Python en Ámbitos Científicos

Facundo Batista & Manuel Carlevaro

7 de noviembre de 2024

VERSIÓN PRELIMINAR

Título: Python en Ámbitos Científicos
Autores: Facundo Batista & Manuel Carlevaro
ISBN-13 (versión electrónica): ???-?-????-????-?

Primera Edición (versión preliminar)
Escrito con XeLaTeX.

Licencia: Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional (CC BY-NC-SA 4.0)
Lugar: Olivos y La Plata, Buenos Aires, Argentina
Año: 2024
Web: http://pyciencia.taniquetil.com.ar/

10 9 8 7 6 5 4 3 2 1

VERSIÓN PRELIMINAR

Prefacio

Cuando la Comunidad Científica notó que las computadoras podían facilitar su trabajo, sus integrantes se convirtieron posiblemente en los adoptantes tempranos más entusiastas de la tecnología. Aunque el uso y la potencia de las computadoras crece continuamente en el ámbito científico, poco se ha avanzado en la formación y desarrollo de habilidades en Desarrollo de Software.

El cálculo científico requiere realizar combinaciones de múltiples tareas de diversa clase. Por ejemplo, es necesario registrar automáticamente datos de un experimento y visualizarlos, realizar cálculos numéricos o simbólicos, ordenar, clasificar, simular, etc. Muchas veces podemos utilizar paquetes de software que realizan esas tareas por nosotros, pero muchas otras ocurre que nadie ha implementado un determinado cómputo en la forma que necesitamos, o simplemente queremos probar ideas nuevas. Cualquiera sea la naturaleza de nuestra actividad en la ciencia o en la tecnología, no resulta infrecuente la necesidad de interactuar con computadoras a través de programas propios.

El uso de Python en aplicaciones científicas ha aumentado sostenidamente en los últimos años, sin embargo es difícil encontrar libros o manuales en castellano de Python que no estén pensados para programadores.

Este libro nace con la idea primaria de acercar Python al mundo científico, en un libro pensado para científiques, a partir de nuestra experiencia en el dictado del curso "Herramientas Computacionales para Científicos"que ofrecemos en la Universidad Nacional de La Plata y la Universidad Tecnológica Nacional, desde 2007. De la misma manera, la elección del castellano como idioma de escritura es un factor crítico, porque aunque sabemos que el inglés es una herramienta fundamental tanto para programar como para hacer ciencia, estamos convencidos que no debería ser una barrera de entrada.

Más allá de las secciones básicas de un libro (índice, bibliografía, etc.), el libro tiene tres grandes partes. La primera habla de Python, tanto de forma introductoria como también sobre otros temas que son fundamentales y algunas bibliotecas importantes. La segunda trata algunas herramientas fundamentales que son base para el trabajo en el resto del libro. Finalmente la tercera parte muestra cómo abordar temas científicos básicos utilizando Python, de forma teórica y práctica.

En todos los casos este libro esquiva la pretensión de ser una referencia absoluta, sino que tiene el propósito de allanar el camino de les científiques para dar los primeros pasos en el lenguaje y solucionar los problemas básicos (pero no por eso menos importantes) de la ciencia y la tecnología.

Olivos y La Plata, Buenos Aires, Argentina,

Facundo Batista & Manuel Carlevaro

VERSIÓN PRELIMINAR

Índice general

Parte I
Python

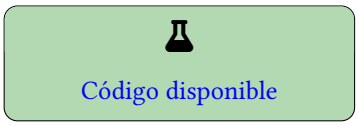
Esta primera parte comprende varios capítulos orientados a proveer la información necesaria de Python para poder entender el resto del libro.

Se sugiere a los lectores que no tienen experiencia previa en el uso de Python, un recorrido secuencial a través de los capítulos que componen esta Parte.

1 | Introducción a Python

En este capítulo mostraremos las bases de Python.

Haremos un recorrido por las distintas características del lenguaje, y luego mostraremos cómo ejecutar programas y usar el intérprete en modo interactivo. Además haremos un repaso de los tipos de datos más usados, los controles de flujo que posee el programa (incluidas las excepciones, para manejo de error), y las distintas formas de encapsular código. Finalmente indicaremos las mejores formas de pedir ayuda.



Este contenido es imprescindible para entender lo suficiente de Python como para poder leer el resto del libro. Por supuesto, no es todo lo que se puede aprender de Python, solamente son las estructuras iniciales que permitirá arrancar con el lenguaje, utilizarlo para los primeros programas, y de allí escalar todo lo que se desee.

1.1. ¿Qué es Python?

Python es un lenguaje de programación de muy alto nivel y multiparadigma.

Es un lenguaje maduro, ya que fue creado en diciembre de 1989 y usado por muchos años en todos los ámbitos posibles (desde electrodomésticos hasta en el espacio). Al mismo tiempo, es un lenguaje que no se quedó estancado, sino que está en constante evolución a través de mejoras permanentes que realiza la comunidad.

Porque es la comunidad la responsable del desarrollo y progreso del lenguaje y sus herramientas relacionadas, así como de llevar adelante conferencias y eventos alrededor del lenguaje. Hay grupos de desarrolladores y usuaries de Python alrededor de todo el mundo, cuyo principal medio de comunicación y coordinación es online.

La comunidad es uno de los puntos fuertes de Python, ya que no sólo ofrece la capacidad técnica de mantener y evolucionar el lenguaje, o administrativa de realizar eventos, sino que provee una estructura social que cobija a les desarrolladores recién llegades lenguaje. Y esto no sucede solamente al principio, sino que es una red que facilita el aprendizaje y el perfeccionamiento a lo largo de toda la vida.

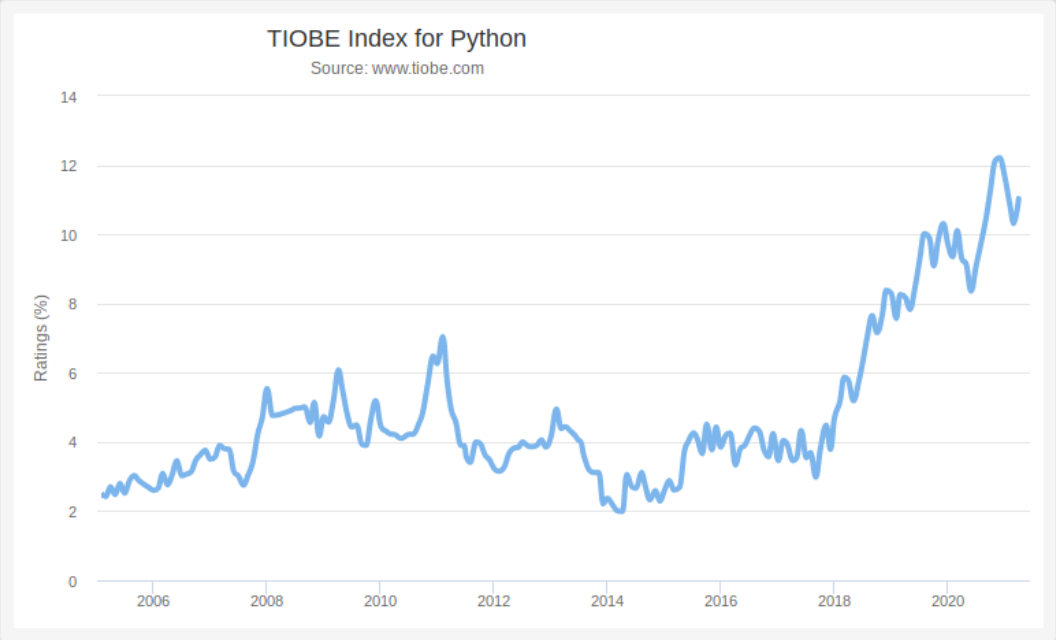
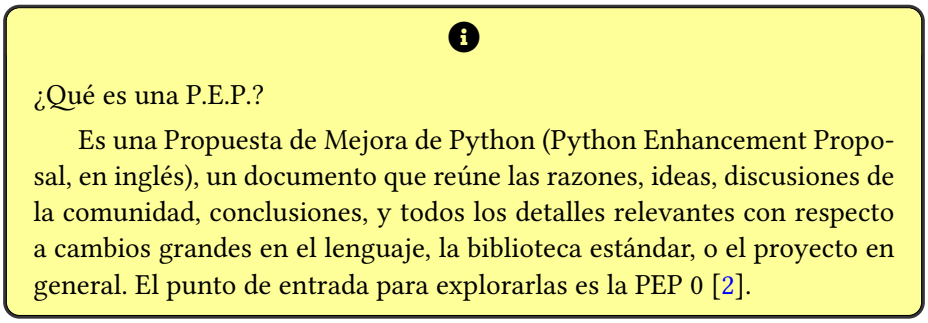


FIGURA 1.1: Evolución de la adopción de Python en el mundo



¿Qué es una P.E.P.?

Es una Propuesta de Mejora de Python (Python Enhancement Proposal, en inglés), un documento que reúne las razones, ideas, discusiones de la comunidad, conclusiones, y todos los detalles relevantes con respecto a cambios grandes en el lenguaje, la biblioteca estándar, o el proyecto en general. El punto de entrada para explorarlas es la PEP 0 [2].

Es la misma comunidad la que garantiza tanto soporte como ayuda, ya que Python no depende de una empresa, y por tanto no corre el riesgo de estancarse por prioridades de esa empresa, conflictos legales por derechos de autor, etc. A este respecto cabe mencionar a la Python Software Foundation [3], una organización sin fines de lucro encargada de proteger el copyright de Python, garantizando su libertad de uso y disponibilidad de forma gratuita.

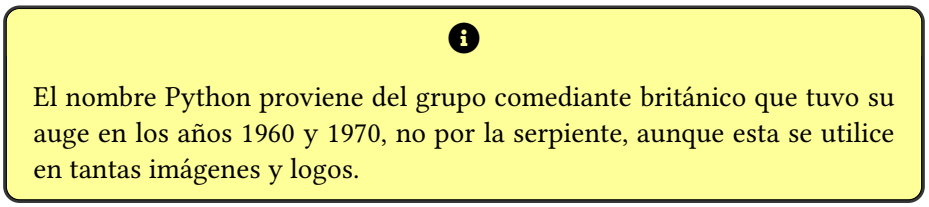
Volviendo a Python como lenguaje propiamente dicho, es muy fácil de aprender ya que por diseño posee una sintaxis muy sencilla. Todo el lenguaje está basado en estructuras conceptuales que se repiten continuemante, bajando muchísimo la curva de adopción del mismo. En parte, es por esto que Python está tan recomendado para enseñar informática, porque todo el impacto pedagógico se puede enfocar en algoritmos y lo que se quiera transmitir, sin tener que aprender primero el lenguaje en sí.

Esto mismo vemos reflejado en el mundo de la Ciencia, donde el uso de Python creció de forma vertiginosa en los últimos años, ya que es un lenguaje que permite enfocarse en el problema a resolver y no es una traba en sí mismo.

Esta simplicidad se transmite en la cultura de Python, en ideas como la de “debería haber una, y preferiblemente solo una, manera obvia de hacerlo”. Esta y otras frases que forman el núcleo

de la filosofía de Python son parte del Zen de Python [4] (que incluimos traducido en el apéndice A), una serie de indicaciones generales para pensar sobre nuestros desarrollos y para enfocar el intercambio de ideas alrededor de un pedazo de código, no reglas duras que haya que cumplir a rajatabla.

El término "pitónico" (o pythónico, o pythonic en inglés) es un neologismo común en la comunidad de Python, que expresa varios conceptos aplicados al estilo del programa. Decir que un código es pitónico es decir que usa correctamente los idiomas de Python, que se adapta a la filosofía minimalista de Python, y hace énfasis en su legibilidad, entre otros aspectos. Por el contrario, si el código es difícil de entender, trabado, o aplica idiomas traídos de otros lenguajes (con estructuras que normalmente se pueden resolver de forma más sencilla en Python), se lo denomina "no pitónico" (*unpythonic*).



El nombre Python proviene del grupo comediante británico que tuvo su auge en los años 1960 y 1970, no por la serpiente, aunque esta se utilice en tantas imágenes y logos.

Por último, y no menos importante, es gratis, libre, y de código abierto. Tiene una licencia muy relajada, por lo cual podemos usar Python de forma segura tanto en ámbitos estatales como privados, sin necesitar consultar con ningún abogado primero :). La única restricción que tiene a este respecto es si quisiéramos distribuir Python nosotros mismos, nada más.

1.1.1. Propiedades del lenguaje

Python es al mismo tiempo de tipado dinámico y de tipado fuerte. Dinámico, porque no hace falta declarar los nombres antes de utilizarlos. Fuerte, porque se respeta el tipo de los objetos y no se realizan conversiones implícitas. Habiendo dicho eso, tenemos que tener en cuenta que las definiciones más estrictas sobre "tipado" son sobre lenguajes que utilizan "variables", y Python no posee variables como tales, sino que son todos objetos y nombres para referenciar esos objetos (más detalle sobre este tema en la sección **??**).

Posee una administración dinámica de la memoria, usando una combinación de conteo de referencias y *garbage collector* ("recolector de basura", en castellano, pero se acostumbra mencionarlo en inglés) para referencias cíclicas. Por lo tanto, aunque hay mecanismos para tener mayor o menor control de lo que sucede en memoria, en general no nos tenemos que preocupar de la misma.

Como mencionamos en la introducción, Python es multiparadigma. Esto implica que aunque en Python todos son objetos, nosotros podemos programar completamente de forma estructurada, e incluso utilizar muchas características y módulos que vienen de la programación funcional u orientada a aspectos. Resaltamos esta versatilidad en su aspecto pedagógico, porque normalmente se aprende programación estructurada primero y el pasaje a la programación orientada a objetos no es para nada trivial, entonces poder utilizar un lenguaje que permite empezar estructurado, e incluir eventualmente la creación de clases propias, mientras estamos expuestos a objetos todo el tiempo, hace que la curva de aprendizaje de la programación orientada a objetos sea gradual.

También en la introducción mencionamos que es un lenguaje de alto nivel. Esto es porque

posee muchas estructuras modernas que permiten expresar la resolución de un algoritmo o la construcción de una solución de manera más cercana a la idea misma que formamos en nuestro cerebro, sin necesitar lidiar con el detalle que se necesitaría al trabajar en un lenguaje más cercano al procesador de la máquina. Resaltamos entre estas estructuras a los conjuntos y diccionarios como tipos de datos integrados, funciones y clases como ciudadanos de primer orden (o sea, son simplemente objetos), módulos y paquetes para estructurar código, iteradores y generadores muy bien integrados al lenguaje, etc. Además posee un manejo moderno de errores a través de excepciones.

Aunque solemos decir simplemente que Python es un lenguaje interpretado, realmente tiene un paso de compilación interno. En detalle, la ejecución de un programa en Python se realiza en dos pasos a partir del código fuente que le pasamos: primero Python compila ese código a una series de instrucciones de la máquina virtual de Python, y luego procede a ejecutar esas instrucciones (que conmunmente denominamos *bytecode*. Esto es similar a otros lenguajes (como Java, VisualBasic, o COBOL, por ejemplo) con la diferencia fundamental que el proceso es automático: no hay intervención humana entre el primer paso de compilación y el segundo de ejecución en la máquina virtual.

Un factor clave para la velocidad de desarrollo en Python es el intérprete interactivo, una herramienta que nos permite ejecutar pequeñas muestras de código sin tener que incluirlas en un programa propiamente dicho, facilitando al mismo tiempo una exploración de los objetos con los que estemos interactuando. Realizando esta exploración y experimentación en el intérprete interactivo, de forma rápida y sencilla, se logra incorporar al código esos algoritmos o líneas de código ya probadas y estables, reduciendo notablemente el ciclo de prueba y error.

Las distintas funcionalidades y herramientas de Python no están integradas todas en su núcleo, sólo algunas forman parte de Python en sí, y el resto están disponibles a través de módulos y paquetes. Una gran cantidad de módulos están disponibles siempre que tengamos Python instalado en el sistema: son aquellos que forman parte de la llamada Biblioteca Estándar. Esta biblioteca es tan variada y extensa que generó la frase "Python viene con las pilas incluidas", ya que gran parte del desarrollo día a día se puede realizar sin echar mano a módulos externos.

Y también es portable, ya que corre en muchas plataformas. Y no estamos hablando solamente de las tres principales (Linux, Windows y MacOS) sino también las derivadas de Unix (como FreeBSD, o Solaris) y algunas más inusuales como Cygwin o AIX.

1.1.2. Biblioteca estándar y módulos externos

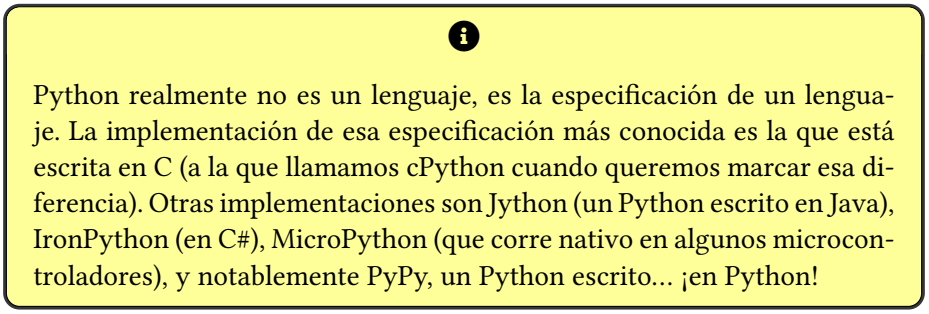
Como decíamos anteriormente, parte de la funcionalidad está integrada en Python como lenguaje, y luego tenemos la biblioteca estándar con gran cantidad de módulos y paquetes. Pero también existen módulos y paquetes de terceros. En esta sección vamos a formalizar esos conceptos, y a echar luz sobre cómo aprovechar estas funcionalidades.

Podemos pensar que tenemos tres zonas. La más interior, donde podemos utilizar directamente lo que allí se encuentra, una intermedia donde para usar la funcionalidad tenemos que importar un módulo, y la más externa, donde tenemos también que importar el módulo pero además debemos instalarlo.

1.1.2.1. Integrada en Python

También denominada *builtin*, es la funcionalidad que está adentro de Python mismo como intérprete del lenguaje. Podemos separarla por un lado en aquella que tenemos a disposición por la sintaxis y semánticas mismas del lenguaje, y por el otro en la gran cantidad de objetos que podemos utilizar directamente.

La sintaxis y semántica del lenguaje en sí se encuentra detalladas en la Referencia del Lenguage [5], una sección de la documentación pensada para entender cómo funciona realmente el lenguaje y sus estructuras internas. Si estamos arrancando con Python no hace falta que nos involucremos con estos documentos ahora, pero es un paso obligado cuando el conocimiento del lenguaje se profundiza.



Python realmente no es un lenguaje, es la especificación de un lenguaje. La implementación de esa especificación más conocida es la que está escrita en C (a la que llamamos cPython cuando queremos marcar esa diferencia). Otras implementaciones son Jython (un Python escrito en Java), IronPython (en C#), MicroPython (que corre nativo en algunos microcontroladores), y notablemente PyPy, un Python escrito... ¡en Python!

Un ejemplo de funcionalidad parte del lenguaje mismo es la capacidad de poder llamar a una función con una cantidad variable de argumentos.

Por otro lado, los objetos que tenemos disponibles para utilizar directamente por estar integrados en el lenguaje están documentados al principio de la Referencia de la Biblioteca Estándar [6], agrupados en funciones, algunas constantes, tipos de datos y excepciones.

Como ejemplos de esto último podemos mencionar la función `len`, que nos permite saber el largo de una estructura, o el tipo de dato `int`, para representar enteros.

1.1.2.2. Parte de la Biblioteca Estándar

El resto de la Referencia de la Biblioteca Estándar [6] versa sobre aquella funcionalidad externa al intérprete de Python en sí mismo, pero disponible en cualquier instalación de Python.

Está agrupada en distintos módulos, y para acceder a ella debemos importar esos módulos (más detalle sobre módulos y paquetes en **??**).

Por ejemplo, para calcular el factorial de un número, podemos usar la función `factorial` del módulo `math`, que es el que debemos importar primero.

1.1.2.3. El resto del mundo

Si queremos funcionalidad de terceros, que no está integrada a Python o en la biblioteca estándar, siempre podemos instalar módulos disponibles en Internet.

La forma más sencilla y directa de hacer esto es utilizar el programa `pip`, que automáticamente descarga lo que indiquemos del Índice de Paquetes de Python (en inglés Python Package Index, al que normalmente conocemos como PyPI) y luego lo instala en nuestro sistema o algún entorno en particular.

Hay una convención para nombrar PyPI y separarlo de PyPy (el Python hecho en Python): PyPI lo nombramos "pai-pi-ai", y a PyPy "pai-pai".

Por supuesto, también podemos utilizar el administrador de paquetes de nuestro sistema (como `apt` en Debian/Ubuntu), utilizar pip para instalar paquetes provenientes de otros repositorios (como Github) o directamente podemos descargar los paquetes necesarios a mano e instalarlos.

A lo largo del libro iremos utilizando varios paquetes que se necesitan instalar de esta forma, los cuales estarán detallados al principio de cada capítulo. En función de eso recomandomos repasar el capítulo donde mostramos más en detalle las distintas alternativas para instalar paquetes **??**.

1.2. Editando, corriendo, e interpretando

En esta sección veremos cómo hacer nuestro primer programa en Python. Algo sencillo, pero que nos permitirá explorar distintas formas de editarlo y ejecutarlo, y otros conceptos aledaños.

1.2.1. Editando y ejecutando Python

Mencionamos arriba que podemos considerar que Python es interpretado, pero realmente es compilado. A fines prácticos, especialmente al arrancar, podemos considerar efectivamente que es interpretado: nosotros le pasamos el código fuente, y Python *lo ejecuta*.

¡Probemos eso! Agarremos cualquier editor (más adelante profundizamos ahí) y escribamos un programita muy simple, y lo grabamos en un archivo `hola.py`:

```
1 print('¡Hola mundo!')
```

Luego, en una terminal ejecutamos nuestro programa…

```
$ python3 hola.py
¡Hola mundo!
```

Es así de simple. Grabamos el archivo, lo ejecutamos, tenemos el resultado. Si queremos realizar modificaciones al archivo, volvemos a grabar, volvemos a ejecutar. Nada más. Este "ciclo corto" hace que iterar sobre la construcción de un programa sea muy eficiente en Python.

Por otro lado, ¿qué es eso de ir a la terminal para ejecutar un programa? Bueno, tenemos principalmente dos tipos de programas, con y sin interfaz gráfica. El que mostramos arriba no posee una interfaz gráfica, entonces lo usamos desde la terminal.

Si nuestro programa fuese de interfaz gráfica, podríamos ir y hacerle doble-click al programa y que se ejecute. En realidad también lo podemos hacer en un programa sin interfaz gráfica, pero todo lo que veríamos es una terminal que nuestro sistema abriría para ejecutar el programa, que se cerraría cuando este termine (y si el programa termina rápidamente, como el ejemplo de arriba, veríamos sólo un parpadeo).

Más adelante nos encargaremos de hacer programas con interfaz gráfica (ver Capítulo **??**), pero por ahora enfoquémosnos en saber al menos cómo ejecutar cualquier tipo de programa: desde la terminal.

Obviamente para ejecutar nuestro programa con Python necesitamos a Python instalado en nuestro sistema. Lo vamos a encontrar ya instalado en cualquier Linux o MacOS, pero vamos a necesitar instalarlo en Windows. En cualquier caso, todas las versiones para todos los sistemas están en la página oficial de descargas [7].

Como mostramos arriba, siempre podemos pasarle nuestro programa al intérprete de Python para que lo ejecute. Si queremos poder ejecutarlo directamente, quizás tengamos que realizar una acción extra, dependiendo de nuestro sistema. En Windows, el instalador mismo de Python asocia la extensión `.py` al intérprete de Python, entonces no necesitamos hacer nada.

En Linux y MacOS, por otro lado, tenemos que hacer que nuestro programa sea ejecutable e indicar en el mismo programa que debe ejecutarse con Python. Veamos esto en detalle. Primero modificamos el programa, agregando una linea muy particular al principio y luego una linea en blanco:

```
1 #!/usr/bin/python3
2
3 print('¡Hola mundo!')
```

Esa linea particular del principio tiene una estructura muy específica: arranca con estos dos caracteres `#!` (llamados *shebang*) que indican que a continuación está el programa que va a interpretar las líneas del archivo, `python3` en nuestro caso.

Luego hacemos ejecutable al programa, y lo corremos directamente:

```
$ chmod +x hola.py
$ ./hola.py
¡Hola mundo!
```

1.2.2. Editando Python

Python tiene una sintaxis tan sencilla y limpia que no se necesita demasiada ayuda del editor que estemos usando para escribir código.

Entonces, cualquier editor de texto que sea útil para escribir programas (NO Microsoft Word, por ejemplo) nos es suficiente. Ejemplos de este tipo son Vim, Emacs, Kate, Textmate o Notepad++.

Habiendo dicho eso, igualmente mucha gente prefiere desarrollar utilizando Entornos de Desarrollo Integrados (en inglés, *Integrated Development Environment*, lo que forma la sigla "IDE" que es la que se usa normalmente también en castellano). Esto es porque un IDE integra el editor (simple, como decíamos) con un navegador de archivos, herramientas de debugging, un interprete interactivo, y muchas funcionalidades más. Ejemplos de IDEs son PyCharm, VisualStudio, o Spyder.

Esas funcionalidades extras que proveen los IDEs son tan útiles que incluso se han ido montando herramientas sobre los editores simples para obtener algunas de ellas. Quizás el ejemplo más famoso de eso es la configuración de Fisa para Vim [8].

En cualquier caso, usar un editor simple más bien pelado o un IDE súper completo (o cualquier intermedio entre esos dos extremos) no deja de ser una elección personal. Nuestra recomendación es que las personas nuevas a la programación no se traben demasiado en arrancar con la herramienta ideal, que elijan una más o menos rápido y se pongan a programar, la elección del editor o IDE ideal para esa persona irá decantando con el tiempo.

Para elegir con qué arrancar siempre es bueno revisar la página de Python Argentina dedicada a editores e IDEs [9].

1.2.3. Usando módulos

Un concepto básico que necesitamos para arrancar es cómo utilizar los módulos que Python trae en la Biblioteca Estándar. Porque el programa que hicimos arriba usa la función integrada `print` pero enseguida vamos a tener que empezar a utilizar otras funciones no integradas.

Para usar un módulo primero tenemos que importarlo. La forma más directa de hacer esto es a través de la declaración **`import`**, y luego podremos acceder a los contenidos del módulo a través de la "notación punto" (este es uno de esos conceptos genéricos del lenguaje: siempre que queramos acceder a algo que está adentro de un objeto, podemos usar el punto: `.`.

Veamos un simple ejemplo.

```
1 import math
2
3 print('Raíz cuadrada de dos:', math.sqrt(2))
```

En la linea 1 vemos que importamos el módulo `math` y luego llamamos a la función que nos calcula la raíz cuadrada usando el "punto": `math.sqrt(2)`.

Pueden encontrar más detalles sobre módulos y distintas formas de importarlos y usarlos en ??.

1.2.4. El intérprete interactivo

En realidad no tenemos que escribir todo un programa para probar algo en Python. Uno de las mejores características del lenguaje es que trae incorporado un intérprete interactivo. Llamamos "interactivo" a este modo del intérprete, porque Python compilará y ejecutará cada linea que escribamos, sin necesidad de otros pasos.

Si en la terminal ejecutamos[1] `python3` sin pasarle ningún programa, Python automáticamente abrirá el intérprete interactivo, mostrándonos un *prompt* y esperando que escribamos algo:

```
$ python3
Python 3.8.2 (default, Jul 16 2020, 14:00:26)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

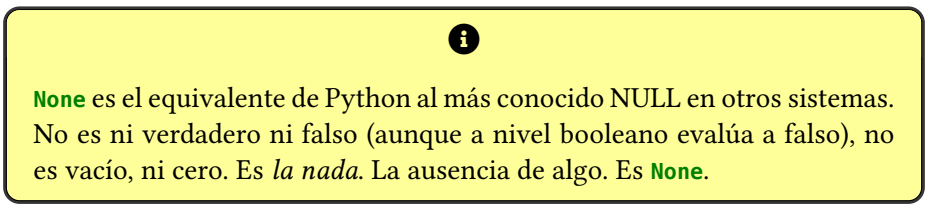
Obviamente las primeras líneas pueden variar en función de la versión de Python que tengamos instalada, pero lo importante es ese >>> que nos habilita a empezar a escribir.

A lo largo de este libro veremos muchísimos ejemplos de código que son realizados directamente en el intérprete interactivo, y justamente nos podemos dar cuenta de eso por el prompt. Por ejemplo, reveamos como importar y usar un módulo, pero sin tener que escribir un programa para ello:

```
1 >>> import math
2 >>> print(math.sqrt(2))
3 1.4142135623730951
```

En la linea 1 del ejemplo tenemos un **`import`** que no devuelve nada, por eso en la linea 2 volvemos a tener directamente el prompt, donde ahora escribimos el `print` que nos mostrará la raiz cuadrada de dos en la linea 3 (que no tiene el prompt, porque no es un linea para que escribamos nosotros algo).

En realidad no es necesario hacer un `print` para ver un resultado en el intérprete interactivo, ya que este luego de ejecutar cada linea nos mostrará abajo el objeto resultante de esa linea, a menos que sea **`None`**.



`None` es el equivalente de Python al más conocido NULL en otros sistemas. No es ni verdadero ni falso (aunque a nivel booleano evalúa a falso), no es vacío, ni cero. Es *la nada*. La ausencia de algo. Es **`None`**.

Esto nos permite usar el intérprete de forma muy sencilla para explorar distintos comportamientos..

1 En Windows, donde el uso de la terminal no está tan extendido, la instalación de Python incluye a IDLE, un entorno integrado y de aprendizaje que muestra un intérprete interactivo.

```
>>> 2 + 3
5
>>> len("hola")
4
```

Hay una diferencia significativa, sin embargo, entre la salida de `print` y lo que nos muestra el intérprete interactivo en cada resultado. Para construir la salida (convertir el objeto resultado a una representación textual a mostrar en la terminal) el `print` utiliza la función integrada `str`, que nos deja la representación más "simple y humana" del objeto, mientras que el intérprete interactivo usa la función integrada `repr`, que apunta a lograr la representación más "exacta" del objeto.

Vemos en el siguiente ejemplo como con el `print` corremos el riesgo de confundir el tipo de dato del resultado (¡parece un número!), mientras que con el `repr` es evidente que es una cadena de texto:

```
>>> print("23")
23
>>> "23"
'23'
```

En realidad el intérprete interactivo procesa la linea que acabamos de escribir cuando esa linea termina, lo que nos lleva a marcar la diferencia entre líneas lógicas y líneas reales: el intérprete interactivo se da cuenta cuando la linea todavía no terminó (aunque hayamos apretado ENTER) porque falta cerrarla lógicamente. Veamos un ejemplo de eso, donde la linea continúa porque falta cerrar el paréntesis del `print`, y veamos como el intérprete nos marca esa "línea continuación" con tres puntos abajo del prompt:

```
>>> print("Hola",
... 123)
Hola 123
```

1.2.5. Jupyter Notebook

El intérprete interactivo que trae Python por default no es el único que existe, sino que tenemos a disposición múltiples alternativas, especializadas para distintos fines.

Quizás el intérprete alternativo más popularizado es IPython [10], un shell interactivo que añade funcionalidades extras al intérprete interactivo incluido en Python, como resaltado de líneas y errores mediante colores, una sintaxis adicional para el shell, etc.

La funcionalidad núcleo de IPython fue evolucionando en el tiempo y se convirtió en una aplicación web llamada Jupyter Notebook. Por su baja barrera de entrada para interactuar con el mismo, soporte para integrar textos y gráficos entre las celdas de ejecución, y facilidad para la distribución del contenido, Jupyter Notebook se ha vuelto el intérprete interactivo de facto en el ámbito científico.

En las próximas secciones utilizaremos casi exclusivamente Jupyter Notebooks para los ejemplos mostrados en el libro (y en todos los casos haremos referencia al notebook real, accesible

en internet, para que puedan descargarlo, modificarlo y jugar con los ejemplos, que es una muy buena manera de aprender).

1.2.6. Explorando

Tanto el intérprete interactivo como Jupyter Notebook nos permite explorar el lenguaje y sus comportamientos. Esto es especialmente evidente si utilizamos las funciones integradas `dir` y `help`, que nos permiten ver los interiores de un objeto y directamente pedir ayuda en la terminal y ver su documentación.

En el siguiente ejemplo vemos como podemos descubrir qué atributos tiene el tipo de dato `list` y pedir ayuda sobre uno de ellos.

```
>>> dir([])
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__',
'__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
'__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__',
'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop',
'remove', 'reverse', 'sort']
>>> help([].count)

Help on built-in function count:

count(value, /) method of builtins.list instance
    Return number of occurrences of value.
```

Siempre que miremos dentro de los distintos objetos vamos a encontrar muchos atributos que empiezan y terminan con doble guión bajo. Estos son métodos especiales que permiten a los objetos integrarse y utilizar la sintaxis y semántica del lenguaje en sí (ya sean estos objetos integrados en Python, creados por nosotros o por terceros. Mientras estemos aprendiendo Python podemos ignorarlos tranquilamente, dejándolos para cuando avancemos con el lenguaje. Habiendo dicho eso, durante el libro mencionaremos y utilizaremos algunos, que explicaremos oportunamente.

Es difícil y cansador decir, por ejemplo para referirse al `__len__`, "doble guión bajo len doble guión bajo", o su equivalente en inglés *double underscore len double underscore*, por eso se creó una forma especial para nombrarlos: usando *dunder* (que es una especie de abreviatura del double underscore en inglés), con lo que para el ejemplo nos quedaría *dunder len*.

1.3. Cómo pedir ayuda

Una de las grandes cosas buenas de Python es su Comunidad, alrededor del mundo y en infinidad de idiomas.

Siempre vamos a encontrar un foro, una lista de correo, o algún recurso gratuito puesto a disposición por gente tratando de ayudar. Y la misma comunidad se autorregula para seguir siendo sana, tanto informalmente como formalmente, por ejemplo creando la Python Software Foundation en Estados Unidos, o la Asociación Civil Python Argentina [11], organizaciones que aprueban y alientan la participación de todes. Nuestra comunidad esta basada en respeto mutuo, tolerancia y fomento, y estamos trabajando para ayudar a cada uno y a cada una a vivir a la altura de estos principios. Queremos que la comunidad sea más diversa [12]: quien quieras que seas, y cualquiera sean tus orígenes, te recibiremos.

Los canales de comunicación virtuales son el principal medio de comunicación e integración de la Comunidad, y es un buen punto de entrada para cuando tenemos consultas y preguntas. Aprender a utilizar estos mecanismos es un activo tan importante en un programador o una desarrolladora de software como saber tal función, tal estructura del lenguaje, o tal detalle de implementación.

Es por esto que queremos hacer mucho énfasis en que busquen como interactuar con la Comunidad, cuales grupos de usuaries tienen cerca, qué lista de correo o foro les apetece más, y empiecen a participar allí. Si ya tienen consultas pueden hacerlas, pero también es muy valioso leer las preguntas de otres y las respuestas que se ofrecen, e incluso tratar de responderlas nosotres. Esta es una de las mejores formas de profundizar nuestro conocimiento en Python.

Te recomendamos explorar el sitio de Python Argentina [13], o entrar directamente en su grupo de Telegram [14] o en el foro [15]. Pero no dejes de de revisar si hay alguna comunidad local más cerca de donde estés [16].

Y siempre está la comunidad global de Python, con muchísimos recursos en su sitio [17], en particular las listas de correo [18] y el foro [19], pero claro, esto es todo en inglés.

Parte II
Herramientas fundamentales

Los capítulos siguientes desarrollan un conjunto de conocimientos sobre herramientas fundamentales que serán aprovechados en la Parte III al abordar temas de aplicaciones específicas.

Parte III
Temas específicos

En esta Parte desarrollaremos capítulos que abordan temas específicos de aplicación de herramientas de Python. Cada capítulo está autocontenido, de forma que el lector o lectora puede acceder directamente al tema de interés sin realizar un recorrido secuencial de los capítulos, aunque serán utilizados conceptos y herramientas tratados en la Parte II.

Parte IV
Apéndices

A | Zen de Python

Incluimos aquí las frases traducidas correspondientes al Zen de Python [4].

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Espaciado es mejor que denso.
- La legibilidad es importante.
- Los casos especiales no son lo suficientemente especiales como para romper las reglas.
- Sin embargo la practicidad le gana a la pureza.
- Los errores nunca deberían pasar silenciosamente.
- A menos que se silencien explícitamente.
- Frente a la ambigüedad, evitar la tentación de adivinar.
- Debería haber una, y preferiblemente solo una, manera obvia de hacerlo.
- A pesar de que esa manera no sea obvia a menos que seas Holandés.
- Ahora es mejor que nunca.
- A pesar de que nunca es muchas veces mejor que *ahora* mismo.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.
- Los espacios de nombres son una gran idea, ¡tengamos más de esos!

Bibliografía

[1] URL: https://creativecommons.org/licenses/by-nc-sa/4.0/deed.es.

[2] python-dev <python-dev at python.org>. *Index of Python Enhancement Proposals (PEPs)*. 13 de jul. de 2000. URL: https://www.python.org/dev/peps/.

[3] URL: https://www.python.org/psf/.

[4] Tim Peters. *The Zen of Python*. 19 de ago. de 2004. URL: https://www.python.org/dev/peps/pep-0020/.

[5] URL: https://docs.python.org/dev/reference/index.html.

[6] URL: https://docs.python.org/dev/library/index.html.

[7] URL: https://www.python.org/downloads/.

[8] URL: http://vim.fisadev.com/.

[9] URL: https://wiki.python.org.ar/ides/.

[10] URL: https://ipython.org/.

[11] URL: https://ac.python.org.ar/.

[12] URL: https://ac.python.org.ar/diversidad/index.html.

[13] URL: https://www.python.org.ar/.

[14] URL: https://t.me/pythonargentina.

[15] URL: https://charlas.python.org.ar/.

[16] URL: https://wiki.python.org/moin/LocalUserGroups.

[17] URL: https://www.python.org/.

[18] URL: https://www.python.org/community/lists/.

[19] URL: https://python-forum.io/.

VERSIÓN PRELIMINAR