Python 3.5.2 tutorial

Actualizado el jueves 16 de abril de 2020.

Publicado el sábado 16 de julio de 2016.

Introducción.

En una publicación anterior tratamos el tema de generar códigos QR con lenguaje Python y hasta escribimos un sencillo cliente FTP para sincronizar los archivos de una carpeta local versus una carpeta remota. He aquí que entonces que necesitamos un tutorial sobre cómo funciona el lenguaje Python, un lenguaje versátil multiplataforma. Específicamente estaremos tratando la versión 3.5.2 porque consideramos que es el futuro y tiene varios años ya de existencia entre nosotros, dado el caso que valga la pena referirnos a alguna versión 2.X lo especificaremos pero la idea es ser lo más sencillo posible con abundancia de ejemplos y la menor teoría posible.

Historia del lenguaje Python.

A finales de los años 1980 el Doctor Guido Van Rossum creó el lenguaje Python para que sustituyera al lenguaje ABC en el Centro para las Matemáticas y la Informática en los Países Bajos. Su nombre proviene en honor del famoso grupo humorístico "Monty Python's Flying Circus". En un principio lo hizo por pura afición, y tiene su "hégira" en diciembre de 1989, mientras estaba de vacaciones. Hasta el día de hoy continúa trabajando en ello, ya que se convirtió en "benevolente dictador de por vida" (aún sigue al frente del desarrollo y es líder de la comunidad de seguidores) .

- En febrero de 1991 publica la versión 1.0
- En octubre de 2000 publica la versión 2.0 con soporte a caracteres Unicode, entre otros aspectos importantes.
- En diciembre de 2008 sale la versión 3 o Python 3K la cual es incompatible con las versiones anteriores (os advierto no creaís a pie juntillas dicha aseveración, aprendamos y con el tiempo sopesaréis por vosotros mismos).

Diferencias entre la versión 2 y la versión 3.

Actualizado el viernes 26 de mayo de 2017.

Las versiones 3 en adelante vinieron a suplir una serie de necesidades del mundo actual. Si bien del del 2008 al 2016 han transcurrido más de 8 años -y ese tiempo en computación es una *eternidad*- y teniendo en cuenta que el único cambio que nos gusta a los seres humanos es el cambio de pañal, aún hoy en día no se ha podido migrar completamente todas las utilerías

populares en Python. *A grosso modo* existen unas 360 y se han migrado unas 339, así que el progreso es de 94% **pero ¡ojo! ésas son las librerías más usadas** no necesariamente quiere decir que lo que ya estaba escrito migre automáticamente y esto debido a que hay serias incompatibilidades entre las versions 2 y 3.

La estrategia de migración de los desarrolladores de Python ha sido muy acertada: en julio del año 2010 (dos años después de haber lanzado la versión 3.0) emitieron una versión que marca un *punto de inflexión* en las versiones 2.x, **la versión 2.7.0** (al momento de escribir estas líneas vamos por la versión 2.7.13).

El objetivo es que para el año 2020 se dejarán, esta vez definitivamente, de dar soporte y actualizaciones a **todas** las versiones 2.x y en estos tres años que faltan ir dando la oportunidad de que las aplicaciones escritas migren a la versión 3.x -"lo único constante es el cambio" así reza el refrán popular-.

De hecho nosotros aún tenemos instalada la versión 2.7.12 junto con la 3.5.2 (y Debian 8.8 aún trae ambas versiones, la 2.7.9 y la 3.4.2) debido a que colaboramos con algunos proyectos que están en versión 2.7.x: ***nuestra recomendación, de plano, si vais a comenzar un proyecto nuevo en Python es hacerlo de una vez en la última versión.*** *Si por el contrario mantenéis alguna aplicación, pues ir migrando poco a poco* teniendo en cuenta las principales diferencias, por nombrar solo algunas:

- **Print:** en la versión 2.x es un comando y en la versión 3.x es una función, por lo que hay que escribirla entre paréntesis. Una ventaja de la nueva versión es que tiene soporte para caracteres Unicode por lo que ya no es necesario anteceder la letra "u" antes de lo que queremos imprimir para indicarle que lo haga en Unicode. Agregad solamente los paréntesis que aún la letra "u" la sigue soportando y no genera problema (lo probamos con la versión 3.5.2).
- **Otras funciones que ahora necesitan paréntesis:** tales como, sin mencionar todas, **exec**, **raise**, **except**, (pueden necesitar, además de los apréntesis, cambios en la sintaxis).
- **División con enteros:** esto de los tipos de variable es un poco largo de explicar pero fundamentalmente al dividir dos números enteros el resutlado es un número entero, para que conserve el mismo comportamiento en vuestros programas debeis sustituir "***/***" por "***//***". En otros lenguajes para hacer una división que solo devuelva el cociente se utiliza la barra invertida "\" *pero por la fuerte tendencia del lenguaje C en Python, dicho caracter tiene un comportamiento totalmente diferente ya que solo trabaja con caracteres, no con valores numéricos.* Si se desea utilizar dichos operadores "//"" en Python 2 primero debemos declarar lo siguiente: **from __future__ import division**.

La transición de Python 2 a Python 3 es muchísimo más profunda de lo que expresamos aquí pero nuestra intención es colocarla como simple referencia en vuestras mentes, después del año 2020 solo Python 3, *esperemos a ver.*

Actualización viernes 9 de febrero de 2018

En el blog colega "*El Array de Jota"* ayer publicaron un excelente artículo sobre una herramienta llamada "2to3" integrada en Python y como podrán adivinar es hecha de manera *exprofesa* para migrar nuestro código de la versión 2 a la versión3. **Ojo, aunque este proceso es automatizado, primero haced una copia de seguridad fuera de vuestro ordenador de vuestro preciados programas y segundo no confíeis a ciegas, recordad la licencia que rige al software libre: *nosotros somos única y exclusivamente los responsables del código y con lo que el código hace, independitemente de que lo hayamos recibido de otras persona, institución y/o repositorio.***

https://twitter.com/elarraydejota/status/961722592540405762

Instalación de Python en nuestro ordenador.

Instalar Python en nuestro ordenador pasa por ejecutar en una ventana terminal, con los derechos suficientes, los siguientes comandos:

```
sudo apt-get update sudo apt-get install python3
```

- La primera línea actualiza nuestros repositorio de aplicaciones contra el respositorio remoto seleccionado, ya sea Debian o algún GNU/Linux derivado: Ubuntu, Canaima, Linex, etc.
- La segunda línea descarga e instala en sí al Python3. Si queréis tener instalada también la versión 2 simplemente escribidlo sin el número tres. Podéis tener ambas instaladas pero debéis invocarlos (por supuesto) de diferente manera, ¿problema? ni tanto *excepto si queréis ejecutar guiones -scripts- automatizados y allí si que cuenta cómo hayaís programado: para la versión 2.X o 3.X. Ojito pues, con esto.*
- Para instalarlo en otros sistemas operativos, acudan al enlace web donde catalogan las descargas; hay múltiples versiones, escoged la adecuada. Recordad que lo que programemos os servirá en múltiples plataformas con mínimos cambios necesarios debido al hardware, más que todo, es el que hace la diferencia.
- *¿Aún utilizan Microsoft Windows®?* En OpenSourceCom publicaron, en inglés, un muy buen artículo sobre cómo instalar Python en ese popular sistema operativo, ¡recomendamos su lectura si es vuestro caso!

¿Para qué sirve el lenguaje Python?

Pregunta obvia, podemos ponerlo en contexto con unos cuantos ejemplos:

- El lenguaje de marcado **HTML** sirve para realizar páginas web como esta que vosotros estáis leyendo, formato de títulos, párrafos, etc.

- El lenguaje **PHP** sirve para generar páginas web dinámicas, incluso buscando contenido en bases de datos **MySQL**, **PostgreSQL** o **MariaDB**: es el caso de **WordPress**, la herramienta que permite la existencia de este nuestro humilde blog.
- Mientras que los dos anteriores se ejecutan en el servidor web, el lenguaje **JavaScript** se ejecuta en vuestras computadoras, es decir, del lado del cliente, para ciertas funciones locales, como el tipo de navegador web e incluso sistema operativo.
- Otros lenguajes especializados como el **Pascal** se especializan en tratamiento de números en el campo de las matemáticas, por eso tienen su nicho, por ejemplo, en las facultades de ingeniería, donde lo conocimos y aprendimos.
- Existe el lenguaje **C** y sus variantes (**C++**, **C#**, etcétera) los cuales con sus seguidores y detractores (Linus Torvalds ama C y odia C#) siempre se ubican en el desarrollo de sistemas operativos como **GNU/Linux**. El lenguaje C poco le sobra por encima del lenguaje **ASSEMBLER** el cual es utilizado por el firmware de dispositivos primitivos -poca potencia de cálculo- y que no se pueden dar el lujo de programarlos con un lenguaje de alto nivel (el que podemos entender nosotros los humanos). El lenguaje **C** es, por tanto, considerado el padre de los demás lenguajes y marca un hito de separación entre la máquina y los humanos, *pero no quiere decir que sea fácil de aprender, se dice que con el lenguaje C hay que "reinventar la rueda" cada vez que se inicia un proyecto de software.* Para evitar esto pues C++ y C# tienen integradas sus propias librerías y asistentes que facilitan enormemente la tarea, aunque aún sigue siendo una tarea larga y a veces compleja.
- **Pues bien, he aquí que el Python nace con la potencia del lenguaje C pero con múltiples módulos escritos por sus seguidores bajo la filosofía de software libre (**su licencia de uso ha cambiado a lo largo de los años**) los cuales están así al alcance de todos nosotros para realizar numerosas tareas en diversos sistemas operativos, con la sencillez de un archivo de procesos por lotes; pero t*al vez sea esto lo que hace la falsa impresión a sus detractores: subestiman al lenguaje python dada su sencillez.***

Así que vamos a numerar las ventajas que posee el lenguaje Python (de sus desventajas hablaremos en otra entrada, en su oportunidad):

1. Aparte de sus funciones integradas, nativas, **hay una enorme variedad de programas que funcionan como módulos**, esto es posible porque con el tiempo ha alcanzado su masa críticas de usuarios.
2. Es rápido y **en unas cuantas líneas fácilmente podemos delegar pequeñas tareas** en nuestro sistema operativo. Esto es posible porque lo que escribimos se interpreta y al "correr" el programa se compila a lenguaje de máquina, y al modificarlo de nuevo se vuelve a compilar, lo complejo no lo vemos pero está allí en los archivos .pyc o .pyo
3. Su entorno de desarrollo (IDE) **está escrito para varios sistemas operativos**: Unix, GNU/Linux, macOs y otros más.
4. Es multiparadigma, es decir, **no obligan al programador a atarse a un estilo de programación**, *más sin embargo se deben acatar las buenas prácticas de sintaxis que son*

muy sencillas pues principalmente se basan en el indentado ya que existen pocas etiquetas y símbolos, en comparación con otros lenguajes.

5. **Es considerado un lenguaje "de pegamento"** ya que une rápidamente complejas piezas y situaciones en páginas web, bases de datos y "sockets" de internet.
6. **Python es interactivo**: podemos escribir línea por línea y podemos usarlo hasta como una simple calculadora. *Dada esta facilidad nosotros comenzaremos este tutorial de esta manera.*
7. **Es software libre (más no gratuito: lo que desarrollemos debe llevar la licencia de uso heredada para así, a futuro, forme parte del entorno de desarrollo: un ciclo sin fín que cada día fortalece nuestra filosofía de vida).**

El "Zen" de Python.

Por "Zen" se entiende lo que es una forma de meditación (budismo) nacida en la India y perfeccionada y bautizada en Japón con el nombre de "Chan" y que luego se latinizó en Zen. Pues bien el lenguaje Python tiene una base firme en las siguientes líneas que es muy recomendable conocerlas (y analizarlas) antes de siquiera escribir nuestra primera línea de código:

1. Hermoso es mejor que feo.
2. Explícito es mejor que implícito.
3. Simple es mejor que complejo.
4. Complejo es mejor que complicado.
5. Plano es mejor que anidado.
6. Disperso es mejor que denso.
7. La legibilidad cuenta.
8. Los casos especiales no son suficientemente especiales como para romper las reglas.
9. Aunque de hecho está lo pragmático, gana a la pureza.
10. Los errores nunca deberían dejarse pasar silenciosamente.
11. A menos que se silencien explícitamente.
12. Cuando te enfrentes a la ambigüedad, rechaza la tentación de adivinar.
13. Debería haber una — y preferiblemente sólo una — manera obvia de hacerlo.
14. Aunque puede que no sea obvia a primera vista a menos que seas holandés. (NT: Guido van Rossum es holandés)
15. Ahora es mejor que nunca.
16. Aunque muchas veces nunca es mejor que *ahora mismo*.
17. Si la implementación es difícil de explicar, es una mala idea.
18. Si la implementación es sencilla de explicar, puede que sea una buena idea.
19. Los espacios de nombres son una gran idea — ¡tengamos más ideas como esas!

En realidad este Zen es la norma número 20 de las Propuestas de Mejora de Python, mejor conocidas en idioma inglés como "Python Enhancement Proposals" o PEP's. Esta que presentamos es la famosa PEP20 (todas están en este enlace web).

Podemos decir que las PEP son como artículos de una Constitución y cada una de ellas tocan temas tales como, por ejemplo, "Guía de Estilo para el lenguaje C" (Nº 7) y "Guía de Estilo para lenguaje Python" (Nº 8) -tal como dijimos, está Python fuertemente influenciado por el lenguaje C-. Cuando estemos muy avanzados en programación y tengamos alguna influencia en la comunidad desarrolladora de software *tal vez tengamos que leer alguna de ellas*, ya sea para definir el rumbo de un proyecto o incluso para dirimir alguna diferencia con nuestros colegas compañeros de trabajo: *allí están las "leyes" para ello*.

Casos exitosos con lenguaje Python.

Una lista completa la podéis hallar en este enlace, no obstante nombraremos algunos casos que nos llaman la atención:

- **"Industrial Light & Magic"** empresa especializada en gráficos computarizados para estudios de cine, ha creado "efectos especiales" en películas famosas y/o taquilleras.
- **"D-Link"** en Australia utiliza Python para actualizar los "firmwares" en sus enrutadores alámbricos e inalámbricos que no pueden iniciar por sí mismos, lo cual aumentó su productividad en un 800% contra el método anterior utilizado.
- El **laboratorio Astra-Zeneca** usa Python para el trabajo colaborativo en el descubrimiento de nuevos principios químicos activos en los medicamentos.
- Fuera del campo de la Python.org existe un grupo de astrofísicos denominados "Enanas Marrones en la ciudad de Nueva York" (Dwarf Brown in New York City -DBCNY-) que mantienen una sección especial para publicar sus programas hechos en este lenguaje, *¡asombroso!*
- Y son muchos casos más, pero uno que NO nombran allí es el caso de **FreeCAD** que lo utilizamos para diseño de piezas, arquitectura e incluso mecanismos; dicho lenguaje está implementado de manera nativa, si desean revisen nuestro artículo sobre FreeCAD publicado no hace mucho tiempo.

Por último lo hacemos notar: **las nuevas generaciones gustan de aprender el lenguaje Python, para muestra un botón (gracias al sr. "Tauceti" de @Conatel por la foto).**

https://twitter.com/_tauceti/status/755405188484194304

"¡Hola mundo!" con lenguaje python.

Modo interactivo.

Comenzaremos por invocar el intérprete de comandos que nos permitirá ***trabajar de manera interactiva*** con Python. Comenzaremos por abrir una ventana terminal (si usted no sabe cómo, puede estudiar primero nuestro tutorial sobre "bash" o línea de comandos). Debemos hacer la

salvedad de que debemos escribir simplemente el comando "**python3**" para que nos ejecute la última versión que tengamos instalada. De no ser así por favor revise de nuevo arriba donde describimos cómo instalar Python en nuestros ordenadores.

Una vez que introduzcamos el comando y presionar la tecla Intro o Enter veremos el entorno interactivo del que tanto hemos hablado (y dado loas). Apreciamos entonces la versión que tenemos, así como unos comandos adicionales para mayor información. Lo que no nos dice es que para volver a la línea de comandos shell: debemos escribir "quit()" y presionar la tecla Intro o Enter. De ahora en adelante damos por sentado que cuando nos referimos a introducir un comando con o sin opciones adicionales, debemos presionar la tecla Intro o Enter.

Ya aprendimos el comando, **pero para mayor velocidad con el teclado** recomendadmos utilizar la combinación de teclas **CTRL+D** (es decir, con la mano derecha presionamos y mantenemos presionada la tecla control del lado derecho MIENTRAS con la mano izquierda presionamos una sola vez la letra "D" -mayúscula o minúscula, no importa- y luego soltamos ambas). Una vez que estemos de nuevo en la línea de comandos shell podemos verificar de nuevo cual versión de Python tenemos instalado, para lo ello debemos ingresar el siguiente comando, tal cual:

```
python3 --version
```

En el ambiente GNU/Linux se acostumbra "pasar" los opciones de comando con dos guiones más una palabra clave única, en este caso "**--version**", *ésta es la versión larga, la que consideramos nemotécnica (evidentemente en este caso en inglés coincide con nuestro idioma castellano... O CASI ¡porque le falta el acento!).* Para rematar esta idea, que en realidad son dos, acotamos lo siguiente:

- Las opciones "cortas" de los comandos de terminal (fuera del interpréte interactivo) solo llevan un guión y 1, 2 ó 3 letras como máximo ***haciendo hincapié en que se distingue de mayúsculas y minúsculas.*** Por ello, si queremos saber la versión del Python que tenemos con la opción corta debemos escribir "**Python -V**" ya que si utilizamos "**-v**" lo que veremos son todos los módulos que tenemos instalados, y la lista es larga. Probad y volved por favor ;-) .
- La otra idea, ya que hablamos de minúsculas y mayúsculas, debemos tocar el tema de nuestro idioma, en particular la letra "ñ" y los acentos graves (y en francés e italiano los acentos agudos). Ya en una entrada anterior explicamos como lidiar con ellos en el lenguaje HTML y allá es bastante más complicado que acá en Python.

Lo que debemos hacer en nuestros archivos con nuestras líneas de programación para que soporten todos los lenguajes a nivel mundial (UNICODE) es colocar la siguiente línea al principio

de cada fichero:

```
# -*- coding: utf-8 -*-
```

Recordad que os hablamos en párrafos anteriores acerca de las PEP: en este caso es la PEP263 la que especifica y acota el tema. Por supuesto, hay más detalles al respecto, y para ello podemos leer (en inglés) las recomendaciones de un ingeniero de software, el señor Evan Jones en su blog personal. Pero no os preocupéis, por ahora esto que os explico es más que suficiente para expresar correctemente nuestros mensajes a nuestros usuarios con nuestras aplicaciones escritas en Python.

"¡Hola mundo!"

Pues ya estamos casi listos para escribir nuestro primer programa en lenguaje Python. Dijimos que vamos a utilizarlo de manera interactiva para aprender los comandos uno a uno de manera práctica *pero primero escribamos uno y lo guardaremos como archivo de texto pero con la extensión* ***.py***

Se acostumbra nombrar los archivos con esta extensión para nosotros saber rápidamente qué contiene e incluso algunos sistemas operativos lo utilizan para ejecutar una aplicación que es capaz de leer y mostrar dicho archivo. Luego veremos que en GNU/Linux esto no es necesariamente así, esperad por favor.

Vamos entonces a usar nuestro editor de texto favorito y escribamos estas líneas y guardemos el archivo con el nombre "que_hubo_mundo.py":

```
# -*- coding: utf-8 -*- def hola():   print('¡Qué hubo mundo!') hola()
```

- La primera línea declara que vamos a trabajar con UTF8 para soportar caracteres de lenguajes humanos a nivel mundial.
- La segunda línea DEFine una función, la cual no necesita argumento o parámetro seguido de ":" lo cual indica que las líneas que vienen pertenecen a la función declarada.
- *Debemos indentar la tercera línea, con un espacio al menos.* Si hubieran más líneas, deberán tambien tener el mismo número de espacios. **Tened cuidado con vuestro procesador de texto: no es lo mismo pulsar la tecla TABULADOR (TAB) que presionar la barra espaciadora para ingresar espacios en blanco.** Nosotros recomendamos, si el editor de texto lo permite, configurarlo para que cada vez que presionemos TAB nos inserte 3 espacios en blanco (o incluso 2 si os parece).
- Observad también que la tercera línea tiene un acento, ***todo este trabajo que hicimos fue no sólamente para que se mostrara correctamente el caracter: es que Python se***

niega a compilar si no hacemos esto.

- En este punto os tenemos que confesar algo: esto no es necesario para Python versión 3.X. Esto lo hicimos totalmente a propósito porque consideramos que es importante para propósitos a nivel de sistema operativo y es una de las características que vale la pena conservar de la versión 2.X
- Con la cuarta línea llamamos la función declarada, la cual tiene la instrucción de imprimir el mensaje deseado.
- Otra línea que consideramos debe contener todos nuestros archivos en Python es la siguiente especificación:

```
#!/usr/bin/python3
```

Esto le indica al sistema operativo, por si quedare alguna duda, con cuál programa abriremos el archivo en cuestión; recordemos el punto 2 de la PEP20: "Explícito es mejor que implícito".

Nuestro primer archivo quedaría de la siguiente manera:

```
#!/usr/bin/python3 # -*- coding: utf-8 -*- def hola(): print('¡Qué hubo m
undo!') hola()
```

Por último llamamos al archivo con el siguiente comando:

```
python3 que_hubo_mundo.py
```

"¡Qué hubo, mundo!" en modo interactivo.

Ya tenemos todas las intrucciones para trabajar en modo interactivo, podemos ahora ejecutar nuestro primer programa en modo interactivo, aquí veréis lo que debemos escribir, (si tenéis algún error revisad los pasos anteriores que describimos en sumo detalle):

Nota: en la imagen olvidamos escribir "#!usr/bin/python3", *fe de errata*.

Trabajando con cadenas de texto en modo interactivo.

Ya vimos rápidamente como declarar una función para imprimir por pantalla un mensaje e

nuestros usuarios. *Tal vez vamos muy rápido.* Por ello veremos cómo trabajar con texto y caracteres y pasamos a considerar varios puntos:

- Podemos utilizar pares de comillas simples o pares de comillas dobles para encerrar nuestro texto **uno que abre y otro que cierra**, *y si utilizamos ambos pues debemos anidarlos debidamente*.
- En el caso de nuestra habla coloquial (gracias al comediante Joselo que popularizó la contracción de la palabra "para") no podemos escribir una sola comilla simple sin su correspondiente par.
- Es entonces que pasamos a considerar los caracteres especiales: deben ser precedidos por una barra "\", lo cual trae una reminiscencia con el lenguaje C.
- Como vimos el caracter de numeral o almohadilla "#" es utilizado para indicarle ciertas configuraciones especiales a Python, siempre y cuando estén al inicio del fichero contentivo de instrucciones. **Pero también es utilizado para poder colocar comentarios, todo lo que está a la derecha de "#" y hasta el final de la línea es mostrado más no compilado (excepto si están al inicio del archivo, repetimos).**

Con la ayuda de la tecla numeral (como mejor la conocemos aquí en América) os publicamos unos cuantos ejemplos comentados y hechos en la cónsola interactiva de Python3:

Para que no se impriman por pantalla la comilla inicial ni la comilla final haremos uso de la función que usamos en nuestro programa **que_hubo_mundo.py:** la función nativa o integrada **print()**. Decimos que es nativa porque no es necesario definirla ni llamarla, ya Python "sabe" a que nos referimos con ella. Dicha función, además hace un retorno de carro, es decir, el "cursor" baja una línea y se ubica a la parte izquierda de la línea. Por ello, mediante ejemplos en modo interactivo, aprenderemos en detalle el uso de la función **print()**.

¿Véis qué bonita quedó la frase de Don Vicente Ignacio Antonio Ramón de Emparan y Orbe? Fue bonita en 1810 y quedó bonita hoy también gracias a la línea nueva que insertamos dentro de la función **print("\n"),** el cual es un "caracter de escape". Debemos hacer la salvedad de que si queremos mostrarle a nuestros usuarios una ruta de archivo o carpeta *debemos indicarle a print() que imprima en modo raw "r"* para que se muestre correctamente la vía:

```
print(r'\home\jimmy\python3\nombres')
```

A lo anterior le decimos "nueva línea" para que se asocie en nuestra memoria con "\n", pero en realidad se llama retorno de carro, como arriba explicamos. Bien podemos usar tantas nuevas líneas como queramos o bien podemos usar otra figura: **el entrecomillado triple por pares**:

Observemos que para prevenir un retorno de carro en cada línea podemos finalizarla con una barra "\" de esta manera incluso aunque escribamos múltiples líneas se mostrará como una sola (por supuesto si usamos varias barras "\"):

Consideraciones al trabajar en modo interactivo.

Para mostraros los ejemplos didácticos hemos cerrado y abierto varias veces el modo interactivo de cónsola, a fin de hacer claro y diáfano el código, valga la redundancia. Más adelante veremos el cómo limpiar la cónsola, lo cual es un poquito avanzado pero por lo pronto recopilaremos algunos detalles para ahorrarnos trabajo en el teclado.

- Con tecla FLECHA ARRIBA del teclado podemos ver la última línea introducida.
- Si seguimos pulsando flecha arriba podremos ver las líneas anteriores, con la flecha abajo podremos a volver a nuestra posición original hasta alcanzar una línea en blanco, osea la línea nueva de comando.
- El cerrar y abrir no influye para que Python recuerde estas líneas, siempre las "recordará".
- En general se comporta como la línea de comandos o "shell" en ventana terminal de nuestro sistema operativo GNU/Linux.

Operaciones con cadenas de texto en modo interactivo.

Ya conocemos cómo mostrar nuestros resultados por cónsola, ahora vamos a realizar unas cuantas operaciones con cadenas de texto, ¡preparados y preparadas que esto se pone interesante! -o sería la taza de café que me tomé ;-) -.

Lo que vamos a practicar es el colocar una cadena de texto en una variable y haremos operaciones de concatenado con ella:

La técnica de indexado de Python nos permitirá practicar nuestra programación, tal vez el ejemplo es tonto **pero es ilustrativo de las capacidades de Python**, veamos:

- Cuando almacenamos una cadena de texto en una variable, automáticamente Python le asigna una "posición" en memoria para cada una de sus letras (¿otro parecido con el lenguaje C, qué piensan ustedes?).
- Debemos tener siempre presente que el primer caracter siempre será el número cero (0), esto es así para la mayoría de las matrices en computación (una cadena de texto es una matriz de una fila por "ene" columnas, es decir, la cantidad de caracteres que contenga)

Podemos hacer una "chuleta" mental con la palabra "murciélago" para ayudarnos a recordar, ***esto***

NO es código para Python, es para nosotros los humanos con propósitos educativos:

```
 +---+---+---+---+---+---+-
--+---+---+---+  | m | u | r | c | i | é | l | a | g | o
 |  +---+---+---+---+---+---+---+---+---+---+    0   1   2   3   4   5
6   7   8   9  -10  -9  -8  -7  -6  -5  -4  -3  -2  -1
```

Recordando lo anterior podemos volver a trabajar con nuestra frase "nagua" y seremos auxiliados por la estructura de programación "**mientras que**" (**while** *condición* **:**), pongan atención:

Podemos evitar que la palabra quede "en modo vertical" agregando un argumento clave a la función **print()** llamado **end** al cual le podemos asignar una cadena **none** (o nula) o el caracter que deseemos y además nos evita el retorno de carro. La línea cuatro quedaría de la siguiente manera:

```
>>>  print(texto[k], end='') >>>  k = k + 1 >>> print()
```

Pruebenlo en su cónsola interactiva y regresen, por favor. ¿Notaron para qué es la última linea agregada?

Si no incrementaramos la variable auxiliar entraríamos en un ciclo sin fin, lo que comúnmente la gente llama "**quedó colgada la computadora**". De una vez vamos trabajando con números como introducción a la siguiente sección pero antes veamos un ejercicio para separar en sílabas esta esdrújula que nos ocupa:

Nota: Si denotamos texto[:3] , es decir , obviamos desde dónde comienza el trozo que queremos, Python asume que desde el principio; y si denomatos texto[8:] asume que es hasta el final de la cadena de texto. Por ello la línea de práctica de la figura quedaría *escrita de una manera más elegante (punto #7 de la PEP20) y así "no vemos" el elemento 10, que no existe:*

```
>>> print(texto[:3]+texto[3:6]+texto[6:8]+texto[8:]) murciélago >>> # Pod
emos referirnos con índices negativos ... >>> print(texto[-7]+texto[-6]+t
exto[-5]) cié
```

Una cosa es el índice de un elemento y otra cosa muy distinta la función que devuelve un rango de elementos, **de hecho existe una función numérica que nos devuelve un rango específico: range() y la utilizaremos en otra sección importante denominada "listas".** Por ahora podemos volver a nuestro ejemplo con la cadena de texto "nagua" para practicar sobre nuestros propios pasos:

Como vemos la variable completa la podemos sustituir por cualquier otra cadena de texto, *pero toda completa, cada uno de los elementos indexados son de "sólo lectura".* Las listas, en cambio, sí que soportan modificaciones de sus elementos pero se deben declarar de manera diferente, antes de pasar a ello terminemos primero de aprender sobre los números en Python.

Trabajando con números en modo interactivo.

Con Python podemos hacer operaciones matemáticas dignas del quinto grado de educación primaria: sumas, restas, multiplicaciones y divisiones (exactas, inexactas, cociente y resto), mirad el ejemplo práctico, *pensamos no necesita mayores explicaciones que las mínimas necesarias:*

Otra operación numéricas BÁSICAS, es decir, soportadas sin llamar ninguna librería o modulo es la potenciación:

```
>>> 7 ** 2 # Area de un cuadrado de 7 unidades de lado. 49 >>> 2**10  # D
os elevado a la décima potencia 1024
```

Trabajando con listas en modo interactivo.

Las listas se diferencian de los indexados en dos aspectos: como se declaran y que es posible no solo modificar su contenido, sino también agregar o quitar elementos. Otra ventaja es poder formar listas anidadas, pero veamos unos ejemplos prácticos por cónsola:

En esta imagen presentamos un nuevo elemento de programación: **el ciclo for.** Dicho ciclo sustituye al que aprendimos primero, **el ciclo while** que necesita una instrucción o evento para poder salir del mismo. En cambio el ciclo for recorre la lista automáticamente, uno a uno, cada uno de los elementos (recordar que cada lista tiene implícito sus índices numéricos) *esto nos ahorra el uso de una variable numérica auxiliar que usamos en **el ciclo while**.*

Sin embargo, si queremos imprimir solamente los elementos 1 y 2 sí que debemos usar una variable numérica auxiliar con la función range() que avizoramos en la sección de elementos

indexados -cadenas de texto-. Lo que debemos escribir es lo siguiente:

```
>>> for k in range(1,3): # Utilizamos un ciclo 'for' con 'range' ...  pri
nt(ciudades[k], end=",") ...  Maturín,Cumaná,>>> print() >>>
```

Observen el último print() y que labor cumple, pronto lo revisaremos de nuevo. Otro elemento que volvimos a utilizar es el argumento especial **end=","** para la función **print()** la cual omite el retorno de carro e inserta el caracter que le asignemos entre comillas por medio del símbolo de igualdad. En este caso es una coma, así la lista impresa termine en una coma sin ningún otro elemento (esto es así para que tengamos en mente el concepto de los delimitadores: por ejemplo los archivos CSV "Comma Separated Values")

Analizemos este código, mejorado, de la última imagen -captura de pantalla- de la cónsola interactiva de Python:

En seguida aprendemos que podemos agregar elementos a la lista mediante el comando **.append(*elemento*)** tantas veces como necesitemos. También el ciclo **for** recibe los datos de la función **range**: serían 3, 4, 5 y 6 pero *¡un momento!* si le metemos seis como índice a la lista nos dará error, ya que va de solamente va de 0 a 5 *¿y el 6?* Pues el seis si que se ejecuta por medio del comando **else** el cual nos permite imprimir un retorno de carro al final de la lista -de nuevo repetimos que finaliza cada línea impresa sin retorno de carro más coma por el argumento **end** de la función **print()**-.

Para no perder el hilo de aprendizaje seguimos agregando ciudades de Venezuela a la lista, pero con una variante: hemos nombrado solamente capitales de estado (y de municipio, implícitamente) pero ahora vamos a agregar otras ciudades del estado Carabobo, capitales de municipio. El chiste del asunto es que queremos esas tres ciudades (Valencia, Naguanagua y San Diego) "aparte" dentro de la lista de capitales de estado. Aprendamos a hacer eso:

Como pueden ver, una lista puede estar contenida otra lista (si os gusta utilizad el término sublista) y podemos nombrar con un segundo índice (juntos pero separados por paréntesis rectos o corchetes) a la ciudad deseada, pero ¿Qué pasa si un elemento con contiene una sublista? Aquí es cuando muchos programadores se frustran por la respuesta que da Python: muchos esperarían un error (no existe) *sin embargo devuelve el elemento en sí de manera indexada y ya sabemos lo que esto significa, su contenido es de solo lectura, algo contradictorio proque estamos trabajando con listas.* ***Abran su mente a Python y mantenganse siempre alerta y receptivos.***

El ciclo condicional y los comandos de interrupción y continuación.

Ya está en nuestra mente el concepto de **else** para el ciclo **for** y ahora vamos a ver el ciclo condicional. Como era de esperarse, el condicional corresponde a la palabra "**if**" en inglés, seguido de una condición a evaluar, que de ser cierta ejecutara una serie de órdenes perfectamente indentadas y si la condición no se cumple ejecutara otra serie de órdenes, también perfectamente indentadas. Para ello haremos un muy sencillo ejemplo que ilustra lo aquí expresado:

OJO aquí nos volvimos repetitivos y la idea es escribir menos código, no más, si tenemos varios pares de números que comparar no vamos a estar repitiendo y repitiendo la estructura. Para ello usaremos **las funciones** a fondo en una próxima entrada, pero no teman, ya en nuestro programa **que_hubo_mundo.py** definimos una pequeña función, así que ya hemos adelantado (ver inicio de este artículo).

Por lo pronto vamos a escribir un ciclo for asumiendo que conocemos el número de pares de cifras que vamos a comparar y conoceremos una nueva función nativa: **input(),** que como podréis imaginar acepta los número que ingresemos por nuestro teclado y los asigna a una variable. Además, como trabajamos con números necesitaremos la función **int()** que convierte los caracteres ingresados a números enteros. Veamos:

A estas alturas hemos de comprobar que ya se queda corto el modo interactivo para nuestros propósitos: el más mínimo error y hay que comenzar de nuevo la estructura. Por ello a aprtir de la próxima entrada (funciones del usuario, osea, nosotros, ¡EJEM! programadores) volveremos a como empezamos: escribiendo en un editor de texto. Nosotros preferimos gedit pero usad el que más os parezca sencillo y accesible. Como tarea os dejamos un ejemplo tomado de la propia página web tutorial de Python y que sirve para hallar números primos y combina varias cosas que hemos aprendido, NOTAD los espacios indentados, trascribidlo a vuestra cónsola y probadlo, por favor:

```
>>> for n in range(2, 10): ...         for x in range(2, n): ...
          if n % x == 0: ...                print(n, 'equals', x, '*'
, n//x) ...                       break ...        else: ...
#          loop          fell          through          without          f
inding a factor ...          print(n, 'is a prime number') ...
```

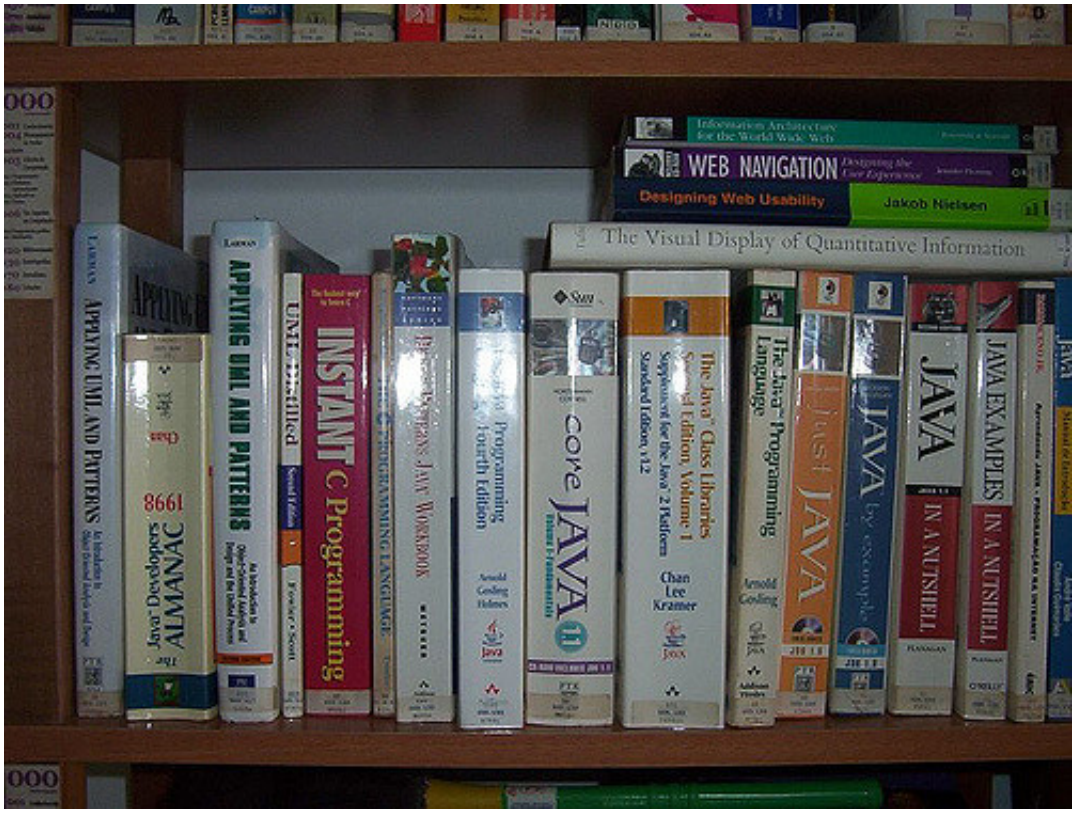
Fuentes consultadas.

Enlaces web en idioma inglés:

- "The Python Tutorial" at Python.org
- "The Python Programming Language" at University of Michigan.
- "The Python Zen: PEP20" by Tim Peters.
- "Python Enhancement Proposals" at Python.org
- "PEP 20 (The Zen of Python) by example" by Hunter Blanks.
- "Hello World Example Program".
- "How to Use UTF-8 with Python" by software engineer Evan Jones.
- "Python 2 vs Python 3: Practical Considerations" by Lisa Tagliaferri at Digital Ocean com.
- "How To Port Python 2 Code to Python 3" by Lisa Tagliaferri at Digital Ocean com.
- «How to package Python applications for Linux»

Enlaces web en idioma castellano:

- Comunidad Python en Venezuela.
- "Las claves para entender el lenguaje de moda: Python" en BBVA.com noticias.
- "El Zen de Python" en MundoGeek.com
- "Programa como un Pythonista: Python Idiomático" por David Goodger.
- "Migrando de 2 a 3 con un ejemplo práctico".



Fuentes consultadas

En idioma castellano

- «».
- «».
- «».

En idioma francés

- «Mémento Python 3» (PDF).
- «».

- «».

En idioma inglés

- «».
- «».
- «».



Crédito de la imagen Gerd Altmann, trabajo, licencia de uso:Pixabay