UNIDADE 6

DOCUMENTACIÓN DE APLICACIÓNS

DESENVOLVEMENTO DE INTERFACES

CS DESENVOLVEMENTO DE APLICACIÓNS MULTIPLATAFORMA

Autor: Manuel Pacior Pérez

Índice

1 INTRODUCIÓN

No mundo do desenvolvemento de software, existe unha parte á que ás veces non se presta a atención e tempo necesarios. Estamos falando da documentación, que en ocasións determina o éxito ou fracaso dun proxecto.

A documentación do software está integrada polo ***conxunto de documentos, axudas e materiais*** realizados para ***usar, modificar, ampliar e manter*** de xeito óptimo ***o noso sistema***.

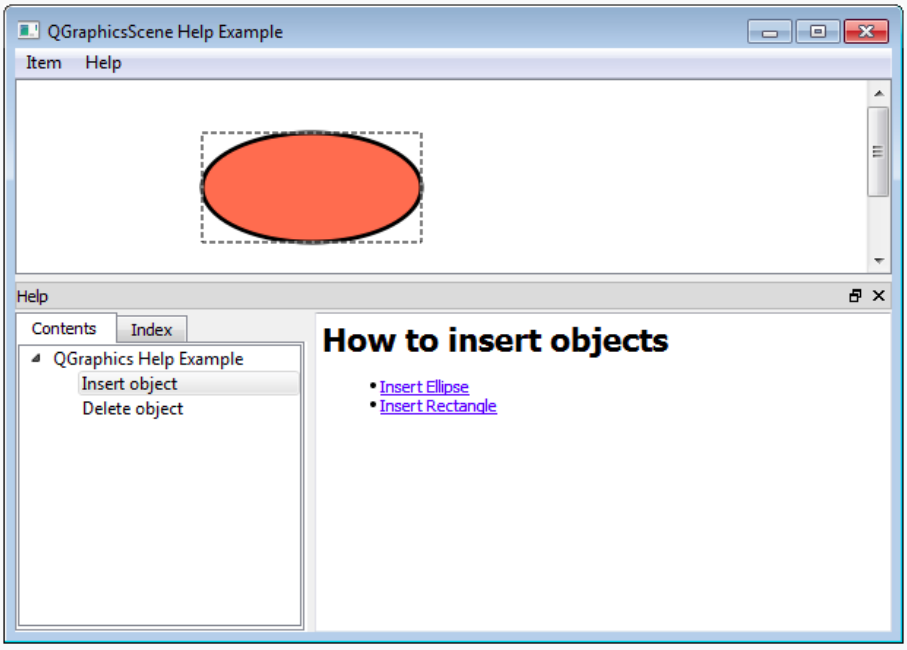
A continuación imos ver os diferentes tipos de axudas e documentos que se crean ao longo do ciclo de vida dun proxecto de desenvolvemento de software.

2 SISTEMAS DE AXUDA INTEGRADOS

Os sistemas de axuda integrados están Inseridos na propia aplicación para ofrecer, dun xeito rápido e directo, información sobre o funcionamento da mesma.

2.1 Desde o menú da aplicación

Desde a propia aplicación podemos crear unha opción que amose unha fiestra que integre as diferentes opcións da aplicación, onde se detalle cada unha delas, tal como se amosa na seguinte imaxe:



Outra opción bastante habitual é facer un manual de usuario ao que se poida acceder desde a propia aplicación, normalmente en formato PDF. Desde Qt podemos usar *QWebEngineView* para visualizar páxinas Web e incluso documentos PDF, tal como se pode ver no seguinte exemplo.

```
# -*- coding: utf-8 -*-
"""
Módulo principal do xogo encargado de controlar e inicializar todos os
compoñentes visuais.
"""
import sys
from PySide2.QtCore import QUrl, QFileInfo
from PySide2.QtWidgets import QMainWindow, QApplication
from PySide2.QtWebEngineWidgets import QWebEngineView, QWebEngineSettings


class PdfWindow(QMainWindow):
    def __init__(self, parent=None):
        super(PdfWindow, self).__init__(parent)
        self.setWindowTitle("Visor PDF")
        self.web = QWebEngineView()
        self.web.settings().setAttribute(QWebEngineSettings.PluginsEnabled,
True)

self.web.settings().setAttribute(QWebEngineSettings.FullScreenSupportEnabled,
True)
        self.web.settings().setAttribute(QWebEngineSettings.PdfViewerEnabled,
True)
        self.setCentralWidget(self.web)

    def showPdf(self, path_to_file: str):
        try:
            absolute_path = QFileInfo(path_to_file).absoluteFilePath()
            url_to_manual = QUrl.fromLocalFile(absolute_path)
            self.web.load(url_to_manual)
            self.showMaximized()
        except Exception as e:
            print(str(e))


if __name__ == '__main__':
    # Definimos a instancia da aplicación principal Qt
    # e pasamos a lista de argumentos do sistema.
    app = QApplication(sys.argv)

    pdf_window = PdfWindow() # Creamos a nosa aplicación que amosa o PDF

    pdf_window.showPdf("doc/manual.pdf") # Amosamos o PDF da ruta indicada

    # Inicia o ciclo de eventos até que se pecha a aplicación
    sys.exit(app.exec_())
```

2.2 Axuda contextual

A axuda contextual nunha aplicación é aquela que obtemos ao situarnos enriba dos elementos, de xeito que se desprega información respecto á funcionalidade que realiza ou representa.

Para realizar isto en Qt temos o compoñente *QToolTip* que nos permite engadir información de contexto aos compoñentes de tipo *QWidget*.

Debemos ter en conta, que se estamos usando un compoñente de tipo *QMenu* será necesario habilitar esta opción, xa que por defecto está deshabilitada.

A continuación deixo un exemplo do uso desta propiedade, da que tedes máis información no seguinte enlace.

```
file_menu = QMenu("Arquivo")
file_menu.setToolTipsVisible(True) # Habilitamos a axuda contextual
open_action = QAction(QIcon('new.png'), "Novo", self)
open_action.setShortcut("Ctrl+N")
QToolTip.setFont(QFont("Decorative", 10, QFont.Bold))
open_action.setToolTip("Comezar un novo xogo")
open_action.triggered.connect(self.open_file)

exit_action = QAction(QIcon('exit.png'), "Saír", self)
exit_action.setToolTip("Saír da aplicación")
exit_action.setShortcut("Ctrl+X")
exit_action.triggered.connect(self.exit_app)

file_menu.addAction(open_action)
file_menu.addAction(exit_action)
```

3 DOCUMENTACIÓN TÉCNICA

A documentación técnica relaciona o código coa linguaxe común, para crear un tipo de texto híbrido no que confiamos cando queremos comprender o que fai unha determinada función ou o que podemos esperar dun atributo concreto. Para a documentación técnica asumimos máis o rol dun escritor que o de programador.

Achegámonos á documentación técnica cando queremos aprender a usar un novo framework ou cando estamos aprendendo unha linguaxe de programación.

Este tipo de documentación indícanos a razón dalgunhas decisións, dannos consellos e indica formas seguras ou rápidas de resolver problemas. Tamén inclúe normalmente o documentado dentro do código.

Podemos atopar este tipo de documentación na maioría dos proxectos de GitHub, GitLab, frameworks, bibliotecas. Podemos comprobalo visitando algúns dos sitios de documentación dalgúns proxectos coñecidos como poden ser:

- Sphinx
- Python 3
- Qt Python
- Django

En resume, ese será o lugar ao que vamos cando temos dúbidas, ou queremos comprender mellor algunha funcionalidade.

Podemos desenvolver a mellor libraría do planeta, mas sen a documentación axeitada serán moi poucos os usuarios ou desenvolvedores o suficientemente pacientes para usala.

3.1 Guía de estilo (PEP8)

Unha guía de estilo trata sobre a consistencia, algo importante á hora de abordar un proxecto de desenvolvemento de software. Python ten a súa propia guía de estilo, chamada PEP-8 que é un recurso moi útil, xa que impón certo acordo entre os programadores de Python, facilitando así a lexibilidade do código en xeral. PEP-8 inclúe suxestións relacionadas con:

- A *indentación* correcta: 4 espazos.
- A extensión dunha liña de código (non máis de 79 caracteres).
- A separación de fragmentos de código por medio de liñas en branco.
- Formato da importación de librerías e módulos.
- Uso apropiado dos comentarios.
- Convencións nos nomes utilizados.
- Suxestións varias sobre funcións, clases e variables.

Adherirse a unha guía de estilo é moi importante. Sexa a PEP-8 ou a que suxira a túa organización. Isto permite programar dun xeito máis eficiente, xa que moitas decisións de formato fanse de xeito automático, ademais tamén fai que a corrección do código sexa máis sinxela.

3.2 Sistemas de rexistros (logs)

Cando desenvolvemos unha aplicación en Python (ou calquera outra linguaxe de programación), xurde a necesidade de ter algún tipo de rexistro onde se garde a información relativa á execución do programa (normalmente faise sobre un ficheiro externo).

Imaxinade que poñedes unha aplicación en produción e que non ten integrado ningún sistema de logs, é dicir, que non xera información do que fai, e que nun momento dado o cliente chama para notificar unha incidencia ao equipo de desenvolvemento. Sen máis información que esa, poder identificar e replicar o erro, pode tornarse unha tarefa tremendamente complexa.

Moitos desenvolvedores teñen cando comezan a programar tenden a imprimir texto por consola mediante o uso de instrucións estándar (por exemplo ***print*** en python). Isto ten varios inconvenientes:

- Non se poden deshabilitar, sempre se van imprimir salvo que os eliminemos ou comentemos. Facer isto non é práctico e moi tedioso.
- As mensaxes saen so por consola, non se gardan en ningún formato persistente (ficheiro ou BD), polo que non é posible consultalos en calquera momento.

Para solucionar isto temos os sistemas de logs, que permiten configurar os niveis de información que desexamos amosar sobre a execución da nosa aplicación, e ademais soporta medios persistentes, para poder consultalos cando desexemos, en caso de que xurda algún problema.

3.2.1 Introdución ao sistema de logging

En Python temos un módulo que facilita a tarefa de creación e xestión dos rexistros de logs chamada ***logging***.

As funcións de rexistro denomínanse en función do nivel ou da gravidade dos eventos que se usan para o rastrexo. A continuación descríbense os niveis estándar e a súa aplicabilidade (en orde crecente de gravidade):

NIVEL	VALOR	CANDO SE USA
NOTSET	0	Saca absolutamente toda a información
DEBUG	10	Información detallada, normalmente só de interese durante o diagnóstico do problema.
INFO	20	Confirmación de que as cousas funcionan como se esperaba.
WARNING	30	Unha indicación de que ocorreu algo inesperado, ou indicativo dalgún

		problema nun futuro próximo (por exemplo, "espazo en disco baixo"). Malia iso, o software segue funcionando como se esperaba.
ERROR	40	Debido a un problema máis grave, o software non foi capaz de realizar algunha función.
CRITICAL	50	Un grave erro, que indica que o propio programa pode non poder seguir traballando.

O nivel predeterminado é ***WARNING***, o que significa que só se fará un seguimento dos eventos neste nivel e nos superiores, a non ser que o paquete de rexistro estea configurado para facer o contrario.

Un exemplo simple do uso desta libraría sería:

```
import logging
logging.error('Hai un erro!')  # Imprimirá esta mensaxe por consola
logging.warning('Coidadiño!')  # Imprimirá esta mensaxe por consola
logging.info('Xa cho dixen!')  # Non imprimirá nada
```

Se escribimos estas liñas nun script e o executamos, veremos que por consola sae a seguinte mensaxe:

```
ERROR:root:Hai un erro!
WARNING:root:Coidadiño!
```

Fixémonos que a mensaxe de nivel INFO non aparece, xa que por defecto o nivel base é WARNING, polo que so sairán os niveis igual ou superiores a ese.

3.2.2 Configuración do rexistro de logging

Os programadores poden configurar o rexistro de tres maneiras:

- Creando rexistradores, manexadores e formatos explicitamente, usando código Python que chama aos métodos de configuración.
- Creando un arquivo de configuración de rexistro e léndoo usando a función *fileConfig()*.
- Creando un dicionario de información de configuración e pasándoo á función *dictConfig()*.

Neste módulo veremos unicamente a configuración básica do rexistro de logs a través do código Python usando un ficheiro de saída mediante o uso do rexistrador por defecto. Aínda así é importante ter en conta que o módulo ***logging*** permite definir varios rexistros

con diferentes manexadores, entre os que destaca 'RotatingFileHandler', o cal permite rotar os ficheiros de saída en base a determinadas regras. Pódese consultar as diferente opcións de configuración de xeito detallado a través do seguinte link.

A continuación vemos un exemplo de funcionamento onde se usa o mecanismo de configuración básica, de xeito que sacamos a un ficheiro chamado 'myapp.log' a información.

simple_logging_module.py

```
import logging

logging.basicConfig(level=logging.DEBUG, format='%(asctime)s %(name)-12s %
(levelname)-8s %(message)s', datefmt='%m-%d %H:%M', filename='file.log')

# 'application' code
logging.debug('debug message')
logging.info('info message')
logging.warning('warn message')
logging.error('error message')
logging.critical('critical message')
```

No exemplo anterior compre destacar as seguintes opcións de configuración:

- **level**: Indicamos o nivel *DEBUG*
- **format**: Representa o xeito no que sacamos a información ao ficheiro. Normalmente é importante que saia como mínimo a data completa, o nivel de rexistro da mensaxe e o propio mensaxe.
- **datefmt**: Configuramos o formato no que desexamos que saia a data.
- **filename**: Indicamos o ficheiro de saída no que se gardarán as mensaxes de log.

Ao executar este código desde a liña de comandos

```
$ python simple_logging_module.py
```

Obtemos un ficheiro, na mesma ruta onde temos simple_logging_module.py, co nome *file.log* que conterá a seguinte información:

```
04-04 03:15 root         DEBUG    debug message
04-04 03:15 root         INFO     info message
04-04 03:15 root         WARNING  warn message
04-04 03:15 root         ERROR    error message
04-04 03:15 root         CRITICAL critical message
```

Como norma xeral, cando unha aplicación entra en produción tende a ter un nivel de

rastrexo alto. Isto faise así, debido a que un nivel moi baixo pode incidir de xeito negativo sobre a fluidez de execución da aplicación, pois implica que esta terá que estar continuamente realizando operacións de E/S a disco (moito máis lento que as operacións de E/S sobre memoria RAM). Ademais o rexistro tamén tería demasiada información e podería chegar a facerse ilexible.

Polo tanto, durante o tempo de proba ou desenvolvemento, pódese empregar o nivel de rexistro máis baixo, DEBUG, pero unha vez que a aplicación está nun ambiente de produción, recoméndase usar só o nivel ERROR ou WARN.

Pode atopar máis información detallada na seguinte ligazón da documentación oficial.

3.3 Documentar o código

3.3.1 Comentarios no código

En Python, podemos facer comentarios de varios xeitos:

- Escribindo o símbolo **#** ao comezo da liña de texto onde queremos engadir o noso comentario.
- Escribindo triplo aspas ao principio e ao final do comentario. Neste caso os comentarios poden ocupar mais dunha liña. Podemos usar comiñas simples ou dobres, mas as de peche deben coincidir coas de inicio (non se poden mesturar as dúas sobre un mesmo bloque comentado).

Ten en conta algunhas convencións cando comentes o código fonte de Python:

- Non satures o teu código con comentarios dunha soa liña.
- Os comentarios dunha soa liña deben estar separados polo menos por dous espazos do signo de comentario (#).
- Úseas cando realmente penses que son necesarias, non comentes un código que sexa fácil de entender.

3.3.2 Docstrings

Python ten outra ferramenta para axudar aos desenvolvedores a documentar o seu código chamada ***docstrings***. Trátase simplemente de comentarios de varias liñas postos despois da declaración dun obxecto. Polo tanto, proporcionan un xeito cómodo de documentar funcións, módulos, clases e métodos en Python para darlle información ao usuario ou desenvolvedor, sobre o uso dalgún obxecto.

Se quixéramos ver a "axuda" sobre o obxecto *dict* en Python, poderíamos escribir o seguinte no intérprete:

```
>>> help(dict)
```

Isto devolvería un *docstring* relacionado co obxecto, xa que Python trae incluídos *docstrings* básicos. Malia iso, podemos e debemos crear as nosas propias cadeas de documentación para cada obxecto novo que deseñemos.

3.3.2.1 Guía de estilo docstrings Python de Google

Nos últimos anos Python está a converterse nunha das linguaxes de máis uso e crecemento, isto é así, entre outras cousas, porque grandes compañías de software comezaron a usalo en moitos dos seus proxectos.

Unha destas compañías é Google, que ademais definiu o seu propio estilo para comentar o código Python facendo uso de docstrings, e que aconsello debido á súa facilidade de uso e boa lexibilidade. Na seguinte ligazón tedes un exemplo completo da definición desa guía de estilo.

3.4 Documentación desde o código fonte

Existen unha serie de ferramentas que permite xerar a documentación técnica dunha aplicación. A continuación indicamos algunhas delas:

- *Sphinx*: Trátase dun xerador de documentación que converte ficheiros reStructuredText en sitios web HTML e outros formatos, incluíndo PDF, EPub e man. Aproveita a natureza extensible de reStructuredText e as súas extensións (por exemplo, para xerar automaticamente documentación a partir do código fonte, escribir notación matemática ou destacar código).
- Doxygen: é un xerador de documentación compatible con múltiples linguaxes de programación (C++, C, Java, Objective-C, Python, PHP ou C#). É facilmente adaptable, funciona na maioría sistemas Unix, así como Windows e Mac OS X. Varios proxectos como KDE usan esta ferramenta para xerar a súa documentación API.

3.4.1 Introdución a Sphinx

Para realizar a documentación da nosa aplicación usaremos Sphinx, que permite xerar a documentación en diferentes formatos: HTML, pdf ou epub.

Esta ferramenta ofrece un equilibrio perfecto de xeito que podemos:

- Crear páxinas de documentación de xeito manual usando o formato ReStructuredText.
- Xera a documentación automaticamente a partires do código ben documentado da aplicación facendo uso da extensión 'sphinx.ext.autodoc'.
- Existe unha grande comunidade, polo que existen unha morea de recursos:
  - ***Extensións***: incrementan a funcionalidade da ferramenta, no seguinte enlace temos un listado de todas as que soporta.
  - ***Temas***: Hai unha chea de temas que podemos utilizar, o que permite que a nosa documentación se xere co deseño que máis se adapte ao que precisamos. Na seguinte ligazón temos unha boa colección deles.

3.4.2 Instalar Sphinx

Para comezar co uso de sphinx o primeiro que temos que facer é instalalo, a miña recomendación é facelo usando PyPI, que é a ferramenta que descarga os paquetes desde o Python Package Index, que normalmente sempre se atopa actualizado ás últimas versións e ademais funciona baixo calquera SO.

```
$ pip install -U sphinx
```

Tedes máis información ao respecto na seguinte ligazón.

3.4.3 Iniciar o noso proxecto de documentación Sphinx

Unha boa práctica é crear dentro do noso proxecto unha carpeta que conterá a documentación, por exemplo podemos chamarlle *doc*,

Unha vez temos a nosa carpeta para a documentación, situámonos dentro dela e executamos o seguinte:

```
$ sphinx-quickstart
```

Ao executar isto faranos varias preguntas,

```
> Separate source and build directories (y/n) [n]: y
> Project name: proyecto
> Author name(s): Breixo Souto Maceiras
> Project release []: 0.1.0
> Project language [en]: gl
```

Recomendamos responder afirmativamente á primeira pregunta (se non se indica por defecto é non), xa que deste xeito teremos separada dentro da carpeta 'doc' en dúas carpetas diferentes a orixe (carpeta *source*) sobre a que iremos construíndo as páxinas que formarán parte da nosa documentación en formato ReStructuredText, e outra *build* na que xeraremos a documentación no formato desexado (HTML, PDF, EPUB).

3.4.4 Configurar Sphinx

3.4.4.1 Modificar o ficheiro de configuración (config.py)

Cando remata a execución, veremos que dentro da carpeta *doc/source* hai varios ficheiros e directorios, entre eles un co nome *conf.py*.

O primeiro que debemos facer é engadir varias extensións:

```
import sphinx_rtd_theme
import os
import sys
import datetime
sys.path.insert(0, os.path.abspath('../../lib'))
```

Descomentamos as primeiras liñas e engadimos a ruta onde se atopa o noso proxecto. Este paso é fundamental para que a extensión *sphinx.ext.autodoc* que configuraremos despois atope o código do noso proxecto e xere automaticamente a documentación a partires dos comentarios. No caso do meu proxecto de exemplo e que tedes subido no gitlab, o código fonte da aplicación atópase no raíz dentro da carpeta *lib*.

A continuación debemos engadir varias extensións, tal como se indica (en letra grosa as obrigatorias):

```
extensions = [
    'sphinx_rtd_theme',
    'sphinx.ext.autodoc',
    'sphinx.ext.napoleon',
]
```

A continuación detallo para que serve cada unha das extensións indicadas arriba:

- *sphinx_mdolab_theme*: Foi o tema que elixín eu para a documentación, podedes usar calquera outra que vos goste. Antes de poder usala hai que instala, neste caso: *pip install sphinx-mdolab-theme*.
- *autodoc*: Esta extensión é a que permite que Sphinx explore o noso proxecto e xere a documentación ao estilo dunha API a partires do noso código ben documentado.

- *napoleon*: Esta extensión é necesaria para que Sphinx sexa compatible coa *guía de estilos docstring definida por Google*.

Outras extensións que poden ser de utilidade son:

- *intersphinx*: Permite facer links entre documentacións
- *todo*: Adiciona ferramentas para manter o controle de "Tarefas" (cousas pendentes de facer).
- *mathjax*: permite adicionar fórmulas matemáticas escritas en LaTeX.
- *viewcode*: Permite ver o código fonte desde a documentación, o que moi cómodo.
- *autosummary*: Xera os arquivos *.rst*, para automatizar aínda mais o traballo que a extensión *autodoc* fai. Usar ou non depende do control que queiramos ter sobre o resultado final.

Podedes usar outras a maiores, no caso de que queirades engadir novas funcionalidades ao sistema, iso si, *sphinx.ext.autodoc* e *sphinx.ext.napoleon* son necesarias para traballar neste módulo.

No caso de que non usedes o tema por defecto, é necesario que o indiquedes nas extensións (como xa vimos), e ademais hai que indicalo tamén no parámetro *html_theme* do seguinte xeito:

```
html_theme = 'sphinx_rtd_theme'
```

Todos os temas teñen unhas instrucións de instalación a realizar sobre o config.py (soen ser moi doadas). No caso do proxecto de exemplo do xogo das parellas que subín ao gitlab podedes consultalas na seguinte ligazón.

3.4.4.2 Configurar o índice (index.rst)

O seguinte paso que debemos realizar é configurar o ficheiro index.rst para engadir as nosas páxinas en formato ReStructuredText. Inicialmente temos un ficheiro cun contido similar a este.

```
.. sample documentation master file, created by
   sphinx-quickstart on Mon Apr  5 12:14:09 2021.
   You can adapt this file completely to your liking, but it should at least
   contain the root `toctree` directive.

Welcome to sample's documentation!
==================================
```

```
.. toctree::
   :maxdepth: 2
   :caption: Contents:


Indices and tables
==================

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

A continuación describimos os elementos máis importantes:

- *Encabezados*: Hai varios títulos de encabezado que están suliñados indicado na liña posterior "=", podemos poñer o nome que queiramos, e debemos ter en conta que o suliñado debe ter o mesmo tamaño que o texto (senón dará erro ao xerar a documentación).
- *toctree*: Ao documentar imos engadindo a este elemento o conxunto de páxinas que forman parte da documentación. Tanto as xeradas automaticamente como as feitas de xeito manual irán indicadas aquí. Neste caso non hai ningunha referencia (*.rst*) indicada, se así fose, Sphinx buscaría cada unha delas á hora de xerar a documentación.
  - *maxdepth*: Este parámetro ten o cometido de indicar o máximo de profundidade no que buscará as referencias. Por defecto é 2 (pode modificarse), isto quere dicir que buscará nas referencias ao ficheiros .rst indicados, e se estas teñen máis referencias a outros .rst, tamén sairán, máis parará aí e non terá en conta as referencias que poida haber a partires do terceiro nivel.
  - *caption*: Representa o título co que se encabezará a documentación, podemos configurar o que queiramos, por exemplo "*Táboa de contidos*"
- *Xeración de índices e táboas*: Sphinx permite automatizar a xeración de índices; o índice xeral, o índice do módulo Python, e a páxina de busca.
  - *genindex*: O índice xeral cúmprese con entradas de módulos, todas as descricións de obxectos xeradores de índices e directivas de índice.
  - *modindex*: O índice do módulo Python contén unha entrada por directiva *py:module*.
  - *search*: A páxina de busca contén un formulario que usa o índice de busca JSON

xerado e JavaScript, para buscar no texto dos documentos xerados as palabras indicadas. Debería funcionar en todos os principais navegadores que admitan JavaScript nunha versión non obsoleta.

A continuación indico unha posible configuración de exemplo para un proxecto do ficheiro *index.rst*:

```
.. Pairs Game documentation master file, created by
   sphinx-quickstart on Sun Apr  4 00:10:58 2021.
   You can adapt this file completely to your liking, but it should at least
   contain the root `toctree` directive.

Benvido á documentación do Xogo de parellas!
============================================

.. toctree::
   :maxdepth: 2
   :caption: Contidos:

   README
   api/modules
   api/game
   api/main


Índices e táboas
================

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

Como se pode ver engadín unha serie de referencias que representan ficheiros en formato ReStructuredText, (malia que non se inclúe o formato .rst) e que detallo a continuación:

- ***README***: Este ficheiro foi realizado de xeito manual, nel fago unha presentación do proxecto usando o formato ReStructuredText.
- ***Documentación auto-xerada***: A continuación indícanse os ficheiros en formato ReStructuredText que conteñen a información relativa á documentación do código fonte do noso proxecto. Estes ficheiros son auto-xerados, a continuación indicarase como se fai, e unha boa práctica é facelo nun directorio propio dentro de ***source***, no meu caso chameille ***api:***
  - api/modules
  - api/game
  - api/main

3.4.5 Xerar a API a partires do código

Xa que estamos a usar *autodoc*, o noso primeiro paso será xerar a extracción da API mediante o comando *sphinx-apidoc*. A continuación indico o exemplo para o proxecto do xogo de pares realizado como exemplo, partindo da base de que nos atopamos baixo o directorio *doc*:

```
$ sphinx-apidoc -f -M -o source/api/ ../src/game/
```

Os parámetros indicados fan o seguinte:

- *-f*: Serve para forzar a rexeneración dos ficheiros, importante se actualizamos a documentación da API cando xa foi xerada con antelación.
- *-M*: localizar primeiro a documentación dos módulos (por defecto son as funcións primeiro, e isto non me parece natural).
- *-o source/api/*: Indicamos a saída (output) onde queremos que se xeren os ficheiros .rst que definen a API, para o que indicamos a ruta (a carpeta *api* que creamos para iso dentro do directorio *source*)
- *../src/game/*: Define a ruta onde se atopa o código ben documentado da nosa aplicación a partires do cal se xera a API.

Unha vez se xeran os ficheiros a partires da API, será cando teremos que engadilos ao ficheiro principal *index.rst* comentado no apartado anterior.

3.4.5.1 Exportar a HTML

Unha vez temos auto-xerada a api en formato ReStructuredText da nosa aplicación, podemos xerar a documentación en formato HTML, para iso debemos executar, baixo o directorio *doc* o seguinte:

```
$ make html
```

Comprobaremos que se crea o directorio *doc/build/html* e dentro teremos un ficheiro *index.html* que podemos abrir directamente con calquera navegador web.

3.4.5.2 Exportar a PDF

Para poder exportar a PDF, antes temos que instalar unha serie de paquetes (como xa indiquei ao comezo de curso, é aconsellable ter unha máquina virtual con Linux, ou ben telo instalalo nunha partición):

```
$ sudo apt-get install texlive-latex-recommended texlive-latex-extra texlive-fonts-recommended latexmk
```

Unha vez feito isto, podemos xerar a documentación en formato PDF executando:

```
$ make latexpdf
```

Comprobaremos que se crea o directorio *doc/build/latex* e dentro teremos un ficheiro *.pdf* por defecto *python.pdf* que terá a documentación do noso proxecto.

3.5 Documentación automatizada de BD

Antes de nada imos falar sobre o concepto de enxeñaría inversa de BD, que non é máis que o conxunto de técnicas que permiten obter unha representación conceptual dun esquema de base de datos a partir da súa codificación.

As súas aplicacións son múltiples:

- Volva documentar, reconstruír e / ou actualizar a documentación da base de datos que falta ou non existe
- Servir como pivote nun proceso de migración de datos
- Axudar na exploración e extracción de datos en bases de datos mal documentadas.
- A información que se pode extraer, dependendo do punto de partida, pode ser:
  - Entidades
  - Relacións
  - Atributos
  - Claves primarias ou foráneas
  - etc.

  A partir destes elementos créanse modelos de datos, como diagramas entidade-relación.

Ademais, se temos a BD con todos os seus compoñentes (táboas, relacións, campos, procedementos almacenados, triggers, ...) correctamente documentados, poderemos obter unha representación bastante completa da mesma mediante o uso de ferramentas que aplican técnicas de enxeñaría inversa.

A continuación indicamos varias ***ferramentas en código aberto*** que permiten aplicar este

tipo de técnicas:

- **dbdocs**: Trátase dunha ferramenta sinxela e gratuíta para crear documentación de base de datos baseada na web mediante código DSL. Está deseñado para desenvolvedores e Integra perfectamente o fluxo de traballo de desenvolvemento. Pódese atopar máis información na web oficial do proxecto.
- **SchemaCrawler**: É unha ferramenta de descubrimento e comprensión de esquemas de base de datos. Xera diagramas de esquemas e pode executar scripts en calquera linguaxe de script estándar contra a base de datos. Admite case todas as bases de datos que teñan un controlador JDBC e funciona con calquera sistema operativo que admita Java SE 8 ou superior. Podemos consultar a información detallada na súa web oficial.
- **SchemaSpy**: Ferramenta similar á anterior que permite xerar a documentación da BD en formato HTML, incluídos os diagramas ER. Podemos consultar máis información na súa páxina oficial.

4 MANUAIS

Durante o desenvolvemento dunha aplicación informática, os desenvolvedores ou desenvolvedores, que a están a implementar, xeran unha documentación do código que se crea, co fin de facilitar o mantemento da aplicación e a corrección dos posibles erros que se producen durante o desenvolvemento e uso. o sistema desenvolvido. Non obstante, na maioría dos casos, os usuarios ou usuarios finais da aplicación serán diferentes aos desenvolvedores da aplicación e posiblemente descoñezan completamente o código da aplicación que usan.

Para facilitar o uso da aplicación creada, é necesario que como parte do proxecto se xeren documentos, tanto en formato electrónico como impresos en papel, que permitan aos usuarios e usuarios finais da aplicación instalar, configurar, usar, e manter, actualizar e eliminar no seu caso. Estes documentos que se incorporan ás aplicacións son os manuais.

Na actualidade, hai unha tendencia a prescindir de manuais impresos en papel, distribuíndose en formato electrónico e podendo descargalos dos sitios web dos desenvolvedores de software.

Entre os tipos de manuais máis comúns que atopamos, aparecen os seguintes:

- Manual do usuario.
- Guía de referencia.

- Guías rápidas.
- Manuais de instalación.

4.1 DESTINATARIOS DOS MANUAIS

Cando se preparan manuais para aplicacións de software ou sistemas operativos, é necesario distinguir os distintos tipos de usuarios aos que están destinados. Cada grupo de usuarios ten un papel definido dentro do sistema informático e cada grupo ten privilexios e coñecementos específicos.

Pódense distinguir os seguintes destinatarios:

- **Administradores**: A este tipo de usuarios soen ir dirixidos os seguintes tipos de documentación:
  - Manual de instalación
  - Manual de configuración e administración
- **Usuarios**: A este tipo de usuarios van dirixidos os seguintes elementos da documentación
  - Manuela de usuario (todo tipo de usuarios)
  - Guía rápida (todos tipo de usuarios)
  - Guía de referencia (usuarios experimentados)

4.2 ESTRUTURA DUN MANUAL

A estrutura dun manual é a forma en que o documento está organizado en capítulos, estes capítulos en seccións e subseccións. O xeito no que organiza un documento ten un grande impacto na lexibilidade e usabilidade, porén o deseño elixido é un factor moi importante.

Non existe un método único para acadar este obxectivo, aínda que si existen unha serie de recomendacións. A estrutura que se recomenda para ter manuais debería ter os seguintes compoñentes:

COMPOÑENTE	DESCRICIÓN
Datos de identificación	Título que identifica de xeito único o documento
Táboa de contidos	Título de cada sección e subsección, xunto co número de páxina
Lista de ilustracións	Número da figura e título

Introdución	Defina o obxectivo do documento cunha breve descrición do contido
Información de uso da documentación	Suxestións para diferentes tipos de lectores sobre como usar a documentación de forma eficiente
Procedementos	Pautas sobre como usar o software para completar as tarefas para as que foi deseñado
Información do comando	Unha descrición de cada comando soportado polo software
Mensaxes de erro e solución de problemas	Unha descrición dos erros que se poden informar e como recuperarse dese erro
Glosario	Definición dos termos especializados empregados
Fontes de información relacionadas	Referencias ou ligazóns a outros documentos que proporcionan información adicional
Elementos de navegación	Elementos que permiten aos lectores atopar a súa situación actual e moverse polo documento
Índice	Unha lista de termos clave e as páxinas ás que se refiren eses termos
Capacidade de busca	Na documentación electrónica, facilidade para atopar un termo específico no documento

Táboa. Estrutura dun manual

4.3 Elaboración de manuais

4.3.1 Manual de usuario

Un manual de usuario é un documento técnico de comunicación que proporciona asistencia aos usuarios que utilizan un sistema rematado. Normalmente, un manual de usuario adoita ser preparado por un escritor ou escritor técnico, aínda que tamén o adoitan escribir programadores, enxeñeiros ou enxeñeiros ou xestores ou xestores de proxectos, ou calquera outro compoñente do equipo de desenvolvemento, especialmente en pequenas empresas.

Os manuais de usuario preparados para aplicacións de software ofrecen unha explicación detallada da instalación, configuración e uso da aplicación. É habitual que incorporen imaxes, cunha captura de pantalla da execución do programa.

As seccións dun manual de usuario normalmente inclúen o seguinte contido:

- Portada.
- Páxina de título e páxina de copyright.
- Prefacio, que mostra detalles de documentos relacionados e información sobre como navegar polo manual do usuario.

- Páxinas de contido, onde se mostra un índice co contido do manual.
- Unha guía sobre como usar, polo menos, as funcións básicas da aplicación.
- Unha sección de resolución de problemas, onde se detallan os problemas e as posibles solucións que poden xurdir, xunto con como solucionalos.
- Unha sección de preguntas frecuentes
- Onde atopar máis axuda para usar a aplicación.
- Un glosario e un índice.

Aínda que os manuais de usuario eran orixinalmente documentos en papel, hoxe a maioría dos manuais de usuario distribúense coa aplicación en formato electrónico. Tamén se poden atopar manuais nos sitios web das empresas que os producen.

4.3.2 Guía de referencia

Unha *guía de referencia* é un tipo específico de manual que, a diferenza dun manual de usuario, ofrécenos información detallada sobre as características e o uso dunha determinada aplicación. As guías de referencia adoitan estar destinadas a usuarios experimentados. Nas guías de referencia normalmente mostran unha lista completa de mensaxes de erro e como detectar e recuperarse dos erros detectados.

Un exemplo típico de *manual de referencia* sería a guía de referencia para o uso dun ambiente de desenvolvemento, onde é posible programar en diferentes linguaxes de programación. Para que o usuario poida usar esta aplicación, o manual do usuario indicará os pasos necesarios para crear un novo proxecto, como navegar polas distintas fiestras da aplicación, as diferentes opcións presentadas no menú, xunto coa súa utilidade. Unha guía de referencia, neste caso, sería o manual que detalla a sintaxe, as palabras reservadas, os tipos de enunciados, etc., específicos da linguaxe de programación coa que imos traballar, por exemplo: Java.

O uso máis estendido das guías de referencia está naquelas aplicacións que requiren un coñecemento altamente especializado dunha lista de comandos ou dunha linguaxe de programación. Por exemplo, cando se traballa con contornos de desenvolvemento ou se traballa con aplicacións que permiten a execución de comandos a través dunha consola, é moi común que se distribúa unha guía de referencia xunto co manual do usuario. Esta guía de referencia incorpora a información técnica e detallada dos comandos (xunto coas súas opcións) que se poden empregar na aplicación ou, se se trata dun contorno de

desenvolvemento, a guía de referencia presentará: a sintaxe, as palabras reservadas, os tipos de instrucións, etc., desa linguaxe de programación.

4.3.3 Guías rápidas

Os usuarios dunha aplicación informática adoitan querer documentación nun formato que:

- Dálles unha visión xenérica dá o mesmo, e
- Permítelles usalos o máis rápido posible.

Guías rápidas ofrece unha pequena versión dun manual. A pesar da brevidade dos materiais de referencia rápida, o proceso de pensamento que implica a creación, organización e organización do seu contido non é trivial, xa que cómpre avaliar o contido do manual e decidir sobre a información máis importante que aquela. sabe.

Cando un desenvolvedor inicia un novo proxecto, é necesario avaliar as vantaxes que pode traer, a implementación dunha guía rápida. Unha guía rápida é ideal nos seguintes casos:

- Aplicación de configuración única: se a aplicación só precisa configurarse unha vez durante o proceso de instalación, a aplicación está a funcionar rapidamente. A guía rápida permite ao usuario ou usuario final os coñecementos básicos para usar a aplicación.
- Aplicación con funcionalidade limitada: se unha aplicación ten un número moi limitado de tarefas, non será necesaria unha extensa documentación.
- Documentación moi extensa: se unha aplicación ten un gran volume de documentación, porque é unha aplicación moi complexa, unha guía rápida permitirá a execución das tarefas principais, de xeito que cando un usuario precise executar tarefas máis complexas, pode consultar o manual, ou El axúdaa, pero xa está adestrado para realizar tarefas básicas.
- Usuarios que non poden gastar moito tempo en adestrar, en aplicacións que migran a unha nova versión, etc.

Para desenvolver a guía rápida, tes que decidir que contido inclúes. Se a guía rápida forma parte dunha documentación máis grande, primeiro desenvólvese. Tomando unha visión global do produto, podemos decidir os aspectos máis importantes que poden permitir que un usuario use a aplicación. As tarefas esenciais serán as que incluiremos na guía rápida.

Hai moitos modelos para desenvolver guías rápidas, con todo, cada aplicación requirirá

un deseño específico. Para o deseño gráfico da guía de referencia rápida, teña presente estes catro aspectos: contraste, repetición, aliñamento e proximidade.

O contraste permítenos establecer relacións xerárquicas entre a información, por exemplo, empregando o contraste entre os títulos e o texto, axuda ao usuario a comprender mellor a estrutura do documento. Tamén o contraste axúdanos a chamar a atención. A repetición dun deseño consistente en todo o documento dá integridade a todo o documento. Por exemplo, a repetición dun logotipo, dunha determinada cor, etc.

O aliñamento do texto é necesario, xa que manter a simetría entre parágrafos, imaxes, cabeceiras e outros elementos mantén os ollos relaxados. A proximidade refírese á agrupación de obxectos interrelacionados. Nunha guía rápida, as tarefas importantes deberían agruparse na columna principal, mantendo as tarefas subordinadas na columna lateral.

4.3.4 Manual de instalación

Os manuais de instalación están destinados a administradores e usuarios de sistemas con privilexios de instalación de aplicacións. Os manuais de instalación ofrecen información detallada sobre como instalar a aplicación nun ambiente específico. Estes manuais de instalación conteñen descricións dos ficheiros que compoñen a aplicación, o sistema operativo para o que foi deseñado, os requisitos mínimos de hardware, etc. O manual de instalación tamén debería proporcionar información sobre as posibles modificacións dos ficheiros do sistema anfitrión, a modificación de variables de contorno, a instalación de software adicional para que a aplicación poida funcionar (por exemplo a máquina virtual Java), así como calquera aspecto necesario, que son necesarios para instalar a aplicación nun sistema específico.

Actualmente, o uso de programas de instalación para aplicacións, tanto en sistemas Windows como en sistemas Linux, fai innecesario, na maioría dos casos, a creación de manuais de instalación, razón pola que moitos vendedores de software xa non os producen. Non obstante, para certas aplicacións grandes, especialmente en contornas multiusuario, os manuais de instalación úsanse para axudar aos administradores do sistema a descubrir e resolver problemas coa instalación.