Corso web developer junior con Python e Django

Alessandro Dentella

THUX

Alessandro Dentella
adentella@thundersystems.it

Documentazione

Un software senza documentazione
È un costo insopportabile

THUX

Alessandro Dentella
adentella@thundersystems.it

Motivazioni

- Un software senza documentazione opportuna in genere si riflette in un *maggior costo* in termini di tempo:
  - Tempo speso per capire cosa può fare
  - Tempo speso a spiegare ai nuovi/ai clienti come usarlo
  - Tempo speso perché rifaccio cose che non ricordavo di avere già disponibili
- Il momento in cui scrivo la documentazione automaticamente sono costretto a rileggere con occhi esterni il mio prodotto ed a fare una prima autocritica

Alessandro Dentella
adentella@thundersystems.it

Come deve essere

Per quanto dibattuta la questione il mio parere è che una buona documentazione deve

- Essere gradevole da leggere
- Essere aiutata da immagini ove servano (esempio widget particolari)
- Contenere esempi pronti all'uso anche da me stesso in un tempo successivo, quando avrò dimenticato come usarla
- Contenere codice da potere usare facilmente, ad esempio cut'n'paste:

```
CELERY_USER=thux PASS=mypass VHOST=thux_production
apt-get install rabbitmq-server
rabbitmqctl add_user $CELERY_USER $PASS
rabbitmqctl add_vhost $VHOST
rabbitmqctl set_permissions -p $VHOST $CELERY_USER ".*" ".*" ".*"
echo BROKER_URL = "amqp://$CELERY_USER:$PASS@localhost:5672/$VHOST"
```


Alessandro Dentella
adentella@thundersystems.it

Sphinx

- Lo strumento principe in questo campo è proprio sviluppato in Python e per Python: Sphinx
- I suoi punti di forza sono:
  - capacità di introspezione del codice (legge le docstring)
  - grande flessibilità nell'aggiungere testo libero. Può essere usato anche per scrivere documentazione non strettamente tecnica e non Python
  - Il markup language usato, ReST: i sorgenti sono testi leggibili anche senza strumenti particolari, editabili con un semplice editor. La suddivisione semantica del testo (necessaria per una rappresentazione graficamente significativa) è passata attraverso una struttura del testo rigida ma molto poco invasiva che a tutta prima pare dettata da un buon gusto estetico più che da una richiesta sintattica.
  - output in varie forme, html in primis, molti temi disponibili, sistema di template completo

Alessandro Dentella
adentella@thundersystems.it

Sphinx & the ReST

- Sphinx a sua volta si appoggia ed estende il pacchetto `docutils`, quindi alcuni elementi della sintassi di Sphinx sono quelli forniti da docutils, alcuni sono specifici di Sphinx stesso (ReST: ReStructured Text)
- In particolare Sphinx aggiunge l'introspezione del codice, i riferimenti incrociati fra pagine diverse e quelli fra pacchetti
- Esitono altri linguaggi di formattazione che hanno avuto molto successo, ad esempio *wiki markup* usato da *wikipedia*, ma il testo sorgente non ha assolutamente la leggibilità di un sorgente rst

Alessandro Dentella
adentella@thundersystems.it

e69_classi_tour.py

File Edit Options Buffers Tools Python Outline Help

Save Undo

```
Tour
====
In this exaple we implement a Tour object that can be filled with an ordered dict of
the cities:

    >>> from collections import OrderedDict
    >>> info_tour = OrderedDict([
        ('Roma', 3),
        ('Orvieto', 1),
        ('Perugia', 2),
    ])
    >>> tour_roma = Tour('Roma e Umbria')
    >>> tour_roma.info = info_tour  # same as: tour_roma.add_info(info_tour)
    >>> for city in tour_roma:
       print city
    Roma
    Orvieto
    Perugia
    >>> 'Roma' in tour_roma
    True
    >>> 'Usmate' in tour_roma:
    False
    >>> tour_roma.get_program():
    Programma del Tour Roma e Umbria:
           Roma: 3
           Orvieto: 1
           Perugia: 2
    >>> tour_roma.get_days()
    6

.. autoclass:: Tour
   :members:

.. autodata:: PROGRAM
"""

from collections import OrderedDict

#: the template we use to print the program of the Tour
PROGRAM = """Programma del Tour '{name}':

{0}"""


class Tour(object):
    """
    A Tour object that implements an iterable interfaces on visited cities
    """

    def __init__(self, name, cities=None, start_city='Milano'):
        """Initializer

        :arg name: nome del tour
        :arg cities: a list of city visited in the tour
        :arg start_city: the city the bus/airplane starts from
        """
        self.name = name
        self.cities = cities or [] # this way it's private to this instance
        self.start_city = start_city

    def __repr__(self):
        return "<Tour: {0}>".format(self.start)

    def __len__(self):
        return len(self.cities)

    def __str__(self):
        return self.name

    def add_info(self, info):
        """Add info to the tour about

        :arg info: an ordered dict whose keys are the cities and values
                   are the
             days the tour stops in the city
        """
        self.info = info

    def __contains__(self, key):
        return key in self.info

    def __iter__(self):
        #return iter(self.info.keys())
        for key in self.info:
            yield key

    def get_days(self):
        "Return the total number of days"
        return sum(self.info.values())

    def display(self, indentation=4):
        """Display the visited cities with indentation

        :arg indentation: the indentation
        """

        txt = ''
        ###
        for city, days in self.info.items():
            # locals restituisce un dizionario con le variabili locali
            # che in questo punto sono
            # locals().keys()= ['city', 'days', 'self', 'txt']
            # **locals() 'spacchetta' questo dizionario:
            # txt += "{city}: {days}\n".format(city=city, days=days, self=self...)
            txt += "{0}{city}: {days}\n".format(
                " "*indentation,
```

IT<-:@--- e69_classi_tour.py 1% L36 (Python Outl Server)

IT<-:@--- e69_classi_tour.py 33% L76 (Python Outl Server)

Wrote /home/sandro/Archivio/thunder/corso-python/esempi/e69_classi_tour.py

Tour

In this exaple we implement a Tour object that can be filled with an ordered dict of the cities:

```
>>> from collections import OrderedDict
>>> info_tour = OrderedDict([
    ('Roma', 3),
    ('Orvieto', 1),
    ('Perugia', 2),
])
>>> tour_roma = Tour('Roma e Umbria')
>>> tour_roma.info = info_tour  # same as: tour_roma.add_info(info_tour)
>>> for city in tour_roma:
   print city
Roma
Orvieto
Perugia
>>> 'Roma' in tour_roma
True
>>> 'Usmate' in tour_roma:
False
>>> tour_roma.get_program():
Programma del Tour Roma e Umbria:
       Roma: 3
       Orvieto: 1
       Perugia: 2
>>> tour_roma.get_days()
6
```

class course.esercizi.e69_classi_tour.**Tour**(*name*, *cities=None*, *start_city='Milano'*) [source]

A Tour object that implements an iterable interfaces on visited cities

Initializer

Parameters:
- **name** – nome del tour
- **cities** – a list of city visited in the tour
- **start_city** – the city the bus/airplane starts from

add_info(*info*) [source]

Add info to the tour about

Parameters: **info** – an ordered dict whose keys are the cities and value

display(*indentation=4*) [source]

Display the visited cities with indentation

docstring del modulo

docstring della classe

Introspezione del codice

Accesso codice sorgente della pagina

Riferimenti per la documentazione

- Suggerisco di partire dal nostro sito

  http://docs.thux.it/jmb.core/jumbo/docs.html#use-sphinx
- Il nostro sito è fatto con una personalizzazione di un tema chiamato cloud_sptheme fatta per potere aggiungere un logo e una navigazione interna ed esterna tramite menu

Alessandro Dentella
adentella@thundersystems.it

Alcuni siti fatti con sphinx

- http://docs.thux.it
- http://docs.python.org
- http://docs.djangoproject.org
- http://sqlkit.argolinux.org
- http://www.reteisi.org
- https://uwsgi-docs.readthedocs.org/en/latest/

Alessandro Dentella
adentella@thundersystems.it

Esempi - AjaxInline

- http://docs.thux.it/jmb.core/_modules/jmb/core/admin/ajax_inlines.html:
  - Titoli
  - Elenchi numerati/puntati
  - Immagini
  - Riferimenti a metodi
  - Riferimenti a capitoli
  - Codice

Alessandro Dentella
adentella@thundersystems.it

Sintassi ReST

La sintassi è studiata per essere meno invasiva possibile, e riprodurre al meglio quello che scriverebbe un autore per usare solo testo. I paragrafi devono sempre essere separati da una riga vuota dal precedente tipo. Riferimento:

http://docutils.sourceforge.net/docs/user/rst/quickref.html

http://www.reteisi.org/progetto/docs.html#rest

- Markup implicito:
  - Titoli: **sottolineato**, con caratteri differenti per i vari livelli (h1, h2,...)
  - Liste puntate/numerate/definition: * / #. :nome:
  - Codice: "::" seguito da paragrafo indentato
  - URL: parola_ ed in fondo: .. _parola: http://....
- Inline markup: **italic**, ****bold****, `` `literal` ``, :ref:`text <name>`
- Markup espliciti: ".. istruzioni". Usato per la maggior parte di costrutti che richiedono trattamento speciale
  - Immagini, note, ….

Alessandro Dentella
adentella@thundersystems.it

Sintassi Sphinx

Guardare il riferimento del sito e gli esempi riportati prima

- `toctree`

  `ref:`name`` & `ref:`text <name>``

- `automodule / autoclass / autodata`

Alessandro Dentella
adentella@thundersystems.it

Quickstart & conf

- Installare Sphinx
  - `pip install Sphinx  (in generale)`
  - `sudo apt-get install python-sphinx (debian/ubuntu)`
- Creare una cartella e lanciare `sphinx-quickstart`:

```
mkdir docs && cd docs
sphinx-quickstart
make html
```

- In questo processo è stato creato il file `conf.py`, un normale modulo python di configurazione. Da lì si possono fare molte personalizzazioni fra cui cambiare il tema

Alessandro Dentella
adentella@thundersystems.it

vscode extension

- VsCode ha una estensione che permette di editare i file `.rst` di Sphinx con una preview a fianco e tutti gli eventuali errori rappresentati correttamente nel codice. È sicuramente il modo più efficace di scrivere pagine con Sphinx: potete installarla così:

```
ext install waderyan.gitblame
```

- PyCharm ha una estensione che permette di visualizzare la preview solo di pagine ResT, ma non Sphinx. Risulta poco utile

Alessandro Dentella
adentella@thundersystems.it

Editare index.rst per aggiungere un file main.rst nel quale poi aggiungere test per sperimentare un po' tutti i costrutti visti di ReST base:

- Titoli di vario livello
- Lists (enumerated, bullet, field)
- Codice python
- Link html esterni
- Formattazione testo (grassetto, corsivo)

- Editare `index.rst` per aggiungere un altro file `info.rst`
  - Aggiungere testo vario
  - Aggiungere dei riferimenti fra un testo e l'altro
  - Aggiungere riferimenti ad un titolo
- Creare un modulo python semplice `mymod.py` con una docstring con titolo e testo e linkare questa dal secondo modulo con l'istruzione (verificare che in conf.py sia abilitata l'estensione '`sphinx.ext.autodoc`'):
  ```
  .. automodule:: mymod
  ```
- Il comando fallisce perché python non trova il modulo `mymod.py`... come fare a risolvere il problema? Cercate la soluzione in `conf.py`

- Aggiungere Sphinx con `quickstart` al progetto corso-python (nella cartella `docs`), risolvere nuovamente il problema di fare trovare il path del corso.

  *La soluzione generale che non abbiamo ancora visto è quella di usare un **ambiente virtuale** nel quale siano definiti tutti i percorsi corretti*

- Generare la documentazione di tutti gli esempi fatti

- Creare nel repo Corso2019 docs con `sphinx-quickstart`
- Aggiungere nell'`index.rst` un riferimento a `calendario.rst`
- Aggiungere in `calendario.rst` il riferimentoa tutti i giorni di lezione fatti e poi ciascuno aggiunga un giorno differente in modo da dovere fare merge di tutti i contributi
- Per ogni giorno aggiungere una nota con un argomento da mettere in evidenza
- Correggere/suggerire modifiche agli argomenti degli altri