ReStructuredText, Sphinx, Sagepedia

Cómo escribir documentación para python y Sage

ReStructuredText Un lenguaje de marcado que permite generar documentación en varios formatos desde un mismo archivo fuente.

Sphinx Un sistema diseñado para la documentación de python usando este lenguaje, además reconoce el código python y comprueba que el resultado obtenido produce el resultado esperado.

Sagepedia Una propuesta que os hago para escribir documentación colaborativa para Sage, à la Wikipedia.

Estilo de texto

El código fuente se compila y produce un resultado (igual que en latex). Tenemos menos control que en latex pero el código es más fácil de leer.

Por ejemplo, este código:

```
*énfasis*, **énfasis fuerte (negrita)**, ``codigo``
```

produce este resultado:
énfasis, **énfasis fuerte (negrita)**, `codigo`

Párrafos

Código

```
Un párrafo, sin importar los
saltos de línea

  Otro párrafo, éste indentado

Una línea de separación indica un nuevo párrafo.
```

Resultado
Un párrafo, sin importar los saltos de línea

Otro párrafo, éste indentado

Una línea de separación indica un nuevo párrafo.

Lista

Código

```
- Item 1
- Item 2
```

Resultado

- Item 1
- Item 2

Lista numerada

Código (tenemos que numerar la lista nosotras!)

```
1. Primer punto
2. Segundo punto
```

Resultado

1. Primer punto
2. Segundo punto

Definiciones

Código

```
Item 1
    descripción

Item 2
    descripción
```

Resultado

Item 1 descripción

Item 2 descripción

Sección literal

Código

```
Ejemplo::

    Una línea termina con :: y la siguiente línea está indentada.
    Se ignoran *asteristos* y demás formato.

La sección termina *cuando termina la indentación* (como en python!)
```

Resultado
Ejemplo:

```
Una línea termina con :: y la siguiente línea está indentada.
Se ignoran *asteriscos* y demás formato.
```

La sección termina *cuando termina la indentación* (como en python!)

Cabeceras de secciones

El título de cada sección va *subrayado* por caracteres iguales. Se reserva un carácter distinto para cada nivel de sección, subsección, etc.

```
Sección 1
,,,,,,,,,

Subsección 1.1
,,,,,,,,,,,,,

Sección 2
,,,,,,,,,
```

Sección 1

Subsección 1.1

Sección 2

Docutils

El paquete de python **docutils** permite *compilar* un mismo archivo rst a distintos formatos:

`rst2html archivo.rst archivo.html` HTML

`rst2tex archivo.rst archivo.tex` LaTeX

`rst2s5 archivo.rst archivo-s5.html` S5 slides: presentación en html

`rst2odt archivo.rst archivo.odt` OpenDocument

Sphinx

- Varios archivos *rst* se organizan en una colección que contiene algunos archivos de configuración. Permite ajustar varias opciones de configuración globales.
- La colección se compila a **pdf** y **html** que podemos ver con el notebook. También se puede compilar a **latex** y **sws** (está en proyecto, no sé si ya está listo).
- Es posible extender el formato rst con construcciones para incluir *latex*, *código fuente*, etcétera

Ejemplo práctico con Sphinx

- *Copio_ y_ pego* un directorio de documentación existente en la carpeta 'es', y le cambio el nombre (por ej: '`ejota`').
- Añado el lenguaje 'es' si no lo está ya (en el archivo `$SAGE_ROOT/devel/sage/doc/common/builder.py`)
- Modifico conf.py, index.rst, borro todos los archivos rst y pongo los míos.
- Ejecuto `sage -docbuild "es/ejota" html` para generar la documentación en html.
- Observo el resultado en `$SAGE_ROOT/devel/sage/doc/output/html/es/ejota`

Tests unitarios

La sintaxis:

::

```
sage: numero = 6
sage: if numero%2==0:
...       print '%d es par'%numero
6 es par
```

permite incluir código Sage, y el resultado que se produce al ejecutar ese código (si usamos sphinx en vez de docutils tenemos resaltado de sintaxis). Sphinx permite, además, comprobar el resultado. Una llamada a:

```
sage -t ~/sage/devel/sage/doc/es/ejota/ejota1.rst
```

informa de todas las diferencias entre el resultado esperado y el resultado obtenido al ejecutar el código.

SagePedia

- Se trata de escribir el código rst en un *servidor online*.
  - Los **editores** podemos modificar el archivo rst.
  - Los **visitantes** pueden ver el documento html, y quizá valorarlo.
  - El servidor comprueba la documentación contra todas las versiones de Sage instaladas.
  - Permite usar un archivo sws para generar la primera versión del fichero rst, usando `sage -sws2rst` (trac #10637).
- No es un sitio web, es un **prototipo**.
- Sólo es una idea, que a lo mejor no es buena.

Veámoslo en acción!

- arrancar el servidor: `python web2py.py -N`
- tarea cron: `python web2py.py -C -D 1`

Reflexiones

- Actualmente se usa el mismo sistema para escribir documentación que para contribuir código python a Sage:
  - El proceso para cambiar una coma es muy tedioso.
  - Hay que comprobar toda la librería de Sage después de cualquier cambio. Esto, que para código fuente es un nivel de exigencia razonable, es exagerado para documentación, porque aunque la corrección de la documentación depende de la versión de Sage, el código de Sage no depende de que la documentación sea correcta.
  - La documentación se publica como parches contra el código de Sage, pese a que son piezas independientes.
- También se pueden poner worksheets en un servidor compartido y dar permisos a varios editores para modificarlas, pero entonces el estándar de calidad es quizá demasiado bajo:
  - Las worksheets acumulan un html bastante *chusco*, derivado del uso de **tinyMCE** (y cualquier otro editor javascript tendrá el mismo problema).
  - No se puede organizar el resultado en colecciones.
  - No es trivial incorporar las worksheets a la documentación oficial de Sage, ni tampoco hacer tests unitarios.
- Por favor, opinad! `pablo.angulo@uam.es`