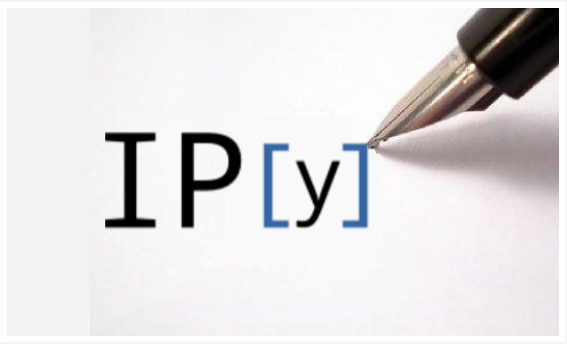
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

miércoles, 6 de agosto de 2014

Facilidades para la escritura en IPython



Estas características requieren tener instalada la biblioteca readline de GNU. A continuación, vamos a describir el comportamiento que tiene predeterminado en una sesión interactiva de IPython.

Uso del tabulador para autocompletar

Mientras escribimos una entrada podemos presionar en cualquier momento la tecla **[Tab]** para completar comandos Python/IPython, nombres de variables, rutas de directorios, etc.

: import sys
: sys.v[Tab]
sys.version sys.version_info

: sys.version

Al presionar **[Tab]** se completará hasta "version" porque aparece en los dos términos encontrados. Si después presionamos **[_]** y **[Tab]** el resultado será:

: sys.version[_Tab]
: sys.version_info[Enter]
sys.version_info(major=3, minor=4, micro=0, releaselevel='final', serial=0)

Búsqueda de comandos en el historial

IPython proporciona dos maneras de buscar en las líneas introducidas con anterioridad para evitar repeticiones en la escritura:

- Empezar a escribir y, a continuación, utilizar las teclas de **[Flecha arriba]** y **[Flecha abajo]** o **[Ctrl-p]** y **[Ctrl-n]** para buscar a través de los elementos del historial aquellos que coincidan con lo escrito en un momento dado.
- Presionar **[Ctrl-r]** para abrir un símbolo de búsqueda. Comenzar a escribir y el sistema buscará en el historial las líneas que contengan lo que se ha escrito hasta el momento, completándolo en la medida de posible.

IPython almacena el historial en una base de datos SQlite que volverá a cargar la próxima vez que iniciemos una sesión de trabajo. El archivo del historial se llama "**history.sqlite**" y se encuentra en la ruta del perfil activo:

/home/usuario/.ipython/profile_name

Sangrado de líneas

IPython puede reconocer las líneas que terminan en dos puntos '**:**' y sangrará la línea siguiente si presionamos **[Enter]**; y al mismo tiempo puede desangrar después de encontrar los términos "**return**" o "**raise**".

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

El módulo random
El módulo random de la librería estándar de Python incluye un conjunto de funciones

Esta función utiliza la biblioteca readline, por lo que respetará la configuración del archivo inputrc, normalmente, en la ubicación ~/.inputrc o donde apunte la variable de entorno INPUTRC.

Es posible personalizar el comportamiento de readline modificando la configuración del archivo .inputrc

Paréntesis y comillas automáticas

Tienen el propósito de permitir teclear menos en situaciones comunes. Objetos como funciones, métodos, etc. pueden ser invocados sin necesidad de escribir paréntesis. Por defecto, está desactivada. Es necesario activarla con **%autocall**:

: area_triangulo base, altura # es equivalente a area_triangulo(base, altura)

Ejemplo:

```
In : def area_triangulo(base, altura):
   return base * altura / 2
  .....:

In : area_triangulo 4, 4
  File "<ipython-input-115-0530e8aac90d>", line 1
    area_triangulo 4, 4
                   ^
SyntaxError: invalid syntax


In : area_triangulo( 4, 4)
Out: 8.0

In : %autocall
Automatic calling is: Smart

In : area_triangulo 4, 4
---> area_triangulo(4, 4)
Out: 8.0
```

Caracteres que fuerzan la inclusión de paréntesis y comillas

Tienen que estar al principio de la línea

```
/funcion     # La "/" incluye paréntesis: 'función()'
/zip (1,2),(5,6)     # La "/" en este caso incluye corchetes: 'zip [(1,2),(5,6)]'
,funcion /home/user     # La "," incluye comillas: funcion("/home/usuario")
;function a b c     # El ";" incluye comillas: funcion("a b c")
```

Omitir salida

Escribir un ';' al final de una línea suprime la impresión de salida. Puede ser útil cuando se hacen cálculos con resultados intermedios que no estamos interesados en conocer.

```
In : a = 43
In : b = 56
In : a + b;
In : a + b
Out: 99
```

Ir al índice del tutorial de IPython

Publicado por Pherkad en 5:39

Etiquetas: IPython, Jupyter

Entrada más reciente Inicio Entrada antigua