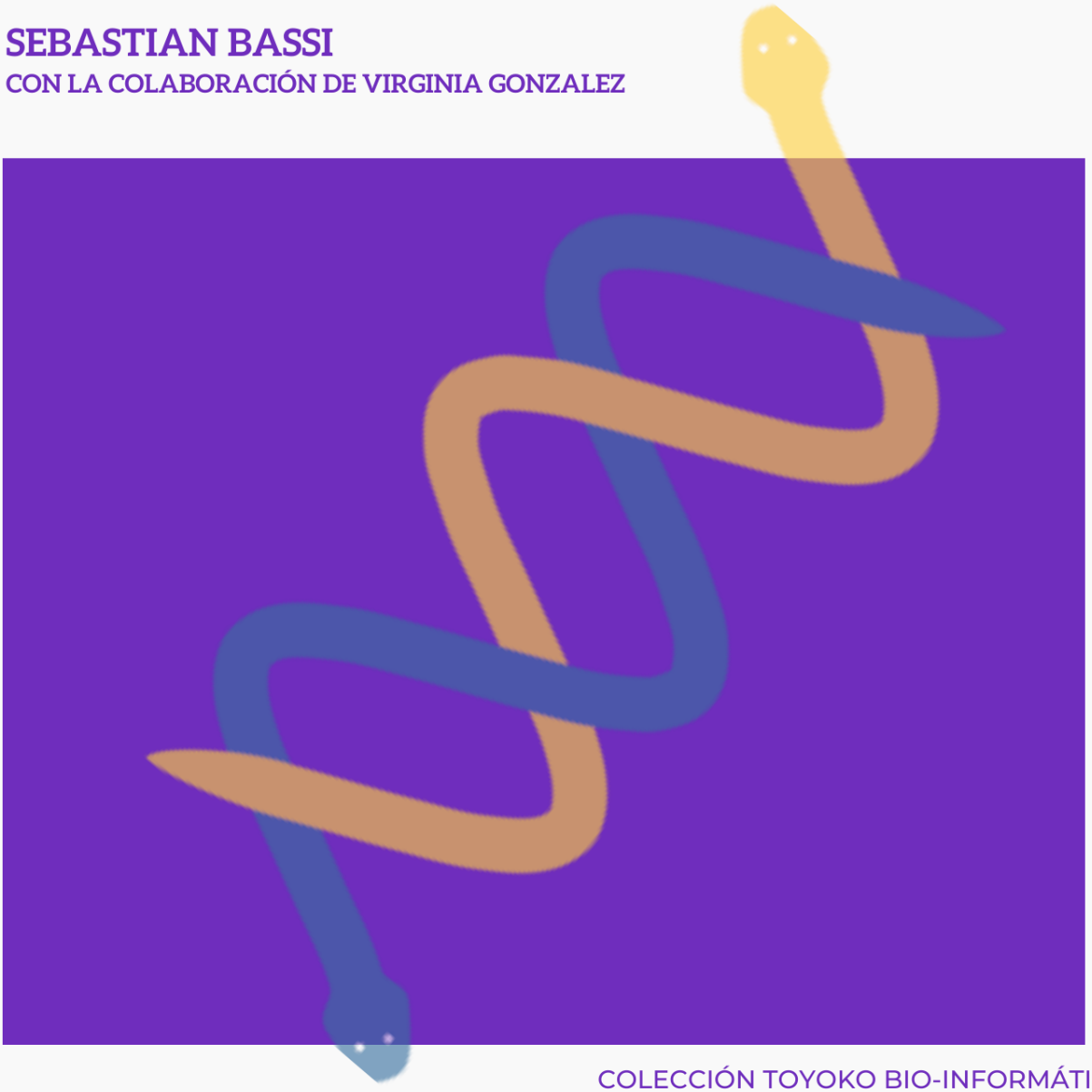
Toyoko

Python para Bioinformática

SEBASTIAN BASSI
CON LA COLABORACIÓN DE VIRGINIA GONZALEZ



COLECCIÓN TOYOKO BIO-INFORMÁTICA

Python para Bioinformática

Sebastian Bassi

Este libro está a la venta en http://leanpub.com/pythonparabioinformatica

Esta versión se publicó en 2021-09-15

Para Virginia Gonzalez

Índice general

Prefacio a la primera edición en Español

La primera edición de Python para bioinformática fue escrita en 2008 y publicada en 2009. Incluso luego de 8 años, las lecciones de ese libro siguen teniendo valor. Esto es un logro en un campo que evoluciona tan rápidamente. Pese a su utilidad, el libro evidencia su edad y se beneficiaría de una segunda edición.

La versión predominante de Python es la 3.6, aunque la 2.7 aún se usa en producción. Como hay mucha incompatibilidad entre estas versiones, se hizo mucho esfuerzo para que todo el código de este libro sea compatible con Python 3.

No solo cambió el software en estos 8 años, pero tanto la actitud de las empresas como el soporte del software libre en general y Python en particular han cambiado drásticamente. Hay nuevos paradigmas que no pueden ser ignorados como el desarrollo colaborativo y cloud computing.

El desarrollo web es otra area de cambió completamente. Aunque este no es un libro sobre desarrollo web, el capítulo "Aplicaciones web" ahora refleja el uso actual de los procesos que corren permanentemente y los frameworks en lugar de CGI/WSGI y aplicaciones basadas en middleware. En el libro anterior los frameworks estaban nombrados como una nota menor, pero ahora el capítulo entero está basado en un framework (**Bottle**) y el método anterior quedó como una nota al pié histórica.

Con respecto a las bases de datos, las del tipo NoSQL se popularizaron, desde ser un item en una lista a tener su propia sección (MongoDB), y una de las recetas fue cambiada para usar esta base NoSQL.

Las librerias gráficas han han mejorado desde 2009, y hay una gran cantidad de librerias gráficas de gran calidad disponibles para Python.

Otro cambio que se ve reflejado en el libro es el uso de Anacoda y Jupyter Notebooks (con todos los códigos del libro en una notebook en la nube provista por Microsoft Azure.[1]

Con respecto al código fuente, hay un repositorio de GitHub[2] donde pódes bajar todo el código y los archivos de ejemplo usados en el libro. También el texto del libro está disponible, por lo que si queres hacer una correción al libro, o agregar algún contenido relevante, podés enviar un *pull request*. Si no saber como hacerlo, podes abrir un *issue* en GitHub comentando el problema para que alguien lo solucione.

Hay correcciones en todos los capítulos. A veces son por errores reales pero en la mayoría de los casos es por la actualización a Python 3 y por estar al día con las buenas prácticas. Con respecto a nuevas

[1] https://notebooks.azure.com/library/py3.us
[2] https://github.com/ToyokoLabs/Py4Bio

correcciones, como esta versión es open source, quienes encuentren errores podrán reportarlo en el sitio de GitHub mencionado arriba.

Ademas de la evolución del software y cambios de paradigmas, también gané experiencia y cambié mi visión sobre temas pedagogicos. Durante estos años trabajé en un proyecto de secuenciamiento genómico en un consorcio internacional y como desarrollador de software en una empresa que cotiza en bolsa (Globant). En los últimos 5 años trabajé para clientes importantes como Salesforce, National Geographic y PLOS (Public Library of Science).

Otro diferencia entre la versión en original y la versión en español es que esta versión tiene una licencia Creative Commons BY-NC-SA 4.0, esto significa que el libro puede distribuirse gratuitamente mietras se haga sin fines comerciales.

Con respecto al logo de Biopython que está en la tapa, aqui está la licencia de uso:

Biopython is currently released under the "Biopython License Agreement" (given in full below). Unless stated otherwise in individual file headers, all Biopython's files are under the "Biopython License Agreement".

Some files are explicitly dual licensed under your choice of the "Biopython License Agreement" or the "BSD 3-Clause License" (both given in full below). This is with the intention of later offering all of Biopython under this dual licensing approach.

Biopython License Agreement

Permission to use, copy, modify, and distribute this software and its documentation with or without modifications and for any purpose and without fee is hereby granted, provided that any copyright notices appear in all copies and that both those copyright notices and this permission notice appear in supporting documentation, and that the names of the contributors or copyright holders not be used in advertising or publicity pertaining to distribution of the software without specific prior permission.

THE CONTRIBUTORS AND COPYRIGHT HOLDERS OF THIS SOFTWARE DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL THE CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

BSD 3-Clause License

1. Introducción

> *La manera más efectiva de hacer algo es hacerlo.* - Amelia Earhart

1.1 QUIÉN DEBERÍA LEER ESTE LIBRO

Este libro es para el investigador en ciencias de la vida que quiere aprender a programar. No es necesario tener experiencia previa en programación para entender este libro, aunque sin duda ayudaría.

El libro está diseñado para ser de utilidad a audiencias disímiles pero relacionadas: estudiantes, graduados, post-doctorandos e investigadores. Todos ellos pueden beneficiarse de saber programar. Exponiendo a los estudiantes a la programación en etapas tempranas de sus carreras los ayudaremos a incrementar su creatividad y pensamiento lógico, habilidades que pueden, y deben, aplicarse en la investigación científica. Para facilitar el proceso de aprendizaje, todos los temas son introducidos con los mínimos pre-requisitos. Al final de cada capítulo, también hay preguntas que pueden ser utilizadas para realizar una autoevaluación.

Los graduados e investigadores, con necesidades concretas de programación, encontrarán muchos ejemplos reales y una abundante cantidad de material de referencia extremadamente valioso.

1.1.1 Qué debería saber el lector antes de leer este libro

El libro ha sido escrito con las siguientes suposiciones:

- No son necesarios conocimientos previos en programación, pero el lector debe poseer un manejo aceptable en el uso de un procesador de texto y de las tareas básicas en su sistema operativo (SO). Dado que Python es un lenguaje multiplataforma, la mayoría de las instrucciones de este libro servirán para los sistemas operativos más comunes (Windows, macOS y Linux). Cuando hay un comando o un procedimiento que se aplica sólo a un SO particular, se hará la aclaración pertinente.
- Ayuda que los lectores estén trabajando (o al menos planeando trabajar) con herramientas bioinformáticas. Incluso trabajos sencillos hechos manualmente como: utilizar NCBI BLAST para identificar una secuencia, alinear proteínas, buscar primers o estimar un árbol filogenético, serán de utilidad para seguir los ejemplos. Cuanto más familiarizado esté el lector con la bioinformática más preparado estará para aplicar los conceptos aprendidos en este libro.

1.2 CÓMO USAR EL LIBRO

1.2.1 Convenciones tipográficas

Hay algunas convenciones tipográficas que he tratado de utilizar de manera uniforme a lo largo del libro. Estas ayudan con la legibilidad y fueron elegidas para poder diferenciar entre nombres dados por los usuarios (o variables) de las palabras propias del lenguaje. Esto es útil cuando estamos aprendiendo un nuevo lenguaje de programación.

Negrita: Para los objetos provistos por los módulos de Python o de terceros. Con esta notación debería ser claro que **round** es parte del lenguaje y no del nombre definido por el usuario. Las negritas también son usadas para destacar las partes de un texto. **No hay manera** de confundir un uso de negrita con el otro.

`Fuente monoespaciada`: Las variables declaradas por el usuario, código y nombres de archivo. Por ejemplo: `sequence = 'MRVLLVALALLALAASATS'`.

Itálica: En los comandos, es usado para denotar una variable que puede tomar diferentes valores. Por ejemplo: en len(*iterable*), "iterable" puede tomar diferentes valores. Usado en un texto, marca una nueva palabra o concepto. Por ejemplo una estructura de datos fundamental es un *diccionario.*

El contenido de las líneas comienza con $ (signo de pesos) significa que es para ser tipeado en la consola de tu sistema operativo (también llamado símbolo del sistema en Windows o terminal en macOS).

<=: Salto de línea. Algunas líneas son demasiado largas para el espacio disponible en una pantalla impresa. Este símbolo es insertado para mostrar que la próxima línea en la página representa la misma línea sobre la pantalla de la computadora. En el código, el símbolo es <=.

1.2.2 Iconos utilizados en este libro

Con este icono se muestran advertencias importantes para tener en cuenta.

Cuando aparece este icono prestá ateción porque es un tip muy útil.

Al final de cada capítulo encontrarás una serie de preguntas y ejercicios para practicar lo aprendido en el capítulo.

1.2.3 Versiones de Python

La versión actual de Python, en este momento, es 3.9.1 Hay una versión 2.7.12 que aún es mantenida[1] porque hay aplicaciones en producción usando la rama 2.7. Las versiones 3.x y 2.x son muy diferentes entre sí, al punto de ser incompatibles. Python 3 es más eficiente que Python 2 en muchos aspectos. Websites de alto tráfico como Instagram migraron de Python 2.7 a Python 3.6 para ahorrar en consumo de CPU y memoria en hasta un 30%. Este libro usa Python 3.6 en adelante. El único escenario donde podés necesitar Python 2.7, además de mantener código viejo, es cuando no está disponible alguna librería específica para Python 3. En este caso, antes de comenzar un proyecto con Python 2.7, trataremos de buscar una librería alternativa. Por ejemplo, si querés conectarte con una base de datos MySQL y te dicen que uses **MySQLdb**, dado que el paquete no es compatible con Python 3; en lugar de usar Python 2.7, podemos usar **mysqlclient** o **mysql-connector-python**, ambos compatibles con Python 3.

1.2.4 Estilo de código

El código fuente de Python que aparece en este libro es presentado como **listados**. Cada línea de estos listados están numeradas. Estos números, no deben ser tipeados, son solo usados para referenciar las líneas en el texto del libro. No necesitan copiar el código desde el libro, dado que el mismo puede ser bajado desde el repositorio de Github en `https://github.com/ToyokoLabs/Py4Bio`. El código puede escribirse de muchas maneras y aún ser válido para el intérprete de Python. El siguiente código es sintácticamente correcto:

```
def GetAverage(X):
    avG=sum(X)/len(X)
    " Calculate the average "
    return avG
```

La siguiente manera también es correcta:

```
def get_average(items):
    """ Calculate the average
    """
    average = sum(items)/len(items)
    return average
```

El código anterior es una muestra del estilo mas aceptado en Python[2] (o "pitónico"). A lo largo del libro encontrarán que la mayoría del código ha sido formateado como se muestra en el segundo ejemplo. Algunos códigos en el libro no seguirán los estilos de código aceptados por las siguientes razones:

[1] Python 2.7.x tiene como fecha de vencimiento el año 2020. No habrá Python 2.8. Para mas información ver PEP-0373 https://www.python.org/dev/peps/pep-0373/

[2] La guía oficial de estilo de Python es PEP-8 https://www.python.org/dev/peps/pep-0008/ y también podés encontrar una guía de estilo http://docs.python-guide.org/en/latest/writing/style/ más fácil de leer.

- Hay algunas instancias donde la manera más didáctica de mostrar una porción particular de código entra en conflicto con la guía de estilo. En pocas ocasiones, me alejé de la guía de estilo en favor de la claridad.
- Dada las limitaciones del formato libro (especialmente el ancho no redimensionable), algunos nombres fueron acortados y se introdujeron otras derivaciones menores al estilo de codificación.
- Para mostrar que hay más de una manera de escribir el mismo código. El estilo de programación es una guía, y no es obligatoria a nivel del lenguaje, algunos programadores no la siguen. Deberían estar preparados para leer código "malo", dado que tarde o temprano tendrán que leer este tipo de código.

1.2.5 Como aprovechar el libro sin leerlo entero

- Si querés aprender a programar **leé desde el capítulo 1 al capítulo 8**.
- Si sabés Python y solo **queres aprender sobre Biopython**, lee primero el capítulo 9. Este capítulo trata sobre los módulos y funciones de Biopython, luego deberías seguir con los programas de los capítulos del 15 al 23.

1.2.6 Recursos online relacionados con el libro

El sitio web del libro está en https://www.py3.us/es. En este sitio encontras erratas, una mailing list para recibir novedades relacionadas con Python y links a los repositorios del código fuente. El repositorio oficial del código fuente está en GitHub[3]. Desde el sitio pódes mirar online o bajar todo el código usado en el libro. Para bajar todos los scripts, ir "Clone o Download" y presionar el botón verde. Si tenés Git instalado en tu computadora (y sabes como usarlo[4]), cloná el repositorio usando esta dirección: `git@github.com:ToyokoLabs/Py4Bio.git`. Otra alternativa es clickear sobre "Download ZIP". Una vez que tenés el repositorio en tu computadora, hay que ir a la carpeta code, donde hay un conjunto de carpetas, cada uno tiene los scripts relacionados a cada capítulo. Por su parte, script en el libro tiene un nombre que se corresponde con el nombre del archivo. Hay otra carpeta llamada `notebooks`, que contiene una serie de Jupyter notebooks que pueden ser ejecutadas localmente. Para más información de como ejecutar una Jupyter notebook, por favor visitá Jupyter Notebook Beginner Guide[5]. Las Jupyter Notebooks disponibles en el repositorio de Github del libro[6]. Estas notebooks pueden ser descargadas y usadas localmente (si uno tiene Jupyter Notebook instalado) o se pueden usar remotamente desde el navegador en el sitio de Mybinder.org[7], para eso hay que cliquear en el logo que dice "Launch Binder" que se encuentra al final de la página de presentación del repositorio (README.md). El enlace al que apunta este logo es https://mybinder.org/v2/gh/ToyokoLabs/Py4Bio/master?filepath=notebooks. En la página de presentación del repositorio también hay enlaces a las notebooks indididuales de cada cápitulo.

[3] https://github.com/Serulab/Py4Bio
[4] En el sitio https://lab.github.com/ hay recursos sobre como usar Github
[5] https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/execute.html
[6] https://github.com/ToyokoLabs/Py4Bio
[7] https://mybinder.org/

Estas notebooks pueder ser vistas (sin modificar) con NBviewer o ejecutadas usando Google Colab. Lo bueno de usar Jupyter Notebooks con Mybinder.org o con Google Colab, es que no necesitas instalar nada en tu computadora para probar de manera interactiva los ejemplos del libro.

1.3 ¿POR QUÉ APRENDER A PROGRAMAR?

Muchas de las tareas que desarrolla un investigador con su computadora son repetitivas: recolectar información de una página web, convertir archivos de un formato a otro, ejecutar o interpretar cientos de resultados del BLAST, diseñar primers, buscar sitios de restricción, etc. En muchos casos es evidente que son tareas que pueden hacerse en una computadora, con menos esfuerzo de nuestra parte y sin la posibilidad de cometer errores por cansancio o por distracción.

Una consideración importante cuando estés evaluando entre crear o no un programa, es la aparente pérdida de tiempo en la definición y formulación del problema, la implementación del código y luego el debugging (corrección de errores) del mismo. Es incorrecto considerar la definición del problema y su evaluación como un desperdicio de tiempo. Generalmente este es el momento preciso del proceso donde entendemos completamente el problema que afrontamos. Es común que durante el intento de formular un problema, nos demos cuenta que muchas de las asunciones iniciales eran incorrectas. Esto también nos ayuda a detectar cuando es necesario reiniciar el proceso de planeamiento. Cuando esto sucede, es mejor que suceda al inicio del proceso que cuando estamos en una etapa avanzada del proyecto. En estos casos, la planificación del programa representa un ahorro de tiempo. Otra ventaja para tener en cuenta es que el tiempo invertido para crear un programa es compensado por la velocidad con la cual las tareas son realizadas cada vez que ejecutamos dicho programa.

Nos solo se pueden automatizar procesos que hacemos manualmente sino que también seremos capaces de hacer cosas que de otra manera no serían posibles.

Algunas veces no está muy claro si una tarea en particular puede ser realizada por un programa, leer un libro como este (incluyendo los ejemplos) te ayudará a identificar cuáles tareas son factibles de automatizar con software y cuales son convenientes realizar manualmente.

1.4 CONCEPTOS BÁSICOS DE PROGRAMACIÓN

Antes de instalar Python, revisaremos algunos fundamentos de programación. Si tenés alguna experiencia previa en programación, podés saltear esta sección y continuar con el capítulo 2, "Instalando Python." Está sección introduce conceptos básicos tales como *instrucciones*, *tipos de datos*, *variables* y algunos otros relacionados a la terminología utilizada a lo largo del libro.

1.4.1 ¿Qué es un programa?

Típicamente las computadoras saben solo lo que el programador le dice que haga[8]. La forma de decirles que hagan algo es a través de un programa. Un programa es un conjunto ordenado de

[8]Esto puede no ser siempre así debido al progreso de las técnicas de IA como Machine Learning, pero en la mayoría de los casos es cierto.

instrucciones diseñadas para ordenar a la computadora a hacer algo. La palabra "ordenada" está allí porque no es suficiente declarar que hacer, tambíen hay que establecer el orden concreto de las instrucciones[9].

A menudo un programa es caracterizado como una receta. Una receta típica consiste en una lista de ingredientes seguida por instrucciones paso a paso de como preparar un plato. Esta analogía es reflejada en muchos sitios web y tutoriales con la palabra "receta" y "libro de recetas". Un protocolo de laboratorio es otra analogía útil. Un protocolo se define como un "método que describa los objetivos, diseño y metodología para la implementación de un experimento científico".

En el Listado 1.1 pueden ver un protocolo típico, que es seguido, casi diariamente, en muchos laboratorios de biología molecular.

Listado 1.1: Protocolo para la digestión de ADN Lambda.

```
Digestión de ADN Lambda
Materiales
5.0 mcL ADN Lambda (0.1 g/l)
2.5 mcL 10x buffer
16.5 mcL H2O
1.0 mcL EcoRI

Procedimiento:

Incubar los reactivos a 37°C por 1 hr.
Agregar 2.5 mcL de colorante e incubar por 15 min.
Sembrar 20 mcL de la mezcla de digestión en un minigel.
```

Hay al menos dos componentes en un protocolo: materiales (o ingredientes) y procedimiento. Un procedimiento provee órdenes específicas como incubar, agregar, mezclar, cargar y muchas otras. Lo mismo sucede con un programa. Los programadores dan órdenes específicas a las computadoras como imprimir, leer, escribir, sumar, multiplicar, redondear, etc.

Como podemos ver, el procedimiento de un protocolo se correlaciona con las instrucciones del programa, mientras que los materiales en el protocolo, hacen el papel de los datos en un programa. En los protocolos los procedimientos son aplicados a materiales:

Mezclar 2.5 mcL de buffer con 5 mcL de ADN Lambda y 16.5 mcL de H2O, sembrar 20 mcL en un minigel.

En un programa las instrucciones son aplicadas a los datos: Imprimir el texto "Hello", sumar dos números enteros, redondear un número flotante.

Como en un protocolo podemos escribir en diferentes idiomas (como inglés, español o francés), hay diferentes lenguajes para programar una computadora. En ciencia, el inglés es el idioma *de facto*.

[9] Hay lenguajes declarativos que establecen que el programa deberían ejecutar más que descubrir cómo ejecutarlo. La mayoría de los lenguajes de programación (incluído Python) son imperativos en lugar de declarativos.

Por diversas razones no existe algo equivalente en programación. Hay muchos lenguajes, cada uno con sus puntos fuertes y sus puntos débiles. Por razones que tendrán sentido en breve, Python fue el lenguaje elegido para este libro.

Veamos un programa simple en Python:

Listado 1.2: `sample.py`: Programa de ejemplo en Python

```
1 seq_1 = 'Hello,'
2 seq_2 = ' you!'
3 total = seq_1 + seq_2
4 seq_size = len(total)
5 print(seq_size)
```

Nota: El número al comienzo de cada línea es solo de referencia, no hay que tipearlo.

Este pequeño programa puede ser leído como "Nombrá a la cadena `Hello`, como `seq_1`. Nombrá a la cadena `you!` como `seq_2`. En la línea 3 sumá las cadenas llamadas `seq_1` y `seq_2` y llamá al resultado con el nombre `total`. En la línea 4, calculá la longitud de la cadena que llamamos `total` y llamá este valor como `seq_size`. Al llegar a la línea 5, imprimí el valor de `seq_size`."

Este programa imprime el número 11.

Como se ve, hay distintos tipos de información (comúnmente llamado "tipos de datos"o solamente "tipos"). Números (enteros o flotantes), cadenas de texto, y otros tipos de datos son abordados en el Capítulo 3. En la línea 5, `print(seq_size)`, la instrucción es `print` y *seq_size* es el nombre de los datos. Los datos son a menudo representados como variables. Una variable es un nombre que es dado para un valor que puede variar durante la ejecución del programa. Con las variables un programador puede representar un comando genérico como "redondear n" en lugar de "redondear 2.9." De esta forma puede tomar en cuenta un valor no fijado (por lo tanto variable). Cuando el programa es ejecutado, "n" tomaría un valor específico dado que no hay manera de "redondear n." [10]. Esto se expresa asi:

```
nombre_de_la_variable = valor
```

Notá que esto **no es una igualdad** como la conocemos en matemática. En una igualdad los términos pueden ser intercambiados, en programación el término a la derecha (valor) toma el nombre del término que está a la izquierda (`variable_name`). Por ejemplo:

```
seq_1 = 'Hello'
```

Después de esta asignación, la variable seq_1 tomará el valor de "Hello" y podrá ser usada, por ejemplo para imprimir su valor,

```
print(seq_1)
```

Esto se puede traducir como "imprimir el valor llamado `seq_1`". Este comando retornará "Hello" porque ese es el valor de esta variable.

[10] Esto puede ser realizado por asignación de un valor a una variable o por unión de un nombre a una variable. En Python está última es la forma más usada. La diferencia entre "asignar un valor a una variable" y "unir un nombre a un valor" está explicada en detalle en el capítulo 3.

1.5 ¿POR QUÉ PYTHON?

Repasemos algunas de las características de Python que vale la pena resaltar.

1.5.1 Características principales de Python

- **Legibilidad**
  Cuando hablamos de legibilidad, nos referimos tanto al programador original como a cualquier otra persona interesada en entender el código. No es raro que escribamos un código y luego volvamos a él unos meses después para encontrarnos con que no lo podemos entender. A veces Python es llamado como un "lenguaje legible por humanos".
- **Características incorporadas**
  Python viene con las "pilas puestas". Tiene una biblioteca estándar amplia y versátil, que está disponible inmediatamente sin necesidad de descargar paquetes separados. Con Python podés, con pocas líneas de código, leer y escribir archivos XML, JSON, generar y enviar e-mails, extraer archivos de un .zip, abrir el contenido de una dirección de internet (URL) y muchas otras posibilidades que en otros lenguajes requerirían un paquete de un tercero.
- **Disponibilidad de paquetes de terceros con un gran espectro de actividades**
  Visualización de datos[11], gráficos, generación de PDF, análisis bioinformáticos[12], procesamiento de imagenes[13], machine learning[14], desarrollo de juegos, acceso a bases de datos [15] y a otros programas. Estas son solo algunos ejemplos de módulos que pueden ser instalados para extender la funcionalidad de Python.
- **Estructuras de datos de alto nivel**
  Diccionarios, conjuntos (sets), listas, tuplas y otras. Estos son útiles para modelar datos del mundo real. También hay módulos de terceros como NumPy, SciPy y Pandas que proveen estructuras complejas como árboles-kd, matrices, series de tiempo, data-frame, y otros.
- **Multiparadigma**
  Python puede ser usado tanto como un lenguaje procedural clásico o como un lenguaje moderno orientado a objetos (POO). La mayoría de los programadores comienzan a escribir sus códigos de manera procedural y cuando tienen la necesidad, se actualizan para usar POO. Python, a diferencia de otros lenguajes, no te fuerza a usar POO para un pequeño programa.
- **Extensibilidad**
  Python puede ser "extendido" usando otros lenguajes de programación. Por eso existen programas "híbridos" que tienen una parte en Python y otra en C o FORTRAN. Típicamente la mayoría está en Python y las secciones mas demandantes de procesador están en

[11] MatPlotLib http://matplotlib.org/ y Bokeh http://bokeh.pydata.org/en/latest/ son los mas usados.
[12] La biblioteca Biopython http://biopython.org/ permite hacer tus propios aplicaciones bioinformáticas.
[13] Paper de Scikit-image: http://peerj.com/articles/453.
[14] http://scikit-learn.org/stable/
[15] https://wiki.python.org/moin/DatabaseProgramming

otro lenguaje. También se lo puede extender a lenguajes especializados de alto nivel como R y MATLAB.

- **Código abierto**
  Python tiene una licencia liberal en lo que se refiere al uso de su código, ya que permite usarlo y distribuirlo con cualquier propósito (incluido el comercial) de manera gratuita. Otra ventaja del código abierto es que es posible saber exactamente que hace el programa y como lo hace, algo que es muy importante en el contexto de la investigación científica.
- **Multiplataforma**
  Un programa hecho en Python puede correr en cualquier computadora que tenga un interprete de Python. Por eso un programa de Python hecho en Windows 10 puede correr en Linux y macOS sin ser modificado. El interprete de Python está disponible para la mayoría de las plataformas de uso corriente, incluso en sistemas embebidos como Raspberry Pi.
- **Comunidad creciente**
  Hoy en día Python es él lenguaje de preferencia de los científicos e investigadores[16]. Esto se traduce en mas bibliotecas para tus proyectos y mas gente a la que le podés pedir ayuda.

1.5.2 Comparando Python con otros lenguajes.

Estarás preguntándote por qué deberías usar Python y no alguno de los otros lenguajes muy conocidos como Java, PHP o C++. Es una buena pregunta. Un lenguaje de programación puede ser considerado como una herramienta y elegir la mejor herramienta para realizar un trabajo tiene mucho sentido.

Legibilidad

Los programadores no profesionales tienden a evaluar la curva de aprendizaje en relación a la legibilidad del código (ambos aspectos íntimamente relacionados). Un programa simple como "Hello world" en Python sería así:

```
print("Hello world!")
```

Comparado con su equivalente en código Java:

```
Public class Hello
{
  public static void main(String[] args) {
  System.out.printf('Hello world!');
 }
}
```

Veamos otro código de ejemplo, esta vez comparado con el lenguaje C. El siguiente programa lee un archivo (input.txt) y copia su contenido en otro archivo (output.txt):

[16]Sobre este punto recomiendo el siguiente artículo: http://www.nature.com/news/programming-pick-up-python-1.16833

```c
#include <stdio.h>
int main(int argc, char **argv) {
  FILE *in, *out;
  int c;
  in = fopen("input.txt", "r");
  out = fopen("output.txt", "w");
  while ((c = fgetc(in)) != EOF) {
     fputc(c, out);
  }
 fclose(out);
 fclose(in);
}
```

El mismo programa en Python es más corto y más fácil de leer:

```python
with open("input.txt") as ifile:
  with open("output.txt") as ofile:
    ofile.writelines(ifile)
```

Ahora veamos un programa en Perl que calcula el promedio de una serie de números:

```perl
sub avg(@_) {
    $sum += $_ foreach @_;
    return $sum / @_ unless @_ == 0;
    return 0;
}
print avg((1..5))."\n";
```

El programa equivalente en Python es:

```python
def avg(data):
    if len(data)==0:
       return 0
    else:
       return sum(data)/len(data)
print(avg([1,2,3,4,5]))
```

El propósito de este programa en Python podría ser completamente entendido por cualquiera que sepa inglés. Python está diseñado para ser un lenguaje con alta legibilidad. El uso de palabras claves, el uso de espacios para limitar los bloques de código y su lógica interna (identación), contribuyen a este fin. Es posible escribir un código difícil de leer en Python, pero eso requiere de un esfuerzo deliberado por parte del programador para ofuscar el código.

Velocidad

Otro criterio a considerar cuando elegimos un lenguaje de programación es la velocidad de ejecución del código. En los inicios de la programación, la computadoras eran tan lentas que las diferencias entre los lenguajes eran muy significativas. Un programa podía tardar una semana en correr en un programa interpretado, mientras que si se ejecutaba en un lenguaje compilado, tardaba menos de un día. La diferencia, propocionalmente se mantiene, pero es menos relevante, ya que un programa compilado que tarda un segundo en correr, su versión no compilada puede tardar 7 a 10 segundos. La diferencia es de un orden de magnitud, no es mucho si tenemos en cuenta que el programa compilado puede tardar 10 veces mas en programarse, y el tiempo del programador suele valer mas que el tiempo de ejecución. Si la tarea es de alta performance, como calculos complejos en tiempo real, puede que esto sea realmente una limitación. En muchos casos se pueden hacer mejoras escribiendo código optimizado. Si el código se escribe con optimizaciones desde el principio, se pueden obtener resultados muy similares al de un código compilado. En los casos donde el programador no esté satisfecho con la velocidad obtenida con Python, puede vincular su programa con una biblioteca escrita en otro lenguaje (como C o Fortran). De esta manera se obtiene lo mejor de 2 mundos: La facilidad de programación en Python con la velocidad de un lenguaje compilado.

1.5.3 ¿Para que se lo usa?

Python tiene un gran rango de aplicaciones. Desde teléfonos celulares a servidores web, hay miles de aplicaciones funcionando con Python en los mas diversos campos. Hay código Python haciendo andar robots en Telegram, produciendo efectos especiales en Industrial Light & Magic, en modems y en software de oficina. Algunos lenguajes son fuertes en un nicho particular, pero Python es mas generalista. Con una base de código común, se pueden hacer aplicaciones de escritorio que se vean nativas en los distintos sistemas operativos. Algunos ejemplos de esta categoría son **BitTorrent** (cliente p2p), **Calibre** (Gestor de e-books), **Sage Math** (sistema de matemática), el cliente de **Dropbox** y otros. Como lenguaje para hacer aplicaciones web, Python está detras de sitios de alto tráfico como Reddit, NationalGeographic, Instagram, Pinterest y NASA. Hay software especializado para hacer sitios web en Python (los llamados web frameworks) como Django, Web2Py, Flask, Bottle y otros. Desde administración de sistemas hasta analisis de datos, Python proveé una variada gama de herramientas:

- Servicios para sistemas operativos incluyendo acceso a archivos
- Compresión de datos
- Comunicación entre procesos y redes
- Funciones de procesamiento de cadenas de texto

Python está siendo utilizado como el lenguaje predeterminado para la comunidad científica. Hay varias bibliotecas orientadas a usuarios de ciencia como **SciPy**[17], **SciKit-Learn**[18] y **Anaconda**[19].

[17] https://www.scipy.org
[18] https://scikit-learn.org/
[19] https://www.anaconda.com/distribution/

Estas bibliotecas integran módulos para algebra linear, procesamiento de señales, optimización, estadística, algoritmos genéticos, interpolación, etc. Con el impulso dado por el crecimiento de "data science" e inteligencia artificial, Python se ha popularizado mucho en este campo. Python está siendo usado cada vez mas en ámbitos de ingenieria, electrónica, astronomía, biología, paleomagnetismo, geografía y muchas otras areas.

1.5.4 ¿Quién usa Python?

Python es usado por muchas empresas, desde las mas pequeñas hasta los líderes de sus industrias como Google, National Geographic, Disney, NASA, NYSE y otras. Es uno de los 4 lenguajes oficiales de Google, junto a Java, C++ y Go. Tienen sitios hechos con Python, programas individuales y hasta soluciones de alojamiento de páginas web funcionando sobre Python[20]. Incluso Microsoft, una empresa que no solía apoyar el software libre, desarrolló una versión de Python para correr en su plataforma ".Net" IronPython[21] y también "Python Tools for Visual Studio", un plugin que convierte Visual Studio en un entorno de desarrollo (IDE) de Python. Muchas distribuciones conocidas de Linux ya usan Python entre sus herramientas, por ejemplo Ubuntu "prefiere que la comunidad contribuya usando Python". Python está tan integrado en Linux que algunas distribuiciones dejan de andar si borramos el interprete de Python.

1.5.5 Variaciones de Python

Aunque en este libro me refiero a Python como un lenguaje de programación, en realidad Python es la definición de un lenguaje. Lo que nosotros usamos es una implementación de dicha definición, que se llama CPython. Esto es, la definición del lenguaje Python implementada en C. Como esta implementación es la más usada, la llamamos Python en lugar de CPython. Las implementaciones mas relevantes de Python son: CPython, PyPy, Stackless, Jython y IronPython. Este libro va a tratar solamente de CPython (a la que llamaremos simplemente Python), pero vale la pena conocer un poco de que se tratan:

- **CPython**
  La versión de Python más usada, a tal punto que en la práctica los términos Cpython y Python se usan como sinónimos. Está hecha casi toda en C (algunos módulos están hechos en Python) y es la versión por defecto disponible desde la página oficial del lenguaje[22]
- **PyPy**
  Una versión de Python hecha en Python. Se hizo para permitir a los programadores Python experimentar con el lenguaje. Ver Pypy.org[23].
- **Stackless**
  Una versión de Python más escalable que la original (CPython) porque no usa el "C stack" que ahora le impide al intérprete por defecto lograr ciertas cosas, como usar micro hilos

[20] https://cloud.google.com/appengine/
[21] http://ironpython.net/
[22] https://www.python.org/
[23] https://pypy.org/

para correr múltiples instrucciones simultáneamente. Se lo ha usado para juegos masivos multijugadores y para ciertas tareas científicas. Stackless Wiki[24].

- **Jython**
  Una versión de Python hecha en Java. Funciona en la Máquina Virtual de Java (JVM por sus siglas en inglés). Se la suele usar para agregar funcionalidad de Python a programas hechos en Java, como por ejemplo el ambiente de desarrollo 3D de aprendizaje Alice. La última version es compatible con Python 2.7. Jython home page[25]
- **IronPython y PythonNet**
  Versión de Python adaptada por Microsoft para funcionar en las plataformas ".Net" y ".Mono". Como el proyecto no está mas mantenido, algunos usan PythonNet[26] que es una alternativa mas actualizada.

1.5.6 Distribuciones de Python especializadas

Aparte de las implementaciones de Python, hay algunas adaptaciones especiales del CPython original que son empaquetadas para propósitos específicos, son las llamadas distribuciones de Python. La mayoría incluye software de terceros tales como editores, módulos de visualización y Jupyter Notebook. Está última es una aplicación web que te permite crear y compartir documentos que contienen código interactivo, ecuaciones, visualizaciones y textos explicativos. Esta es una lista de las distribuciones más útiles[27].

- **ActivePython**
  Para usuarios corporativos, ActiveState[28] provee una distribución de Python precompilada, soportada y con calidad asegurada. Desde un punto de vista técnico esta distribución ofrece todas las versiones modernas de Python con la mayoría de los módulos externos preinstalados. También tiene su propio manejo de paquetes y repositorio de módulos externos (PyPM). Es comercial pero tiene una versión gratuita.
- **Enthought Canopy**
  Otra de las soluciones todo en uno de Python. Incluye más de 450 paquetes de análisis científico como Numpy, SciPy, Python, visualizadores 2D y 3D, adaptadores de bases de datos y otros. También incluye un editor de código con soporte para Jupyter Notebook. Todo está disponible para instalar con un simple clic en los tres sistemas operativos principales. Esta distribución es recomendada para usuarios científicos, está hecha por los mismos que hicieron NumPy y SciPy. Hay diferentes licencias, una libre para la academia y varias pagas para uso comercial. Ver Enthought.com[29].
- **WinPython**
  Se define a si misma como una distribución open source portable del lenguaje de programación Python para Windows 7/8/10, para uso científico y educativo. También

[24] https://github.com/stackless-dev/stackless/wiki
[25] http://www.jython.org/
[26] https://pythonnet.github.io/
[27] Para una lista completa de implementaciones y distribuciones ver https://www.python.org/download/alternatives/
[28] https://www.activestate.com/activepython
[29] https://www.enthought.com/product/canopy/

incluye paquetes útiles para científicos, científicos de datos y educación (NumPy, SciPy, Sympy, Matplotlib, Pandas, pyqtgraph, etc.) Usa Spyder (Scientific PYthon Development Environment) como editor de código por defecto y es portable en el sentido de que el usuario puede mover el directorio WinPython y todos los parámetros son mantenidos. Podes tener copias múltiples de instalaciones aisladas y autoconsistentes de WinPython. Se puede bajar desde su página de Github[30]

- **Anaconda**
  Una distribución de Python y R para computación científica. Incluye más de 720 paquetes para preparación de datos, análisis de datos, visualización de datos, machine learning y ciencia de datos interactivos. El usuario al que apuntan es el mismo tipo al que apunta Enthought Canopy. También incluye Spyder como el editor de código por defecto. Anaconda tiene muchos productos que la diferencian de otras distribuciones de Python, como Repository, Accelerate, Scale, Mosaic, Notebooks y Fusion. La mayoría de estos servicios están disponibles con suscripciones muy caras. Si no usas ninguno de estos servicios aún tenes una excelente distribución de Python del tipo todo en uno. Anaconda[31], anteriormente llamada Continuum Analytics, es un socio institucional del proyecto Jupyter, lo cual significa que ellos soportan el desarrollo de Jupyter Notebook, la aplicación web para correr código Python en un navegador que fue comentada anteriormente.

Podes estar preguntandote cual de estas usar (o solo usar el Python como viene por defecto). No hay una respuesta simple o correcta a esta pregunta, dado que esto dependerá de tus necesidades, hábitos de trabajo, presupuesto y preferencias personales. Personalmente yo tiendo a usar Python, como viene por defecto, en los servidores y Anaconda en las computadoras que uso para desarrollar software.

1.6 RECURSOS ADICIONALES

- Interactive notebooks[32]: Sharing the code. Interactive notebooks: Sharing the code. The free IPython notebook makes data analysis easier to record, understand and reproduce. Helen Shen. Nature 515, 151–152 (06 November 2014) doi:10.1038/515151a
- Python for feature film[33]
- Alternative Python implementations[34]
- IPython[35]: an interactive computing environment.
- bpython[36]: A fancy interface to the Python interpreter for Unix-like operating systems.
- Python history[37], a blog by Guido van Rossum.

[30] http://winpython.github.io/
[31] https://anaconda.org/
[32] https://goo.gl/HfBJ12
[33] http://dgovil.com/blog/2016/11/30/python-for-feature-film/
[34] https://www.python.org/download/alternatives/
[35] http://ipython.org/
[36] https://www.bpython-interpreter.org
[37] http://python-history.blogspot.com