': 7}  
>>> del notas["Juan"]  
>>> notas  
{'Maria': 6, 'Pedro': 7}  
>>> del notas  
>>> notas  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'notas' is not defined  
>>>
```

## Iteraciones sobre un diccionario

A diferencia de las listas, los diccionarios pueden no tener ordenados los pares key,value dentro del mismo.

Entonces, si se quieren hacer ciclos, debe recorrer usando las keys.

Por ejemplo si se quiere recorrer con un for el diccionario, la sentencia con la que se recorre es:

```
for <variable_de_recorrida> in <diccionario>,
```

Con ésto, la variable de recorrida va tomando cada una de las llaves.

### Ejemplo

```
>>>notas ={ 'Pedro' : 7, 'Juan' : 6, 'Maria' : 10, 'Laura' : 5}  
>>>for variable in notas:  
    print(variable)  
Pedro  
Juan  
María  
Laura
```

Por otra parte, para obtener los values de cada una de las keys:

```
>>>for variable in notas  
    print(notas[variable])  
7  
6  
10  
5
```

También se puede acceder a todos los valores usando los métodos “.keys” y “.values”

```
>>>for variable in notas.keys():
    print(variable)
Pedro
Juan
María
Laura

>>>for variable in notas.values():
    print(variable)

7
6
10
5
```

Por otra parte, si se quiere acceder a los pares de datos (o sea, key y value al mismo tiempo), se usa el método “.items”

```
>>for variable in notas.items():
    print(variable)
("Pedro", 7)
("Juan", 6)
("María", 10)
("Laura", 5)
```

Lo que devuelve el método es cada uno de los valores del diccionario, pareados en tuplas.

Pero si se quiere recorrer con 2 variables y que cada par sea ejecutado independiente dentro del ciclo:

```
>>for key, val in notas.items():
    print(key)
    print(val)
Pedro
7
Juan
6
María
10
Laura
5
```

## Tuplas

Son estructuras parecidas a las listas, pero la diferencia es que no se pueden mutar, o sea, una vez creada, no se puede cambiar ni los valores dentro de ellas, ni el tamaño.

En el caso de las tuplas, se crean usando paréntesis curvos:

Notación:

`<nombre_de_la_tupla> = (<elemento_1>, ..., <elementoN>)` siendo elemento, cualquier variable que se les ocurra, int, listas, diccionarios, o lo que quieras, y el uso de parentesis curvos, si bien no es necesario (con las comas separando los elementos ya está), es recomendable por una cuestión de que facilita la lectura del código por otros usuarios e incluso uno mismo.

O sea decir `a=(1, 2, 3, 4)` y `a=1, 2, 3, 4` para python es lo mismo, lo toma como una tupla de 4.

Por otra parte, por ejemplo `a=(1, )` o “`a=1,`” crea una tupla con un solo elemento (lo toma como tupla cuando tiene una coma al final)

En cuanto a la característica de inmutabilidad, una vez creada una tupla, lo único que se puede hacer es borrar la tupla entera, no se pueden cambiar los elementos dentro de la tupla:

```
>>> tupla = (1, 2, 3, 4)
>>> del tupla[0]      ← en éste sentido, se trabaja igual que con las listas, con índices
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
```

La única excepción de inmutabilidad es cuando se tienen elementos mutables en las tuplas como las listas y diccionarios. En ese caso se puede alterar la lista o el diccionario, pero en las listas no se lo puede borrar elementos:

Ejemplo:

```
>>> tupla = (1, 2, [3, 4])
>>>tupla [2][1] = 5 ← toma el elemento de indice 1 de la tupla, que es la lista, y sustituye 4
                           por 5
>>>tupla
(1, 2, [3, 5])
>>> del tupla[3][0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: tuple index out of range
```

## Repl.IT

Es un sitio web donde permite desarrollar código en línea. La página es <https://repl.it>

Básicamente es un IDE on-line, un entorno de desarrollo en la nube para varios lenguajes incluyendo Python, Java, Lua, JavaScript entre otros.

El sitio permite crear archivos (a los que repl.it los llama sesiones), ejecutar y compartir código

Pero además lo más curioso es que ustedes pueden tener varios archivos para realizar diferentes proyectos.

Lo único que requiere es crearse una cuenta.

Lo bueno que tiene es que ustedes no requieren instalar nada en la máquina en la cual se desarrolla y por otra parte, guarda un historial por cada archivo, por lo cual, si se debe volver a una versión anterior, se puede hacer. O sea, es un buen entorno para aprender a programar.

El problema es que no es lo ideal para proyectos grandes. En la vida real lo que se hace es codificar en nuestros propios entornos, y después compilar los archivos o hacer lo que tengamos que hacer con ellos. Por otra parte, se depende mucho de la conectividad que tenga uno en internet, y en realidad, no se sabe a ciencia cierta, por cuánto tiempo tendrán soporte y a qué lenguajes.

### Aplicación de lo visto anteriormente: Lista de compras

Todos usamos listas en el mundo real para diferentes cosas, y una de ellas es la lista de compras del super.

La aplicación que se va a desarrollar debe permitir 3 cosas: agregar, remover y ver la lista

Lo primero que hay que hacer es crear una lista, la cual, al principio está vacía.

```
lista_articulos=list()
```

Luego se va a escribir toda la fase de interacción con el usuario pues es la parte que se ve del programa y es la más propensa a dar errores

```
print("Bienvenido a nuestra lista de compras")
print()
while True:
    print("Estas son las operaciones que puedes realizar: ")
    print("1 - Agregar artículo")
    print("2 - Eliminar artículo")
    print("3 - Ver la lista de artículos")
    print("4 - Salir")
    operación = int(input("Ingresa la operación:"))
```

Es una buena práctica que uno se tome el tiempo para programar la interfaz de usuario pues usualmente, uno no hace programas para uno mismo, entonces, cuanto más fácil y flexible sea la interfaz de entrada, mejor.

```
if operacion == 1:
    agregar_articulo() #acá se va a requerir definir ésta función
elif operacion == 2:
    remover_articulo()
elif operación == 3:
    ver_articulos()
elif operación ==4:
    break
else:
    print("def remover_articulo():")
```

```
for indice in range (0, len(lista_articulos)):  
    print(str(indice +1) + " " + lista_articulos[indice])  
  
try:  
    borrar=int(input("Escriba el numero de artículo que desea  
borrar: "))  
except ValueError or borrar < 0 or borrar > len(lista_articulos):  
    print("Lo ingresado no es un articulo valido")  
else:  
    del lista_articulos[borrar -1]Lo ingresado no es una operación  
valida")
```

Recordar que éstos ifs van dentro del while, por lo tanto el break lo rompe.

Fuera del while

```
print("Gracias por usar nuestra lista de compras)
```

Ahora lo que queda es definir las funciones que realizan cada operación del programa:

```
def agregar_articulo():  
    print()  
    articulo = input("Ingresa el nombre del artículo nuevo")  
    lista_articulos.append(articulo)  
    print("Articulo Agregado")  
    print()  
  
def ver_articulos():  
    print ("la lista de articulos es:")  
    for articulo in lista_articulos:  
        print (articulo)  
  
def remover_articulo():  
    for indice in range (0, len(lista_articulos)):  
        print(str(indice +1) + " " + lista_articulos[indice])  
    try:
```

```
borrar=int(input("Escriba el numero de artículo que desea borrar:"))
except ValueError:
    print("Lo ingresado no es un articulo valido")
else:
    elif borrar < 0 or borrar > len(lista_articulos):
        print("Lo ingresado no es un articulo valido")
    else:
        del lista_articulos[borrar -1]
```

Todo éste código puede mejorar en cuanto se tenga presente que siempre hay que pensar en la interfaz de usuario lo mas amena posible, por ejemplo existe el método `capitalize` que lo que hace es tomar la cadena de texto que le ingresa al usuario y pasar la primera a mayuscula.

En el código presentado, se fuerza al usuario a que elija una opción y no lo deja escribir el nombre de artículo. Lo que se puede hacer como alternativa, es, pasar todo lo que ingrese el usuario por el método `.upper`, `.lower` o `.capitalize`, usar el método `.remove` en vez de `del`, y la estructura `try` para controlar que el elemento esté efectivamente en la lista.

Lo recomendado siempre es usar funciones que ya estén establecidas y no estar accediendo directamente a los datos (

### Agenda telefónica: trabajo con diccionarios

En éste ejercicio lo que se plantea es hacer una agenda telefónica simple que guarde los nombres de las personas y sus números de contacto.

Como en toda agenda, el usuario debe tener la opción de agregar contacto, remover contactos, ver toda la agenda y/o un contacto solo, actualizar los datos y salir del programa.

### **Programa principal**

De nuevo, lo primero que debe hacerse es sentarse a diseñar teniendo cuidado, sobre todo con los ciclos `while`, de que no entre en loops infinitos.

Para ello, se diseña primero la interfaz de entrada y un diccionario vacío ya que, con la estructura de diccionarios, permite crear y acceder a los datos con un indicador único.

En éste caso, el identificador que se va a usar como key es el nombre pues es el indicativo de una persona, y como value el número de teléfono, el cual es lo que puede variar en una agenda.

```
####la agenda es vacía
agenda=dict()
```

```
continuar = True
#####Programa Principal#####
print ("Bienvenido a tu agenda")
while continuar == True:
    print ("Lo que puedes hacer es:")
    print ("1) Ver tu agenda")
    print ("2) Buscar un contacto")
    print ("3) Agregar un contacto nuevo")
    print ("4) Eliminar un contacto")
    print ("5) Actualizar contacto")
    print ("6) Salir de la agenda")
try:
    orden = int(input("¿Qué deseas hacer? (marca la opcion con un numero)\n"))
except ValueError:
    print("La opcion no es reconocida")
else:
    if orden == 1:
        desplegar_agenda()
    elif orden == 2:
        ver_contacto()
    elif orden == 3:
        agregar_contacto()
    elif orden == 4:
        remover_contacto()
    elif orden == 5:
        actualizar_contacto()
    elif orden == 6:
        continuar = False
    else:
        print("La orden no es reconocida")
```

```
print ("Gracias por usar la agenda")
```

Con la parte de **try: y except ValueError:** hace que el programa, cuando no ponga cadenas de texto o caracteres raros, le informe que la operación es desconocida.

Por otra parte, los if y elif, hace que si pone operaciones del 1 al 5, llame a la función correspondiente, si pone 6, actualiza la variable continuar (**continuar = False**), sale del while y se termina de ejecutar el programa.

Por último y si pone otro valor numerico que no sea la operación dada, no la reconoce (**else:**  
**print ("La orden no es reconocida")**)

Ahora lo que queda es definir cada función.

Ésta parte es escribir código a intento y error, pero lo que se debe tener claro es que cada función es independiente una de otra, y si el programa llegó a una función determinada, es porque ya pasó las pruebas que les pone el programa principal

### Función Agregar Contacto:

La función agregar contacto, lo que hace es simplemente, primero chequear si el contacto ya existe o no, si existe, le avisa al usuario y vuelve al programa principal, si no existe, guarda en la variable agenda el nombre y el número de teléfono.

```
def agregar_contacto():
    print("agregar contacto nuevo\n")
    nombre= input("Ingresa el nombre de la persona \n")
    if nombre in agenda.keys():
        print("El contacto que deseas agregar ya existe")
    else:
        numero_de_telefono = input ("Ingresa el numero de telefono\n")
        agenda[nombre] = numero_de_telefono
```

### Función Remover Contacto:

La función agregar contacto, lo que hace es simplemente, primero chequear si el contacto ya existe o no, si existe, le avisa al usuario y vuelve al programa principal, si no existe, guarda en la variable agenda el nombre y el número de teléfono.

```
def remover_contacto():
    nombre= input("Escribe el nombre del contacto tal cual aparece en la agenda:\n")
    if nombre in agenda.keys():
        del agenda[nombre]
```

```
else:  
    print("El contacto que quieres borrar no esta en la agenda")
```

### Función Ver Contacto:

Ésta función despliega un contacto de la agenda ingresado por el usuario. Si está, lo despliega, si no, le avisa al usuario que no existe

```
def ver_contacto():  
  
    nombre= input("Escribe el nombre de la persona tal cual esta en la  
agenda:")  
  
    if nombre in agenda.keys():  
  
        print ("El contacto que buscas es este:")  
        print ("Nombre: "+nombre+" Telefono:"+agenda[nombre])  
  
    else:  
  
        print("El contacto que buscas no esta en la agenda")
```

### Función Desplegar la agenda:

Ésta función lo que hace es, si hay contactos, los despliega, si no, le avisa al usuario que la agenda está vacía

```
def desplegar_agenda():  
  
    if agenda :  
  
        print("Estos son tus contactos:")  
        print ("Nombre | Telefono")  
  
        for nombre,telefono in agenda.items():  
            print (nombre+" | "+telefono)  
  
    else:  
  
        print("La agenda esta vacia")
```

La sentencia **if agenda:** devuelve true cuando la variable agenda no está vacía

El ciclo for recorre con 2 variables pareadas, “**nombre**” y “**teléfono**” pues agenda.items() devuelve de la variable agenda cada key con su value pareado.

### Función actualizar contacto:

```
def actualizar_contacto():  
  
    nombre=input("Ingresa el nombre del contacto\n")
```

```
if nombre in agenda.keys():
    agenda[nombre]=input("Ingresa el nuevo numero\n")
else:
    print("El contacto no existe")
```

En síntesis, la versión final:

```
#####Agergar contacto#####
def agregar_contacto():
    print("agregar contacto nuevo\n")
    nombre= input("Ingresa el nombre de la persona \n")
    if nombre in agenda.keys():
        print("El contacto ya existe")
    else:
        numero_de_telefono = input ("Ingresa el numero de telefono\n")
        agenda[nombre] = numero_de_telefono
#####Fin de agregar contacto#####

#####Inicio de remover contacto#####
def remover_contacto():
    nombre= input("Escribe el nombre del contacto tal cual aparece en
la agenda:\n")
    if nombre in agenda.keys():
        del agenda[nombre]
    else:
        print("El contacto que quieres borrar no esta en la agenda")
#####Fin de remover contacto#####

#####Ver un contacto#####
def ver_contacto():
    nombre= input("Escribe el nombre de la persona tal cual esta en la
agenda:")
    if nombre in agenda.keys():
```

```
print ("El contacto que buscas es este:")
print ("Nombre: "+nombre+" Telefono:"+agenda[nombre])
else:
    print("El contacto que buscas no esta en la agenda")
#####
#####ver toda la agenda#####
def desplegar_agenda():
    if agenda :
        print("Estos son tus contactos:")
        print ("Nombre | Telefono")
        for nombre,telefono in agenda.items():
            print (nombre+" | "+telefono)
    else:
        print("La agenda esta vacia")
#####
##Fin de desplegar agenda #####
#####

#####
def actualizar_contacto():
    nombre=input("Ingresa el nombre del contacto\n")
    if nombre in agenda.keys():
        agenda[nombre]=input("Ingresa el nuevo numero\n")
    else:
        print("El contacto no existe")
#####
####la agenda es vacía
agenda=dict()
continuar = True

###Programa Principal#####
```

```
print ("Bienvenido a tu agenda")
while continuar == True:
    print ("Lo que puedes hacer es:")
    print ("1) Ver tu agenda")
    print ("2) Buscar un contacto")
    print ("3) Agregar un contacto nuevo")
    print ("4) Eliminar un contacto")
    print ("5) Actualizar contacto")
    print ("6) Salir de la agenda")
try:
    orden = int(input("¿Qué deseas hacer? (marca la opcion con un numero)\n"))
except ValueError:
    print("La opcion no es reconocida")
else:
    if orden == 1:
        desplegar_agenda()
    elif orden == 2:
        ver_contacto()
    elif orden == 3:
        agregar_contacto()
    elif orden == 4:
        remover_contacto()
    elif orden == 5:
        actualizar_contacto()
    elif orden == 6:
        continuar = False
    else:
        print("La orden no es reconocida")

print ("Gracias por usar la agenda")
```

## Apreciaciones

Éste código puede mejorar muchísimo, para eso, lo que se sugiere es compartmentarizar el problema y hacer aparte todo lo que tiene que ver con interfaz de usuario, cómo se despliegan las cosas, tener una función aparte de chequeo de que la agenda pueda variar o no, ofrecerle al usuario salir del programa en el medio, que pueda modificar los nombres, usar expresiones regulares, en fin, python tiene la capacidad de hacerlo, es solo buscar.

Lo que se recomienda es **separar las tareas** para hacer los testeos más fácil.

### TIP: truco

En python existe un truco que es asignar una variable, por lo general caracteres:

```
variable = None
```

Eso crea una variable vacía.

## Aplicación adivina el número

Una de las formas más fáciles (y más divertidas, opinión personal) de aprender un programa es programar un juego.

Por ejemplo guardar en variables el número de vidas que te quedan, el nombre del personaje, la cantidad de monedas que vas juntando, la lista de items que tenés y sus características, se utilizan ciclos para saber si estás vivo o no en un juego, en fin, no cosas.

Usualmente para juegos web usa javascript, para PC, C#, sony tiene algun lenguaje, x...todo en algún momento pasó por un lenguaje de programación.

En éste caso se va a programar un juego de adivinar el número, el cual es un juego fácil:

La computadora va a crear un número aleatorio en un rango determinado y el jugador va a tener que adivinar el número.

El programa puede dar pistas de si nos acercamos o nos alejamos.

### Import

Para que la máquina cree un numero aleatorio, se va a usar código desarrollado por otros programadores. Cabe destacar que eso ya lo veníamos haciendo al usar funciones como .upper(), .lower(), .items().

En éste caso, como la función básica no está en nada de lo que se vió, (y para usar código de librerías) se debe importar.

### Sintaxis de import:

```
import <nombre_de_la_librería> (as <nombre_que_querramos_ponerme>)
```

Lo que está entre paréntesis curvos, no siempre es necesario

Otra cosa importante: los imports **siempre** van en la cabecera porque es lo primero que tiene que hacer python para poder trabajar

En el caso del juego hay que importar la función random:

```
import random
```

Random es una librería de python que tiene que ver con manejar cosas de randomización

Para generar un número al azar, hay que decirle que genere un numero entero al azar:

```
numero_aleatorio = random.randint(<numero_de_inicio>, <numero_final>)
```

La función randint, necesita 2 parametros, el inicio y el final

si el import hubiera sido:

```
import random as rand
```

Entonces

```
numero_aleatorio= rand.randint(<numero_de_inicio>, <numero_final>)
```

### Versión sencilla del juego:

```
import random

#####Programa Principal#####
numero_aleatorio = random.randint(1,10)
print("Bienvenido a nuestro juego adivina el número")
print("Estoy pensando en un número del 1 al 10:")
while True:
    numero_del_jugador = int(input("Ingresa el número: "))
    if numero_del_jugador == numero_aleatorio:
        print("Adivinaste!!!")
        break
    else:
        print("Fallaste!!!")
print (Gracias por jugar con nuestro progama)
exit()

#####Fin de programa#####
```

Este programa es limitado en cuanto no le da pistas al usuario, puede seguir hasta que no le atine al número, si el usuario ingresa algo que no sea números y si da error.

Entonces, primero lo que se tiene que hacer es una lista de tareas, las cuales permiten determinar tanto las funciones como el programa principal:

- Decirle al usuario que tiene que adivinar el número en, por ejemplo, del 1 al 10
- Decirle al usuario que tiene x intentos
- darle pistas de si es mayor el número que tiró o es menor
- Preguntarle al usuario si quiere jugar de nuevo
- Preguntarle al usuario en qué modalidad quiere jugar: Facil, mediano o dificil.

En el facil es como ya está, mediano es hacer un rango de numeros mas grande y dificil, que sea el reto del mediano pero al final tenga menos intentos

### Tip: Format

Format lo que toma es un string con corchetes y se le debe pasar como argumento (dentro de los parentecis curvos, la variable que se quiere poner en el lugar donde van los corchetes. El return de .format es otro string con el valor de la variable. Por ejemplo:

```
for i in range(1,11)
    print("{} elefante se balanceaba...".format(i))
1 elefantes se balanceaban...
2 elefantes se balanceaban...
3 elefantes se balanceaban...
4 elefantes se balanceaban...
5 elefantes se balanceaban...
6 elefantes se balanceaban...
7 elefantes se balanceaban...
8 elefantes se balanceaban...
9 elefantes se balanceaban...
10 elefantes se balanceaban...
```

### Solución final:

```
import random
#####Este programa es para que el usuario adivine el numero correcto
en un rango de números determinado #####
#####
#####Esta funcion define el reto#####
def desafio():
    numero_aleatorio = random.randint(numero_inicio,numero_fin)
```

```
print("Estoy pensando en un numero del "+ str(numero_inicio)+" al "+str(numero_fin))
print("Tienes "+str(vidas)+" para lograrlo")
return numero_aleatorio
#####
#####Esta funcion es para preguntarle si quiere seguir jugando#
#####Lo que retorna es un boolean#####
#####No va adejar en paz al jugador hasta que no haga lo que#
#####yo quiero#####
def jugar_de_nuevo():
    intentar = input("Deseas seguir jugando? (Si/No) : ")
    if intentar.lower() == "si":
        return True
    elif intentar.lower() == "no":
        return False
    else:
        print("La opcion marcada no es reconocida")
        jugar_de_nuevo()
#####
#####Pista#####
def pista():
    if int(numero_aleatorio) > int(numero_jugador):
        print("El numero que estoy pensando es mas grande")
    else:
        print("El numero que estoy pensando es mas chico")
#####
#####Manejo de ValueError#####
def recibe_numero():
    try:
        n_jugador = int(input("Adivina que numero es: "))
    except ValueError:
        print ("Por favor inserta un numero valido:")
        return recibe_numero()
    else:
        return n_jugador
#####
#####Esta funcion define las constantes según el grado de
dificultad#####
#####Elegido por el usuario#####
def que_tan_facil_es():
    print("Elige la dificultad en la que quieras jugar (indica con un
numero):")
    print ("1- Facil")
    print ("2- Medio")
    print ("3 - Dificil")
```

```
try:
    dificultad=int(input(": "))
except ValueError:
    print("No se reconoce la operacion")
    return que_tan_facil_es()
else:
    if dificultad ==1:
        vidas = 5
        numero_inicio = 1
        numero_fin = 10
        return (vidas,numero_inicio,numero_fin)
    elif dificultad ==2:
        vidas = 5
        numero_inicio = 1
        numero_fin = 50
        return (vidas,numero_inicio,numero_fin)
    elif dificultad == 3:
        vidas = 3
        numero_inicio = 1
        numero_fin = 50
        return (vidas,numero_inicio,numero_fin)
    else:

        return que_tan_facil_es()

#####
#####programa principal#####
print("Bienvenido a nuestro juego adivina el número")
#####ésta variables generan el número aleatorio, el rango de
numeros y los intentos que tiene#####

otra_vez = True

#####
#####programa principal#####
while otra_vez:
    (vidas, numero_inicio, numero_fin) = que_tan_facil_es()
    numero_aleatorio = desafio()
    numero_jugador = recibe_numero()
    vidas_del_jugador = vidas
    #####En éste ciclo trata de adivinar#####
    #####Si sale de éste while es porque adivino o#####
    #####Perdio#####
    while vidas_del_jugador > 0 :
        if vidas != vidas_del_jugador:
            numero_jugador = recibe_numero()
        if numero_aleatorio != numero_jugador:
            print("Fallaste!!!")
            pista()
            vidas_del_jugador -=1
```

```

        print("Te quedan " + str(vidas_del_jugador) +" vidas")
    else:
        break

    if numero_aleatorio != numero_jugador:
        print("Perdiste")
        print("El numero que estaba pensando es "+str(numero_aleatorio))
    else:
        print("Adivinaste!!!")

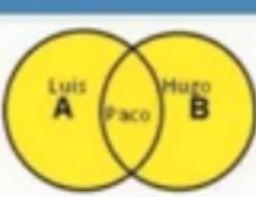
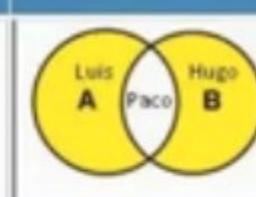
otra_vez = jugar_de_nuevo()
#####
print("Gracias por jugar")

```

### Conjuntos (Sets)

Al igual que en la vida real, en python hay implementadas operaciones para hacer con conjuntos.

En conjuntos hay 3 operaciones básicas:

	Union	Intersección	Diferencia	Diferencia Simétrica
En el caso de la				
	Las personas que cuenten con un medio de transporte: Auto o Bicicleta	Las personas que cuenten Auto y Bicicleta	Las personas que cuenten Bicicleta pero no con Auto	Las personas que cuentan con Auto o Bicicleta, pero no los dos
	Hugo, Paco, Luis	Paco	Hugo	Hugo, Luis

diferencia, el orden del los factores si altera el producto porque  $A-B = \text{Luis}$  y  $B-A=\text{Paco}$

La función **set()** crea conjuntos nuevos.

En Python hay funciones que hacen las operaciones:

Operación	Equivalente	Resultado
<code>len(s)</code>		Devuelve la cantidad de elementos (cardinal)
<code>x in s</code>		Da True si el elemento x está en el conjunto
<code>x not in s</code>		Da True si el elemento x no está en el conjunto
<code>s.issubset(t)</code>	<code>s &lt;= t</code>	Testea si todos los elementos del conjunto s están dentro del conjunto t
<code>s.issuperset(t)</code>	<code>s &gt;= t</code>	Testea si todos los elementos del conjunto t están dentro del conjunto s
<code>s.union(t)</code>	<code>s   t</code>	Une los conjuntos s y t
<code>s.intersection(t)</code>	<code>s &amp; t</code>	Calcula la intersección de los conjuntos s y t
<code>s.difference(t)</code>	<code>s - t</code>	Calcula la diferencia s-t (recordar que t -s no da lo mismo)
<code>s.symmetric_difference(t)</code>	<code>s ^ t</code>	Calcula la diferencia simétrica entre s y t
<code>s.copy()</code>		Hace una copia del conjunto

**Practica de conjuntos en python (conjuntos I yII de video del curso)**

Lo primero que hay que hacer es armar una lista de las cosas que queremos que haga el programa en Python

- Obtener los sets del usuario a los que llamaremos A y B
- Hacer las funciones de union, intersección, diferencia y diferencia simetrica
- Hacer la función de devolverle al usuario el conjunto resultado.

Entonces:

La funcion **split** es una función de cadena que crea recibe una lista, si no se pone nada entre parentesis, usa el espacio vacío.

Ahora la interfase de usuario:

```
def ver_instrucciones():
    print("Operaciones que puedes realizar:")
    print("1- Union")
    print("2 - Interseccion")
    print("3 - Diferencia")
    print("4 - Diferencia Simétrica")
    print("5 - Ver instrucciones")
    print("6 - Salir")

def calculadora_conjuntos():
    print(Bienvenidos a los conjuntos)
    print(Introduce los elementos del conjunto separados por
espacios")
    print(Ejemplo: 1 2 3 4 0 8 5)
    conjunto_A= set(input("Conjunto A: ").split())
    conjunto_B= set(input("Conjunto B: ").split())
    while True:
        ver_instrucciones()
        operación = int(input("Elige una: "))
        if operación == 1:
            print ("Union:")
        elif operación == 2:
            print ("Intersección:")
        elif operación == 3:
            print ("Diferencia:")
        elif operación == 4:
            print ("Diferencia simétrica:")
        elif operación == 5:
            ver_instrucciones()
        elif operación == 6:
            print ("Salir")
            break
    else: #este else es del while
```

```

print("No reconozco ésta operación")
#####
calculadora_conjuntos()
print ("Gracias por usar nuestro programa")

```

Éste código así como está, solo despliega las instrucciones y si se pone 6, sale del programa, pero el esqueleto del programa ya está.

Ahora lo que se va a hacer sustituir cada uno de los prints e ir definiendo cada uno de los prints por las funciones correspondientes:

### Definición de Funciones de operaciones (conjuntos III, IV)

#### Función de union

La función union va a tomar como parámetros dos conjuntos (o sea, va a tomar dos parámetros, en, este caso el conjunto a y el conjunto b y va a devolver la union)

Usando la tabla colocada más arriba nos queda que

```

def union_conjuntos(conj_a,conj_b):
    print()
    print("La unión de A y B es: {}".format(conj_a.union(conj_b)))
    print()

```

El print hace primero la operación con los conjuntos, luego formatea los conjuntos a strings y completa la frase.

#### Función de intersección

```

def intersec_conjuntos(conj_a,conj_b):
    print()
    print("La unión de A y B es"
    "{}".format(conj_a.intersection(conj_b)))
    print()

```

#### Función de Diferencia Simétrica:

```

def dif_sym_conjuntos(conj_a,conj_b):
    print()
    print("La diferencia simétrica entre A y B es:"
    "{}".format(conj_a.symmetric_difference(conj_b)))
    print()

```

#### Función de Diferencia:

Ésta funcione es un poco más sensible porque no es lo mismo hacer conjuntos  $A - B$  que  $B - A$ , entonces hay que darle al usuario la opción de que conjunto va a restar:

```

def diferencia_conjuntos(conj_a,conj_b):
    print("¿Qué diferencia quieres realizar")
    print ("1 - Conjunto A - B")

```

```
print ("1 - Conjunto B - A")
operación = int(input("Elije una con números:"))
if operación == 1:
    print("La diferencia A - B es:")
    print("{}".format(conj_a.difference(conj_b)))
    print()
elif operación == 2:
    print("La diferencia B - A es:")
    print("{}".format(conj_b.difference(conj_a)))
    print()
else:
    print("La opción no es reconocida")
    diferencia_conjuntos(conj_a,conj_b)
```

### Mejorado de la robustez del programa (conjuntos V y VI)

Si el usuario en nuestro programa pone al desplegar opciones, cosas que no son numeros, da error de tipo ValueError, entonces se usa try, except:

```
def calculadora_conjuntos():
    print(Bienvenidos a los conjuntos)
    print("Introduce los elementos del conjunto separados por
espacios")
    print(Ejemplo: 1 2 3 4 0 8 5)
    conjunto_A= set(input("Conjunto A: ").split())
    conjunto_B= set(input("Conjunto B: ").split())
    while True:
        ver_instrucciones()
        try:
            operación = int(input("Elije una: "))
        except ValueError:
            print ("La opcion no es reconocida")
        else:
            if operación == 1:
                union_conjuntos(conjunto_A,conjuntoB)
            elif operación == 2:
                intersec_conjuntos(conjunto_A,conjuntoB)
            elif operación == 3:
                diferencia_conjuntos(conjunto_A,conjuntoB)
            elif operación == 4:
                dif_sym_conjuntos(conj_a,conj_b)
            elif operación == 5:
                ver_instrucciones()
            elif operación == 6:
                print ("Salir")
                break
    else: #este else es del while
        print("No reconozco ésta operación")
```

```
#####
calculadora_conjuntos()
print ("Gracias por usar nuestro programa")
```

Lo mismo pasa con la función diferencia:

```
def diferencia_conjuntos(conj_a,conj_b):
    print("¿Que diferencia quieres realizar")
    print ("1 - Conjunto A - B")
    print ("1 - Conjunto B - A")
    try:
        operación = int(input("Elije una con números:"))
    except ValueError:
        print("\nDebes poner 1 o 2:\n")
        diferencia_conjuntos(conj_a,conj_b) #si no se hace ésto, sale de
la funcion
    else
        if operación == 1:
            print("La diferencia A - B es:
{}".format(conj_a.difference(conj_b)))
            print()
        elif operación == 2:
            print("La diferencia B - A es:
{}".format(conj_b.difference(conj_a)))
            print()
    else:
        print("La opción no es reconocida")
        diferencia_conjuntos(conj_a,conj_b)
```

La idea con ésto es que, si da un ValueError, y no se llama a la función, sale de la misma y va al programa principal.

## Programación orientada a objetos

### Introducción

Podemos llegar muy lejos con funciones en Python podríamos tener algo así como un solo archivo con cientos o incluso miles de funciones y líneas de código y al final simplemente ejecutarlo.

Y esto no tendría ninguna clase de impacto en el rendimiento o en la funcionalidad.

Pero imagínense darle a otro desarrollador un solo archivo de Python que tenga un millón de líneas de código y cientos de miles de funciones.

Tal vez nosotros lo entendamos porque nosotros lo escribimos pero para él no sería nada agradable tratar de leer ese archivo y mucho menos mantenerlo o modificar el código.

Ahí es cuando entra la programación orientada a objetos.

La programación orientada a objetos se basa en envolver las piezas de código en clases que nos

ayudan a tener funciones y atributos que son comunes en nuestro código.

En el mundo real tenemos un refrigerador que tiene un atributo como su color su año de manufacturación; tiene una función como es crear hielos tiene la función de tal vez darte agua refrigerar alimentos etc.

En nuestro código de Python también podemos tener clases por ejemplo la clase cadena que hemos utilizado ya realmente hasta este punto que se encarga de crear cadenas.

Tiene también atributos como la longitud de la cadena tiene el número de caracteres tiene funciones como volverla todas mayúsculas o todas minúsculas etc.

Acá hay nuevos conceptos como: clases, objetos y atributos, los cuales se desarrollarán a continuación.

## Clases

En informática, una clase es una plantilla para la creación de objetos de datos según un modelo predefinido. Las clases se utilizan para representar entidades o conceptos.

Por ejemplo, la variable (u objeto) lapiz, pertenece a la clase cartuchera, y la clase cartuchera tiene métodos: abrir, cerrar, guardar objeto y sacar objeto

O sea, una clase es un modelo que define un conjunto de variables el estado, y métodos apropiados para operar con dichos datos el comportamiento. Cada objeto creado a partir de la clase se denomina instancia de la clase.

En python para definir una clase, se emplea la palabra reservada **class** y se define así:

```
class <nombre de la clase>():
    <codigo 1>
    <codigo 2>
    .
    .
    .
```

Por ejemplo:

En un archivo cartuchera.py, el cual va a ser nuestra librería, que llamaremos Cart, se define la clase Cartuchera

```
class Cart():
    lapiz = "Lapiz"
    goma = "Goma"
    sacapuntas = "Sacapuntas"
    regla = "Regla"
```

Para usarla, se crea otro archivo, por ejemplo main.py:

```
import Cart from cartuchera
print (Cart.lapiz())
```

Cuando se ejecute main.py, va a importar la clase Cart y va a imprimir la cadena que devuelve la variable lapiz, el cual, dentro de la clase lápiz, es un string “lapiz”.

Ésto sigue la misma filosofía que en el juego andvina el número al importar la función **random**, solo que en vez de usar una librería de python ya armada, las clases las hacemos nosotros.

### **Convección:**

Si bien los nombres de los archivos y de las clases en Python pueden poner en mayúsculas, es una convención entre programadores que los nombres de los archivos de librerías se escriban todos en minúsculas y las clases, lleven en mayusculas la primera letra.

### **Instancias de clases (Objetos)**

Son buenas para guardar información por supuesto pero si hacemos un cambio en ellas afectarían a todas las partes en las que las estamos utilizando; por esto, lo mejor es crear instancias de nuestra clase mejor conocidas como objetos.

Los objetos se pueden entender como copias de nuestra clase pero cada una tiene sus atributos individuales.

Por ejemplo, si se tiene definida la siguiente clase dentro de un archivo llamado player.py :

```
class Player():
    hit_points = 50
    mana = 50
    vocation = "No vocation"
```

En nuestro caso cada instancia de Player cada objeto de jugador tendría sus hit points su maná y su vocación.

Podemos tener un jugador que sea mago otro que sea nada otro que sea Paladín, Knight, Druid etcétera.

Como crear un objeto a partir de una clase:

Simple, en el programa principal, en nuestro ejemplo se crea una variable llamada sorcerer  
**sorcerer = Player()**

Esto quiere decir que vamos a crear un nuevo objeto que es de la clase Player. Éste objeto va a tener sus hit\_points, su mana y su vocación

Si queremos ver sus hit\_points por ejemplo, en el program principal nos queda:

```
import Player from player
sorcerer = Player()
print(sorcerer.hit_points)
print(sorcerer.mana)
print(sorcerer.vocation)
```

La variable **sorcerer** es un objeto que, a su vez, es una instancia de nuestra clase pues ya **sorcerer**, hit points, mana y vocation.

### Afectación de objetos

Nuestro jugador es exactamente igual a cualquier jugador, tiene 50 puntos de vida 50 de Maná y no tiene vocación pero a nuestro sorcerer podemos afectarle sus atributos sin modificar la clase en particular:

```
sorcerer.vocation="sorcerer"
```

Como es un hechicero, suelen tener menos vida, pero mucho mana para poder castear spells:

```
sorcerer.hit_points= 40
```

```
sorcerer.mana = 80
```

Si se ejecuta el siguiente código:

```
import Player from player

generico = Player()
sorcerer = Player()

sorcerer.vocation="sorcerer"
sorcerer.hit_points= 40
sorcerer.mana = 80
Print("Sorcerer:")
print(sorcerer.hit_points)
print(sorcerer.mana)
print(sorcerer.vocation)

print ("#####")
Print("Personaje Generico:")
print(generico.hit_points)
print(generico.mana)
print(generico.vocation)
```

Vemos que, para el personaje genérico quedan los valores por default del archivo player.py, mientras que el del Sorcerer cambió.

```
Python 3.7.4 (default, Jul  9 2019, 00:06:43)
[GCC 6.3.0 20170516] on linux
Sorcerer:
40
80
sorcerer
#####
Personaje Generico:
50
50
No vocation
> []
```

Esto es porque **genérico** y **sorcerer** son 2 instancias (objetos) de la clase player, tienen las mismas variables (hit\_points, mana y vocación), pero al sorcerer se le cambiaron los valores que venían por defecto. Cada objeto se basa en la clase pero podemos modificar sus atributos para que se ajusten a nuestras necesidades.

## **Métodos**

Se declaran igual que las funciones, y toman sí o si, un parámetro de entrada que es **self** porque es para modificar instancias de la clase.

Ésto se usa para que, cuando se quiera cambiar una variable de las que vienen por default.

Por ejemplo, si se tienen 2 archivos, el principal y el de la librería player que es nuestra, en player.py:

```
class Player():
    hit_points = 50
    mana = 50
    vocation = "No vocation"
    hechizo = "Puff" #Por default un personaje sin vocation no castea

    def lanzar_hechizo(self):
        return self.hechizo = "utevo lux"
```

**Self lo que va a tomar es una instancia de la clase.**

Por lo cual, si se toma como ejemplo lo anterior, en el programa principal, solo va a lanzar hechizo el sorcerer:

```
from player import Player

generico = Player()
sorcerer = Player()

sorcerer.vocation="sorcerer"
sorcerer.hit_points= 40
sorcerer.mana = 80
print ("Sorcerer:")
print (sorcerer.hit_points)
print (sorcerer.mana)
print (sorcerer.vocation)
sorcerer.hechizo = "utevo lux"
print (sorcerer.lanzar_hechizo())

print ("#####")
print ("Personaje Generico:")
print (generico.hit_points)
print (generico.mana)
print (generico.vocation)
print (generico.lanzar_hechizo())
```

La salida nos queda que:

```
jenifer@pop-os:~/Documentos/qbit/Python_MM0$ python3 main.py
Sorcerer:
40
80
sorcerer
utevo lux
#####
Personaje Generico:
50
50
No vocation
Puff
```

Ésto ocurre porque **sorcerer** y **genérico** son 2 instancias de la clase Player. Como por default, el genérico no castea nada, **generico.lanzar\_hechizo()** devuelve “Puff”, como se escribió **sorcerer.hechizo= “utevo lux”**, entonces **sorcerer.lanzar\_hechizo()** devuelve eso y el print lo imprime. Si se me ocurriese un personaje nuevo, lo único que debo hacer es, crear otra instancia en el main y ya está:

```
from player import Player

generico = Player()
sorcerer = Player()

sorcerer.vocation="sorcerer"
sorcerer.hit_points= 40
sorcerer.mana = 80
print("Sorcerer:")
print(sorcerer.hit_points)
print(sorcerer.mana)
print(sorcerer.vocation)
sorcerer.hechizo = "utevo lux"
print(sorcerer.lanzar_hechizo())

print ("#####")
print("Personaje Generico:")
print(generico.hit_points)
print(generico.mana)
print(generico.vocation)
print(generico.lanzar_hechizo())
print ("#####")

knight = Player()
knight.vocation = "knight"
knight.hechizo = "pegar"
print(knight.hit_points)
```

```
print(knight.mana)
print(knight.vocation)
print(knight.lanzar_hechizo())
```

Esto nos da que:

```
jenifer@pop-os:~/Documentos/qbit/Python_MM$ python3 main.py
Sorcerer:
40
80
sorcerer
utevo lux
#####
Personaje Generico:
50
50
No vocation
Puff
#####
Knight:
50
50
knight
pegar
```

### Método init:

`__init__` (va con 2 guiones bajos) es un método especial en python que se emplea para inicializar variables dentro de una clase. Es importante porque bajo éste puede manejar entradas del usuario, entre otras cosas.

En el ejemplo de la clase player de arriba, se reescribe de la siguiente forma:

```
class Player():
    def __init__(self, hit_points, mana, vocation, hechizo):
        self.hit_points = hit_points
        self.mana = mana
        self.vocation = vocation
        self.hechizo = hechizo #Por default un personaje sin vocation
no castea

    def lanzar_hechizo(self):
        return self.hechizo
```

Y ahora para llamar a player en el programa principal, puedo usar directamente las variables:  
from player import Player

```
generico = Player(80, 40, "No vocation", "Puff")
sorcerer = Player(80, 40, "Sorcerer", "utevo lux")

print("Sorcerer:")
print(sorcerer.hit_points)
print(sorcerer.mana)
print(sorcerer.vocation)
print(sorcerer.lanzar_hechizo())

print ("#####")
print("Personaje Generico:")
print(generico.hit_points)
print(generico.mana)
print(generico.vocation)
print(generico.hechizo)

print ("#####")
knight = Player(80, 40, "No vocation", "Pegar")
print("Knight:")
print(knight.hit_points)
print(knight.mana)
print(knight.vocation)
print(knight.lanzar_hechizo())
```

El tema es que así como está escrito el método `__init__`, obliga a que le pase siempre todos los parámetros. Para evitar que ponga eso, se puede poner valores por default:

```
class Player():
    def __init__(self, hit_points = 50, mana = 50, vocation="No
vocation", hechizo="Puff"):
        self.hit_points = hit_points
        self.mana = mana
        self.vocation = vocation
        self.hechizo = hechizo #Por default un personaje sin vocation
no castea
```

Con ésto cuando se llame a `Player` en el programa `main`, si no se le pasa ningún parámetro, el jugador va a tener 50 hit points, 50 de mana, no va a tener `vocation` y va a castear `puff`

## Clases y Diccionarios

Los diccionarios tambien sirven para pasar variables a las clases en el sentido que, en los diccionarios, existe `**kwargs`.

Cuando una función o un método, recibe como parámetro **\*\*kargs**, lo que python entiende es que el nombre del atributo es el key, y el valor es el value.

En nuestro ejemplo, dentro de la clase Player:

```
class Player():
    def __init__(self, **kargs):
        self.hit_points = kargs.get(hit_points, 50)
        self.mana = kargs.get(mana, 50)
        self.vocation = kargs.get(vocation, "No vocation")
        self.hechizo = kargs.get(hechizo, "Puff")
```

Y en el programa principal

```
mi_personaje=Player(hit_points= 40, mana = 60, vocation
="sorcerer", hechizo = "utevo lux")
```

En éste caso, el objeto mi\_personaje, va a ser un sorcerer, los hit points van a tener 40, y va a poder castear “utevo lux”, y en mi personaje 2:

```
mi_personaje2=Player(hit_points= 100, vocation ="Knight", hechizo =
"pegar")
```

Va a tener 100 puntos de vida, el mana va a ser 50 que es el valor por default, va a ser un Knight, y su “Hechizo” va a ser “pegar”.

Lo bueno también de usar diccionarios, es que al llamar a la clase, no importa el orden de los parámetros dado que es un diccionario.

## **Heredabilidad**

Todos somos objetos en una gran escala del universo.

Nosotros como personas somos objetos.

Nuestro planeta Tierra es un objeto nuestra galaxia es un objeto más para el universo.

Éste concepto aplica para lenguajes de programación como Python, y muy comúnmente oiremos a los desarrolladores de dicho lenguaje decir que todo es un objeto porque así es.

Los diccionarios, los conjuntos las listas las cadenas todo lo que hemos usado hasta este punto del curso, son objetos y además resulta que todas las clases que tenemos, heredan de la clase objeto.

## **Que es la herencia**

Hay que pensar en la herencia como algo del mundo real.

Nosotros heredamos cosas de nuestros padres tenemos tal vez sus ojos tenemos su color de cabello tenemos. Su color de piel y tal vez nos gustan algunas cosas similares pero no solamente somos iguales, tambien tenemos cosas que nos distinguen de nuestros padres.

Tal vez yo soy un poco más alto que mi papá tal vez a mí me gusta el fútbol mientras que a él le gusta el basquetbol etcétera etcétera.

La herencia nos permite heredar metodos y atributos de otras clases para no tener que repetir sus atributos y sus funciones.

Pero además podemos sobrescribir sus atributos y poner nuestras propias funciones para tener más funcionalidad y no repetir nuestro código.

### Ejemplos con clases

El ser humano es un mamífero, por lo tanto la clase ser humano hereda 2 atributos que hereda de la clase Mamiferos que son Pelo, Mamas y Sexo. La diferencia por ejemplo con la clase Perro, es por ejemplo el número de mamas. Por otra parte, puedo hacer una clase Uruguayo, que va a heredar todos los atributos de Ser Humano, pero además vaclase Uruguayo, a tener un atributo nuevo que es su CI. La clase perro, como no está bajo la clase ser humano ni bajo la clase Uruguayo la no va a heredar nunca el atributo CI

Para indicarle a python que una clase se hereda de otra, se pone entre paréntesis

Pero yendo al ejemplo de las clases en nuestro MMO usando python:

En el archivo de librería (player.py):

```
class Player():
    vocation="No vocation"
    hechizo = "Puff"
    movement_speed = 50
    def __init__(self, **kargs):
        self.hit_points = kargs.get(hit_points,50)
        self.mana = kargs.get(mana,50)
    def lanzar_hechizo(self):
        return self.hechizo

#####Acá indica que hereda todos los atributos#####
#####Y metodos de Player #####
class Sorcerer (Player):
    vocation = "Sorcerer"
    hechizo = "Utevo Lux"
    movement_speed = 30
```

O sea, esto hace que, lo único que recibe la clase Player del programa principal son vida y mana, y si se invoca a la clase Sorcerer, cambia los atributos de vocacion, hechizo y movement\_speed.

Para llamar en main:

```
import player
```

```
generico = player.Player(hit_points = 50, mana = 50)
```

```
sorcerer = player.Sorcerer(hit_points = 40, mana = 90)

print("Sorcerer:")
print("HP: {}".format(sorcerer.hit_points))
print("Mana: {}".format(sorcerer.mana))
print("Vocation: {}".format(sorcerer.vocation))
print("Su hechizo es: {}".format(sorcerer.lanzar_hechizo()))
print(("Su velocidad de moverse es:
{}".format(sorcerer.movement_speed)))

print ("#####")
print("Personaje Generico:")
print("HP: {}".format(generico.hit_points))
print("Mana: {}".format(generico.mana))
print("Vocation: {}".format(generico.vocation))
print("Su hechizo es: {}".format(generico.lanzar_hechizo()))
print(("Su velocidad de moverse es:
{}".format(generico.movement_speed)))
```

Y la salida va a ser

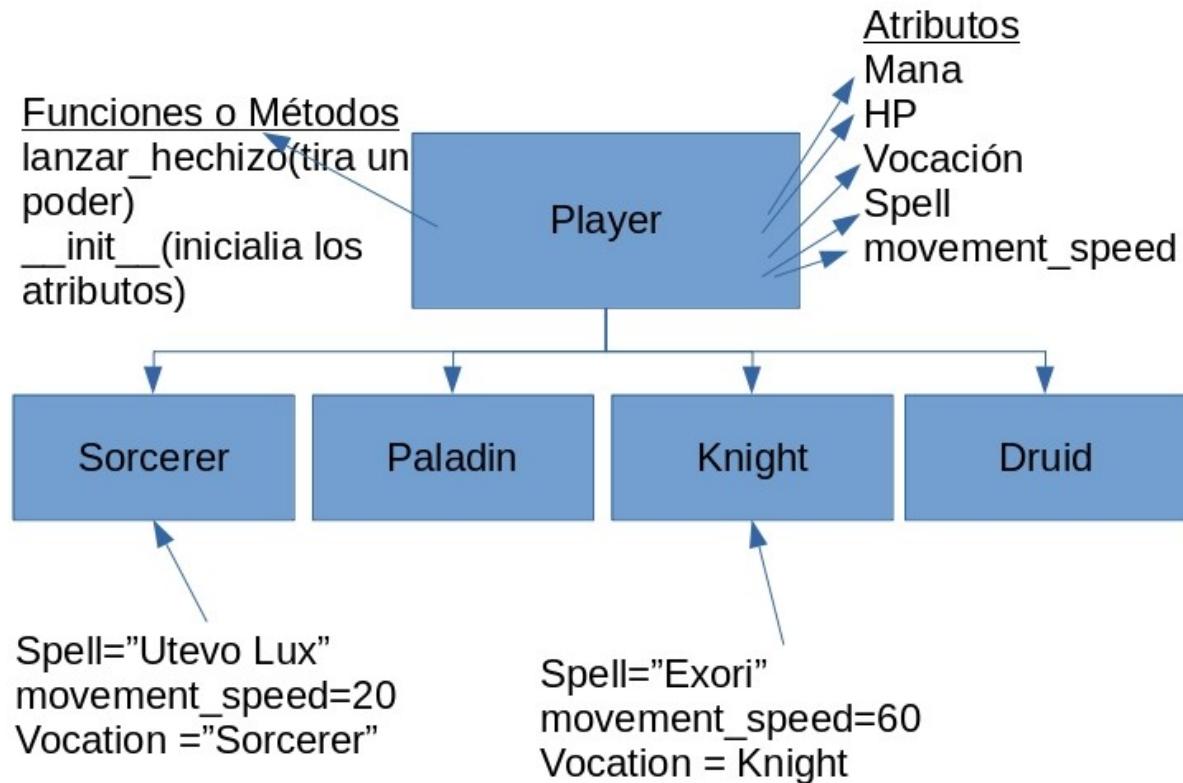
```
jenifer@pop-os:~/Documentos/qbit/Python_MMO$ python3 main.py
Sorcerer:
HP: 40
Mana: 90
Vocation: Sorcerer
Su hechizo es: Utevo Lux
Su velocidad de moverse es: 30
#####
Personaje Generico:
HP: 50
Mana: 50
Vocation: No vocation
Su hechizo es: Puff
Su velocidad de moverse es: 50
```

En síntesis, cuando quiero llamar a un método de una clase en el programa principal:

```
import <mi_libreria>
<mi_variable>=<mi_libreria>.<Nombre_de_la_clase>(<parámetros>)
```

## Herencia II:

En el contexto de nuestro MMO, la estructura va a ser:



Entonces cada subclase de la clase player va a tener cada una sus atributos y no se van a mezclar entre ellos, si hago un knight, no va a usar utevo lux porque es de sorcerer.

En código:

En player.py

```

class Player():
    vocation="No vocation"
    hechizo = "Puff"
    movement_speed = 50
    def __init__(self, **kargs):
        self.hit_points = kargs.get("hit_points",50)
        self.mana = kargs.get("mana",50)
    def lanzar_hechizo(self):
        return self.hechizo

#####Acá undica que hereda todos los atributos#####
#####Y metodos de Player #####
class Sorcerer (Player):
    vocation = "Sorcerer"
    hechizo = "Utevo Lux"
    movement_speed = 30

```

```
class Knight(Player):
    vocation = "Knight"
    hechizo = "Exori"
    movement_speed = 60
```

Mientras que en el programa principal:

```
import player

generico = player.Player(hit_points = 50, mana = 50)
sorcerer = player.Sorcerer(hit_points = 40, mana = 90)
caballero = player.Knight(hit_points = 100, mana = 40)
print("Sorcerer:")
print("HP: {}".format(sorcerer.hit_points))
print("Mana: {}".format(sorcerer.mana))
print("Vocation: {}".format(sorcerer.vocation))
print("Su hechizo es: {}".format(sorcerer.lanzar_hechizo()))
print(("Su velocidad de moverse es:
{}").format(sorcerer.movement_speed))

print ("#####")
print("Personaje Generico:")
print("HP: {}".format(generico.hit_points))
print("Mana: {}".format(generico.mana))
print("Vocation: {}".format(generico.vocation))
print("Su hechizo es: {}".format(generico.lanzar_hechizo()))
print(("Su velocidad de moverse es:
{}").format(generico.movement_speed))

print ("#####")
print("Caballero:")
```

Y el resultado de todo ésto es:

```
jenifer@pop-os:~/Documentos/qbit/Python_MM0$ python3 main.py
Sorcerer:
HP: 40
Mana: 90
Vocation: Sorcerer
Su hechizo es: Utevo Lux
Su velocidad de moverse es: 30
#####
Personaje Generico:
HP: 50
Mana: 50
Vocation: No vocation
Su hechizo es: Puff
Su velocidad de moverse es: 50
#####
Caballero:
HP: 100
Mana: 40
Vocation: Knight
Su hechizo es: Exori
Su velocidad de moverse es: 60
```

## Herencia II:

Ahora definimos todos los atributos de todas las clases:

```
class Player():
    vocation="No vocation"
    hechizo = "Puff"
    movement_speed = 50
    def __init__(self, **kargs):
        self.hit_points = kargs.get("hit_points",50)
        self.mana = kargs.get("mana",50)
    def lanzar_hechizo(self):
        return self.hechizo

#####Acá undica que hereda todos los atributos#####
#####Y metodos de Player #####
class Sorcerer (Player):
    vocation = "Sorcerer"
    hechizo = "Utevo Lux"
    movement_speed = 30

class Knight(Player):
    vocation = "Knight"
    hechizo = "Exori"
    movement_speed = 60

class Druid(Player):
```

```

vocation = "Knight"
hechizo = "Exura Sio"
movement_speed = 20

class Paladin(Player):
    vocation = "Paladin"
    hechizo = "Exana"
    movement_speed = 80

```

### Nota sobre .format

format es una función que puede tomar tantos argumentos como llaves encuentre en la cadena, por ejemplo, en nuestro programa

```

pal = player.Paladin(hit_points=60, mana = 60)
pal=player.Paladin(hit_points=60, mana = 60)
print("El {} tiene {} HP, {} mana, puede castear {} y corre con una
velocidad de {}".format(pal.vocation,
pal.hit_points,
pal.mana,
pal.lanzar_hechizo(),
pal.movement_speed))

```

Lo cual imprime en pantalla:

```
El Paladin tiene 60 HP, 60 mana, puede castear Exana y corre con una
velocidad de 80
```

### Método \_\_str\_\_ (va con doble guión bajo)

El método \_\_str\_\_ es un método especial de Python que decide cómo se van a imprimir los atributos de una clase y lo que retorna es una cadena.

En nuestro ejemplo del MMO nos queda

```

class Player():
    vocation="No vocation"
    hechizo = "Puff"
    movement_speed = 50
    def __init__(self, **kargs):
        self.hit_points = kargs.get("hit_points",50)
        self.mana = kargs.get("mana",50)

    def __str__(self):
        return "El {} tiene {} HP, {} mana, puede castear {} y corre
con una velocidad de {}".format(self.vocation,self.hit_points,
self.mana,self.lanzar_hechizo(),self.movement_speed)

    def lanzar_hechizo(self):
        return self.hechizo

```

En nuestro main.py si lo modificamos:

```
import player

generico = player.Player(hit_points = 50, mana = 50)
sorcerer = player.Sorcerer(hit_points = 40, mana = 90)
caballero = player.Knight(hit_points = 100, mana = 40)
paladin = player.Paladin(hit_points = 60, mana = 60)
druida = player.Druid(hit_points = 50, mana = 90)

print(generico)
print ("#####")

print(sorcerer)
print ("#####")

print(caballero)
print ("#####")

print(paladin)
print ("#####")

print(druida)
print ("#####")
```

La salida es:

```
jenifer@pop-os:~/Documentos/qbit/Python_MM$ python3 main.py
El No vocation tiene 50 HP, 50 mana, puede castear Puff y corre con una velocidad de 50
#####
El Sorcerer tiene 40 HP, 90 mana, puede castear Utevo Lux y corre con una velocidad de 30
#####
El Knight tiene 100 HP, 40 mana, puede castear Exori y corre con una velocidad de 60
#####
El Paladin tiene 60 HP, 60 mana, puede castear Exana y corre con una velocidad de 80
#####
El Druida tiene 50 HP, 90 mana, puede castear Exura Sio y corre con una velocidad de 20
#####
```

¿Que pasa si no declaramos el método `__str__`? Pues print hace lo que `__str__` haría por defecto, imprime los objetos y su locación en memoria:

```
jenifer@pop-os:~/Documentos/qbit/Python_MM$ python3 main.py
<player.Player object at 0x7f69afb39f98>
#####
<player.Sorcerer object at 0x7f69afb39fd0>
#####
<player.Knight object at 0x7f69afb3f080>
#####
<player.Paladin object at 0x7f69afb567b8>
#####
<player.Druid object at 0x7f69afb568d0>
#####
```

### Obteniendo un nombre de usuario (entradas y salidas):

El tema es que, en nuestro caso, se quieren implementar un juego, tenemos que darle al usuario que elija el nombre de su jugador. Para eso, lo que hacemos es crearle un nuevo atributo a la clase Player (dados que todos los players tienen un nombre)

```
class Player():
    vocation="No vocation"
    hechizo = "Puff"
    movement_speed = 50
    def __init__(self, **kargs):
        self.hit_points = kargs.get("hit_points",50)
        self.mana = kargs.get("mana",50)
        self.nombre = input("Escribe el nombre de tu jugador: ")

    def __str__():
        return "{} es un {} tiene {} HP, {} mana, puede castear {} y corre con una velocidad de {}".format(self.nombre,
                                                    self.vocation,
                                                    self.hit_points,
                                                    self.mana,
                                                    self.lanzar_hechizo(),
                                                    self.movement_speed)

    def lanzar_hechizo(self):
        return self.hechizo.
```

`self.nombre = input("Escribe el nombre de tu jugador: ")` hace que le pregunte al usuario los nombres del jugador.

Como el programa principal ahora tiene 6 variables declaradas, una para cada clase, va a pedir 6 nombres y luego va a imprimir con el formato dado en `__str__`

```
jenifer@pop-os:~/Documentos/qbit/Python_MMO$ python3 main.py
Escribe el nombre de tu jugador: Jenifer
Escribe el nombre de tu jugador: Aldo
Escribe el nombre de tu jugador: Max
Escribe el nombre de tu jugador: Ana
Escribe el nombre de tu jugador: Diego
Jenifer es un No vocation tiene 50 HP, 50 mana, puede castear Puff y corre con una velocidad de 50
#####
Aldo es un Sorcerer tiene 40 HP, 90 mana, puede castear Utevo Lux y corre con una velocidad de 30
#####
Max es un Knight tiene 100 HP, 40 mana, puede castear Exori y corre con una velocidad de 60
#####
Ana es un Paladin tiene 60 HP, 60 mana, puede castear Exana y corre con una velocidad de 80
#####
Diego es un Druida tiene 50 HP, 90 mana, puede castear Exura Sio y corre con una velocidad de 20
#####
```

## Herencia y modificación de métodos

Así como las instancias de los atributos de una clase, pueden ser cambiados por subclases, también pueden sobre\_escribir métodos.

Ésto solo se puede hacer para cada subclase y solo modifican el método dentro de esa subclase.

Para poder hacer ésto, se define dentro de la subclase, un método llamado igual que el de la clase padre.

Ésto lo que hace es que, si al correr el programa, se llama a esa función, y la instancia es correcta, va a tomar como prioridad el método de la clase hija.

## Reto OOP

El reto consiste en programar una clase nueva que constituya los NPCS del juego

## Fechas y horas en Python

### Introducción

Todos los desarrolladores o por lo menos el 90 por ciento de nosotros nos tenemos que enfrentar en algún punto de nuestras carreras a la fecha y la hora.

Ésto se debe a que en la mayoría de los lenguajes de programación hay una forma un poco diferente de desplegar y trabajar con la fecha y la hora en algunos lenguajes de programación del lado del servidor como PHP. Tenemos que trabajar con la hora utilizando Unix tan importantes en algunos lenguajes.

En Python tenemos librerías que nos ayudan a trabajar con la fecha en este lenguaje, tenemos `DaltaTime` y tenemos `daytime`. A veces también tenemos que ver cómo desplegamos la fecha y la hora en nuestras entradas de nuestros blogs, en fin.

Es por todo ésto, que es importante que aprendamos cómo manejar la fecha y la hora en Python y los métodos y las clases, cosa de para usarla adecuadamente.

## Fecha y Hora

### **import datetime**

Este import es importante, siempre que se trabaje con fechas y horas se debe importar la librería `datetime`.

Para saber los módulos que contiene toda la una librería:

```
dir(<nombre_de_la_librería>)
```

en nuestro caso, en la consola:

```
>>> import datetime
>>> dir(datetime)
['MAXYEAR', 'MINYEAR', '__builtins__', '__cached__', '__doc__',
 '__file__', '__loader__', '__name__', '__package__', '__spec__',
 'date', 'datetime', 'datetime_CAPI', 'sys', 'time', 'timedelta',
 'timezone', 'tzinfo']
```

Acá vemos que tiene varios módulos como datetime, timezone, deltatime, etc. Esos 3 son los que más se usan.

### Función now

Es una función que permite saber la hora exacta en el momento que se ejecuta y en la máquina que se está ejecutando. Now está dentro del módulo datetime:

```
>>> datetime.datetime.now()
datetime.datetime(2019, 9, 5, 8, 28, 52, 457690)
```

En nuestro ejemplo anterior: nos devuelve que es 5/9/2019, son las 8 hs 28 min52 seg y 457690 microsegundos.

Si en vez de imprimir en pantalla, se guarda en una variable, luego podemos acceder a ella dado que datetime.datetime.now, lo que devuelve es un objeto de la clase datetime, por lo cual, podemos acceder a sus valores llamando a los métodos de dicha clase:

```
>>> import datetime
>>> now = datetime.datetime.now()
>>> now.day
5
>>> now.month
9
>>> now.year
2019
>>> now.hour
8
>>> now.minute
38
>>> now.second
14
```

Pero ahora, si quiero hacer un cambio en las fechas, se usa el método `.replace`

```
>>> now.replace(minute = 0)
```

```
datetime.datetime(2019, 9, 5, 8, 0, 14, 136526)
```

Con ésto pasa a que la hora es 8:00 en vez de 8:38, pero el resto de los valores sigue igual.

```
>>> now
```

```
datetime.datetime(2019, 9, 5, 8, 38, 14, 136526)
```

El tema es que, como vemos aquí, ésto no modifica la variable now, solo devolvió una copia.

Para cambiar efectivamente la variable:

```
>>> now = now.replace(minute=0, second=0, microsecond = 0)
```

### timedelta

Al hacer de nuevo todo lo anterior:

```
>>> import datetime
>>> now = datetime.datetime.now()
>>> now
datetime.datetime(2019, 9, 5, 19, 44, 9, 448201)
>>> now = now.replace(minute= 1, second = 1, microsecond=1)
>>> now
datetime.datetime(2019, 9, 5, 19, 1, 1, 1)
```

Si se calcula la diferencia entre la fecha actual y la fecha que guardamos en now

```
>>> tiempo_transcurrido = datetime.datetime.now() - now
```

Al hacer:

```
>>> tiempo_transcurrido
datetime.timedelta(seconds=2734, microseconds=275321)
```

Nos da un objeto de la clase timedelta, que es la diferencia entre 2 fechas y horas del sistema

Para calcular los minutos de diferencia:

```
>>> minutos = tiempo_transcurrido.seconds/60
>>> minutos
45.56666666666667
>>> minutos = round(minutos)
>>> minutos
46
```

Apreciaciones:

- La función round redondea hacia arriba
- .seconds devuelve un int

Pero ésta no es la única funcionalidad de datetime, también se usa para adelantar o atrasar fechas y horas:

```
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import datetime
>>> now = datetime.datetime.now()
>>> now + datetime.timedelta(days = 5)
datetime.datetime(2019, 9, 10, 20, 22, 31, 415716)
>>> now
datetime.datetime(2019, 9, 5, 20, 22, 31, 415716)
>>> now - datetime.timedelta(minutes =3, hours =2, days = 4)
datetime.datetime(2019, 9, 1, 18, 19, 31, 415716)
```

También es válido poner por ejemplo:

```
>>>now + datetime.timedelta(days= -4)
```

El número negativo hace que la fecha se atrasé.

### .date y .time

Son dos métodos que se usan para saber solo la fecha y la hora.

```
>>> now.date()
datetime.date(2019, 9, 5)
>>> now.time()
datetime.time(20, 22, 31, 415716)
```

## Today, Now y Combine

### .today y .now

Si hacemos:

```
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import datetime
>>> now=datetime.datetime.now()
>>> hoy = datetime.datetime.today()
>>> now
datetime.datetime(2019, 9, 6, 8, 9, 33, 955010)
>>> hoy
datetime.datetime(2019, 9, 6, 8, 10, 1, 112685)
```

Vemos que ambos métodos devuelven un objeto de la clase datetime, pero la diferencia es que a .now() solo se le puede pasar una zona horaria mientras que a .today() se le puede pasar más de una.

O sea, .today puede devolver más de una hora, si se quiere por ejemplo, saber qué hora es en distintos puntos del planeta.

### .combine

Si ponemos

```
>>>datetime.time()
```

devuelve:

```
datetime.time(0, 0)
```

O sea, las horas valen 0 y los minutos valen 0.

Para poner la fecha y la hora de hoy, se emplea `.combine`:

```
>>> fecha_hoy = datetime.datetime.combine(datetime.datetime.today(), datetime.time())
>>> fecha_hoy
datetime.datetime(2019, 9, 6, 0, 0)
```

O sea `.combine` tomó 2 objetos de tipo `.datetime.datetime(today y time)`, y los combinó modificando los valores de minutos y segundos de `today` por los de `time`, que son 0 horas y 0 minutos.

Para comprobar ésto y acceder a cada valor se usa:

```
>>> fecha_hoy.year
2019
>>> fecha_hoy.month
9
>>> fecha_hoy.day
6
>>> fecha_hoy.minute
0
>>> fecha_hoy.second
0
```

## Formateando fechas

Para darle formato a las fechas se usa el metodo `.strftime`, que lo que hace es devolver un string, la forma de usarla es:

`<nombre_de_la_instancia>.strftime("<argumentos que quiero que se impriman>")`

Para formatear, los comodines (%) están en la siguiente tabla:

Presets	Date	Time	
Date			
%m/%d/%Y	06/05/2013		
%A, %B %e, %Y	Sunday, June 5, 2013		
%b %e %a	Jun 5 Sun		
Time			
%H:%M	23:05		
%I:%M %p	11:05 PM		
Used by Ruby, UNIX date, and many more.			
	Date	Time	
	%a	Sun	Weekday
	%A	Sunday	
	%W	0..6 (Sunday is 0)	
	%y	13	Year
	%Y	2013	
	%b	Jan	Month
	%B	January	
	%m	01..12	
	%d	01..31	Day
	%e	1..31	
	%l	1	Hour
	%H	00..23	24h Hour
	%I	01..12	12h Hour
	%M	00..59	Minute
	%S	00..60	Second
	%p	AM	AM or PM
	%Z	+08	Time zone
	%j	001..366	Day of the year
	%%	%	Literal % character

Para más info: <http://strftime.org/>

Por ejemplo:

```
>>> now = datetime.datetime.now()  
>>> now.strftime("%B %d, %Y, %H:%M")  
'September 06, 2019, 10:58'
```

### strptime

Por otra parte, en python se puede hacer lo inverso, es decir, a partir de una cadena, generar un objeto del tipo datetime:

```
>>> now = datetime.datetime.now()  
>>> now.strftime("%B %d, %Y, %H:%M")  
'September 06, 2019, 10:58'  
>>> fecha_cadena = "06/09/2019"  
>>> fecha_formateada = datetime.datetime.strptime(fecha_cadena, "%d/%m/%Y")  
>>> fecha_formateada  
datetime.datetime(2019, 9, 6, 0, 0)
```

O sea, en forma general

`datetime.datetime.strptime(<cadena_a_convertir>, "<indicacion_de_los_comodines_y_los_separadores>").`

Otro ejemplo:

```
>>> fecha_cadena = "Hoy es 06 del mes 9 de 2019"  
>>> fecha_formateada2 = datetime.datetime.strptime(fecha_cadena, "Hoy es %d del mes %m de %Y")  
>>> fecha_formateada2  
datetime.datetime(2019, 9, 6, 0, 0)
```

### TimeZone:

Las zonas horarias son algo muy difícil de lo que tenemos que encargarnos los desarrolladores. Nunca se sabe a ciencia cierta a priori en qué zona horaria se encuentra tu usuario. También puede que hayan configurado mal su teléfono, puede que el dato que están recibiendo de la web estén mal. O sea, es algo muy problemático. En general, si está bien que estemos concientes de cómo se manejan las zonas horarias con esta clase de baile.

En Python primero importamos datetime. Para editar cómo creamos una zona horaria primero vamos a poner una variable llamada **central\_time** que va a contener la zona horaria:

```
>>> import datetime  
>>> central_time = datetime.timezone(datetime.timedelta(hours= -3))
```

El -3 es porque Uruguay está a -3 horas de UTC y entonces **central\_time** es un objeto de la clase **timezone** (ver tercer renglón del código)

A continuación vamos a crear un Pacific Time:

```
>>> pacific_time = datetime.timezone(datetime.timedelta(hours= -8))  
>>> pacific_time  
datetime.timezone(datetime.timedelta(days=-1, seconds=57600))
```

Pacific time es un objeto de tipo timezone.

Con ésto ya tenemos una timezone y el cómo la usamos para darle la zona horaria a nuestros objetos de fecha y hora.

Recordarás que en anteriormente el datetime.now nos da la opción de asignarle una zona horaria, entonces y en lugar de dejarlo vacío, vamos a ponerle la hora en la que nos encontramos

Yo estoy en Uruguay así que le pondré:

```
>>> import datetime
>>> central_time = datetime.timezone(datetime.timedelta(hours= -3))
>>> pacific_time = datetime.timezone(datetime.timedelta(hours = +8))
>>> estern_time = datetime.timezone(datetime.timedelta(hours = -5))
>>> now = datetime.datetime.now(central_time)
>>> now
datetime.datetime(2019, 9, 7, 26, 38, 816010, tzinfo=datetime.timezone(datetime.timedelta(days=-1, seconds=75600)))
```

Y ahora si vemos que es lo que contiene la variable now vemos que nos da la información de la zona horaria, en nuestro caso, en la zona horaria en la que estamos es un día anterior respecto a la hora UTC, lo cual, para nosotros eso es muy útil si queremos por ejemplo convertir esta zona horaria en otra hora.

Digamos que ya sabemos que son las 7:26 en Uruguay y quiero saber la hora en San Francisco porque quiero ir allá. Lo único que tendríamos que hacer en vez de andar calculando la diferencia de horas y todo eso es poner todo junto:

**<variable>.astimezone(<zona\_horaria>)**

Por ejemplo:

```
>>> now.astimezone(pacific_time)
datetime.datetime(2019, 9, 7, 18, 26, 38, 816010, tzinfo=datetime.timezone(datetime.timedelta(seconds=28800)))
>>> now
datetime.datetime(2019, 9, 7, 26, 38, 816010, tzinfo=datetime.timezone(datetime.timedelta(days=-1, seconds=75600)))
```

La fecha actual en pacific\_time es 7/9/2019 y son las 18:26, mientras que acá es 7/9/2019 pero son las 7:26

Nota, no entendí el ejemplo porque si en Uruguay está a UTC -3 y Pacific time está a UTC -5, deberían ser 2 horas de diferencia. Acá creo que hay algo suspicious.



### Reloj del mundo

En ésta sección vamos a programar el reloj del mundo:

Va a ser un programa que pueda interactuar el usuario , el cual va a tener las siguientes condiciones:

- Va a recibir del usuario la hora en qué timezone está
- Devolver los relojes que ya tenga programados a pedido del usuario

Entonces lo primero que va a hacer nuestro main program es importar datetime y decirle qué operaciones debe hacer:

```
import datetime
```

```

print ("Bienvenido al reloj del mundo")
print ("Estas son las acciones que puedes realizar:")
print ("1) Ver la hora")
print ("2) Ver la fecha y hora")
print ("3) Ver la hora en Nueva York")
print ("4) Ver la hora en San Francisco")
print ("5) Ver las instrucciones nuevamente")
print ("6) Salir")

otra_vez = True

while otra_vez:
    operacion = input("Elige la opción con número:")
    if operacion == "1":
        print(datetime.datetime.now().time()) #esto es porque solo queremos la hora
    
```

```
elif operacion == "2":  
    print(datetime.datetime.now())  
elif operacion == "3":  
    print("Hora en Nueva York: ") #esto va a ser modificado  
elif operacion == "4":  
    print("Hora en San Francisco: ") #esto va a ser modificado  
elif operacion == "5":  
    print ("Ver instrucciones")  
elif operacion == "6":  
    otra_vez = False  
else:  
    print("No se reconoce la operación")  
  
print ("Gracias por usar nuestro programa")
```

Lo que interesa en ésta fase de desarrollo es que la interfaz con el usuario funcione correctamente.

Importante:

Una de las “contras” que tiene estar trabajando con timezones es que .now() despliega la hora del sistema donde se esté ejecutando python. Si se hace desde el IDE, despliega la hora de la máquina en la que estamos trabajando, mientras que, si se usan servidores remotos, como es el caso de repl.it por ejemplo, el .now despliega la hora en ése servidor

Po ortra parte, si miramos con atención:

```
>>> import datetime  
>>> datetime.datetime.now().time()  
datetime.time(12, 6, 2, 415907)  
>>> print(datetime.datetime.now())  
2019-09-07 12:06:24.741116
```

La diferencia entre un resultado y otro es que **print** tiene un método **\_\_str\_\_** que le permite darse cuenta de que lo que se le ingresó fué una fecha y le da el formato correspondiente.

### Definiciones de las funciones:

#### **ver\_reloj**

Ésta función va a constar de todo el código que escribimos dentro del while:

```
def ver_reloj():  
    otra_vez = True  
    while otra_vez:  
        operacion = input("Elige la opción con número:")  
        if operacion == "1":  
            print(datetime.datetime.now().time()) #esto es porque  
solo queremos la hora  
        elif operacion == "2":  
            print(datetime.datetime.now())  
        elif operacion == "3":
```

```

        print("Hora en Nueva York: ") #esto va a ser modificado
    elif operacion == "4":
        print("Hora en San Francisco: ") #esto va a ser
modificado
    elif operacion == "5":
        print ("Ver instrucciones")
    elif operacion == "6":
        otra_vez = False
    else:
        print("No se reconoce la operación")

```

### **ver\_instrucciones**

En ésta función van a ir todos los prints hechos hasta ahora:

```

def ver_instrucciones():
    print ("Estas son las acciones que puedes realizar:")
    print ("1) Ver la hora")
    print ("2) Ver la fecha y hora")
    print ("3) Ver la hora en Nueva York")
    print ("4) Ver la hora en San Francisco")
    print ("5) Ver las instrucciones nuevamente")
    print ("6) Salir")

```

Y se la va a llamar en 3 casos:

- Al iniciar el programa
- Al no reconocer la instrucción dada
- cuando el usuario quiera desplegar las instrucciones

### **función ver\_hora**

Dado que pueden haber diferentes zonas horarias, se define ésta función para que muestre la hora actual en Uruguay, lo cual hace que el código se independice de la hora del servidor un poco:

```

def ver_hora():
    timezone= datetime.timezone(datetime.timedelta(hours = -3)) #esto
es porque la hora en uruguay es
    hora_actual=datetime.datetime.now(timezone).time()
    print(hora_actual)

```

Y se va a cambiar el print de la opción 2 en ver\_reloj, por la invocación a ésta función:

```

if operacion == "1":
    ver_hora() #esto es porque solo queremos la hora

```

### **función ver\_fecha\_y\_hora**

El concepto es el mismo, solo que además de la hora, imprime la fecha

```

def ver_fecha_y_hora():
    timezone= datetime.timezone(datetime.timedelta(hours = -3)) #esto
es porque la hora en uruguay es

```

```

hora_actual=datetime.datetime.now(timezone).time()
fecha_actual = datetime.datetime.now(timezone).date()
print("Fecha actual:")
print(fecha_actual)
print("Hora actual:")
print(hora_actual)

```

### Formatear la salida de las funciones anteriores

Para eso se va a crear una variable auxiliar llamada formato.

En el caso de ver\_hora():

```

def ver_hora():
    formato = "%H:%M:%S"
    timezone= datetime.timezone(datetime.timedelta(hours = -3)) #esto
    es porque la hora en uruguay es
    hora_actual=datetime.datetime.now(timezone).time()
    hora_formateada = hora_actual.strftime(formato)
    print("La hora exacta es: {}".format(hora_formateada))

```

En el caso de fecha y hora:

```

def ver_fecha_y_hora():
    formato_hora = "%H:%M:%S"
    formato_fecha = "%d/%m/%Y"

    timezone= datetime.timezone(datetime.timedelta(hours = -3))
    hora_actual=datetime.datetime.now(timezone).time()
    fecha_actual = datetime.datetime.now(timezone).date()

    fecha_formateada = fecha_actual.strftime(formato_fecha)
    hora_formateada = hora_actual.strftime(formato_hora)

    print("Fecha actual: {}".format(fecha_formateada))
    print("Hora actual: {}".format(hora_formateada))

```

O sea, se usa la función .strftime para modificar al formato que queremos.

`fecha_actual.strftime(formato_fecha)` siendo el formato, `formato_fecha = "%d/%m/%Y"`

### Hora para San Francisco y Nueva York:

El despliegue de datos es el mismo, la única diferencia radica en `datetime.timedelta(hours = -3)`, en particular en la variable `hours` pues para New York es -5 y Para San Francisco es -8  
La forma inteligente de hacerlo es pasarle un parámetro a ver\_hora que sea la diferencia de horas:

```

def ver_hora(zona_horaria):
    if zona_horaria == -3:
        geo = "Uruguay"
    elif zona_horaria == -5:

```

```
geo = "New York"
elif zona_horaria == -8:
    geo = "San Fransisco"
else:
    print("me falla la zona horaria")
formato = "%H:%M:%S"
timezone= datetime.timezone(datetime.timedelta(hours =
zona_horaria))
hora_actual=datetime.datetime.now(timezone).time()
hora_formateada = hora_actual.strftime(formato)
print("La hora exacta en {} es: {}".format(geo,hora_formateada))
```

### El reto:

El reto en ésta sección consiste en cambiar el formato de la fecha, y elegir 2 zonas horarias más:

Challenge Accepted:

```
import datetime
```

```
print ("Bienvenido al reloj del mundo")
```

```
def ver_instrucciones():
    print ("Estas son las acciones que puedes realizar:")
    print ("1) Ver la hora")
    print ("2) Ver la fecha y hora")
    print ("3) Ver la hora en Nueva York")
    print ("4) Ver la hora en San Francisco")
    print ("5) Ver la hora en Siria")
    print ("6) Ver la hora en Japón")
    print ("7) Ver las instrucciones nuevamente")
    print ("8) Salir")
```

```
def ver_hora(zona_horaria):
    if zona_horaria == -3:
        geo = "Uruguay"
    elif zona_horaria == -5:
        geo = "New York"
    elif zona_horaria == -8:
        geo = "San Fransisco"
    elif zona_horaria== +3:
        geo = "Siria"
    elif zona_horaria == +9:
        geo = "Japón"
    formato = "%I:%M %p"
    timezone= datetime.timezone(datetime.timedelta(hours =
zona_horaria))
    hora_actual=datetime.datetime.now(timezone).time()
```

```
hora_formateada = hora_actual.strftime(formato)
print("La hora exacta en {} es: {}".format(geo,hora_formateada))

def ver_fecha_y_hora():
    formato_hora = "%I:%M %p"
    formato_fecha = "%a, %B %e, %Y"
    timezone= datetime.timezone(datetime.timedelta(hours = -3)) #esto
es porque la hora en uruguay es
    hora_actual=datetime.datetime.now(timezone).time()
    fecha_actual = datetime.datetime.now(timezone).date()
    fecha_formateada = fecha_actual.strftime(formato_fecha)
    hora_formateada = hora_actual.strftime(formato_hora)
    print("Fecha actual: {}".format(fecha_formateada))
    print("Hora actual: {}".format(hora_formateada))

def ver_reloj():
    otra_vez = True
    while otra_vez:
        ver_instrucciones()
        operacion = input("Elige la opción con número: ")
        if operacion == "1":
            ver_hora(-3) #esto es porque solo queremos la hora local
        elif operacion == "2":
            ver_fecha_y_hora()
        elif operacion == "3":
            ver_hora(-5) #Ésto es porque la zona horaria en New York
es -5
        elif operacion == "4":
            ver_hora(-8) #Ésto es porque la zona horaria en San
Francisco es -5
        elif operacion == "5":
            ver_hora(+3) #Ésto es porque la zona horaria en Siria es
-5
        elif operacion == "6":
            ver_hora(+9) #Ésto es porque la zona horaria en Japon -5
        elif operacion == "7":
            ver_instrucciones()
        elif operacion == "8":
            otra_vez = False
        else:
            print("No se reconoce la operación")
            ver_instrucciones()

#####Acá cuando imprimió lo que el usuario quería#####
#####Le pregunta si quiere continuar o no#####
```

```
#####Si dice que si, vuelve a desplegar las instrucciones
#####porque entra a un nuevo ciclo de while#####
#####Si no quiere seguir, pone que no#####
#####Si pone otra cosa, le indica que la operación#####
#####No es válida y lo lleva de nuevo al while#####
    if operacion != "8" and operacion != "7":
        continuar = input("¿Deseas continuar? Si/No: ")
        if continuar.lower() == "no":
            otra_vez = False
        elif continuar.lower() == "si":
            otra_vez == True
        else:
            print("No se reconoce la operación")

ver_reloj()
print ("Gracias por usar nuestro programa")
```

## Manejo de archivos y Expresiones Regulares en Python

### Introducción

En ésta sección se va a aprender a lidiar con archivos en python y cómo buscar lo que queramos, siempre y cuando el archivo a analizar siga un cierto patrón, por ejemplo, extraer de un archivo de listas de correos electrónicos, aquellos correos que sean de gmail, o por ejemplo, en un archivo con nombres y números de teléfonos, extraer los que tengan determinada característica.

El manejo de expresiones regulares depende del tipo de archivo y de lo que se está buscando, lo cual hace que la forma de codificar las expresiones regulares a buscar sea muy personal, es decir, depende del estilo de programación de la persona y de la situación bajo la que se usa el código escrito.

### Cómo leer archivos en python

Lo más fácil para aprender a buscar expresiones regulares es que el formato del archivo de entrada sea en texto plano.

Tip:

Existe una página interactiva llamada [regex.com](http://regex.com) donde se pueden testear expresiones regulares

Si se guarda el siguiente texto en un archivo .txt:

RegExr was created by gskinner.com, and is proudly hosted by Media Temple.

Edit the Expression & Text to see matches. Roll over matches or the expression for details. PCRE & Javascript flavors of RegEx are supported.

The side bar includes a Cheatsheet, full Reference, and Help. You can also Save & Share with the Community, and view patterns you create or favorite in My Patterns.

Explore results with the Tools below. Replace & List output custom results. Details lists capture groups. Explain describes your expression in plain English.

Sample text for testing:

```
abcdefghijklmnopqrstuvwxyz ABCDEFGHYJKLMNOPQRSTUVWXYZ  
0123456789 _+-.,!@#$^&*();\|<>  
12345 -98.7 3.141 .6180 9,000 +42  
555.223.4567 +1.(800)-(555)-(2468)  
foo@demo.net bar.ba@test.co.uk  
www.demo.com http://:foo.co.uk/  
http://regexr.com/  
https://media.sample.net/
```

Se puede hacer un programa main.py que acepte como entrada el texto de ese archivo texto.txt, busque y devuelva en su salida, solo lo que coincidió con la expresión regular.

Para eso, se debe importar la librería **re** que es la librería de Python para expresiones regulares:

```
import re
```

Y para abrir un archivo:

```
<variable> = open("<nombre_del_archivo>",  
encoding="<la_codificación_del_texto>")
```

Las comillas son obligatorias, y en caso de que el archivo no esté bajo el mismo directorio donde está el script de python, se debe incluir su dirección para que Python sepa donde buscarlo.

Ejemplo:

```
archivo= open("texto.txt", encoding ="UTF-8")
```

Por otra parte, el encoding se asegura de qué tabla de caracteres se está usando.

### .read

Ésta variable file es una referencia al archivo, no contiene su información. Para acceder a ella en nuestro ejemplo:

```
informacion= archivo.read()
```

### .close

Una vez empleado el archivo, para liberar el espacio en memoria, se debe cerrarlo. En nuestro ejemplo:

```
archivo.close()
```

### Código del programa:

```
import re  
archivo = open("texto.txt")
```

```
informacion= archivo.read()
archivo.close()
print(informacion)
```

Y al ejecutar ésto, nos imprime en pantalla lo que contiene la variable información, que es el contenido del archivo .txt

```
jenifer@pop-os:~/Documentos/qbit/Prog_Py$ python3 parser.py
RegExr was created by gskinner.com, and is proudly hosted by Media Temple.

Edit the Expression & Text to see matches. Roll over matches or the expression for details. PCRE & Javascript flavors of RegEx are supported.

The side bar includes a Cheatsheet, full Reference, and Help. You can also Save & Share with the Community, and view patterns you create or favorite in My Patterns.

Explore results with the Tools below. Replace & List output custom results. Details lists capture groups. Explain describes your expression in plain English.

Sample text for testing:
abcdefghijklmnopqrstuvwxyz ABCDEFGHYJKLMNOPQRSTUVWXYZ
0123456789 _+-.,!@#$^&*();\|<>
12345 -98.7 3.141 .6180 9,000 +42
555.223.4567 +1.(800)-(555)-(2468)
foo@demo.net bar.ba@test.co.uk
www.demo.com http://foo.co.uk/
http://regexr.com/
https://media.sample.net/
```

## Match y Search

### **.match**

Recibe como parámetro 2 cosas, la expresión a buscar y donde buscarla de la siguiente manera

```
re.match(r"<expresión_a_buscar>", <variable_o_cadena_donde_buscar>)
```

En nuestro ejemplo

```
print(re.match(r"abcdefghijklmnopqrstuvwxyz", informacion))
```

La r significa que es raw data, y en este caso devuelve la primer coincidencia exacta. Con ésta opción no es necesario hacer escape de caracteres especiales

(acá queda para encontrar en el debe porque a mi me devuelve none)

Además, match solo busca en la primera linea.

### **.search**

Busca en toda la variable en lugar de sólo en la primera linea

```
re.search(r"<expresión_a_buscar>", <variable_donde_buscar>)
```

## Caracteres y Expresiones Regulares

Se usan para encontrar patrones dentro de archivos, en particular de textos. Son útiles cuando las ocurrencias que se quieren encontrar se dan más de una vez, y lo que se quiere encontrar sigue un patrón, por ejemplo, encontrar dentro de un .txt direcciones de sitios web, números de teléfono. Las expresiones regulares se emplean para encontrar éstos patrones.

Secuencia de escape	Significado
\n	Nueva línea (new line). El cursor pasa a la primera posición de la línea siguiente.
\t	Tabulador. El cursor pasa a la siguiente posición de tabulación.
\\\	Barra diagonal inversa
\v	Tabulación vertical.
\ooo	Carácter ASCII en notación octal.
\xhh	Carácter ASCII en notación hexadecimal.
\xhhhh	Carácter Unicode en notación hexadecimal.

### Clases de caracteres

Se pueden especificar clases de caracteres encerrando una lista de caracteres entre corchetes [], la que encontrará uno cualquiera de los caracteres de la lista. Si el primer símbolo después del "[" es "^", la clase encuentra cualquier carácter que no está en la lista.

### Metacaracteres

Los metacaracteres son caracteres especiales que son la esencia de las expresiones regulares. Como son sumamente importantes para entender la sintaxis de las expresiones regulares y existen diferentes tipos que hacen match con:

Metacaracter	Descripción
^	inicio de línea.
\$	fin de línea.
\A	inicio de texto.
\Z	fin de texto.
.	cualquier carácter en la línea.
\b	encuentra límite de palabra.
\B	encuentra distinto a límite de palabra.
\w	un carácter alfanumérico (incluye " _").
\W	un carácter no alfanumérico.
\d	un carácter numérico.
\D	un carácter no numérico.
\s	cualquier espacio (lo mismo que [\t\n\r\f]).
\S	un no espacio.

Existe otro tipo de metacaracteres que son los iteradores. Usando estos metacaracteres se puede especificar el número de ocurrencias del carácter previo, de un metacaracter o de una subexpresión. Ellos son:

Metacaracter	Descripción
*	cero o más, similar a {0,}.
+	una o más, similar a {1,}.
?	cero o una, similar a {0,1}.
{n}	exactamente n veces.
{n,}	por lo menos n veces.
{n,m}	por lo menos n pero no más de m veces.
*?	cero o más, similar a {0,}?.
+?	una o más, similar a {1,}?.
??	cero o una, similar a {0,1}?.
{n}?	exactamente n veces.
{n,}?	por lo menos n veces.
{n,m}?	por lo menos n pero no más de m veces.

En estos metacaracteres, los dígitos entre llaves de la forma {n,m}, especifican el mínimo número de ocurrencias en n y el máximo en m.

En suma:

- `\t` — Representa un tabulador.
- `\r` — Representa el "retorno de carro" o "regreso al inicio" o sea el lugar en que la línea vuelve a iniciar.
- `\n` — Representa la "nueva línea" el carácter por medio del cual una línea da inicio. Es necesario recordar que en **Windows** es necesaria una combinación de `\r\n` para comenzar una nueva línea, mientras que en **Unix** solamente se usa `\n` y en **Mac OS** clásico se usa solamente `\r`.
- `\a` — Representa una "campana" o "beep" que se produce al imprimir este carácter.
- `\e` — Representa la tecla "Esc" o "Escape"
- `\f` — Representa un salto de página
- `\v` — Representa un tabulador vertical
- `\x` — Se utiliza para representar caracteres **ASCII** o ANSI si conoce su código. De esta forma, si se busca el símbolo de derechos de autor y la fuente en la que se busca utiliza el conjunto de caracteres **latín-1** es posible encontrarlo utilizando `\xA9`.
- `\u` — Se utiliza para representar caracteres **Unicode** si se conoce su código. "`\u00A2`" representa el símbolo de centavos. No todos los motores de Expresiones Regulares soportan Unicode. El .Net Framework lo hace, pero el EditPad Pro no, por ejemplo.
- `\d` — Representa un dígito del 0 al 9.
- `\w` — Representa cualquier carácter **alfanumérico**.
- `\s` — Representa un espacio en blanco.
- `\D` — Representa cualquier carácter que no sea un dígito del 0 al 9.
- `\W` — Representa cualquier carácter no alfanumérico.
- `\S` — Representa cualquier carácter que no sea un espacio en blanco.
- `\A` — Representa el inicio de la cadena. No un carácter sino una posición.
- `\Z` — Representa el final de la cadena. No un carácter sino una posición.
- `\b` — Marca la posición de una palabra limitada por espacios en blanco, puntuación o el inicio/final de una cadena.
- `\B` — Marca la posición entre dos caracteres alfanuméricos o dos no-alfanuméricos.

Para más de ésto: [https://es.wikipedia.org/wiki/Expresión\\_regular](https://es.wikipedia.org/wiki/Expresión_regular)

Finalmente están los grupos anónimos, se establecen cada vez que se encierra una expresión regular en paréntesis, por lo que la expresión "< ([a-zA-Z]\w\*)? >" define un grupo anónimo. El motor de búsqueda almacenará una referencia al grupo anónimo que corresponda a la expresión encerrada entre los paréntesis.

La forma más inmediata de utilizar los grupos que se definen, es dentro de la misma expresión regular, lo cual se realiza utilizando la barra inversa "\\" seguida del número del grupo al que se desea hacer referencia de la siguiente forma: "< ([a-zA-Z]\w\*)? >.\*?</\1>" Esta expresión regular encontrará tanto la cadena "<font>Esta</font>" como la cadena "<b>prueba</b>" en el texto "<font>Esta</font> es una <b>prueba</b>" a pesar de que la expresión no contiene los literales "font" y "B".

Por ejemplo, en el texto:

```
abcdefghijklmnopqrstuvwxyz ABCDEFGHYJKLMNOPQRSTUVWXYZ  
0123456789 _+-.,!@#$^&*() ; \ / | <>  
12345 -98.7 3.141 .6180 9,000 +42  
555.223.4567 +1 (800)-(555)-(2468)  
foo@demo.net bar.ba@test.co.uk  
www.demo.com http://foo.co.uk/  
http://regexr.com/  
https://media.sample.net/
```

Si queremos buscar **+1 (800) – (555) – (2468)** usando nuestro ejemplo:

```
print(re.search(r"\+\d \(\d\d\d\)-\(\d\d\d\)-\(\d\d\d\d", informacion))
```

Notar que se puso `\+`, `\(` y `\)` para que los tome como caracteres a buscar y no como una expresión regular.

En el caso visto anteriormente, y en todos, el uso de las expresiones regulares depende de qué tan específico se quiera ser.

### Cuantidicadores

En el ejemplo anterior se escribió que precisábamos 11 dígitos. Se supone que una expresión regular debe captar todos los datos que se están buscando con la menor cantidad de código

Para evitar ésto se que se usan los metacaracteres de tipo iteración:

- a{1,5}      que aparezca la a de 1 a 5 veces
- a{2,}      que aparezca la a un mínimo de 2 veces
- a\*      que el carácter es opcional
- a|b      que el carácter que aparezca allí es una a o una b
- a?      que a sea un carácter opcional

Reescribiendo en nuestro ejemplo:

```
print(re.search(r"\+\d \(\d{3}\)-\(\d{3}\)-\(\d{4}\)", informacion))
```

Ésta expresión matchea tanto si se escribe **+1 (800) – (555) – (2468)** como **+1 800–555–2468** porque el paréntesis es opcional (`\(?)`), y se le está indicando con `\d{3}` y `\d{4}` en ese orden porque primero espera 2 bloques de 3 dígitos (`\d{3}`) y luego uno de 4 (`\d{4}`)

¿Cachai?

### Comjuntos como expresiones regulares

Para usar conjuntos de datos como patrones de búsqueda se usan los caracteres `[]`

- [abc]      busca todo lo que contenga los elementos a, b o c (en minúsculas)
- [^abc]      busca todo lo que no contenga los elementos a, b o c (en minusculas)
- [a-g]      busca todo lo que contenga los elementos en el rango del abecedario desde la a hasta la g en minúsculas

Por ejemplo, en el texto:

abcdefghijklmнопqrstuvwxyz    AZBc

- [A-Z]      encuentra AZB
- ([A-Z])\w    encuentra 2 cosas AZ y Bc
- \b[a-z]\b    encuentra solo la palabra “abcdefghijklmнопqrstuvwxyz”

Si se quiere buscar **+1 (800) – (555) – (2468)** en un texto, se puede simplificar otra vez:

```
print(re.search(r"\+\d [-\(\d\)]+", informacion))
```

**Nota:**

Para parsear cosas concretas como validación de sitios web, correos electrónicos y números de teléfono, o para hacer cosas concretas y específicas, lo mejor es usar a nuestro querido Tío Google y pedirle que busque expresiones regulares para...(lo que sea que estemos buscando)

**.findall**

Para buscar más de una ocurrencia en Python, se usa ésta función porque .match y .search sólo devuelven la primera ocurrencia.

Lo que devuelve la función.findall es una lista de strings que matchean

En nuestro ejemplo:

```
print(re.findall(r"\+\d [-\(\d\)]+", informacion))
```

El string es escaneado de izquierda a derecha y retorna la lista ordenada de acuerdo a qué encontró primero.

**Escribir archivos en python**

El proceso para escribir un archivo en Python es muy parecido al de leer uno:

```
<variable>=open("<nombre_del_archivo_nuevo>", flag)
```

Ejemplo

```
archivo_lista = open("lista.txt", "w")
```

Si el archivo no existe, lo crea, y si ya existe, lo va a sobreescibir o modificar.

Por otra parte, las banderas o flags se usan para indicarle si vamos a trabajar con el archivo solo para lectura, para escritura o ambas cosas, en nuestro caso, como vamos a escribirlo, se usa la bandera "**w**" del inglés "write".

La lista de flags:

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	open for exclusive creation, failing if the file already exists
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)
'U'	universal newlines mode (deprecated)

Una vez guardada la variable, se usa el método **.write()** para escribir lo que queramos en el archivo:

```
archivo_lista.write("Hola mundo\n")
```

```
archivo_lista.write("Este es mi primer archivo en Python")
```

Una vez escrito el archivo nuevo, se debe cerrarlo

```
archivo_lista.close()
```

Notar que .write se comporta en cierto modo como print en el sentido de que lo anterior también se puede escribir como:

```
archivo_lista.write("Hola mundo.\nEste es mi primer archivo en Python")
```

### Haciendo funciones con agregar archivos: Ejemplo

Supongamos que lo que quiero es hacer una lista de compras para el super:

Entonces:

```
def agregar_articulo(articulo):
    archivo_lista = open("lista.txt", "a")
    archivo_lista.write("{}\n".format(articulo))
    archivo_lista.close()

agregar_articulo(input("Artículo que deseas agregar: " ))
```

Notar que el flag usado en este caso es “**a**”, que lo que hace es un append, o sea, si existe el archivo, lo abre y pone todo lo nuevo a partir de la última linea.

## Peewee, y manejo de DB en Python

### Introducción

En el caso de Python, se puede trabajar con bases de datos usando un ORM (Object-Relational Mappers), que en realidad son librerías donde se trabaja con las tablas de DB usando programación orientada a objetos, lo cual permite una abstracción de los conceptos de SQL

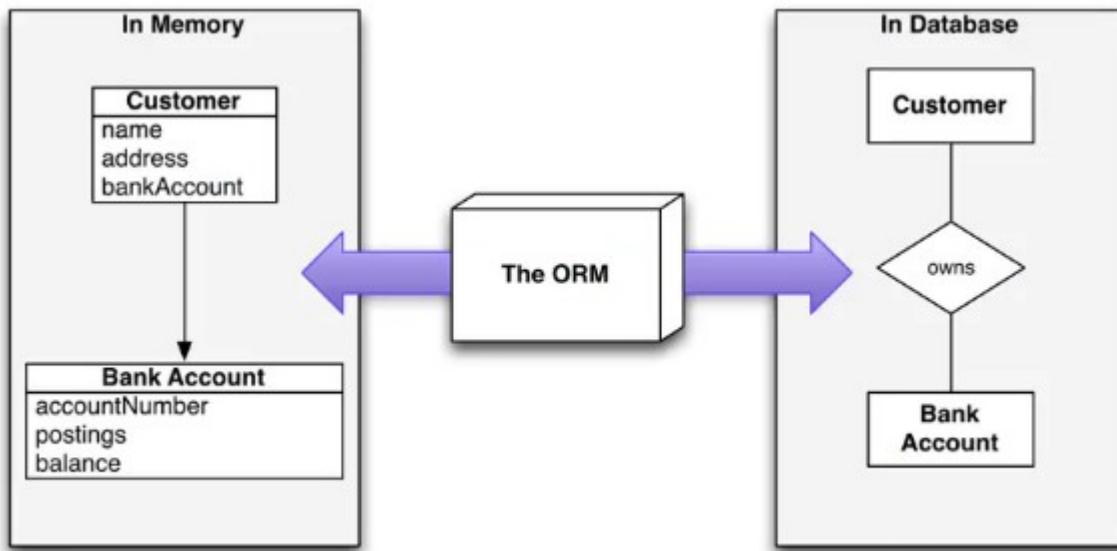
En particular, para el presente manual/curso, se va a utilizar la librería Peewee.

Peewee consiste en una librería, que usa la programación orientada a objetos.

La razón por la cual se prefiere usar un ORM es porque si bien SQL es una herramienta poderosa, es difícil de lidear con ella día a día.

Por otra parte la idea es la siguiente:

Cada tabla se corresponde con una clase, cada objeto es una fila y sus atributos son las columnas.



En la figura se ve cómo están relacionados los objetos en Python y como se representan en SQL.

El ORM (en nuestro caso Peewee) es quien interactúa entre Python y SQL para manipular la base de datos, con lo cual, no hace falta escribir en SQL las consultas para una base de datos. En este caso, el cliente tiene un nombre, una dirección y un numero de cuenta de banco que es la que lo relaciona a la tabla del banco. Por otra parte, la tabla del banco tienen 3 cosas: el numero de cuenta, las transacciones y el balance.

### Instalación de Peewee

Existen varias formas de instalar Peewee.

La más común es:

**pip install peewee** tanto en linux como en windows.

En linux, si se quiere instalar para python 3:

**pip3 install peewee**

En linux tambien se puede usar el gestor de paquetes de linux para buscarlo (en el caso de debian/ubuntu, synaptic)

Tambien se puede usar **sudo apt-get install python3-peewee**

Más sobre ésto, en la documentación <http://docs.peewee-orm.com/en/latest/> está disponible cómo instalar, entre otras cosas.

Si todo en la instalación, si se ejecuta en la consola de python3:

**from peewee import \***

no debe dar ningún error:

```

jenifer@azeroth:~$ python3
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from peewee import *
>>> 

```

## **Revisión de documentación en Peewee**

Si bien la mejor manera de aprender a programar es programando, es importante leer la documentación de las librerías y de los lenguajes en los cuales se va a programar pues así, se cometen menos errores al programar dado que así, por ejemplo, se sabe exactamente qué tipos de datos devuelve una función o un método en determinada librería; o también, las buenas prácticas de programación para una librería determinada.

En ésta sección se va a explorar la documentación que va a ser empleada para Peewee destacando las partes importantes:

1. Instalación y testing: <http://docs.peewee-orm.com/en/latest/peewee/installation.html>
2. Quickstart: Contiene toda la parte de los features principales de Peewee, como crear una tabla de DB, cómo se tabajan los modelos y ejemplos. <http://docs.peewee-orm.com/en/latest/peewee/quickstart.html>
3. Example app: en ésta sección se enseña cómo hacer una app similar a Twiter con Peewee. Nova a tener toda la funcionalidad de Twitter, pero sí sirve como ejemplo de aplicación útil que trabaja con Peewee y Flask <http://docs.peewee-orm.com/en/latest/peewee/example.html>
4. Qwerying: Es una de las unidades más extensas pero más útiles de la documentación. Describe cada una de las Qwerries que se pueden hacer a una DB. Como hacer Alias, Joins, Selects, ManyToManyField, uso de llaves foraneas, en fin, todas las operaciones con bases de datos. <http://docs.peewee-orm.com/en/latest/peewee/querying.html>
5. Qwery operators: En ésta sección están todas las operaciones comparativas. [http://docs.peewee-orm.com/en/latest/peewee/query\\_operators.html](http://docs.peewee-orm.com/en/latest/peewee/query_operators.html)
6. API documentation: Acá explica cómo funciona todo en peewee internamente, cómo estan implementadas sus clases y sus relaciones entre sí, sus modelos, en fin, todo. <http://docs.peewee-orm.com/en/latest/peewee/api.html>

## **Creación del primer proyecto y modelo con Peewee**

Lo que se va a hacer en ésta sección es crear una base de datos con 2 tablas en un directorio a gusto del consumidor con una tabla para personas.

Para eso se va a crear un archivo .py que contenga lo siguiente:

```
from peewee import * ← ésto importa peewee  
from datetime import date ← ésto es para que importe la función date de la librería datetime
```

```
db = SqliteDatabase('people.db') ← ésto guarda en la variable db la base de datos a  
crear/modificar
```

```
class Person(Model): #la clase Person es hija de la clase Model de  
peewee  
    name = CharField() #la tabla tiene como campos el nombre,  
    cumpleaños en formato fecha (por eso el import date)
```

```
birthday = DateField() #Ysi esta relacionada o no (por eso es un BooleanField
is_relative = BooleanField()

class Meta:
    database = db # éste modelo usa la DB "people.db".
```

Observar que para cada tabla de la base de datos se crea una clase nueva (Person), que es subclase de la clase **Model**, la cual pertenece a la librería e peewee.

Por otra parte, cada una de las clases de la tabla, tiene una subclase llamada **Meta**, que es obligatoria y es donde se le dice a Python y Peewee cuál es la base de datos que está describiendo el modelo y sus características. En nuestro caso, la DB va a ser “**db**” (**database = db**), y **db** que es una variable que va a manipular una base de datos de SQLite llamada people.db (**db = SqliteDatabase('people.db')**)

Recordar que las clases se mapean como tablas, los atributos como columnas y las instancias de cada objeto son las distintas filas.

Por otra parte, Peewee tiene muchos tipos de datos(**CharField()**, **DateField()**, **BooleanField()**, entre otros).

En fin, el producto de ejecutar ésto va a ser una tabla así:

PERSON TABLE		
NAME	BIRTHDAY	IS_RELATIVE

Con ésto, cada instancia u objeto de la clase Person, sería una entrada en la tabla

**Nota:**

Es importante que todos los scripts a crear en ésta parte queden bajo el mismo directorio y que allí queden los proyectos y las bases de datos.

Siguiendo ésto solamente, si se ejecuta, lo hace correctamente, pero sólo define las clases, ésto no crea ni modifica nada. Ésto se debe a que, así como está, no hay métodos para crear o modificar la bases de datos.

**Creación y conexión a la base de datos nueva**

Para crear la base de datos, primero hay que crearla:

```
def create_and_connect():
    db.connect() #éste metodo, si detecta que la DB no existe, la
    crea
    db.create_tables([Person], safe=True)
```

Ésto lo que hace es decirle a Python que se conecte a la base de datos guardada en la variable **db**.

**.connect()** la crea y **.create\_tables()**, crea la tabla nueva. **.create\_tables** lo que recibe es una lista con los nombres de las tablas, los cuales deben ser iguales a los nombres de sus clases correspondientes, y tiene un parámetro opcional llamado **safe** el cual está seteado por defecto en false, lo cual complica a Peewee si la tabla en la base de datos ya existe, por lo cual se fija dicho parámetro en true.

Luego en nuestro archivo, se define una función donde se van a especificar los objetos manualmente:

```
def create_family_members():
    uncle_tomy = Person(name="Tom", is_relative=True, birthday =
date(1960, 1, 15)) #acá se incluye la fecha
    uncle_tomy.save() #Ésto es lo que guarda en la tabla Person
    grandma = Person.create(name='Ana', birthday=date(1935, 3, 1),
is_relative=False) #el false es solo para experimentar
    grandma_rosa = Person.create(name='Rosa', birthday=date(1935, 3,
1), is_relative=False)
```

Acá en la tabla persons se agregan 3 personas: Tom, Rosa y Ana usando sus respectivas variables.

Al final del archivo se llama a los métodos:

```
create_and_connect()
create_family_members()
```

**Claves Foraneas**

Las claves foraneas son aquellas columnas (atributos) se emplean para relacionar tablas de bases de datos entre sí (o sea, relación entre 2 o más clases de tipo Model).

Para ver éste concepto, se va a crear otro modelo que va a contener una lista de mascotas, cuyos dueños se encuentran en la tabla Person:

```
class Pet(Model):
    name = CharField()
    animal_type = CharField()
    owner = ForeignKeyField(Person, related_name='pets') #acá le está diciendo que el campo donde debe buscar es en uno nuevo llamado 'pets' de la tabla Person
    class Meta:
        database = db
```

y la linea de create and conect cambia:

```
def create_and_connect():
    db.connect() #éste metodo, si detecta que la DB no existe, la def db.create_tables([Person,Pet],safe=True) ← acá se agrega la base de datos Pet
```

Si se mira el código, aparece el statement `owner = ForeignKeyField(Person, related_name='pets')` que significa que el atributo owner (o nombre del dueño), es una clave foranea de la tabla Person, y que, dentro de dicha tabla, las claves de las mascotas van a ir dentro de un campo llamado ‘pets’.

Los tipos de datos que son `ForeignKeyField()` representan a las llaves foraneas, los cuales son campos en una tabla que hacen referencia a un campo de otro objeto en otra tabla y nos dice la relación entre ellas.

En nuestro caso `owner` apunta una fila (objeto) de la tabla Person que es el dueño de la mascota.

Ésto ya crea las 2 tablas, lo cual se puede confirmar usando el comando `.tables` en SQLite:

```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/pew$ sqlite3 people.db
SQLite version 3.27.2 2019-02-25 16:06:06
Enter ".help" for usage hints.
sqlite> .tables
person  pet
sqlite> █
```

Por otra parte, para saber el esquema de la tabla, se hace dentro de SQLite:

`>.schema <tabla (opcional)>`

Si no se pone el nombre de la tabla, va a poner el esquema de todas las tablas

```
sqlite> .schema person
CREATE TABLE IF NOT EXISTS "person" ("id" INTEGER NOT NULL PRIMARY KEY, "name" V
ARCHAR(255) NOT NULL, "birthday" DATE NOT NULL, "is_relative" INTEGER NOT NULL);
sqlite> █
```

Para la tabla Person se tiene, como se esperaba 4 campos “id” que es su primary key, “name” que es de tipo varchar, “birthday” que es de tipo fecha, e “is\_relative” que es un integer not null donde 0 es True y cualquier otro número es False.

Para la tabla pet:

```
sqlite> .schema pet
CREATE TABLE IF NOT EXISTS "pet" ("id" INTEGER NOT NULL PRIMARY KEY, "name" VARCHAR(255) NOT NULL, "animal_type" VARCHAR(255) NOT NULL, "owner_id" INTEGER NOT NULL, FOREIGN KEY ("owner_id") REFERENCES "person" ("id"));
CREATE INDEX "pet_owner_id" ON "pet" ("owner_id");
```

Tienen un “id” que es su clave primaria, “name” que es su nombre, “animal\_type” que es de tipo VARCHAR y representa qué tipo de mascota es y finalmente “owner\_id” que es una clave foranea que hace referencia al “id” de la tabla person.

### Cómo asignar llaves foraneas

En nuestro caso, vamos a verlo con un ejemplo creando un método para asignar valores en la tabla Pet:

```
def create_family_members():
    uncle_tomy = Person(name="Tom", is_relative=True, birthday =
date(1960, 1, 15)) #acá se incluye la fecha
    uncle_tomy.save() #Ésto es lo que guarda en la tabla Person

    tom_pet = Pet.create(name = "Fido", animal_type = "dog", owner=
uncle_tomy)
```

Esto hace es crear una variable nueva (objeto nuevo, “tom\_pet”) de la tabla pet (o sea una instancia de la clase Pet), le da un nombre a la mascota (name = “Fido”), un “animal\_type” que es “dog” y nos dice que el owner es el objeto llamado uncle\_tomy, que es un objeto perteneciente a la clase Person que ya tiene sus valores. (Ver más abajo en crear y hacer update de registros nuevos).

### Como guardar información en las tablas:

Para guardar datos en DB en las tablas existen 2 formas:

1. <variable> = <Clase>(<atributos>)
   
<variable>.save() ← Ésto es lo que guarda en la tabla

Ejemplo:

```
uncle_tomy = Person(name="Tom", is_relative=True, birthday =
date(1960, 1, 15))
uncle_tomy.save()
```

2. <variable> = <Clase>.create(<atributos>)

Ejemplo:

```
grandma = Person.create(name='Ana', birthday=date(1935, 3, 1),
is_relative= False)
```

En el segundo caso, no hace falta hacer el `.save`

Para guardar los datos efectivamente, lo que se hace es crear métodos que contengan éstas variables, los cuales después son invocados:

```
def create_family_members():
    uncle_tomy = Person(name="Tom", is_relative=True, birthday =
date(1960, 1, 15)) #acá se incluye la fecha
```

```

uncle_tomy.save() #Ésto es lo que guarda en la tabla Person
grandma = Person.create(name='Ana', birthday=date(1935, 3, 1),
is_relative= False) #el false es solo para experimentar
grandma_rosa = Person.create(name='Rosa', birthday=date(1935, 3,
1), is_relative= False)

tom_pet = Pet.create(name = "Fido", animal_type = "dog", owner=
uncle_tomy)
grandma_pet = Pet.create(name = "Mishi", animal_type =
"cat", owner= grandma)

create_and_connect()
create_family_members()

```

Acá en éste ejemplo se crearon 2 filas en la tabla Persons, una para Tom (objeto uncle\_tomy) y otra para Ana (objeto grandma), y en la tabla Pet, también se crearon 2 filas con sus respectivas mascotas (objetos tom\_pet y grandma\_pet).

### Actualización de campos en una tabla

Para cambiar el valor de un campo en una tabla, funciona igual que en OOP (ver arriba en el módulo correspondiente), con la salvedad de que luego del statement de modificación, hay que invocar al método `.save`:

```

<variable_que_quiero_modificar>.atributo = <valor_nuevo>
<variable_que_quiero_modificar>.save()

```

Ejemplo:

```

tom_pet.name = "Firulais" #ésto cambia el nombre del perro
tom_pet.save() #esto guarda el cambio de la variable en la DB

```

Para saber si lo cambió al agregar éstas líneas de código al método `create_family_members`, vamos a SQLite3:

```

sqlite> .tables
person  pet
sqlite> select * from pet
...> ;
1|Firulais|dog|1
2|Mishi|cat|2
3|Firulais|dog|4
4|Mishi|cat|5

```

Como vemos, actualizó correctamente el dato del nombre del perro de “Fido” a “Firulais”

### Obtención de datos de una tabla ya creada

Una vez creada una base de datos con tablas, se puede acceder a todos sus campos o consultar uno en particular.

#### Cómo obtener todas las filas de la tabla

Para hacer ésto, se escribe, en el caso ejemplo, el siguiente método:

```
def get_family_members():
    for person in Person.select():
        print("Nombre: {}".format(person.name))
        print("Cumpleaños: {}".format(person.birthday))
```

O sea, lo que se utiliza es el método `.select()`, el cual pertenece a la clase Model de Peewee, y se recorre toda la tabla con un for. El select sin nada sería el equivalente de hacer la consulta en SQLite `select * from person`.

Por otra parte, `person.name` nos da el atributo “name” del objeto person (el cual va a ir tomando todos los valores de la tabla Person de a uno), y `person.birthday` su atributo “birthday”

Si llamamos éste método en nuestro programa nos da:

```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/pew$ python3 persons.py
Nombre: Tom
Cumpleaños: 1960-01-15
Nombre: Ana
Cumpleaños: 1935-03-01
Nombre: Rosa
Cumpleaños: 1935-03-01
```

#### Obtener un registro particular

Para obtener un registro en particular la sintaxis es la siguiente:

```
<variable>=
<Clase_correspondiente_a_la_tabla>.select().where(<Clase_correspondiente_a_la_tabla>.<columna> <signo_de_comparación>
<valor_a_buscar>).get()
```

Por ejemplo:

```
grandma_rosa= Person.select().where(Person.name == 'Rosa').get()
selecciona y guarda en la variable grandma_rosa, a la primer persona que se llame así en la tabla Person.
```

Para que imprima en pantalla:

```
def get_family_member():
    grandma_rosa= Person.select().where(Person.name == 'Rosa').get()
    print ("La abuela Rosa nació el
{}".format(grandma_rosa.birthday))
```

Si llamamos en el script a todos los métodos en el siguiente orden:

```
create_and_connect()
create_family_members()
print ("#####Todos los miembros de la familia#####")
get_family_members()
print ("#####Un solo miembro de la familia##")
print("get_family_member()")
```

Al ejecutar nos da:

```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/pew$ python3 persons.py
#####Todos los miembros de la familia#####
Nombre: Tom
Cumpleaños: 1960-01-15
Nombre: Ana
Cumpleaños: 1935-03-01
Nombre: Rosa
Cumpleaños: 1935-03-01
Nombre: Rosa
Cumpleaños: 1936-03-01
#####
###Un solo miembro de la familia##
La abuela Rosa nació el 1935-03-01
```

Lo otro que se puede hacer es modificar la función `get_family_member()` para que acepte como parámetro el nombre que queremos buscar:

```
def get_family_member(nombre):
    grandma_rosa= Person.select().where(Person.name == nombre).get()
    print ("La abuela Rosa nació el
{}".format(grandma_rosa.birthday))

get_family_member("Rosa")
```

Si se corre el programa, se puede ver que los registros se están repitiendo porque cada vez que se corre el programa, los crea una y otra vez

Hablando del programa, el código hasta ahora es:

```
from peewee import *
from datetime import date

db = SqliteDatabase('people.db')

#####Clases (Tablas)#####
class Person(Model):
    #la clase Person es hija de la clase Model de
    peewee
```

```
    name = CharField() #la tabla tiene como campos el nombre,
cumpleaños
    birthday = DateField() #Ysi esta relacionada o no (por eso es un
BooleanField
    is_relative = BooleanField()

    class Meta:
        database = db # éste modelo usa la DB "people.db".

class Pet(Model):
    name = CharField()
    animal_type = CharField()
    owner = ForeignKeyField(Person, related_name='pets') #acá le está
diciendo que el campo donde debe buscar es en uno nuevo llamado
'pets' de la tabla Person
    class Meta:
        database = db
#####
#####Métodos#####
def create_and_connect():
    db.connect() #éste metodo, si detecta que la DB no existe, la
crea
    db.create_tables([Person,Pet],safe=True)

def create_family_members():
    uncle_tomy = Person(name="Tom", is_relative=True,birthday =
date(1960, 1, 15)) #acá se incluye la fecha
    uncle_tomy.save() #Ésto es lo que guarda en la tabla Person
    grandma = Person.create(name='Ana', birthday=date(1935, 3, 1),
is_relative= False) #el false es solo para experimentar
    grandma_rosa = Person.create(name='Rosa', birthday=date(1935, 3,
1), is_relative= False)

    tom_pet = Pet.create(name = "Fido", animal_type = "dog",owner=
uncle_tomy)
    grandma_pet = Pet.create(name = "Mishi", animal_type =
"cat",owner= grandma)
    tom_pet.name = "Firulais" #ésto cambia el nombre del perro
    tom_pet.save() #esto guarda el cambio de la variable en la DB

def get_family_members():
```

```

        for person in Person.select(): #este select es el equivalente al
select * de SQL porque no se le pasó nada como parámetro y es de la
tabla Person
        print("Nombre: {}".format(person.name))
        print("Cumpleaños: {}".format(person.birthday))

def get_family_member(nombre):
    family_member= Person.select().where(Person.name == nombre).get()
    print ("{} nació el {}".format(nombre, family_member.birthday))
#####
#####Main Program#####
create_and_connect()
create_family_members()
print ("####Todos los miembros de la familia####")
get_family_members()
print ("####")
print()
print("##Un solo miembro de la familia##")
get_family_member('Rosa')

```

Y la salida es:

```

jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/pew$ python3 persons.py
#####Todos los miembros de la familia#####
Nombre: Tom
Cumpleaños: 1960-01-15
Nombre: Ana
Cumpleaños: 1935-03-01
Nombre: Rosa
Cumpleaños: 1935-03-01
#####
##Un solo miembro de la familia##
Rosa nació el 1935-03-01

```

Otra cosa que se puede hacer al consultar, es en nuestro ejemplo, imprimir las mascotas y sus dueños:

```

def mascotas_y_dueños():
    query = (Pet.select(Pet,
Person).join(Person).where(Pet.animal_type == 'cat'))
    for pet in query:
        print(pet.name, pet.owner.name)

```

Éste qwery devuelve los nombres de todos los gatos y el nombre de sus dueños

### Consulta para un único registro

Si lo que se tiene que buscar es un registro único, existe una forma más fácil de hacer el qwery:

```
<variable>=<Nombre_de_la_clase>.get(<parámetros_de_búsqueda>)
```

Ejemplo

```
family_member= Person.get(Person.name == nombre)
```

Ésto devuelve un solo registro de lo que queremos.

## Borrar registros

Si tomamos el script y lo corremos varias veces, se van a ir acumulando registros duplicados tanto en la tabla person como en la tabla pet de people.db.

### Para borrar múltiples registros

Para borrar un registro en una DB lo que se hace es crear un nuevo método que reciba como parámetro lo que se quiere borrar y se van a usar los métodos **.delete** y **.execute** de la clase Model de peewee.

Si no se usa el .execute, no se va a borrar nada.

El uso es el siguiente:

```
<variable>
=<Clase_tabla>.delete().where(<condición_que_se_debe_cumplir_para_borrar_registro>)
<variable>.execute
```

Ejemplo:

```
def delete_pet(name):
    query = Pet.delete().where(Pet.name == name)
    query.execute()

delete_pet("Mishi")
```

Si se ejecuta y luego se va a la tabla en SQLite, vemos que ahora, los registros de la mascota “Mishi”, no están.

```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/pew$ sqlite3 people.db
SQLite version 3.27.2 2019-02-25 16:06:06
Enter ".help" for usage hints.
sqlite> select * from pet
...> ;
1|Firulais|dog|1
3|Firulais|dog|4
5|Firulais|dog|7
7|Firulais|dog|10
9|Firulais|dog|13
11|Firulais|dog|16
```

Con éste comando hay que tener cuidado, porque si no se le pone ningún argumento, o se ponen los argumentos mal, **.delete** va a borrar todos los registros de la tabla.

Otro dato curioso es que, query.execute() nos puede decir cuántos registros fueron borrados.

Para eso, se modifica la linea del `.execute`:

```
delited_queries = qquery.execute()
```

Éste comando guarda en la variable `delete_queries` un integer que indica cuantos registros fueron borrados.

Entonces si se modifica el método de la siguiente forma:

```
def delete_pet(name):
    qquery = Pet.delete().where(Pet.name == name)
    deleted_entries= qquery.execute()
    print("{} registros borrados".format(deleted_entries))
```

y lo llamamos al final del main program así:

```
delete_pet("Mishi")
```

El resultado de la salida es:

```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/pew$ python3 persons.py
#####Todos los miembros de la familia#####
Nombre: Tom
Cumpleaños: 1960-01-15
Nombre: Ana
Cumpleaños: 1935-03-01
Nombre: Rosa
Cumpleaños: 1935-03-01
#####
##Un solo miembro de la familia##
Rosa nació el 1935-03-01
#####
mascotas y sus dueños:
Mishi Ana
1 registros borrados
```

Y si nos fijamos en la base de datos en SQLite vemos que modificó la tabla `pet` pero la tabla `person` no la tocó:

```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/pew$ sqlite3 people.db
SQLite version 3.27.2 2019-02-25 16:06:06
Enter ".help" for usage hints.
sqlite> .tables
person  pet
sqlite> select * from pet
...> ;
1|Firulais|dog|1
sqlite> select * from person;
1|Tom|1960-01-15|1
2|Ana|1935-03-01|0
3|Rosa|1935-03-01|0
```

### Cómo borrar un único registro en una tabla:

Para borrar uno y solo un registro, se puede usar todo lo anterior, pero hay una manera más fácil:

```
qquery=Pet.get(Pet.name == nombre)
```

```
qquery.delete_instance()
```

Con `.delete_instance()` nos aseguramos que solo borre 1 campo que contenga el nombre guardado en la variable `name`

### **Creando una aplicación de diario íntimo usando Peewee**

La idea de crear la aplicación de diario, se va a sacar de la documentación de peewee (<https://charlesleifer.com/blog/dear-diary-an-encrypted-command-line-diary-with-python/> y <https://github.com/coleifer/peewee/blob/master/examples/diary.py>), no la diferencia de que no se van a encriptar las entradas y no va a pedir passwords.

Las funcionalidades van a ser:

- Agregar entradas nuevas
- Ver todas las entradas del diario
- Borrar entradas
- Salir del diario.

Para ésto vamos a crear otra carpeta donde va a estar guardado el proyecto de diario y el archivo de tablas va a ser `diary.py`

En ése archivo, lo primero que se debe hacer es importar Peewee:

```
from peewee import *
```

Luego vamos a crear una nueva clase que es de tipo Model de peewee:

```
from peewee import *
from datetime import *

db = SqliteDatabase('diary.db')

class Entry(Model):
    #contenido de la entrada
    #Fecha y hora de la entrada

    class Meta:
        database = db
```

También vamos a necesitar 4 métodos:

1. El del menú: `menu_loop()`
2. El de agregar: entradas: `add_entry()`
3. El de ver entradas: `view_entries()`
4. La de buscar entradas: `search_entry()`
5. El de borrar entradas: `delete_entry(entry)`

```
def menu_loop():
    """Show Menu"""

def add_entry():
    """Add Entry"""
```

```
def view_entries():
    """View Entries"""

def delete_entry(entry):
    """delete entry"""

def search_entry(entry):
    """search entry"""
```

Las triples comillas son de escape para que, cuando interprete, Python se de cuenta de que no están definidos los métodos y no crashee por eso.

### Docstrings (vista general y aplicado al código del diario)

Los docstrings son texto que se arman para generar documentación de lo que se está desarrollando para evitar que otros programadores tengan que entrar sí o sí a ver nuestras clases y métodos con el fin de inferir el cómo usarlos.

Lo que va entre comillas triples, es lo que va a docstring.

### Name == main.

Cuando se tienen métodos dentro de un archivo ejecutable .py, pero solo se quiere importar todas sus clases y métodos, sin que ejecute el main program que es donde se hace los llamados a los métodos, lo que se hace es poner el statement:

```
if __name__ == "__main__":
```

```
from peewee import *
from datetime import *

db = SqliteDatabase('diary.db')

class Entry(Model):
    """Arma la tabla de entradas en la base de datos diary.db"""
    #contenido de la entrada
    #Fecha y hora de la entrada

    class Meta:
        database = db

def menu_loop():
    """Show Menu:
    Muestra el menú de que quiere hacer el usuario"""

def add_entry():
    """Add Entry:
    Agrega una nueva entrada en el diario"""

def view_entries():
```

```
"""View Entries"""

def search_entry(entry):
    """search entry"""

def delete_entry(entry):
    """delete entry"""

def hello_world():
    print ("Hello world")

if __name__ == "__main__":
    hello_world()
```

Si se ejecuta así:

`python3 diary.py`

nos da:

`Hello world`

pero si ejecutamos:

```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/pew/diary$ python3
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from diary import *
>>> █
```

solo importa los métodos pero no ejecuta el programa principal.

Si no se pone el statement, el resultado del “`from diary import *`”, importaría toda la librería pero imprimiría tambien en pantalla la frase “Hello world”

```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/pew/diary$ python3
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from diary import *
Hello world
```

O sea `__name__ == "__main__"` le indica a python que ejecute lo que está dentro del if statement si se llama al programa directamente y no cuando se están importando sus librerías.

### Creación del modelo

Para crear nuestro modelo de diario vamos a editar la clase Entry y crear un nuevo método que cree las bases de datos:

```
class Entry(Model):
    """Arma la tabla de entradas en la base de datos diary.db"""
    content = TextField()#contenido de la entrada la diferencia con
    #charfield es que TextField puede contener cualquier texto sin
    #importar su largo
```

```

timestamp = DateTimeField(default = datetime.datetime.now())
#Fecha y hora de la entrada

class Meta:
    database = db

def create_and_connect():
    """Connects to the database and create new tables"""
    db.connect()
    db.create_tables([Entry], safe = True)

```

Como resultado de ejecutar python diary.py, nos construye un archivo nuevo de SQLite, con una sola tabla, que es “entry” y su esquema es el siguiente:

```

jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/pew/diary$ sqlite3 diary.db
SQLite version 3.27.2 2019-02-25 16:06:06
Enter ".help" for usage hints.
sqlite> .tables
entry
sqlite> .schema entry
CREATE TABLE IF NOT EXISTS "entry" ("id" INTEGER NOT NULL PRIMARY KEY, "content"
TEXT NOT NULL, "timestamp" DATETIME NOT NULL);
sqlite> █

```

### Diccionarios ordenados y menú del programa

Usualmente los diccionarios en Python no conservan ningún orden, pero hay veces en que es útil tenerlos ordenados.

Para eso se usa el método `OrderedDict` de la librería `collections`, el cual recibe como parámetro una lista cuyos valores son tuplas que corresponde con la pareja `key, value` del diccionario.

En el código ejemplo:

```

from collections import OrderedDict
menu_dict = OrderedDict([
    ("a", "add_entry"),
    ("b", "view_entries"),
    ("c", "search_entry"),
    ("d", "delete_entry")
])

```

### **Uso del diccionario ordenado para presentar el menú (`__doc__`):**

Una forma fácil de hacerlo es que, en lugar de escribir el menú como está arriba, poner como value las funciones correspondientes y en la función `menu_loop`, usar el método `__doc__` para que en cada opción, la presente al usuario con la documentación de cada función.

Eso sí, hay que tener cuidado que la variable `menu` sea declarada después de las funciones.

```

def menu_loop():
    """Show Menu:

```

```

Muestra el menú de que quiere hacer el usuario"""
choice = None
while choice != "q":
    print ("Press q to quit")
    for key,value in menu.items():
        print ("{} {}) {}".format(key,value.__doc__))
choice = input("Please choose an option: ").lower().strip()

if choice in menu: #si la opción está en el menu
    menu[choice]() #ejecuta la función correspondiente

menu_dict = OrderedDict([
    ("a", "add_entry"),
    ("b", "view_entries"),
    ("c", "search_entry"),
    ("d", "delete_entry")
])

```

La función strip, es para que al input le saque todos los caracteres especiales tipo espacios, tabs y enters; por otra parte `menu[choice]()` se puede usar porque menu[choice] es el nombre de una de las funciones que existen dentro del diccionario ordenado.

### Añadir registros en el diario

Antes que nada, se va a importar la librería sys que sirve para obtener mejor las entradas del usuario.

```
import sys
```

Ahora con ésto se arma el método de agregar entradas:

```

def add_entry():
    """Add Entry:
    Agrega una nueva entrada en el diario"""
    print ("Enter your thoughts: (press Ctrl+D to finish)")
    data = sys.stdin.read().strip()
    if data:
        if input("Do you want to save your data? (Y/n)")\
        ".lower().strip() !='n':
            Entry.create(content = data)
            print ("Entry created succesfully")

```

### Ver registros en el diario

Para ver todos los registros, se los recorre con un for:

```

def view_entries():
    """View Entries:
    Muestra todas las entradas del diario"""
    print ("view entries")
    print ("-----")
    for data in Entry:

```

```
    print(data.timestamp)
    print(data.content)
    print("-----")
```

Entonces, si usamos la opción a y luego la b, nos queda por ejemplo:

```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/pew/diary$ python3 diary.py
Press d to quit
a) Add Entry:
    Agrega una nueva entrada en el diario
b) View Entries:
    Muestra todas las entradas del diario
c) delete entry:
    Toma una entrada del diario y la borra
Please choose an option: A
Enter your thoughts: (press Ctrl+D to finish)
Estoy siguiendo los videos de Aldo.
Do you want to save your data? (Y/n)y
Entry created succesfully
Press d to quit
a) Add Entry:
    Agrega una nueva entrada en el diario
b) View Entries:
    Muestra todas las entradas del diario
c) delete entry:
    Toma una entrada del diario y la borra
Please choose an option: b
view entries
```

```
-----
2019-09-17 15:39:25.208981
pienso leer python
-----
2019-09-17 15:57:47.562831
Estoy siguiendo los videos de Aldo.
-----
Press d to quit
a) Add Entry:
    Agrega una nueva entrada en el diario
b) View Entries:
    Muestra todas las entradas del diario
c) delete entry:
    Toma una entrada del diario y la borra
Please choose an option: d
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/pew/diary$
```

Si se quiere que el orden en que se presentan las entradas sea descendente y con un lindo formato de fecha y hora:

```
def view_entries():
    """View Entries:
    Muestra todas las entradas del diario"""
    data = Entry.select().order_by(Entry.timestamp.desc())
    print ("view entries")
```

```
print("-----")
for entry in data:

    print(entry.timestamp.strftime("%A %B %d, %Y %I:%M %p"))
    print(entry.content)
    print("-----")
```

Y el resultado es:

```
a) Add Entry:
    Agrega una nueva entrada en el diario
b) View Entries:
    Muestra todas las entradas del diario
c) delete entry:
    Toma una entrada del diario y la borra
Please choose an option: b
view entries
-----
Tuesday September 17, 2019 03:57 PM
Estoy siguiendo los videos de Aldo.
-----
Tuesday September 17, 2019 03:39 PM
pienso leer python
-----
Press d to quit
a) Add Entry:
    Agrega una nueva entrada en el diario
b) View Entries:
    Muestra todas las entradas del diario
c) delete entry:
    Toma una entrada del diario y la borra
Please choose an option: d
```

Por otra parte, si se quiere que muestre los registros de a uno y preguntarle al usuario todo el tiempo si quiere ver el siguiente registro se agrega un if en el for:

```
def view_entries():
    """View Entries:
    Muestra todas las entradas del diario"""
    data = Entry.select().order_by(Entry.timestamp.desc())
    print ("view entries")
    print("-----")
    for entry in data:

        print(entry.timestamp.strftime("%A %B %d, %Y %I:%M %p"))
        print(entry.content)
        print("-----")
```

```
        action= input("Do you want to view next entry?  
(Y/n) " ).lower().strip()  
        if action == "n":  
            break
```

### **Buscar entradas en la DB**

Para buscar en una tabla de base de datos, simplemente lo que se hace es definir una función search\_qquery, que va a tener un parámetro interno donde se va a usar el input que el usuario le va a decir qué termino buscar.

Una vez que se guarda eso en la variable interna, se llama va view\_entries, y a ésta función se le va a pasar la variable obtenida.

Para éso se va a escribir el código de search entry, reescribir el view\_entries, se le va a modificar de forma tal que, la primera linea para que pueda aceptar éste parámetro, pero si no se le pasa nada, el parámetro es none:

```
def view_entries(search_qquery = None):  
    """View Entries"""\n    data = Entry.select().order_by(Entry.timestamp.desc())\n\n    if search_qquery: #si hay un patrón de búsqueda, guarda en la  
    variable data, todo lo que coincida\n        data = data.where(Entry.content.contains(search_qquery)) #Para  
    eso se usa el .contains\n\n\n    print ("view entries")\n    print("-----")\n    for entry in data:\n\n        print(entry.timestamp.strftime("%A %B %d, %Y %I:%M %p"))\n        print(entry.content)\n        print("-----")\n\n    action= input("Do you want to view next entry?  
(Y/n) " ).lower().strip()\n    if action == "n":\n        break\n\n\ndef search_entry():\n    """Search an entry"""\n    search_qquery = input("Search entry in diary: ").strip()\n    view_entries(search_qquery)
```

Notar que la única diferencia es que **view\_entries** tiene 2 líneas de código al principio que le indican que, si el usuario quiere buscar alguna entrada en la tabla con algún término, asigna a la variable **data** solo los valores que contengan ese término, lo cual logra así:

```
data = data.where(Entry.content.contains(search_qquery))
```

Si se ejecuta el código:

```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/pew/diary$ python3 diary.py
Press q to quit
a) Add Entry
b) View Entries
c) Search an entry
d) delete entry
Please choose an option: c
Search entry in diary: python
view entries
-----
Tuesday September 17, 2019 03:39 PM
pienso leer python
-----
Do you want to view next entry? (Y/n)
Press q to quit
a) Add Entry
b) View Entries
c) Search an entry
d) delete entry
Please choose an option: q
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/pew/diary$ █
```

Como se ve en la figura, sólo se despliegan los items que cumplen con la condición dada por el usuario.

### Borrando una entrada

Para borrar una entrada, lo primero que hay que hacer es preguntarle al usuario si está seguro de que la quiere borrar, porque luego de borrado, es muy difícil de recuperar.

Entonces:

```
def delete_entry(entry):
    """Delete entry"""
    action = input("Are you sure you want to delete this entry? (Y/n")
").lower().strip()
    if action == "y":
        entry = entry.delete_instance()
```

Y en **menu\_loop** se hace otra excepción en el if:

```
if choice in menu:
    menu[choice]()
elif choice == "d":
    delete_entry(entry)
```

### Resumiendo, el código nos queda:

```
from peewee import *
import datetime
from collections import OrderedDict
import sys

db = SqliteDatabase('diary.db')

class Entry(Model):
    """Arma la tabla de entradas en la base de datos diary.db"""
    content = TextField()#contenido de la entrada la diferencia con
    #charfield es que TextField puede contener cualquier texto sin
    #importar su largo
    timestamp = DateTimeField(default = datetime.datetime.now())
    #Fecha y hora de la entrada

    class Meta:
        database = db

def create_and_connect():
    """Connects to the database and create new tables"""
    db.connect()
    db.create_tables([Entry], safe = True)

def menu_loop():
    """Show Menu:
    Muestra el menú de que quiere hacer el usuario"""
    choice = None
    while choice != "q":
        print ("Press q to quit")
        for key,value in menu.items():
            print ("{} {}) {}".format(key,value.__doc__))
        choice = input("Please choose an option: ").lower().strip()

        if choice in menu:
            menu[choice]()
        elif choice == "d":
            delete_entry(entry)

def add_entry():
    """Add Entry"""
    print ("Enter your thoughts: (press Ctrl+D to finish)")
    data = sys.stdin.read().strip()
```

```
if data:
    if input("Do you want to save your data?
(Y/n) ").lower().strip() !='n':
        Entry.create(content = data)
        print ("Entry created successfully")

def view_entries(search_qquery = None):
    """View Entries"""
    data = Entry.select().order_by(Entry.timestamp.desc())

    if search_qquery: #si hay un patrón de búsqueda, guarda en la
variable data, todo lo que coincide
        data = data.where(Entry.content.contains(search_qquery)) #Para
eso se usa el .contains

    print ("view entries")
    print("-----")
    for entry in data:

        print(entry.timestamp.strftime("%A %B %d, %Y %I:%M %p"))
        print(entry.content)
        print("-----")
        print("what do you want to do next:")
        action= input("a) view next entry\nb) delete this entry\nc)
go to main program\nChoose: " ).lower().strip()
        if action == "a":
            continue
        elif action == "b":
            delete_entry(entry)
        elif action == "c":
            break
        else:
            print("I don't know what to do, so I'll take yo to main
program")
            break

def search_entry():
    """Search an entry"""
    search_qquery = input("Search entry in diary: ").strip()
    view_entries(search_qquery)

def delete_entry(entry):
    """delete entry"""
```

```

action = input("Are you sure you want to delete this entry? (Y/n)
").lower().strip()
if action == "y":
    entry = entry.delete_instance()

menu = OrderedDict([
    ("a", add_entry),
    ("b", view_entries),
    ("c", search_entry),
])
if __name__ == "__main__":
    create_and_connect()
    menu_loop()

```

### Reto de la sección

Agregar las siguientes funcionalidades:

1. Buscar entradas por fecha
2. Paginar mejor la lista de entradas
3. Opción de editado de entradas
4. ¿Aregar un web front end con flask y Django?

#### 1) Buscar entrada por fecha

Para hacer ésto, la solución que encontré fué modificar la función search\_query de la siguiente manera:

```

def search_entry():
    """Search an entry"""
    how_to_search= input("Do you want to search:\na)by pattern\nb)by
date (day/month/year)\nChoose: ").lower().strip()

    if how_to_search == "a":
        by_date = False
    elif how_to_search == "b":
        by_date = True
    else:
        print ("I don't recognize this option, I'll take you to main
program")
        return      ← si no reconoce la orden, te lleva al programa main
    search_qwery = input("Search entry in diary: ").strip()
    view_entries(search_qwery,by_date)

```

Lo que resulta de que a la función view\_entries le agregué un flag que indica si el search\_query quiere hacerlo por fecha o por patrón de entrada (**view\_entries(search\_qwery,by\_date)**).

Por otra parte, en view\_entries, las banderas por default es que search\_query = none y by\_date = False, lo cual hace que, si el usuario no busca nada, despliegue todas las entradas.

Por otra parte, si hay banderas, en la función view\_entries, va a entrar a los siguientes ifs:

```
data = Entry.select().order_by(Entry.timestamp.desc())
```

```
if search_qquery: #si hay un patrón de búsqueda, guarda en la variable data, todo lo que coincide
    if by_date: ← entra éste if si busca por fecha
        data = data.where((Entry.timestamp.year ==
datetime.datetime.strptime(search_qquery, "%d/%m/%Y").year) &
(Entry.timestamp.month ==
datetime.datetime.strptime(search_qquery, "%d/%m/%Y").month) &
(Entry.timestamp.day ==
datetime.datetime.strptime(search_qquery, "%d/%m/%Y").day))
    else: ← entra a éste else, si busca por patrón
        data = data.where(Entry.content.contains(search_qquery))
```

En suma, el código entero de view\_entries es:

```
def view_entries(search_qquery = None, by_date = False):
    """View Entries"""
    data = Entry.select().order_by(Entry.timestamp.desc())

    if search_qquery: #si hay un patrón de búsqueda, entra acá
        if by_date:
            data = data.where((Entry.timestamp.year ==
datetime.datetime.strptime(search_qquery, "%d/%m/%Y").year) &
(Entry.timestamp.month ==
datetime.datetime.strptime(search_qquery, "%d/%m/%Y").month) &
(Entry.timestamp.day ==
datetime.datetime.strptime(search_qquery, "%d/%m/%Y").day))
        else:
            data = data.where(Entry.content.contains(search_qquery))
    #Para eso se usa el .contains

    print ("view entries")
    print("-----")
    for entry in data:

        print(entry.timestamp.strftime("%A %B %d, %Y %I:%M %p"))
        print(entry.content)
        print("-----")
        print("what do you want to do next:")
        action= input("a) view next entry\nb) delete this entry\n")
        go to main program\nChoose: " ).lower().strip()
        if action == "a":
```

```
        continue
    elif action == "b":
        delete_entry(entry)
    elif action == "c":
        break
    else:
        print("I don't know what to do, so I'll take you to main
program")
    break
```

**QUEDA PENDIENTE EL RESTO DEL DESAFÍO**

## Mejorando el código en Python

### Introducción

En las secciones pasadas, se ha aprendido a escribir gran cantidad de código en Python: aprendimos cómo crear funciones, variables, cadenas, cómo trabajar con fechas y horas, cómo trabajar con bases de datos, además aprendimos cómo trabajar con estructuras de datos y manejo de archivos en Python.

Todo eso es muy importante para que nosotros sepamos cómo escribir grandes cosas en Python desde páginas web hasta aplicaciones standalone en una computadora, réplicas de Twitter y tal vez muchas cosas grandiosas.

El único problema de escribir cosas grandiosas es que muchas veces se pueden volver complicadas y no muy agradables de ver. Es muy común ver código en la web o en el proyecto de otros o incluso en tus propios proyectos que es muy difícil de entender y realmente no se sabe ni qué es lo que querías hacer con una función.

Es por eso que en esta sección aprenderemos cómo escribir mejor código en Python que se apega a las normas que ya están establecidas con los PEPs.

### PEPs

La sigla PEP significa *Python Enhancement Proposal* o Propuesta de Mejora a Python.

Python al ser un lenguaje de programación de código abierto permite que cualquiera de nosotros haga propuestas de cómo queremos que se mejore el lenguaje, así que muchos usuarios a lo largo de los años han hecho propuestas de cómo se debe de escribir el código en Python: Cómo se debe describir la documentación en Python o qué cambios debe haber para la siguiente versión tal vez etc..

El PEP, y en particular, PEP 8 es específicamente una guía de estilo para el código escrito en Python. Nos da una guía cómo debemos de escribir nuestro código para que sea más fácil de entender para otros desarrolladores y para que pueda estar como apegado a las convenciones que muchas personas siguen.

Se puede ver documentación de PEP 8 en la página <https://www.python.org/dev/peps/pep-0008/>

Aquí está toda la documentación de este PEP pueden ver que dice PEP 8.

O sea, es una guía de Estilo para el código en Python y su autor es Guido Van Rossum, Nick Coghlan y Barry Watson.

Es fácilmente el Pep más famoso de todos y se puede ver todo el contenido.

Es muy importante que tener en cuenta muchas de las cosas que dicen pero repasaremos unos de los puntos más importantes en esta sección:

### **A Foolish Consistency is the Hobgoblin of Little Minds**

Básicamente, lo que dice ésta parte de la documentación es que tenemos que aplicar las reglas de este PEP cuando tenga sentido para nuestro entorno. Por ejemplo, si se está trabajando en una empresa donde están haciendo código en Python y tienen su propia guía de estilos y el estandares que deben de aplicar deben de aplicar ese estándar y no el de PEP. En caso contrario, si no se tiene una guía de estilo, es recomendable usar el PEP8.

Lo importante al programar es que se tenga consistencia, a menos que la consistencia no sea aplicable. Hay que saber diferenciar esos casos.

### **Code Lay-out: Indentación**

Lo que se recomienda para indentar cada linea es que sea de 4 espacios vacíos por nivel de indentación. Por otra parte, se muestra cómo separar funciones que toman varios parámetro para no marear a quien esté leyendo el código.

Hay ejemplos de cómo deberían de ser algunos delimitadores en las funciones, cómo separar funciones con muchos argumentos para que no se vuelva confuso, pero quizás lo más importante simplemente utiliizar cuatro espacios por nivel de indentación y no uno, dos o tres, o tabuladores.

Es decir si tienen una función y ponen su bloque de código va a tener cuatro espacios por su indentación, y si se escribe por ejemplo dentro de la función un while, el bloque de código con seguridad va a tener cuatro espacios de indentación adicionales y así sucesivamente.

### **Code Lay-out: Número de caracteres**

Otro punto muy importante es el máximo número de caracteres en una línea nos dice que debe de ser como máximo 79 caracteres.

Ésto es porque, si una linea se vuelve muy larga, el lector del código se va a perder, con éste límite el usuario se puede dar cuenta que la line siguiente, es continuación de la anterior y no es un newline para el intérprete de python

O sea, entre más larga es una línea de código es más difícil de entender.

Si se pone un if con 20 condiciones va a ser un verdadero dolor de cabeza para otros desarrolladores comprenderlo así que hay que procurar que siempre esté limitado a 79 caracteres.

Si tiene más de eso tal vez puedan inicializar una variable antes o chequear unas condicionantes.

## Blank Lines

Otra parte muy importante es la de las Blank Lines.

Todas las funciones de Top Level es decir que no están dentro de una función están por si mismas dentro de nuestro script de python, lo cual hace que deban estar bien separadas.

Lo recomendado es separar una función de otra usando 2 líneas en blanco y los métodos dentro de una clase deben de estar separados por una sola línea en blanco.

Por otra parte, cuando se tienen funciones que están relacionadas entre sí porque son las de las operaciones matemáticas como suma resta multiplicación y división usen su propio criterio para utilizarlas Blank Lines.

## Imports

En esta parte dice que los imports deben de estar agrupados en el siguiente orden:

1. Los que son de la librería estándar de Python por ejemplo la de la fecha que utilizamos nosotros.
2. Imports de librerías externas.
3. Imports que son específicos que hemos desarrollado nosotros para nuestra aplicación.

También debemos de poner una línea en blanco entre cada tipo de estos imports.

## White Spaces en expresiones y statements

Ésta sección indica cuando aplicar o no espacios en blanco en las funciones.

Los casos donde **NO** es recomendable:

1. Inmediatamente después de cualquier tipo de paréntesis o llaves
2. Entre una coma que implica un rango, seguida de un paréntesis:  
Si: foo = (0,  
No: bar = (0, )
3. Inmediatamente después de una coma, un punto y coma o 2 puntos  
Si: if x == 4: print x, y; x, y = y, x  
No: if x == 4 : print x , y ; x , y = y , x
4. Sin embargo, en un segmento, el ":" actúa como un operador binario, y debe tener cantidades iguales en ambos lados (tratándolo como el operador con la prioridad más baja). En un corte extendido, ambos dos puntos deben tener la misma cantidad de espacio aplicado. Excepción: cuando se omite un parámetro de segmento, se omite el espacio.
5. Inmediatamente después de llamar una variable y antes de abrir un paréntesis:  
Si: spam(1)  
No: spam (1)  
Si: dct['key'] = lst[index]  
No: dct ['key'] = lst [index]
6. Más de un espacio en blanco al asignar una variable:

Si:

x = 1

y = 2

long\_variable = 3

No:

x = 1

y = 2

long\_variable = 3

7. Al final de nuestras expresiones siempre que sea posible. Lo que tiene de importante es que usualmente es invisible pero puede llegar a ser confuso y hay algunos editores que no te dejan ejecutar tu código de Python si ven estos espacios en blanco.

### Naming conventions

Son reglas para tener en cuenta para nombrar sus constantes los nombres de sus clases y las excepciones, tipos de nombre a evitar.

Los nombres que son visibles para el usuario (programador) como partes públicas de la API deben seguir las convenciones que reflejan el uso en lugar de la implementación.

<https://www.python.org/dev/peps/pep-0008/#naming-conventions>

### Recapitulando

Si se tienen dudas sobre si se está siguiendo bien o no la convención, existe la página PEP8 online (<http://pep8online.com/>), donde se puede chequear éstas cosas y lo analizará para decirles qué es lo que les falta o dónde están mal.

### Arreglando el código del diario usando PEP online

Si se corre el código original de diary.py, encontraba líneas vacías, espacios antes de los paréntesis, variables que al ser de tipo input, eran muy largas, por lo cual se modificó el código a lo siguiente:

```
from peewee import *
import datetime
from collections import OrderedDict
import sys
```

```
#####Definición de la DB y su tabla#####
db = SqliteDatabase('diary.db')
```

```
class Entry(Model):
    """Arma la tabla de entradas en la base de datos diary.db"""
    #contenido de la entrada la diferencia con
    #charfield es que TextField puede contener
```

```
#cualquier texto sin importar su largo
content = TextField()

#Fecha y hora de la entrada
timestamp = DateTimeField(default=datetime.datetime.now())

class Meta:
    database = db
#####
#####Función de creación de DB#####
def create_and_connect():
    """Connects to the database and create new tables"""
    db.connect()
    db.create_tables([Entry], safe=True)
#####

#####Función de despliegue de menú principal#####
def menu_loop():
    """Show Menu"""
    choice = None

    while choice != "q":
        print ("Press q to quit")

        for key, value in menu.items():
            print ("{} {}) {}".format(key, value.__doc__))

        choice = input("Please choose an option: ").lower().strip()

        if choice in menu:
            menu[choice]()

        elif choice == "d":
            delete_entry(entry)
#####

#####Función Add Entry#####
def add_entry():
    """Add Entry"""
    print ("Enter your thoughts: (press Ctrl+D to finish)")
    data = sys.stdin.read().strip()
    if data:
```

```
        option = input("Do you want to save your data? (Y/n)\n").lower().strip()
        if option != 'n':
            Entry.create(content=data)
            print ("Entry created successfully")
#####
#####Funcion View Entries#####
def view_entries(search_qquery=None, by_date=False):
    """View Entries"""
    data = Entry.select().order_by(Entry.timestamp.desc())

    if search_qquery: #si hay un patrón de búsqueda
        #guarda en la variable data, todo lo que coincida

        if by_date:
            data = data.where(
                (Entry.timestamp.year ==
datetime.datetime.strptime(search_qquery, "%d/%m/%Y").year) &
                (Entry.timestamp.month ==
datetime.datetime.strptime(search_qquery, "%d/%m/%Y").month) &
                (Entry.timestamp.day ==
datetime.datetime.strptime(search_qquery, "%d/%m/%Y").day)
            )

        else:
            data = data.where(Entry.content.contains(search_qquery))
#Para eso se usa el .contains

    print ("view entries")
    print("-----")
    for entry in data:

        print(entry.timestamp.strftime("%A %B %d, %Y %I:%M %p"))
        print(entry.content)
        print("-----")
        print("what do you want to do next:")
        print("a) view next entry")
        print("b) delete this entry")
        print("c) go to main program")
        action = input("Choose: ").lower().strip()

        if action == "a":
            continue

        elif action == "b":
```

```
        delete_entry(entry)

    elif action == "c":
        break

    else:
        print("I don't know what to do, so I'll take you to main
program")
        break
#####
#####Función Search Entry #####
def search_entry():
    """Search an entry"""
    print()
    print("a)by pattern")
    print("b)by date (day/month/year)")
    how_to_search = input("Choose: ").lower().strip()

    if how_to_search == "a":
        by_date = False
    elif how_to_search == "b":
        by_date = True
    else:
        print ("I don't recognize this option, I'll take you to main
program")
        return
    search_qquery = input("Search entry in diary: ").strip()
    view_entries(search_qquery, by_date)
#####

#####Función Delete Entry #####
def delete_entry(entry):
    """delete entry"""
    action = input("Are you sure you want to delete this entry? (Y/n)
").lower().strip()
    if action == "y":
        entry = entry.delete_instance()
#####

#####Función de edit entry#####
def edit_entry():
    """Edit Entry"""
#####
```

```
#####Menú de opciones#####
menu = OrderedDict([
    ("a", add_entry),
    ("b", view_entries),
    ("c", search_entry),
    ("d", edit_entry)
])
#####

if __name__ == "__main__":
    create_and_connect()
    menu_loop()
```

Ésta versión así escrita sigue dando errores con la parte de los comentarios, pero yo decidí ignorar esos errores, porque los comentarios, así como están colocados, separan bien las cosas.

El otro error que ignoré fué los correspondientes a las lineas:

```
if by_date:
    data = data.where(
        (Entry.timestamp.year ==
datetime.datetime.strptime(search_qquery, "%d/%m/%Y").year) &
        (Entry.timestamp.month ==
datetime.datetime.strptime(search_qquery, "%d/%m/%Y").month) &
        (Entry.timestamp.day ==
datetime.datetime.strptime(search_qquery, "%d/%m/%Y").day)
```

La razón por la que lo hice es porque no se me ocurrió una forma de reescribir este condicional, lo cual implicaría hacer variables auxiliares que al implementarlas, no sé si den resultado.

Lo que si se modificó fue la presentación de los menues y el como recibe el input en la función search\_entries:

```
def search_entry():
    """Search an entry"""
    print("You can search your entry:")
    print("a)by pattern")
    print("b)by date (day/month/year)")
    how_to_search = input("Choose: ").lower().strip()

    if how_to_search == "a":
        by_date = False
    elif how_to_search == "b":
        by_date = True
    else:
        print ("I don't recognize this option, I'll take you to main
program")
        return
    search_qquery = input("Search entry in diary: ").strip()
```

```
view_entries(search_qquery, by_date)
```

En suma, arreglando todo nos queda un archivo con los siguientes errores:

Check results

=====

```
E265:6:1:block comment should start with '# '
E265:12:5:block comment should start with '# '
E265:13:5:block comment should start with '# '
E265:14:5:block comment should start with '# '
E265:17:5:block comment should start with '# '
E265:25:1:block comment should start with '# '
E265:33:1:block comment should start with '# '
E265:54:1:block comment should start with '# '
E265:67:1:block comment should start with '# '
E262:72:23:inline comment should start with '# '
E265:73:9:block comment should start with '# '
E501:77:80:line too long (101 > 79 characters)
W291:77:102:trailing whitespace
E501:78:80:line too long (103 > 79 characters)
W291:78:104:trailing whitespace
E501:79:80:line too long (97 > 79 characters)
E262:83:70:inline comment should start with '# '
E501:83:80:line too long (98 > 79 characters)
E265:113:1:block comment should start with '# '
E265:133:1:block comment should start with '# '
E501:136:80:line too long (87 > 79 characters)
E265:142:1:block comment should start with '# '
E265:147:1:block comment should start with '# '
```

El formato de éste txt separa cada columna con el carácter “：“, en la primer columna va el número de error, la segunda es el número de linea en el código, la tercera, es la posición del carácter ofensivo en la linea, y la cuarta, describe el tipo de error.

### Arreglando el código de agenda.py para que cumpla con PEP8

Al igual que en diary.py, decidí ignorar algunos errores. El archivo corregido con PEP queda:

```
#####
#Acá va a definir las funciones que
quiero#####
#####
#####Agregar contacto
#####
```

```
def agregar_contacto():
    print("agregar contacto nuevo\n")
    nombre = input("Ingresa el nombre de la persona \n")
    if nombre in agenda.keys():
        print("El contacto ya existe")
    else:
        numero_de_telefono = input("Ingresa el numero de
telefono\n")
        agenda[nombre] = numero_de_telefono
#####
#####Remover contacto
#####

def remover_contacto():
    nombre = input("Escribe el nombre del contacto tal cual aparece
en la agenda:\n")
    if nombre in agenda.keys():
        del agenda[nombre]
    else:
        print("El contacto que quieres borrar no esta en la agenda")
#####

#####Ver un contacto
solo#####

def ver_contacto():
    nombre = input("Escribe el nombre de la persona tal cual esta en
la agenda:")
    if nombre in agenda.keys():
        print ("El contacto que buscas es este:")
        print ("Nombre: "+nombre+" Telefono:"+agenda[nombre])
    else:
        print("El contacto que buscas no esta en la agenda")
#####

#####Ver Toda la
agenda#####

def desplegar_agenda():
```

```
if agenda:
    print("Estos son tus contactos:")
    print ("Nombre | Telefono")
    for nombre, telefono in agenda.items():
        print (nombre+ " | "+telefono)
else:
    print("La agenda esta vacia")
#####
#####Actualizar
contacto#####

def actualizar_contacto():
    nombre = input("Ingresa el nombre del contacto\n")
    if nombre in agenda.keys():
        agenda[nombre] = input("Ingresa el nuevo numero\n")
    else:
        print("El contacto no existe")
#####
#####Inicializador de
variables#####
agenda = dict()
continuar = True
#####
#####Programa
Principal#####
print ("Bienvenido a tu agenda")
while continuar:
    print ("Lo que puedes hacer es:")
    print ("1) Ver tu agenda")
    print ("2) Buscar un contacto")
    print ("3) Agregar un contacto nuevo")
    print ("4) Eliminar un contacto")
    print ("5) Actualizar contacto")
    print ("6) Salir de la agenda")
    try:
        orden = int(input("¿Qué deseas hacer? (marca la opcion con un
numero)\n"))
    except ValueError:
        print("La opcion no es reconocida")
    else:
```

```
if orden == 1:  
    desplegar_agenda()  
elif orden == 2:  
    ver_contacto()  
elif orden == 3:  
    agregar_contacto()  
elif orden == 4:  
    remover_contacto()  
elif orden == 5:  
    actualizar_contacto()  
elif orden == 6:  
    continuar = False  
else:  
    print("La orden no es reconocida")  
  
print ("Gracias por usar la agenda")
```

Y el archivo con los errores que decidí ignorar:

```
Check results  
=====
```

```
E265:1:1:block comment should start with '# '  
E265:4:1:block comment should start with '# '  
E265:17:1:block comment should start with '# '  
E501:17:80:line too long (80 > 79 characters)  
E501:21:80:line too long (85 > 79 characters)  
E501:26:80:line too long (80 > 79 characters)  
E265:29:1:block comment should start with '# '  
E501:29:80:line too long (80 > 79 characters)  
E501:33:80:line too long (81 > 79 characters)  
E501:39:80:line too long (80 > 79 characters)  
W291:39:81:trailing whitespace  
E265:41:1:block comment should start with '# '  
E501:41:80:line too long (80 > 79 characters)  
E501:52:80:line too long (81 > 79 characters)  
E265:54:1:block comment should start with '# '  
E501:54:80:line too long (81 > 79 characters)  
E501:63:80:line too long (80 > 79 characters)  
E265:65:1:block comment should start with '# '  
E501:65:80:line too long (80 > 79 characters)  
E501:68:80:line too long (80 > 79 characters)  
E265:70:1:block comment should start with '# '  
E501:70:80:line too long (80 > 79 characters)  
E501:81:80:line too long (82 > 79 characters)
```

## Mensajes de error

Existen infinidades de mensajes de error, pero los que me encontré yo fueron:

Error	Qué significa
Trailing Whitespace	Por alguna razón, hay espacios vacíos al final de las declaraciones
line too long	La linea tiene mas de 79 caracteres
block comment should start with '#'	Entre el símbolo del comentario y el comentario en sí, debe haber un espacio
Expected 2 blank lines, found 1	Entre función y función debe haber como mínimo 2 espacios
Missing whitespace after ","	Dentro de una función, los parámetros que se le pasan deben estar separados por la coma y un espacio en blanco
whitespace before ":"	No debe haber ningún espacio entre la declaración y los 2 puntos
to many blanklines (x)	Demasiadas líneas en blanco (x = cantidad de líneas)
indentation is not multiple of four	La indentación no es múltiplo de 4
Missing whitespace around operator	No hay espacios entre el nombre de la variable, el signo de igual y tampoco entre el signo de igual y el valor

### El Zen (o PEP20) de Python

Es una filosofía que indica cómo hacer las cosas y pensar en Python:

*"Beautiful is better than ugly.*

*Explicit is better than implicit.*

*Simple is better than complex.*

*Complex is better than complicated.*

*Flat is better than nested.*

*Sparse is better than dense.*

*Readability counts.*

*Special cases aren't special enough to break the rules.*

*Although practicality beats purity.*

*Errors should never pass silently.*

*Unless explicitly silenced.*

*In the face of ambiguity, refuse the temptation to guess.*

*There should be one-- and preferably only one --obvious way to do it.*

*Although that way may not be obvious at first unless you're Dutch.*

*Now is better than never.*

*Although never is often better than \*right\* now.*

*If the implementation is hard to explain, it's a bad idea.*

*If the implementation is easy to explain, it may be a good idea.*

*Namespaces are one honking great idea -- let's do more of those!"*

### **Traducción y significado de algunos de los aforismos:**

Hermoso es mejor que feo: O sea, si hay más de una implementación posible, usar la más elegante.

Python está diseñado de forma tal que, si se escribe bien el código, al seguir ésta regla de implementación, es como si se estuviera leyendo un texto en inglés.

Explícito es mejor que implícito: En Python no se suelen hacer cosas “detrás de cámaras”, entonces lo que se recomienda es no esconder lo que se está haciendo, que lo que se esté desarrollando sea muy claro. Es por ésto que existen funciones como **del** que es para explicitar que se van a borrar cosas.

Simple es mejor que complejo: Tiene que ver con la primera linea. De haber más de una implementación, hacer la más simple posible. Por otra parte, a diferencia de otros lenguajes, Python no es “verbose”, por lo cual, escribir lo menos posible.

Los errores nunca se deben de pasar por alto (A menos que sea silenciado explícitamente): Con ésto se quiere decir que, apenas se detecte un error, corregirlo antes de seguir adelante con el programa que se esté armando. Siempre tratar de usar **try-except-else** para capturar los errores y que el programa crusee.

Para ver el PEP20 dentro de la consola de Python solo hay que importarlo:

```
>>>import this
```

## Introducción a Flask

### Introducción

Flask es un micro-framework para Python basado Werkzeug y Jinja2 (y buenas intenciones).

O sea es un framework para desarrollo web usando Python.

Fué creado en 2010 por Armin Ronacher siendo una librería chica para hacer pequeños desarrollos a diferencia, e introducción, a otros frameworks como Django.

### **Definición de framework**

Un framework es una o varias librerías con mucho código que facilitan la existencia al programar, y en el caso particular de Django y Flask, para desarrollo web.

En el caso particular, se puede escribir por uno mismo en python para aplicaciones web, pero al usar frameworks, facilita el debugging y es un framework mantenido por su comunidad, lo cual hace que se cometan menos errores y evita reinventar la rueda.

### Instalación de Flask en su versión más reciente (En Debian)

Primero hay que instalar setuptools para python3:

```
sudo apt-get install python3-setuptools
```

Como en el repositorio está una versión antigua, hay que bajar la más actual

(<https://pypi.org/project/Flask/#files>):

Filename, size	File type	Python version	Upload date	Hashes
<a href="#">Flask-1.1.1-py2.py3-none-any.whl</a> (94.5 kB)	Wheel	py2.py3	Jul 8, 2019	<a href="#">View</a>
<a href="#">Flask-1.1.1.tar.gz</a> (625.5 kB)	Source	None	Jul 8, 2019	<a href="#">View</a>

Luego se descomprime el archivo, y en la linea de comando:

```
sudo python3 setup.py
```

Y con eso queda lista la instalación de Flask

Para usarlo, se va a seguir la documentación:

<https://flask.palletsprojects.com/en/1.1.x/> donde se describe el uso básico y hay ejemplos de aplicaciones

### Hola mundo en Flask

Antes que nada, se recomienda crear una carpeta nueva para hacer los proyectos en flask.

En linux se usa el comando mkdir y luego el comando cd para ir a la carpeta:

```
jenifer@azeroth:~$ cd Documentos/Backup_2019/qbit/Prog_Py/
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py$ mkdir flask
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py$ cd flask
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/flask$ █
```

Para usar flask, lo primero que hay que hacer es crear un archivo app.py donde lo primero que se hace es importar flask:

```
from flask import Flask
```

Luego se tiene que hacer que la app apunte a flask, o sea, crear un objeto llamado app, el cual apunte a flask, o sea, lo que se crea es una instancia (app) de la clase Flask.

```
app = Flask(__name__)
```

Lo que se está haciendo con ésto es que nuestra app siempre apunte a sí misma.

Después lo que hay que crear es una ruta, la cual le dice a Flask qué URL debe activar ésta app.

```
@app.route("/")
```

Como no tenemos nada, se pone el "/"

Por último se crea la función que va a ejecutar la app:

```
def hello_world():
    return "Hola mundo!"
```

Por último, hay que decirle al archivo app.py que, si se lo llama como main program, presente al usuario la instancia.

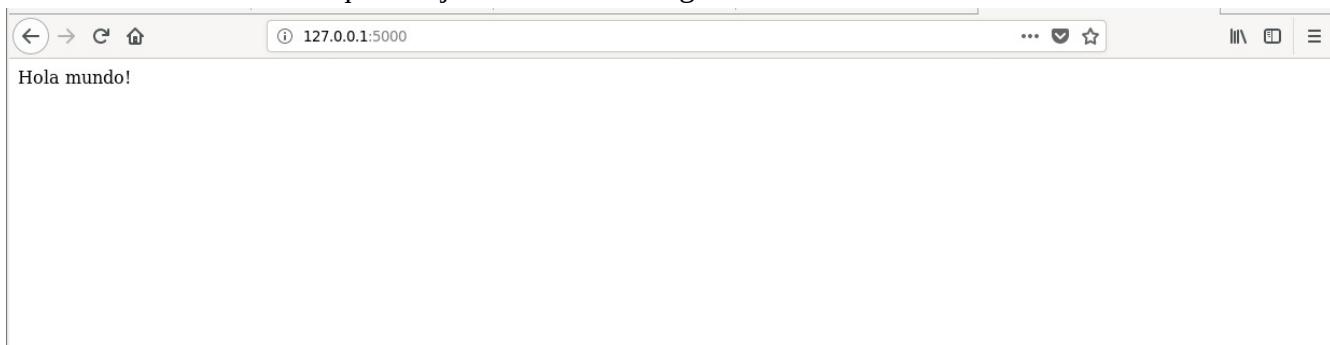
```
if __name__ == "__main__":
    app.run()
```

El resultado de ejecutar app.py:

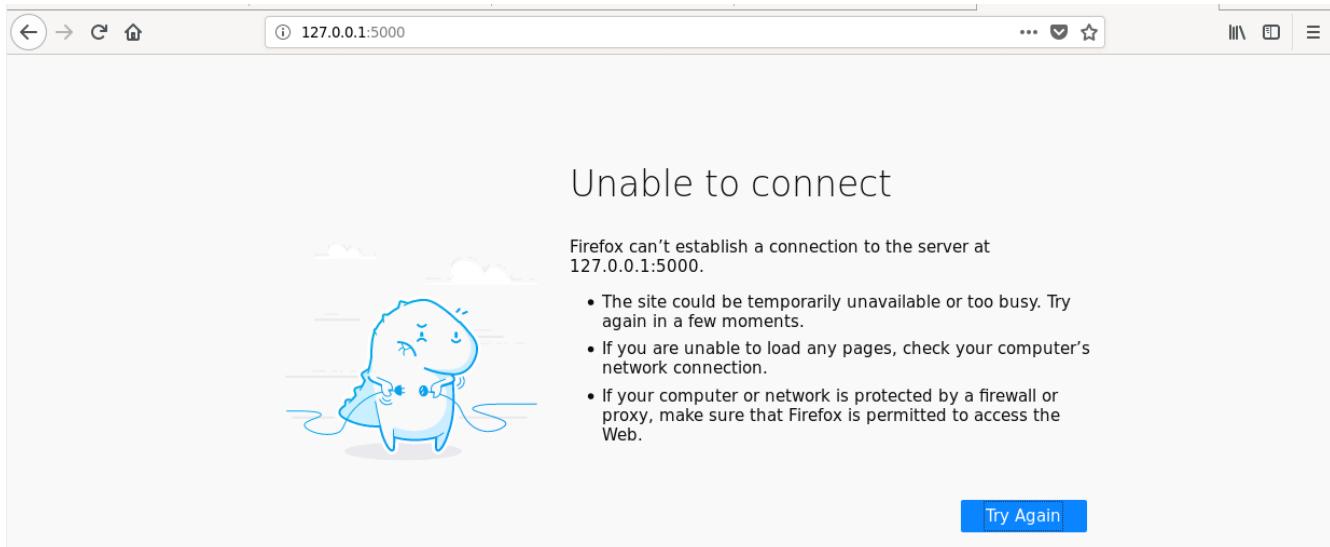
```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/flask$ python3 app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

^Cjenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/flask$
```

Como se ve, está andando todo relativamente bien, salvo por los Warnings, porque, al momento de hacer la guía, toma a la computadora como un entorno de desarrollo, no como uno de producción. Si ahora se consulta la IP que arroja Flask en el navegador nos dá todo correctamente:



Atención: Ésta IP solo funciona mientras app.py está corriendo, si se interrumpe el programa, el navegador va a largar error.

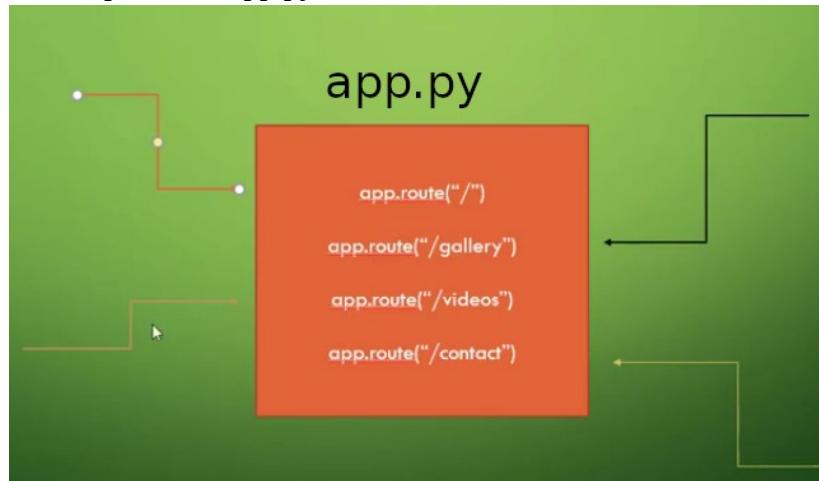


## Rutas

A todo lo que se desarrolla en Python usando Flask, se lo llama apps, aunque el nombre del archivo no depende de eso.

Las apps de Flask lo que hacen es encargarse de recibir las peticiones web que vienen hacia todo lo que el desarrollador cree y se encarga de redirigir las a la función, que en Flask se le llama **vistas**, lo cual hace a través de las **rutas** (o routes).

Por ejemplo, si se tiene la aplicación app.py:



App.py se encarga de resolver todas las peticiones que le llegan y las redirige a las funciones creadas dentro de la misma ya sea desplegar un texto, una imagen, videos, una lista de contactos, en fin, lo que tenga la app desarrollada adentro y la route es la forma que tiene Flask de saber cuál función usar.

En el ejemplo de hello\_world:

```
@app.route("/")
```

Va a buscar en la ip, "/", y por eso imprimió "Hola Mundo". Pero en su lugar, se pueden hacer varias cosas, por ejemplo, desplegar consultas en Json, una respuesta en xml para hacer un servicio web, haber desplegado cosas en HTML y CSS para desplegar cosas en un sitio web.

Las peticiones se guardan en @<variable>.

Por ejemplo: si queremos que se despliegue una imagen cuando el usuario quiera visitar el sitio:  
<sitio>/images

El nombre de la función puede ser cualquier cosa por lo cual, es recomendable que su nombre tenga consistencia con lo que se está desarrollando.

En el caso del ejemplo:

```
from flask import Flask
```

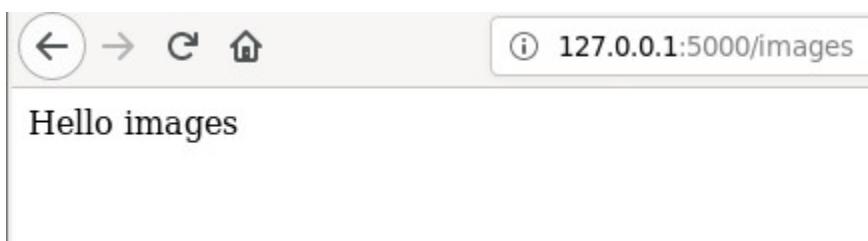
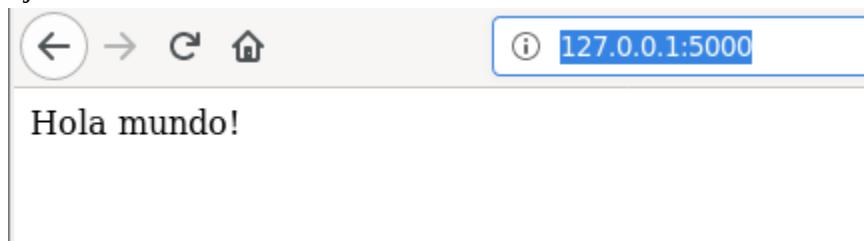
```
app= Flask(__name__)
```

```
@app.route("/")
def hello_world():
    return "Hola mundo!"
```

```
@app.route("/images")
def hello_images():
    return "Hello images"
```

```
if __name__ == "__main__":
    app.run()
```

Da que cuando se ejecuta:



O sea, cada vez que el usuario haga una petición de images, Flask va a desplegar su ruta correspondiente.

Por otra parte, si se quiere que actualice los cambios en el código, y que el cambio sea actualizado enseguida al guardar el script de Flask:

```
if __name__ == "__main__":
    app.run(debug = True)
```

Con el parámetro debug = True lo que se hace es que no sea necesario cerrar y volver a ejecutar el archivo app.py para que, en vez de decir “Hola mundo”, diga “Hello World”.

En la consola:

```
^Cjenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/flask$ python3 app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 267-253-681
127.0.0.1 - - [22/Sep/2019 10:59:18] "GET / HTTP/1.1" 200 -
 * Detected change in '/home/jenifer/Documentos/Backup_2019/qbit/Prog_Py/flask/app.py', reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 267-253-681
127.0.0.1 - - [22/Sep/2019 10:59:40] "GET / HTTP/1.1" 200 -
```

Ahora lo que cambió es que está avisando que está corriendo en modo debug, el cual es bueno para desarrollo, pero se puede quitar cuando el software pase a producción.

Otra cosa, cuando el debugger está activo, avisa en la consola que un cambio fué detectado y reinicia el script (app.py en nuestro caso)

## Decorators

Lo que hacen los decoradores es agregar funcionalidades a funciones concretas cuando ya está abstracto su algoritmo, pero se quieren agregar cosas nuevas. O sea son funciones de por si, reciben como parámetro otra función y su return es una función nueva a partir de la vieja.

Ejemplo:

```
# A, B y C son funciones
# A recibe como parámetro a B y devuelve C
```

```
def decorador(func): # función A y B
    def funcion_nueva():
        pass
    return funcion_nueva # función C

def saluda():
```

```
print("Hola Mundo")
```

```
saluda()
```

Ahora, ésto es útil si se quiere ejecutar código antes o después de ejecutar la función que entra al decorador:

```
def decorador(func): # función A, B (func)
    def funcion_nueva():
        #Aregar código, or ejemplo:

        func()
        #Aregar código
    return funcion_nueva # función C
```

Ahora, para llamar a la función decorador:

```
def decorador(func): # función A
    def funcion_nueva():
        #Aregar código, por ejemplo
        print("Acá vamos a ejecutar la función")
        func()
        #Aregar código, por ejemplo
        print("Acá termina de ejecutar la función")
    return funcion_nueva # función C
```

```
@decorador
def saluda():
    print("Hola Mundo")
```

```
@decorador
def saluda2():
    print("Hello World")
```

```
saluda()
print("-"*10)
saluda2()
```

Lo cual va a dar:

```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/flask$ python3 decorator_ex  
ample.py  
Acá vamos a ejecutar la función  
Hola Mundo  
Acá termina de ejecutar la función  
-----  
Acá vamos a ejecutar la función  
Hello World  
Acá termina de ejecutar la función
```

Por otra parte, si la función a decorar recibe parámetros, hay que modificar el decorador:

```
def decorador(func): # función A  
    def funcion_nueva(*args, **kwargs):  
        #Aregar código, por ejemplo  
        print("Acá vamos a ejecutar la función")  
        func(*args, **kwargs)  
        #Aregar código, por ejemplo  
        print("Acá termina de ejecutar la función")  
    return funcion_nueva # función C
```

En nuestro ejemplo:

```
# A, B son C funciones  
# A recibe como parámetro a B y devuelve C
```

```
def decorador(func): # función A  
    def funcion_nueva(*args, **kwargs):  
        #Aregar código, por ejemplo  
        print("Acá vamos a ejecutar la función")  
        func(*args, **kwargs)  
        #Aregar código, por ejemplo  
        print("Acá termina de ejecutar la función")  
    return funcion_nueva # función C
```

```
@decorador  
def saluda():  
    print("Hola Mundo")
```

```
@decorador  
def saluda2():  
    print("Hello World")
```

```
@decorador  
def suma(num1, num2):  
    print(num1 + num2)  
saluda()
```

```

print("-"*20)
saluda2()
print("-"*20)
suma(num1=20, num2=30)
print("-"*20)
suma(40, 80)
nos da:

```

```

jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/flask$ python3 decorator_ex
ample.py
Acá vamos a ejecutar la función
Hola Mundo
Acá termina de ejecutar la función
-----
Acá vamos a ejecutar la función
Hello World
Acá termina de ejecutar la función
-----
Acá vamos a ejecutar la función
50
Acá termina de ejecutar la función
-----
Acá vamos a ejecutar la función
120
Acá termina de ejecutar la función

```

Por último, si lo que queremos es que retorne valores en vez de imprimir:

```

# A, B son C funciones
# A recibe como parámetro a B y devuelve C

```

```

def decorador(func):    # función A
    def funcion_nueva(*args, **kwargs):
        #Aregar código, por ejemplo
        print("Acá vamos a ejecutar la función")
        resultado = func(*args, **kwargs)
        #Aregar código, por ejemplo
        print("Acá termina de ejecutar la función")
        return resultado
    return funcion_nueva    # función C

@decorador
def suma(num1, num2):
    return num1 + num2

```

O sea hay que hacer un **return** en la función original y dentro del la función decoradora  
Entonces, para que un decorador funcione, se necesitan 3 funciones:

1. Wrapper: La que engloba todo, que en nuestro caso es la función A.

2. La función a decorar: Es la función que recibe nuestro decorador, en nuestro caso func o B
3. Función de salida

## Decoradores y Flask

En el caso de flask

```
@app.route("/")
```

```
def hello_world(nombre = "stranger"):  
    nombre = request.args.get("name", nombre)  
    return "Hello " + nombre
```

La función decoradora es `.route` que lo que hace es procesar las rutas y la función a ser decorada es `hello_world`.

## Query String

Ésto se usa cuando se requiere que el usuario ingrese información, por ejemplo, que ingrese texto, o quiere ver una imagen en particular.

Para el caso de texto, se usan los Query Strings, que es todo lo que se pone en un navegador después de un signo de interrogación y donde hay keys y values, por ejemplo:

<http://127.0.0.1:5000/?name=pepe&edad=18> (desplegar en el navegador en caso de que el nombre sea igual a Pepe y la edad sea igual a 18).

Para que flask entienda que va a recibir entradas del usuario, se hace:

```
from flask import request
```

Y se modifica de la siguiente manera:

```
@app.route("/")
```

```
def hello_world(nombre = "stranger"):  
    nombre = request.args.get("name", nombre)  
    return "Hello " + nombre
```

Si el usuario no pone ningún nombre (<http://127.0.0.1:5000/>), va a imprimir “Hello stranger”, pero si pone en el navegador por ejemplo <http://127.0.0.1:5000/?name=Jenifer> va a imprimir “Hello Jenifer”, pues en la linea `request.args.get("name", nombre)` lo que ocurre es que, primero accede al request, luego a los argumentos (args), va a cargar en un parámetro llamado name, el valor que le pase el usuario.

Por otra parte, si el usuario pone en el request algo que no está definido en `request.args.get()` también va a imprimir “Hello stranger”.

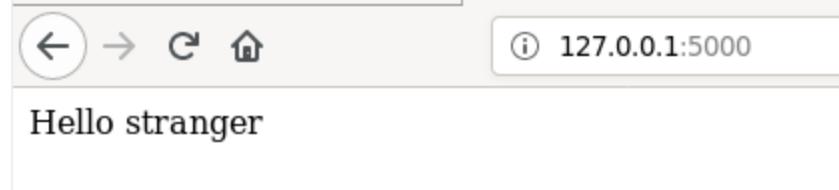
Si se quiere hacer otro ejemplo:

```
@app.route("/")
```

```
def hello_world(nombre = "stranger", edad = None):  
    nombre = request.args.get("name", nombre)  
    edad = request.args.get("age", edad)  
    if edad:  
        return "Hello {}\\n Tienes {} años".format(nombre, edad)
```

```
else:  
    return "Hello {}".format(nombre)
```

En éste caso, si la variable edad no se la pasa como parámetro, solo escribe o “Hello” y el nombre o “Hello stranger”



Y si se especifica la edad:



### Route Parameters

Las query strings son muy útiles para obtener parámetros, pero cuando son muchos datos, se puede volver un dolor de cabeza manejarlas.

Además, cuando son muchos queries, la url de acceso puede volverse bastante fea.

Para evitar eso, es que, en ésta sección particular, se va a tratar de obtener los queries del usuario a través de los parámetros de las rutas.

Supongamos ahora que queremos hacer una página que lo que haga es sumar 2 números que ponga el usuario. Entonces:

```
@app.route("/suma")  
def suma(num1=0, num2=0):
```

```

num1 = int(request.args.get("num1", num1))
num2 = int(request.args.get("num2", num2))
return "{} mas {} da {}".format(num1, num2, num1 + num2)

```

Pero una forma mejor para que la URL a usar no sea /suma?num1=1&num2=2 es:

```

@app.route("/suma/<num1>/<num2>")
def suma(num1=0, num2=0):
    num1 = int(request.args.get("num1", num1))
    num2 = int(request.args.get("num2", num2))
    return "{} mas {} da {}".format(num1, num2, num1 + num2)

```

Observaciones:

- `@app.route("/suma/<num1>/<num2>")` es para que funcione cuando se pone en la URL por ejemplo /suma/1/2, de:  
1 mas 2 da 3
- `num1 = int(request.args.get("num1", num1))` es para que cuando haga la suma en .format, no los tome las variables como string y concatene, sino que haga la suma. Si ésto no se hace, el resultado de URL/suma/1/2 es: 1 mas 2 da 12 en vez de 1 mas 2 da 3.
- Si se pone en la URL /suma/1/<cualquier\_otra\_cosa\_menos\_un int>, cuando busque an la página, le va a dar un value error.

Por otra parte, se lo puede mejorar más todavía indicando en la URL que lo que estamos ingresando son números:

```

@app.route("/suma/<int:num1>/<int:num2>")
def suma(num1=0, num2=0):
    return "{} mas {} da {}".format(num1, num2, num1 + num2)

```

Aquí se puede observar que `@app.route("/suma/<int:num1>/<int:num2>")` ya está indicando que lo que espera son integers, con lo cual, no es necesaria la conversión en .format para hacer la suma.

## Limpieza de URLs

En ésta sección se probará limpiar las URLs para que sean más legibles

En el código ejemplo:

```

@app.route("/")
def hello_world(nombre="stranger", edad=None):
    nombre = request.args.get("name", nombre)
    edad = request.args.get("age", edad)
    if edad:
        return "Hello {}\n Tienes {} años".format(nombre, edad)
    else:
        return "Hello {}".format(nombre)

```

Resulta que hay otra forma más prolífica y que obtenga el nombre del usuario (sí como us edad).

Para eso se hace lo siguiente:

```
@app.route("/")
@app.route("/<nombre>")
@app.route("/<int:edad>")
@app.route("/<nombre>/<int:edad>")
def hello_world(nombre="stranger", edad=None):
    if edad:
        return "Hello {} \n Tienes {} años".format(nombre, edad)
    else:
        return "Hello {}".format(nombre)
```

Con `@app.route("/<nombre>")`, `@app.route("/<int:edad>")` y `@app.route("/<nombre>/<int:edad>")` ya obtiene el nombre de la barra de la URL y ya no se necesita llamar al método request de flask, por lo cual, se elimina el import de esa función (principio KISS).

Por otra parte, cabe destacar que, como no hay otras rutas que coincidan dentro de la app, va a buscar dentro de la función que está declarada abajo.

Por otra parte, la función de suma puede quedar así:

```
@app.route("/suma/<int:num1>/<int:num2>")
@app.route("/suma/<float:num1>/<int:num2>")
@app.route("/suma/<int:num1>/<float:num2>")
@app.route("/suma/<float:num1>/<float:num2>")
def suma(num1=0, num2=0):
    return "{} mas {} da {}".format(num1, num2, num1 + num2)
```

Ésto se hace para que pueda aceptar floats y numeros enteros a la vez. Si alguno de los routes no está, la página va a dar un error 404 porque no va a encontrar cuando se ponga en la URL/suma/(num1)/(num2)

### Desplegado de código HTML

En las secciones anteriores se vió cómo desplegar texto plano en Flask, pero las páginas web no funcionan así, con solo texto plano, sino que usan desplegar colores, reproducir videos embeded en las páginas, para lo cual se usa HTML, CSS y JavaScript.

Para HTML:

#### **Primera forma:**

Se utilizan 3 comillas de abrir y 3 para cerrar porque es la forma que Flask entiende que lo que va en el medio es código HTML.

Recordar del curso de TICs software y TICs programación, que el html tiene un encabezado y un cuerpo, entonces, para hacerlo en Flask con la función suma:

```
def suma(num1=0, num2=0):
    return '''
<!DOCTYPE html>
```

```
<html>
    <head></head>
    <body>
        </body>
</html>
'''
```

En el Head va el título de la página:

```
<head>
    <title>Sumar</title>
</head>
```

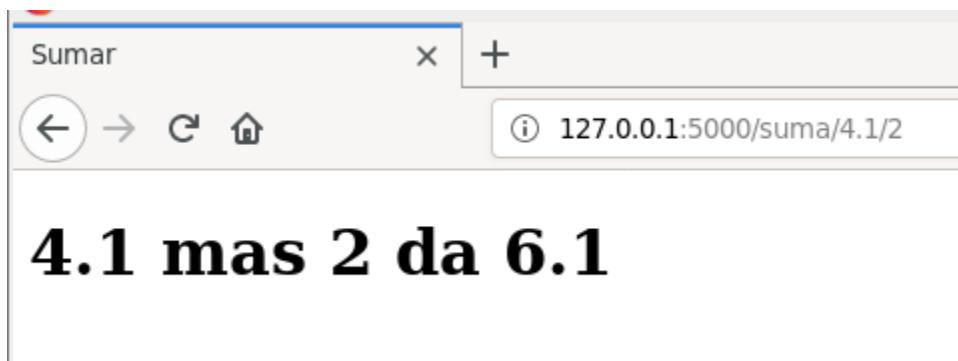
Y en el cuerpo de la página, un título (head) grande:

```
<h1>{} mas {} da {}</h1>
```

Entonces, para que python entienda que en los corchetes tiene que poner las variables, la función suma queda así:

```
def suma(num1=0, num2=0):
    return ''
<!DOCTYPE html>
<html>
    <head>
        <title>Sumar</title>
    </head>
    <body>
        <h1>{} mas {} da {}</h1>
    </body>
</html>
''.format(num1, num2, num1 + num2)
```

Si se corre el script se vé que la pestaña de la página tiene su título y que el cuerpo de la página se despliega como un Header:



Si se elige ver el código fuente de la página (click derecho → ver código fuente) se muestra el código HTML:



The screenshot shows a browser window with the URL `view-source:http://127.0.0.1:5000/suma/4.1/2`. The page content is a simple HTML document with line numbers 1 through 11 on the left. The code is as follows:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Sumar</title>
5   </head>
6   <body>
7     <h1>4.1 mas 2 da 6.1</h1>
8   </body>
9 </html>
10
11
```

Éste método es fácil, pero es medio engorroso si se quieren hacer multiples páginas. Es por ésto que hay otras maneras de insertar código HTML para una página, lo cual consiste en usar templates (ver más abajo)

## Segunda forma: Usando templates

Para usar un template, se deben crear un directorio llamado así porque si no, Flask no lo encuentra. Dentro de ese directorio, se guardan todos los archivos HTML que van a servir de templates. Primero se crea un archivo y se extrae del programa (en nuestro caso, de la app.py) las líneas correspondientes que hacen la operación, creando un nuevo archivo, en nuestro ejemplo: suma.html

Luego en el archivo principal se hace un import para que pueda manejar los templates:

```
from flask import render_template
```

y en donde estaba la función:

```
def suma(num1=0, num2=0):
    return render_template("suma.html", num1 = num1, num2 = num2)
```

`render_template` pertenece a Jinja2 que es lo que Flask utiliza para parsear archivos HTML, toma como primer parámetro el archivo del template y el resto de los parámetros, las variables que va a sustituir en el archivo html:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sumar</title>
  </head>
  <body>
    <h1>{{ num1 }} mas {{ num2 }} da {{ num1 + num2 }}</h1>
  </body>
</html>
```

Nota: Los espacios entre los corchetes y el nombre de la variable son importantes, porque si no, jinja da error

Hay otra variante de ésto que es usando desempaquetamiento de diccionarios:

```
def suma(num1=0, num2=0):
    context= {"num1" : num1, "num2" : num2}
    return render_template("suma.html", **context)
```

Nota: Los keys del diccionario tienen que ser de tipo string porque si no, no reconoce el patrón (da TypeError).

La forma de diccionario es cómoda cuando se tienen que pasar varios parámetros.

Ahora, si se hace lo mismo para la parte de nombres y edades de la app ejemplo, el código al final queda:

```
from flask import Flask
from flask import render_template

app = Flask(__name__)

# HTML con templates para el nombre#####
@app.route("/")
@app.route("/<nombre>")
@app.route("/<int:edad>")
@app.route("/<nombre>/<int:edad>")
def hello_world(nombre="stranger", edad=None):

    context = {"name": nombre, "age": edad}

    if edad:
        return render_template("Hola_edad.html", **context)
    else:
        return render_template("index.html", name=nombre)
# #####
# HTML con templates para la suma#####

@app.route("/suma/<int:num1>/<int:num2>")
@app.route("/suma/<float:num1>/<int:num2>")
@app.route("/suma/<int:num1>/<float:num2>")
@app.route("/suma/<float:num1>/<float:num2>")
def suma(num1=0, num2=0):
    context= {"num1": num1, "num2": num2}
    return render_template("suma.html", **context)
```

```
# #####
```

```
if __name__ == "__main__":
    app.run(debug=True)
```

Y los templates:

index.html:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Saludo</title>
    </head>
    <body>
        <h1>Hola {{ name }}</h1>
    </body>
</html>
```

Hola\_edad.html:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Saludo</title>
    </head>
    <body>
        <h1>Hola {{ name }}.
        Tienes {{ age }} años</h1>
    </body>
</html>
```

Y finalmente, suma.html:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Sumar</title>
    </head>
    <body>
        <h1>{{ num1 }} mas {{ num2 }} da {{ num1 + num2 }}</h1>
    </body>
</html>
```

Notar que ahora, el archivo HTML principal se llama index.html. La razón por la que se el cambió el nombre es porque, por convención, y de hecho en otros sistemas de desarrollo web es obligatorio, la página principal se le llama así.

### Tercera forma: usando una planilla general (Templates y Herencia)

Como se vió anteriormente, el cuerpo de todos los htmls de las plantillas generadas tienen la misma estructura: un head, un title y un body y lo único que cambia el título de la página y el body que es lo que despliega en el navegador.

Para solucionar ésto, se puede crear un solo template inheritance, que va a tener un HTML base y en app.py va a decirle que desplegar.

Para eso se crea un archivo HTML en la carpeta templates que puede llamar de cualquier manera, pero por convención, se llamará base\_layout y contendrá todo lo que es común en todos los HTML generados:

**Template padre:**

```
<!DOCTYPE html>
<html>
    <head>
        <title>{{ block title }} {{ endblock }}</title>
    </head>
    <body>
        <h1>{{ block content }} {{ endblock }}</h1>
    </body>
</html>
```

Las llaves y %block <bloque>% son para que Jinja2 entienda que es en esos lugares que tiene que sustituir.

Luego se editan los HTML **Templates hijas**:

index.html:

```
{% extends 'base_layout.html' %}

{% block title %}Saludo{{ endblock %}

{% block content %}<h1>Hola {{ name }} </h1>{{ endblock %}}
```

Hola\_edad.html:

```
<!{ %extends 'base_layout.html' %}>

{% block title %}Saludo{{ endblock %}

{% block content %}<h1>Hola {{ name }} Tienes {{ age }} años</h1>{{ endblock %}}
```

suma.html:

```
{% extends 'base_layout.html' %}
```

```
{% block title %}Suma{% endblock %}

{% block content %} <h1> {{ num1 }} mas {{ num2 }} da {{ num1 +
num2 }} </h1> {% endblock %}
```

La importancia de tener herencia el los templates HTML tiene varias ventajas (opinión personal), entre ellas:

- No se tiene que escribir tanto código en las templates hijas.
- Se arma un orden jerárquico de las templates, lo cual hace que sea más fácil de debuggear
- Da consistencia al código, si falla una clase hija, y es independiente de las otras clases hijas, el error no se propaga.
- Resuelve patrones que se repiten, por ejemplo el logo de una página, el footer donde aparecen los datos de copyright, los menues importantes, poniendolos todos en el Base Layout.  
Entonces, si se quiere modificar algo, no hace falta ir a cada una de las Templates hijas, con modificar el Base Layout es suficiente.
- ...etc

## Static

Static es una carpeta que toma Flask del proyecto que estamos armando, que le indica que contiene todo el código para CSS y JavaScript.

La carpeta debe ir en el directorio raíz del proyecto al igual que templates.

Dentro de ésta carpeta, se va a poner un archivo ejemplo llamado style.css:

```
body{
    background: #228844;
    color : black;
}
```

Y dentro de base\_layout:

```
<!DOCTYPE html>
<html>
    <head>
        <title>{{ block title }}</title>
        <link rel="stylesheet" href="/static/style.css">
    </head>
    <body>
        <h1>{{ block content }}</h1>
    </body>
</html>
```

## Reto de la sección:

El reto consiste en que la página pueda hacer suma, resta, multiplicación y división.

Para cumplirlo lo que se hizo fue modificar el archivo app.py donde se da la orden:

```
@app.route("/suma/<int:num1>/<int:num2>")
@app.route("/suma/<float:num1>/<int:num2>")
@app.route("/suma/<int:num1>/<float:num2>")
@app.route("/suma/<float:num1>/<float:num2>")
def suma(num1=0, num2=0):
    context= {"num1": num1, "num2": num2}
    return render_template("suma.html", **context)

@app.route("/resta/<int:num1>/<int:num2>")
@app.route("/resta/<float:num1>/<int:num2>")
@app.route("/resta/<int:num1>/<float:num2>")
@app.route("/resta/<float:num1>/<float:num2>")
def resta(num1=0, num2=0):
    context= {"num1": num1, "num2": num2}
    return render_template("resta.html", **context)

@app.route("/multiplicacion/<int:num1>/<int:num2>")
@app.route("/multiplicacion/<float:num1>/<int:num2>")
@app.route("/multiplicacion/<int:num1>/<float:num2>")
@app.route("/multiplicacion/<float:num1>/<float:num2>")
def multiplicacion(num1=0, num2=0):
    context= {"num1": num1, "num2": num2}
    return render_template("multiplicacion.html", **context)

@app.route("/division/<int:num1>/<int:num2>")
@app.route("/division/<float:num1>/<int:num2>")
@app.route("/division/<int:num1>/<float:num2>")
@app.route("/division/<float:num1>/<float:num2>")
def division(num1=0, num2=0):
    if num2 !=0:
        context= {"num1": num1, "num2": num2}
        return render_template("division.html", **context)
    else:
        return "No se puede dividir entre 0"
```

Y por otro lado, crear templates HTML para cada una de las operaciones con la única diferencia entre ellos, la linea donde hace la operación:

Resta:

```
{% block content %} <h1> {{ num1 }} menos {{ num2 }} da {{ num1 - num2 }} </h1> {% endblock %}
```

Multiplicación:

```
{% block content %} <h1> {{ num1 }} multiplicado por {{ num2 }} da {{ num1 * num2 }} </h1> {% endblock %}
```

Division:

```
{% block content %} <h1> {{ num1 }} dividido {{ num2 }} da {{ num1 / num2 }} </h1> {% endblock %}
```

## **Formularios**

En las páginas web es muy común que tengan la funcionalidad de llenar formularios, por ejemplo, en facebook, cuando se publica un estado, de hecho lo que se llena es un formulario.

También se usan formularios en algunas páginas para captar datos de sus usuarios.

Pero todo eso se reduce a que, un usuario ingresa información en un campo de formulario y luego, con un click o un enter, se guarda y se envía la información.

## **Armado de formularios**

Lo primero que se debe hacer cuando se quiere usar un formulario, es crearlo.

Para ésto se crea un template, por ejemplo contacto.html, que va a tener la misma estructura de las otras templates hijas en cuanto va a heredar del template base\_layout.html:

```
{% extends 'base_layout.html' %}
```

```
{% block title %}Contacto{% endblock %}

{% block content %}

(Acá va a ir todo el contenido)

{% endblock %}
```

Donde va todo el contenido:

```
{% block content %}
<form action = "index.html" methods="POST">
    <label for ="nombre">Nombre: </label><input type= "text" name="nombre" value=""><br>
    <label for ="e-mail">e-mail: </label><input type= "text" name="e-mail" value=""><br>
    <label for ="comentario">Comentario: </label><input type= "text" name="comentario" value="">
    <input type= "submit" value="submit"><br>
</form>
{% endblock %}
```

Lo que hace el código anterior es crear un formulario html donde pide el nombre, una dirección de correo electrónico, el comentario y crea el botón de submit la información.

Luego en el archivo de aplicación (app.py):

```
@app.route("/contacto")
def contenido():
    return render_template("contacto.html")
```

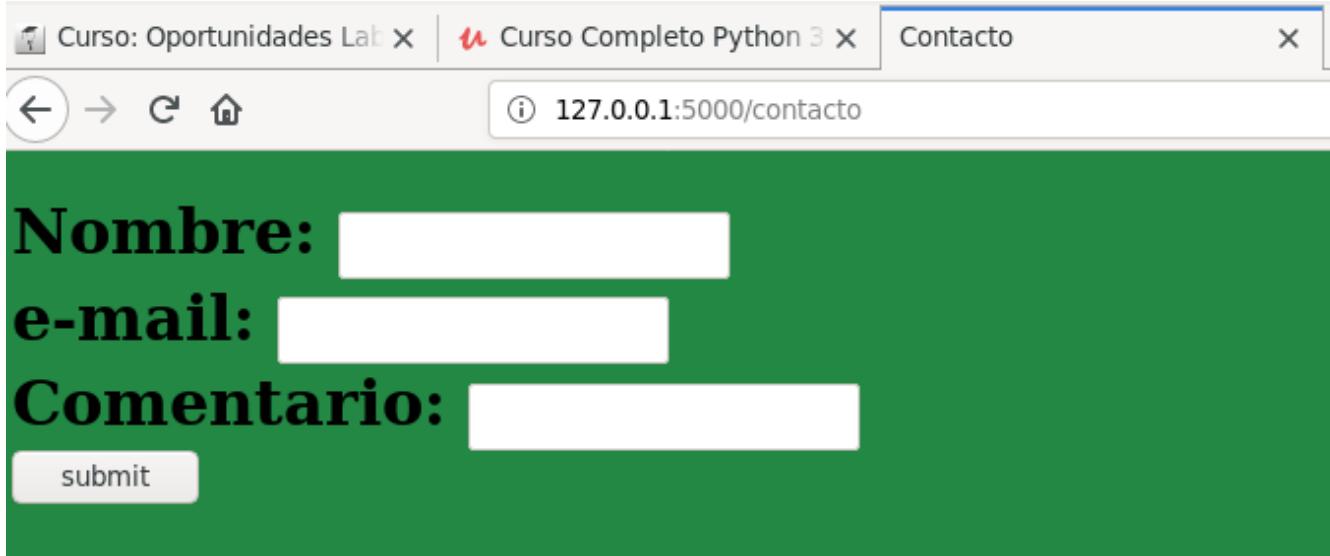
Y contacto.html va en la carpeta de templates y se escribe:

```
{% extends 'base_layout.html' %}

{% block title %}Contacto{% endblock %}

{% block content %}
<form action = "index.html" methods="POST">
    <label for ="nombre">Nombre: </label><input type= "text"
name="nombre" value=""><br>
    <label for ="e-mail">e-mail: </label><input type= "text" name="e-
mail" value=""><br>
    <label for ="comentario">Comentario: </label><input type= "text"
name="comentario" value=""><br>
    <input type= "submit" value="submit">
</form>
{% endblock %}
```

Con ésto, al ejecutar app.py e ir en el navegador a URL/contacto:



Submitir formularios

El problema con ésto, es que si se llena el formulario, va a ir a una dirección que no especificamos en ninguno de los casos de app.py:

The screenshot shows two browser windows. The top window displays a contact form with fields for Nombre (jenifer), e-mail (jenifer@malinator.com), and Comentario (abc), with a submit button. A blue arrow points down to the bottom window, which shows the URL 127.0.0.1:5000/index.html?nombre=jenifer&e-mail=jenifer%40malinator.com&comentario=abc. This URL is circled in red.

**Nombre:** jenifer

**e-mail:** jenifer@malinator.com

**Comentario:** abc

submit

127.0.0.1:5000/contacto

**Hola index.html**

127.0.0.1:5000/index.html?nombre=jenifer&e-mail=jenifer%40malinator.com&comentario=abc

O sea, fué a la página de saludo y como nombre tomó la cadena “index.html”.

Guardó en la URL los campos que se llenaron, pero no los puso en ninguna base de datos o en archivos ni nada por el estilo.

Para conseguir ésto, se va a crear una nueva view en app.py, la cual solo es accesible si se envió un formulario o POST:

```
@app.route("/enviado", methods=["GET", "POST"])
def enviado():
    return "Formulario enviado con éxito"
```

Y en el archivo html del formulario:

```
{% extends 'base_layout.html' %}

{% block title %}Contacto{% endblock %}

{% block content %}
<form action = "{{ url_for("enviado") }}>
    <label for="nombre">Nombre: </label><input type="text"
name="nombre" value=""><br>
```

```
<label for="e-mail">e-mail: </label><input type="text" name="e-mail" value=""><br>
<label for="comentario">Comentario: </label><input type="text" name="comentario" value=""><br>
<input type="submit" value="submit">
</form>
{%- endblock %}
```

Con amarillo fluorescente se indica cuál fué el cambio que se hizo en contacto.html, el cual le indica a jinja2 que, una vez mandados los datos, vaya a la página de enviado (que lo que hace es devolver el return de la función).

Con rosado fluorescente es lo que se hizo distinto a lo que hizo Aldominum en el video para que funcionara la página como se esperaba.

## Cookies

Ya en la sección anterior se vió cómo capturar datos de un formulario, pero el enviar correos ya escapa del scope del curso pues requeriría un server dedicado a eso y desarrollar un entorno para eso.

Lo que se va a hacer es dejar de lado la parte de mandar correos y se verá cómo guardar los datos en cookies.

Un cookie es un pequeño archivo de texto (de no más de 4kb) que crea un sitio web y que se guarda en el ordenador del usuario de forma temporal o de forma permanente en el disco duro del usuario (persistente cookie),

Este archivo permite reconocer a cada usuario y sus preferencias.

Para hacer ésto, con el formulario y redirigir al usuario a la página principal.

Primero hay que cambiar la sentencia de import:

```
from flask import Flask, render_template, redirect, url_for
@app.route("/enviado", methods=['GET', 'POST'])
def enviado():
    response = redirect(url_for("hello_world"))
```

Lo que hace en la variable response, es buscar la url correspondiente de la view con `url_for("hello_world")` y redirige a esa dirección con la función `redirect`.

Pero solo con el redirect no sirve de mucho.

Para crear una cookie se usa el método `.set_cookie`:

```
response.set_cookie(json.dumps(dict(request.form.items()))))
```

Acá lo que se dice es que se quiere crear una cookie que va a ser un archivo de tipo json que crea un diccionario con llaves y valores, por ejemplo nombre = Jenifer, e-mail = [jenifer@mailinator.com](mailto:jenifer@mailinator.com) y comentario = "abc"

O sea, va a crear una cookie en base a un diccionario extraido de la request que se haga en el formulario (`request.form.items()`).

Y para que ésto funcione hay que importar json y request de flask:

```
import json
```

```
from flask import Flask, render_template, redirect, url_for, request
```

Con ésto, al ejecutar el script, luego de llenado el formulario, va a la página de index.html pero si nos fijamos en herramientas del desarrollador → inspector, y de ahí se va a la parte de storage, se ve que guardó la request:

The screenshot illustrates the process of submitting a form and viewing the resulting cookie storage. At the top, a browser window shows a contact form with fields for Nombre, e-mail, and Comentario, all filled with placeholder text. A blue arrow points from this form to a second browser window below, which displays the text "Hola stranger". A red arrow points from the "Hola stranger" window down to a developer tools storage panel at the bottom. This panel is titled "Storage" and shows a table of cookies. A specific row for the domain "http://127.0.0.1:5000" is highlighted with a red oval, containing the key-value pair "nombre": "jenifer", "e-mail": "jenifer@mailinator.com".

Name	Domain	Path	Expires on	Last accessed on	Value	HttpOnly	sameSite
nombre": "jenifer", "e-mail": "jenifer@mailinator.com	127.0.0.1	/	Session	Fri, 27 Sep 2019 13...		false	Unset

Y si se hace una nueva request por ejemplo Nombre = Pablo, e-mail = [pablo@mailinator.com](mailto:pablo@mailinator.com) comentario = def, guarda una nueva cookie:

The screenshot shows a browser window with three tabs: 'Curso Completo Python 3', 'JSON - Wikipedia, la enciclopedia libre', and 'Saludo'. The active tab is 'Saludo' at '127.0.0.1:5000'. The page content is 'Hola stranger'. Below the browser is the Chrome DevTools Storage panel. Under 'Cookies', there are two entries for 'http://127.0.0.1:5000': one for 'jenifer' and one for 'pablo'. Both have empty values.

Name	Domain	Path	Expires on	Last accessed on	Value	HttpOnly	sameSite
{"nombre": "jenifer", "e-mail": "jenifer@malinator.com"}	127.0.0.1	/	Session	Fri, 27 Sep 2019 13...		false	Unset
{"nombre": "pablo", "e-mail": "pablo@malinator.com"}	127.0.0.1	/	Session	Fri, 27 Sep 2019 13...		false	Unset

Las cookies se guardan en el navegador del usuario por un tiempo determinado (a menos que sea una persistant coockie)

Por otro lado, si se selecciona la coockie, se pueen ver sus datos:

The screenshot shows the developer tools Storage panel with the 'Cookies' section selected. A specific cookie for 'http://127.0.0.1:5000' is selected, showing its details in the right-hand panel. The cookie has an empty value.

Name	Domain	Path	Expires on	Last accessed on	Value	HttpOnly	sameSite
{"nombre": "jenifer", "e-mail": "jenifer@malinator.com"}	127.0.0.1	/	Session	Fri, 27 Sep 2019 13...		false	Unset

```

name: {"nombre": "jenifer", "e-mail": "jenifer@malinator.com"}
host: "127.0.0.1"
path: "/"
expires: "Session"
creationTime: "Fri, 27 Sep 2019 13:27:03 GMT"
lastAccessed: "Fri, 27 Sep 2019 13:46:25 GMT"
value: ""
isDomain: false
isSecure: false
  
```

Si se ve la figura anterior, se ve que en campo value no contiene nada, lo cual es un error de programación ya que las coockies tienen un nombre y un valor pues un sitio puede tener más de una cookie, por ejemplo, una guarda la información de inicio de sesión mientras otra guarda lo que está en el carrito de compras, otra para ver qué visitó el usuario en su vez anterior para dar sugerencias, en fin, pueden ser muchas cookies asociadas a un mismo usuario, por lo que se requieren armar cookies con su llave y valor correspondiente.

Ésto se arregla así:

```
response.set_cookie("data", json.dumps(dict(request.form.items())))
```

Lo cual hace que la clave sea "data" y el valor sean los campos ingresados en el formulario:

The screenshot shows the developer tools Storage panel with the 'Cookies' section selected. Multiple cookies are listed, including one named 'data' which contains JSON data. A red arrow points to this 'data' cookie, and another red arrow points to the right pane where the cookie's value is shown as a JSON object.

Name	Domain	Path	Expires on	Last accessed on	Value	HttpOnly	sameSite
data	127.0.0.1	/	Session	Fri, 27 Sep 2019 14...	{"nombre": "jenifer", "e-mail": "jenifer@malinator.com"}	false	Unset
{"nombre": "jenifer", "e-mail": "jenifer@malinator.com"}	127.0.0.1	/	Session	Fri, 27 Sep 2019 14...		false	Unset
{"nombre": "luisa", "e-mail": "luisa@luisa.com"}	127.0.0.1	/	Session	Fri, 27 Sep 2019 14...		false	Unset
{"nombre": "pablo", "e-mail": "pablo@malinator.com"}	127.0.0.1	/	Session	Fri, 27 Sep 2019 14...		false	Unset

```

{"nombre": "jenifer", "e-mail": "jenifer@malinator.com"}
  
```

Por otra parte, para generar más de una cookie por sesión, se puede hacer:

```
@app.route("/enviado", methods=['GET', 'POST'])
def enviado():
    response = redirect(url_for("hello_world"))
    response.set_cookie("data",
json.dumps(dict(request.form.items())))
    response.set_cookie("sesion", "Información de inicio de sesión")
    return response
```

Y como resultado de llenar el formulario, obtener 2 cookies en vez de una:

The screenshot shows the 'Storage' tab in a browser's developer tools. It lists two cookies under the 'Cookies' section:

Name	Domain	Path	Expires on	Last accessed on	Value	HttpOnly	sameSite
data	127.0.0.1	/	Session	Fri, 27 Sep 2019 14...	{"nombre": "aldo"\054 \'e-mail\'...}	false	Unset
sesion	127.0.0.1	/	Session	Fri, 27 Sep 2019 14...	"Informaci\303\263n de inicio de ses...	false	Unset

### Cómo obtener información de las cookies

Para obtener los datos de las cookies en vez de la url, se va a editar la view de hello\_world como ejemplo:

```
data= json.loads(request.cookies.get("data"))
```

Ésto lo que hace es extraer los datos de la cookie llamada “data” y lo guarda en una variable llamada data. Por otra parte, se elimina a nombre del route:

```
@app.route("/")
@app.route("/<nombre>")
@app.route("/<int:edad>")
@app.route("/<nombre>/<int:edad>")
def hello_world(nombre="stranger", edad=None):
    try:
        data= json.loads(request.cookies.get("data"))
    except TypeError:
        if edad:
            context = {"name": nombre, "age": edad}
            return render_template("Hola_edad.html", **context)
        else:
            return render_template("index.html", name=nombre)

    else:
        nombre = data.get("nombre")
        edad = data.get("age")
        if data.get("age"):
            context = {"name": nombre, "age": edad}
            return render_template("Hola_edad.html", **context)
        else:
            return render_template("index.html", name=nombre)
```

Además se cambió el formulario de contacto para que acepte el campo edad:

```
% extends 'base_layout.html' %}
```

```
{% block title %}Contacto{% endblock %}

{% block content %}
<form action = {{ url_for('enviado') }} method="POST">
    <label for ="nombre">Nombre: </label><input type= "text"
name="nombre" value=""><br>
    <label for ="age">Edad: </label><input type= "int" name="age"
value=""><br>
    <label for ="e-mail">e-mail: </label><input type= "text" name="e-
mail" value=""><br>
    <label for ="comentario">Comentario: </label><input type= "text"
name="comentario" value=""><br>
    <input type= "submit" value="submit">
</form>
{% endblock %}
```

Pero, qué significa todo ésto:

Bueno, en app.py se modificó para que, si se ingresa algo en el campo edad, lo presenta en la página de inicio (`edad = data.get("age")`), por otra parte, también captura el nombre (`nombre = data.get("nombre")`) Pero todo eso sucede si existe una cookie llamada data(`try: data=json.loads(request.cookies.get("data"))`). Si ésto no existe, va a funcionar como antes, o sea:

- va a poner “Hello stranger” si no se pone ningun nombre en la url
- si se pone solo la edad en la URL va a devolver “Hello stranger. Tienes x años”
- si se pone el nombre y la edad, va a devolver “Hello <nombre>. Tienes x años”

Pero todo ésto si no hay una cookie llamada data, pero si ese archivo existe, va a priorizar los datos de la cookie.

127.0.0.1:5000/contacto

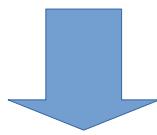
**Nombre:** luisa

**Edad:** 12

**e-mail:** luisa@luisa.com

**Comentario:** abc

submit



127.0.0.1:5000

**Hola luisa**

**Tienes 12 años**

Notas:

- 1) cada vez que se ejecuta el formulario, sobreescribe la cookie, y por lo tanto, actualiza lo que despliega
- 2) la variable context debeser declarada **después** de los gets, porque si nó, usa los valores que le venían por default

## Django

### Introducción

#### ¿Que es Django?

Según sus propios creadores, es un framework diseñado para perfeccionistas con deadlines, y que logran ésto haciendo que sea fácil de construir mejores sitios webs con menor cantidad de código. Es un Framework de Python de alto nivel cuya filosofía es promover un desarrollo rápido con un diseño limpio y pragmático de lo que se implemente con él.

Fué construido por programadores web experimentados de forma tal que Django se encarge de la mayor parte de las molestias del desarrollo web (conexiones a DB, storage de datos, manejo de memoria, etc.), cosa de que el programador solo deba ocuparse del desarrollo de su aplicación sin tener que “reinventar la rueda”.

Django se encarga de suministrar todo lo necesario para hacer una aplicación web sin que se deba recurrir a librerías de terceros.

Es tal vez el Framework más conocido y popular del mundo y es usado por sitios como Instagram, Pinterest y Mozilla.

## Características

A diferencia de otros frameworks como Flask, Django se enfoca en darte absolutamente todo lo que necesites para desarrollar tus aplicaciones web sin que tengas que recurrir a librerías de terceros. Con tener importado a Django ya da. En Django no se necesitan importar librerías extras como pweeee ni ningun otro ORMs para hacer árboles, conectarte a Dbs.

Debido a la edad que tiene ya Django que es algo vieja está ok, es muy madura y poderosa, también tiene muchas librerías de terceros que nos ayudan a solventar cualquier problema que Django por sí solo no pueda resolver.

Es gratis y OpenSource, lo cual hace que pueda ser ampliamente utilizado en cualquier entorno de desarrollo y si se quiere saber cómo están implementadas las cosas, es tan fácil como ir a su sitio web (<https://www.djangoproject.com/>) y está todo allí; y se puede usar para hacer proyectos tanto personales como comerciales.

## Vuelta por el sitio web de Django

Es un sitio web bien hecho donde incluso se cuenta con documentación para quienes quieran aprender a hacer páginas web con Django por su cuenta.

En la **página principal** dice las características de Django y permite que te suscribas a sus maillists para obtener ayuda usando Django o para contribuir, donde te puedes unir al google group correspondiente.

Por otra parte, en la página de **Download** (<https://www.djangoproject.com/download/>) dice cómo bajar e instalarlo, pero básicamente, con **pip install** ya queda. A efectos de éste curso lo que se hizo fue **sudo pip3 install django==1.10.5** dado que es la versión de Django que Aldominum usó para éste curso.

En la parte de **Documentación** (<https://docs.djangoproject.com/en/1.10/> y <https://docs.djangoproject.com/en/2.2/>) se pueden encontrar, tutoriales topic guides, reference guides y how-to receipts donde dice cómo solucionar problemas comunes durante el desarrollo habiendo respuestas para prácticamente todo. Además, si se quiere aprender cómo funciona Django, en la parte de Overview tiene la parte de ORM (<https://docs.djangoproject.com/en/2.2/intro/overview/> y <https://docs.djangoproject.com/en/1.10/intro/overview/>) nos da un pantallazo de cómo funciona y como se programa usando Django, cómo crear un modelo de datos, cómo desarrollar una app, como usar templates. Viene también con un tutorial de cómo hacer una página de encuestas y otros más complejos de cómo hacer un patch, aplicaciones reusables, entre otras cosas.

Finalmente, en la parte de **code** (<https://github.com/django/django>) se dice cómo llegar al código fuente de Django con su repositorio en GitHub. Es en ésta sección donde se puede explorar Django.

## **Proyecto en Django**

Usualmente, lo primero que se hace para aprender a usar un framework cualquiera es hacer un blog ya sea en Flask, Ruby on Rails o cualquier otro web Framework y Django no es la excepción.

Pero eso es lo usual, y para salir un poco del esquema, el docente decidió hacer una mini-réplica de un sitio de aprendizaje on-line como codeacademy y otros.

Todos esos sitios se caracterizan por tener cursos con sus nombres y descripciones, y una serie de lecciones asociados con sus nombres y descripciones que el usuario debe realizar para aprobarlos.

Desarrollar un sitio así, es el objetivo de las siguientes secciones.

Nota personal: el sitio va a ser medio manco porque no va a tener en cuenta usuarios ni autenticación ni se va a meter al detalle con mucha cosa, solo va a desarrollarse y desplegar una pequeña base de datos con los cursos y sus lecciones y no mucho más, con css y javascripts simples, pero va a servir lo suficiente para aprender.

## **Creación del proyecto**

Lo primero que se va a hacer es crear la carpeta donde se va a guardar el proyecto.

Luego, para crear un nuevo proyecto de Django escribir en el command line:

**`django-admin startproject <nombre_del_proyecto>`**

Ésto crea una carpeta nueva con el nombre del proyecto y dentro de esa carpeta hay un archivo llamado manage.py que es donde se van a poner los comandos para ejecutar las acciones de configuraciones, migraciones, y otras cosas; y por otra parte una carpeta con el nombre del proyecto que contiene

archivos .py que indican los settings globales del sitio(settings e \_\_int\_\_.py), las URLs de configuración global del sitio(urls.py)

Por el momento se va a dejar de lado un poco ésta carpeta y se va a trabajar priero con manage.py

### Inicialización del servidor

manage.py es el punto de entrada a la aplicación. Sirve para iniciar el servior, hacer migraciones y otras cosas.

Para inicializar el servidor, va a ser muy similar a lo hecho con flask:

```
python3 manage.py runserver <dirección_del_host>
```

En el caso de localhost:

```
python3 manage.py runserver 127.0.0.1:5000
```

Cuando ésto se ejecuta, el resultado es:



Esto nos dice que hubo 13 migraciones que no fueron aplicadas...

```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/django/educational$ python3 manage.py runserver 127.0.0.1:5000
Performing system checks...

System check identified no issues (0 silenced).

You have 13 unapplied migration(s). Your project may not work properly until you
apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.

September 28, 2019 - 00:46:56
Django version 1.10.5, using settings 'educational.settings'
Starting development server at http://127.0.0.1:5000/
Quit the server with CONTROL-C.
[28/Sep/2019 00:47:01] "GET / HTTP/1.1" 200 1767
Not Found: /favicon.ico
Not Found: /favicon.ico
[28/Sep/2019 00:47:01] "GET /favicon.ico HTTP/1.1" 404 1941
[28/Sep/2019 00:47:01] "GET /favicon.ico HTTP/1.1" 404 1941
```

## Migraciones iniciales

Una migración es una forma de mover el diseño en la DB de una forma a otra dado que el diseño de una DB puede cambiar en cualquier momento. Hay muchas razones por las cuales una tabla de DB puede cambiar, por ejemplo, agregar o quitar campos, modificar tipos de dato que almacena, la estructura que tiene, etc. Las migraciones son una forma de hacer estos cambios de forma segura (que no queden datos corruptos en la tabla nueva, por ejemplo).

Cada vez que se quiera hacer un cambio en la aplicación, se debe hacer una migración, si se agrega una tabla, se hace eso y luego una migración, si se quita una tabla, lo mismo.

Ésto ayuda a tener persistencia y poder mover las tablas y el diseño de la DB en la aplicación de un sistema a otro por ejemplo una tabla MySQL a PosgreSQL.

En cuanto a las bases de datos que se tienen en el proyecto...cuando se crea uno, Django crea una nueva DB en SQLite3 porque es una forma sencilla de "jugar" con la aplicación, lo cual es suficiente para el proyecto que se quiere hacer.

Por otra parte, para hacer las migraciones, hay que hacer exactamente lo que nos dice el warning:

```
python3 manage.py migrate
```

Lo cual nos da:

```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/django/educational$ python
3 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying sessions.0001_initial... OK
```

Y si se ejecuta el runserver, esta vez no da ningún error:

```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/django/educational$ python
3 manage.py runserver 127.0.0.1:5000
Performing system checks...

System check identified no issues (0 silenced).
September 28, 2019 - 01:16:27
Django version 1.10.5, using settings 'educational.settings'
Starting development server at http://127.0.0.1:5000/
Quit the server with CONTROL-C.
```

### Primer página: Hola Mundo

En ésta sección, se va a crear nuestra primera view ya que, al igual que Flask, el diseño web se basa en la creación de vistas, estructuras y controladores.

En el caso de Django se tienen templates, que son las encargadas de desplegar las cosas en las páginas web, y las funciones, cuya función es manejar las peticiones que le llegan a esos templates y son los encargadas de decidir que template desplegar, los cuales se llaman vistas o views.

Para crear la primera view, el mejor lugar es en directorio donde está el archivo manage.py del proyecto, en mi caso, la carpeta educational y el archivo de las views, se va a llamar views.py.

En ese archivo, se va a importar lo que tiene django para responder a requests:

```
from django.http import HttpResponse
```

Luego se define la view:

```
def hello_world(request):
    return HttpResponse("Hola mundo")
```

Todas las views aceptan un parámetro que es el request y siempre se debe obtener. Por otra parte, la respuesta que queremos es que despliegue la frase “Hola mundo”

Pero co ésto no es suficiente, pues hay que darle los datos para que sepa la route, o sea, donde desplegar ésto...

### URLs

Se utilizan para indicarle a Django donde desplegar que (al igual que Flask, necesita la URL route).

Eso se hace en el archivo urls.py. Desde ese archivo es que se le indica dese onde tiene que devolver cada view.:

```
from django.conf.urls import url
from django.contrib import admin
```

```
urlpatterns = [
```

```
    url(r'^admin/', admin.site.urls),  
]
```

Y dentro de la variable urlpatterns, se agrega la view

```
urlpatterns = [ url(r'^admin/', admin.site.urls), url(r'^$',  
views.hello_world)  
]
```

Como se puede ver, a diferencia de otros frameworks, Django acepta expresiones regulares (r'^admin/' , donde r es una cadena raw) .

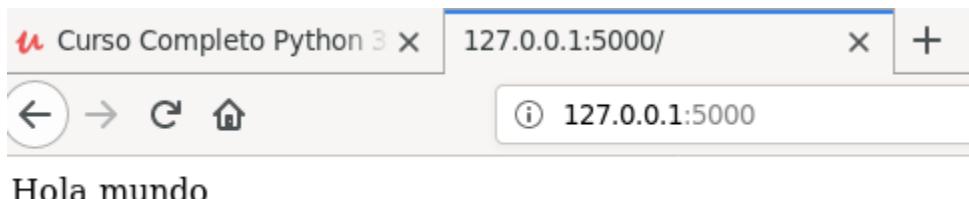
En nuestra nueva url, como es la dirección principal, se usa la expresión regular r'^\$' que significa cadena vacía, y la view que se quiere implementar es views.hello\_world. Siendo views el archivo .py y hello\_world la función que crea la view.

Por otra parte, antes de correr ésto, se debe importar en urls.py el view

```
from . import views
```

El punto significa que el archivo está en el mismo directorio del proyecto y que el archivo que queremos importar es views (notar que views.py actúa como una librería de Python)

Con ésto si se corre el programa y se va a localhost, da resultado:



Entonces, el archivo de urls.py nos queda que:

```
from django.conf.urls import url  
from django.contrib import admin  
  
from . import views  
  
urlpatterns = [  
    url(r'^admin/', admin.site.urls),  
    url(r'^$', views.hello_world),  
]
```

## Django Apps

Una app en Django es una pieza de funcionalidad encapsulada que sirve para realizar un propósito específico, por ejemplo una app que despliegue cursos, otra que maneje las preguntas sobre un curso, otra para manejar las reviews del curso, otra para el sistema de usuarios en un sitio web, en fin, son piezas encapsuladas donde cada app cumple con su función.

### **Armado de la aplicación de cursos y estudiantes**

Lo que se va a hacer es el esqueleto de un sitio web de aprendizaje dentro de 1 o más apps.

Lo primero que se va a desarrollar es la app de manejo de los cursos. En la consola:

```
python manage.py startapp <nomobre_de_la_aplicación>
```

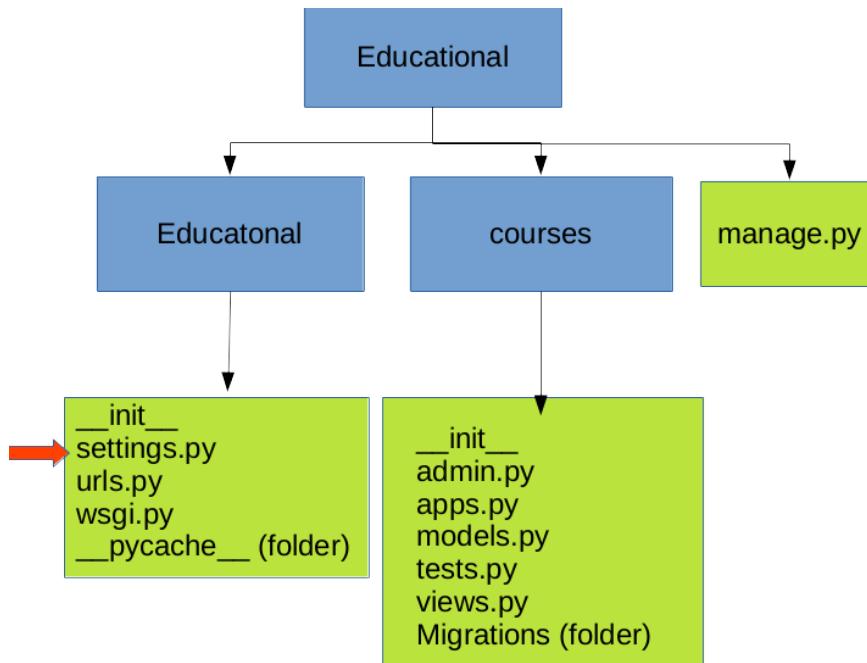
en mi caso

```
python3 manage.py startapp cursos
```

Nota: por convención, los nombres de las apps van con plural

Eso crea una nueva carpeta en el directorio raíz del proyecto donde va la app, con sus archivos correspondientes.

Luego, para informarle a Django y Python que se tiene una nueva app, lo que se hace es editar settings.py de la carpeta llamada educational en mi caso. En el siguiente dibujo se muestra la estructura de las carpetas y con una flecha roja, cuál es el archivo de settings.py que hay que editar:



En el archivo settings.py se encuentran todas las apps.

Lo que se va a hacer es agregar en la parte de **INSTALLED\_APPS**, la app llamada courses:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'courses',  
]
```

El resto se deja igual.

Si se observa detenidamente el archivo settings.py, hay una parte que dice:

```
# SECURITY WARNING: don't run with debug turned on in production!
```

```
DEBUG = True
```

Ésto hace referencia a que en el momento de desarrollo conviene que el modo debug esté activado pero no es así en la etapa de producción pues eso revelaría información que es confidencial.

También tiene definidas las DB que se van a usar, las validaciones de las contraseñas, y dice cosas respecto a la internacionalización:

```
LANGUAGE_CODE = 'en-us'  
TIME_ZONE = 'UTC'  
USE_I18N = True  
USE_L10N = True  
USE_TZ = True
```

El lenguaje se va a dejar igual, pero se va a cambiar el **TIME\_ZONE** que en el caso de Uruguay es:

```
TIME_ZONE = 'America/Montevideo'
```

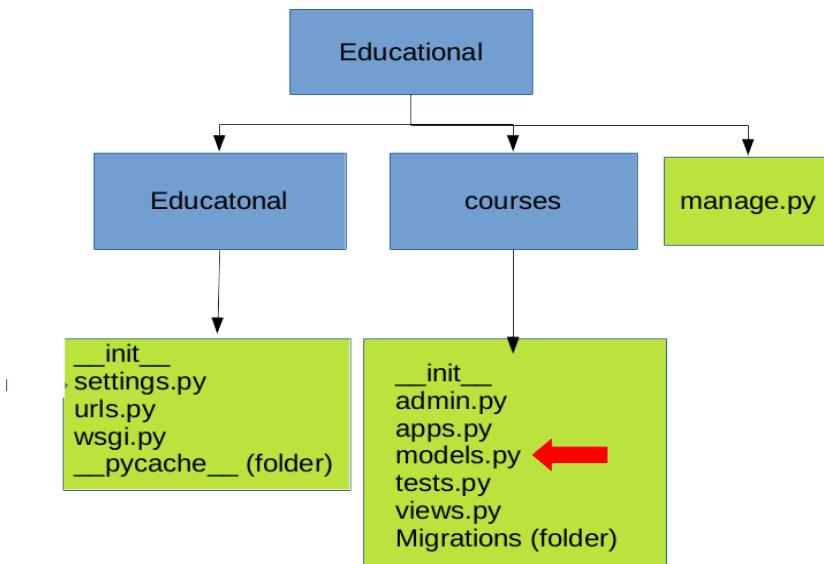
Con esto modificado, si se corre manage.py, todo sigue funcionando correctamente.

### Creación del primer modelo

Al igual que los ORMs como peewee, para tratar con DB se va a hacer un modelo, en el cual, cada clase es una tabla, los atributos son las columnas y cada instancia de los objetos son las filas.

En el caso de la DB del tutorial, se va a armar una clase que sean los cursos donde cada curso tenga su nombre, fecha de creación y una descripción.

Para ésto se va a abrir la carpeta de models que está dentro de courses:



Y se edita el archivo models.py:

```

class Course(models.Model):
    created_at = models.DateTimeField(auto_now_add=True)
    name = models.CharField(max_length=180)
    description = models.TextField()
  
```

Finalmente, para que haga la migración:

```
python3 manage.py makemigrations
```

```

jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/django/educational$ python
3 manage.py makemigrations
Migrations for 'courses':
  courses/migrations/0001_initial.py:
    - Create model Course
  
```

Y después `python3 manage.py migrate courses`

```

jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/django/educational$ python
3 manage.py migrate courses
Operations to perform:
  Apply all migrations: courses
Running migrations:
  Applying courses.0001_initial... OK
  
```

## Creación de instancias del modelo

Primero se va a activar la consola de django en python. En el caso del tutorial:

```
python manage.py shell
```

Con eso, es como si se hubiera entrado a una consola de Python común y corriente (que de hecho, lo es):

```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/django/educational$ python
3 manage.py shell
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> █
```

La diferencia es que ésta terminal cargó las funcionalidades de Django.

En ésta consola se va a importar la clase Course

```
>>>from courses.models import Course
```

Para saber todos los objetos del curso:

```
>>>Course.objects.all()
```

```
>>> from courses.models import Course
>>> Course.objects.all()
<QuerySet []>
>>> █
```

Ahí se ve que es un QuerySet que no contiene nada pues no se ha creado ninguna instancia de la clase Course.

Para crear un nuevo curso se hace de la misma forma que crear un objeto nuevo:

```
>>> curso_python.name = "Curso de Python"
>>> curso_python.description = "Curso de Python llevado a cabo por Aldominum"
>>>curso_python.save()
```

La fecha e creación no hace falta ponerla porque ya la autogeneró (`created_at = models.DateTimeField(auto_now_add=True)`)

```
>>> curso_python.name = "Curso de Python"
>>> curso_python.description = "Curso de Python llevado a cabo por Aldominum"
>>> curso_python.save()
>>> Course.objects.all()
<QuerySet []>
```

Como se ve en la imagen, QuerySet ya no está vacío, sino que tiene un campo de curso que es un objeto de tipo Course.

Otra forma de crear objetos fácilmente es:

```
>>> Course(name="Android", description="Curso de programación para Android").save()
```

Si se ejecuta de nuevo `Courses.objects.all()` muestra que hay 2 objetos creados del mismo tipo:

```
>>> Course.objects.all()
<QuerySet [<Course: Course object>, <Course: Course object>]>
>>> █
```

Y hay una tercera forma:

```
>>> Course.objects.create(name="Curso de HTML", description="Curso de programación en HTML")
>>> Course.objects.create(name="HTML", description="Curso de programación en HTML")
<Course: Course object>
>>> █
```

A diferencia de los métodos para crear instancias vistos anteriormente `.objects.create` devuelve el tipo de objeto que se creó, lo cual puede ser muy útil a la hora de programar.

### Despliegue de objetos (sobreescribiendo `__str__`)

Para definir cómo se van a imprimir los objetos creados, es sobreescribir el método `__str__` lo cual se hace desde el archivo `models.py`

```
class Course(models.Model):
    created_at = models.DateTimeField(auto_now_add=True)
    name = models.CharField(max_length=180)
    description = models.TextField()

    def __str__(self):
        return self.name
```

Notar que `__str__` se encuentra dentro de la clase `Course`.

Con ésto, al ejecutar de nuevo (porque no guarda automáticamente)

```
python3 manage.py shell
>>>from courses.models import Course
>>>Course.objects.all()
```

```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/django/educational$ python
3 manage.py shell
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from courses.models import Course
>>> Course.objects.all()
<QuerySet [<Course: Curso de Python>, <Course: Android>, <Course: HTML>]>
>>> █
```

Lo que hace Django es regresar un QuerySet y lo que imprime es en formato string lo que se haya definido para la función.

Bonus, si modificamos `__str__` de la siguiente manera:

```
def __str__(self):
    return "Nombre: {}, Curso: {}".format(self.name,
self.description)
```

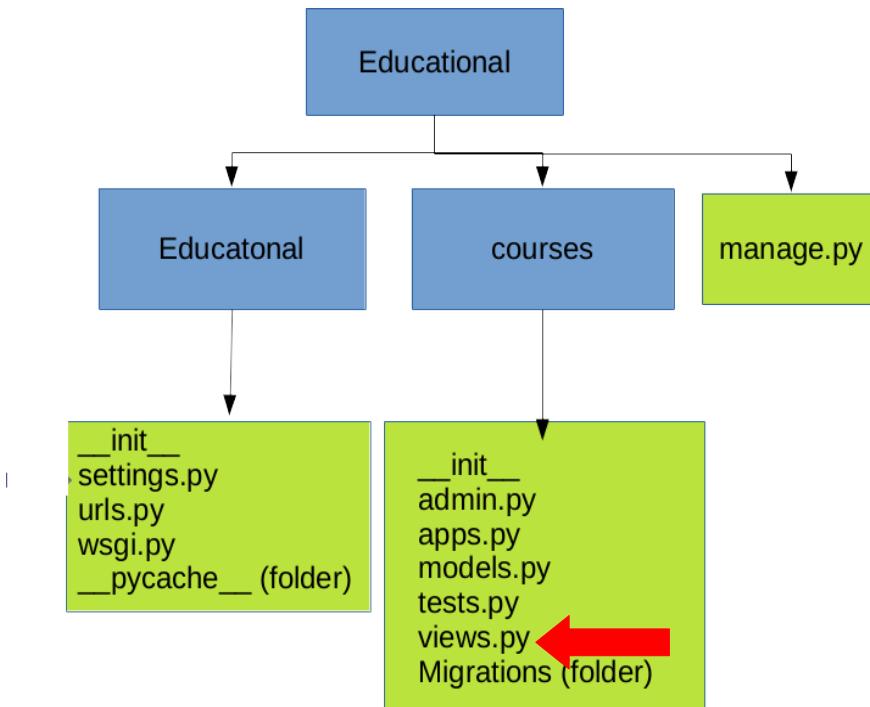
Lo que retorna es:

```
>>> from courses.models import Course
>>> Course.objects.all()
<QuerySet [<Course: Nombre: Curso de Python, Curso: Curso de Python llevado a ca-
bo por Aldominum>, <Course: Nombre: Android, Curso: Curso de programación para A-
ndroid>, <Course: Nombre: HTML, Curso: Curso de programación en HTML>]>
>>> █
```

### Creación de views (en el ejemplo, views de los cursos)

En nuestro caso, se va a crear una view que despliegue la lista de cursos.

Para eso, hay que editar el archivo de views



Como queremos que responda a un query del usuario, lo primero que se debe hacer es importar **HttpResponse** y la importación de la clase Course que está en models.py, además del que ya tenía éste archivo:

```

from django.http import HttpResponse
from django.shortcuts import render
from .models import Course

```

Luego se define una función que va a ser la vista:

```

def courses(request):
    courses = Course.objects.all()
    course_list = "\n".join(courses)
    return HttpResponse(course_list)

```

Nota:

Recordar que todas las vistas deben recibir el parámetro **request** porque es lo que recibe del usuario en su navegador

## Definición de las URLs

Para hacerlo, hay que definir un nuevo archivo en la carpeta de courses llamado urls.py y en el, poner lo siguiente:

```
from django.conf.urls import url
from django.contrib import admin
from . import views #notar que el import . es porque queremos
importar de ésta carpeta

urlpatterns = [url(r'^$', views.courses),
]
```

Y por otro lado, en el archivo urls.py de la carpeta educational, se le pide que incluya las urls definidas en urls.py de la carpeta courses:

```
from django.conf.urls import url, include
from django.contrib import admin

from . import views
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', views.hello_world),
    url(r'^courses/', include('courses.urls')),
]
```

Al hacer ésto, nos da un TypeError cuando se va a la url del curso:

TypeError at /courses/  
sequence item 0: expected str instance, Course found

Request Method: GET  
Request URL: http://127.0.0.1:5000/courses/  
Django Version: 1.10.5  
Exception Type: TypeError  
Exception Value: sequence item 0: expected str instance, Course found  
Exception Location: /home/jenifer/Documentos/Backup\_2019/qbit/Prog\_Py/django/educational/courses/views.py in courses, line 9  
Python Executable: /usr/bin/python3  
Python Version: 3.7.3  
Python Path: ['/home/jenifer/Documentos/Backup\_2019/qbit/Prog\_Py/django/educational',  
 '/usr/lib/python3.7',  
 '/usr/lib/python3.7/lib-dynload',  
 '/usr/local/lib/python3.7/dist-packages',  
 '/usr/local/lib/python3.7/dist-packages/Flask-1.1.1-py3.7.egg',  
 '/usr/local/lib/python3.7/dist-packages/Werkzeug-0.16.0-py3.7.egg',  
 '/usr/local/lib/python3.7/dist-packages/Jinja2-2.10.1-py3.7.egg',  
 '/usr/lib/python3/dist-packages']  
Server time: Sun, 29 Sep 2019 09:59:19 -0300

**Traceback** [Switch to copy-and-paste view](#)

```
/usr/local/lib/python3.7/dist-packages/django/core/handlers/exception.py in inner
39.     response = get_response(request)
...> Local vars
```

Ese error es porque lo que le estamos dando al join no es una cadena, sino que es un objeto:

```
/home/jenifer/Documentos/Backup_2019/qbit/Prog_Py/django/educational/courses/views.py in courses
9.     course_list = "\n".join(courses)
...> Local vars
response = wrapped_callback(request, *callback_args, **callback_kwargs)
File "/home/jenifer/Documentos/Backup_2019/qbit/Prog_Py/django/educational/courses/views.py", line 9, in courses
    course_list = "\n".join(courses)
TypeError: sequence item 0: expected str instance, Course found
```

Lo que indica el error es que espera que la instancia a devolver sea una cadena pero sin embargo se encontró con un objeto de la clase Course.

Para arreglar ésto:

### Como hacer un join con los objetos del modelo

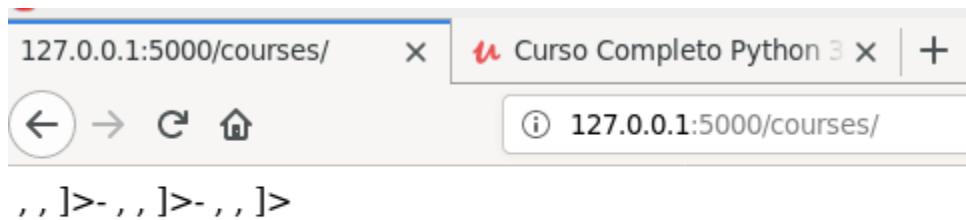
El hacer un join de una cadena con un QuerySet de cursos, no es algo que tenga sentido para Python.

Para hacerlo correctamente en views.py:

```
def courses(request):
    courses = Course.objects.all()
    course_list = "- ".join([str(course) for course in courses])
    return HttpResponse(course_list)
```

**NO ME ANDA... PREGUNTAR EN EL FORO**

**Me da ésto:**



### Solución parcial

```
def courses(request):
    course_list = list(Course.objects.all())
    return HttpResponse(course_list)
```

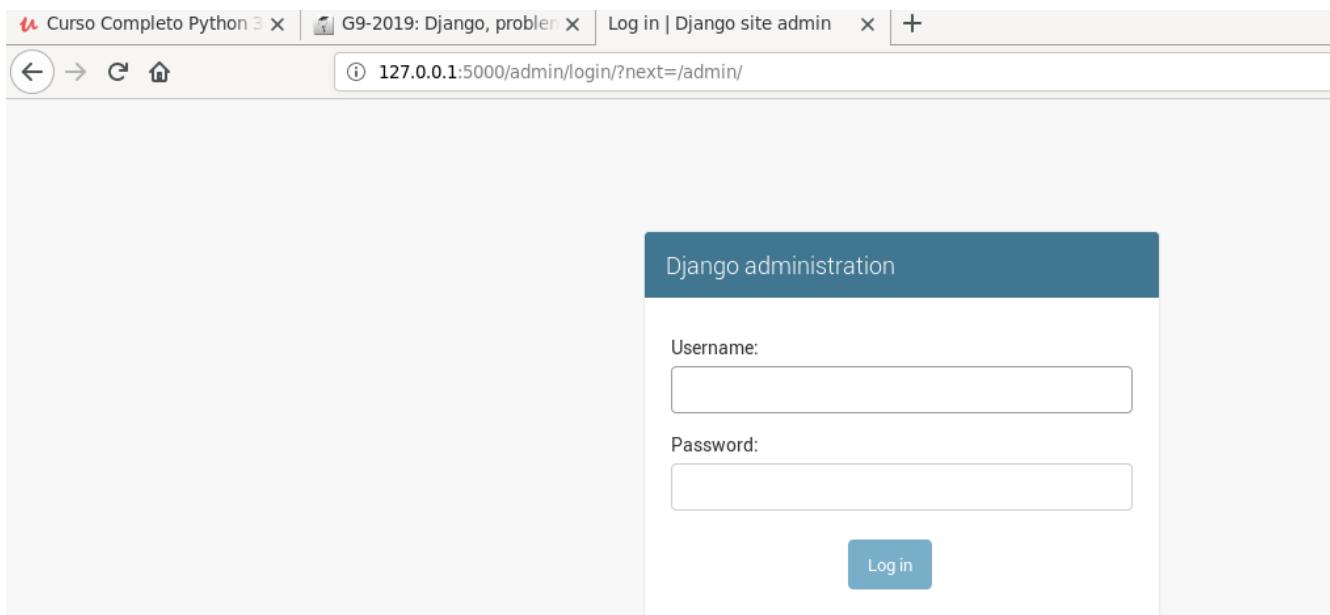
### Admin Panel

El amin panel, es el panél del administrador del sitio, que es donde se pueden hacer cosas grandes sin tener que estar tocando mucha cosa. A diferencia de otros lenguajes como php que se debe tener otro servidor instalado y hacer configuraciones para obtener un panel funcional, acá se puede hacer directamente.

Si se ve el archivo urls.py en la carpeta educational en nuestro caso, hay una linea que indica el sitio web del admin:

```
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', views.hello_world),
    url(r'^courses/', include('courses.urls')),
]
```

Si se va a la url de admin 127.0.0.1:5000 nos lleva a una página de login:

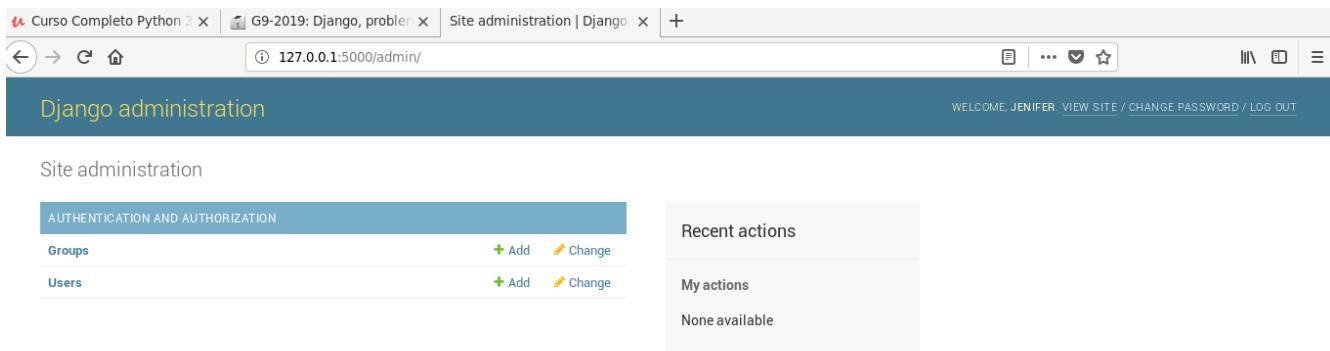


Para que el login funcione, hay que crear al usuario y darle los permisos correspondientes a través de `manage.py`, el cual tiene 2 comandos para ésto: **createsuperuser** y **changepassword**

Si se ejecuta **`python3 manage.py createsuperuser`**, pide todos los datos del superusuario en el prompt:

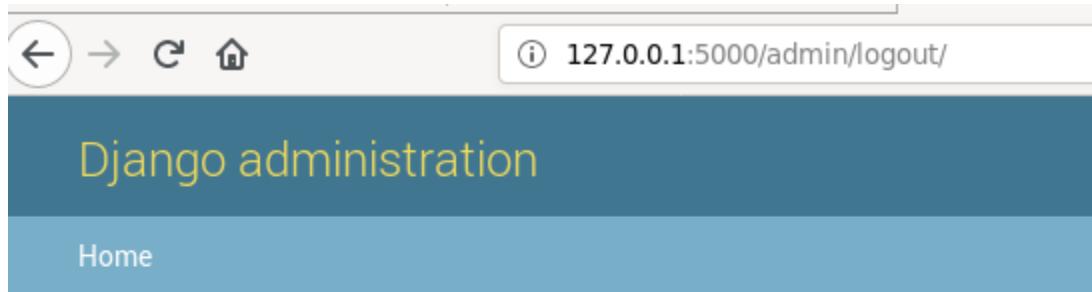
```
jenifer@azeroth:~/Documentos/Backup_2019/qbit/Prog_Py/django/educational$ python
3 manage.py createsuperuser
Username (leave blank to use 'jenifer'): jenifer
Email address: jenifer@mailinator.com
Password:
Password (again):
Superuser created successfully.
```

Una vez creado el usuario, si se accede a la página, la página de administración que presenta es:



En ésta página se pueden crear usuarios y grupos de usuarios nuevos. Ir a la página principal, cambiar la contraseña y salir.

Al salir:



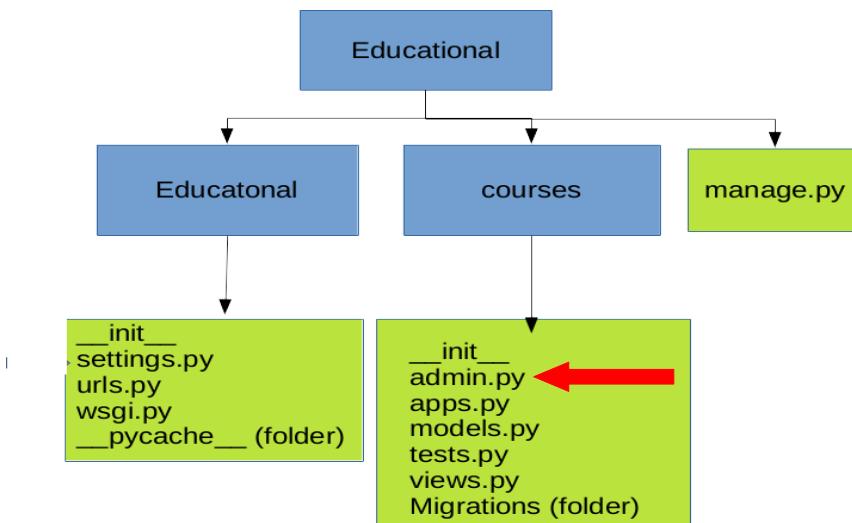
### Registro de Modelos en el portal de Admin

En el portal de admin estaban ya cargadas de usuarios de la página y de grupos de usuarios, pero en nuestro caso, interesa que también contenga la tabla de los cursos.

Los modelos no se cargan automáticamente porque puede ser que no les interese a los desarrolladores de Django que queden todos los modelos expuestos en la página por razones de seguridad.

Entonces, lo que se debe hacer “a mano” es subir los modelos que sí interesan que estén cargados en la página del admin y de los usuarios.

Para hacer eso, se debe editar el archivo admin.py:

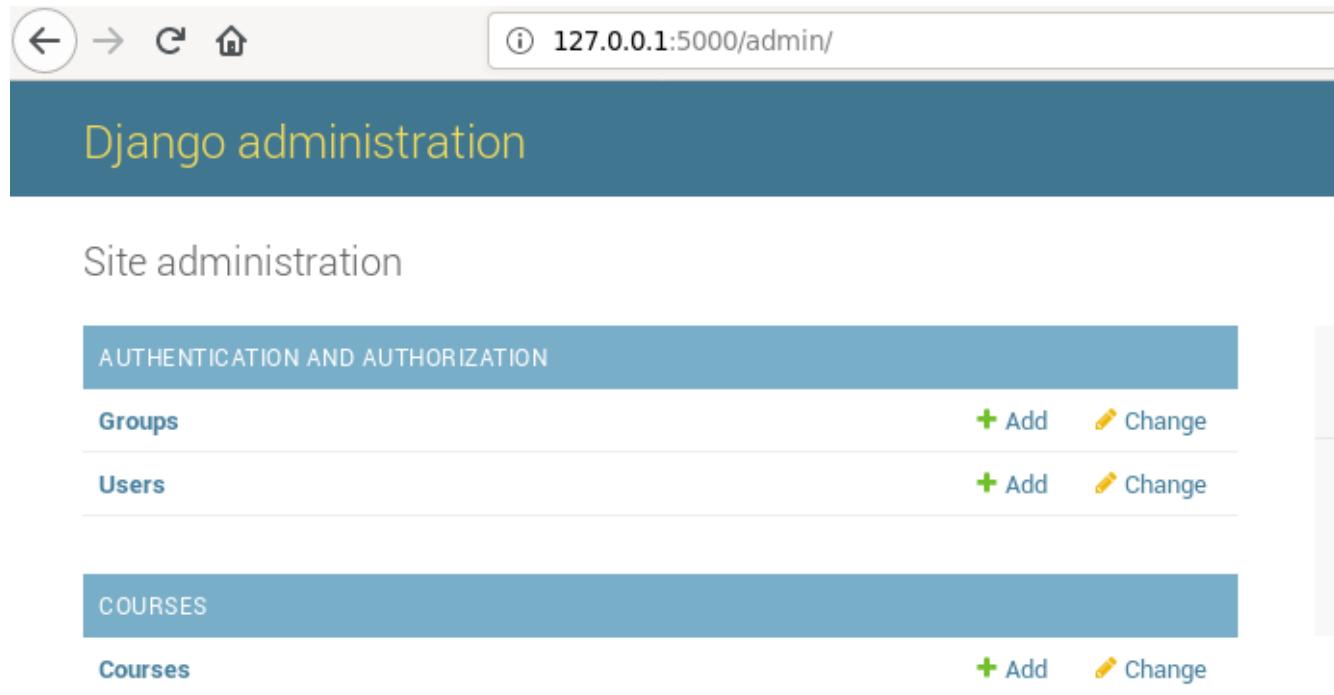


Dentro de ese archivo:

En ese archivo se incluye la siguiente linea:

```
from django.contrib import admin  
from .models import Course  
  
admin.site.register(Course)
```

Como resultado, despliega la tabla de cursos:



The screenshot shows the Django Admin interface at the URL `127.0.0.1:5000/admin/`. The top navigation bar includes links for back, forward, search, and home. The main title is "Django administration". Below it, the "Site administration" header is visible. The "AUTHENTICATION AND AUTHORIZATION" section contains links for "Groups" and "Users", each with "Add" and "Change" buttons. The "COURSES" section contains a single link for "Courses", also with "Add" and "Change" buttons.

Y si se va a la tabla, se despliegan los tres cursos que se definieron por consola y de hecho, da la opción de agregar uno nuevo y si se va a cualquiera de ellos, se puede editar tanto el nombre como la descripción:

Y si se va a la tabla, se despliegan los tres cursos que se definieron por consola y de hecho, da la opción de agregar uno nuevo y si se va a cualquiera de ellos, se puede editar tanto el nombre como la descripción:

Select course to change ADD COURSE +

Action: -----  0 of 3 selected

COURSE

Nombre: HTML, Curso: Curso de programación en HTML

Nombre: Android, Curso: Curso de programación para Android

Nombre: Curso de Python, Curso: Curso de Python llevado a cabo por Aldominum

3 courses

Change course | Django site admin - Mozilla Firefox  
127.0.0.1:5000/admin/courses/course/3/change/

Django administration

Welcome, JENIFER. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home > Courses > Courses > Nombre: HTML, Curso: Curso de programación en HTML

Change course

Name:

Description:

Delete Save and add another Save and continue editing SAVE

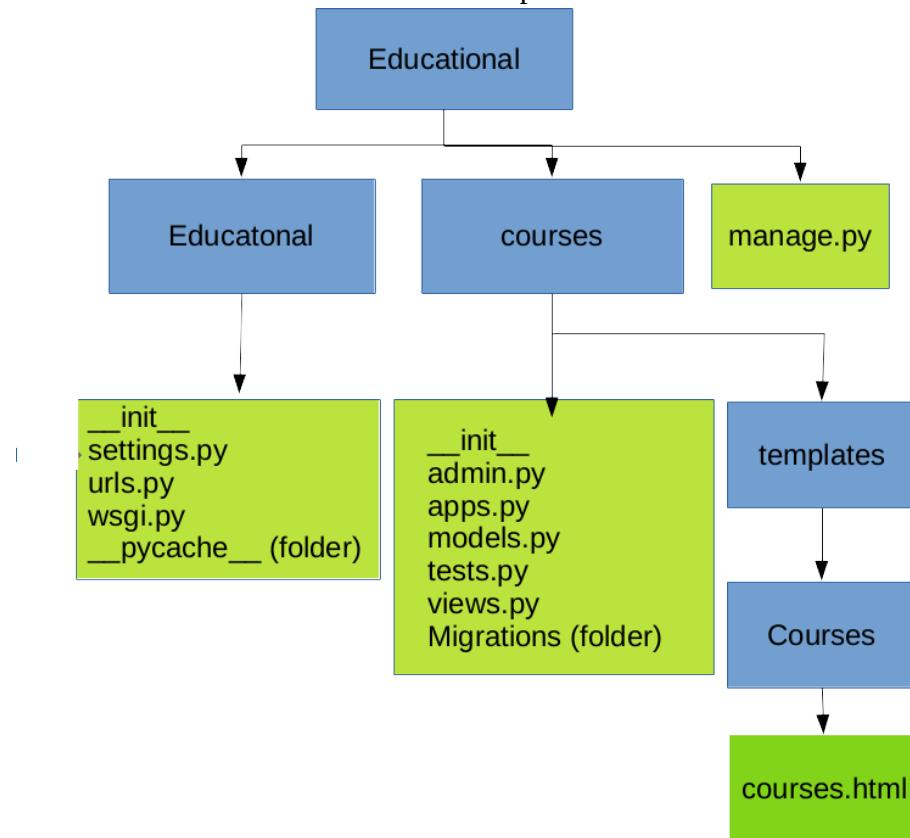
Por otra parte, se puede ver que el campo de fecha de creación no lo despliega. Ésto se debe a que es un campo que se genera automáticamente y Django supone que no hay razones para cambiarlo. Notar que ésta interfase gráfica permite hacer las cosas de manera más amena que en la consola.

## Templates

## Introducción

Para usar templates en Django, lo que hay que hacer es crear una carpeta de templates dentro del directorio donde está la aplicación, en nuestro caso, courses.

A continuación, lo que se va hacer es, dentro de la carpeta templates, crear una nueva carpeta que tenga el mismo nombre que la carpeta padre de templates, en nuestro caso "courses" y finalmente, en esa carpeta van los archivos con el código html. Ésto debe cumplirse siempre porque es la forma de que Django y Python se dan cuenta de donde sacar cada template.



En cuanto al nuestro caso, el template .html debe desplegar la lista de cursos. Para eso, se crea el archivo courses.html con la siguiente forma:

```
{% for course in courses %}
<h1>{{ course.name }}</h1>
<p>{{ course.description }}</p>
{% endfor %}
```

Ésto es porque en html, los ciclos se ejecutan usando "%<ciclo>%" "%end <ciclo>%"

Por otra parte, para desplegar variables se usa "{{ <variable> }}

Una vez hecho ésto, se va a cambiar la view:

```
from django.shortcuts import render
```

```
from .models import Course
```

```
def courses(request):
    courses = Course.objects.all()
    return render(request, "courses/courses.html",
{"courses":courses})
```

**render** acepta 3 parámetros: el request del usuario, la ubicación del html y un diccionario que está compuesto por 1 llave llamada “courses” y su valor es el DictQuery que es donde están guardados todos los objetos de la tabla creados.

Con éstos cambios, si se corre manage.py, en la página de courses:



# Curso de Python

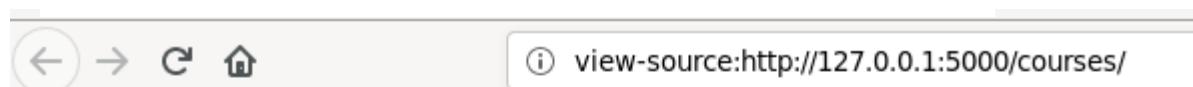
Curso de Python llevado a cabo por Aldominum

## Android

Curso de programación para Android

## HTML

Curso de programación en HTML



```
1 <h1>Curso de Python</h1>
2 <p>Curso de Python llevado a cabo por Aldominum</p>
3 <h1>Android</h1>
4 <p>Curso de programación para Android</p>
5 <h1>HTML</h1>
6 <p>Curso de programación en HTML</p>
7
8
9
10
11
```

### Template para la página principal

El template de página principal no tiene código HTML. Para crear un template para la página principal, se genera una carpeta nueva llamada templates en la carpeta raíz del proyecto, se crea el .html, y para decirle a Django que busque templates en nuestra nueva carpeta, hay que editar settings.py para agregar la carpeta nueva:

```

TEMPLATES = [
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': ["templates", ],
    'APP_DIRS': True,
    'OPTIONS': {
        'context_processors': [
            'django.template.context_processors.debug',
            'django.template.context_processors.request',
            'django.contrib.auth.context_processors.auth',
            'django.contrib.messages.context_processors.messages',
        ],
    },
},
]

```

La variable **TEMPLATES** es una lista que contiene un diccionario (y hasta puede tener más de uno) donde se le indica qué usar para hacer los renders ('**BACKEND**':

'**django.template.backends.django.DjangoTemplates**', lo cual se puede cambiar a jinja2 si se quiere), entre otras opciones. La que nosotros vamos a modificar es la variable **DIRS** pues antes era una lista vacía y se le agregó el directorio templates recién creado.

Por último se modifica a views.py de la carpeta educational:

```

from django.shortcuts import render

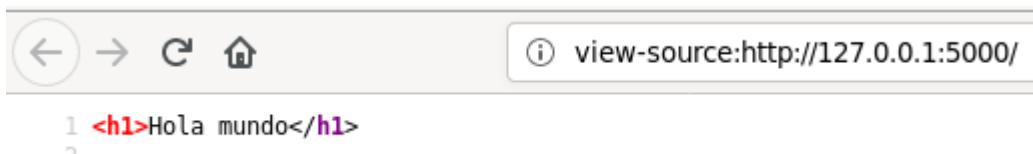
def hello_world(request):
    return render(request, "index.html")

```

Y si se ejecuta:



## Hola mundo



### Herencia de templates en Django

Ésta facilidad hace que se puedan heredar características entre templates hijas y padres y son útiles para no tener que escribir todo el código HTML de todas las páginas. Por ejemplo, que todas las páginas tengan el mismo footer o el mismo header.

Para crear el HTML básico se va armar uno llamado base\_layout.html dentro de la carpeta templates.

En ese html va todo el código que debe tener todo el sitio:

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>{% block title %}{% endblock %}</title>
  </head>
  <body>
    {% block body %} {% endblock %}
  </body>
  <footer>Created by eni</footer>
</html>
```

Para indicar de donde hereda, en la template hija se escribe {% extends "base\_layout.html" %}, por ejemplo, en inex.html que es la página de home:

```
{% extends "base_layout.html" %}
{% block title %}Home{% endblock %}
{% block body %}<h1>Hola mundo</h1> {% endblock %}
```

Y en la página que espliega los cursos:

```
{% extends "base_layout.html" %}
{% block title %}Cursos{% endblock %}
{% block body %}
  {% for course in courses %}
    <h1>{{ course.name }}</h1>
    <p>{{ course.description }}</p>
  {% endfor %}
{% endblock %}
```

Si se va a la url principal y a la que espliega los cursos:



## Hola mundo

Created by eni



## Curso de Python

Curso de Python llevado a cabo por Aldominum

### Android

Curso de programación para Android

### HTML

Curso de programación en HTML

Created by eni

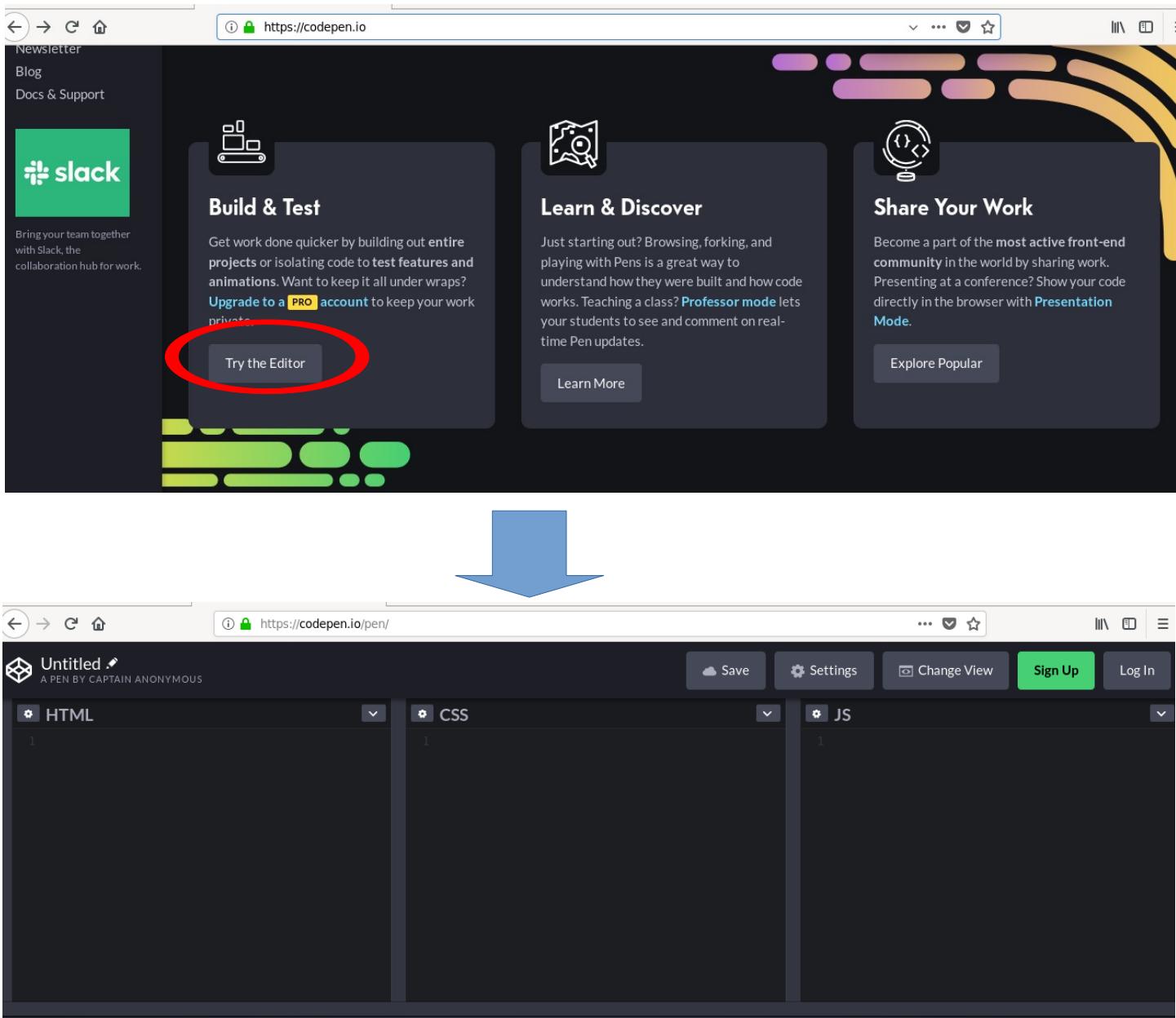
Se puede ver que lo que cambió fué el título de la página, y que en ambas incluyó el mismo footer porque es lo que está en el base\_layout.html

## Extra: HTML

### CodePen

CodePen es un sitio web que permite el desarrollo online de código HTML, CSS y JavaScript.

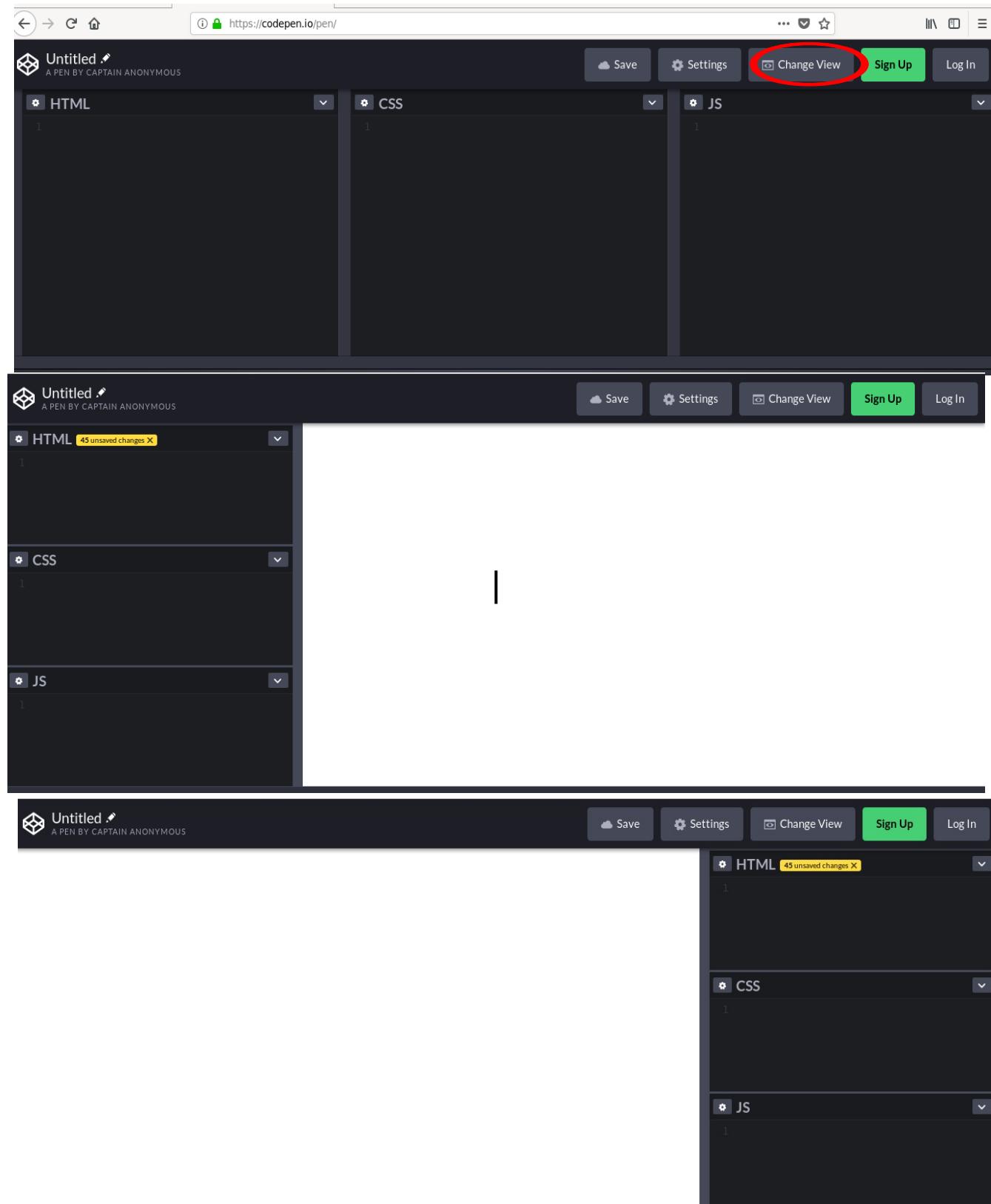
En su sitio <https://codepen.io/> y <https://codepen.io/pen/> es donde se va a desarrollar ésta parte del curso.



Como se puede ver, es un entorno interactivo para el desarrollo de código y se independiza del tipo de navegador porque las cosas pueden variar de un navegador a otro.

Por otra parte, los cambios se pueden ver en tiempo real.

Además, se puede cambiar la vista de forma que los cuadros de programación y el canvas aparezcan en otro orden:



## Acerca de HTML

Mucha gente cree que HTML es un lenguaje de programación, pero en realidad, ésto no es exactamente cierto. En realidad es un Markup Language, lo cual quiere decir que es un lenguaje de formato y lo único que hace es darle formato al texto (o audio, o video, o imagen) que tengamos que desplegar en el navegador.

HTML son las siglas en inglés de HyperText Markup Language, el cual es una convención standard para documentos diseñados para ser desplegados en un navegador de internet. Vendría a ser como un editor de texto como LibreOffice Writer o Microsoft Word en cuanto a que, lo único que hace es formatear cómo se ven las cosas en un navegador, por ejemplo si el texto es centrado o alineado de izquierda a derecha, si va subrayado con colores, en fin, ese tipo de cosas.

## Tags

Los tags son todas las características que están dentro de los signos “<” y “>” y es la forma que tiene HTML de diferenciar el formato que debe tener cada cosa, por ejemplo:

**<title>Home</title>** indica que el título de la página, que es lo que va sobre la pestaña del navegador, es la palabra “Home”.

A su vez, hay 2 tipos de tags:

- A nivel de bloque: empuja todo lo que viene después a una linea nueva
- A nivel de linea: Actúa solo en la linea y no da enter.

Por ejemplo el tag **<div>** que se usa para dar formato a nivel de bloque:

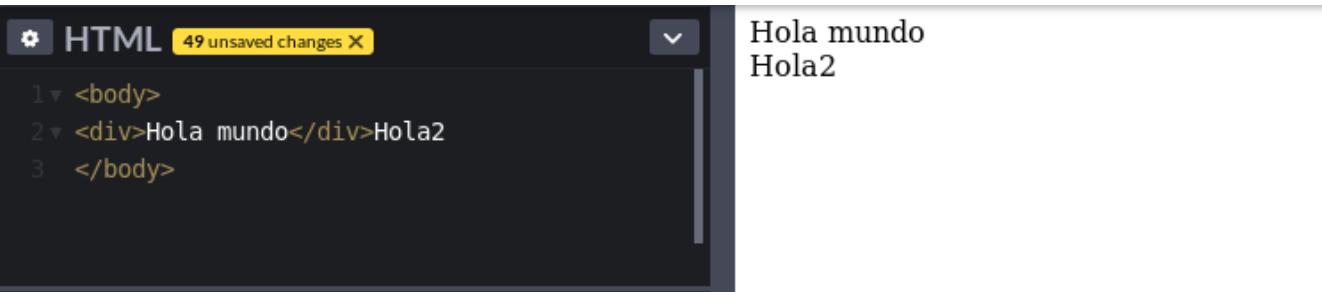
**<div>Hola mundo</div>**

Este tag es a nivel e bloque porque lo que hace es empujar a la siguiente linea todo lo que va después.

Por ejemplo:

```
<body>
<div>Hola mundo</div>Hola2
</body>
```

O sea **<div>** es como un enter. Si se escribe eso, el resultado es:



The screenshot shows a code editor window with the title "HTML 49 unsaved changes". The code in the editor is:

```
1 <body>
2 <div>Hola mundo</div>Hola2
3 </body>
```

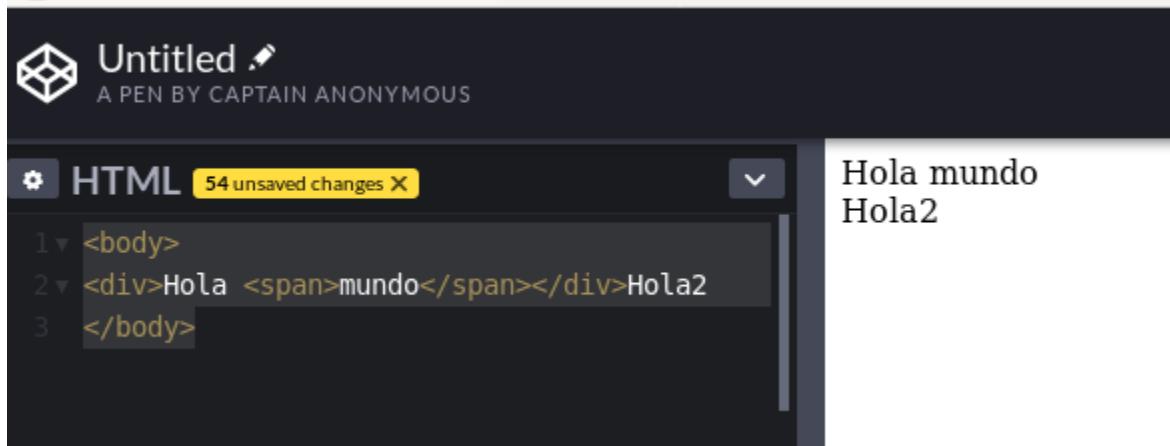
To the right of the editor, the rendered output is displayed:

Hola mundo  
Hola2

Por otra parte, el tag de **<span>** es a nivel de linea. Si se escribe:

```
<body>
<div>Hola <span>mundo</span></div>Hola2
</body>
```

y se ve en la pantalla impresa en el navegador, que no puso ningún enter entre Hola y mundo:



```
Untitled
A PEN BY CAPTAIN ANONYMOUS

HTML 54 unsaved changes X

1 <body>
2 <div>Hola <span>mundo</span></div>Hola2
3 </body>

Hola mundo
Hola2
```

Ademas <div> y <span> se diferencia en que **div** se usa para formatear grandes lineas de texto mientras que **span**, se usa para formatear pocas lineas.

Por otra parte <div> se usa para organizar la parte lógica de la página y era el tag más usado antes de HTML5. Ahora no se usa tanto porque en HTML5 hay tags específicos para hacer cosas que sólo con div, no se recomiendan hacer.

### **Tags de Párrafos y encabezados**

Éstos tags son útiles para organizar adecuadamente el texto y además es una forma fácil de organizar el HTML

#### **Párrafos**

En el caso de párrafo, el tag es la letra p:

<p>Texto</p>

Ejemplo:

```
<p>
  Alice was beginning to get very tired of sitting by her sister on
  the
  bank, and of having nothing to do. Once or twice she had peeped into
  the
  book her sister was reading, but it had no pictures or conversations
  in
  it, "and what is the use of a book," thought Alice, "without pictures
  or
  conversations?"</p>
<p>
  So she was considering in her own mind (as well as she could, for the
  day made her feel very sleepy and stupid), whether the pleasure of
  making a daisy-chain would be worth the trouble of getting up and
  picking the daisies, when suddenly a White Rabbit with pink eyes ran
  close by her.
</p>
<p>
```

There was nothing so very remarkable in that, nor did Alice think it so  
very much out of the way to hear the Rabbit say to itself, "Oh dear!  
Oh  
dear! I shall be too late!" </p>  
<p>But when the Rabbit actually took a watch  
out of its waistcoat-pocket and looked at it and then hurried on,  
Alice  
started to her feet, for it flashed across her mind that she had  
never  
before seen a rabbit with either a waistcoat-pocket, or a watch to  
take  
out of it, and, burning with curiosity, she ran across the field  
after  
it and was just in time to see it pop down a large rabbit-hole, under  
the hedge.</p>  
<p>In another moment, down went Alice after it!  
</p>

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do. Once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice, "without pictures or conversations?"

So she was considering in her own mind (as well as she could, for the day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.

There was nothing so very remarkable in that, nor did Alice think it so very much out of the way to hear the Rabbit say to itself, "Oh dear! Oh dear! I shall be too late!"

But when the Rabbit actually took a watch out of its waistcoat-pocket and looked at it and then hurried on, Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and, burning with curiosity, she ran across the field after it and was just in time to see it pop down a large rabbit-hole, under the hedge.

In another moment, down went Alice after it!

A diferencia de div, p es para organizar párrafos.

Tambien es tag e tipo bloque, por lo tanto, empuja todo lo que viene después del tag a una nueva linea.

## Encabezado

Para los encabezados, se usa el tag **h**.

<h1>Texto</h1>

El número indica que nivel de encabezado se quiere.

Ejemplo:

<h1>Alice in wonderland</h1>

<h2>by Louis Carol</h2>

da como resultado:

# Alice in wonderland

## by Louis Carol

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do. Once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice, "without pictures or conversations?"

So she was considering in her own mind (as well as she could, for the day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.

There was nothing so very remarkable in that, nor did Alice think it so very much out of the way to hear the Rabbit say to itself, "Oh dear! Oh dear! I shall be too late!"

But when the Rabbit actually took a watch out of its waistcoat-pocket and looked at it and then hurried on, Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and, burning with curiosity, she ran across the field after it and was just in time to see it pop down a large rabbit-hole, under the hedge.

In another moment, down went Alice after it!

En total hay 6 niveles de encabezado.

Para

```
<h1>Alice in wonderland: Encabezado Nivel 1</h1>
<h2>by Louis Carol -- Encabezado Nivel 2</h2>
<h3>Encabezado Nivel 3</h3>
<h4>Encabezado Nivel 4</h4>
<h5>Encabezado Nivel 5</h5>
<h6>Encabezado Nivel 6</h6>
```

Lo que muestra la página:

# Alice in wonderland: Encabezado Nivel 1

## by Louis Carol -- Encabezado Nivel 2

**Encabezado Nivel 3**

**Encabezado Nivel 4**

**Encabezado Nivel 5**

**Encabezado Nivel 6**

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do. Once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice, "without pictures or conversations?"

So she was considering in her own mind (as well as she could, for the day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.

There was nothing so very remarkable in that, nor did Alice think it so very much out of the way to hear the Rabbit say to itself "Oh dear! Oh dear! I shall be too late!"

La apariencia de los encabezados se pueden cambiar con código CSS pero los navegadores pueden interpretar los heads y ponerlos con distinto formato que el que usan en los párrafos.

### Tags Strong y Emfasis

#### Énfasis

Es un tag que se usa generalmente en los párrafos para poner cosas en itálica.

Su uso:

```
<em>Texto</em>
```

Ejemplo:

```
<h1>Alice in wonderland: Encabezado Nivel 1</h1>
<h2><em>by Louis Carol </em>-- Encabezado Nivel 2</h2>
```

Ésto da como resultado:

**Alice in wonderland: Encabezado Nivel 1**  
***by Louis Carol -- Encabezado Nivel 2***

Si bien, se puede con CSS poner cosas en itálica, el tag `<em>` es para que señale sólo la parte de texto que se quiere convertir.

Por otra parte, hay que destacar que es un tag de tipo linea.

#### Strong

Su uso:

```
<strong>texto</strong>
```

Y lo que hace es poner cosas en negrita:

Por ejemplo:

```
<p>
    <strong>Alice</strong> was beginning to get very tired of sitting
    by her sister on the
    bank, and of having nothing to do. Once or twice she had peeped into
    the
    book her sister was reading, but it had no pictures or conversations
    in
    it, "and what is the use of a book," thought Alice, "without pictures
    or
    conversations?"</p>
```

Nos da:

**Alice** was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do. Once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice, "without pictures or conversations?"

Tanto **strong** como **em** sirven para estacar cosas dentro de un texto determinado.

## Quotes

Hay 2 tipos de tag de quotes:

- Block Quote: se comporta como el tag de parrafo.en cuanto a que se utiliza para citar gran cantidad de texto. Se comporta como **<p>** y como **<div>** en cuanto a que hace que todo lo que le sigue después, pase a una nueva linea. Su función es poner sangría antes del texto.
- q: Es un tag de linea que sirve para poner comillas y citar frases cortas

### **Block quote**

En el caso de blockquote:

```
<blockquote>texto</blockquote>
```

Ejemplo:

```
<blockquote>So she was considering in her own mind (as well as she could, for the day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.  
</blockquote>
```

Nos da:

## **Alice in wonderland**

**by Louis Carol**

**Alice** was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do. Once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice, "without pictures or conversations?"

So she was considering in her own mind (as well as she could, for the day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.

There was nothing so very remarkable in that, nor did Alice think it so very much out of the way to hear the Rabbit say to itself, "Oh dear! Oh dear! I shall be too late!"

But when the Rabbit actually took a watch out of its waistcoat-pocket and looked at it and then hurried on, Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and, burning with curiosity, she ran across the field after it and was just in time to see it pop down a large rabbit-hole, under the hedge.

In another moment, down went Alice after it!

Este tag indica a la persona que está leyendo, que ese parrafo es una cita de lo que dijo alguien más. Por otra parte, éste tag tiene un atributo, donde se le puede colocar a quién pertenece la cita.

En general, los atributos de un tag, son datos adicionales que se le pueden poner a un tag para que adquiera funcionalidades extra.

En el caso de blockquote:

```
<blockquote cite="(nombre_del_atributo)">Texto</blockquote>
```

En el caso de el atributo particular de cite, no se ve reflejado en el texto desplegado en la página, pero si se desea acceder a el a traves de código CSS por ejemplo, se puede hacer.

## q

El tag de `<q>texto</q>` se emplea para citar frases cortas

Ejemplo:

`<p>`

`There was nothing so very remarkable in that, nor did Alice think it  
so  
very much out of the way to hear the Rabbit say to itself, <q>Oh  
dear! Oh  
dear! I shall be too late!</q> said the rabbit </p>`

Nos da:

There was nothing so very remarkable in that, nor did Alice think it so very much out of the way to hear the Rabbit say to itself, "Oh dear! Oh dear! I shall be too late!" said the rabbit

Como se puede ver, el tag no puso una nueva linea y el navegador solo puso las comillas

Por otra parte, el tag `<q>` también tiene el atributo cite:

`<q cite="(cita)">Texto</q>`

## Tags de formato previo y código

### Tag Pre

Si se usa el tag `<p>` y se copia y pega texto de algún otro lado, el tag no respeta el formato del texto previo, sino que pone todo seguido en un mismo parrafo.

Para evitar ésto, se usa el tag `<pre>`

Ejemplo:

The screenshot shows a code editor interface with two panes. The left pane is labeled 'HTML' and shows the following code:

```
1 ▾ <p>Hola, ¿como estás?  
2 Bien y tu</p>  
3 ▾ <pre>Hola, ¿como estás?  
4 Bien y tu</pre>
```

The right pane shows the resulting HTML output:

Hola, ¿como estás? Bien y tu

Hola, ¿como estás?  
Bien y tu

The output shows that the first paragraph (`<p>`) does not preserve the new line between the two sentences, while the second (`<pre>`) preserves it.

En la figura se ve que al usar el tag `<pre>` sí respeta el formato que se le dió al texto, es decir, el enter entre una frase y otra.

### Tag de código

Se utiliza cuando se quiere exponer un código en algún lenguaje de programación

Ejemplo:

`<p>La siguiente función imprime en Python la frase "Hola mundo"</p>  
<code>print ("Hola mundo") </code>`

Nos da:

La siguiente función imprime en Python la frase "Hola mundo"

```
print("Hola mundo")
```

Como se puede ver en la imagen, los estilos de párrafo son distintos. Por otra parte, hay que notar que, a los diferentes tags, se les puede editar el formato usando CSS.

Otra cosa a notar es que éste tag no significa que esa linea de código se vayaa ejecutar, sino que sirve sólo a propósito de formatear la salida.

Por último el tag <code> es un tag de linea, o sea, no imprime enteros al final.

## Listas

Hay veces que se requiere impriir en el navegador cosas en formato de lista. En HTML hay varios tipos de listas

### **Listas desordenadas**

Para éste tipo de datos, el tag que se usa es <ul> para englobar la lista, y el tag <li> para indicar cada elemento de la lista.

Ejemplo:

```
<ul>
  <li>pan</li>
  <li>azucar</li>
  <li>leche</li>
  <li>huevos</li>
</ul>
```

Da como resultado:

- pan
- azucar
- leche
- huevos

Nota. Indentación en HTML:

En realidad, HTML es independiente de la indentación, pero es una buena práctica hacerla pues así, el código queda legible.

### **Lista ordenada**

Para éste tipo de listas, se usa el tag <ol>

Ejemplo:

```
<ol>
```

```
<li>pan</li>
<li>azucar</li>
<li>leche</li>
<li>huevos</li>
</ol>
```

Da:

- 
1. pan
  2. azucar
  3. leche
  4. huevos

En éste caso, los elementos tienen un número asociado.

### Listas dentro de listas

Otra cosa que se puede tener es sublistas dentro de una lista, por ejemplo:

```
<ol>
  <li>huevos</li>
  <li>frutas
    <ul>
      <li>manzanas</li>
      <li>bananas</li>
      <li>duraznos</li>
    </ul>
  </li>
  <li>pan</li>
  <li>azucar</li>
  <li>leche</li>
</ol>
```

Da como resultado que las frutas en el ejemplo, caigan dentro de una sublista:

- 
1. huevos
  2. frutas
    - manzanas
    - bananas
    - duraznos
  3. pan
  4. azucar
  5. leche

### Definition list

Éste tipo de listas, se diferencian de las anteriores en cuanto a que se usan para definir elementos particulares de una lista.

Su tag es **<dl>** y los tags internos **<dt>** para que sea el título de la lista y **<dd>** para los elementos de la lista

Por ejemplo:

```
<dl>
  <dt>Libros:</dt>
  <dd>Alice in Wonderland</dd>
  <dd>Coraline</dd>
  <dd>Lord of the Rings</dd>
</dl>
```

Da como resultado:

---

Libros:  
Alice in Wonderland  
Coraline  
Lord of the Rings

### Tags para links

Los links son importantes pues permiten compartir cosas con los usuarios de un navegador que visita una página web. Dichos links pueden ser a una galería de imágenes, links a videos, links a artículos que estén o no en la misma página, en fin, su uso es ampliamente extendido.

Para poner links en una página se usa el tag **<a>** (del inglés anchor).

Ejemplo:

```
<a href = "www.google.com">Este es un enlace a google</a>
```

Imprime:

Este es un enlace a google

O sea, le da un formato a la frase que le indica al usuario que lo que hay englobado en esa frase es un link a google.

Un anchor se compone de 2 partes: el tag en sí que es donde se pone la dirección del link (**href**), y la otra es el texto asociado a ese link.

Por otra parte, si se quiere que los links naveguen dentro de la misma página, se pueden usar los anchors:

```
<a id="inicio"><h1>Alice in Wonderland</h1></a>
<p>
  Alice was beginning to get very tired of sitting by her sister on
  the
  bank, and of having nothing to do. Once or twice she had peeped into
  the
```

book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice, "without pictures or conversations?"</p><p>So she was considering in her own mind (as well as she could, for the day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.</p><p>There was nothing so very remarkable in that, nor did Alice think it so very much out of the way to hear the Rabbit say to itself, "Oh dear! Oh dear! I shall be too late!"</p><p>But when the Rabbit actually took a watch out of its waistcoat-pocket and looked at it and then hurried on, Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and, burning with curiosity, she ran across the field after it and was just in time to see it pop down a large rabbit-hole, under the hedge.</p><p>In another moment, down went Alice after it!</p><a href="#inicio">Este link vuelve al título</a>

Da:

## Alice in Wonderland

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do. Once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice, "without pictures or conversations?"

So she was considering in her own mind (as well as she could, for the day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.

There was nothing so very remarkable in that, nor did Alice think it so very much out of the way to hear the Rabbit say to itself, "Oh dear! Oh dear! I shall be too late!"

But when the Rabbit actually took a watch out of its waistcoat-pocket and looked at it and then hurried on, Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and, burning with curiosity, she ran across the field after it and was just in time to see it pop down a large rabbit-hole, under the hedge.

In another moment, down went Alice after it!

[Este link vuelve al título](#)

`<a id="inicio">Texto</a>` lo que hace es crear un anchor con un id llamado inicio, para después, más adelante en la página, si se quiere un link a esa sección, se puede usar **href** (`<a href="#inicio> texto </a>`)

### Imágenes en HTML

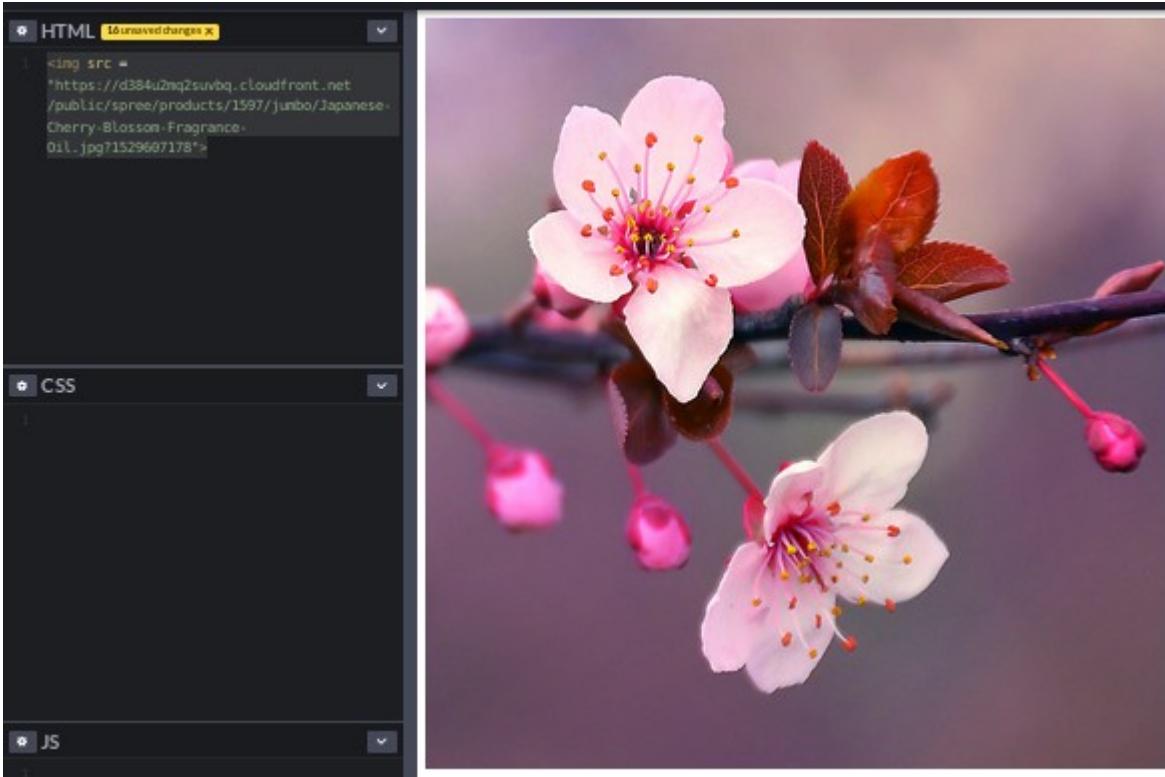
Para incluir imágenes en HTML se usa el tag `<img>`

A diferencia de los tags vistos anteriormente, `<img>` no tiene ninguna parte de abrir y cerrar.

Lo que si tiene es un atributo llamado source que es el origen de la imagen, ejemplo:

```
<img src =  
"https://d384u2mq2suvbq.cloudfront.net/public/spree/products/1597/  
jumbo/Japanese-Cherry-Blossom-Fragrance-Oil.jpg?1529607178">
```

Nos da:



Pero además hay que poner un texto alterno con el atributo alt:

```
<img src =  
"https://d384u2mq2suvbq.cloudfront.net/public/spree/products/1597/  
jumbo/Japanese-Cherry-Blossom-Fragrance-Oil.jpg?1529607178" alt =  
"sakura flower">
```

Hay muchas razones para poner un texto alternativo:

- Que despliegue ese texto mientras cargue la página.
- Si un usuario busca imágenes en google, google lo que hace es buscar en el texto alterno.
- Si el usuario usa un screen reader para que redacte el contenido de la página.

Otra cosa: no necesariamente tiene que ser una imagen de una url externa, si la imagen es local, se puede poner la ruta a la imagen.

## Tablas en HTML

Las tablas en HTML funcionan de manera similar a las tablas de excel en el sentido de que están organizadas por columnas y filas, pero no deben ser usadas para organizar todo el contenido de una página web.

El tag de tabla es <table> y para cada fila y columna es <tr> (table row) y <td> respectivamente: Por otra parte, se le pone bordes a la tabla con el tag “border” seguido del tamaño en pixels del borde.

Ejemplo:

```
<table border="1px">
  <tr>
    <td>a</td>
    <td>b</td>
    <td>c</td>
  </tr>
  <tr>
    <td>d</td>
    <td>e</td>
    <td>f</td>
  </tr>
</table>
```

a	b	c
d	e	f

Las tablas usualmente necesitan encabezados, los cuales se hace con el tag <th>:

```
<table border="1px">
  <th>Encabezado 1</th>
  <th>Encabezado 2</th>
  <th>Encabezado 3</th>
  <tr>
    <td>a</td>
    <td>b</td>
    <td>c</td>
  </tr>
  <tr>
    <td>d</td>
    <td>e</td>
    <td>f</td>
  </tr>
</table>
```

Encabezado 1	Encabezado 2	Encabezado 3
a	b	c
d	e	f

## thead y colspan

Para propósitos de organización, existe el tag llamado <thead> que se encarga de indicar que lo que engloba es el header de la tabla.

```
<thead>
  <th>Encabezado 1</th>
  <th>Encabezado 2</th>
  <th>Encabezado 3</th>
```

```
</thead>
```

Por otra parte, el tag **<foot>** es para dar formato al pie de página y el atributo **colspan** es para indicarle a la tabla que el pie de página ocupe las 3 celdas:

```
<table border="1px">
  <thead>
    <th>Encabezado 1</th>
    <th>Encabezado 2</th>
    <th>Encabezado 3</th>
  </thead>
  <tr>
    <td>a</td>
    <td>b</td>
    <td>c</td>
  </tr>
  <tr>
    <td>d</td>
    <td>e</td>
    <td>f</td>
  </tr>
  <tfoot>
    <tr>
      <td colspan = 3 >Pie de pagina</td>
    </tr>
  </tfoot>
</table>
```

Por otro lado, existe otro atributo llamado **align**, que permite decir cómo se va alinear el texto:

```
<tfoot>
<tr>
  <td colspan = 3 align = "center" >Pie de pagina</td>
</tr>
</tfoot>
```

Encabezado 1	Encabezado 2	Encabezado 3
a	b	c
d	e	f
Pie de pagina		

Si se pone **align = "left"** alinea hacia la izquierda y **align = "right"**, hacia la derecha.

Por último, existe otro atributo llamado **cellpadding** del tag **<table>** que lo que hace es determinar el tamaño de las celdas pues lo que cambia es el espacio interno de las celdas.

```
<table border="10px" cellpadding = "10px">
```

Encabezado 1	Encabezado 2	Encabezado 3
a	b	c
d	e	f
Pie de pagina		

```
<table border="10px" cellpadding = "5px">
```

Encabezado 1	Encabezado 2	Encabezado 3
a	b	c
d	e	f
Pie de pagina		

¿Se nota la diferencia?

Para cambiar el espacio entre celdas, el atributo es **cellspacing**

```
<table border="10px" cellpadding = "5px" cellspacing = "15px">
```

Encabezado 1	Encabezado 2	Encabezado 3
a	b	c
d	e	f
Pie de pagina		

Por otra parte el atributo para alinear verticalmente es **valign** (cuyos valores pueden ser top, bottom o center) y el tag de salto de linea es <br>

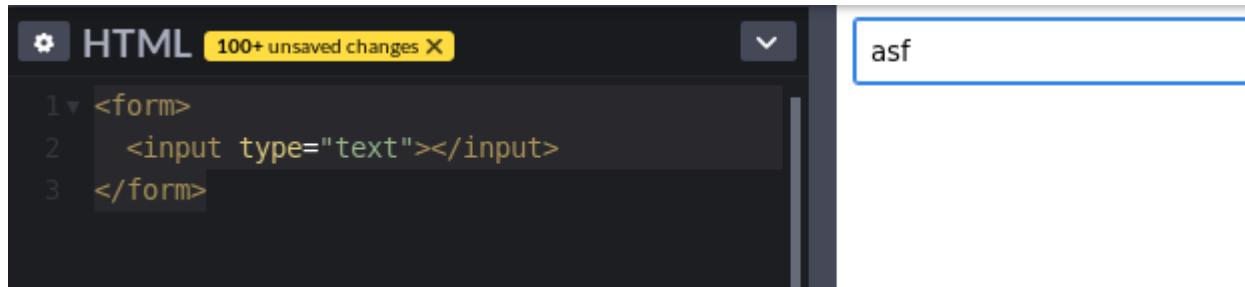
### Forms e input text

Los formularios se cargan con texto para darle información a una página.

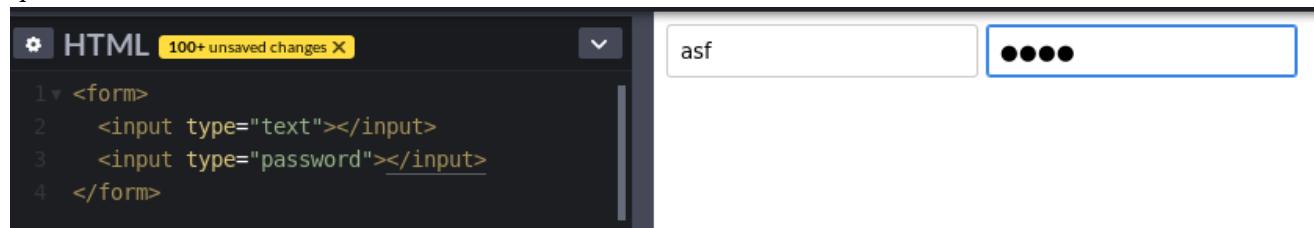
En html, el tag correspondiente es <form>

Dentro de éste tag se pone cada campo a llenar y el tipo de datos que va a recibir, siendo **text** el más común.

```
<form>
  <input type="text"></input>
</form>
```

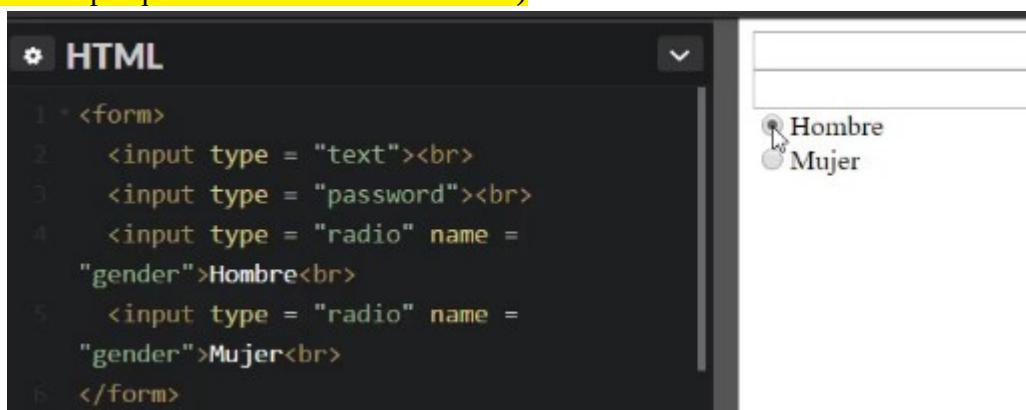


Hay otro tipo importante de datos que es **password** que se utiliza para pedir contraseñas, pues oculta lo que se está escribiendo, entre otras características:



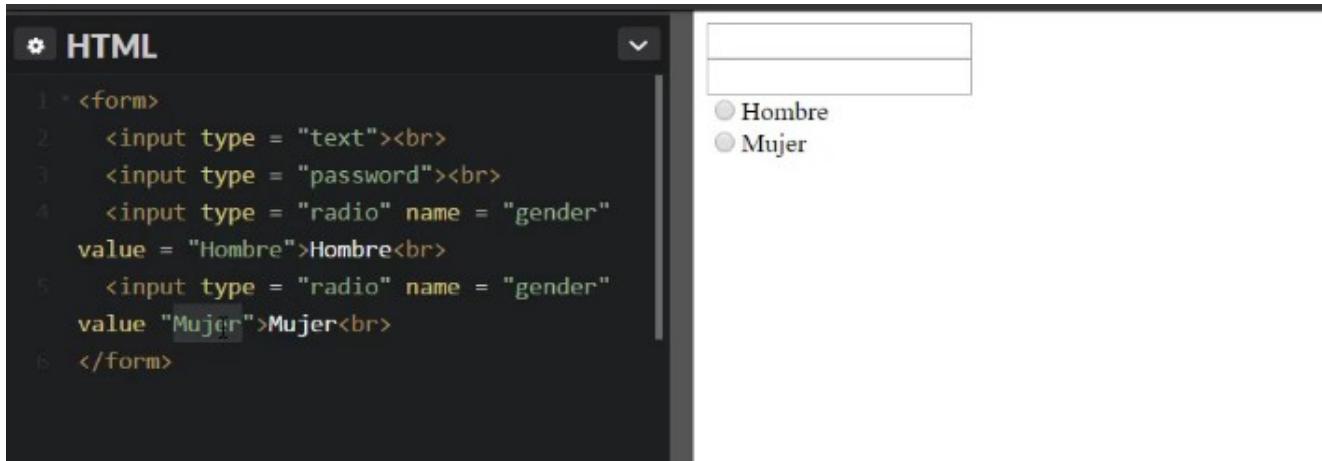
Otro tipo de datos es el de botones **radio**, que permite que se elijan varias opciones:

(Consultar porque a mi no me funciona radio)



Cuando se le da el mismo nombre a los elementos de radio en éste caso, el usuario sólo puede elegir una opción.

Pero, para usar correctamente los ratios:



A diferencia del código anterior, se le dà un value al input, lo cual hace más facil que código PHP o CSS, puedan parsear la entrada de mejor manera pues sabe los valores que adquiere al seleccionar el radio.

Por otra parte, el atributo **action** del tag **<form>** se utiliza para saber a donde redirigir la página una vez que el formulario esté completo:

```
<form action = "#">
<form action = "Pagina de gracias">
<form action = "thankyou.com">
```

Para darle al usuario la posibilidad de mandar los datos se crea el botón de submit, lo cual se hace:

```
<input type = "submit">
```

Por último, hay que darle el atributo **method** a la form, el cual puede ser “POST” o “GET” y el nombre del formulario

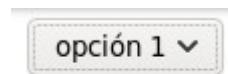
```
<form method = "POST" name = "Formulario">
```

### Select, text area y label

#### Select

El tag de **select** lo que hace es desplegar un menú de opciones y el usuario va a tener que seleccionar una con el tag option.

```
<form method = "POST" name = "Formulario">
  <select>
    <option>opción 1</option>
    <option>opción 2</option>
    <option>opción 3</option>
  </select>
</form>
```



Por otra parte, dentro del select, se pueden agrupar las elecciones con el tag **<optgroup>**

The screenshot shows a code editor on the left and a browser window on the right. The code editor contains the following HTML:

```

4 <input type = "radio" name = "gender" value = "Hombre">Hombre<br>
5 <input type = "radio" name = "gender" value = "Mujer">Mujer<br>
6 <select>
7 <optgroup label = "Adolescente">
8 <option>13</option>
9 <option>14</option>
10 <option>15</option>
11 <option>16</option>
12 <option>17</option>
13 <option>18</option>
14 <option>19</option>
15 </optgroup>
16 <optgroup label = "Adulto">

```

The browser window displays a radio button group for gender ('Hombre' and 'Mujer') and a dropdown menu for age. The dropdown menu has two sections: 'Adolescente' (with options 13, 14, 15, 16, 17, 18, 19) and 'Adulto' (with options 20-30, 31-40, 41-50, 51+). The option '17' is selected in the dropdown.

Acá se usó el atributo **label** para nombrar a cada grupo.

Para distinguir cuando usar select y cuando usar ratios:

Si el número de opciones es chico, lo mejor es usar ratios, y si el número de opciones es grande, se usa el select.

## Text area

Permite que el usuario ingrese una gran cantidad de información.

Para darle un tamaño, se le debe dar una cantidad de columnas y filas.

```
<textarea rows="20" cols="60">
</textarea>
```

## Label

Se usa para etiquetar los campos e los formularios:

```
<label for = (acá va el id del campo)>Texto</label>
```

El id del campo debe ser único en toda la página.

The screenshot shows a code editor on the left and a browser window on the right. The code editor contains the following HTML:

```

1 <form action = "thankyou.com" method = "POST" name = "admission">
2   <label for = "usuario">Usuario:</label>
3   <input type = "text" id = "usuario"><br>
4   <label for = "password">Password:</label>
5   <input type = "password" id = "password"><br>
6   <input type = "radio" name = "gender" value = "Hombre">Hombre<br>
7   <input type = "radio" name = "gender" value = "Mujer">Mujer<br>

```

The browser window displays a form with a text input field labeled 'Usuario', a password input field labeled 'Password', a radio button group for gender ('Hombre' and 'Mujer'), and a dropdown menu for age. The dropdown menu has two sections: 'Adolescente' (with options 13, 14, 15, 16, 17, 18, 19) and 'Adulto' (with options 20-30, 31-40, 41-50, 51+). The option '13' is selected in the dropdown.

## Name

Es un atributo que indica el nombre de cada elemento, lo cual es bueno poner pues los lenguajes de programación de backend van a obtener los datos que ingresa el usuario a partir del nombre del campo de la página.

## Creación de una página web con html

La primer página que se va a crear es la página principal, la cual debe llamarse siempre “index.html”. Por otra parte hay que tener en cuenta que todos los elementos de la página web y sus subpáginas, deben ir todas dentro de una misma carpeta.



# Jenifer

## Creación de la estructura de un sitio

Como se vió, ya se tiene el encabezado, pero así no es la forma correcta de organizar un sitio pues tirando líneas de código así nomás es un poco desordenado y causa muchos problemas cuando la página deba interactuar con cosas como javascript, CSS o PHP.

Existe un standard que se debe seguir.

Lo primero es agregar un tag nuevo llamado **<! DOCTYPE>** que le dice al navegador qué tipo de archivo está manejando, en nuestro caso:

**<!DOCTYPE html>**

Luego se debe indicar que el siguiente código es html lo cual se hace con el tag **<html>**.

Luego, dentro de éste bloque hay 2 componentes principales: el **<head>** y el **<body>**.

En el **head** va la parte del título de la página (que es lo que figura como título en la pestaña y su tag **<title>**) y el encoding del texto de la página (**<meta charset="utf-8">**).

En el head también van los datos de formateo y los recursos externos que se van a usar.

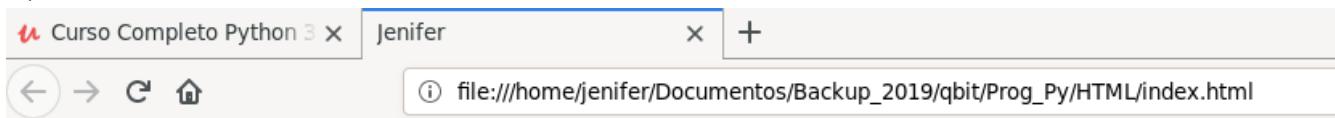
Luego, dentro del **body** van los encabezados, los párrafos, las forms, las tablas y todo lo que es el cuerpo de la página en sí.

Finalmente, y esto es opcional, puede ir un pie de página o **footer** donde usualmente van los datos del desarrollador.

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
```

```
<title>Jenifer</title>
</head>
<body>
  <h1>Encabezado</h1>
  <p>Este es el cuerpo de página</p>
  <footer><p>Sitio creado por Jenifer</p></footer>
</body>

</html>
```



## Encabezado

Este es el cuerpo de página

Sitio creado por Jenifer

Como se puede ver en la imagen, en el título de la pestaña puso lo que estaba en el tag **<title>** (por default pone el nombre del archivo que en éste caso es index.html) y el resto lo desplegó en la página y dado que el encoding es UTF-8, aceptó el tilde en la letra a de la palabra “página”.

### Sitios de apoyo

w3schools.com

En éste sitio se enumeran todos los tags que existen para html. [www.w3schools.com/tags](http://www.w3schools.com/tags)

Validator.nu

[www.validator.nu](http://www.validator.nu) es como la página de pep8online pero para validar que el código html que escribimos siga el standard.

### Desarrollo del esqueleto de una página

#### Tag **<header>**

Éste tag va dentro del body de la página, el cual sirve para poner todas las cosas que refieren a la página en sí como por ejemplo el logo de la página, el título, etc. O sea, todo lo que va arriba, en nuestro ejemplo, podemos poner el encabezado

#### Tag **<section>**

Éste tag ayuda a dividir a nuestra página en secciones. Antes se usaba el tag **<div>** con distintos ids y clases, pero ya con html5 no es necesario. Éste tag soporta (o puede tener opcionalmente) atributos globales como **id**.

#### Tag **<footer>**

Como ya se vió, sirve para poner el pié de página.

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Jenifer</title>
  </head>
  <body>
    <header>
      <h1>Encabezado</h1>
    </header>
    <section id = "seccion 1">
      <p>Este es el cuerpo de página</p>
    </section>
    <section id = "seccion 2">
      <p>Esta es otra sección de la página</p>
    </section>
    <footer><p>Sitio creado por Jenifer</p></footer>
  </body>
</html>
```

### Armado de la navegación del sitio web (nav y poniendo links videos 194 y 195)

La navegación de una página es lo que permite al usuario ir de un lado a otro dentro de una página web. Para estructurar una navegación correctamente hay que usar el tag llamado **<nav>** y dentro de ese tag, se va a armar la navegación, que en nuestro caso va a ser en base a links en una lista en forma horizontal.

Por una parte se van a utilizar los tags de listas (**<ul>** y **<li>**), y para poner los links, se va a usar el tag de anchor (**<a>**).

```
<nav>
  <ul>
    <li><a href="index.html">Inicio</a></li>
    <li><a href="about.html">Acerca de mi</a></li>
    <li><a href="contact.html">Contacto</a></li>
  </ul>
</nav>
```



- [Inicio](#)
- [Acerca de mi](#)
- [Contacto](#)

## Encabezado

Este es el cuerpo de página

Esta es otra sección de la página

Sitio creado por Jenifer

Los links van a apuntar a diferentes archivos html, los cuales se deben armar.

### Añadiendo imágenes

Las imágenes del sitio deben estar todas guardadas en una carpeta adecuada y tiene que estar en la misma carpeta donde se encuentra index.html. En nuestro caso, la carpeta se va a llamar “img”.

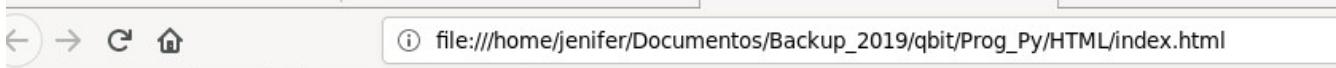
Luego lo que se hace es diseñar la estructura para desplegar las imágenes.

En nuestro caso, se van a desplegar en listas.

Para hacer ésto:

```
<section id = "seccion 3">
    <p>Esta es la sección de desplegado de imágenes</p>
    <ul>
        <li></li>
        <li></li>
        <li></li>
        <li></li>
```

```
</ul>
</section>
```



Esta es otra sección de la página

Esta es la sección de desplegado de imágenes



El atributo source (**src**) indica de donde tiene que sacar las imágenes para desplegar..

Si no se ponen en list elements separados, va a poner a cada elemento en la misma fila:

```
<section id = "seccion 3">
    <p>Esta es la sección de desplegado de imágenes</p>
    <ul>
        <li>
        
        
        </li>
    </ul>
</section>
```

- [Inicio](#)
- [Acerca de mi](#)
- [Contacto](#)

## Encabezado

Este es el cuerpo de página

Esta es otra sección de la página

Esta es la sección de despliegado de imágenes



Sitio creado por Jenifer

Por otra parte, el atributo **alt** sirve para que despliegue un texto en lugar de una imagen, lo cual es útil para que despliegue ese texto mientras se carga la página y también ayuda a los softwares de leer la pantalla, lo cual es muy usado por usuarios con discapacidad visual.

Para desplegar un texto debajo de cada figura, se usa el tag **<p>**

```
<ul>
    <li>
        <p>Cherry Blossum 1</p>
    </li>
    <li>
        <p>Cherry Blossum 2</p>
    </li>
    <li>
        <p>Cherry Blossum 3</p>
    </li>
    <li>
        <p>Cherry Blossum 4</p>
    </li>
</ul>
```

### Subtítulo

Para poner un subtítulo, se edita lo que está debajo del tag **<nav>**:

```
<h1>Jenifer</h1>
<h2>Python Student</h2>
```

- [Inicio](#)
- [Acerca de mi](#)
- [Contacto](#)

# Jenifer

## Python Student

Este es el cuerpo de página

Esta es otra sección de la página

Esta es la sección de despliegado de imágenes



Cherry Blossum 1

Cabe destacar que los headers también pueden usar el atributo **align** para desplegar si se quiere el texto centrado, alineado a la derecha o a la izquierda.

### Añadiendo páginas

Existen varias formas de estructurar una página con su flujo de trabajo o desarrollo. En nuestro caso, el flujo que se va a seguir es editar cada una de las páginas y luego se van a editar sus colores, sus fondos, el estilo de letras y su presentación con el CSS.

Éste no es el único flujo de trabajo, y el flujo elegido depende de las circunstancias.

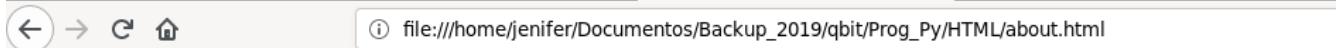
En nuestro caso, se va a desarrollar cada uno de los archivos html y luego se va a trabajar con archivos CSS.

En nuestro caso, se van a copiar todos los elementos que se repiten en index.html, pero hay que notar que ésto no es necesario si se trabaja en el backend con lenguajes como PHP, Python, Perl, Ruby, etc. Esta característica de los lenguajes de backend hace que todo sea más sencillo y se cometan menos errores. Además, ésta característica hace que, si se quiere cambiar la navegación del sitio, no hace falta modificar todos los archivos html involucrados, solo con cambiar el principal ya está.

## Página about

En nuestro caso se copia todo lo que tiene en común con la página de inicio, que son el header, nav y el footer y se edita el cuerpo de la sección.

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>Jenifer</title>
    </head>
    <body>
        <header>
            <nav>
                <ul>
                    <li><a href="index.html">Inicio</a></li>
                    <li><a href="about.html">Acerca de mi</a></li>
                    <li><a href="contact.html">Contacto</a></li>
                </ul>
            </nav>
            <section id="Acerca de mi">
                <h3>Acerca de mi</h3>
                <p>Hola, mi nombre es Jenifer, tengo 33 años y soy  
bióloga</p>
                <p>Mi espacialidad dentro de la biología es la Bioinformática  
y en particular, la Genómica Computacional</p>
                <p>Actualmente me encuentro estudiando para obtener la  
certificación de Analista en IT a través de CUTI e INEFOP</p>
                <p>Mis hobbies son leer, escuchar música, mirar series y  
anime, y me gustan los videojuegos.</p>
            </section>
            <footer><p>Sitio creado por Jenifer</p></footer>
        </body>
    </html>
```



- [Inicio](#)
- [Acerca de mi](#)
- [Contacto](#)

## Acerca de mi

Hola, mi nombre es Jenifer, tengo 33 años y soy bióloga

Mi espacialidad dentro de la biología es la Bioinformática y en particular, la Genómica Computacional

Actualmente me encuentro estudiando para obtener la certificación de Analista en IT a través de CUTI e INEFOP

Mis hobbies son leer, escuchar música, mirar series y anime, y me gustan los videojuegos.

Sitio creado por Jenifer

Observar que el header que se puso en la página es **h3**, lo cual se debe a que se debe respetar la jerarquía de los encabezados.

## Página contact

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Jenifer</title>
  </head>
  <body>
    <header>
      <nav>
        <ul>
          <li><a href="index.html">Inicio</a></li>
          <li><a href="about.html">Acerca de mi</a></li>
          <li><a href="contact.html">Contacto</a></li>
        </ul>
      </nav>
      <section id="Aviso">
        <h3>Aviso</h3>
        <p>En ésta parte puedes ponerte en contacto conmigo</p>
      </section>
      <section id="Contact">
        <h3>Contacto</h3>
        <p>Contact details</p>
        <ul>
          <li><a href="mailto:jenifer@example.com?subject=feedback">Mi correo electrónico</a></li>
          <li><a href="http://www.facebook.com/dummyuser">Mi facebook</a></li>
          <li><a href="http://www.twitter.com/dummyuser">Mi twiter</a></li>
        </ul>
      </section>

      <footer><p>Sitio creado por Jenifer</p></footer>
    </body>
  </html>
```

Notar que, al ser un link externo, lo que se pone el anchor es "

Por otra parte, si se quiere dar un link que abra un cliente de correo electrónico, se usa

## Cómo usar CSS externo

Los archivos CSS van en una carpeta aparte llamada CSS que debe estar en la misma carpeta que index.html.

### **normalize.css**

Es un archivo que sirve para renderizar las páginas de forma tal que se vean igual independientemente del navegador que se utilice y así sobreescribir el estilo por default que aplica cada navegador a las páginas web.

Para cargarlo, se va a index.html y se edita el head:

```
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="/css/normalize.css">
  <title>Jenifer</title>
</head>
```

### **Pequeña introducción a CSS**

CSS (Cascading Style Sheet, hoja de estilo en cascada) sirve para aplicarle estilo a las páginas html siguiendo una prioridad de arriba hacia abajo, o sea, una propiedad definida en la parte superior puede ser sobreescrita por la parte de abajo, lo cual se hace cuando por ejemplo cambiamos el estilo de una sección sin que se quiera cambiar el estilo del resto de las secciones de una página.

Tambien este concepto aplica cuando hay un css que remplaze algunos parámetros de otro, por ejemplo, el css que armemos, va a sobreescibir parámetros de normalize.css

Para usar código CSS en un html, se modifica el head:

```
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="/css/normalize.css">
  <style>
    h1{
      color: blue;
    }
    h2 {
      color: green;
    }
  </style>
  <title>Jenifer</title>
</head>
```

**Jenifer**  
**Python Student**

Lo de cascada es, si se pone:

```
h1{
  color: blue;
}
h1 {
  color: green;
```

```
}
```

Lo que resulta es que el `h1` queda en verde pues fué lo último que se definió. Ésto es útil cuando se quieren sobreescribir reglas en algunas partes de la página web.

## Componentes del css

```
h1 {  
    color: blue;  
}
```

**Selector:** ej `h1`, es lo que selecciona qué contenido va a llevar el estilo. El de el caso ejemplo es un element selector pues selecciona un elemento del html. Hay otros tipos de selectores como id o class selectores.

**Regla de CSS:** Lo compone el selector mas las reglas que se le van a aplicar, que pueden tener más de una declaración.

**Declaración:** Es la propiedad que se desea cambiar y el valor que se le va a dar. La propiedad se asigna con : y la declaración se cierra con ;

## Creación de archivos CSS

Usualmente no es buena práctica poner código CSS dentro de archivos HTML. Por lo general lo que se hace es crear CSS externos e invocarlos en el HTML.

En nuestro caso, el archivo que se va a crear es uno llamado `main.css`.

Para incluirlo en nuestra página, se edita el head de `index.html`:

```
<head>  
    <meta charset="utf-8">  
    <link rel="stylesheet" href="/css/normalize.css">  
    <link rel="stylesheet" href="/css/main.css">  
    <title>Jenifer</title>  
</head>
```

Dada la naturaleza en cascada de CSS, es importante que `main.css` vaya debajo de `normalize.css`.

Por otra parte, en el `main.css`:

```
body {  
    background-color: pink;  
}
```

Ésto nos da:

- [Inicio](#)
- [Acerca de mi](#)
- [Contacto](#)

# Jenifer

## Python Student

Este es el cuerpo de página

Esta es otra sección de la página

Esta es la sección de desplegado de imágenes



### ID selectors

Como se quiere trabajar con la página de forma tal que las imágenes no ocupen toda la pantalla pero lo demás si, se usa en inex.html el tag **div** con un atributo **id** para poder editar eso en el .css:

```
<div id="wrapper">
    <section id = "seccion 3">
        <p>Esta es la sección de desplegado de imágenes</p>
        <ul>
            <li>
                <p>Cherry Blossom 1</p>
            </li>
            <li>
                <p>Cherry Blossom 2</p>
            </li>
            <li>
                <p>Cherry Blossom 3</p>
            </li>
            <li>
                <p>Cherry Blossom 4</p>
            </li>
        </ul>
    </section>
</div>
```

Y en el main.css:

```
body {
    background-color: pink;
}
#wrapper{
    background-color: violet;
```

}

El # es para que el archivo CSS busque ese id.

Como resultado:

- [Inicio](#)
- [Acerca de mi](#)
- [Contacto](#)

# Jenifer

## Python Student

Este es el cuerpo de página

Esta es otra sección de la página

Esta es la sección de desplegado de imágenes

### Max-with, margin y padding

#### **Max-with**

A veces no es útil que todo el wrap ocupe toda la pantalla.

Para limitar el tamaño que va a ocupar una sección, se usa max-with.

En nuestro ejemplo:

```
#wrapper{  
    background-color: violet;  
    max-width: 960px;  
}
```

Da como resultado:

- [Inicio](#)
- [Acerca de mi](#)
- [Contacto](#)

# Jenifer

## Python Student

Este es el cuerpo de página

Esta es otra sección de la página

Esta es la sección de desplegado de imágenes

## Margin

Se usa para determinar la posición de la sección que se está editando:

```
#wrapper{  
    background-color: violet;  
    max-width: 960px;  
    margin: 0 auto;  
}
```

0 significa que no deje espacios arriba ni abajo y **auto** es para que deje espacios a la derecha o a la izquierda de forma tal que quede centrado el wrapper en el navegador y deje un margen de igual tamaño hacia la izquierda y hacia la derecha.

Al hacer ésto:

The screenshot shows a web page with a central content area. At the top left, there is a navigation menu with three items: 'Inicio', 'Acerca de mi', and 'Contacto'. Below the menu, the main content area has a pink background. It contains the text 'Jenifer' and 'Python Student'. Underneath this, there is a section titled 'Esta es el cuerpo de página' followed by the text 'Este es otra sección de la página'. At the bottom of the content area, there is a heading 'Esta es la sección de despliegado de imágenes' followed by a large image of cherry blossoms against a blue sky. To the right of the image, there is a vertical orange bar.

## Padding

El padding es la distancia entre lo que engloba todo y los elementos que hay dentro

```
#wrapper{  
    background-color: violet;  
    max-width: 960px;  
    margin: 0 auto;  
    padding: 0 5%;  
}
```

Con ésto se quiere decir que la distancia hacia arriba sea 0 y que hacia la derecha e izquierda haya una distancia del 5% del wrap:

The screenshot shows the same web page as before, but now with padding applied. The central content area has a pink background with a white inner border. The text 'Jenifer' and 'Python Student' is centered within this area. Below it, the sections 'Esta es el cuerpo de página' and 'Esta es otra sección de la página' are also centered. At the bottom, the image of cherry blossoms is centered within its own white-bordered box. The vertical orange bar on the right side remains the same.

¿Se nota la diferencia? A la izquierda es sin el padding y a la derecha es con.

## Border

Determina el borde:

```
#wrapper{
```

```
border: 5px solid red;
background-color: violet;
max-width: 960px;
margin: 0 auto;
padding: 0 5%
}
```

## Jenifer

### Python Student

Este es el cuerpo de página

Esta es otra sección de la página

Esta es la sección de desplegado de imágenes



Cherry Blossom 1

## Text align

Se emplea para alinear texto y sus valores pueden ser center, right, left, justificado, o heredar de una clase padre.

```
#card{
    text-align: center;
    margin: 0
}
nav{
    text-align: right;
}
```

- [Inicio](#)
- [Acerca de mi](#)
- [Contacto](#)

## Jenifer

### Python Student

Este es el cuerpo de página

Esta es otra sección de la página

Esta es la sección de desplegado de imágenes



## Comentarios y colores

Para poner comentarios en CSS, se abren con /\* y se cierran con \*/

Los comentarios se utilizan para destacar cosas en el código pero que van a ser ignoradas por el navegador.

Por otra parte, por default, los navegadores ponen decoraciones como subrrayado a los links.

Para que no lo haga:

```
a{ /*selecciona todos los anchors */
    text-decoration: none;
}
```

En cuanto a los colores y el tipo de letra, se pueden modificar dentro de la sentencia adecuada.

```
header{
    background: pink;
    background-color: lightblue;
    font-style: italic;
    color: green;
}
```



Por otra parte nav a:visited hace que regule cómo se va a ver ése link una vez visitado:

```
nav a, nav a:visited{
    color: green;
    text-align: right;
}
```

En éste caso, se va a ver igual que en los links no visitados.

## Subclases

Al usuario hay que arle alguna forma de saber en qué página se encuentra y también qué ocurre cuando pasa el mouse por encima del link (hover), lo cual se hace asignando clases en el html y poniendo en el CSS que queremos que ilumine esa clase:

En el HTML:

```
<nav>
    <ul>
        <li><a href="index.html" class="selected">Inicio</a></li>
        <li><a href="about.html">Acerca de mi</a></li>
        <li><a href="contact.html">Contacto</a></li>
```

```
</ul>
</nav>
```

Y en el CSS:

```
a.selected, a:hover {
    background-color: violet;
}
```

Con eso:



Para los archivos about y contact.html lo que hay que hacer es poner la clase “selected” donde corresponda.

### Dando formato al menú

Para desplegar el menú correctamente, sin los bullets:

```
ul{
    list-style-type: none;
    margin: 0
    padding: 0
}
```

Pot otra parte, para hacerlo horizontal en vez de vertical, se le dice a cada uno de los elementos de la lista que tienen que ir en la misma linea:

```
li{
    display: inline;
}
```

Pero para que aplique solo a los elementos de nav, se debe sustituir **li** por **nav li**

Con ésto:



Este es el cuerpo de página

Por otra parte se puede fijar la barra para que permanezca dibujada aunque el usuario haga scroll:

```
ul{
    list-style-type: none;
    position: fixed;
    margin: 0
    padding: 0
}
```

Ojo, ésto sobreescribe que nav a esté alineado a la derecha.  
Por otra parte ésto me rompe el sitio.

### Añadir estilo al body

```
body {  
    background-color: pink;  
    color: darkgreen;  
}
```

Le da un color verde a las letras y rosa al fondo.

### Organización de CSS

Lo que se hace es poner comentarios acerca de lo que hace cada comando de CSS poniendo tambien arriba todo lo que es configuración general y luego lo que tiene que ver con secciones particulares:

```
*****
```

#### Configuración General

Acá van las configuraciones que aplican en todo el sitio, que es el header, el body y los links

```
*****
```

```
/*Hace que el encabezado tenga un color de fondo azul  
y la letra sea itálica*/
```

```
header{  
    background: pink;  
    background-color: lightblue;  
    font-style: italic;  
    color: green;  
}
```

```
/*ésto hace que el cuerpo de la página tenga  
letras verdes y fondo rosa*/
```

```
body {  
    background-color: pink;  
    color: darkgreen;  
}
```

```
/*ésto hace que la sección de imágenes de la página  
tenga fondo violeta, con un ancho máximo de 960px,  
un margen de 0 hacia arriba y alineado parejo a la derecha  
y a la izquierda, y una distancia de 5% entre el borde y los  
elementos que están dentro de la sección*/
```

```
#wrapper{  
    background-color: violet;  
    max-width: 960px;
```

```
margin: 0 auto;
padding: 0 5%
}

/*selecciona todos los anchors y les saca el decorado
que les pone el navegador por defecto, que es color azul
y subrayado*/
a{
    text-decoration: none;
}

/*selecciona el anchor correspondiente a la clase "selected"
y les da un color de fondo violeta. hace lo mismo para cuando
el mouse se para sobre los links*/
a.selected, a:hover {
    background-color: violet;
}

/*Fin de Configuración General*/
```

```
*****
```

Configuración de la barra de navegación:

```
*****
/*le saca los bullets a los elementos de la lista*/
ul{
    list-style-type: none;
    margin: 0
    padding: 0
}

/*hace que los elementos de la lista que están en nav, se
presenten en una sola linea, lo cual hace que la barra de navegación
sea horizontal*/
nav li{
    display: inline;
}

nav a, nav{
    color: green;
    text-align: right;

}
/*Fin de Configuración de la barra de navegación: */
```

```
*****
```

### Configuración del título de la página

```
*****  
/*ésto hace que el título de la página principal esté  
centrado*/  
#card{  
    text-align: center;  
    margin: 0  
}
```

### Uso de fonts externos

Para usar fonts externos, se deben referenciar tanto en el html como en el CSS, en nuestro caso:

HTML:

```
<head>  
    <meta charset="utf-8">  
    <link rel="stylesheet" href="css/normalize.css">  
    <link rel="stylesheet" href="css/main.css">  
    <link href="https://fonts.googleapis.com/css?  
family=Dancing+Script&display=swap" rel="stylesheet">
```

CSS:

```
header{  
    background: pink;  
    background-color: lightblue;  
    font-family: 'Dancing Script', cursive;  
    color: green;  
}  
body {  
    background-color: pink;  
    color: darkgreen;  
    font-family: 'Dancing Script', cursive;  
}
```



## em, font-family y font-size

### **font-family**

font-family lo que hace es especificar el font para un selector en particular y, si no puede cargar esa fuente, hay que especificar una alterna (en nuestro caso pusimos cursive)

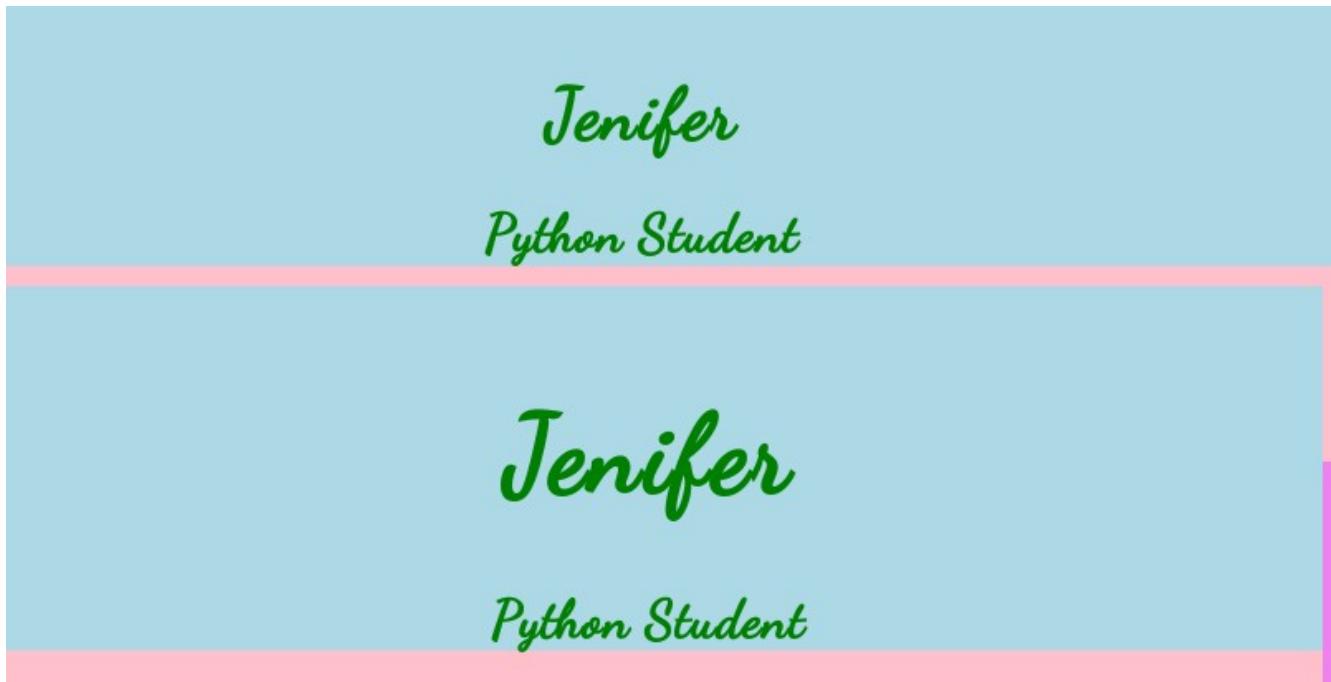
### **em**

Es una unidad que se le pone al font-size que es relativa a otra, por ejemplo

```
#wrapper{  
    background-color: violet;  
    max-width: 960px;  
    margin: 0 auto;  
    padding: 0 5%  
}
```

```
h1{  
    color: green;  
    font-size: 3em;  
    font-family: 'Dancing Script', cursive;  
}
```

Lo que hace es darle al Header 1, un tamaño de 3 veces el tamaño que tienen actualmente todas las letras.



En la figura se ve el antes y después de aplicado el em en el h1

### **font-weight**

Lo que hace esto es decirle cómo desplegar el H1, en nuestro caso, como no se quiere que se despliegue en bold, se pone:

```
h1{  
    color: green;  
    font-size: 3em;  
    font-family: 'Dancing Script', cursive;  
    margin: 20px 0;  
    font-weight: normal;  
}
```



### Line-height

```
h1{  
    color: green;  
    font-size: 3em;
```

```
font-family: 'Dancing Script', cursive;
margin: 20px 0;
font-weight: normal;
line-height: 0.6em;
}

h2{
    color: green;
    font-family: 'Dancing Script', cursive;
    font-weight: normal;
    margin-top: -5px
}
```

Lo que hace ésto es separar un poco h1 de h2 (**line-height** en h1) pues **line-height** fija el espaciamiento que hay entre las líneas de texto.

Por otra parte, con **margin-top**, lo que se indica es que queremos un margen de arriba de -5px. Es una práctica común cuando se quiere que un elemento esté más pegado hacia otro.

### Fuente para el cuerpo de la página

Para el cuerpo de la página, se va a usar open sans.

En el CSS:

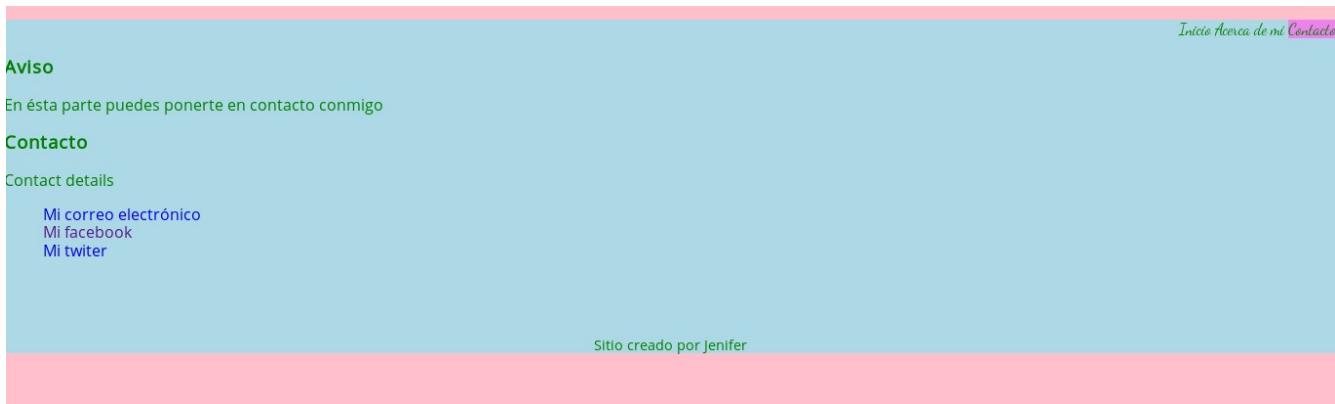
```
body {
    background-color: pink;
    color: darkgreen;
    font-family: 'Open Sans', sans-serif;
}

#Acercademi, #Aviso, #Contact{
    font-family: 'Open Sans', sans-serif;
}

footer{
    font-size: 0.9em;
    text-align: center;
    padding-top: 50px;
    font-family: 'Open Sans', sans-serif;
}
```

En los HTMLs:

```
<link href="https://fonts.googleapis.com/css?family=Open+Sans&display=swap" rel="stylesheet">
```



Con ésto se hizo que el cuerpo y el footer tengan como font “Open Sans”, y que el footer quede centralizado con letra más chica.

### Toques finales de navegación

Se quiere que primero se vea el título y luego la barra de navegación, por lo tanto se cambia el orden en el HTML:

```
<div id="card">
    <h1>Jenifer</h1>
    <h2>Python Student</h2>
</div>
<nav>
    <ul>
        <li><a href="index.html" class="selected">Inicio</a></li>
        <li><a href="about.html">Acerca de mi</a></li>
        <li><a href="contact.html">Contacto</a></li>
    </ul>
</nav>
```

Y en el CSS:

```
nav a, nav{
    color: green;
    background-color: grey;
    text-align: center;
    font-family: 'Dancing Script',
    cursive;
}
```

Se cambió el background color a gris de la barra de nav para que se note que es una barra de menú:

### Arreglado de elementos de la galería

La idea es que primero se vea bien en pantallas móviles que son más pequeñas.

Para eso, se debe modificar la galería de forma tal que despliegue correctamente los elementos de la misma.



Para eso, a la sección correspondiente, se le va a poner el id de gallery y se lo va a llamar como **#gallery** desde el archivo CSS:

```
*****
```

## Galería

```
*****
```

```
#gallery{  
    margin: 0;  
    padding: 0;  
    list-style: none;  
}
```

Esto hace que le saque los bullets a la lista y no haya espacios arriba entre la sección de la galería y el resto.



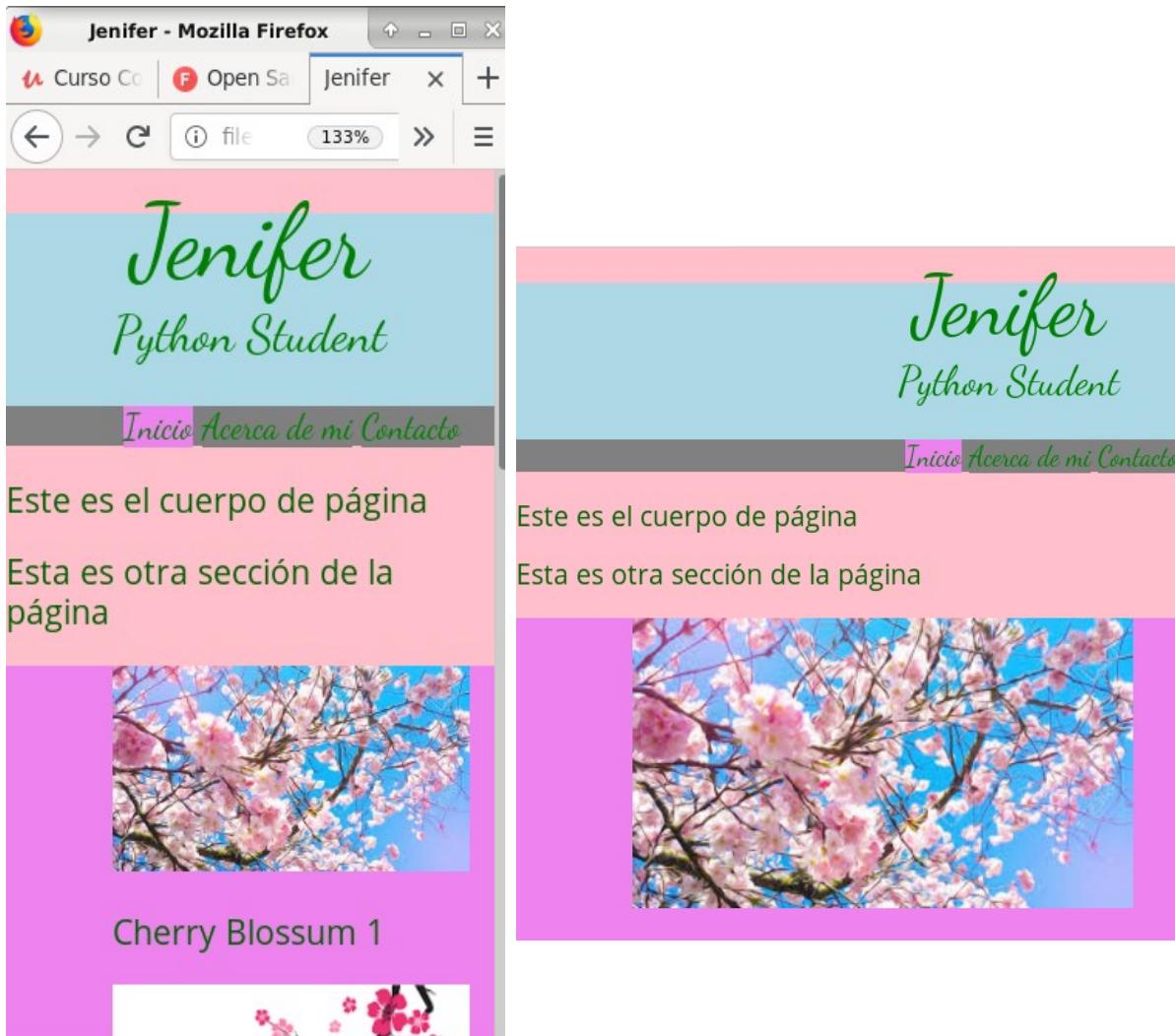
## Floats

Floats sirve para en nuestro caso, que las fotos ocupen como máximo 2 o 3 espacios.

Primero lo que se va a hacer es en la configuración general de CSS:

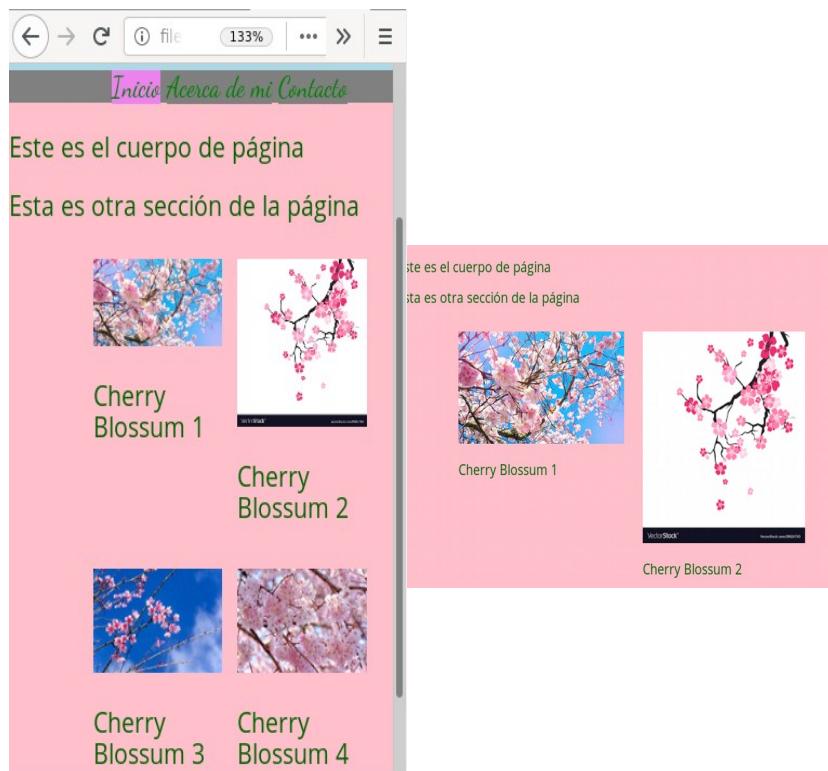
```
img{  
    max-width: 100%;  
}
```

Con ésto, lo que se hace es decirle a la página que, si la pantalla es pequeña, achique las imágenes porque se le está diciendo que solo ocupe hasta el 100% del display:



Ahora, para que despliegue la galería con hasta un máximo de 2 fotos por fila:

```
#gallery li{  
    width: 45%;  
    margin: 2.5%;  
    float: left;  
}  
}
```

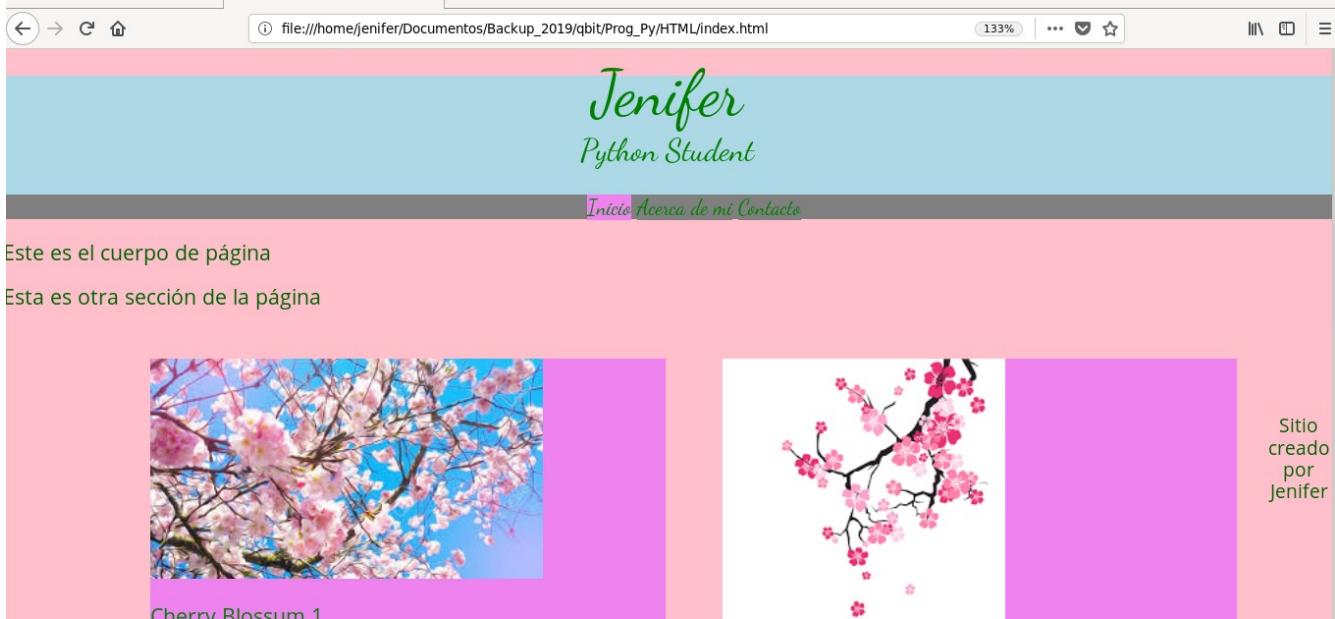


Si se agranda o achica la pantalla no importa, siempre se van a ver 2 imágenes por linea porque el float de cada elemento es de 45% y sus destancias entre sí son 5%.

### **Arreglando el background y los pies de fotos**

```
#gallery li{  
    width: 45%;  
    margin: 2.5%;  
    float: left;  
    background-color: violet;  
    color: green;  
}
```

Para que el background de cada foto sea violeta y las letras sean verdes.



### Arreglando el footer

El problema con el footer de la página es que, cuando se ponen elementos flotantes, todo lo que sigue a continuación se pone al costado conglomerándose en los espacios libres.

Para evitar ésto se usa la propiedad clear:

```
footer{
    font-size: 0.9em;
    text-align: center;
    padding-top: 50px;
    font-family: 'Open Sans', sans-serif;
    clear: both;
}
```

Ésto lo que hace es limpiar todo el espacio que quede a la izquierda y a la derecha.

### Arreglando el header



Para que se arregle el título y pasar de tenerlo como está a la izquierda a como está en la derecha:

```
header{
    background: pink;
    background-color: lightblue;
```

```
font-family: 'Dancing Script', cursive;
color: green;
float: left;
margin: 0 0 30px 0; /*margenes top left bottom right*/
padding: 5px 0 0 0;
width: 100%;
```

```
}
```

El width es porque, al ser flotante, pierde su característica de ocupar toda la parte superior. Para recuperarla, se pone una anchura del 100%.

### Inline blocks

```
nav ul{
    list-style: none;
    margin: 0 15px; /*margen de 0 arriba y abajo y 10 de derecha e
izquierda*/
    padding: 0;
}
```

```
nav li{
    display: inline-block;
}
```

Ésto hace que los elementos internos de la lista, se desplieguen todos en la misma linea.

Éste tipo de display hace que los links tengan parte de comportamiento como si fuera de tipo de linea y no de tipo bloque.

### Agregado de padding y font-weight en la barra de navegación.

Ésto se hace para que los links no estén tan pegados entre si.

```
nav a, nav{
    color: green;
    background-color: grey;
    padding: 10px;
    text-align: center;
    font-family: 'Dancing Script', cursive;
```

```
}
```

```
nav a{
    padding: 10px 10px;
    font-weight: bold;
}
```

Con ésto se separan unos items de otros.

### Diseño de la página about.html y contact.html (videos 225 y 226)

Para editar éstas páginas, lo único que se hace es copiar todo index.html y sustituir la sección de galería por lo que se quiere en cada sección.

Por otra parte, para quitar el espacio entre en header 3 y la barra de navegación:

```
h3{
    margin: 0 0 1em 0;
}
```

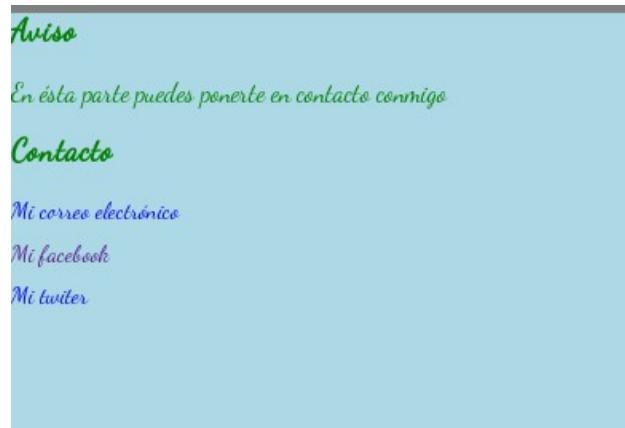
Por otra parte, en contact.html, se modificó lo siguiente:

```
<section class="aviso">
    <h3>Aviso</h3>
    <p>En ésta parte puedes ponerte en contacto conmigo</p>
</section>
<section>
    <h3>Contacto</h3>
    <p>Contact details</p>
    <ul class="contact-info">
        <li><a href="mailto:jenifer@example.com?subject=feedback">Mi correo electrónico</a></li>
        <li><a href="http://www.facebook.com/dummyuser">Mi facebook</a></li>
        <li><a href="http://www.twiter.com/dummyuser">Mi twiter</a></li>
    </ul>
</section>
```

Pues son las clases que se van a usar para el CSS

```
.contact-info{
    list-style: none;
    font-size: 0.9em;
    margin: 0;
    padding: 0;
}

.contact-info li{
    margin-bottom: 10px;
}
```



El punto es un class selector, el margin 0 es para que no deje márgen en las listas y el margin-bottom es para que deje un espacio de 10px entre los elementos de la lista.

### Creación de hoja de estilo nueva para pantallas grandes

Para eso se va a crear un nuevo archivo CSS que maneje las resoluciones de pantalla, al cual llamaremos responsive.css y se debe incluir en los cabecales de todos los HTMLs:

```
<link rel="stylesheet" href="css/normalize.css">
<link rel="stylesheet" href="css/main.css">
<link rel="stylesheet" href="css/responsive.css">
```

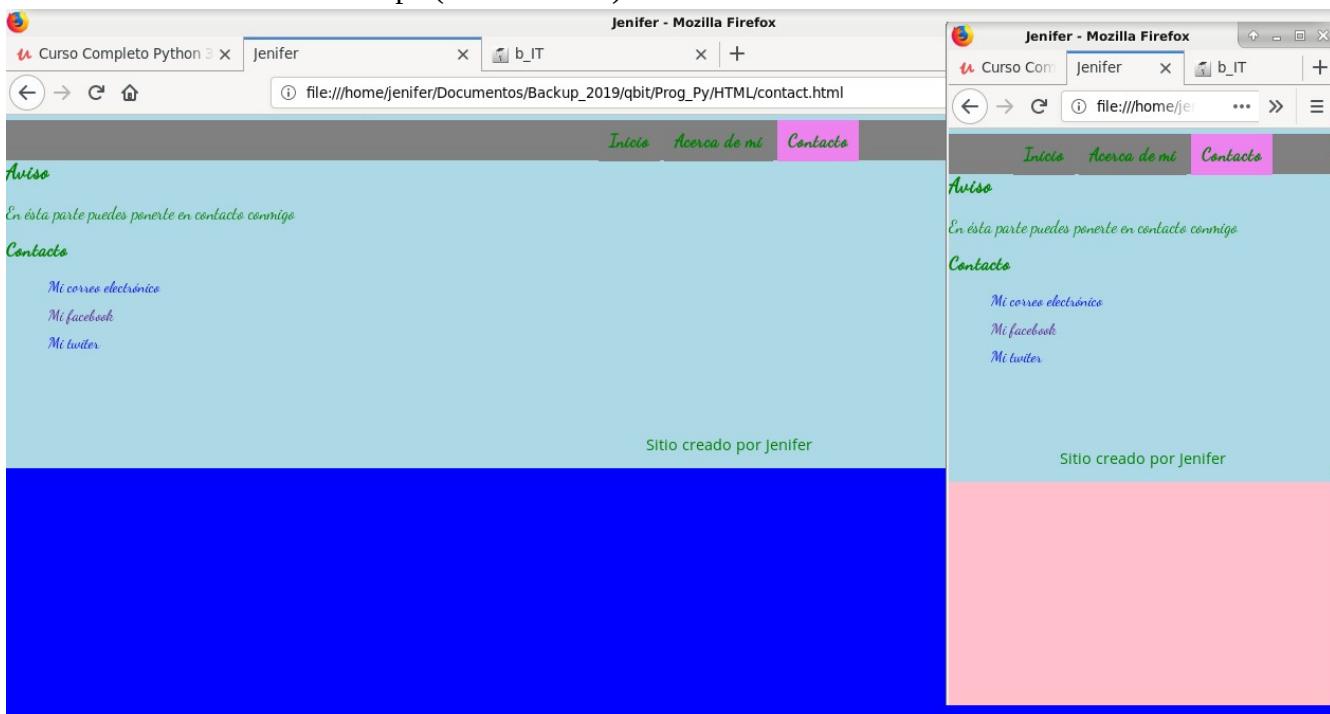
## Media queries

Los media queries son una forma de poner puntos donde se van a ejecutar ciertos códigos CSS:

Ejemplo:

```
@media screen and (min-width: 480px) {
  body{
    background-color: blue;
  }
}
```

Lo que significa es que, si se trata de un monitor (screen) y su tamaño mínimo es 480px, hace que el body text tome un color verde. El 480px es porque es la resolución mínima de las pantallas de celulares cuando están en modo landscape (en horizontal).



Sin embargo lo que se recomienda es que los media queries se hagan dependiendo de los break points del contenido, no de el tamaño popular de lo de los de las pantallas de los teléfonos en los que se estén creando los sitios web.

Para modificar la parte de contacto, en el CSS si la pantalla es grande:

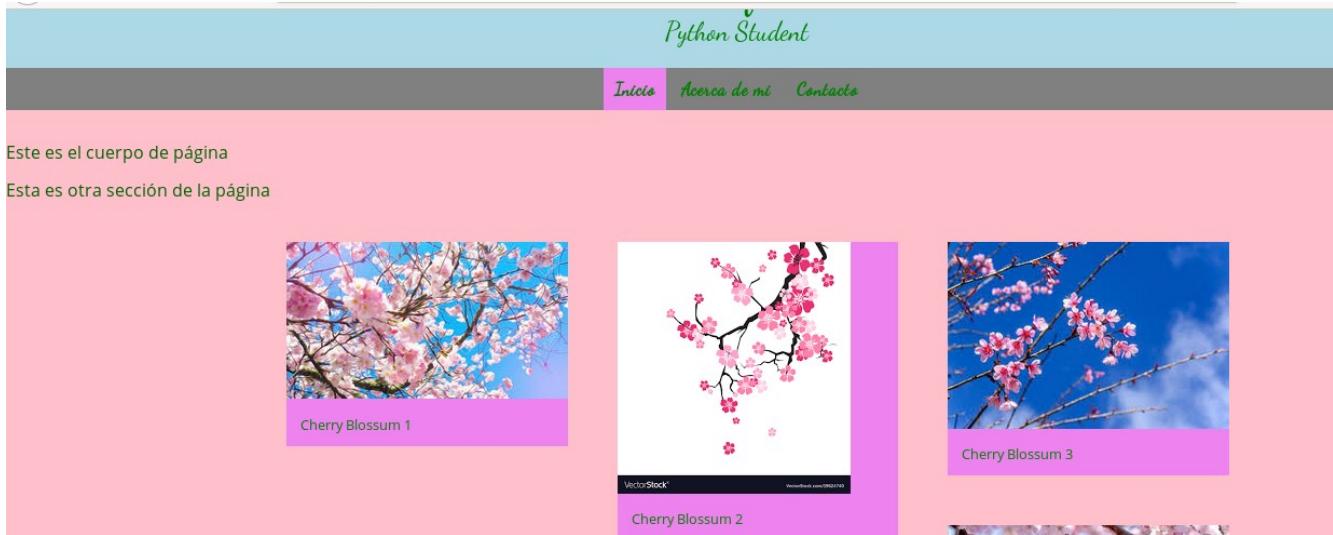
```
@media screen and (min-width: 480px) {
    #first{
        float:left;
        width: 60%;

    }
    #second{
        float:right;
        width: 40%;
    }
}
```



Por otra parte, para que ponga 3 fotos por fila cuando la pantalla es grande:

```
#gallery li{
    width: 28.33% /*ésto es para que sean 3 imágenes por fila*/
}
```

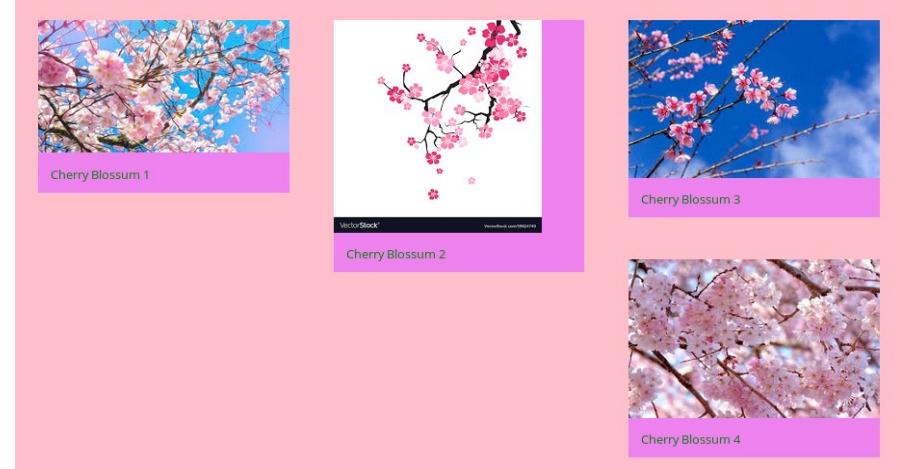


### **nth-child**

Con éstos cambios, dado que el pié de foto es grande, luego del tercer elemento, al no encontrar espacio, la cuarta foto se despliega al final de la linea. Para arreglar ésto se usa **nth-child**:

```
#gallery li:nth-child(4n){
    clear: left;
```

Cada 4to elemento hijo, limpia el espacio hacia la izquierda



```
#gallery li:nth-child(4n){  
    clear: left;  
}
```



### Mejorando la navegación para sitios con pantalla grande (videos 232 y 233)

Lo mejor para sitios grandes, es cambiar la configuración de **nav** para que se despliegue a la derecha y no tenga un fondo y además, alinear lo que esté en el tag id **card** que es donde está el título de la página:

```
@media screen and (min-width: 768px) {  
    nav{  
        width: 45%;  
        text-align: right;  
        margin-right: 5%;  
        font-size: 2m;
```

```
    float: right;
    background: none
}

#card{
    text-align: left;
    width: 45%;
    margin-left: 5%;
    float: left;
}
}
```

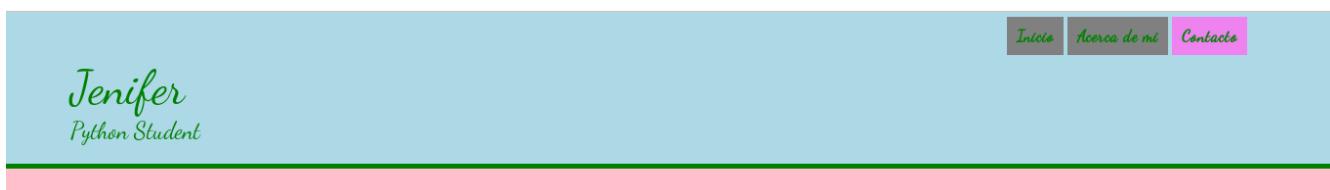
Luego se trabaja con el h1, h2 y el header:

```
h1 {
    font-size: 3em;
}

h2 {
    font-size: 1.5em;
}

header{
    margin-bottom: 50px;
    border-bottom: 5px solid
}
```

La propiedad border toma hasta 3 parámetros en el siguiente orden: el ancho, el tipo de borde y el color. Como en nuestro caso, no pusimos el color, toma el color fijado antes, que en nuestro caso es verde:



Recordar que éstos cambios solo aplican cuando la pantalla es de más de 768px.

## Referencias

Curso de Aldo Olivares <https://www.udemy.com/course/python-curso-completo/>

OOP en Python: <https://www.youtube.com/watch?v=iliKayKaGtc>

Testeo web de expresiones regulares: <https://regrexr.com/>

Expresiones regulares <https://docs.python.org/2/library/re.html>

Cheat Sheet de Expresiones Regulares: <https://www.cheatography.com/davechild/cheat-sheets/regular-expressions/>

Expresiones Regulares con Python <https://relopezbrigia.github.io/blog/2015/07/19/expresiones-regulares-con-python/>

Peewee Documentation: <http://docs.peewee-orm.com/en/latest/>

Dear Diary: An encrypted command-line diary on Python <https://charlesleifer.com/blog/dear-diary-an-encrypted-command-line-diary-with-python/>

PEP 8: Guia de usuario <https://www.python.org/dev/peps/pep-0008/>

PEP8 online: Para chequear si lo que estamos escribiendo siguen las convenciones  
<http://pep8online.com/>

Flask: <https://pypi.org/project/Flask>

Documentación de Flask: <https://flask.palletsprojects.com/en/1.1.x/> y  
<https://web.archive.org/web/20160604162342/http://mitsuhiko.pocoo.org/flask-pycon-2011.pdf>  
<https://buildmedia.readthedocs.org/media/pdf/flask/latest/flask.pdf>

Funciones decoradoras: <https://www.youtube.com/watch?v=c9J7FHLjBds&feature=youtu.be>

Página principal de Django: <https://www.djangoproject.com/>

CodePen: <https://codepen.io/>

Normalize.css: <https://necolas.github.io/normalize.css/>