AutoPython: Una herramienta para la automatización de sesiones interactivas de Python

Germán Osella Massa, Cecilia De Vito, Claudia Russo, Hugo Ramón
Instituto de Investigación y Transferencia en Tecnología (ITT), Escuela de Tecnología,
Universidad Nacional del Noroeste de la Provincia de Buenos Aires (UNNOBA)
Sarmiento y Newbery (CP 6000), Junín, Buenos Aires, Argentina. Tel: (0236) 4636945/44
{german.osella, cecilia.devito, claudia.russo, hugo.ramon}@itt.unnoba.edu.ar

Resumen

En este trabajo se describe la herramienta AutoPython, ideada y desarrollada para automatizar sesiones interactivas del shell de Python, con el fin de agilizar y potenciar el dictado de clases, presentaciones o charlas técnicas mediante la ejecución "en vivo" de código. Se detalla el uso de la herramienta y se propone una metodología para combinarla con el uso de software específico para presentaciones como PowerPoint, por ejemplo. Finalmente, se presentan algunos resultados obtenidos con el uso de la herramienta y se proponen futuros desarrollos a encarar.

Palabras clave: Python, automatización, presentaciones interactivas, live coding.

Introducción

Python 0 es un lenguaje de programación claro y minimalista, que busca expresar conceptos preferiblemente de una única manera. El uso de la indentación para definir el alcance de los bloques que definen la estructura de un programa fuerza a que se incorpore la buena costumbre de escribir código claro y legible. Si bien es un lenguaje orientado a objetos, lo puede disimular muy bien, dando la apariencia de ser un lenguaje procedural imperativo. Es fuertemente tipado, dinámico e interpretado. Admite correr una sesión interactiva con ayudas tales como la documentación en línea y el completado automático de código. La documentación en línea permite obtener ayuda sobre una función o método, tanto incorporados en el lenguaje así como escrito por el usuario. Cuenta con la facilidad para escribir casos de pruebas dentro de la documentación del código, usando una sintaxis idéntica a la de la sesión interactiva.

Python ha ido aumentando su popularidad año a año y, de acuerdo a los índices informales TIOBE 0 y PYPL 0, a comienzos del 2016 se encuentra ubicado entre los primeros cinco lenguajes de programación más populares en el mundo, de acuerdo con las métricas usadas en cada índice. Por sus características, también ha ganando aceptación en el ámbito académico, utilizado principalmente como lenguaje científico y para la introducción a la programación. Según estadísticas recientes 0, se ha convertido en el lenguaje introductorio más utilizado en las universidades más prestigiosas de Estados Unidos.

Por ser un lenguaje que cuenta con un shell interactivo que permite ser usado como REPL[55], Python es ideal para fomentar la exploración y descubrimiento tanto del lenguaje como de su biblioteca estándar a partir de la ejecución de sentencias o breves fragmentos de código. Tras el ingreso de los mismos en el shell interactivo, se obtiene una respuesta inmediata tanto para mostrar un resultado así como para señalar errores si los hubiera, sin tener que pasar por el tedioso proceso de compilación y ejecución que otros lenguajes exigen, manteniendo además en

55 REPL es la abreviatura de *Read-Eval-Print-Loop*, es decir, la repetición (loop) del ingreso de una sentencia (read) junto con su evaluación (eval) e impresión del resultado devuelto por ésta (print).

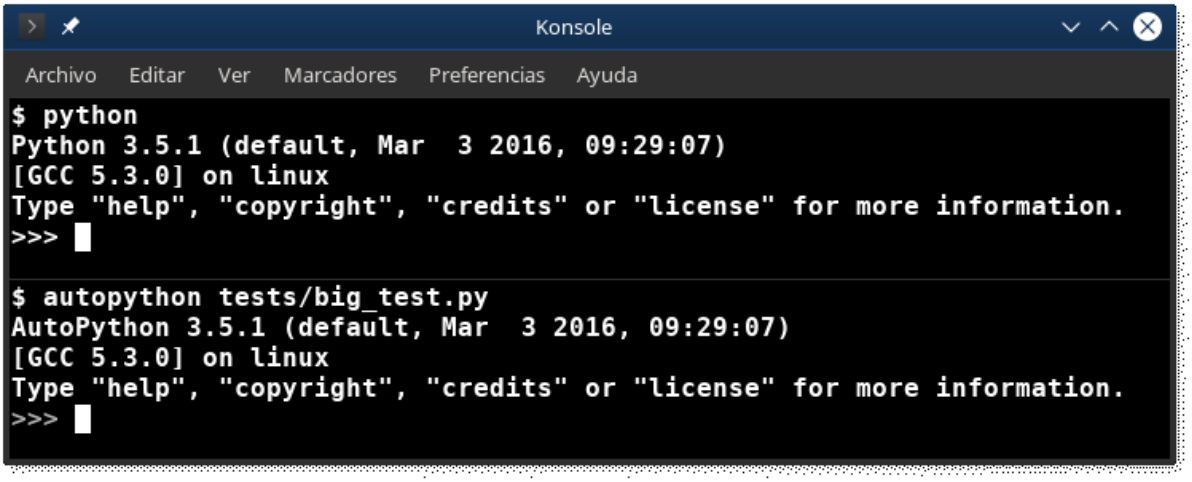
todo momento el estado actual del programa, sin necesidad de reconstruirlo tras realizar cambios en el código que se está probando.

Sin embargo, utilizar únicamente el shell interactivo para la exposición de un tema en el ámbito de una clase o conferencia exige un gran esfuerzo de parte del orador: Deberá poder lograr explicar con fluidez y claridad lo que está intentando transmitir al mismo tiempo que está obligado a tipear el código para ejemplificar lo expuesto, todo con la suficiente rapidez y precisión como para mantener a su audiencia interesada sin caer en interrupciones abruptas a raíz de errores causados por comandos mal escritos o de largas pausas mientras prepara el próximo ejemplo a mostrar. Para complicar aún más las cosas, el expositor necesitará poseer una muy buena memoria para retener en su cabeza todos los ejemplos a mostrar o, en su defecto, de un apunte con los ejemplos previamente preparados, que use de guía para ingresarlos en la consola cuando resulte propicio.

Una alternativa al esquema anterior consiste en armar una presentación estática, en la que se muestre el código del ejemplo junto con la salida que éste produce, habiéndolos previamente copiado luego de su ejecución desde la consola donde corre el shell de Python. La preparación del material de esta forma, además de ser laboriosa y volverse bastante monótona rápidamente, conlleva un riesgo mayor: El software para las presentaciones no permite ejecutar código real y por lo tanto no puede validar que la salida mostrada sea la correcta. Cualquier cambio que se haga requerirá de la constancia del creador del material para volver a ejecutar el nuevo código en el intérprete real, trasladando las respuestas obtenidas de vuelta a la presentación. Es experiencia de los autores de este trabajo que muchas veces se terminan haciendo esos cambios "a mano", modificando directamente el código y escribiendo el resultado que uno "sabe" que produciría, sin llegar a ejecutarlo realmente en el intérprete. Esto lleva indefectiblemente a que se deslicen errores, los que lamentablemente terminan siendo detectados durante la presentación o el estudio posterior del material armado.

Otra alternativa diferente es utilizar la herramienta Online Python Tutor 0, la cual resulta invaluable para visualizar el estado de un programa escrito en Python junto con el flujo del control del mismo, permitiendo ejecutar cada una de las sentencias de un programa, inspeccionando en todo momento el estado de las variables y de la pila de llamadas a funciones, avanzando instrucción a instrucción o incluso retrocediendo en la ejecución del programa, "volviendo el tiempo atrás". A pesar de su enorme utilidad como herramienta didáctica, el precio que se paga por utilizarlo es la pérdida de flexibilidad: El programa que se visualiza no puede modificarse mientras se lo está mostrando y el resultado del código ejecutado sólo puede observarse usando la función `print()` para mostrarlo como salida del programa o asignándoselo a una variable con el único fin de que se pueda inspeccionar en la visualización. Como se decía, con Python Tutor se pierde el dinamismo del shell interactivo. Si bien existe un mecanismo para integrar Python Tutor con el shell interactivo de IPython/Jupyter 0 para visualizar el estado del shell a medida que se ingresan las sentencias 0, su utilización requiere de la instalación local del servidor de Python Tutor junto con una forma particular de arrancar IPython desde la línea de comandos, lo que tal vez pueda resultar complicado o engorroso de hacer, pero fundamentalmente el uso del intérprete en esta modalidad no evita las cuestiones planteadas anteriormente con respecto al ingreso de las sentencias durante una clase o exposición.

Herramienta desarrollada



Dadas las necesidades anteriormente planteadas y ante la falta de software existente que las satisfaga, se encaró la creación de una herramienta informática a la que se denominó AutoPython[56], especialmente desarrollada para el dictado de clases o charlas que utilicen principalmente al intérprete interactivo de Python como medio para mostrar la ejecución de código que ilustre los conceptos que se desea impartir.

La finalidad de AutoPython es simple: A partir de un *script* de Python conteniendo cada una de las sentencias o fragmentos de código a mostrar, esta herramienta se encargará de simular durante una clase o presentación el ingreso de cada una de esas sentencias, como si se las estuviera escribiendo directamente en el propio shell de Python.

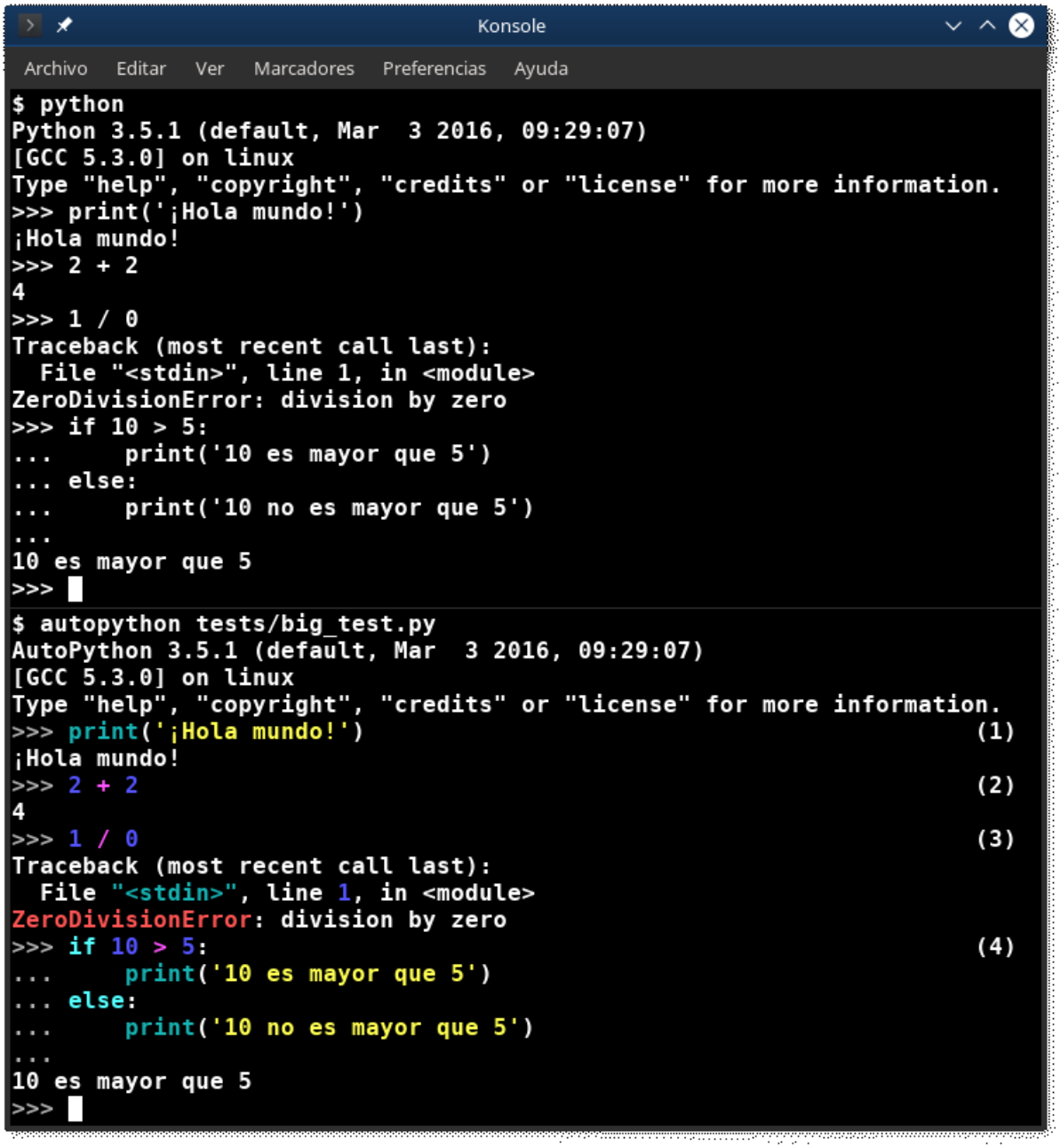
Tras invocar a AutoPython con el nombre del *script* previamente preparado, se observa una consola prácticamente idéntica a la que se vería en el shell incorporado al intérprete de Python: Una leyenda con la versión y demás información, seguido del prompt '>>>' que señala la espera del ingreso de la próxima sentencia a ejecutar. En la Imagen 1 se puede observar en la mitad superior de la misma al intérprete real mientras que en la mitad inferior se muestra a AutoPython a punto de comenzar con la presentación de un *script* cuya ruta fue indicada como parámetro.

A partir de este punto es donde esta herramienta difiere radicalmente del shell real. En lugar de permitir el ingreso libre de código, esperará que se presionen determinadas teclas para controlar lo que vaya a suceder a continuación.

Si se oprime la tecla de avance (que puede ser tanto `Av.Pág` como `Espacio` o `Enter`) AutoPython procede a simular el ingreso por teclado de la primer sentencia contenida en el *script* dado. La sentencia irá apareciendo en la consola de a un carácter a la vez, a una velocidad configurable pero variable de tipeo, insertando pausas aleatorias para darle más realismo a la simulación. Una vez que se completó dicha sentencia, el cursor quedará al final de la última línea escrita, en el punto donde, si fuera ingresada en el shell real, sólo faltaría presionar `Enter` para ejecutarla.

Si a continuación se vuelve a oprimir nuevamente la tecla de avance, AutoPython finaliza el ingreso de la sentencia simulando presionar `Enter` y mostrando en la consola exactamente el mismo resultado que dicho

56 AutoPython viene del juego de palabras entre *autopilot* (piloto automático, en inglés) y *Python*.
El porqué del nombre quedará en claro tras conocer cómo opera esta herramienta.



código produciría si fuera ingresado en el shell de Python. Oprimiendo reiteradas veces la tecla de avance permite ir mostrando y ejecutando las sucesivas sentencias contenidas en el resto del *script*.

Es importante destacar que la ejecución de cada una de las sentencias no es simulada sino que se utiliza al propio intérprete de Python para dicho fin. Todo resultado que se observe será el mismo que se obtendría al ingresar la misma sentencia en el shell de Python.

En la Imagen 2 puede compararse nuevamente la vista del intérprete real de Python junto con la de AutoPython tras haber ejecutado las mismas cuatro sentencias tomadas de un *script*. La herramienta

desarrollada imita la apariencia y el comportamiento del propio shell incorporado de Python y se prestó gran atención en reproducir el comportamiento del segundo, como cuando se escribe una sentencia de varias líneas, donde el *prompt* alterna entre '>>>' y '. . .' según si se continúa o no con el ingreso de dicha sentencia. Además, el código fuente se escribe en colores empleando resaltado de sintaxis, si una sentencia es muy larga, se la corta automáticamente en varias líneas para mostrarla completa. En la Imagen 2 también se puede apreciar que cada sentencia ejecutable es numerada con un índice (comenzando en 1) mostrando el número de sentencia entre paréntesis al final de la primera (y posiblemente, única) línea de código. Esto último es importante para poder hacer referencia fácilmente a una sentencia en particular. En el caso que existan comentarios dentro del código fuente, éstos no serán considerados como sentencias ejecutables y simplemente se los escribirá sin pausas, deteniéndose recién al llegar a la próxima sentencia.

AutoPython permite volver hacia atrás, utilizando la tecla de retroceso (configurada como `Re.Pág` o la letra `P`), regresando a sentencias que fueron mostradas anteriormente. Este comportamiento es análogo a volver a escribir una sentencia ya ingresada y no tiene ningún otro efecto en el estado interno del intérprete: Retroceder no deshace los efectos causados por una sentencia previamente ejecutada. Si se usa la tecla de retroceso en el punto en el que se está por ejecutar una sentencia (o sea, se oprimió una vez la tecla de avance, se mostró la sentencia pero aún no se la ejecutó), se simula el comportamiento de usar `Control-C` para cancelar dicha entrada y se procede a simular el ingreso de la sentencia anterior a ésta. Si, en cambio, se usa la tecla de retroceso luego de ejecutar una sentencia (es decir, se oprimió dos veces la tecla de avance), AutoPython vuelve a repetir el ingreso de la última sentencia ejecutada. Un efecto similar se consigue con la tecla de repetición (tecla `R`) que repite el ingreso de la última sentencia mostrada, independientemente de si ésta se llegó a ejecutar.

La herramienta aprovecha el hecho de que todas las sentencias ejecutadas son numeradas con un índice y permite saltar directamente a una sentencia indicando su número, utilizando la tecla de salto (tecla `G`). En este caso, el resultado observado varía de acuerdo a donde se encuentre la sentencia a saltar con respecto a la que se mostró por última vez: Si se desea saltar a una instrucción posterior, se procede a ejecutar una a una todas las sentencias necesarias para llegar desde donde está actualmente hasta la sentencia pedida. Dicha ejecución se realiza sin efectos de tipeo, para no producir pausas innecesarias. Si, en cambio, se pidió saltar a una sentencia anterior, se asume que se desea volver a escribir dicha sentencia en el contexto en que originalmente se previó su ejecución, por lo que se reinicia el estado interno del intérprete (como si recién hubiera arrancado) y se vuelven a ejecutar desde el principio todas y cada una de las sentencias necesarias para llegar a la sentencia indicada. El número de sentencia sigue la semántica de los índices en las secuencias de Python, donde valores positivos indican desplazamientos a partir del inicio mientras que valores negativos señalan desplazamientos desde el final de la secuencia. Así, el índice -1 indica que se quiere mostrar la ejecución de la última sentencia del *script*.

AutoPython provee la posibilidad de que en cualquier momento el docente u orador decida tomar el control del intérprete y comenzar a introducir sentencias de la misma manera que lo haría en el propio shell de Python. Esto se logra presionando la tecla de shell (tecla `S`). Cuando la herramienta entra en modo interactivo, toda sentencia que se escriba por teclado procederá a ejecutarse inmediatamente en el contexto del intérprete usado durante el ingreso automático de AutoPython, de forma que todo efecto que dichas sentencias produzcan afectará al resto

de las sentencias que se ejecuten más tarde (ya sean ingresadas manualmente o en forma automática). El modo de simulación se abandona utilizando la combinación de teclas EOF (*End Of File* o fin de archivo) que sobre los sistemas operativos basados en Windows se indica con la combinación de teclas `Control-Z` mientras que en sistemas operativos derivados de Unix se utiliza la combinación `Control-D`.

Finalmente, la tecla de salida (tecla `Q`) permite terminar la ejecución de la herramienta en cualquier punto de la presentación, cancelando toda sentencia pendiente de ser ejecutada y mostrando la llamada a la función `quit()`, que normalmente causa el cierre del intérprete.

Durante una presentación, AutoPython genera un archivo de bitácora en donde se almacena todo lo realizado: Inicio de la presentación, avance a la próxima sentencia, ejecución de la sentencia, retrocesos, repeticiones, saltos y cambios a modo interactivo junto con todas las sentencias ingresadas manualmente.

Toda esta información se guarda junto con una marca de tiempo indicando en qué momento se realizó cada acción. Esto se hace con la finalidad de poder analizar y depurar el *script* preparado, permitiendo detectar largas pausas entre sentencias, saltos aleatorios dentro de la secuencia prevista o sentencias ingresadas por el orador que probablemente deberían estar contenidas como parte de la exposición armada.

Metodología de uso de la herramienta

AutoPython se desarrolló específicamente como una herramienta de soporte para el dictado de clases o exposiciones en las que se desea mostrar la ejecución de una gran cantidad de líneas de código escrito en Python, liberando al orador del tedioso acto de ingresar manualmente dichas sentencias, permitiéndole enfocar la atención en lo que realmente intenta transmitir.

Con esto en mente, el orador deberá preparar previamente un *script* con código Python conteniendo todas las sentencias que quiera ejecutar, en el orden propicio para ilustrar cada uno de los puntos que desee profundizar. Dicho archivo es un *script* regular de Python, de los que pueden ser ejecutados directamente por el intérprete. Esto permite que se lo escriba usando todas las herramientas de desarrollo disponibles para Python, aprovechando las facilidades de completado y análisis de código existentes, el uso de depuradores, herramientas para pruebas, etc.

A pesar de todo, probablemente resulte que el mecanismo de comunicación empleado por AutoPython sea insuficiente ya que es normal requerir del uso de imágenes o diagramas para poder ilustrar situaciones complejas de expresar únicamente a través de código. También puede ser necesario visualizar información con distintas tipografías y/o colores para poder ordenar una idea con mayor rapidez. Estar limitado a una consola de texto que sólo muestra código restringe notablemente la posibilidad de incluir este tipo de información.

Así, la experiencia de uso óptima para esta herramienta consiste es hacer un uso mixto de algún software para presentaciones complementándolo con la ejecución automática de código provista por AutoPython. Típicamente, se prepara una presentación con el título y la agenda de los puntos a tratar, seguido de una breve descripción del primer tema hasta llegar al momento donde se señala en la presentación que se va a realizar una demostración con código. Allí se cambia a la ventana donde corre AutoPython para mostrar la ejecución de la secuencia de sentencias necesarias para ejemplificar el tema, marcando el final de la misma usando algo similar a "`pass # Regreso a la presentación`" la que, siendo una sentencia válida, no

produce ningún efecto en el intérprete pero sirve para pausar la ejecución de AutoPython, señalando el requerido retorno a la presentación. La coordinación de esta manera, si bien es rudimentaria, resulta simple y efectiva pues la elección de las principales teclas que controlan a AutoPython (las de avance y de retroceso) permite que dicha herramienta sea controlada no sólo desde el teclado de la máquina sino también desde un presentador inalámbrico usado habitualmente para software de presentaciones, pues estos dispositivos suelen simular las mismas teclas de `Av.Pág` y `Re.Pág` para avanzar y retroceder respectivamente las diapositivas. De esa manera, es posible utilizar el mencionado dispositivo para controlar tanto a la presentación como el ritmo del avance del código a mostrar.

Si bien es factible dotar a AutoPython de la capacidad de controlar directamente al propio software para presentaciones, la gran variedad de alternativas existentes (Microsoft PowerPoint, Apple Keynote, LibreOffice Impress, Adobe Acrobat, Prezi o incluso diversos sistemas para presentaciones que se muestran en un navegador web como Reveal.js, Impress.js, Bespoke.js o Deck.js, entre otros) hace que sea compleja la coordinación con cada uno de ellos, optando por dejar en manos del orador en intercambio manual entre las dos herramientas, que usualmente consiste en aplicar la combinación de teclas `Alt-Tab` o los correspondientes clics para cambiar de una aplicación a la otra. Por el momento, escapa al alcance de esta herramienta convertirse en un sistema completo que abarque el armado y reproducción de presentaciones.

Finalmente, existen algunas cuestiones importantes a prever durante el preparado del material y, sobre todo, del código a mostrar, que involucran el uso de la consola en la que se visualiza la ejecución del intérprete de Python. Debe considerarse que dicha consola deberá estar configurada para usar una tipografía cuyo tamaño sea el apropiado para una presentación. Así como una fuente de 9pt resulta insuficiente para leer cómodamente el texto escrito en un PowerPoint, lo mismo sucederá con el código presentado en una consola que emplea una fuente de tamaño similar. La mayoría de las consolas modernas admiten configurar el tamaño y la tipografía a utilizar, permitiendo el uso de fuentes de mayor tamaño y peso. Eso, sin embargo, lleva a otra cuestión a considerar: Aumentar el tamaño de la fuente reduce el número de caracteres que se pueden mostrar a la vez en una línea junto con la cantidad de líneas que entran en la pantalla, limitando el volumen de información que puede aparecer simultáneamente en un instante de tiempo dado. Tras aumentar considerablemente el tamaño de la fuente usada, no es extraño encontrarse con la necesidad de tener que restringirse al uso de una resolución de 80 columnas por 24 filas de caracteres (considerando además que la primer línea de cada sentencia será acotada aún más por AutoPython debido a la inclusión del índice entre paréntesis para numerar dicha sentencia). Las sentencias a ejecutar, sobre todo si se está mostrando una función cuya definición abarque varias líneas o una clase con sus correspondientes métodos, deberán prepararse en forma acorde. Esta restricción, a pesar de su connotación inicial negativa, resulta en el fondo beneficiosa puesto que la cantidad restringida de código que cabe en esta pantalla fuerza a presentarlo en fragmentos acotados y auto-contenidos, facilitando así su explicación y comprensión.

Resultados del uso de AutoPython

Hasta el momento, AutoPython ha sido utilizado a modo de experiencia piloto por el grupo de desarrollo del proyecto, todos docentes pertenecientes a la Escuela de Tecnología de la UNNOBA[57] en el dictado de asignaturas que emplean a Python como lenguaje en el cual expresar las ideas tratadas.

57 Universidad Nacional del Noroeste de Buenos Aires http://www.unnonba.edu.ar/

También fue utilizado durante una charla sobre "Iteradores y Generadores" en el contexto de un evento asociado a la comunidad de Python realizado en el 2015. La causa de este uso restringido se debe entre otros motivos al deseo de lograr un correcto funcionamiento de la herramienta tras haberla probado en un contexto real, buscando eliminar posibles errores tanto en la lógica de la aplicación así como en su usabilidad. A pesar de que la cantidad de usuarios actuales de AutoPython es realmente pequeña, los resultados obtenidos hasta el momento han sido muy positivos. Una vez superado el impacto inicial que provoca en la audiencia la sorpresa de ver como el código se escribe automáticamente en la consola mientras el orador se encuentra lejos del teclado, ésto rápidamente deja de ser una distracción cediendo el foco de atención al código mostrado en la pantalla. El hecho de que las sentencias vayan apareciendo gradualmente permite concentrar la atención en cada línea mostrada, de forma similar a cuando se lo escribe en un pizarrón, dándole al estudiante la oportunidad de analizarlo y comprenderlo simultáneamente con la presentación de las mismos. La ventaja de usar AutoPython en lugar del pizarrón es que el código se ejecutará verdaderamente en el contexto de un intérprete real. Por otro lado, la pausa dramática antes de la ejecución de dicho código habilita al orador a jugar con las expectativas de los oyentes, permitiendo la realización de encuestas de opinión para buscar la formación de hipótesis en cada uno de ellos al mismo tiempo que le permite medir el grado de comprensión de lo expuesto hasta el momento. La inmediata ejecución del código visualizado junto con los resultados producidos permiten validar o refutar cada una de las hipótesis previamente planteadas, preparando el terreno para posibles discusiones constructivas sobre lo observado. Todo debate iniciado, dentro de la medida justa, resultará beneficioso para enriquecer la clase, al contrastar opiniones y puntos de vista. La facilidad con la que AutoPython permite regresar rápidamente a sentencias previamente ejecutadas y observar nuevamente algún resultado que se pone en duda o la capacidad de ingresar nuevas sentencias no previstas que esclarezca el punto discutido le otorgan un dinamismo a la experiencia que resulta difícil de igualar usando una presentación estática o, peor aún, sólo el pizarrón.

Trabajo a futuro

AutoPython fue desarrollado usando Python 3 y su código fuente se encuentra disponible en un repositorio dentro de la plataforma de desarrollo colaborativo de software GitHub[58], bajo la licencia de software libre GPL versión 3. Se espera que tras la liberación de esta herramienta surjan nuevos requerimientos por parte de los potenciales futuros usuarios, además de la posibilidad de mejorar la funcionalidad actualmente provista. Es la intención de los autores continuar manteniendo y mejorando esta herramienta.

Dado que hasta el momento los resultados observados tras usar AutoPython han sido de naturaleza más bien anecdótica, se espera realizar estudios precisos y reproducibles para medir el impacto real que esta herramienta genera. En el contexto educativo, se prevé realizar encuestas tanto a los alumnos como a los docentes que utilicen este software en su asignatura. También sería apropiado contrastar el número de aprobados observados en cursos donde se la aplique con otros en los que no la usen.

En cuanto a nuevos desarrollos, AutoPython genera actualmente durante una presentación un archivo de bitácora conteniendo marcas de tiempo señalando los instantes en que se mostró y en que se ejecutó cada una de las sentencias visualizadas. También registra todo código ingresado en el modo shell. Esto permite realizar un análisis posterior de la charla o clase dada, observando si el flujo

58 AutoPython actualmente es desarrollado en: https://github.com/gosella/autopython

previsto para las sentencias fue el adecuado o si fue necesario avanzar y retroceder para cambiar ese orden preestablecido, estudiar en qué sentencias se invirtió la mayor parte del tiempo o recordar qué código fue necesario ingresar manualmente para compensar la ausencia del mismo. No obstante, el estudio de la bitácora no es trivial, a pesar de ser un archivo de texto legible por un ser humano. Un posible desarrollo a encarar es el de una herramienta para un análisis visual de esta bitácora, sugiriendo posibles puntos de mejora tanto con respecto a los tiempos o el orden de las sentencias, facilitando además el sencillo re-acomodamiento del código contenido en el *script* de la presentación.

Otro desarrollo a futuro es desacoplar la parte de presentación del código en la consola de la lógica tras la propia presentación, de forma que AutoPython pueda simular o controlar remotamente a otros shells además del provisto por el intérprete oficial. Incluir soporte para el shell de IPython en simultáneo con el que actualmente se provee está dentro de los próximos objetivos a alcanzar como parte del desarrollo de AutoPython.

Referencias

Guido van Rossum et al, "The Python Language Reference", Python Software Foundation; https://docs.python.org/3/reference/index.html

"TIOBE Programming Community Index". TIOBE Software BV. http://www.tiobe.com/tiobe_index

Pierre Carbonnelle, "PYPL: PopularitY of Programming Language Index". http://pypl.github.io/PYPL.html

Philip Guo, "Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities", BLOG@CACM, Comunications of the ACM, http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext

Philip J. Guo. Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education. In Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE), March 2013. URL: http://www.pythontutor.com/

Fernando Pérez, Brian E. Granger, IPython: A System for Interactive Scientific Computing, Computing in Science and Engineering, vol. 9, no. 3, pp. 21-29, May/June 2007, doi: 10.1109/MCSE.2007.53. URL: http://ipython.org

Python Tutor extension for the IPython shell: https://github.com/pgbovine/OnlinePythonTutor/blob/master/v3/opt-ipy.py