The Python Programming Language (Supplementary Material)

Hemanth P. Sethuram

12 Mar 2019

Outline

Regular Expression Module

- Regular-expression are patterns specified as strings containing a mix of text and special-character sequences.

Regular Expression Module

- Regular-expression are patterns specified as strings containing a mix of text and special-character sequences.
- `re` module provides facilities for regular expression pattern matching and replacement in strings.

Regular Expression Module

- Regular-expression are patterns specified as strings containing a mix of text and special-character sequences.
- `re` module provides facilities for regular expression pattern matching and replacement in strings.
- While using regular expressions, it is recommended to use raw strings, such as, `r'(?P<int>\d+)\.(\d*)'`.

Regular Expression Module

- Regular-expression are patterns specified as strings containing a mix of text and special-character sequences.
- `re` module provides facilities for regular expression pattern matching and replacement in strings.
- While using regular expressions, it is recommended to use raw strings, such as, `r'(?P<int>\d+)\.(\d*)'`.
- Consult the Python manual to know how to construct regular expressions. For a more exhaustive reference, see the book, "Mastering Regular Expressions', by Jeffrey E.F. Friedl.

Methods for using Regular Expressions using re

- `compile(str [, flags])` compiles a regular-expression pattern string into a regular-expression object.

Methods for using Regular Expressions using re

- `compile(str [, flags])` compiles a regular-expression pattern string into a regular-expression object.
- `findall(pattern, string [,flags])` returns a list of all nonoverlapping matches of `pattern` in `string`, including empty matches. If the pattern has groups, a list of the text matched by the groups is returned.

Methods for using Regular Expressions using re

- `compile(str [, flags])` compiles a regular-expression pattern string into a regular-expression object.
- `findall(pattern, string [,flags])` returns a list of all nonoverlapping matches of `pattern` in `string`, including empty matches. If the pattern has groups, a list of the text matched by the groups is returned.
- `match(pattern, string [, flags])` checks whether zero or more characters at the beginning of `string` match `pattern.` Returns a `MatchObject` on success or `None` otherwise.

Methods for using Regular Expressions using re

- `compile(str [, flags])` compiles a regular-expression pattern string into a regular-expression object.
- `findall(pattern, string [,flags])` returns a list of all nonoverlapping matches of `pattern` in `string`, including empty matches. If the pattern has groups, a list of the text matched by the groups is returned.
- `match(pattern, string [, flags])` checks whether zero or more characters at the beginning of `string` match `pattern.` Returns a `MatchObject` on success or `None` otherwise.
- `search(pattern, string [, flags])` Searches `string` for the first match of `pattern.` Returns a `MatchObject` on success or `None` if no match was found.

Methods for using Regular Expressions using re ..2

- `split(pattern, string [, maxsplit = 0])` splits `string` by the occurrences of `pattern.` Returns a list of strings including the text matched by any groups in the pattern. `maxsplit` is the maximum number of splits to perform.

Methods for using Regular Expressions using re ..2

- `split(pattern, string [, maxsplit = 0])` splits `string` by the occurrences of `pattern.` Returns a list of strings including the text matched by any groups in the pattern. `maxsplit` is the maximum number of splits to perform.
- `sub(pattern, repl, string [, count = 0])` replaces the leftmost nonoverlapping occurrences of `pattern` in `string` by using the replacement `repl. repl` can be a string or a function.

Methods for using Regular Expressions using re ..2

- `split(pattern, string [, maxsplit = 0])` splits `string` by the occurrences of `pattern.` Returns a list of strings including the text matched by any groups in the pattern. `maxsplit` is the maximum number of splits to perform.
- `sub(pattern, repl, string [, count = 0])` replaces the leftmost nonoverlapping occurrences of `pattern` in `string` by using the replacement `repl. repl` can be a string or a function.

Note: The match objects and compiled objects have methods you can use to query and access different parts of the patterns and strings.

A Regular Expressions Example


```
1  import re
2  text = "Harry will be out of the office from 12/15/2012 - 1/3/2013."
3
4  # A regex pattern for a date.
5  datepat = re.compile('(\d+)/(\d+)/(\d+)')
6
7  # Find and print all dates
8  for m in datepat.finditer(text):
9      print(m.group())
10
11 # Find all dates, but print in a different format
12 monthnames = [None,'Jan','Feb','Mar','Apr','May','Jun',
13                'Jul','Aug','Sep','Oct','Nov','Dec']
14 for m in datepat.finditer(text):
15     print("%s %s, %s" % (monthnames[int(m.group(1)],
16         m.group(2), m.group(3)))
17
18 # Replace all dates with fields in the European format (day/month/ye
19 def fix_date(m):
20     return "%s/%s/%s" % (m.group(2),m.group(1),m.group(3))
21 newtext = datepat.sub(fix_date, text)
22
23 # An alternative replacement
24 newtext = datepat.sub(r'\2/\1/\3', text)
```

A Note of Warning

"Some people, when confronted with a problem, think 'I know, I'll use regular expressions.' Now they have two problems." - Jamie Zawinski

A Note of Warning

"Some people, when confronted with a problem, think 'I know, I'll use regular expressions.' Now they have two problems." - Jamie Zawinski

- Use regular expressions if the string methods won't serve you or make your code bulky.

A Note of Warning

"Some people, when confronted with a problem, think 'I know, I'll use regular expressions.' Now they have two problems." - Jamie Zawinski

- Use regular expressions if the string methods won't serve you or make your code bulky.
- If you have to chose between verbosity and compact code, chose verbosity if it makes your code easier to understand.

A Note of Warning

"Some people, when confronted with a problem, think 'I know, I'll use regular expressions.' Now they have two problems." - Jamie Zawinski

- Use regular expressions if the string methods won't serve you or make your code bulky.
- If you have to chose between verbosity and compact code, chose verbosity if it makes your code easier to understand.
- Regular expressions make your code ugly and difficult to maintain. Comment your code using regex profusely.

A Note of Warning

"Some people, when confronted with a problem, think 'I know, I'll use regular expressions.' Now they have two problems." - Jamie Zawinski

- Use regular expressions if the string methods won't serve you or make your code bulky.
- If you have to chose between verbosity and compact code, chose verbosity if it makes your code easier to understand.
- Regular expressions make your code ugly and difficult to maintain. Comment your code using regex profusely.

Remember, "With great power comes great responsibility".

sys

- `sys` provides access to variables maintained by interpreter and functions that interact with the interpreter

Attribute	Description
`argv`	list of command line arguments
`path`	list of directories searched for modules
`exc_info()`	information on currently handled exception
`exit([arg])`	exit from Python with the given exit value
`modules`	dict of modules already loaded
`stdin, stdout`	pre-opened interactive console I/O files
`stderr`	stderr file handle for prompt and errors

os

- The os module provides a portable way of using operating system dependent functionality.

```
1  import os
2  # environ is a dictionary of environment variables
3  print(os.environ["HOME"], os.environ["USERPROFILE"])
4  print(os.getcwd()) # get current directory
5  os.chdir("C:\\") # change to C:\
6  print("Current PID:", os.getpid(), "Parent:", os.getppid())
7  print(os.listdir(os.environ["HOME"])) # list files in directory
8  os.makedirs("C:\\testdir\\level2\\level3") # create dirs
9  os.removedirs("C:\\testdir\\level2\\level3")
10 print(os.stat("C:\\install.ini")) # print file status
11 # os.path provides all path operations portably
12 print(os.path.splitext("filename.txt")
13 print(os.path.join("C:\\", "TEMP"))
```

Example of os

- `os.walk()` is a generator function that recursively traverses a directory tree and returns a tuple of current path, list of subdirectories and a list of filenames in the current path.

Example of os

- `os.walk()` is a generator function that recursively traverses a directory tree and returns a tuple of current path, list of subdirectories and a list of filenames in the current path.

```
1  import os
2  import fnmatch
3  def gen_find(filepat,top):
4      for path, dirlist, filelist in os.walk(top):
5          for name in fnmatch.filter(filelist,filepat):
6              yield os.path.join(path,name)
7
8  # You can invoke it as
9  startpath = os.environ["USERPROFILE"] # start from user directory
10 pyfiles = gen_find("*.py", startpath) # generator of .py file collec
11 txtfiles = gen_find("*.txt", startpath) # generator of .txt file col
```

Parsing Command Line Arguments

- `argparse` module provides facilities to process command line arguments for the Python programs. For a detailed tutorial, `https://docs.python.org/3/howto/argparse.html`

Parsing Command Line Arguments

- `argparse` module provides facilities to process command line arguments for the Python programs. For a detailed tutorial, `https://docs.python.org/3/howto/argparse.html`

```
1  import argparse
2
3  parser = argparse.ArgumentParser(description="calculate X to the pow
4  group = parser.add_mutually_exclusive_group()
5  group.add_argument("-v", "--verbose", action="store_true")
6  group.add_argument("-q", "--quiet", action="store_true")
7  parser.add_argument("x", type=int, help="the base")
8  parser.add_argument("y", type=int, help="the exponent")
9  args = parser.parse_args()
10 answer = args.x**args.y
11
12 if args.quiet:
13     print(answer)
14 elif args.verbose:
15     print("{} to the power {} equals {}".format(args.x, args.y, answ
16 else:
17     print("{}^{} == {}".format(args.x, args.y, answer))
```

Calling External Programs

- The `subprocess` module provides useful functions to allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.

Calling External Programs

- The `subprocess` module provides useful functions to allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.

Function	Description
`call()`	Run a command, wait for its completion and return a value
`check_call()`	Same as above. On error, `CalledProcessError` is raised.
`check_output()`	Same as above. Output is in a separate `output` argument
`Popen()`	Creates a `Popen` object to manage programs in a child process. Has far more knobs to turn compared to other functions above

Other useful modules

- `datetime` module provides useful objects to create and manipulate date and time objects. You can do addition and subtraction of these objects using overloaded arithmetic operators.

Other useful modules

- `datetime` module provides useful objects to create and manipulate date and time objects. You can do addition and subtraction of these objects using overloaded arithmetic operators.
- `zipfile` module allows easy creation and manipulation of compressed file archives. Encryption and updation of archives are also supported.

Other useful modules

- `datetime` module provides useful objects to create and manipulate date and time objects. You can do addition and subtraction of these objects using overloaded arithmetic operators.
- `zipfile` module allows easy creation and manipulation of compressed file archives. Encryption and updation of archives are also supported.
- `bz2`, `filecmp`, `fnmatch`, `glob` and `gzip` provide file and directory handling functions.

Other useful modules

- `datetime` module provides useful objects to create and manipulate date and time objects. You can do addition and subtraction of these objects using overloaded arithmetic operators.
- `zipfile` module allows easy creation and manipulation of compressed file archives. Encryption and updation of archives are also supported.
- `bz2`, `filecmp`, `fnmatch`, `glob` and `gzip` provide file and directory handling functions.
- `sqlite3` provides relational database programming facility using **SQLITE3** embedded database. `shelf` provides a dictionary like persistent storage on disk for Python objects.

Support for Internet & Web Programming

- `socket` module provides socket programming facilities.

Support for Internet & Web Programming

- `socket` module provides socket programming facilities.
- `urllib` module provides facilities for processing URL strings and downloading URL assets.

Support for Internet & Web Programming

- `socket` module provides socket programming facilities.
- `urllib` module provides facilities for processing URL strings and downloading URL assets.
- `json` module handles encoding and decoding Python objects to JSON objects. `csv` module provides facilities to handle data in comma separated values format.

Support for Internet & Web Programming

- `socket` module provides socket programming facilities.
- `urllib` module provides facilities for processing URL strings and downloading URL assets.
- `json` module handles encoding and decoding Python objects to JSON objects. `csv` module provides facilities to handle data in comma separated values format.
- `ftplib`, `smtplib`, `xmlrpc` and `http` packages provide modules to write the servers and clients of their respective protocols.

Threading

- `threading` along with the `queue` module provides high level threading interface to python programs.

Threading

- `threading` along with the `queue` module provides high level threading interface to python programs.
- The `threading` module supports various synchronization objects in addition to the `Thread` class.

Threading

- `threading` along with the `queue` module provides high level threading interface to python programs.
- The `threading` module supports various synchronization objects in addition to the `Thread` class.
  - `Lock` and `RLock` (re-entrant lock)

Threading

- `threading` along with the `queue` module provides high level threading interface to python programs.
- The `threading` module supports various synchronization objects in addition to the `Thread` class.
  - `Lock` and `RLock` (re-entrant lock)
  - `Condition` - condition variable

Threading

- `threading` along with the `queue` module provides high level threading interface to python programs.
- The `threading` module supports various synchronization objects in addition to the `Thread` class.
  - `Lock` and `RLock` (re-entrant lock)
  - `Condition` - condition variable
  - `Semaphore` and `BoundedSemaphore` (for maximum value)

Threading

- `threading` along with the `queue` module provides high level threading interface to python programs.
- The `threading` module supports various synchronization objects in addition to the `Thread` class.
  - `Lock` and `RLock` (re-entrant lock)
  - `Condition` - condition variable
  - `Semaphore` and `BoundedSemaphore` (for maximum value)
  - `Event` objects

Threading

- `threading` along with the `queue` module provides high level threading interface to python programs.
- The `threading` module supports various synchronization objects in addition to the `Thread` class.
  - `Lock` and `RLock` (re-entrant lock)
  - `Condition` - condition variable
  - `Semaphore` and `BoundedSemaphore` (for maximum value)
  - `Event` objects
  - `Timer` objects. After timer expires, a user provided function is called.

Threading

- `threading` along with the `queue` module provides high level threading interface to python programs.
- The `threading` module supports various synchronization objects in addition to the `Thread` class.
  - `Lock` and `RLock` (re-entrant lock)
  - `Condition` - condition variable
  - `Semaphore` and `BoundedSemaphore` (for maximum value)
  - `Event` objects
  - `Timer` objects. After timer expires, a user provided function is called.
  - `Barrier` class provides a simple synchronization primitive for use by a fixed number of threads that need to wait for each other.

Threading Example 1

You can launch a function as a thread.

Threading Example 1

You can launch a function as a thread.

```
1  import threading
2  import time
3
4  def clock(interval):
5      while True:
6          print("The time is %s" % time.ctime())
7          time.sleep(interval)
8
9  t = threading.Thread(target=clock, args=(15,))
10 t.daemon = True
11 t.start() # this starts the thread
```

Threading Example 2

- Alternatively, you can derive a class from `threading.Thread` and override the `run()` method to do an action in a separate thread.

Threading Example 2

- Alternatively, you can derive a class from `threading.Thread` and override the `run()` method to do an action in a separate thread.

```
1  import threading
2  import time
3
4  class ClockThread(threading.Thread):
5      def __init__(self,interval):
6          threading.Thread.__init__(self)
7          self.daemon = True
8          self.interval = interval
9      def run(self):
10         while True:
11             print("The time is %s" % time.ctime())
12             time.sleep(self.interval)
13
14 t = ClockThread(15)
15 t.start()
```

Multiprocessing

- `multiprocessing` is a package that supports spawning processes using an API similar to the `threading` module.
- This package offers both local and remote concurrency.

Multiprocessing

- `multiprocessing` is a package that supports spawning processes using an API similar to the `threading` module.
- This package offers both local and remote concurrency.

```
1  from multiprocessing import Process, Queue
2
3  def f(q):
4      q.put([42, None, 'hello'])
5
6  if __name__ == '__main__': # mandatory in multiprocessing programs
7      q = Queue()
8      p = Process(target=f, args=(q,))
9      p.start()
10     print(q.get())    # prints "[42, None, 'hello']"
11     p.join()
```

pdb

- The module pdb defines an interactive source code debugger for Python programs.

pdb

- The module pdb defines an interactive source code debugger for Python programs.
- It supports setting (conditional) breakpoints and single stepping at the source line level, inspection of stack frames, source code listing, and evaluation of arbitrary Python code in the context of any stack frame.

pdb

- The module pdb defines an interactive source code debugger for Python programs.
- It supports setting (conditional) breakpoints and single stepping at the source line level, inspection of stack frames, source code listing, and evaluation of arbitrary Python code in the context of any stack frame.
- It also supports post-mortem debugging and can be called under program control.

pdb

- The module pdb defines an interactive source code debugger for Python programs.
- It supports setting (conditional) breakpoints and single stepping at the source line level, inspection of stack frames, source code listing, and evaluation of arbitrary Python code in the context of any stack frame.
- It also supports post-mortem debugging and can be called under program control.
- It provides facilities similar to gdb like enabling tracing, step, next, run, setup/clear breakpoints, print stack trace, etc.

pdb

- The module pdb defines an interactive source code debugger for Python programs.
- It supports setting (conditional) breakpoints and single stepping at the source line level, inspection of stack frames, source code listing, and evaluation of arbitrary Python code in the context of any stack frame.
- It also supports post-mortem debugging and can be called under program control.
- It provides facilities similar to gdb like enabling tracing, step, next, run, setup/clear breakpoints, print stack trace, etc.
- Many IDEs also provide GUI based debugging support.

Installing from source packages

- Python modules packaged using the `distutils` module contain a `setup.py` script to copy the relevant files to the Python installation directory. You can run it as,

```
1 $ python setup.py install
```

- Normally the package is installed under the directory: `<python-dir>/lib/sitepackages/<package>/`

Installing from Python repository

- The Python repository for many Python packages is located at `https://pypi.python.org`. A utility called `pip` can be used to download packages from this repository and install.

Installing from Python repository

- The Python repository for many Python packages is located at `https://pypi.python.org`. A utility called `pip` can be used to download packages from this repository and install.
- Remember to check that the environment variables `http_proxy` and `https_proxy` are set to `http://proxyname:proxyport` if you are behind a proxy.

Installing from Python repository

- The Python repository for many Python packages is located at `https://pypi.python.org`. A utility called `pip` can be used to download packages from this repository and install.
- Remember to check that the environment variables `http_proxy` and `https_proxy` are set to `http://proxyname:proxyport` if you are behind a proxy.

```
1 $ pip install SomePackage           # latest version
2 $ pip install SomePackage==1.0.4    # specific version
3 $ pip install 'SomePackage>=1.0.4' # minimum version
4 $ pip install --upgrade --no-deps SomePackage # upgrade
5 $ pip list # list installed packages
6 $ pip uninstall SomePackage # uninstall the package
```

Interfacing with C/C++ Libraries

- Functionality of Python can be extended by creating extension modules in C and C++.

Interfacing with C/C++ Libraries

- Functionality of Python can be extended by creating extension modules in C and C++.
- Many third party tools are available to simplify creation and wrapping of these extensions, viz., `Cython`, `cffi`, `SWIG` and `Numba`.

Interfacing with C/C++ Libraries

- Functionality of Python can be extended by creating extension modules in C and C++.
- Many third party tools are available to simplify creation and wrapping of these extensions, viz., `Cython`, `cffi`, `SWIG` and `Numba`.
- **Cython** (`http://cython.org`) is an optimising static compiler that allows one to write C extensions in Python-like high level language and generate optimized C code for it.

Interfacing with C/C++ Libraries - 2

Interfacing with C/C++ Libraries - 2

- `CFFI` provides Foreign Function Interface for Python calling C code. DLLs can be opened and their functions can be called from Python code. Extension modules link to `libffi` to provide this functionality.

Interfacing with C/C++ Libraries - 2

- `CFFI` provides Foreign Function Interface for Python calling C code. DLLs can be opened and their functions can be called from Python code. Extension modules link to `libffi` to provide this functionality.
- `ctypes` allows wrapping DLLs or shared libraries in pure Python. All C data types and structures are available in equivalent Python.

venv

- The venv module provides support for creating lightweight **virtual environments** with their own site directories, optionally isolated from system site directories.

venv

- The venv module provides support for creating lightweight **virtual environments** with their own site directories, optionally isolated from system site directories.
- Each virtual environment has its own Python binary (allowing creation of environments with various Python versions) and can have its own independent set of installed Python packages in its site directories.

venv

- The venv module provides support for creating lightweight **virtual environments** with their own site directories, optionally isolated from system site directories.
- Each virtual environment has its own Python binary (allowing creation of environments with various Python versions) and can have its own independent set of installed Python packages in its site directories.

```
1 c:\Temp>c:\Python34\python -m venv myenv
```

venv

- The `venv` module provides support for creating lightweight **virtual environments** with their own site directories, optionally isolated from system site directories.
- Each virtual environment has its own Python binary (allowing creation of environments with various Python versions) and can have its own independent set of installed Python packages in its site directories.

```
1 c:\Temp>c:\Python34\python -m venv myenv
```

- The above command creates a directory `myenv` under `c:/Temp`. You can run the `activate` script in the new directory tree to change to that virtual environment.
- To switch out, run the `deactivate` script.

IPython

- **ipython** (`http://ipython.org/`) provides much richer interactive environment than the default python console.

IPython

- **ipython** (`http://ipython.org/`) provides much richer interactive environment than the default python console.
- It provides interactive data visualization with rich GUI.

IPython

- **ipython** (`http://ipython.org/`) provides much richer interactive environment than the default python console.
- It provides interactive data visualization with rich GUI.
- It also provides a feature-rich notebook which can be used to write reproducible programs along with inline documentation. See `http://nbviewer.ipython.org/` for examples.

Alternative Python Distributions

- There are alternative Python distributions that come with many pre-packaged modules for specific domains.

Alternative Python Distributions

- There are alternative Python distributions that come with many pre-packaged modules for specific domains.
- E.g., Anaconda is a Python distribution for large-scale data processing, predictive analytics, and scientific computing. See `https://store.continuum.io/cshop/anaconda/`

Windows Binaries

- Unofficial Windows(R) binaries for many popular Python modules can be obtained at `http://www.lfd.uci.edu/~gohlke/pythonlibs/`.