**Python Intermedio**

1. El Zen de Python se compone por los principios para escribir tu código de manera clara, sencilla y precisa. Estos son:

* **Bello es mejor que feo**:  
  Pyhton es estéticamente superior a cualquier otro lenguaje de programación. Al momento de escribir código, es mejor que sea de manera limpia y estética.
* **Explícito es mejor que implícito**:  
  Hacer más fácil que las otras personas entiendan el código.
* **Simple es mejor que complejo**:  
  Es mejor tener una implementación simple, que ocupe pocas lineas de código y sea entendible, a que sea una larga y complicada.
* **Complejo es mejor que complicado**:  
  Si tenemos que extendernos en la implementación y hacerla más compleja para que el código si se entienda, esto es mejor que hacerlo simple y mal.
* **Plano es mejor que anidado**:  
  El *anidamiento* es cuando tenemos un bloque de código dentro de otro bloque de código (dependiendo de este). Esto se nota en Python por la identación, nos quedarían estos bloques muy corridos a la derecha.  
  Es mejor evitar el anidamiento, y hacer las cosas de **manera plana**.
* **Espaciado es mejor que denso**:  
  Por la identación de Python (sus sangrías), este principio se nos hace imposible de esquivar. El código inevitablemente es espaciado.
* **La legibilidad es importante**:  
  Es importante que otros programadores puedan entender lo que estamos escribiendo. Esto hace más fáciles las cosas cuando trabajemos con otros en los proyectos.
* **Los casos especiales no son lo suficientemente especiales cpmo para romper las reglas (sin embargo, la practicidad le gana a la pureza)**:  
  Siempre que podamos respetar estas reglas que nos plantea Python, es mejor así. Sin embargo, si por el hecho de hacer un código muy puro o muy ‘Pythonista’, este pierde legibilidad, es mejor ser más prácticos y romper o saltearnos algunas de estas reglas para que el código sea más eficiente. Por lo tanto, llegado el momento debermos decidir si es mejor hacer las cosas de manera pura o práctica.
* **Los errores nunca deberían pasar silenciosamente (a menos que se silencien explícitamente)**:  
  Manejar los erroes es fundamental. Cada error nos dice algo y hay que prestarle atención. A menos que seas capaz de silenciar un error explícitamente, aunque para esto hay que tener criterio.
* **Frente a la ambiguedad, evitar la tentación de adivinar**:  
  Nuestro código debería solamente tener una interpretación. Si en un contexto significa algo, y en otro otra cosa, es mejor que lo revisemos y busquemos una solución.
* **Debería haber una, y preferiblemente sola, una manera obvia de hacerlo. (A pesar de que esa manera no sea obvia a menos que seas holandés)**:  
  Esto hace referencia al creador de Python ''Guido van Rossum", que de manera muy inteligente encontrar las soluciones precisas a los problemas, y deberíamos imitarlo.
* **Ahora es mejor que nunca**:  
  Es mejor desarrollar nuestra solución cuánto antes, no dejarlo para mañana o para mas adelante.
* **A pesar de que nunca es muchas veces mejor que *ahora* mismo**:  
  Si por hacer las cosas ya y tenemos poco tiempo, si es mejor dejarlo para después y no hacerlo apurado y mal.
* **Si la implementación es díficil de explicar, es una mala idea, y si es fácil de explicar, es una buena idea**:  
  Si somos capaces de explicar nuestra implementación a otros desarrolladores paso a paso, es una buena idea. En cambio si no podemos hacerlo, significa que ni nosotros entendemos la implementación y deberíamos repensar nuestra forma de encarar la solución.
* \*\*Los espacios de nombres son una gran idea, ¡Tengamos más de esos! (namespaces):  
  Es el nombre que se ha indicado luego de la palabra import, es decir la ruta (namespace) del módulo. (Lo veremos a profundidad más adelante).

## 2. Creando un ambiente virtual con VENV

Creación de ambiente Virtual: (significa que ya no usaremos el Python global de nuestra computadora, sino que estamos usando un Python clonado que solamente funciona en este proyecto. De esta manera también, cuando instalamos módulos con pip, se instalan solamente en este proyecto y no de manera global en la computadora)

python3 -m venv nombre\_venv

* Usualmente el nombre del ambiente virtual es venv.

Activación del ambiente virtual:

* Windows:

.\venv\Scripts\activate

* Unix o MacOS:

source venv/bin/activate

Desactivar el ambiente virtual:

deactivate

Crear un alias en linux/mac:

alias nombre-alias="comando"

`alias avenv=“source venv/bin/activate”``

Crear alias en Windows

‘alias avenv=.\venv\Scripts\activate’

Clase del curso de [Git y Github](https://platzi.com/clases/1557-git-github/19952-tags-y-versiones-en-git-y-github/) 👩🏽‍💻

Aquí les dejo los pasos para crear un alias en Linux Ubuntu:  
.  
Para hacerlo en este sistema operativo, necesitamos que cada que la terminal cargue, el alias sea leído, para ello, la terminal tiene un archivo llamado .bashrc que contiene la configuración inicial, y usualmente se encuentra en nuestro home, por lo que hacemos lo siguiente:  
.

1. Ejecutar sudo nano ~/.bashrc
2. Ir al final del archivo
3. Agregar el comando: alias avenv='source venv/bin/activate'
4. Guardar presionando ctrl + o y luego salir con ctrl + x
5. Reejecutar la configuración de la terminal: source ~/.bashrc
6. Activar el entorno vitual avenv

.  
De esa forma persistirá siempre, ya que el alias se guarda dentro del archivo de configuración de la termial 😄

3. **Pip (package installer for python)** Nos permite descargar paquetes de terceros para utilizarlos en nuestro enviroment, ademas se puede definir una versión especifica del paquete.  
|

* pip install <paquete> instala el paquete(pandas , matplotlib, bokeh, etc) que se especifique
* pip freeze muestra todos los paquetes instalados en tu ambiente virtual

|  
Si quisiéramos que alguien mas pueda ejecutar nuestro proyecto es importante compartir que librería y versión hemos empleado; eso se realiza con el comando:

pip **freeze** > requirements.txt

El resultado de pip freeze se escribe en requirements.txt (puedes usar otro nombre pero el mostrado es una buena practica)  
|  
para instalar paquetes desde un archivo como requirements.txt ejecutamos:

pip **install** -r requirements.txt

4. Buenas prácticas  
**[Dejar dos espacios (líneas de código) entre las funciones]  
[Crear una función principal]**  
Esta línea de código:  
if \_\_name\_\_ == "\_\_main\_\_": main() //es el punto de entrada de un programa de Python. Una vez que se encuentra Python con esta línea de código, empieza a ejecutar todo lo qué esté abajo, cómo en este ejemplo, la función "main".

También, dejar siempre dos espacios de salto de linea entre las funciones

## 4 List comprehensions

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

5. lambdaInterfaz de usuario gráfica, Texto

Descripción generada automáticamente

6. Filter, map y reduce

La diferencia entre **filter y map**:

* *filter* devuelve True or False según el valor esté dentro de los criterios buscados o no. En caso de que no cumpla con la condición, no será devuelto y la lista se verá reducida por este filtro.
* *Map* funciona muy parecido, pero su diferencia radica en que no puede eliminar valores de la lista del array entregado. Es decir, el output tiene la misma cantidad de valores que el input.

Cómo funciona **reduce**:

* *Reduce* toma 2 valores entregados como parámetros y el iterador como otro parámetro. Realiza la función con estos 2 valores, y luego con el resultado de esto y el valor que le sigue en el array. Y así hasta pasar por todos los valores de la lista.

Filter

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Map

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Reduce

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja

Acá lo que hace el reduce es multiplicar 2x2 = 4, después toma el 4 como primer valor y lo multiplica por el siguiente 2, entonces 4x2 = 8, después el 8 pasa como primer valor y se multiplica por el siguiente 2, quedando 8x2 = 16 y después igual que antes, el 16 pasa como primer valor y se multiplica por el siguiente 2, quedando finalmente 16x2 = 32.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Para acceder a la función reduce se debe importar de la siguiente manera.

En reduce, en la primera iteración a y b son los dos primeros elementos de la lista, en este caso sería 2 y 2, pero en la segunda iteración el resultado de la primera iteración que sería 2x2=4 pasaría a ser el valor de a y b pasaría a ser el siguiente valor de la lista, quedando entonces, a = 4 y b = 2 y así sucesivamente…

Texto

Descripción generada automáticamente

1. Try, except, else, finally

Algo que aparece casi al final de la lectura recomendada en el documentación de Python es que se puede agregar un “else” al try-except.

**TRY**: En el try se coloca código que esperamos que pueda lanzar algún error.

**EXCEPT**: En el except se maneja el error, es decir, si ocurre un error dentro del bloque de código del try, se deja de ejecutar el código del try y se ejecuta lo que se haya definido en el Except.

**ELSE**: El else se ejecuta sólo si no hubo ninguna excepción lanzada desde el try

**FINALLY**: Se ejecuta SIEMPRE, haya sido lanzada la excepción o no haya sido lanzada.

Texto

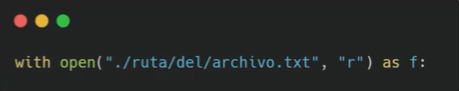
Descripción generada automáticamente

## 9. ¿Cómo trabajar con archivos?

**Modos de Apertura**

* **r** -> Solo lectura
* **r+** -> Lectura y escritura
* **w** -> Solo escritura. Sobre escribe el archivo si existe. Crea el archivo si no existe
* **w+** -> Escritura y lectura. Sobre escribe el archivo si existe. Crea el archivo si no existe
* **a** -> Añadido (agregar contenido al final). Crea el archivo si éste no existe
* **a+** -> Añadido (agregar contenido al final) y lectura. Crea el archivo si éste no existe.

⭐ Si no escriben el modo de apertura, Python lo toma por Default como si fuera **rt**  
.  
Read and Text.  
.  
Además, **f** es el nombre por convención, entonces pues ese ponemos.



Existen varias extensiones de archivos con los que podemos interactuar con python (js,csv,py,css,json,xml)  
Para abrir un archivo seguimos las siguiente estructura

**with** open(<ruta>, <modo\_apertura>) **as** <nombre>

with Es un manejador contextual, nos ayuda a controlar el flujo del archivo (sirve para que el archivo no se dañe cuando existe algún cierre inesperado)

open(ruta,modo\_apertura): es una función que necesita de dos parámetros

* ruta: es donde se encuentra nuestro archivo en nuestro equipo
* modo\_de\_apertura: como vamos a abrir el archivo, los modificadores son:  
  r → modo de lectura  
  w → modo de escritura (sobreescribe el archivo)  
  a → modo append (añade información al final del archivo)

as <nombre> nos ayuda a darle una abreviatura o un nombre a los datos que acabamos de leer