Materiales del curso de programación en Python - Nivel básico

Publicación 0.1

Leonardo J. Caballero G.

26 de September de 2017

Covantec R.L., Santa Cruz de Mara, Mara, Zulia. 4046.
Telf. +58-262-879.18.80 / +58-426-771.35.73 / +58-414-979.80.83
Contactos: covantec.ve@gmail.com - https://coderwall.com/team/covantec

Índice general

Esta es la documentación para el capacitación **"Programación en Python - Nivel básico"**.

Sobre esta capacitación

Para dominar Python se tiene pensado como un entrenamiento de 2 a 3 días para las personas que son nuevas en Python o los que quieren aprender acerca de las actuales mejores prácticas del desarrollo en Python.
La planificación de este capacitación se estima en:

- Una capacitación de **nivel básico** (*2 a 3 días*) que cubre los *diez (10) capítulos*.

Tabla de contenidos:

CAPÍTULO 1

Introducción a Python

- ¿Qué es Python?[1].
- Características[2].
- ¿Por qué Python?[3].

Recursos Web de Python

- Pagina Web Oficial[4].
- Descarga Python[5].

Documentación de Python

- Documentación oficial de Python 2.7[6].
- Tutorial de Python 2.7[7].

Manuales de Python

- Python para programadores con experiencia[8].
- Introducción a la programación con Python[9].
- Python Tutorial[10].

Soporte comunitario de Python

- Comunidad Python Venezuela[11].

[1] http://es.wikipedia.org/wiki/Python
[2] http://es.wikipedia.org/wiki/Python#Caracter.C3.ADsticas_y_paradigmas
[3] http://es.wikipedia.org/wiki/Python#Filosof.C3.ADa
[4] https://www.python.org/
[5] https://www.python.org/downloads/
[6] https://docs.python.org/2.7/
[7] http://docs.python.org.ar/tutorial/2/contenido.html
[8] http://es.diveintopython.net/odbchelper_divein.html
[9] http://www.mclibre.org/consultar/python/
[10] http://www.tutorialspoint.com/python/index.htm
[11] http://www.python.org.ve/

- Comunidad Python Argentina[12].

Vídeo tutorial

- Tutorial Python 1 - Introducción al Lenguaje de Programación[13].

[12] http://www.python.org.ar/
[13] https://www.youtube.com/watch?v=CjmzDHMHxwU

Instalando Python

- Instalando Python en Windows[1].
- Instalando Python en una Mac[2].

Recursos de Descargas de Python

- Descarga Python[3].
- PyPI - the Python Package Index[4].

Vídeo tutorial

- Tutorial Python 2 - Instalación[5].

[1] https://www.youtube.com/watch?v=VTykmP-a2KY
[2] http://es.wikibooks.org/wiki/Inmersi %C3 %B3n_en_Python/Instalaci %C3 %B3n_de_Python/Python_en_Mac_OS_X
[3] https://www.python.org/downloads/
[4] https://pypi.python.org/pypiorg/wiki/Inmersi %C3 %B3n_en_Python/Instalaci %C3 %B3n_de_Python/Python_en_Mac_OS_X
[5] https://www.youtube.com/watch?v=VTykmP-a2KY

CAPÍTULO 3

Inmersión al modo interactivo de Python

Sobre este artículo

Autor(es) Leonardo J. Caballero G.
Correo(s) leonardocaballero@gmail.com[a]
Compatible con Python 2.4 o versiones superiores
Fecha 31 de Diciembre de 2013

[a] leonardocaballero@gmail.com

Descripción general

Este articulo se basa en el documento Una pequeña inmersión al modo interactivo de Python[1] generado por la fundación Cenditel[2] y la idea principal de este tutorial es para alguien que **NUNCA** ha trabajando con el interprete de Python[3] pueda tener un primer acercamiento **SIN PROGRAMAR**, solamente con conocer el uso del interprete y sus comandos básicos.

Características de Python

- Es un lenguaje de programación multiparadigma[4].
- Soporta orientación a objetos[5], programación imperativa[6] y, en menor medida, programación funcional[7].
- Es un lenguaje interpretado[8], usa tipado dinámico[9], es fuertemente tipado[10] y es multiplataforma[11].

Introspección en Python

Según el libro Inmersión en Python ...Como usted sabe, todo en Python es un objeto[12], y la introspección es código que examina como objetos otros módulos y funciones en memoria, obtiene información sobre ellos y los maneja.

[1] http://plataforma.cenditel.gob.ve/wiki/Plone %3AUnaPequenaInmersionPython
[2] https://twitter.com/cenditel
[3] http://www.python.org/
[4] http://es.wikipedia.org/wiki/Lenguaje_de_programaci %C3 %B3n_multiparadigma
[5] http://es.wikipedia.org/wiki/Programaci %C3 %B3n_orientada_a_objetos
[6] http://es.wikipedia.org/wiki/Programaci %C3 %B3n_imperativa
[7] http://es.wikipedia.org/wiki/Programaci %C3 %B3n_funcional
[8] http://es.wikipedia.org/wiki/Lenguaje_interpretado
[9] http://es.wikipedia.org/wiki/Tipado_din %C3 %A1mico
[10] http://es.wikipedia.org/wiki/Lenguaje_de_programaci %C3 %B3n_fuertemente_tipado
[11] http://es.wikipedia.org/wiki/Multiplataforma
[12] http://es.diveintopython.org/odbchelper_objects.html

De paso, usted podrá definir las funciones sin nombre, las llamará a funciones con argumentos sin orden, y podrá hacer referencia a funciones cuyos nombres desconocemos.

Python a través de su interprete

Es importante conocer Python a través de su interprete debido a varios factores:

- Conocer las clases, sus funciones y atributos propios, a través de la introspección del lenguaje.
- Disponibilidad de consultar la documentación del lenguaje desde el interprete, por mucho tiempo no estaba disponible documentación tipo Javadoc[13] o diagramas de clases[14] del propio lenguaje por lo cual muchas programadores **Python** se acostumbraron a estudiar su código de esta forma, así que le recomiendo que use el interprete `python` para eso.
- Hoy en día existente herramientas que te permiten generar documentación desde los códigos fuentes Python como Sphinx[15].

La forma mas fácil es iniciar tu relación con Python simplemente ejecutando el comando `python` de la siguiente forma:

```
$ python
Python 2.5.2 (r252:60911, Jan  4 2009, 17:40:26)
[GCC 4.3.2] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Pidiendo la ayudar del interprete de Python

```
>>> help
Type help() for interactive help, or help(object) for help about object.
>>> help()

Welcome to Python 2.5!  This is the online help utility.

If this is your first time using Python, you should definitely check
out the tutorial on the Internet at http://www.python.org/doc/tut/.

Enter the name of any module, keyword, or topic to get help on
writing Python programs and using Python modules. To quit this help
utility and return to the interpreter, just type "quit".

To get a list of available modules, keywords, or topics, type
"modules", "keywords", or "topics".  Each module also comes with
a one-line summary of what it does; to list the modules whose
summaries contain a given word such as "spam", type "modules spam".
help>
```

Para ejecutar la ayuda disponible sobre la sintaxis Python ejecute el siguiente comando:

```
help> modules

  Please wait a moment while I gather a list of all available
  modules...

  /usr/lib/python2.5/site-packages/apt/__init__.py:18: FutureWarning:
  apt API not stable yet
    warnings.warn("apt API not stable yet", FutureWarning)
  Data Dir: /usr/share/colorblind
```

[13] http://es.wikipedia.org/wiki/Javadoc
[14] http://es.wikipedia.org/wiki/Diagrama_de_clases
[15] http://en.wikipedia.org/wiki/Sphinx_%28documentation_generator%29

```
Data Dir: /usr/share/gnome-applets/invest-applet
Alacarte            _ctypes             gksu                platform
AppInstall          _ctypes_test        gksu2               plistlib
ArgImagePlugin      _curses             glchess             popen2
ArrayPrinter        _curses_panel       glob                poplib
BaseHTTPServer      _dbus_bindings      gmenu               posix
Bastion             _dbus_glib_bindings gnome               posixfile
BdfFontFile         _elementtree        gnome_sudoku        posixpath
BeautifulSoup       _functools          gnomeapplet         pprint
BeautifulSoupTests  _hashlib            gnomecanvas         profile
BmpImagePlugin      _heapq              gnomedesktop
pspersistence
BufrStubImagePlugin _hotshot            gnomekeyring        pstats
CDROM               _imaging            gnomeprint          pty
CGIHTTPServer       _imagingft          gnomevfs            pwd
Canvas              _imagingmath        gobject             pxssh
ConfigParser        _ldap               gopherlib
py_compile
ContainerIO         _locale             grp                 pyatspi
Cookie              _lsprof             gst                 pyclbr
Crypto              _multibytecodec     gtk                 pydoc
CurImagePlugin      _mysql              gtkhtml2            pyexpat
DLFCN               _mysql_exceptions   gtkmozembed         pygst
DcxImagePlugin      _numpy              gtksourceview       pygtk
Dialog              _random             gtksourceview2      pynotify
DocXMLRPCServer     _socket             gtkspell
pythonloader
EpsImagePlugin      _sqlite3            gtkunixprint
pythonscript
ExifTags            _sre                gtop                pyuno
FileDialog          _ssl                gzip                quopri
FitsStubImagePlugin _strptime           hashlib             random
FixTk               _struct             heapq               re
FliImagePlugin      _symtable           hitcount            readline
FontFile            _testcapi           hmac                repr
FpxImagePlugin      _threading_local    hotshot             resource
Ft                  _types              hpmudext            rexec
GMenuSimpleEditor   _weakref            htmlentitydefs      rfc822
GbrImagePlugin      aifc                htmllib
rlcompleter
GdImageFile         anydbm              httplib
robotparser
GifImagePlugin      apt                 ibrowse             rsvg
GimpGradientFile    apt_inst            idlelib             runpy
GimpPaletteFile     apt_pkg             igrid               scanext
GribStubImagePlugin aptsources          ihooks              sched
HTMLParser          argparse            imaplib             select
Hdf5StubImagePlugin array               imghdr
serpentine
IN                  arrayfns            imp                 sets
IPy                 astyle              imputil
setuptools
IPython             asynchat            inspect             sexy
IcnsImagePlugin     asyncore            invest              sgmllib
IcoImagePlugin      atexit              ipipe               sha
ImImagePlugin       atk                 ipy_app_completers  shelve
Image               atom                ipy_autoreload      shlex
ImageChops          audiodev            ipy_bzr             shutil
```

```
ImageColor          audioop             ipy_completers      signal
ImageDraw           base64              ipy_constants       site
ImageDraw2          bdb                 ipy_defaults
sitecustomize
ImageEnhance        binascii            ipy_editors         smtpd
ImageFile           binhex              ipy_exportdb        smtplib
ImageFileIO         bisect              ipy_extutil         sndhdr
ImageFilter         bonobo              ipy_fsops           socket
ImageFont           brlapi              ipy_gnuglobal       spwd
ImageGL             bsddb               ipy_greedycompleter sqlite3
ImageGrab           bugbuddy            ipy_jot             sqlobject
ImageMath           bz2                 ipy_kitcfg          sre
ImageMode           cPickle             ipy_legacy
sre_compile
ImageOps            cProfile            ipy_leo
sre_constants
ImagePalette        cStringIO           ipy_lookfor         sre_parse
ImagePath           cairo               ipy_p4              stat
ImageQt             calendar            ipy_profile_doctest statvfs
ImageSequence       cgi                 ipy_profile_none    string
ImageStat           cgitb               ipy_profile_scipy   stringold
ImageTransform      chunk               ipy_profile_sh
stringprep
ImageWin            clearcmd            ipy_profile_zope    strop
ImtImagePlugin      cmath               ipy_pydb            struct
InterpreterExec     cmd                 ipy_rehashdir
subprocess
InterpreterPasteInput code                ipy_render          sunau
IptcImagePlugin     codecs              ipy_server          sunaudio
JpegImagePlugin     codeop              ipy_signals         svn
McIdasImagePlugin   collections         ipy_stock_completers symbol
MicImagePlugin      colorblind          ipy_system_conf     symtable
MimeWriter          colorsys            ipy_traits_completer sys
MpegImagePlugin     commands            ipy_vimserver       syslog
MspImagePlugin      compileall          ipy_which           tabnanny
MySQLdb             compiler            ipy_winpdb          tarfile
Numeric             configobj           ipy_workdir         telnetlib
Numeric_headers     constants           itertools           tempfile
ORBit               contextlib          jobctrl
templatetags
OggConvert          cookielib           keyword
terminatorlib
OleFileIO           copy                ldap                termios
PIL                 copy_reg            ldapurl             test
PSDraw              crypt               ldif                textwrap
PaletteFile         csv                 ledit               this
PalmImagePlugin     ctypes              libsvn              thread
PcdImagePlugin      cups                libxml2             threading
PcfFontFile         cupsext             libxml2mod          time
PcxImagePlugin      cupsutils           linecache           timeit
PdfImagePlugin      curses              linuxaudiodev
tkColorChooser
PhysicalQInput      datetime            locale
tkCommonDialog
PhysicalQInteractive dbhash              logging
tkFileDialog
PixarImagePlugin    dbm                 macpath             tkFont
PngImagePlugin      dbus                macurl2path
```

```
tkMessageBox
PpmImagePlugin      dbus_bindings       mailbox
tkSimpleDialog
Precision           debconf             mailcap             toaiff
PsdImagePlugin      decimal             markupbase          token
Queue               deskbar             marshal             tokenize
ScrolledText        difflib             math                totem
SgiImagePlugin      dircache            md5                 trace
SimpleDialog        dis                 mediaprofiles       traceback
SimpleHTTPServer    distutils           metacity            tty
SimpleXMLRPCServer  django              mhlib               turtle
SocketServer        doctest             mimetools           types
SpiderImagePlugin   drv_libxml2         mimetypes           umath
StringIO            dsextras            mimify
unicodedata
SunImagePlugin      dsml                mmap                unittest
TYPES               dumbdbm             modulefinder        uno
TarIO               dummy_thread        multiarray          unohelper
TgaImagePlugin      dummy_threading     multifile           urllib
TiffImagePlugin     easy_install        mutex               urllib2
TiffTags            egg                 nautilusburn        urlparse
Tix                 email               netrc               user
Tkconstants         encodings           new                 uu
Tkdnd               envbuilder          nis                 uuid
Tkinter             envpersist          nntplib             validate
UserArray           errno               ntpath
virtualenv
UserDict            evolution           nturl2path
virtualenv_support
UserList            exceptions          numeric_formats     vte
UserString          ext_rescapture      numeric_version     warnings
WalImageFile        fcntl               opcode              wave
WmfImagePlugin      fdpexpect           operator            weakref
XVThumbImagePlugin  filecmp             optparse
webbrowser
XbmImagePlugin      fileinput           orca                whichdb
XpmImagePlugin      fnmatch             os                  win32clip
_LWPCookieJar       foomatic            os2emxpath          wnck
_MozillaCookieJar   formatter           ossaudiodev         wsgiref
__builtin__         formencode          pango               xdg
__future__          fpformat            pangocairo          xdrlib
_ast                ftplib              parser              xml
_bisect             functools           pcardext            xmllib
_bsddb              gc                  pdb                 xmlrpclib
_codecs             gconf               pexpect             xxsubtype
_codecs_cn          gda                 pickle              z3c
_codecs_hk          gdata               pickleshare         zc
_codecs_iso2022     gdbm                pickletools         zipfile
_codecs_jp          gdl                 pip                 zipimport
_codecs_kr          getopt              pipes               zlib
_codecs_tw          getpass             pkg_resources       zopeskel
_csv                gettext             pkgutil

Enter any module name to get more help.  Or, type "modules spam" to
search for modules whose descriptions contain the word "spam".

help> os
Help on module os:
```

```
NAME
    os - OS routines for Mac, NT, or Posix depending on what
    system we're on.

FILE
    /usr/lib/python2.5/os.py

MODULE DOCS
    http://www.python.org/doc/current/lib/module-os.html

DESCRIPTION
    This exports:
      - all functions from posix, nt, os2, mac, or ce, e.g. unlink, stat, etc.
      - os.path is one of the modules posixpath, ntpath, or macpath
      - os.name is 'posix', 'nt', 'os2', 'mac', 'ce' or 'riscos'
      - os.curdir is a string representing the current directory ('.' or ':')
      - os.pardir is a string representing the parent directory ('..' or '::')
      - os.sep is the (or a most common) pathname separator ('/' or ':' or '\\')
      - os.extsep is the extension separator ('.' or '/')
      - os.altsep is the alternate pathname separator (None or '/')
      - os.pathsep is the component separator used in $PATH etc
      - os.linesep is the line separator in text files ('\r' or '\n' or '\r\n')
      - os.defpath is the default search path for executables
      - os.devnull is the file path of the null device ('/dev/null', etc.)

    Programs that import and use 'os' stand a better chance of
    being portable between different platforms.  Of course,
    they must then only use functions that are defined by all
    platforms (e.g., unlink and opendir), and leave all pathname
    manipulation to os.path
:
```

Entonces presione la convinación de tecla **Crtl+d** para salir de la ayuda.

Luego realice la importación de la librería del estándar[16] Python llamada `os`

```
>>> import os
>>>
```

Previamente importada la librería usted puede usar el comando `dir` para listar o descubrir que atributos, métodos de la clase están disponibles con la importación

```
>>> dir(os)
['EX_CANTCREAT', 'EX_CONFIG', 'EX_DATAERR', 'EX_IOERR', 'EX_NOHOST',
'EX_NOINPUT', 'EX_NOPERM', 'EX_NOUSER', 'EX_OK', 'EX_OSERR', 'EX_OSFILE',
'EX_PROTOCOL', 'EX_SOFTWARE', 'EX_TEMPFAIL', 'EX_UNAVAILABLE',
'EX_USAGE', 'F_OK', 'NGROUPS_MAX', 'O_APPEND', 'O_CREAT', 'O_DIRECT',
'O_DIRECTORY', 'O_DSYNC', 'O_EXCL', 'O_LARGEFILE', 'O_NDELAY',
'O_NOCTTY', 'O_NOFOLLOW', 'O_NONBLOCK', 'O_RDONLY', 'O_RDWR', 'O_RSYNC',
'O_SYNC', 'O_TRUNC', 'O_WRONLY', 'P_NOWAIT', 'P_NOWAITO', 'P_WAIT',
'R_OK', 'SEEK_CUR', 'SEEK_END', 'SEEK_SET', 'TMP_MAX', 'UserDict',
'WCONTINUED', 'WCOREDUMP', 'WEXITSTATUS', 'WIFCONTINUED', 'WIFEXITED',
'WIFSIGNALED', 'WIFSTOPPED', 'WNOHANG', 'WSTOPSIG', 'WTERMSIG',
'WUNTRACED', 'W_OK', 'X_OK', '_Environ', '__all__', '__builtins__',
'__doc__', '__file__', '__name__', '_copy_reg', '_execvpe', '_exists',
'_exit', '_get_exports_list', '_make_stat_result',
'_make_statvfs_result', '_pickle_stat_result', '_pickle_statvfs_result',
'_spawnvef', 'abort', 'access', 'altsep', 'chdir', 'chmod', 'chown',
```

[16] http://pyspanishdoc.sourceforge.net/tut/node12.html

```
'chroot', 'close', 'confstr', 'confstr_names', 'ctermid', 'curdir',
'defpath', 'devnull', 'dup', 'dup2', 'environ', 'errno', 'error',
'execl', 'execle', 'execlp', 'execlpe', 'execv', 'execve', 'execvp',
'execvpe', 'extsep', 'fchdir', 'fdatasync', 'fdopen', 'fork', 'forkpty',
'fpathconf', 'fstat', 'fstatvfs', 'fsync', 'ftruncate', 'getcwd',
'getcwdu', 'getegid', 'getenv', 'geteuid', 'getgid', 'getgroups',
'getloadavg', 'getlogin', 'getpgid', 'getpgrp', 'getpid', 'getppid',
'getsid', 'getuid', 'isatty', 'kill', 'killpg', 'lchown', 'linesep',
'link', 'listdir', 'lseek', 'lstat', 'major', 'makedev', 'makedirs',
'minor', 'mkdir', 'mkfifo', 'mknod', 'name', 'nice', 'open', 'openpty',
'pardir', 'path', 'pathconf', 'pathconf_names', 'pathsep', 'pipe',
'popen', 'popen2', 'popen3', 'popen4', 'putenv', 'read', 'readlink',
'remove', 'removedirs', 'rename', 'renames', 'rmdir', 'sep', 'setegid',
'seteuid', 'setgid', 'setgroups', 'setpgid', 'setpgrp', 'setregid',
'setreuid', 'setsid', 'setuid', 'spawnl', 'spawnle', 'spawnlp',
'spawnlpe', 'spawnv', 'spawnve', 'spawnvp', 'spawnvpe', 'stat',
'stat_float_times', 'stat_result', 'statvfs', 'statvfs_result',
'strerror', 'symlink', 'sys', 'sysconf', 'sysconf_names', 'system',
'tcgetpgrp', 'tcsetpgrp', 'tempnam', 'times', 'tmpfile', 'tmpnam',
'ttyname', 'umask', 'uname', 'unlink', 'unsetenv', 'urandom', 'utime',
'wait', 'wait3', 'wait4', 'waitpid', 'walk', 'write']
>>>
```

Otro ejemplo de uso, es poder usar el método `file` para determinar la ubicación de la librería importada de la siguiente forma:

```
>>> os.__file__
'/usr/lib/python2.5/os.pyc'
>>>
```

También puede consultar la documentación de la librería `os` ejecutando el siguiente comando:

```
>>> os.__doc__
"OS routines for Mac, NT, or Posix depending on what system we're
on.\n\nThis exports:\n  - all functions from posix, nt, os2, mac, or ce,
e.g. unlink, stat, etc.\n  - os.path is one of the modules posixpath,
ntpath, or macpath\n  - os.name is 'posix', 'nt', 'os2', 'mac', 'ce' or
'riscos'\n  - os.curdir is a string representing the current directory
('.' or ':')\n  - os.pardir is a string representing the parent directory
('..' or '::')\n  - os.sep is the (or a most common) pathname separator
('/' or ':' or '\\\\')\n  - os.extsep is the extension separator ('.' or
'/')\n  - os.altsep is the alternate pathname separator (None or '/')\n
- os.pathsep is the component separator used in $PATH etc\n  - os.linesep
is the line separator in text files ('\\r' or '\\n' or '\\r\\n')\n  -
os.defpath is the default search path for executables\n  - os.devnull is
the file path of the null device ('/dev/null', etc.)\n\nPrograms that
import and use 'os' stand a better chance of being\nportable between
different platforms.  Of course, they must then\nonly use functions that
are defined by all platforms (e.g., unlink\nand opendir), and leave all
pathname manipulation to os.path\n(e.g., split and join).\n"
>>>
```

Ejecute el comando exit() para salir del interprete...

```
>>> exit()
```

Interprete interactivo de Python

Para mejorar la experiencia con el interprete Python le sugerimos instalar el programa IPython, según su documentación:

Según Wikipedia

> "IPython es un shell interactivo que añade funcionalidades extra al modo interactivo[17] incluido con Python, como resaltado de líneas y errores mediante colores, una sintaxis adicional para el shell, autocompletado mediante tabulador de variables, módulos y atributos; entre otras funcionalidades. Es un componente del paquete SciPy[18]."

Para mayor información visite su página principal de ipython[19] y si necesita instalar este programa ejecute el siguiente comando:

```
# aptitude install ipython python-pip
```

Luego cierra sesión de **root** y vuelve al usuario y sustituya el comando `python` por `ipython` de la siguiente forma:

```
$  ipython
Python 2.5.2 (r252:60911, Jan 24 2010, 17:44:40)
Type "copyright", "credits" or "license" for more information.

IPython 0.8.4 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ?? prints
more.

In [1]:
```

Un ejemplo de uso del comando `help` es consultar la ayuda del comando `dir` y se ejecuta de la siguiente forma:

```
In [1]: help(dir)
Help on built-in function dir in module __builtin__:

dir(...)
    dir([object]) -> list of strings

    Return an alphabetized list of names comprising (some of) the
    attributes of the given object, and of attributes reachable
    from it:

    No argument:  the names in the current scope.
    Module object:  the module attributes.
    Type or class object:  its attributes, and recursively the
    attributes of its bases.
    Otherwise:  its attributes, its class's attributes, and
    recursively the attributes of its class's base classes.
```

Entonces presione la tecla **q** para salir de la ayuda

De nuevo realice la importación de la librería del estándar Python llamada `os`

```
In [2]: import os
```

También consultar los detalles acerca del 'objeto' para esto use como ejemplo la librería `os` ejecutando el siguiente comando:

```
In [2]: os?
Type:           module
Base Class:     <type 'module'>
String Form:    <module 'os' from '/usr/lib/python2.5/os.pyc'>
Namespace:      Interactive
```

[17] http://es.wikipedia.org/wiki/Python#Modo_interactivo
[18] http://en.wikipedia.org/wiki/SciPy
[19] http://ipython.scipy.org/

```
File:           /usr/lib/python2.5/os.py
Docstring:
    OS routines for Mac, NT, or Posix depending on what system
    we're on.

    This exports:
      - all functions from posix, nt, os2, mac, or ce, e.g. unlink, stat, etc.
      - os.path is one of the modules posixpath, ntpath, or macpath
      - os.name is 'posix', 'nt', 'os2', 'mac', 'ce' or 'riscos'
      - os.curdir is a string representing the current directory ('.' or ':')
      - os.pardir is a string representing the parent directory ('..' or '::')
      - os.sep is the (or a most common) pathname separator ('/' or ':' or '\\')
      - os.extsep is the extension separator ('.' or '/')
      - os.altsep is the alternate pathname separator (None or '/')
      - os.pathsep is the component separator used in $PATH etc
      - os.linesep is the line separator in text files ('\r' or '\n' or '\r\n')
      - os.defpath is the default search path for executables
      - os.devnull is the file path of the null device ('/dev/null', etc.)

    Programs that import and use 'os' stand a better chance of
    being portable between different platforms.  Of course,
    they must then only use functions that are defined by all
    platforms (e.g., unlink and opendir), and leave all pathname
    manipulation to os.path (e.g., split and join).
```

Escriba la librería *os.* y luego escribe dos **underscore** y presione *dos veces la tecla tabular* para usar la autocompletado del interprete al estilo de completación de lineas de comandos[20] en el shell UNIX/Linux para ayudar a la introspección del lenguaje y sus librerías.

```
In [3]: os.__
os.__all__           os.__class__         os.__dict__
os.__file__          os.__hash__          os.__name__
os.__reduce__        os.__repr__          os.__str__
os.__builtins__      os.__delattr__       os.__doc__
os.__getattribute__  os.__init__          os.__new__
os.__reduce_ex__     os.__setattr__
```

De nuevo ejecute el método `file` para determinar la ubicación de la librería importada

```
In [4]: os.__file__
Out[4]: '/usr/lib/python2.5/os.pyc'
```

También puede consultar la documentación de la librería `os` de la siguiente forma:

```
In [5]: os.__doc__
Out[5]: "OS routines for Mac, NT, or Posix depending on what system
we're on.\n\nThis exports:\n  - all functions from posix, nt, os2, mac,
or ce, e.g. unlink, stat, etc.\n  - os.path is one of the modules
posixpath, ntpath, or macpath\n  - os.name is 'posix', 'nt', 'os2',
'mac', 'ce' or 'riscos'\n  - os.curdir is a string representing the
current directory ('.' or ':')\n  - os.pardir is a string representing
the parent directory ('..' or '::')\n  - os.sep is the (or a most common)
pathname separator ('/' or ':' or '\\\\')\n  - os.extsep is the extension
separator ('.' or '/')\n  - os.altsep is the alternate pathname separator
(None or '/')\n  - os.pathsep is the component separator used in $PATH
etc\n  - os.linesep is the line separator in text files ('\\r' or '\\n'
or '\\r\\n')\n  - os.defpath is the default search path for executables\n
- os.devnull is the file path of the null device ('/dev/null',
etc.)\n\nPrograms that import and use 'os' stand a better chance of
being\nportable between different platforms.  Of course, they must
```

[20] http://en.wikipedia.org/wiki/Command_line_completion

```
then\nonly use functions that are defined by all platforms (e.g.,
unlink\nand opendir), and leave all pathname manipulation to
os.path\n(e.g., split and join).\n"
```

Otro ejemplo es imprimir el **nombre de la clase** con el siguiente comando:

```
In [6]: os.__name__
Out[6]: 'os'
```

Y otra forma de consultar la documentación de la librería `os` es ejecutando el siguiente comando:

```
In [7]: help(os)
Help on module os:

NAME
    os - OS routines for Mac, NT, or Posix depending on what
    system we're on.

FILE
    /usr/lib/python2.5/os.py

MODULE DOCS
    http://www.python.org/doc/current/lib/module-os.html

DESCRIPTION
    This exports:
      - all functions from posix, nt, os2, mac, or ce, e.g. unlink, stat, etc.
      - os.path is one of the modules posixpath, ntpath, or macpath
      - os.name is 'posix', 'nt', 'os2', 'mac', 'ce' or 'riscos'
      - os.curdir is a string representing the current directory ('.' or ':')
      - os.pardir is a string representing the parent directory ('..' or '::')
      - os.sep is the (or a most common) pathname separator ('/' or ':' or '\\')
      - os.extsep is the extension separator ('.' or '/')
      - os.altsep is the alternate pathname separator (None or '/')
      - os.pathsep is the component separator used in $PATH etc
      - os.linesep is the line separator in text files ('\r' or '\n' or '\r\n')
      - os.defpath is the default search path for executables
      - os.devnull is the file path of the null device ('/dev/null', etc.)

    Programs that import and use 'os' stand a better chance of
    being portable between different platforms.  Of course,
    they must then only use functions that are defined by all
    platforms (e.g., unlink and opendir), and leave all pathname
    manipulation to os.path
:
```

Entonces presione la tecla **q** para salir de la ayuda

Y para borrar la sesión con el IPython ejecute el siguiente comando:

```
In [8]: exit()
Do you really want to exit ([y]/n)? y
```

Interprete interactivo con el paquete bpython

Alternativamente puedes usar el *paquete bpython* que mejora aun mas la experiencia de trabajo con el paquete *ipython*

Para mayor información visite su página principal de bpython[21] y si necesita instalar este programa ejecute el siguiente comando:

[21] http://bpython-interpreter.org/

```
# pip install bpython
```

Luego cierra sesión de **root** y vuelve al usuario y sustituya el comando `python` por `ipython` de la siguiente forma:

```
$  bpython
```

Dentro de interprete Python puede apreciar que ofrece otra forma de presentar la documentación y la estructura del lenguaje, con los siguientes comandos de ejemplos:

```
>>> print 'Hola mundo'
Hola mundo
>>> for item in xrange(
+---------------------------------------------------------------------+
| xrange: ([start, ] stop[, step])                                    |
| xrange([start,] stop[, step]) -> xrange object                      |
|                                                                     |
| Like range(), but instead of returning a list, returns an object that |
| generates the numbers in the range on demand.  For looping, this is   |
| slightly faster than range() and more memory efficient.             |
+---------------------------------------------------------------------+

 <C-r> Rewind  <C-s> Save  <F8> Pastebin  <F9> Pager  <F2> Show Source
```

Conclusiones

Como puede apreciar este tutorial no le enseña a programar sino a simplemente aprender a conocer como manejarse en el modo interactivo de Python/IPython con el fin de conocer a través de la introspección del lenguaje, las librerías estándar / propias de Python que tienes instalado en tu sistema.

Ver también:

- Python[22].
- Inmersión en Python[23].
- Guía de aprendizaje de Python[24].
- La librería estándar de Python[25].
- Guide to Python introspection[26].

Referencias

- Una pequeña inmersión al modo interactivo de Python[27] de la fundación Cenditel.

[22] http://www.python.org/
[23] http://es.diveintopython.org/
[24] http://pyspanishdoc.sourceforge.net/tut/tut.html
[25] http://pyspanishdoc.sourceforge.net/tut/node12.html
[26] http://www.ibm.com/developerworks/linux/library/l-pyint.html
[27] http://plataforma.cenditel.gob.ve/wiki/Plone %3AUnaPequenaInmersionPython

CAPÍTULO 4

Escribiendo su primer programa - ¡Hola, mundo!

En informática, un programa **Hola mundo** es el que imprime el texto *«¡Hola, mundo!»* en un dispositivo de visualización, en la mayoría de los casos una pantalla de monitor. Este programa suele ser usado como introducción al estudio de un lenguaje de programación, siendo un primer ejercicio típico, y se lo considera fundamental desde el punto de vista didáctico.

El *Hola Mundo* se caracteriza por su sencillez, especialmente cuando se ejecuta en una interfaz de línea de comandos. En interfaces gráficas la creación de este programa requiere de más pasos.

El programa *Hola Mundo* también puede ser útil como prueba de configuración para asegurar que el compilador, el entorno de desarrollo y el entorno de ejecución estén instalados correctamente y funcionando.

Ingresando y ejecutando un programa en Linux

Cree un directorio llamado `proyectos` el `home` de su usuario y dentro de este, cree un archivo de texto plano con el siguiente nombre `holamundo.py` y escriba la siguiente sintaxis:

Python 2.x:

```
print "Hola Mundo"
```

Python 3.x:

```
print("Hola Mundo");
```

Luego ejecute desde la consola de comando el siguiente comando:

```
python $HOME/proyectos/holamundo.py
```

Usted debe ver la línea *Hola Mundo*.

Enhorabuena, ha ejecutar tu primer programa Python.

Ingresando y ejecutando un programa en Windows

Cree un directorio llamado `proyectos` la unidad `C:\` y dentro de este, cree un archivo de texto plano con el siguiente nombre `holamundo.py` y escriba la siguiente sintaxis:

```
print "Hola Mundo"
```

Luego ejecute desde la consola de `MS-DOS` el siguiente comando:

```
C:\Python27\python C:\proyectos\holamundo.py
```

Usted debe ver la línea *Hola Mundo*.

Enhorabuena, ha ejecutar tu primer programa Python.

Ingresando y ejecutando un programa en OSX

1. Haga clic en `Archivo` y luego la nueva Ventana del `Finder`.
2. Haga clic en `Documentos`.
3. Haga clic en `Archivo` y luego en `Nueva carpeta`.
4. Llame a la carpeta `proyectos`.
5. Usted va a almacenar todos los programas relacionados con la clase allí.
6. Haga clic en `Aplicaciones` y, a continuación `TextEdit`.
7. Haga clic en `TextEdit` en la barra de menú y seleccione `Preferencias`.
8. Seleccione `Texto plano`.
9. En el vacío `TextEdit` tipo de ventana en el siguiente programa, tal y como se da:

```
print "Hola Mundo"
```

1. Desde el archivo de menú en TextEdit.
2. Haga clic en `Guardar como`.
3. En el campo Guardar como: escriba `holamundo.py`.
4. Seleccione `Documentos` y la carpeta de archivos `proyectos`.
5. Haga clic en `Guardar`.

Funcionamiento de su Primer Programa

1. Seleccione `Aplicaciones`, a continuación, `Utilidades y Terminal`.
2. En la ventana `Terminal` ejecute `ls` y presione la tecla `Enter`. Se debe dar una lista de todas las carpetas de nivel superior. Usted debe ver la carpeta de `Documentos`.
3. Ejecute `cd Documentos` y presione `Enter`.
4. Ejecute `ls` y presione `Enter` y debería ver la carpeta `proyectos`.
5. Ejecute `cd proyectos` y presione `Enter`.
6. Ejecute `ls` y presione `Enter` y usted debería ver el archivo `holamundo.py`.
7. Para ejecutar el programa, escriba el siguiente comando `python holamundo.py` y presione `Enter`.
8. Usted debe ver la línea *Hola Mundo*.

Enhorabuena, ha ejecutar tu primer programa Python.

Vídeo tutorial

- Tutorial Python 3 - Hola Mundo[1].

Referencias

- Getting Started with Python[2]

[1] https://www.youtube.com/watch?v=OtJEj7N9T6k
[2] http://www.cs.utexas.edu/ mitra/bytes/start.html

CAPÍTULO 5

Tipos de datos básicos y variables Python

En Python tenemos como tipos de datos simples números: enteros, de coma flotante y complejos, como pueden ser 3, 15.57 o 7 + 5j; cadenas de texto, como "Hola Mundo" y valores booleanos: `True` (cierto) y `False` (falso).

Vamos a crear un par de variables a modo de ejemplo. Una de tipo cadena y una de tipo entero:

```
# esto es una cadena
c = "Hola Mundo"

# y esto es un entero
e = 23

# podemos comprobarlo con la función type
type(c)
type(e)
```

Como puede ver en Python, a diferencia de muchos otros lenguajes, no se declara el tipo de la variable al crearla. En Java, por ejemplo, escribiríamos:

```
String c = "Hola Mundo";
int e = 23;
```

También nos ha servido nuestro pequeño ejemplo para presentaros los comentarios inline en Python: cadenas de texto que comienzan con el carácter '#' y que Python ignora totalmente. Hay más tipos de comentarios, de los que hablaremos más adelante.

Tipos de Enteros

Números

Como decíamos, en Python se pueden representar números enteros, reales y complejos.

Enteros

Los números enteros son aquellos que no tienen decimales, tanto positivos como negativos (además del cero). En Python se pueden representar mediante el tipo `int` (de integer, entero) o el tipo `long` (largo). La única diferencia es que el tipo long permite almacenar números más grandes. Es aconsejable no utilizar el tipo `long` a menos que sea necesario, para no malgastar memoria.

El tipo `int` de Python se implementa a bajo nivel mediante un tipo `long` de C. Y dado que Python utiliza C por debajo, como C, y a diferencia de Java, el rango de los valores que puede representar depende de la plataforma. En la mayor parte de las máquinas el `long` de C se almacena utilizando 32 bits, es decir, mediante el uso de una variable de tipo `int` de Python podemos almacenar números de -2^{31} a $2^{31} - 1$, o lo que es lo mismo, de -2.147.483.648 a 2.147.483.647. En plataformas de 64 bits, el rango es de -9.223.372.036.854.775.808 hasta 9.223.372.036.854.775.807.

El tipo `long` de Python permite almacenar números de cualquier precisión, limitado por la memoria disponible en la máquina.

Al asignar un número a una variable esta pasará a tener tipo `int`, a menos que el número sea tan grande como para requerir el uso del tipo `long`.

```
# type(entero) daría int
entero = 23
```

También podemos indicar a Python que un número se almacene usando long añadiendo una L al final:

```
# type(entero) daría long
entero = 23L
```

El literal que se asigna a la variable también se puede expresar como un `octal`, anteponiendo un cero:

```
# 027 octal = 23 en base 10
entero = 027
```

o bien en hexadecimal, anteponiendo un 0x:

```
# 0x17 hexadecimal = 23 en base 10
entero = 0x17
```

Reales

Los números reales son los que tienen decimales. En Python se expresan mediante el tipo `float`. En otros lenguajes de programación, como C, tenemos también el tipo `double`, similar a `float` pero de mayor precisión (double = doble precisión). Python, sin embargo, implementa su tipo `float` a bajo nivel mediante una variable de tipo `double` de C, es decir, utilizando 64 bits, luego en Python siempre se utiliza doble precisión, y en concreto se sigue el estándar IEEE 754: 1 bit para el signo, 11 para el exponente, y 52 para la mantisa. Esto significa que los valores que podemos representar van desde $\pm$2,2250738585072020 x 10^{-308} hasta $\pm$1,7976931348623157×10^{308}.

La mayor parte de los lenguajes de programación siguen el mismo esquema para la representación interna. Pero como muchos sabréis esta tiene sus limitaciones, impuestas por el hardware. Por eso desde Python 2.4 contamos también con un nuevo tipo *Decimal*[1], para el caso de que se necesite representar fracciones de forma más precisa. Sin embargo este tipo está fuera del alcance de este tutorial, y sólo es necesario para el ámbito de la programación científica y otros relacionados. Para aplicaciones normales podeis utilizar el tipo `float` sin miedo, como ha venido haciéndose desde hace años, aunque teniendo en cuenta que los números en coma flotante no son precisos (ni en este ni en otros lenguajes de programación).

Para representar un número real en Python se escribe primero la parte entera, seguido de un punto y por último la parte decimal.

```
real = 0.2703
```

También se puede utilizar notación científica, y añadir una e (de exponente) para indicar un exponente en base 10. Por ejemplo:

```
real = 0.1e-3
```

sería equivalente a 0.1 x 10^{-3} = 0.1 x 0.001 = 0.0001

Complejos

Los números complejos son aquellos que tienen parte imaginaria. Si no conocías de su existencia, es más que probable que nunca lo vayas a necesitar, por lo que puedes saltarte este apartado tranquilamente. De hecho la mayor parte de lenguajes de programación carecen de este tipo, aunque sea muy utilizado por ingenieros y científicos en general.

En el caso de que necesitéis utilizar números complejos, o simplemente tengáis curiosidad, os diré que este tipo, llamado `complex` en Python, también se almacena usando coma flotante, debido a que estos números son una

[1] https://www.python.org/dev/peps/pep-0327/

extensión de los números reales. En concreto se almacena en una estructura de C, compuesta por dos variables de tipo `double`, sirviendo una de ellas para almacenar la parte real y la otra para la parte imaginaria.

Los números complejos en Python se representan de la siguiente forma:

```
complejo = 2.1 + 7.8j
```

Ejemplo de tipos de enteros

```
# -*- coding: utf8 -*-

# Entero INT / LONG
a = 7
print a
print type(a)

a = 7L
print a
print type(a)

# Reales simple
real = 0.348
print real
print type(real)

# Este numero tiene un exponente en base 10
# es decir, multiplicado por 10 a la N
real = 0.56e-3
print real
print type(real)
```

Tipo Cadenas

Las cadenas no son más que texto encerrado entre comillas simples (‘cadena’) o dobles (“cadena”). Dentro de las comillas se pueden añadir caracteres especiales escapándolos con ‘\’, como ‘\n’, el carácter de nueva línea, o ‘\t’, el de tabulación.

Una cadena puede estar precedida por el carácter ‘u’ o el carácter ‘r’, los cuales indican, respectivamente, que se trata de una cadena que utiliza codificación **Unicode** y una cadena `raw` (del inglés, cruda). Las cadenas `raw` se distinguen de las normales en que los caracteres escapados mediante la barra invertida (\) no se sustituyen por sus contrapartidas. Esto es especialmente útil, por ejemplo, para las expresiones regulares.

```
unicode = u"äóè"
raw = r"\n"
```

También es posible encerrar una cadena entre triples comillas (simples o dobles). De esta forma podremos escribir el texto en varias líneas, y al imprimir la cadena, se respetarán los saltos de línea que introdujimos sin tener que recurrir al carácter \n, así como las comillas sin tener que escaparlas.

Las cadenas también admiten operadores como la suma (concatenación de cadenas) y la multiplicación.

```
a = "uno"
b = "dos"

c = a + b # c es "unodos"
c = a * 3 # c es "unounouno"
```

Ejemplo de tipos de cadenas

```
# -*- coding: utf8 -*-

# Comillas simples
cadenaa = 'Texto entre comillas simples'
print cadenaa
print type(cadenaa)

# Comillas dobles
cadenab = "Texto entre comillas dobles"
print cadenab
print type(cadenab)

# Cadena con codigo escapes
cadenaesc = 'Texto entre \n\tcomillas simples'
print cadenaesc
print type(cadenaesc)

# Cadena multilinea
cadenac = """Texto linea 1
linea 2
linea 3
linea 4
.
.
.
.
.
linea N
"""
print cadenac
print type (cadenac)

# Repeticion de cadena
cadrep = "Cadena" * 3
print cadrep
print type (cadrep)

# Concatenacion de cadena
nombre = "Leonardo"
apellido = "Caballero"
nombre_completo = nombre + " " + apellido
print nombre_completo
print type (nombre_completo)

print "Tamano de cadena '", nombre_completo, "' es:", len(nombre_completo)

# acceder a rango de la cadena
print nombre_completo[3:13]
```

Tipos de booleanos

Como decíamos el tipo booleano sólo puede tener dos valores: `True` (cierto) y `False` (falso). Estos valores son especialmente importantes para las expresiones condicionales y los bucles, como veremos más adelante.

En realidad el tipo `bool` (el tipo de los booleanos) es una subclase del tipo `int`. Puede que esto no tenga mucho

sentido para tí si no conoces los términos de la orientación a objetos, que veremos más adelantes, aunque tampoco es nada importante.

Ejemplo de tipos de booleanos

```
# -*- coding: utf8 -*-

print '\nTipos de datos booleanos'
print '========================\n'

# Tipos de datos booleanos
aT = True
print "El valor es Verdadero:", aT, ", el cual es de tipo", type(aT), "\n"

aF = False
print "El valor es Falso:", aF, ", el cual es de tipo", type(aF)

print '\nOperadores booleanos'
print '====================\n'

# Operadores booleanos
aAnd = True and False
print "SI es Verdadero Y Falso, entonces es:", aAnd, ", el cual es de tipo", type(aAnd),

aOr = True or False
print "SI es Verdadero O Falso, entonces es:", aOr, ", el cual es de tipo", type(aOr), '

aNot = not True
print "Si NO es Verdadero, entonces es:", aNot, ", el cual es de tipo", type(aNot), "\n"
```

Tipos de conjuntos

Un conjunto, es una colección no ordenada y sin elementos repetidos. Los usos básicos de éstos incluyen verificación de pertenencia y eliminación de entradas duplicadas.

Ejemplo de tipos de conjuntos

```
# -*- coding: utf8 -*-

"""
    Un conjunto, es una colección no ordenada y sin elementos repetidos.
    Los usos básicos de éstos incluyen verificación de pertenencia y
    eliminación de entradas duplicadas.
"""

# crea un conjunto sin repetidos
plato = ['pastel', 'tequeno', 'papa', 'empanada', 'mandoca', 'queso']
print plato
print type(plato)
bebida = ['refresco', 'malta', 'jugo', 'cafe']
print bebida
print type(bebida)

# establece un conjunto a una variable
para_comer = set(plato)
```

```
print para_comer
print type(para_comer)

para_tomar = set(bebida)
print para_tomar
print type(para_tomar)

# Ejemplo practico de los condicionales
hay_tequeno = 'tequeno' in para_comer
hay_fresco = 'refresco' in para_tomar

print "\nTostadas A que Pipo!"
print "===================="

# valida si un elemento esta en el conjunto
print "Teneis tequeno?: ", 'tequeno' in para_comer

# valida si un elemento esta en el conjunto
print "Teneis pa tomar fresco?: ", 'refresco' in para_tomar

if (hay_tequeno and hay_fresco):
        print "Desayuno vergatario"
else:
    print "Desayuno ligero"
```

Tipos de listas

La lista en Python son variables que almacenan arrays, internamente cada posición puede ser un tipo de datos distinto.

En Python tiene varios tipos de datos *compuestos*, usados para agrupar otros valores. El más versátil es la *lista*, la cual puede ser escrita como una lista de valores separados por coma (ítems) entre corchetes. No es necesario que los ítems de una lista tengan todos el mismo tipo.

```
>>> a = ['pan', 'huevos', 100, 1234]
>>> a
['pan', 'huevos', 100, 1234]
```

Ejemplo de tipos de listas

```
# -*- coding: utf8 -*-

"""
   La lista en Python son variables que almacenan arrays,
   internamente cada posicion puede ser un tipo de datos distinto.
"""

# Coleccion ordenada / arreglos o vectores
l = [2, "tres", True, ["uno", 10]]
print l

# Accesar a un elemento especifico
l2 = l[1]
print l2

# Accesar a un elemento a una lista anidada
l3 = l[3][0]
```

```
print l3

# establecer nuevo valor de un elemento de lista
l[1] = 4
print l
l[1] = "tres"

# Obtener un rango de elemento especifico
l3 = l[0:3]
print l3

# Obtener un rango con soltos de elementos especificos
l4 = l[0:3:2]
print l4

l5 = l[1::2]
print l5
```

Tipos de tuplas

Una tupla es una lista inmutable. Una tupla no puede modificarse de ningún modo después de su creación.

Ejemplo de tipos de tuplas

```
# -*- coding: utf8 -*-

"""
    Una tupla es una lista inmutable. Una tupla no puede
    modificarse de ningún modo después de su creación.
"""

# Ejemplo simple de tupla
tupla = 12345, 54321, 'hola!'

# Ejemplo de tuplas anidadas
otra = tupla, (1, 2, 3, 4, 5)

# operación asinacion de valores de una tupla en variables
x, y, z = tupla


print "\nDefiniendo conexion a BD MySql"
print "==============================\n"

conexion_bd = "127.0.0.1","root","123456","nomina",
print "Conexion tipica:", conexion_bd
print type(conexion_bd)
conexion_completa = conexion_bd, "3307","10",
print "\nConexion con parametros adicionales:", conexion_completa
print type(conexion_completa)

print "\n"

print "Acceder a la IP de la bd:",  conexion_completa[0][0]
print "Acceder al usuario de la bd:",  conexion_completa[0][1]
print "Acceder a la clave de la bd:",  conexion_completa[0][2]
```

```
print "Acceder al nombre de la bd:",  conexion_completa[0][3]
print "Acceder al puerto de conexion:", conexion_completa[1]
print "Acceder al tiempo de espera de conexion:", conexion_completa[2]

print "\nMas informacion sobre Mysql y Python http://mysql-python.sourceforge.net/MySQLd
```

Tipos de diccionarios

El diccionario, que define una relación uno a uno entre claves y valores.

Ejemplo de tipos de diccionarios

```
# -*- coding: utf8 -*-

"""
    El diccionario, que define una relación
    uno a uno entre claves y valores.
"""

datos_basicos = {
    "nombres":"Leonardo Jose",
    "apellidos":"Caballero Garcia",
    "cedula":"14522590",
    "fecha_nacimiento":"03121980",
    "lugar_nacimiento":"Maracaibo, Zulia, Venezuela",
    "nacionalidad":"Venezolana",
    "estado_civil":"Complicado"
}

print "\nDetalle del diccionario"
print "=======================\n"

print "\nClaves del diccionario:", datos_basicos.keys()
print "\nValores del diccionario:", datos_basicos.values()
print "\nElementos del diccionario:", datos_basicos.items()


# Ejemplo practico de los diccionarios
print "\nInscripcion de Curso"
print "===================="

print "\nDatos de participante"
print "---------------------"

print "Cedula de identidad:", datos_basicos['cedula']
print "Nombre completo: " + datos_basicos['nombres'] + " " + datos_basicos['apellidos']
```

Operadores aritméticos

Los valores numéricos son además el resultado de una serie de operadores aritméticos y matemáticos:

Operador	Descripción	Ejemplo
+	Suma	r = 3 + 2 # r es 5
-	Resta	r = 4 – 7 # r es -3
-	Negación	r = -7 # r es -7
*	Multiplicación	r = 2 * 6 # r es 12
**	Exponente	r = 2 ** 6 # r es 64
/	División	r = 3.5 / 2 # r es 1.75
//	División entera	r = 3.5 // 2 # r es 1.0
%	Módulo	r = 7 % 2 # r es 1

Puede que tengáis dudas sobre cómo funciona el operador de módulo, y cuál es la diferencia entre división y división entera.

El operador de módulo no hace otra cosa que devolvernos el resto de la división entre los dos operandos. En el ejemplo, 7 / 2 sería 3, con 1 de resto, luego el módulo es 1.

La diferencia entre división y división entera no es otra que la que indica su nombre. En la división el resultado que se devuelve es un número real, mientras que en la división entera el resultado que se devuelve es solo la parte entera.

No obstante hay que tener en cuenta que si utilizamos dos operandos enteros, Python determinará que queremos que la variable resultado también sea un entero, por lo que el resultado de, por ejemplo, 3 / 2 y 3 // 2 sería el mismo: 1.

Si quisiéramos obtener los decimales necesitaríamos que al menos uno de los operandos fuera un número real, bien indicando los decimales

```
r = 3.0 / 2
```

o bien utilizando la función float (no es necesario que sepas lo que significa el término función, ni que recuerdes esta forma, lo veremos un poco más adelante):

```
r = float(3) / 2
```

Esto es así porque cuando se mezclan tipos de números, Python convierte todos los operandos al tipo más complejo de entre los tipos de los operandos.

Ejemplo de operadores numéricos

```
# -*- coding: utf8 -*-

"""
    Operadores numericos
"""

a = 26
b = 11.3
c = 5
d = 3.5

# Suma, Añade valores a cada lado del operador
print a + b

# Resta, Resta el operando de la derecha del operador del lado izquierdo
print c - a

# Multiplicacion, Multiplica los valores de ambos lados del operador
print d * c
```

```
20
21 # Exponente, Realiza el cálculo exponencial (potencia) de los operadores
22 print c ** 2
23
24 # Division
25 print float(c) / a
26
27 # Division entera,
28 print 7 / 3
29
30 # Cociente de una división, La división de operandos que el resultado es el cociente
31 # en el cual se eliminan los dígitos después del punto decimal.
32 print a // c
33
34 # Modulo, Divide el operando de la izquierda por el
35 # operador del lado derecho y devuelve el resto.
36 print 7 % 3
```

Operadores relacionales

Los valores booleanos son además el resultado de expresiones que utilizan operadores relacionales (comparaciones entre valores):

Operador	Descripción	Ejemplo
==	¿son iguales a y b?	r = 5 == 3 # r es False
!=	¿son distintos a y b?	r = 5 != 3 # r es True
<	¿es a menor que b?	r = 5 < 3 # r es False
>	¿es a mayor que b?	r = 5 > 3 # r es True
<=	¿es a menor o igual que b?	r = 5 <= 5 # r es True
>=	¿es a mayor o igual que b?	r = 5 >= 3 # r es True

Ejemplo de operadores relacionales

```
1  a = 5
2  b = 5
3  a1 = 7
4  b1 = 3
5  c1 = 3
6
7  cadena1 = 'Hola'
8  cadena2 = 'Adios'
9
10 lista1 = [1, 'Lista Python', 23]
11 lista2 = [11, 'Lista Python', 23]
12
13 # igual
14 c = a == b
15 print c
16
17 cadenas = cadena1 == cadena2
18 print cadenas
19
20 listas = lista1 == lista2
21 print listas
22
23 # diferente
```

```
24 d = a1 != b
25 print d
26
27 cadena0 = cadena1 != cadena2
28 print cadena0
29
30 # mayor que
31 e = a1 > b1
32 print e
33
34 # menor que
35 f = b1 < a1
36 print f
37
38 # mayor o igual que
39 g = b1 >= c1
40 print g
41
42 # menor o igual que
43 h = b1 <= c1
44 print h
```

Vídeo tutorial

- Tutorial Python 4 - Enteros, reales y operadores aritméticos[2].
- Tutorial Python 5 - Booleanos, operadores lógicos y cadenas[3].

Referencia

- Python - Tipos básicos[4].

[2] https://www.youtube.com/watch?v=ssnkfbBbcuw
[3] https://www.youtube.com/watch?v=ZrxcqbFYjiw
[4] http://mundogeek.net/archivos/2008/01/17/python-tipos-basicos/

CAPÍTULO 6

Sentencias IF

La sentencia `If` se usa para tomar decisiones, este evalua basicamente una operación logica, es decir una expresión que de como resultado verdadero o false (`True` o `False`), y ejecuta la pieza de codigo siguiente siempre y cuando el resultado sea verdadero.

Ejemplo de Sentencias IF

```
# -*- coding: utf8 -*-

"""
    Sentencias conficional if
"""

numero = int(raw_input("Ingresa un entero, por favor: "))

if numero < 0:
    numero = 0
    print 'Numero negativo cambiado a cero'
elif numero == 0:
    print 'Numero es Cero'
elif numero == 1:
    print 'Numero Simple'
else:
    print 'Mayor que uno'
```

Operadores de Asignaciones

Los operadores de asignación se utilizan para basicamente asignar un valor a una variable, así como cuando utilizamos el "=".

Los operadores de asignación son `=, +=, -=, \*=, /=, \*\*=, //=`, a continuación algunos ejemplos.

- = , igual a, es el mas simple de todos y asigna a la variable del lado izquierdo cualquier variable o resultado del lado derecho.
- **+= , suma a la variable del lado izquierdo el valor del lado derecho.** ej. si "a" es igual a 5 y a+=10, entonces "a" sera igual a 15
- **-= , resta a la variable del lado izquierdo el valor del lado derecho.** ej. si "a" es igual a 5 y a-=10, entonces "a" sera igual a -5
- ***=, multiplica a la variable del lado izquierdo el valor del lado derecho.** ej. si "a" es igual a 5 y a*=10, entonces "a" sera igual a 50

Espero que hasta el momento hayas podido encontrar este tutorial de ayuda, espero tus comentarios.

Ejemplo de Operadores de Asignaciones

```
# -*- coding: utf8 -*-

"""
    Operadores de asignaciones
"""

a = 21
b = 10
c = 0

print "el valor de la variable 'a' es:", a
print "el valor de la variable 'b' es:", b

c = a + b
print "Operador + | El valor de la variable 'c' es ", c

c += a
print "Operador += | El valor de la variable 'c' es ", c

c *= a
print "Operador *= | El valor de la variable 'c' es ", c

c /= a
print "Operador /= | El valor de la variable 'c' es ", c

c = 2
c %= a
print "Operador %= | El valor de la variable 'c' es ", c

c **= a
print "Operador **= | El valor de la variable 'c' es ", c

c //= a
print "Operador //= | El valor de la variable 'c' es ", c
```

Operadores de Comparación

Los valores booleanos son además el resultado de expresiones que utilizan operadores relacionales (comparaciones entre valores):

Operador	Descripción	Ejemplo
==	¿son iguales a y b?	r = 5 == 3 # r es False
!=	¿son distintos a y b?	r = 5 != 3 # r es True
<	¿es a menor que b?	r = 5 < 3 # r es False
>	¿es a mayor que b?	r = 5 > 3 # r es True
<=	¿es a menor o igual que b?	r = 5 <= 5 # r es True
>=	¿es a mayor o igual que b?	r = 5 >= 3 # r es True

Ejemplo Operadores de Comparación

```
# -*- coding: utf8 -*-

"""
```

```
    Operadores de comparacion
"""

a = 21
b = 10

print "el valor de la variable 'a' es:", a
print "el valor de la variable 'b' es:", b

if ( a == b ):
   print "Comparacion == | a es igual a b"
else:
   print "Comparacion == | a no es igual a b"

if ( a != b ):
   print "Comparacion != | a no es igual a b"
else:
   print "Comparacion != | a es igual a b"

if ( a <> b ):
   print "Comparacion <> | a no es igual a b"
else:
   print "Comparacion <> | a es igual a b"

if ( a < b ):
   print "Comparacion < | a es menor que b"
else:
   print "Comparacion < | a no es menor que b"

if ( a > b ):
   print "Comparacion > | a es mayor que b"
else:
   print "Comparacion > | a no es mayor que b"

c = 5;
d = 20;

print "el valor de la variable 'c' es:", c
print "el valor de la variable 'd' es:", d

if ( c <= d ):
   print "Comparacion <= | la variable 'c' es menor o igual a la variable 'd'"
else:
   print "Comparacion <= | la variable 'c' es ni menor de ni igual a la variable 'd'"

if ( d >= c ):
   print "Comparacion >= | la variable 'd' es o bien mayor que o igual a la variable 'c'
else:
   print "Comparacion >= | la variable 'd' es ni mayor que ni igual a la variable 'c'"
```

Operadores de Lógicos

Estos son los distintos tipos de operadores con los que podemos trabajar con valores booleanos, los llamados operadores lógicos o condicionales:

Operador	Descripción	Ejemplo
and	¿se cumple a y b?	r = True and False # r es False
or	¿se cumple a o b?	r = True or False # r es True
not	No a	r = not True # r es False

Ejemplo de Operadores de Lógicos

```
# -*- coding: utf8 -*-

"""
    Operadores logicos
"""

a = 10
b = 20

print "el valor de la variable 'a' es:", a
print "el valor de la variable 'b' es:", b

if ( a and b ):
   print "Operador and | a y b son VERDADERO"
else:
   print "Operador and | O bien la variable 'a' no es VERDADERO o la variable 'b' no es

if ( a or b ):
   print "Operador or | O bien la variable 'a' es VERDADERA o la variable 'b' es VERDADE
else:
   print "Operador or | Ni la variable 'a' es VERDADERA ni la variable 'b' es VERDADERA'

if not( a and b ):
   print "Operador not | Ni la variable 'a' NO es VERDADERA o la variable 'b' NO es VERD
else:
   print "Operador not | las variables 'a' y 'b' son VERDADERAS"
```

Vídeo tutorial

- Tutorial Python 10 - Sentencias condicionales[1].

Referencia

- Sentencias IF[2].
- Condicionales if y else en Python[3].
- Python - Tipos básicos[4].
- Operadores basicos de Python[5].

[1] https://www.youtube.com/watch?v=hLqKvB7tGWk
[2] http://docs.python.org.ar/tutorial/2/controlflow.html#la-sentencia-if
[3] http://codigoprogramacion.com/cursos/tutoriales-python/condicionales-if-y-else-en-python.html
[4] http://mundogeek.net/archivos/2008/01/17/python-tipos-basicos/
[5] http://codigoprogramacion.com/cursos/tutoriales-python/operadores-basicos-de-python.html

CAPÍTULO 7

Bucles WHILE

En Python tenemos una palabra reservada llamada "**while**" que nos permite ejecutar ciclos, o bien secuencias periódicas que nos permiten ejecutar código múltiples veces.

El ciclo `while` nos permite realizar múltiples iteraciones basándonos en el resultado de una expresión lógica que puede tener como resultado un valor verdadero o falso (true o false).

Tipos de Bucles 'while'

Bucles 'while' controlado por Conteo

```
print "\nWhile controlado con Conteo"
print "============================\n"

print "Un ejemplo es un sumador numérico hasta 10, \ncomo se muestra a continuación:\n"

suma = 0
numero = 1
while numero <= 10:
    suma = numero + suma
    numero = numero + 1
print "La suma es " + str(suma)
```

En este ejemplo tenemos un contador con un valor inicial de cero, cada iteración del `while` manipula esta variable de manera que incremente su valor en 1, por lo que después de su primera iteración el contador tendrá un valor de 1, luego 2, y así sucesivamente. Eventualmente cuando el contador llegue a tener un valor de 10, la condición del ciclo `numero <= 10` sera falsa, por lo que el ciclo terminará arrojando el siguiente resultado.

Bucles 'while' controlado por Evento

```
print "\nWhile controlado con Evento"
print "============================\n"

print "Un ejemplo es calcular el promedio de grado, \ncomo se muestra a continuación:\n"

promedio = 0.0
total = 0
contar = 0

print "Introduzca valor numerico de un grado (-1 para salir): "
grado = int(raw_input())
while grado != -1:
    total = total + grado
```

```
    contar = contar + 1
    print "Introduzca valor numerico de un grado (-1 para salir): "
    grado = int(raw_input())
promedio = total / contar
print "Promedio de grado: " + str(promedio)
```

Sentencias utilitarias

Usando la sentencia 'break'

```
print "\nWhile con sentencia break"
print "=========================\n"

variable = 10
while variable > 0:
    print 'Actual valor de variable:', variable
    variable = variable -1
    if variable == 5:
        break
```

Adicionalmente existe una forma alternativa de interrumpir o cortar los ciclos utilizando la palabra reservada `break`. Esta nos permite salir del ciclo incluso si la expresión evaluada en `while` (o en otro ciclo como `for`) permanece siendo verdadera. Para comprender mejor usaremos el mismo ejemplo anterior pero interrumpiremos el ciclo usando `break`.

Usando la sentencia 'continue'

```
print "\nWhile con sentencia continue"
print "============================\n"

variable = 10
while variable > 0:
   variable = variable -1
   if variable == 5:
      continue
   print 'Actual valor de variable:', variable
```

La sentencia `continue` hace que pase de nuevo al principio del bucle aunque no se haya terminado de ejecutar el ciclo anterior.

Ejemplos

Ejemplo de fibonacci

Ejemplo de la Sucesión de Fibonacci[1] con bucle `while`:

```
# -*- coding: utf8 -*-

"""
    módulo de Sucesión de números Fibonacci
    Mas informacion en http://es.wikipedia.org/wiki/Sucesi%C3%B3n_de_Fibonacci
"""

```

[1] http://es.wikipedia.org/wiki/Sucesi %C3 %B3n_de_Fibonacci

```
a, b = 0, 1
while b < 100:
    print b,
    a, b = b, a + b
```

Ejemplo de bucle while

Ejemplo de completo de bucles `while`:

```
# -*- coding: utf8 -*-

"""
    Ejemplo de uso de bucle While
"""

print "\nWhile controlado con Conteo"
print "============================\n"

print "Un ejemplo es un sumador numérico hasta 10, \ncomo se muestra a continuación:\n"

suma = 0
numero = 1
while numero <= 10:
    suma = numero + suma
    numero = numero + 1
print "La suma es " + str(suma)

print "\nWhile controlado con Evento"
print "============================\n"

print "Un ejemplo es calcular el promedio de grado, \ncomo se muestra a continuación:\n"

promedio = 0.0
total = 0
contar = 0

print "Introduzca valor numerico de un grado (-1 para salir): "
grado = int(raw_input())
while grado != -1:
    total = total + grado
    contar = contar + 1
    print "Introduzca valor numerico de un grado (-1 para salir): "
    grado = int(raw_input())
promedio = total / contar
print "Promedio de grado: " + str(promedio)



print "\nWhile con sentencia break"
print "=========================\n"

variable = 10
while variable > 0:
    print 'Actual valor de variable:', variable
    variable = variable -1
    if variable == 5:
        break

```

```
50 print "\nWhile con sentencia continue"
51 print "============================\n"
52
53 variable = 10
54 while variable > 0:
55     variable = variable -1
56     if variable == 5:
57         continue
58     print 'Actual valor de variable:', variable
```

Vídeo tutorial

- Tutorial Python 11 - Bucles[2].

Referencia

- Introducción a Bucles 'while'[3].
- Ciclo while en Python[4].

[2] https://www.youtube.com/watch?v=IyI2ZuOq_xQ
[3] http://docs.python.org.ar/tutorial/2/introduction.html#primeros-pasos-hacia-la-programacion
[4] http://codigoprogramacion.com/cursos/tutoriales-python/ciclo-while-en-python.html

CAPÍTULO 8

Bucles FOR

La sentencia *for* en Python difiere un poco de lo que uno puede estar acostumbrado en lenguajes como C o Pascal. En lugar de siempre iterar sobre una progresión aritmética de números (como en Pascal) o darle al usuario la posibilidad de definir tanto el paso de la iteración como la condición de fin (como en C), la sentencia *for* de Python itera sobre los ítems de cualquier secuencia (una lista o una cadena de texto), en el orden que aparecen en la secuencia.

Tipos de Bucles 'for'

Bucles 'for' con Listas

```
print "\nItera una lista de animales"
print "============================\n"

# Midiendo cadenas de texto
lista_animales = ['gato', 'ventana', 'defenestrado']

for animal in lista_animales:
    print "El animal es:", animal, ", la cantidad de letras de la posicion son:", len(ar

# Si se necesita iterar sobre una secuencia de números.
# Genera una lista conteniendo progresiones aritméticos
print "\nRango de 15 numeros:", range(15)

print "\nItera una cadena con rango dinamico"
print "==================================\n"

frases = ['Mary', 'tenia', 'un', 'corderito']
for palabra in range(len(frases)):
    print "La palabra es:", frases[palabra], "su posicion en la frase es:", palabra
```

Bucles 'for' con Tuplas

```
print "\nItera una tupla de parametros"
print "============================\n"

conexion_bd = "127.0.0.1","root","123456","nomina"
for parametro in conexion_bd:
    print parametro
```

Bucles 'for' con Diccionarios

```
print "\nItera un diccionario datos basicos"
print "==================================\n"

datos_basicos = {
    "nombres":"Leonardo Jose",
    "apellidos":"Caballero Garcia",
    "cedula":"14522590",
    "fecha_nacimiento":"03121980",
    "lugar_nacimiento":"Maracaibo, Zulia, Venezuela",
    "nacionalidad":"Venezolana",
    "estado_civil":"Complicado"
}

dato = datos_basicos.keys()
valor = datos_basicos.values()
cantidad_datos = datos_basicos.items()

for dato, valor in cantidad_datos:
    print dato + ": " + valor
```

Ejemplo de bucle for

Ejemplo de completo de bucles `for`:

```
# -*- coding: utf8 -*-

"""
    Ejemplo de uso de bucle For

"""

print "\nItera una lista de animales"
print "===========================\n"

# Midiendo cadenas de texto
lista_animales = ['gato', 'ventana', 'defenestrado']

for animal in lista_animales:
    print "El animal es:", animal, ", la cantidad de letras de la posicion son:", len(ar

# Si se necesita iterar sobre una secuencia de números.
# Genera una lista conteniendo progresiones aritméticos
print "\nRango de 15 numeros:", range(15)

print "\nItera una cadena con rango dinamico"
print "===================================\n"

frases = ['Mary', 'tenia', 'un', 'corderito']
for palabra in range(len(frases)):
    print "La palabra es:", frases[palabra], "su posicion en la frase es:", palabra

##################################################

print "\nItera una tupla de parametros"
print "=============================\n"
```

```
32
33 conexion_bd = "127.0.0.1","root","123456","nomina"
34 for parametro in conexion_bd:
35     print parametro
36
37 ##################################################
38
39 print "\nItera un diccionario datos basicos"
40 print "==================================\n"
41
42 datos_basicos = {
43     "nombres":"Leonardo Jose",
44     "apellidos":"Caballero Garcia",
45     "cedula":"14522590",
46     "fecha_nacimiento":"03121980",
47     "lugar_nacimiento":"Maracaibo, Zulia, Venezuela",
48     "nacionalidad":"Venezolana",
49     "estado_civil":"Complicado"
50 }
51
52 dato = datos_basicos.keys()
53 valor = datos_basicos.values()
54 cantidad_datos = datos_basicos.items()
55
56 for dato, valor in cantidad_datos:
57     print dato + ": " + valor
```

Vídeo tutorial

- Tutorial Python 11 - Bucles[1].

Referencia

- Introducción a Bucles 'for'[2].

[1] https://www.youtube.com/watch?v=IyI2ZuOq_xQ
[2] http://docs.python.org.ar/tutorial/2/controlflow.html#la-sentencia-for

CAPÍTULO 9

Funciones

Introducción a Funciones[1] - ¿Por qué?.

Definiendo Funciones

```
def iva():
    ''' funcion basica de Python '''
    iva = 12
    costo = input('¿Cual es el monto a calcular?: ')
    calculo = costo * iva / 100
    print calculo
```

Llamando Funciones

```
>>> iva()
¿Cual es el monto a calcular?: 300
36
```

Funciones con Argumentos Múltiple

```
def suma(numero1,numero2):
    print numero1 + numero2

def imprime_fibonacci(n):
    ''' escribe la sucesión Fibonacci hasta n '''
    a, b = 0, 1
```

Y se invoca de la siguiente forma:

```
        a, b = b, a + b
```

- Funciones de Predicado.

Ejemplo de Funciones

Ejemplo de completo de Funciones

[1] http://docs.python.org.ar/tutorial/2/controlflow.html#definiendo-funciones

```python
# -*- coding: utf8 -*-

"""
    Funciones en Python
"""

def iva():
    ''' funcion basica de Python '''
    iva = 12
    costo = input('¿Cual es el monto a calcular?: ')
    calculo = costo * iva / 100
    print calculo

def suma(numero1,numero2):
    print numero1 + numero2

def imprime_fibonacci(n):
    ''' escribe la sucesión Fibonacci hasta n '''
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a + b

def devuelve_fibonacci(n):
    ''' devuelve la sucesión Fibonacci hasta n '''
    resultado = []
    a, b = 0, 1
    while b < n:
        resultado.append(b)
        a, b = b, a + b
    return resultado

print "El calculo de IVA es :", iva()

print "La suma de dos numeros es:", suma(13,37)

print "El calculo de IVA es :", iva(10)

print "La sucesión Fibonacci hasta 10 es:", imprime_fibonacci(10)

print "La sucesión Fibonacci hasta 50 es:", devuelve_fibonacci(50)
```

Funciones Recursiva

TODO.

Objetos de función

TODO.

Funciones anónimas

TODO.

Funciones de orden superior

TODO.

Vídeo tutorial

- Tutorial Python 12 - Funciones[2].

Referencia

- Introducción a Funciones[3] - ¿Por qué?.

[2] https://www.youtube.com/watch?v=_C7Uj7O5o_Q
[3] http://docs.python.org.ar/tutorial/2/controlflow.html#definiendo-funciones

Depuración con pdb

En este tutorial se exploran herramientas que ayudan a entender tu código: depuración para encontrar y corregir *bugs* (errores).

El depurador python, `pdb`: http://docs.python.org/library/pdb.html, te permite inspeccionar tu código de forma interactiva.

Te permite:

- Ver el código fuente.
- Ir hacia arriba y hacia abajo del punto donde se ha producido un error.
- Inspeccionar valores de variables.
- Modificar valores de variables.
- Establecer `breakpoints` (punto de parada del proceso).

print

Sí, las declaraciones `print` sirven como herramienta de depuración. Sin embargo, para inspeccionar en tiempo de ejecución es más eficiente usar el depurador.

Invocando al depurador

Formas de lanzar el depurador:

1. Postmortem, lanza el depurador después de que se hayan producido errores.
2. Lanza el módulo con el depurador.
3. Llama al depurador desde dentro del módulo.

Postmortem

Situación: Estás trabajando en ipython y obtienes un error (*traceback*).

En este caso estamos depurando el fichero `'index_error.py <http://pybonacci.github.io/scipy-lecture-notes-ES/_downloads/index_error.py>'_`. Cuando lo ejecutes verás como se lanza un `IndexError`. Escribe `%debug` y entrarás en el depurador.

```
In [1]: %run index_error.py
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/home/varoquau/dev/scipy-lecture-notes/advanced/debugging_optimizing/index_error.py in <
      6
```

```
      7 if __name__ == '__main__':
----> 8     index_error()
      9

/home/varoquau/dev/scipy-lecture-notes/advanced/debugging_optimizing/index_error.py in i
      3 def index_error():
      4     lst = list('foobar')
----> 5     print lst[len(lst)]
      6
      7 if __name__ == '__main__':

IndexError: list index out of range

In [2]: %debug
> /home/varoquau/dev/scipy-lecture-notes/advanced/debugging_optimizing/index_error.py(5)
      4     lst = list('foobar')
----> 5     print lst[len(lst)]
      6

ipdb> list
      1 """Small snippet to raise an IndexError."""
      2
      3 def index_error():
      4     lst = list('foobar')
----> 5     print lst[len(lst)]
      6
      7 if __name__ == '__main__':
      8     index_error()
      9

ipdb> len(lst)
6
ipdb> print lst[len(lst)-1]
r
ipdb> quit

In [3]:
```

Depuración post-mortem sin ipython

En algunas situaciones no podrás usar IPython, por ejemplo para depurar un *script* que ha sido llamado desde la línea de comandos. En este caso, puedes ejecutar el *script* de la siguiente forma `python -m pdb script.py`:

```
$ python -m pdb index_error.py
> /home/varoquau/dev/scipy-lecture-notes/advanced/debugging_optimizing/index_error.py(1
-> """Small snippet to raise an IndexError."""
(Pdb) continue
Traceback (most recent call last):
File "/usr/lib/python2.6/pdb.py", line 1296, in main
    pdb._runscript(mainpyfile)
File "/usr/lib/python2.6/pdb.py", line 1215, in _runscript
    self.run(statement)
File "/usr/lib/python2.6/bdb.py", line 372, in run
    exec cmd in globals, locals
File "<string>", line 1, in <module>
File "index_error.py", line 8, in <module>
    index_error()
File "index_error.py", line 5, in index_error
    print lst[len(lst)]
IndexError: list index out of range
Uncaught exception. Entering post mortem debugging
Running 'cont' or 'step' will restart the program
> /home/varoquau/dev/scipy-lecture-notes/advanced/debugging_optimizing/index_error.py(5
-> print lst[len(lst)]
(Pdb)
```

Ejecución paso a paso

Situación: Crees que existe un error en un módulo pero no estás seguro donde.

Por ejemplo, estamos intentado depurar wiener_filtering.py[1]. A pesar de que el código se ejecuta, observamos que el filtrado no se está haciendo correctamente.

- Ejecuta el *script* en IPython con el depurador usando `%run -d wiener_filtering.py`:

```
In [1]: %run -d wiener_filtering.py
*** Blank or comment
*** Blank or comment
*** Blank or comment
Breakpoint 1 at /home/varoquau/dev/scipy-lecture-notes/advanced/debugging_optimizing
NOTE: Enter 'c' at the ipdb>  prompt to start your script.
> <string>(1)<module>()
```

- Coloca un *breakpoint* en la línea 34 usando `b 34`:

```
ipdb> n
> /home/varoquau/dev/scipy-lecture-notes/advanced/debugging_optimizing/wiener_filter
      3
1---> 4 import numpy as np
      5 import scipy as sp

ipdb> b 34
Breakpoint 2 at /home/varoquau/dev/scipy-lecture-notes/advanced/debugging_optimizing
```

- Continua la ejecución hasta el siguiente *breakpoint* con `c(ont(inue))`:

[1] http://pybonacci.github.io/scipy-lecture-notes-ES/_downloads/wiener_filtering.py

```
ipdb> c
> /home/varoquau/dev/scipy-lecture-notes/advanced/debugging_optimizing/wiener_filter
     33     """
2--> 34     noisy_img = noisy_img
     35     denoised_img = local_mean(noisy_img, size=size)
```

- Da pasos hacia adelante y detrás del código con `n(ext)` y `s(tep)`. `next` salta hasta la siguiente declaración en el actual contexto de ejecución mientras que `step` se moverá entre los contextos en ejecución, i.e. permitiendo explorar dentro de llamadas a funciones:

```
ipdb> s
> /home/varoquau/dev/scipy-lecture-notes/advanced/debugging_optimizing/wiener_filter
2    34     noisy_img = noisy_img
---> 35     denoised_img = local_mean(noisy_img, size=size)
     36     l_var = local_var(noisy_img, size=size)

ipdb> n
> /home/varoquau/dev/scipy-lecture-notes/advanced/debugging_optimizing/wiener_filter
     35     denoised_img = local_mean(noisy_img, size=size)
---> 36     l_var = local_var(noisy_img, size=size)
     37     for i in range(3):
```

- Muévete unas pocas líneas y explora las variables locales:

```
ipdb> n
> /home/varoquau/dev/scipy-lecture-notes/advanced/debugging_optimizing/wiener_filter
     36     l_var = local_var(noisy_img, size=size)
---> 37     for i in range(3):
     38         res = noisy_img - denoised_img
ipdb> print l_var
[[5868 5379 5316 ..., 5071 4799 5149]
 [5013  363  437 ...,  346  262 4355]
 [5379  410  344 ...,  392  604 3377]
 ...,
 [ 435  362  308 ...,  275  198 1632]
 [ 548  392  290 ...,  248  263 1653]
 [ 466  789  736 ..., 1835 1725 1940]]
ipdb> print l_var.min()
0
```

Oh dear, solo vemos enteror y variación 0. Aquí está nuestro error, estamos haciendo aritmética con enteros.

Lanzando excepciones en errores numéricos

Cuando ejecutamos el fichero wiener_filtering.py[a], se lanzarán los siguientes avisos:

```
In [2]: %run wiener_filtering.py
wiener_filtering.py:40: RuntimeWarning: divide by zero encountered in divide
    noise_level = (1 - noise/l_var )
```

Podemos convertir estos avisos a excepciones, lo que nos permitiría hacer una depuración post-mortem sobre ellos y encontrar el problema de manera más rápida:

```
In [3]: np.seterr(all='raise')
Out[3]: {'divide': 'print', 'invalid': 'print', 'over': 'print', 'under': 'ignore'}
In [4]: %run wiener_filtering.py
---------------------------------------------------------------------------
FloatingPointError                        Traceback (most recent call last)
/home/esc/anaconda/lib/python2.7/site-packages/IPython/utils/py3compat.pyc in execfile(
    176             else:
    177                 filename = fname
--> 178             __builtin__.execfile(filename, *where)

/home/esc/physique-cuso-python-2013/scipy-lecture-notes/advanced/debugging/wiener_filte
     55 pl.matshow(noisy_lena[cut], cmap=pl.cm.gray)
     56
---> 57 denoised_lena = iterated_wiener(noisy_lena)
     58 pl.matshow(denoised_lena[cut], cmap=pl.cm.gray)
     59

/home/esc/physique-cuso-python-2013/scipy-lecture-notes/advanced/debugging/wiener_filte
     38         res = noisy_img - denoised_img
     39         noise = (res**2).sum()/res.size
---> 40         noise_level = (1 - noise/l_var )
     41         noise_level[noise_level<0] = 0
     42         denoised_img += noise_level*res
FloatingPointError: divide by zero encountered in divide
```

[a] http://pybonacci.github.io/scipy-lecture-notes-ES/_downloads/wiener_filtering.py

Otras formas de comenzar una depuración

- **Lanzar una excepción *break point* a lo pobre**

  Si encuentras tedioso el tener que anotar el número de línea para colocar un *break point*, puedes lanzar una excepción en el punto que quieres inspeccionar y usar la 'magia' `%debug` de ipython. Destacar que en este caso no puedes moverte por el código y continuar después la ejecución.

- **Depurando fallos de pruebas usando nosetests**

  Podemos ejecutar `nosetests --pdb` para saltar a la depuración post-mortem de excepciones y `nosetests --pdb-failure` para inspeccionar los fallos de pruebas usando el depurador.

  Además, puedes usar la interfaz IPython para el depurador en **nose** usando el plugin de **nose** ipdbplugin[2]. Podremos, entonces, pasar las opciones `--ipdb` y `--ipdb-failure` a los *nosetests*.

- **Llamando explícitamente al depurador**

  Inserta la siguiente línea donde quieres que salte el depurador:

  ```
  import pdb; pdb.set_trace()
  ```

[2] http://pypi.python.org/pypi/ipdbplugin

Advertencia: Cuandos e ejecutan `nosetests`, se captura la salida y parecerá que el depurador no está funcionando. Para evitar esto simplemente ejecuta los `nosetests` con la etiqueta `-s`.

Depuradores gráficos y alternativas

- Quizá encuentres más conveniente usar un depurador gráfico como winpdb[a]. para inspeccionar saltas a través del código e inspeccionar las variables
- De forma alternativa, pudb[b] es un buen depurador semi-gráfico con una interfaz de texto en la consola.
- También, estaría bien echarle un ojo al proyecto pydbgr[c]

[a] http://winpdb.org/
[b] http://pypi.python.org/pypi/pudb
[c] http://code.google.com/p/pydbgr/

Comandos del depurador e interacciones

`l(list)`	Lista el código en la posición actual
`u(p)`	Paso arriba de la llamada a la pila (*call stack*)
`d(own)`	Paso abajo de la llamada a la pila ((*call stack*)
`n(ext)`	Ejecuta la siguiente línea (no va hacia abajo en funciones nuevas)
`s(tep)`	Ejecuta la siguiente declaración (va hacia abajo en las nuevas funciones)
`bt`	Muestra el *call stack*
`a`	Muestra las variables locales
`!command`	Ejecuta el comando **Python** proporcionado (en oposición a comandos pdb)

Advertencia: Los comandos de depuración no son código Python
No puedes nombrar a las variables de la forma que quieras. Por ejemplo, si estamos dentro del depurador no podremos sobreescribir a las variables con el mismo y, por tanto, **habrá que usar diferentes nombres para las variables cuando estemos teclenado código en el depurador**.

Obteniendo ayuda dentro del depurador

Teclea `h` o `help` para acceder a la ayuda interactiva:

```
ipdb> help

Documented commands (type help <topic>):
========================================
EOF    bt         cont      enable  jump  pdef    r        tbreak   w
a      c          continue  exit    l     pdoc    restart  u        whatis
alias  cl         d         h       list  pinfo   return   unalias  where
args   clear      debug     help    n     pp      run      unt
b      commands   disable   ignore  next  q       s        until
break  condition  down      j       p     quit    step     up

Miscellaneous help topics:
==========================
exec  pdb

Undocumented commands:
======================
retval  rv
```

Vídeo tutorial

- Depurando um programa Python com pdb - Python Debugger[3].

Referencia

- pdb — The Python Debugger[4].

[3] https://www.youtube.com/watch?v=N4NtB4r28h0
[4] https://docs.python.org/2/library/pdb.html

Entrada / Salida en Python

Nuestros programas serían de muy poca utilidad si no fueran capaces de interaccionar con el usuario. Para mostrar mensajes en pantalla, se utiliza el uso de la palabra clave `print`.

Para pedir información al usuario, debe utilizar las funciones `input` y `raw_input`, así como los argumentos de línea de comandos.

Ejemplo de E/S en Python

Este ejemplo simula a sala de chat del servicio LatinChat.com, validando datos de entradas numerico y tipo cadena e interactua con el usuario y en base a condicionales muestra un mensaje.

```
# -*- coding: utf8 -*-

"""Ilustración de ingreso interactivo en Python.

Simula a sala de chat del servicio LatinChat.com.

Validando datos de entradas numerico y tipo cadena.

E interactua con el usuario y en base a condicionales
muestra un mensaje.

"""

print "\nSimulando a LatinChat"
print "====================="

print "\nLatinChat > De 20 a 30 anos"
print "---------------------------\n"

print 'Pepe: '
nombre = raw_input('¿Cómo te llamás?: ')
print 'Pepe: Hola', nombre, ', encantado de conocerte :3'

print 'Pepe: '
edad = input('¿Cual es tu edad?: ')
print 'Tu tienes', edad, 'y yo no tengo soy un programa xD'

print 'Pepe: '
tiene_WebCam = raw_input('¿Tienes WebCam?, ingrese "si" o "no", por favor!: ')

if tiene_WebCam in ('s', 'S', 'si', 'Si', 'SI'):
        print "Pon la WebCam para verte :-D"
```

```
33 elif tiene_WebCam in ('n', 'no', 'No', 'NO'):
34         print "Lastima por ti :'( Adios"
```

Vídeo tutorial

- Tutorial Python 30 - Entrada Estandar rawInput[1].
- Tutorial Python 31 - Salida Estandar rawInput[2].

Referencia

- Python Programming / Input and Output[3].
- Python - Entrada / Salida. Ficheros[4].

[1] https://www.youtube.com/watch?v=AzeUCuMvW6I
[2] https://www.youtube.com/watch?v=B-JPXgxK3Oc
[3] http://en.wikibooks.org/wiki/Python_Programming/Input_and_Output
[4] http://mundogeek.net/archivos/2008/04/02/python-entrada-salida-ficheros/

CAPÍTULO 12

Scaffolding en proyectos Python

Sobre este artículo

Autor(es) Leonardo J. Caballero G.
Correo(s) leonardocaballero@gmail.com[a]
Compatible con Python 2.4 o versiones superiores
Fecha 03 de Octubre de 2014

[a] leonardocaballero@gmail.com

La estructura del *paquete Egg* Python es poco compleja. Por lo cual para empezar con su primer proyecto y diversos módulos, puede usar el concepto **Scaffolding** para crear un esqueleto de código usando las plantillas adecuadas para *paquetes Python*.

Este concepto *scaffolding*, es muy útil para del arranque de su desarrollo, ofreciendo una serie de colecciones de plantillas *esqueletos* que permiten iniciar rápidamente proyectos, existente diversos *esqueletos* orientados a tipos de desarrollos específicos.

¿Qué es PasteScript?

Es una herramienta de linea de comando basada en plugins que le permiten crear estructuras de paquetes de proyectos Python además sirve aplicaciones web, con configuraciones basadas en paste.deploy[1].

Instalación

Dentro de su entorno virtual[2] activado debe instalar el paquete PasteScript[3], ejecutando el siguiente comando:

```
(python)$ pip install PasteScript
```

Nota: No olvidar que estos paquetes han sido instalados con el entorno virtual que previamente usted activo, eso quiere decir que los paquetes previamente instalados con easy_install[4] están instalados en el directorio `~/virtualenv/python/lib/python2.x/site-packages/` en ves del directorio de su versión de Python de sistema `/usr/lib/python2.x/site-packages/`

Al finalizar la instalación podrá opcionalmente consultar cuales plantillas tiene disponible para usa, ejecutando el siguiente comando:

```
(python)$ paster create --list-templates
  Available templates:
```

[1] http://pypi.python.org/pypi/PasteDeploy
[2] https://plone-spanish-docs.readthedocs.-org/es/latest/python/creacion_entornos_virtuales.html
[3] http://pypi.python.org/pypi/PasteScript
[4] https://plone-spanish-docs.readthedocs.org/es/latest/python/setuptools.html#que-es-easyinstall

```
    basic_package:       A basic setuptools-enabled package
    paste_deploy:        A web application deployed through paste.deploy
```

Usted puede usar el comando **paster** para crear paquetes Python.

```
(python)$ paster create -t basic_package mipaquetepython

  Selected and implied templates:

    PasteScript#basic_package  A basic setuptools-enabled package

  Variables:
    egg:      mipaquetepython
    package:  mipaquetepython
    project:  mipaquetepython
  Enter version (Version (like 0.1)) ['']: 0.1
  Enter description (One-line description of the package) ['']: Mi Paquete Básico
  Enter long_description (Multi-line description (in reST)) ['']: Mi Paquete Básico para
  Enter keywords (Space-separated keywords/tags) ['']: PasteScript Basic Package Demo
  Enter author (Author name) ['']: Pedro Picapiedra
  Enter author_email (Author email) ['']: pedro@acme.com
  Enter url (URL of homepage) ['']: http://github.com/pyve/mipaquetepython
  Enter license_name (License name) ['']: GPL
  Enter zip_safe (True/False: if the package can be distributed as a .zip file) [False]:
  Creating template basic_package
  Creating directory ./mipaquetepython
    Recursing into +package+
      Creating ./mipaquetepython/mipaquetepython/
      Copying __init__.py to
      ./mipaquetepython/mipaquetepython/__init__.py
    Copying setup.cfg to ./mipaquetepython/setup.cfg
    Copying setup.py_tmpl to ./mipaquetepython/setup.py
  Running /home/macagua/virtualenv/python/bin/python setup.py egg_info
```

Usted puede verificar el paquete previamente creado y observará como este paquete básico ha habilitado el Setuptools[5].

```
(python)$ tree mipaquetepython/
  mipaquetepython/
  |-- mipaquetepython
  |   `-- __init__.py
  |-- mipaquetepython.egg-info
  |   |-- PKG-INFO
  |   |-- SOURCES.txt
  |   |-- dependency_links.txt
  |   |-- entry_points.txt
  |   |-- not-zip-safe
  |   `-- top_level.txt
  |-- setup.cfg
  `-- setup.py
```

Para instalar este paquete ejecute el siguiente comando:

```
(python)$ cd mipaquetepython/mipaquetepython/
(python)$ vim app.py
```

Escriba un simple código que solicita un valor y luego lo muestra:

```
var = raw_input("Introduzca alguna frase: ")
print "Usted introdujo: ", var
```

[5] https://plone-spanish-docs.readthedocs.org/es/latest/python/setuptools.html

Guarde los cambios en el archivo `app.py`, luego importe su aplicación `app.py` en el archivo `__init__.py` con el siguiente código fuente:

```
from mipaquetepython import app
```

Para comprobar su instalación ejecute el siguiente comando:

```
(python)$ python
```

Y realice una importación del paquete `mipaquetepython` ejecutando el siguiente comando:

```
>>> import mipaquetepython
Introduzca alguna frase: Esta cadena
Usted introdujo:  Esta cadena
>>> exit()
```

De esta forma tienes creado un *paquete Egg* Python.

Esqueletos en diversos proyectos Python

A continuación se muestran algunos esqueletos útiles:

- Esqueletos de proyectos Zope/Plone[6].
- Esqueletos de proyectos OpenERP[7].

  Nota: OpenERP[8], es un sistema ERP y CRM programado con Python, de propósito general.

- **Esqueletos de proyectos Django**:

  Nota: Django[9], es un Framework Web Python, de propósito general.

  - django-project-templates[10], plantillas Paster para crear proyectos Django.
  - fez.djangoskel[11], es una colección de plantillas Paster para crear aplicaciones Django como *paquetes Egg*.
  - django-harness[12], es una aplicación destinada a simplificar las tareas típicas relacionadas con la creación de un sitio web hechos con Django, el mantenimiento de varias instalaciones (local, producción, etc) y cuidando su instalación global y su estructura de "esqueleto" actualizado del sitio de manera fácil.
  - lfc-skel[13], Provee una plantilla para crear una aplicación django-lfc[14] CMS.

- **Esqueletos de proyectos Pylons**:

  Nota: Pylons[15], es un Framework Web Python, de propósito general.

  - Pylons[16], al instalarse usando la utilidad easy_install[17] instala dos plantillas de proyectos Pylons.
  - PylonsTemplates[18], le ofrece plantillas adicionales `paster` para aplicaciones Pylons, incluyendo implementación de `repoze.what`.

[6] https://plone-spanish-docs.readthedocs.org/es/latest/python/skel_proyectos_plone.html
[7] https://plone-spanish-docs.readthedocs.org/es/latest/python/skel_proyectos_openerp.html
[8] https://www.openerp.com/
[9] https://www.djangoproject.com/
[10] http://pypi.python.org/pypi/django-project-templates
[11] http://pypi.python.org/pypi/fez.djangoskel
[12] http://pypi.python.org/pypi/django-harness
[13] http://pypi.python.org/pypi/lfc-skel/
[14] http://pypi.python.org/pypi/django-lfc
[15] http://pypi.python.org/pypi/Pylons/
[16] http://pypi.python.org/pypi/Pylons/
[17] https://plone-spanish-docs.readthedocs.org/es/latest/python/setuptools.html#que-es-easyinstall
[18] http://pypi.python.org/pypi/PylonsTemplates/

  - BlastOff[19], Una plantilla de aplicación Pylons[20] que proporciona un esqueleto de entorno de trabajo configurado con `SQLAlchemy`, `mako`, `repoze.who`, `ToscaWidgets`, `TurboMail`, `WebFlash` y (opcionalmente) `SchemaBot`. La aplicación generada esta previamente configurada con autenticación, inicio de sesión y formularios de registro, y (opcionalmente) confirmación de correo electrónico. `BlastOff` ayudar a acelerar el desarrollo de aplicaciones en Pylons por que genera un proyecto con una serie de dependencias configuraciones previamente.

- **Esqueletos de proyectos CherryPy**:

  Nota: CherryPy[21], es un MicroFramework Web Python, de propósito general.

  - CherryPaste[22], Usar CherryPy dentro Paste.

- **Esqueletos de proyectos Trac**:

  Nota: Trac[23], es un sistema de gestión de proyectos de desarrollos de software.

  - TracLegosScript[24], TracLegos es un software diseñado para ofrecer plantillas para proyectos Trac y asiste con la creación de proyecto trac.
  - trac_project[25], Plantilla de proyecto Trac de software de código abierto.

Recomendaciones

Si desea trabajar con algún proyecto de desarrollo basado en esqueletos o plantillas `paster` y Buildout simplemente seleccione cual esqueleto va a utilizar para su desarrollo y proceso a instalarlo con easy_install[26] o PIP[27] (como se explico anteriormente) y siga sus respectivas instrucciones para lograr con éxito la tarea deseada.

Descarga código fuente

Para descargar el código fuente de este ejemplo ejecute el siguiente comando:

```
$ svn co https://svn.plone.org/svn/collective/spanishdocs/tags/0.1rc/src/mini-tutoriales
```

Referencias

- Gestión de proyectos con Buildout, instalando Zope/Plone con este mecanismo[28] desde la comunidad de Plone Venezuela.

[19] http://pypi.python.org/pypi/BlastOff/
[20] http://pypi.python.org/pypi/Pylons/
[21] http://pypi.python.org/pypi/CherryPy
[22] http://pypi.python.org/pypi/CherryPaste
[23] http://pypi.python.org/pypi/Trac
[24] http://trac-hacks.org/wiki/TracLegosScript
[25] http://trac-hacks.org/browser/traclegosscript/anyrelease/example/oss
[26] https://plone-spanish-docs.readthedocs.org/es/latest/python/setuptools.html#que-es-easyinstall
[27] https://plone-spanish-docs.readthedocs.org/es/latest/python/distribute_pip.html
[28] http://coactivate.org/projects/ploneve/gestion-de-proyectos-con-buildout

Errores y excepciones

Hasta ahora los mensajes de error no habían sido más que mencionados, pero si probaste los ejemplos probablemente hayas visto algunos. Hay (al menos) dos tipos diferentes de errores: *errores de sintaxis* y *excepciones*.

Errores de sintaxis

Los errores de sintaxis, también conocidos como errores de interpretación, son quizás el tipo de queja más común que tenés cuando todavía estás aprendiendo Python:

```
>>> while True print 'Hola mundo'
Traceback (most recent call last):
...
    while True print 'Hola mundo'
                   ^
SyntaxError: invalid syntax
```

El intérprete repite la línea culpable y muestra una pequeña 'flecha' que apunta al primer lugar donde se detectó el error. Este es causado por (o al menos detectado en) el símbolo que *precede* a la flecha: en el ejemplo, el error se detecta en el `print`[1], ya que faltan dos puntos (`':'`) antes del mismo. Se muestran el nombre del archivo y el número de línea para que sepas dónde mirar en caso de que la entrada venga de un programa.

Excepciones

Incluso si la declaración o expresión es sintácticamente correcta, puede generar un error cuando se intenta ejecutarla. Los errores detectados durante la ejecución se llaman *excepciones*, y no son incondicionalmente fatales: pronto aprenderás cómo manejarlos en los programas en Python. Sin embargo, la mayoría de las excepciones no son manejadas por los programas, y resultan en mensajes de error como los mostrados aquí:

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and 'int' objects
```

[1] http://docs.python.org/2/reference/simple_stmts.html#print

La última línea de los mensajes de error indica qué sucedió. Las excepciones vienen de distintos tipos, y el tipo se imprime como parte del mensaje: los tipos en el ejemplo son: `ZeroDivisionError`, `NameError` y `TypeError`. La cadena mostrada como tipo de la excepción es el nombre de la excepción predefinida que ocurrió. Esto es verdad para todas las excepciones predefinidas del intérprete, pero no necesita ser verdad para excepciones definidas por el usuario (aunque es una convención útil). Los nombres de las excepciones estándar son identificadores incorporados al intérprete (no son palabras clave reservadas).

El resto de la línea provee un detalle basado en el tipo de la excepción y qué la causó.

La parte anterior del mensaje de error muestra el contexto donde la excepción sucedió, en la forma de un *trazado del error* listando líneas fuente; sin embargo, no mostrará líneas leídas desde la entrada estándar.

Manejando excepciones

Es posible escribir programas que manejen determinadas excepciones. Mirá el siguiente ejemplo, que le pide al usuario una entrada hasta que ingrese un entero válido, pero permite al usuario interrumpir el programa (usando `Control-C` o lo que sea que el sistema operativo soporte); notá que una interrupción generada por el usuario se señaliza generando la excepción `KeyboardInterrupt`.

```
>>> while True:
...     try:
...         x = int(raw_input(u"Por favor ingrese un número: "))
...         break
...     except ValueError:
...         print u"Oops!  No era válido.  Intente nuevamente..."
...
```

La declaración `try`[2] funciona de la siguiente manera:

- Primero, se ejecuta el *bloque try* (el código entre las declaración `try`[3] y `except`[4]).
- Si no ocurre ninguna excepción, el *bloque except* se saltea y termina la ejecución de la declaración `try`[5].
- Si ocurre una excepción durante la ejecución del *bloque try*, el resto del bloque se saltea. Luego, si su tipo coincide con la excepción nombrada luego de la palabra reservada `except`[6], se ejecuta el *bloque except*, y la ejecución continúa luego de la declaración `try`[7].
- Si ocurre una excepción que no coincide con la excepción nombrada en el `except`[8], esta se pasa a declaraciones `try`[9] de más afuera; si no se encuentra nada que la maneje, es una *excepción no manejada*, y la ejecución se frena con un mensaje como los mostrados arriba.

Una declaración `try`[10] puede tener más de un `except`[11], para especificar manejadores para distintas excepciones. A lo sumo un manejador será ejecutado. Sólo se manejan excepciones que ocurren en el correspondiente `try`[12], no en otros manejadores del mismo `try`[13]. Un `except`[14] puede nombrar múltiples excepciones usando paréntesis, por ejemplo:

```
... except (RuntimeError, TypeError, NameError):
...     pass
```

[2] http://docs.python.org/2/reference/compound_stmts.html#try
[3] http://docs.python.org/2/reference/compound_stmts.html#try
[4] http://docs.python.org/2/reference/compound_stmts.html#except
[5] http://docs.python.org/2/reference/compound_stmts.html#try
[6] http://docs.python.org/2/reference/compound_stmts.html#except
[7] http://docs.python.org/2/reference/compound_stmts.html#try
[8] http://docs.python.org/2/reference/compound_stmts.html#except
[9] http://docs.python.org/2/reference/compound_stmts.html#try
[10] http://docs.python.org/2/reference/compound_stmts.html#try
[11] http://docs.python.org/2/reference/compound_stmts.html#except
[12] http://docs.python.org/2/reference/compound_stmts.html#try
[13] http://docs.python.org/2/reference/compound_stmts.html#try
[14] http://docs.python.org/2/reference/compound_stmts.html#except

El último except[15] puede omitir nombrar qué excepción captura, para servir como comodín. Usá esto con extremo cuidado, ya que de esta manera es fácil ocultar un error real de programación. También puede usarse para mostrar un mensaje de error y luego re-generar la excepción (permitiéndole al que llama, manejar también la excepción):

```
import sys

try:
    f = open('miarchivo.txt')
    s = f.readline()
    i = int(s.strip())
except IOError as (errno, strerror):
    print "Error E/S ({0}): {1}".format(errno, strerror)
except ValueError:
    print "No pude convertir el dato a un entero."
except:
    print "Error inesperado:", sys.exc_info()[0]
    raise
```

Las declaraciones try[16] ... except[17] tienen un *bloque else* opcional, el cual, cuando está presente, debe seguir a los except. Es útil para aquel código que debe ejecutarse si el *bloque try* no genera una excepción. Por ejemplo:

```
for arg in sys.argv[1:]:
    try:
        f = open(arg, 'r')
    except IOError:
        print 'no pude abrir', arg
    else:
        print arg, 'tiene', len(f.readlines()), 'lineas'
        f.close()
```

El uso de else[18] es mejor que agregar código adicional en el try[19] porque evita capturar accidentalmente una excepción que no fue generada por el código que está protegido por la declaración try[20] ... except[21].

Cuando ocurre una excepción, puede tener un valor asociado, también conocido como el *argumento* de la excepción. La presencia y el tipo de argumento depende del tipo de excepción.

El except[22] puede especificar una variable luego del nombre (o tupla) de excepción(es). La variable se vincula a una instancia de excepción con los argumentos almacenados en `instance.args`. Por conveniencia, la instancia de excepción define `__str__()` para que se pueda mostrar los argumentos directamente, sin necesidad de hacer referencia a `.args`.

Uno también puede instanciar una excepción antes de generarla, y agregarle cualquier atributo que se desee:

```
>>> try:
...     raise Exception('carne', 'huevos')
... except Exception as inst:
...     print type(inst)     # la instancia de excepción
...     print inst.args      # argumentos guardados en .args
...     print inst           # __str__ permite imprimir args directamente
...     x, y = inst          # __getitem__ permite usar args directamente
...     print 'x =', x
...     print 'y =', y
...
<type 'exceptions.Exception'>
```

[15] http://docs.python.org/2/reference/compound_stmts.html#except
[16] http://docs.python.org/2/reference/compound_stmts.html#try
[17] http://docs.python.org/2/reference/compound_stmts.html#except
[18] http://docs.python.org/2/reference/compound_stmts.html#else
[19] http://docs.python.org/2/reference/compound_stmts.html#try
[20] http://docs.python.org/2/reference/compound_stmts.html#try
[21] http://docs.python.org/2/reference/compound_stmts.html#except
[22] http://docs.python.org/2/reference/compound_stmts.html#except

```
('carne', 'huevos')
('carne', 'huevos')
x = carne
y = huevos
```

Si una excepción tiene un argumento, este se imprime como la última parte (el 'detalle') del mensaje para las excepciones que no están manejadas.

Los manejadores de excepciones no manejan solamente las excepciones que ocurren en el *bloque try*, también manejan las excepciones que ocurren dentro de las funciones que se llaman (inclusive indirectamente) dentro del *bloque try*. Por ejemplo:

```
>>> def esto_falla():
...     x = 1/0
...
>>> try:
...     esto_falla()
... except ZeroDivisionError as detail:
...     print 'Manejando error en tiempo de ejecucion:', detail
...
Manejando error en tiempo de ejecucion: integer division or modulo by zero
```

Levantando excepciones

La declaración `raise`[23] permite al programador forzar a que ocurra una excepción específica. Por ejemplo:

```
>>> raise NameError('Hola')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: Hola
```

El único argumento a `raise`[24] indica la excepción a generarse. Tiene que ser o una instancia de excepción, o una clase de excepción (una clase que hereda de `Exception`).

Si necesitás determinar cuando una excepción fue lanzada pero no querés manejarla, una forma simplificada de la instrucción `raise`[25] te permite relanzarla:

```
>>> try:
...     raise NameError('Hola')
... except NameError:
...     print u'Voló una excepción!'
...     raise
...
Voló una excpeción!
Traceback (most recent call last):
  File "<stdin>", line 2, in ?
NameError: Hola
```

Excepciones definidas por el usuario

Los programas pueden nombrar sus propias excepciones creando una nueva clase excepción (mirá el apartado de Clases[26] para más información sobre las clases de Python). Las excepciones, típicamente, deberán derivar de la clase `Exception`, directa o indirectamente. Por ejemplo:

[23] http://docs.python.org/2/reference/simple_stmts.html#raise
[24] http://docs.python.org/2/reference/simple_stmts.html#raise
[25] http://docs.python.org/2/reference/simple_stmts.html#raise
[26] http://docs.python.org.ar/tutorial/2/classes.html#tut-classes

```
>>> class MiError(Exception):
...     def __init__(self, valor):
...         self.valor = valor
...     def __str__(self):
...         return repr(self.valor)
...
>>> try:
...     raise MiError(2*2)
... except MyError as e:
...     print u'Ocurrió mi excepción, valor:', e.valor
...
Ocurrió mi excepción, valor: 4
>>> raise MiError('oops!')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
__main__.MiError: 'oops!'
```

En este ejemplo, el método `__init__()` de `Exception` fue sobrescrito. El nuevo comportamiento simplemente crea el atributo *valor*. Esto reemplaza el comportamiento por defecto de crear el atributo *args*.

Las clases de Excepciones pueden ser definidas de la misma forma que cualquier otra clase, pero usualmente se mantienen simples, a menudo solo ofreciendo un número de atributos con información sobre el error que leerán los manejadores de la excepción. Al crear un módulo que puede lanzar varios errores distintos, una práctica común es crear una clase base para excepciones definidas en ese módulo y extenderla para crear clases excepciones específicas para distintas condiciones de error:

```
class Error(Exception):
    """Clase base para excepciones en el modulo."""
    pass

class EntradaError(Error):
    """Excepcion lanzada por errores en las entradas.

    Atributos:
        expresion -- expresion de entrada en la que ocurre el error
        mensaje -- explicacion del error
    """

    def __init__(self, expresion, mensaje):
        self.expresion = expresion
        self.mensaje = mensaje

class TransicionError(Error):
    """Lanzada cuando una operacion intenta una transicion de estado no
    permitida.

    Atributos:
        previo -- estado al principio de la transicion
        siguiente -- nuevo estado intentado
        mensaje -- explicacion de porque la transicion no esta permitida
    """
    def __init__(self, previo, siguiente, mensaje):
        self.previo = previo
        self.siguiente = siguiente
        self.mensaje = mensaje
```

La mayoría de las excepciones son definidas con nombres que terminan en “Error”, similares a los nombres de las excepciones estándar.

Muchos módulos estándar definen sus propias excepciones para reportar errores que pueden ocurrir en funciones

propias. Se puede encontrar más información sobre clases en el capítulo Clases[27].

Definiendo acciones de limpieza

La declaración `try`[28] tiene otra cláusula opcional que intenta definir acciones de limpieza que deben ser ejecutadas bajo ciertas circunstancias. Por ejemplo:

```
>>> try:
...     raise KeyboardInterrupt
... finally:
...     print 'Chau, mundo!'
...
Chau, mundo!
KeyboardInterrupt
Traceback (most recent call last):
  File "<stdin>", line 2, in ?
```

Una *cláusula finally* siempre es ejecutada antes de salir de la declaración `try`[29], ya sea que una excepción haya ocurrido o no. Cuando ocurre una excepción en la cláusula `try`[30] y no fue manejada por una cláusula `except`[31] (o ocurrió en una cláusula `except`[32] o `else`[33]), es relanzada luego de que se ejecuta la cláusula `finally`[34]. `finally`[35] es también ejecutada "a la salida" cuando cualquier otra cláusula de la declaración `try`[36] es dejada via `break`[37], `continue`[38] or `return`[39]. Un ejemplo más complicado (cláusulas `except`[40] y `finally`[41] en la misma declaración `try`[42]):

```
>>> def dividir(x, y):
...     try:
...         result = x / y
...     except ZeroDivisionError:
...         print "¡division por cero!"
...     else:
...         print "el resultado es", result
...     finally:
...         print "ejecutando la clausula finally"
...
>>> dividir(2, 1)
el resultado es 2
ejecutando la clausula finally
>>> dividir(2, 0)
¡division por cero!
ejecutando la clausula finally
>>> divide("2", "1")
ejecutando la clausula finally
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
```

[27] http://docs.python.org.ar/tutorial/2/classes.html#tut-classes
[28] http://docs.python.org/2/reference/compound_stmts.html#try
[29] http://docs.python.org/2/reference/compound_stmts.html#try
[30] http://docs.python.org/2/reference/compound_stmts.html#try
[31] http://docs.python.org/2/reference/compound_stmts.html#except
[32] http://docs.python.org/2/reference/compound_stmts.html#except
[33] http://docs.python.org/2/reference/compound_stmts.html#else
[34] http://docs.python.org/2/reference/compound_stmts.html#finally
[35] http://docs.python.org/2/reference/compound_stmts.html#finally
[36] http://docs.python.org/2/reference/compound_stmts.html#try
[37] http://docs.python.org/2/reference/simple_stmts.html#break
[38] http://docs.python.org/2/reference/simple_stmts.html#continue
[39] http://docs.python.org/2/reference/simple_stmts.html#return
[40] http://docs.python.org/2/reference/compound_stmts.html#except
[41] http://docs.python.org/2/reference/compound_stmts.html#finally
[42] http://docs.python.org/2/reference/compound_stmts.html#try

```
  File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

Como podés ver, la cláusula `finally`[43] es ejecutada siempre. La excepción `TypeError` lanzada al dividir dos cadenas de texto no es manejado por la cláusula `except`[44] y por lo tanto es relanzada luego de que se ejecuta la cláusula `finally`[45].

En aplicaciones reales, la cláusula `finally`[46] es útil para liberar recursos externos (como archivos o conexiones de red), sin importar si el uso del recurso fue exitoso.

Acciones predefinidas de limpieza

Algunos objetos definen acciones de limpieza estándar que llevar a cabo cuando el objeto no es más necesitado, independientemente de que las operaciones sobre el objeto hayan sido exitosas o no. Mirá el siguiente ejemplo, que intenta abrir un archivo e imprimir su contenido en la pantalla.:

```
for linea in open("miarchivo.txt"):
    print linea
```

El problema con este código es que deja el archivo abierto por un periodo de tiempo indeterminado luego de que termine de ejecutarse. Esto no es un problema en scripts simples, pero puede ser un problema en aplicaciones más grandes. La declaración `with`[47] permite que objetos como archivos sean usados de una forma que asegure que siempre se los libera rápido y en forma correcta.

```
with open("miarchivo.txt") as f:
    for linea in f:
        print linea
```

Luego de que la declaración sea ejecutada, el archivo *f* siempre es cerrado, incluso si se encuentra un problema al procesar las líneas. Otros objetos que provean acciones de limpieza predefinidas lo indicarán en su documentación.

Vídeo tutorial

- Tutorial Python 13 - Clases y Objetos[48].

Referencia

- Clases — Tutorial de Python v2.7.0[49].

[43] http://docs.python.org/2/reference/compound_stmts.html#finally
[44] http://docs.python.org/2/reference/compound_stmts.html#except
[45] http://docs.python.org/2/reference/compound_stmts.html#finally
[46] http://docs.python.org/2/reference/compound_stmts.html#finally
[47] http://docs.python.org/2/reference/compound_stmts.html#with
[48] https://www.youtube.com/watch?v=VYXdpjCZojA
[49] http://docs.python.org.ar/tutorial/2/classes.html

CAPÍTULO 14

Programación orientada a objetos

El mecanismo de clases de Python agrega clases al lenguaje con un mínimo de nuevas sintaxis y semánticas. Es una mezcla de los mecanismos de clase encontrados en C++ y Modula-3. Como es cierto para los módulos, las clases en Python no ponen una barrera absoluta entre la definición y el usuario, sino que más bien se apoya en la cortesía del usuario de no "forzar la definición". Sin embargo, se mantiene el poder completo de las características más importantes de las clases: el mecanismo de la herencia de clases permite múltiples clases base, una clase derivada puede sobreescribir cualquier método de su(s) clase(s) base, y un método puede llamar al método de la clase base con el mismo nombre. Los objetos pueden tener una cantidad arbitraria de datos.

En terminología de C++, todos los miembros de las clases (incluyendo los miembros de datos), son *públicos*, y todas las funciones miembro son *virtuales*. Como en Modula-3, no hay atajos para hacer referencia a los miembros del objeto desde sus métodos: la función método se declara con un primer argumento explícito que representa al objeto, el cual se provee implícitamente por la llamada. Como en Smalltalk, las clases mismas son objetos. Esto provee una semántica para importar y renombrar. A diferencia de C++ y Modula-3, los tipos de datos integrados pueden usarse como clases base para que el usuario los extienda. También, como en C++ pero a diferencia de Modula-3, la mayoría de los operadores integrados con sintaxis especial (operadores aritméticos, de subíndice, etc.) pueden ser redefinidos por instancias de la clase.

(Sin haber una terminología universalmente aceptada sobre clases, haré uso ocasional de términos de Smalltalk y C++. Usaría términos de Modula-3, ya que su semántica orientada a objetos es más cercana a Python que C++, pero no espero que muchos lectores hayan escuchado hablar de él).

Ejemplo de POO

Ejemplo de la clase Persona:

```
class Persona:

        def __init__(self):
            print "soy un nuevo objeto"
```

Ejemplo de la clase Persona con funcion interna:

```
class Persona:

        def __init__(self):
                print "soy un nuevo objeto"

        def hablar(self, mensaje):
                print mensaje
```

Vídeo tutorial

- Tutorial Python 13 - Clases y Objetos[1].

Referencia

- Clases — Tutorial de Python v2.7.0[2].

[1] https://www.youtube.com/watch?v=VYXdpjCZojA
[2] http://docs.python.org.ar/tutorial/2/classes.html

Listas de comprensión

Introducción a Listas de comprensión

TODO.

Usando Listas de comprensión con Archivos

TODO.

Vídeo tutorial

- Tutorial Python 25 - Comprensión de Listas[1].

[1] https://www.youtube.com/watch?v=87s8XQbUv1k

CAPÍTULO 16

Iteradores

Entendiendo Iteradores

Simplicidad

La duplicación del esfuerzo es un derroche y reemplazar varios de los enfoques propios con una característica estándar, normalmente, deriva en hacer las cosas más legibles además de más interoperable.

Guido van Rossum — Añadiendo tipado estático opcional a Python (Adding Optional Static Typing to Python[a])

[a] http://www.artima.com/weblogs/viewpost.jsp?thread=86641

Un iterador es un objeto adherido al 'protocolo de iterador' (iterator protocol[1]) — básicamente esto significa que tiene un método *next* <*iterator.next*> ('next' por siguiente), el cual, cuando se le llama, devuelve la siguiente 'pieza' (o 'item') en la secuencia y, cuando no queda nada para ser devuelto, lanza la excepción *StopIteration* <*exceptions.StopIteration*>.

```
>>> nums = [1,2,3]
>>> iter(nums)
<listiterator object at 0xb712ebec>
>>> nums.__iter__()
<listiterator object at 0xb712eb0c>
>>> nums.__reversed__()
<listreverseiterator object at 0xb712ebec>
```

Usando 'iter' y 'next'

Cuando se usa en un bucle, finalmente se llama a StopIteration y se provoca la finalización del bucle. Pero si se invoca de forma explícita podemos ver que, una vez que el iterador está 'agotado', al invocarlo nuevamente veremos que se lanza la excepción comentada anteriormente.

```
>>> it = iter(nums)
>>> it.next()
1
>>> it.next()
2
>>> it.next()
3
>>> it.next()
Traceback (most recent call last):
```

[1] http://docs.python.org/dev/library/stdtypes.html#iterator-types

```
  File "<stdin>", line 1, in <module>
StopIteration
>>> f = open('/etc/fstab')
>>> f is f.__iter__()
True
```

- Iteradores y Diccionarios.
- Otros Iteradores.
- Ejercicio 1.

Vídeo tutorial

- Tutorial Python 25 - Comprensión de Listas[2].
- Tutorial Python 26 - Generadores[3].
- Tutorial Python 27 - Decoradores[4].

Advertencia: Tenga en cuenta que este documento no está completo sin la palabra hablada de un instructor a pesar de que tratamos de incluir las partes más importantes de lo que enseñamos en la narrativa no puede considerarse completa sin la palabra hablada.

[2] https://www.youtube.com/watch?v=87s8XQbUv1k
[3] https://www.youtube.com/watch?v=tvHbC_OZV14
[4] https://www.youtube.com/watch?v=TaIWx9paNIA

CAPÍTULO 17

Apéndices

Glosario

Sobre este artículo

Autor(es) Leonardo J. Caballero G.
Correo(s) leonardoc@plone.org[a]
Compatible con Python 2.x, Python 3.x
Fecha 26 de Marzo de 2015

[a] leonardoc@plone.org

A continuación una serie de términos usados en las tecnologías Python / Zope / Plone

buildout En la herramienta buildout[1], es un conjunto de partes que describe como ensamblar una aplicación.

bundle Ver *Paquete bundle*.

Catalog Sinónimo en Ingles del termino *Catálogo*.

Catálogo Es un índice interno de los contenidos dentro de Plone para que se pueda buscar. El objetivo del catálogo es que sea accesible a través de la ZMI[2] a través de la herramienta portal_catalog[3].

Cheese shop Ver *PyPI*.

Collective Es un repositorio de código comunitario, para Productos Plone y productos de terceros, y es un sitio muy útil para buscar la ultima versión de código fuente del producto para cientos de productos de terceros a Plone. Los desarrolladores de nuevos productos de Plone son animados a compartir su código a través de Collective para que otros puedan encontrarlo, usarlo, y contribuir con correcciones / mejoras.

En la actualidad la comunidad ofrece dos repositorio Collective un basado en **Git** y otro **Subversion**.

Si usted quiere publicar un nuevo producto en el repositorio *Git de Collective* de Plone necesita obtener acceso de escritura[4] y seguir las reglas en github/collective, también puede consultarlo en la cuenta en github.com[5].

Si usted quiere publicar un nuevo producto en el repositorio *Subversion de Collective* de Plone necesita obtener acceso de escritura al repositorio[6] y crear su estructura básica de repositorio[7] para su producto, también puede consultarlo vía Web consulte el siguiente enlace[8].

Declaración ZCML El uso concreto de una *Directiva ZCML* dentro de un archivo *ZCML*.

[1] https://plone-spanish-docs.readthedocs.org/es/latest/buildout/replicacion_proyectos_python.html#que-es-zc-buildout
[2] https://plone-spanish-docs.readthedocs.org/es/latest/zope/zmi/index.html
[3] https://plone-spanish-docs.readthedocs.org/es/latest/zope/zmi/index.html#portal-catalog
[4] http://collective.github.io/
[5] http://github.com/collective
[6] http://plone.org/countries/conosur/documentacion/obtener-acceso-de-escritura-al-repositorio-svn-de-plone
[7] http://plone.org/countries/conosur/documentacion/crear-un-nuevo-proyecto-en-el-repositorio-collective-de-plone
[8] http://svn.plone.org/svn/collective/

Directiva ZCML Una "etiqueta" *ZCML* como `<include />` o `<utility />`.

Egg Ver *paquetes Egg*.

esqueleto Los archivos y carpetas recreados por un usuario el cual los genero ejecutando alguna plantilla `templer` (`PasteScript`).

estructura 1) Una clase Python la cual controla la generación de un árbol de carpetas que contiene archivos.

2) Una unidad de carpetas y archivos proveídos por el sistema `templer` para ser usado en una plantilla o plantillas. Las estructuras proporcionan recursos estáticos compartidos, que pueden ser utilizados por cualquier paquete en el sistema de `templer`.

Las estructuras diferencian de las plantillas en que no proporcionan las *vars*.

filesystem Termino ingles File system, referido al sistema de archivo del sistema operativo.

Flujo de trabajo Es una forma muy poderosa de imitar los procesos de negocio de su organización, es también la forma en se manejan la configuración de seguridad de Plone.

Flujo de trabajos Plural del termino *Flujo de trabajo*.

grok Ver la documentacion del proyecto grok[9].

Instalación de Zope El software propio del servidor de aplicaciones.

Instancia de Zope Un directorio específico que contiene una configuración completa del servidor Zope.

local command Una clase Paste[10] la cual provee funcionalidad adicional a una estructura de esqueleto de proyecto que ha sido generada.

modulo Del Ingles `module`, es un archivo fuente Python; un archivo en el sistema de archivo que típicamente finaliza con la extensión `.py` o `.pyc`. Los modules son parte de un *paquete*.

Nombre de puntos Python Es la representación Python del "camino" para un determinado objeto / módulo / función, por ejemplo, `Products.GenericSetup.tool.exportToolset`. A menudo se utiliza como referencia en configuraciones `Paste` y `setuptools` a cosas en Python.

paquete Ver *Paquete Python*.

Paquete bundle Este paquete consististe en un archivo comprimido con todos los módulos que son necesario compilar o instalar en el *PYTHONPATH* de tu interprete `Python`.

paquete Egg Es una forma de empaquetar y distribuir paquetes Python. Cada Egg contiene un archivo `setup.py` con metadata (como el nombre del autor y la correo electrónico y información sobre el licenciamiento), como las dependencias del paquete.

La herramienta del *setuptools <que_es_setuptools>*, es la librería Python que permite usar el mecanismo de paquetes egg, esta es capaz de encontrar y descargar automáticamente las dependencias de los paquetes Egg que se instale.

Incluso es posible que dos paquetes Egg diferentes necesiten utilizar simultáneamente diferentes versiones de la misma dependencia. El formato de paquetes Eggs también soportan una función llamada `entry points`, una especie de mecanismo genérico de plug-in. Mucha más detalle sobre este tema se encuentra disponible en el sitio web de PEAK[11].

Paquete Python Es un termino generalmente usando para describir un módulo Python. en el más básico nivel, un paquete es un directorio que contiene un archivo `__init__.py` y algún código Python.

paquetes Egg Plural del termino *paquete Egg*.

Paquetes Python Plural del termino *Paquete Python*.

part En la herramienta *buildout*, es un conjunto opciones que le permite a usted construir una pieza de la aplicación.

[9] http://grok.zope.org/
[10] http://pythonpaste.org/
[11] http://peak.telecommunity.com/DevCenter/setuptools

plantilla 1) Una clase Python la cual controla la generación de un esqueleto. Las plantillas contiene una lista de variables para obtener la respuesta de un usuario. Las plantillas son ejecutadas con el comando `templer` suministrando el nombre de la plantilla como un argumento `templer basic_namespace my.package`.

2) Los archivos y carpetas proveídas un paquete `templer` como contenido a ser generado. Las respuestas proporcionadas por un usuario en respuesta a las variables se utilizan para rellenar los marcadores de posición en este contenido.

Producto Es una terminología usada por la comunidad Zope / Plone asociada a cualquier implementación de módulos / complementos y agregados que amplíen la funcionalidad por defecto que ofrece Zope / Plone. También son conocidos como *"Productos de terceros"* del Ingles Third-Party Products[12].

Producto Plone Es un tipo especial de paquete Zope usado para extender las funcionalidades de Plone. Se puede decir que son productos que su ámbito de uso es solo en el desde la interfaz gráfica de Plone.

Producto Zope Es un tipo especial de paquete Python usado para extender Zope. En las antiguas versiones de Zope, todos los productos eran carpetas que se ubican dentro de una carpeta especial llamada `Products` de una instancia Zope; estos tendrían un nombre de módulo Python que empiezan por "**Products.**". Por ejemplo, el núcleo de Plone es un producto llamado `CMFPlone`, conocido en Python como Products.CMFPlone[13].

Este tipo de productos esta disponibles desde la interfaz administrativa de Zope (ZMI)[14] de su instalación[15] donde deben acceder con las credenciales del usuario Administrador de Zope. Muchas veces el producto simplemente no hay que instalarlo por que se agregar automáticamente.

Productos Plural del termino *Producto*.

Productos Plone Plural del termino *Producto Plone*.

Productos Zope Plural del termino *Producto Zope*.

profile Una configuración "predeterminada" de un sitio, que se define en el sistema de archivos o en un archivo tar.

PyPI Siglas del termino en Ingles *Python Package Index*, es el servidor central de *paquetes Egg* Python ubicado en la dirección http://pypi.python.org/pypi/.

Python Package Index Ver *PyPI*.

PYTHONPATH Una lista de nombre de directorios, que contiene librerías Python, con la misma sintaxis como la declarativa `PATH` del shell del sistema operativo.

recipe En la herramienta *buildout*, es el software usado para crear partes de una instalación basada en sus opciones. Mas información consulte el articulo Recipes Buildout[16].

setup.py El archivo `setup.py` es un modulo de Python, que por lo general indica que el módulo / paquete que está a punto de instalar ha sido empacado y distribuidos con `Distutils`, que es el estándar para la distribución de módulos de Python.

Con esto le permite instalar fácilmente paquetes de Python, a menudo es suficiente para escribir:

```
python setup.py install
```

Entonces el módulo Python se instalará.

Ver también:

- http://docs.python.org/install/index.html

Temas / Apariencias Por lo general si un producto de Tema esta bien diseñado y implementado debe aplicarse de una ves al momento de instalarlo. En caso que no se aplique de una puede acceder a la sección Configuración de Temas[17] y cambiar el **Tema predeterminado** por el de su gusto.

[12] http://plone.org/documentation/kb/add-ons/tutorial-all-pages
[13] http://pypi.python.org/pypi/Products.CMFPlone
[14] https://plone-spanish-docs.readthedocs.org/es/latest/zope/zmi/index.html
[15] http://localhost:8080/manage
[16] https://plone-spanish-docs.readthedocs.org/es/latest/buildout/recipes.html
[17] http://localhost:8080/Plone/@@skins-controlpanel

Tipos de contenidos Los tipos de contenidos son productos que extienden la funcionalidad de **Agregar elemento** que permite agregar nuevos tipos de registros (Contenidos) a tu sitio. Esto quiere decir que si instala un tipo de contenido exitosamente debería poder acceder a usarlo desde el menú de **Agregar elemento** en el sitio Plone. Opcionalmente algunos productos instalan un panel de control del producto que puede acceder a este en la sección Configuración de Productos Adicionales[18].

var Diminutivo en singular del termino *variable*.

variable 1) Una pregunta que debe ser respondida por el usuario cuando esta generando una estructura de esqueleto de proyecto usando el sistema de plantilla `templer`. En este caso una variable (var) es una descripción de la información requerida, texto de ayuda y reglas de validación para garantizar la entrada de usuario correcta.

2) Una declarativa cuyo valor puede ser variable o constante dentro de un programa Python o en el sistema operativo.

variables Plural del termino *variable*.

vars Diminutivo en plural del termino *variable*.

Workflow Ver *Flujo de trabajo*.

ZCA, Zope Component Architecture La arquitectura de componentes de Zope (alias ZCA)[19], es un sistema que permite la aplicación y la expedición enchufabilidad complejo basado en objetos que implementan una interfaz.

ZCatalog Ver *Catalog*.

ZCML Siglas del termino en Ingles *Zope Configuration Mark-up Language*.

ZCML-slug Los así llamados "ZCML-slugs", era configuraciones que estaban destinados a enlazar dentro de un directorio una configuración especial en una instalación de Zope, por lo general se ven como `collective.foo-configure.zcml`. Estas configuraciones ya no están más en uso, pueden ser eliminados agregando las configuraciones del paquete z3c.autoinclude[20].

Zope Configuration Mark-up Language Es un dialecto XML utilizado por Zope para las tareas de configuración. ZCML es capaz de realizar diferentes tipos de declaración de configuración. Es utilizado para extender y conectar a los sistemas basados en la *Zope Component Architecture*.

`Zope 3` tiene la política de separar el código actial y moverlo a los archivos de configuración independientes, típicamente un archivo `configure.zcml` en un buildout. Este archivo configura la instancia Zope. El concepto 'Configuración' podría ser un poco engañoso aquí y debe ser pensado o tomarse más cableado.

`ZCML`, el lenguaje de configuración basado en `XML` que se utiliza para esto, se adapta a hacer el registro de componentes y declaraciones de seguridad, en su mayor parte. Al habilitar o deshabilitar ciertos componentes en ZCML, puede configurar ciertas políticas de la aplicación general. En `Zope 2`, habilitar y deshabilitar componentes significa eliminar o remover un determinado producto `Zope 2`. Cuando está ahí, se importa y se carga automáticamente. Este no es el caso en `Zope 3` Si no habilita explícitamente, no va a ser encontrado.

El *grok* proyecto ha adoptado un enfoque diferente para el mismo problema, y permite el registro de componentes, etc haciendo declarativa de código Python. Ambos enfoques son posibles en Plone.

Licenciamientos

Reconocimiento-CompartirIgual 3.0 Venezuela de Creative Commons

[18] http://localhost:8080/Plone/prefs_install_products_form
[19] https://plone-spanish-docs.readthedocs.org/es/latest/programacion/zca/zca-es.html#zca-es
[20] http://pypi.python.org/pypi/z3c.autoinclude

CAPÍTULO 18

Índices y tablas

- *genindex*
- *search*

APÉNDICE A

Glosario

Sobre este artículo

Autor(es) Leonardo J. Caballero G.
Correo(s) leonardoc@plone.org[a]
Compatible con Python 2.x, Python 3.x
Fecha 26 de Marzo de 2015

[a] leonardoc@plone.org

A continuación una serie de términos usados en las tecnologías Python / Zope / Plone

buildout En la herramienta buildout[1], es un conjunto de partes que describe como ensamblar una aplicación.

bundle Ver *Paquete bundle*.

Catalog Sinónimo en Ingles del termino *Catálogo*.

Catálogo Es un índice interno de los contenidos dentro de Plone para que se pueda buscar. El objetivo del catálogo es que sea accesible a través de la ZMI[2] a través de la herramienta portal_catalog[3].

Cheese shop Ver *PyPI*.

Collective Es un repositorio de código comunitario, para Productos Plone y productos de terceros, y es un sitio muy útil para buscar la ultima versión de código fuente del producto para cientos de productos de terceros a Plone. Los desarrolladores de nuevos productos de Plone son animados a compartir su código a través de Collective para que otros puedan encontrarlo, usarlo, y contribuir con correcciones / mejoras.

En la actualidad la comunidad ofrece dos repositorio Collective un basado en **Git** y otro **Subversion**.

Si usted quiere publicar un nuevo producto en el repositorio *Git de Collective* de Plone necesita obtener acceso de escritura[4] y seguir las reglas en github/collective, también puede consultarlo en la cuenta en github.com[5].

Si usted quiere publicar un nuevo producto en el repositorio *Subversion de Collective* de Plone necesita obtener acceso de escritura al repositorio[6] y crear su estructura básica de repositorio[7] para su producto, también puede consultarlo vía Web consulte el siguiente enlace[8].

Declaración ZCML El uso concreto de una *Directiva ZCML* dentro de un archivo *ZCML*.

Directiva ZCML Una "etiqueta" *ZCML* como `<include />` o `<utility />`.

Egg Ver *paquetes Egg*.

[1] https://plone-spanish-docs.readthedocs.org/es/latest/buildout/replicacion_proyectos_python.html#que-es-zc-buildout
[2] https://plone-spanish-docs.readthedocs.org/es/latest/zope/zmi/index.html
[3] https://plone-spanish-docs.readthedocs.org/es/latest/zope/zmi/index.html#portal-catalog
[4] http://collective.github.io/
[5] http://github.com/collective
[6] http://plone.org/countries/conosur/documentacion/obtener-acceso-de-escritura-al-repositorio-svn-de-plone
[7] http://plone.org/countries/conosur/documentacion/crear-un-nuevo-proyecto-en-el-repositorio-collective-de-plone
[8] http://svn.plone.org/svn/collective/

esqueleto Los archivos y carpetas recreados por un usuario el cual los genero ejecutando alguna plantilla `templer (PasteScript)`.

estructura 1) Una clase Python la cual controla la generación de un árbol de carpetas que contiene archivos.

2) Una unidad de carpetas y archivos proveídos por el sistema `templer` para ser usado en una plantilla o plantillas. Las estructuras proporcionan recursos estáticos compartidos, que pueden ser utilizados por cualquier paquete en el sistema de `templer`.

Las estructuras diferencian de las plantillas en que no proporcionan las *vars*.

filesystem Termino ingles File system, referido al sistema de archivo del sistema operativo.

Flujo de trabajo Es una forma muy poderosa de imitar los procesos de negocio de su organización, es también la forma en se manejan la configuración de seguridad de Plone.

Flujo de trabajos Plural del termino *Flujo de trabajo*.

grok Ver la documentacion del proyecto grok[9].

Instalación de Zope El software propio del servidor de aplicaciones.

Instancia de Zope Un directorio específico que contiene una configuración completa del servidor Zope.

local command Una clase Paste[10] la cual provee funcionalidad adicional a una estructura de esqueleto de proyecto que ha sido generada.

modulo Del Ingles `module`, es un archivo fuente Python; un archivo en el sistema de archivo que típicamente finaliza con la extensión `.py` o `.pyc`. Los modules son parte de un *paquete*.

Nombre de puntos Python Es la representación Python del "camino" para un determinado objeto / módulo / función, por ejemplo, `Products.GenericSetup.tool.exportToolset`. A menudo se utiliza como referencia en configuraciones `Paste` y `setuptools` a cosas en Python.

paquete Ver *Paquete Python*.

Paquete bundle Este paquete consististe en un archivo comprimido con todos los módulos que son necesario compilar o instalar en el *PYTHONPATH* de tu interprete `Python`.

paquete Egg Es una forma de empaquetar y distribuir paquetes Python. Cada Egg contiene un archivo `setup.py` con metadata (como el nombre del autor y la correo electrónico y información sobre el licenciamiento), como las dependencias del paquete.

La herramienta del *setuptools* <*que_es_setuptools*>, es la librería Python que permite usar el mecanismo de paquetes egg, esta es capaz de encontrar y descargar automáticamente las dependencias de los paquetes Egg que se instale.

Incluso es posible que dos paquetes Egg diferentes necesiten utilizar simultáneamente diferentes versiones de la misma dependencia. El formato de paquetes Eggs también soportan una función llamada `entry points`, una especie de mecanismo genérico de plug-in. Mucha más detalle sobre este tema se encuentra disponible en el sitio web de PEAK[11].

Paquete Python Es un termino generalmente usando para describir un módulo Python. en el más básico nivel, un paquete es un directorio que contiene un archivo `__init__.py` y algún código Python.

paquetes Egg Plural del termino *paquete Egg*.

Paquetes Python Plural del termino *Paquete Python*.

part En la herramienta *buildout*, es un conjunto opciones que le permite a usted construir una pieza de la aplicación.

plantilla 1) Una clase Python la cual controla la generación de un esqueleto. Las plantillas contiene una lista de variables para obtener la respuesta de un usuario. Las plantillas son ejecutadas con el comando `templer` suministrando el nombre de la plantilla como un argumento `templer basic_namespace my.package`.

[9] http://grok.zope.org/
[10] http://pythonpaste.org/
[11] http://peak.telecommunity.com/DevCenter/setuptools

2) Los archivos y carpetas proveídas un paquete `templer` como contenido a ser generado. Las respuestas proporcionadas por un usuario en respuesta a las variables se utilizan para rellenar los marcadores de posición en este contenido.

Producto Es una terminología usada por la comunidad Zope / Plone asociada a cualquier implementación de módulos / complementos y agregados que amplíen la funcionalidad por defecto que ofrece Zope / Plone. También son conocidos como *"Productos de terceros"* del Ingles Third-Party Products[12].

Producto Plone Es un tipo especial de paquete Zope usado para extender las funcionalidades de Plone. Se puede decir que son productos que su ámbito de uso es solo en el desde la interfaz gráfica de Plone.

Producto Zope Es un tipo especial de paquete Python usado para extender Zope. En las antiguas versiones de Zope, todos los productos eran carpetas que se ubican dentro de una carpeta especial llamada `Products` de una instancia Zope; estos tendrían un nombre de módulo Python que empiezan por "**Products.**". Por ejemplo, el núcleo de Plone es un producto llamado `CMFPlone`, conocido en Python como Products.CMFPlone[13].

Este tipo de productos esta disponibles desde la interfaz administrativa de Zope (ZMI)[14] de su instalación[15] donde deben acceder con las credenciales del usuario Administrador de Zope. Muchas veces el producto simplemente no hay que instalarlo por que se agregar automáticamente.

Productos Plural del termino *Producto*.

Productos Plone Plural del termino *Producto Plone*.

Productos Zope Plural del termino *Producto Zope*.

profile Una configuración "predeterminada" de un sitio, que se define en el sistema de archivos o en un archivo tar.

PyPI Siglas del termino en Ingles *Python Package Index*, es el servidor central de *paquetes Egg* Python ubicado en la dirección http://pypi.python.org/pypi/.

Python Package Index Ver *PyPI*.

PYTHONPATH Una lista de nombre de directorios, que contiene librerías Python, con la misma sintaxis como la declarativa `PATH` del shell del sistema operativo.

recipe En la herramienta *buildout*, es el software usado para crear partes de una instalación basada en sus opciones. Mas información consulte el articulo Recipes Buildout[16].

setup.py El archivo `setup.py` es un modulo de Python, que por lo general indica que el módulo / paquete que está a punto de instalar ha sido empacado y distribuidos con `Distutils`, que es el estándar para la distribución de módulos de Python.

Con esto le permite instalar fácilmente paquetes de Python, a menudo es suficiente para escribir:

```
python setup.py install
```

Entonces el módulo Python se instalará.

Ver también:

- http://docs.python.org/install/index.html

Temas / Apariencias Por lo general si un producto de Tema esta bien diseñado y implementado debe aplicarse de una ves al momento de instalarlo. En caso que no se aplique de una puede acceder a la sección Configuración de Temas[17] y cambiar el **Tema predeterminado** por el de su gusto.

Tipos de contenidos Los tipos de contenidos son productos que extienden la funcionalidad de **Agregar elemento** que permite agregar nuevos tipos de registros (Contenidos) a tu sitio. Esto quiere decir que si instala un tipo de contenido exitosamente debería poder acceder a usarlo desde el menú de **Agregar elemento** en el

[12] http://plone.org/documentation/kb/add-ons/tutorial-all-pages
[13] http://pypi.python.org/pypi/Products.CMFPlone
[14] https://plone-spanish-docs.readthedocs.org/es/latest/zope/zmi/index.html
[15] http://localhost:8080/manage
[16] https://plone-spanish-docs.readthedocs.org/es/latest/buildout/recipes.html
[17] http://localhost:8080/Plone/@@skins-controlpanel

sitio Plone. Opcionalmente algunos productos instalan un panel de control del producto que puede acceder a este en la sección Configuración de Productos Adicionales[18].

var Diminutivo en singular del termino *variable*.

variable 1) Una pregunta que debe ser respondida por el usuario cuando esta generando una estructura de esqueleto de proyecto usando el sistema de plantilla `templer`. En este caso una variable (var) es una descripción de la información requerida, texto de ayuda y reglas de validación para garantizar la entrada de usuario correcta.

2) Una declarativa cuyo valor puede ser variable o constante dentro de un programa Python o en el sistema operativo.

variables Plural del termino *variable*.

vars Diminutivo en plural del termino *variable*.

Workflow Ver *Flujo de trabajo*.

ZCA, Zope Component Architecture La arquitectura de componentes de Zope (alias ZCA)[19], es un sistema que permite la aplicación y la expedición enchufabilidad complejo basado en objetos que implementan una interfaz.

ZCatalog Ver *Catalog*.

ZCML Siglas del termino en Ingles *Zope Configuration Mark-up Language*.

ZCML-slug Los así llamados "ZCML-slugs", era configuraciones que estaban destinados a enlazar dentro de un directorio una configuración especial en una instalación de Zope, por lo general se ven como `collective.foo-configure.zcml`. Estas configuraciones ya no están más en uso, pueden ser eliminados agregando las configuraciones del paquete z3c.autoinclude[20].

Zope Configuration Mark-up Language Es un dialecto XML utilizado por Zope para las tareas de configuración. ZCML es capaz de realizar diferentes tipos de declaración de configuración. Es utilizado para extender y conectar a los sistemas basados en la *Zope Component Architecture*.

`Zope 3` tiene la política de separar el código actial y moverlo a los archivos de configuración independientes, típicamente un archivo `configure.zcml` en un buildout. Este archivo configura la instancia Zope. El concepto 'Configuración' podría ser un poco engañoso aquí y debe ser pensado o tomarse más cableado.

`ZCML`, el lenguaje de configuración basado en `XML` que se utiliza para esto, se adapta a hacer el registro de componentes y declaraciones de seguridad, en su mayor parte. Al habilitar o deshabilitar ciertos componentes en ZCML, puede configurar ciertas políticas de la aplicación general. En `Zope 2`, habilitar y deshabilitar componentes significa eliminar o remover un determinado producto `Zope 2`. Cuando está ahí, se importa y se carga automáticamente. Este no es el caso en `Zope 3` Si no habilita explícitamente, no va a ser encontrado.

El *grok* proyecto ha adoptado un enfoque diferente para el mismo problema, y permite el registro de componentes, etc haciendo declarativa de código Python. Ambos enfoques son posibles en Plone.

[18] http://localhost:8080/Plone/prefs_install_products_form
[19] https://plone-spanish-docs.readthedocs.org/es/latest/programacion/zca/zca-es.html#zca-es
[20] http://pypi.python.org/pypi/z3c.autoinclude

Licenciamientos

Reconocimiento-CompartirIgual 3.0 Venezuela de Creative Commons

Sobre esta licencia

Esta documentación se distribuye bajo los términos de la licencia Reconocimiento-CompartirIgual 3.0 Venezuela de Creative Commons[a].

[a] http://creativecommons.org/licenses/by-sa/3.0/ve/

Usted es libre de:

- Compartir — copiar y redistribuir el material en cualquier medio o formato.
- Adaptar — remezclar, transformar y crear a partir del material.
- Para cualquier propósito, incluso comercialmente.
- El licenciante no puede revocar estas libertades en tanto usted siga los términos de la licencia.

Bajo los siguientes términos:

- Reconocimiento - Usted debe dar el crédito apropiado, proporcionar un enlace a la licencia, y de indicar si se han realizado cambios. Usted puede hacerlo de cualquier manera razonable, pero no en una manera que sugiere el licenciante a usted o que apruebe su utilización.
- CompartirIgual - Si usted mezcla, transforma o crea nuevo material a partir de esta obra, usted podrá distribuir su contribución siempre que utilice la misma licencia que la obra original.

Términos de la licencia: http://creativecommons.org/licenses/by-sa/3.0/ve/