3 - Advanced topics in Python

December 4, 2015

1 JSON / XML Parsing

1.1 JSON Parsing

```
In [ ]: # importing the Python json library
        import json
```

1.1.1 Importing JSON to Dict

From string

```
In [ ]: # Example string
        json_string = '{"first_name": "Guido", "last_name":"Rossum"}'
        json_string
```

```
In [ ]: parsed_json = json.loads(json_string)
        parsed_json
```

From file

```
In [ ]: # %load files/example.json
```

```
In [ ]: with open('files/example.json', 'r') as f:
            content = f.read()

        parsed_json = json.loads(content)
        parsed_json
```

1.1.2 Exporting Dict { } to JSON

```
In [ ]: # Example dict
        d = { 'first_name': 'Guido',
              'second_name': 'Rossum',
              'titles': ['BDFL', 'Developer'],
            }
        d
```

```
In [ ]: json_string = json.dumps(d)
        json_string
```

1.2 XML Parsing

```
In [ ]: import xml.etree.ElementTree as ET
```

1.2.1 Importing XML to ElementTree

From string

```
In [ ]: # Example string
        xml_string = """<catalog>
           <book id="bk101">
              <author>Gambardella, Matthew</author>
              <title>XML Developer's Guide</title>
              <genre>Computer</genre>
           </book>
        </catalog>"""
        xml_string
```

```
In [ ]: root = ET.fromstring(xml_string)
        print root.tag
        print root.attrib
```

From file

```
In [ ]: # %load files/example.xml
```

```
In [ ]: tree = ET.parse('files/example.xml')
        root = tree.getroot()
        print root.tag
        print root.attrib
```

1.2.2 Exporting ElementTree to XML

```
In [ ]: tree.write("files/example_output.xml")
```

```
In [ ]: # %load files/example_output.xml
```

1.2.3 Getting items

```
In [ ]: # Each XML element contains a tag, an attribute (optional), a text and a list of childs
        # XML Element: <tag attr:attr_value> text </tag>

        print root.tag            # tag    : name of XML element
        print root.attrib         # attrib : attribute of XML element
        print root.text           # text   : content of XML element
        for child in root: # loop through all subchildrens
            print "\t", child.tag, child.attrib, child.text
            for subchild in child:
                print "\t\t", subchild.tag, subchild.attrib, subchild.text
            print

        children = list(root)     #Get children

        print children
        print root
```

```
In [ ]: # iter('element_tag') - search all subtrees
        for title in root.iter('title'):
            print title.text
```

```
In [ ]: # findall('element_tag') - get direct childs of parent
        for book in root.findall('book'):
            # Get items
            book_id = book.get('id')          # Get attribute 'id' of 'book' element
            title = book.find('title')        # Get first child named 'title'
            author = book.find('author')
            genre = book.find('genre')
            price = book.find('price')
            pdate = book.find('publish_date')
            descr = book.find('description')

            # Print items
            print "ID ", book_id
            print title.tag, title.text
            print author.tag, author.text
            print genre.tag, genre.text
            print price.tag, price.text
            print pdate.tag, pdate.text
            print descr.tag, descr.text
            print

In [ ]: # find('element_tag') - finds first child with tag
        for book_content in root.find('book'):
            print book_content
```

1.2.4 Modifying items

```
In [ ]: # set('attribute', 'attribute_value')
        for price in root.iter('price'):
            price.text = str(float(price.text) + 1)
            price.set('increased', 'yes')
            print price.text, price.attrib

In [ ]: # Create new element
        new_book = ET.Element('book', {'id': 'bk113'})
        new_book.text = "\n"

        # Create sub elements
        author = ET.SubElement(new_book, 'author')
        title = ET.SubElement(new_book, 'title')
        genre = ET.SubElement(new_book, 'genre')
        price = ET.SubElement(new_book, 'price')
        pdate = ET.SubElement(new_book, 'publish_date')
        descr = ET.SubElement(new_book, 'description')

        # Populate sub elements
        author.text = "J.K Rowlings"
        title.text = "Harry Potter and the Sorcerer's Stone"
        genre.text = "Fantasy"
        price.text = "31.50"
        pdate.text = "2001-10-16"
        descr.text = "A very nice fantasy book."

        # Add element to existing tree
        root.append(new_book)
```

```
        # ET.dump(new_book)
```

In [ ]:
```
# A ten times nicer way of doing it (focus on what matters)

def create_new_ET(element_name, attributes={}, elements={}):
    book = ET.Element('book', attributes)
    for key in elements:
        new_elem = ET.SubElement(book, key)
        new_elem.text = elements[key]
    return book

# Create new element
name     = 'book'
attr     = {'id': "bk114"}
elements = {'author': "J.K Rowlings",
            'title': "Harry Potter and the Chamber of Secrets",
            'genre': "Fantasy",
            'price': "35.50",
            'publish_date': "2002-10-15",
            'description': "The second volume of a very nice fantasy book."}
new_book = create_new_ET(name, attr, elements)

# Append to existing tree
root.append(new_book)

# ET.dump(root)
```

1.3 Bonus: Pretty Print and Conversion

1.3.1 Pretty Print

In [ ]:
```
def indent(elem, level=0):
    i = "\n" + level*"  "
    if len(elem):
        if not elem.text or not elem.text.strip():
            elem.text = i + "  "
        if not elem.tail or not elem.tail.strip():
            elem.tail = i
        for elem in elem:
            indent(elem, level+1)
        if not elem.tail or not elem.tail.strip():
            elem.tail = i
    else:
        if level and (not elem.tail or not elem.tail.strip()):
            elem.tail = i

indent(root)
tree.write("files/example_output.xml")
ET.dump(tree)
```

In [ ]:
```
# %load files/example_output.xml
```

1.3.2 Etree to Dict

```
In [ ]: from collections import defaultdict

        def etree_to_dict(t):
            d = {t.tag: {} if t.attrib else None}
            children = list(t)
            if children:
                dd = defaultdict(list)
                for dc in map(etree_to_dict, children):
                    for k, v in dc.iteritems():
                        dd[k].append(v)
                d = {t.tag: {k:v[0] if len(v) == 1 else v for k, v in dd.iteritems()}}
            if t.attrib:
                d[t.tag].update(('@' + k, v) for k, v in t.attrib.iteritems())
            if t.text:
                text = t.text.strip()
                if children or t.attrib:
                    if text:
                      d[t.tag]['#text'] = text
                else:
                    d[t.tag] = text
            return d

In [ ]: d = etree_to_dict(root)
        d
```

1.3.3 Dict { } to Etree

```
In [ ]: def dict_to_etree(d):
            def _to_etree(d, root):
                if not d:
                    pass
                elif isinstance(d, basestring):
                    root.text = d
                elif isinstance(d, dict):
                    for k,v in d.items():
                        assert isinstance(k, basestring)
                        if k.startswith('#'):
                            assert k == '#text' and isinstance(v, basestring)
                            root.text = v
                        elif k.startswith('@'):
                            assert isinstance(v, basestring)
                            root.set(k[1:], v)
                        elif isinstance(v, list):
                            for e in v:
                                _to_etree(e, ET.SubElement(root, k))
                        else:
                            _to_etree(v, ET.SubElement(root, k))
                else: assert d == 'invalid type', (type(d), d)
            assert isinstance(d, dict) and len(d) == 1
            tag, body = next(iter(d.items()))
            node = ET.Element(tag)
            _to_etree(body, node)
            return node
```

```
In [ ]: t = dict_to_etree(d)
        indent(t)
        ET.dump(t)
```

1.4 Exercise: Analyze an XML file

1.4.1 Problem

Objectives: - Extract Maven plugins information from `base-corporate-pom`. - Add a new Maven plugin to `base-corporate-pom`. - Write Maven plugins information to a new file "'files/pom_maven_plugins.xml""

Information:

- `base-corporate-pom` pom.xml is located in `files/pom.xml`
- XML plugin structure:

```
<project>
<!-- Plugin version -->
<properties>
    <maven-surefire-plugin.version>2.12.4</maven-surefire-plugin.version>
</properties>
<!-- Plugin info -->
<build>
    <pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugin</groupId>
                <artifactId>maven-surefire-plugin</version>
                <version>$(maven-surefire-plugin.version)</version>
             </plugin>
        </plugins>
     </pluginManagement>
</build>
</project>
```

- Some plugins don't have a <`version`> or a <`groupId`> tag. Print "N/A" when there is no tag.
- All plugins have an <`artifactId`> tag.

Desired output:
maven-surefire

```
groupId: org.apache.maven.plugins
artifactId: maven-surefire-plugin
version: 2.12.4
```

1.4.2 Solution

```
In [4]: import xml.etree.ElementTree as ET

        root = ET.parse('files/pom.xml').getroot()

        # Variables
        plugin_list = []
        base = "{http://maven.apache.org/POM/4.0.0}"
```

```
        # Find 'build' and 'properties' XML tags in root
        build = root.find(base + 'build')
        properties = root.find(base + 'properties')

        # Find 'pluginManagement' XML tag in 'build'
        pluginManagement = build.find(base + 'pluginManagement')

        # Find 'plugins' XML tag in 'pluginManagement'
        plugins = pluginManagement.find(base + 'plugins')

        # Populate plugin_list
        plugin_list = list(plugins)

        # Loop through plugin_list elements (of class ElementTree)
        for plugin in plugin_list:

                # Get  plugin information
                artifactId_str = plugin.find(base + 'artifactId').text
                name = artifactId_str.replace('-plugin', '')

                # Not all plugins have a groupId !
                try:
                    groupId_str = plugin.find(base + 'groupId').text
                except:
                    groupId_str = "N/A"

                # Not all plugins have a version !
                try:
                    version_str = plugin.find(base + 'version').text
                    # Get plugin from <properties> element
                    version_str = version[2:-1] # strip '£', '{', and '}' from version
                    version_str = properties.find(base + version).text
                except:
                    version_str = "N/A"

                # Print plugin information
                print name
                print "\t" + "groupId: " + groupId_str
                print "\t" + "artifactId: " + artifactId_str
                print "\t" + "version: " + version_str
                print
```

```
maven-surefire
        groupId: org.apache.maven.plugins
        artifactId: maven-surefire-plugin
        version: N/A

properties-maven
        groupId: org.codehaus.mojo
        artifactId: properties-maven-plugin
        version: N/A

maven-release
        groupId: org.apache.maven.plugins
```

```
        artifactId: maven-release-plugin
        version: N/A

maven-install
        groupId: org.apache.maven.plugins
        artifactId: maven-install-plugin
        version: N/A

maven-deploy
        groupId: org.apache.maven.plugins
        artifactId: maven-deploy-plugin
        version: N/A

versions-maven
        groupId: org.codehaus.mojo
        artifactId: versions-maven-plugin
        version: N/A

ccc-versions-maven
        groupId: com.cccis.build.maven
        artifactId: ccc-versions-maven-plugin
        version: N/A

oc4j-admin-maven
        groupId: com.cccis.build.maven
        artifactId: oc4j-admin-maven-plugin
        version: N/A

weblogic-maven
        groupId: com.oracle.weblogic
        artifactId: weblogic-maven-plugin
        version: N/A

wls-maven
        groupId: com.oracle.weblogic
        artifactId: wls-maven-plugin
        version: N/A

gmaven
        groupId: org.codehaus.gmaven
        artifactId: gmaven-plugin
        version: N/A

maven-enforcer
        groupId: org.apache.maven.plugins
        artifactId: maven-enforcer-plugin
        version: N/A

maven-eclipse
        groupId: org.apache.maven.plugins
        artifactId: maven-eclipse-plugin
        version: N/A

lifecycle-mapping
```

```
        groupId: org.eclipse.m2e
        artifactId: lifecycle-mapping
        version: N/A

maven-site
        groupId: org.apache.maven.plugins
        artifactId: maven-site-plugin
        version: N/A

maven-assembly
        groupId: N/A
        artifactId: maven-assembly-plugin
        version: N/A

deployment-maven
        groupId: com.cccis.build.maven
        artifactId: deployment-maven-plugin
        version: N/A

site-publish-helper
        groupId: com.cccis.build.maven
        artifactId: site-publish-helper
        version: N/A

maven-antrun
        groupId: org.apache.maven.plugins
        artifactId: maven-antrun-plugin
        version: N/A
```

1.4.3 Solution 2 (improved)

```
In [9]: import xml.etree.ElementTree as ET
        import re

        def concat(list_tags):
            # Small function to concatenate the base with the tag name for a list of names.
            # Return a tuple containing the new names.
            t = tuple()
            for tag in list_tags:
                t += (base + tag,)
            return t

        def build_path(list_, base = ''):
            # Create a path from a list of subpaths.
            # If a base is defined, append the based before each path element.
            path = "."
            for a in list_:
                if base:
                    path += "/" + base + a
                    continue
                path += "/" + a
            return path

        def get_plugin_info(plugin):
```

```
    # Get concatenated names
    names = ['artifactId', 'groupId', 'version']
    aId, gId, v = concat(names)

    # Get elements from plugin ET
    artifactId = plugin.find(aId)
    groupId = plugin.find(gId)
    version = plugin.find(v)

    # Set output values

    try:
        aId_str = artifactId.text
    except Exception as e:
        raise(e)

    try:
        gId_str = groupId.text
    except:
        gId_str = "N/A"

    try:
        version_str = version.text # strip '£', '{', and '}' from version.text
        try:
            float(version_str)
            version_str = str(version_str)
        except ValueError:
            version_str = version_str[2:-1]
            version_str = properties.find(base + version_str).text
    except:
        version_str = "N/A"

    # Return plugin information
    name_str = aId_str.replace('-plugin','')
    return (aId_str, gId_str, version_str, name_str)

def format_plugin_info(plugin):
    artifactId, groupId, version, name = get_plugin_info(plugin)
    to_print = name + "\n" + \
    "\t" + "groupId: " + groupId + "\n" + \
    "\t" + "artifactId: " + artifactId + "\n" + \
    "\t" + "version: " + version
    return to_print

def print_plugin(plugin):
    print format_plugin_info(plugin)

def write_plugin_to_file(plugin, filepath):
    to_print = format_plugin_info(plugin)
    with open(filepath, 'a') as f:
        f.write(to_print)

# main function
if __name__ == '__main__':
```

```
            root = ET.parse('files/pom.xml').getroot()
            base = "{http://maven.apache.org/POM/4.0.0}"

            # Generate plugins path and properties path
            plugins_path    = build_path(['build', 'pluginManagement', 'plugins'], base=base)
            properties_path = build_path(['properties'], base=base)

            # Get plugins ET and properties ET
            plugins = root.find(plugins_path)

            # Get plugins info and print plugins
            for p in plugins.findall(base + 'plugin'):
                print_plugin(p)
                write_plugin_to_file(p, 'files/pom_maven_plugins.xml')
```

```
maven-surefire
        groupId: org.apache.maven.plugins
        artifactId: maven-surefire-plugin
        version: 2.12.4
properties-maven
        groupId: org.codehaus.mojo
        artifactId: properties-maven-plugin
        version: 1.0-alpha-2
maven-release
        groupId: org.apache.maven.plugins
        artifactId: maven-release-plugin
        version: 2.2.1
maven-install
        groupId: org.apache.maven.plugins
        artifactId: maven-install-plugin
        version: 2.4
maven-deploy
        groupId: org.apache.maven.plugins
        artifactId: maven-deploy-plugin
        version: 2.7
versions-maven
        groupId: org.codehaus.mojo
        artifactId: versions-maven-plugin
        version: 2.1
ccc-versions-maven
        groupId: com.cccis.build.maven
        artifactId: ccc-versions-maven-plugin
        version: 0.0.38
oc4j-admin-maven
        groupId: com.cccis.build.maven
        artifactId: oc4j-admin-maven-plugin
        version: 0.0.37
weblogic-maven
        groupId: com.oracle.weblogic
        artifactId: weblogic-maven-plugin
        version: 10.3.4
wls-maven
        groupId: com.oracle.weblogic
        artifactId: wls-maven-plugin
```

```
        version: 12.1.1.0
gmaven
        groupId: org.codehaus.gmaven
        artifactId: gmaven-plugin
        version: 1.3
maven-enforcer
        groupId: org.apache.maven.plugins
        artifactId: maven-enforcer-plugin
        version: 1.2
maven-eclipse
        groupId: org.apache.maven.plugins
        artifactId: maven-eclipse-plugin
        version: N/A
lifecycle-mapping
        groupId: org.eclipse.m2e
        artifactId: lifecycle-mapping
        version: 1.0.0
maven-site
        groupId: org.apache.maven.plugins
        artifactId: maven-site-plugin
        version: 3.3
maven-assembly
        groupId: N/A
        artifactId: maven-assembly-plugin
        version: 2.3
deployment-maven
        groupId: com.cccis.build.maven
        artifactId: deployment-maven-plugin
        version: 0.0.41
site-publish-helper
        groupId: com.cccis.build.maven
        artifactId: site-publish-helper
        version: 0.0.37
maven-antrun
        groupId: org.apache.maven.plugins
        artifactId: maven-antrun-plugin
        version: 1.7
```

1.4.4 Solution 3 (OOP approach)

In [2]:
```
import xml.etree.ElementTree as ET
import re

class Plugin(object):
    """ A plugin has a name, an artifactId, a groupId and a version."""
    def __init__(self, plugin, base):
        self.base = base
        self.__get_plugin_info(plugin)

    def __str__(self):
        to_print = self.name + "\n" + \
        "\t" + "groupId: " + self.groupId + "\n" + \
        "\t" + "artifactId: " + self.artifactId + "\n" + \
        "\t" + "version: " + self.version + "\n"
```

```
            return to_print

        # PRIVATE
        def __concat(self, list_tags):
            """Small function to concatenate the base with the tag name for a list of names.
            Return a tuple containing the new names."""
            t = tuple()
            for tag in list_tags:
                t += (self.base + tag,)
            return t

        def __get_plugin_info(self, plugin):
            """ Takes a plugin of class ElementTree and populates this Plugin object.
            Fields are: name, artifactId, groupid, version"""
            # Get concatenated names
            names = ['artifactId', 'groupId', 'version']
            aId, gId, v = self.__concat(names)

            # Get elements from plugin ET
            artifactId = plugin.find(aId)
            groupId = plugin.find(gId)
            version = plugin.find(v)

            # Set output values
            try:
                aId_str = artifactId.text
            except Exception as e:
                raise(e)

            try:
                gId_str = groupId.text
            except:
                gId_str = "N/A"

            try:
                version_str = version.text # strip '£', '{', and '}' from version.text
                try:
                    float(version_str)
                    version_str = str(version_str)
                except ValueError:
                    version_str = version_str[2:-1]
                    version_str = properties.find(base + version_str).text
            except:
                version_str = "N/A"

            name_str = aId_str.replace('-plugin','')

            # Populate Plugin object
            self.artifactId = aId_str
            self.groupId = gId_str
            self.version = version_str
            self.name = name_str

    class POMPluginExtractor(object):
```

```
        """This class serves as extractor of plugins from any POM file."""
        def __init__(self, filepath, base):
            self.filepath = filepath
            self.base = base
            self.plugins = []
            self.__get_root()
            self.__get_plugins()

        def print_plugins(self):
            for p in self.plugins:
                print p

        def save_plugins(self, filepath):
            with open(filepath, 'w') as f:
                for p in self.plugins:
                    print >>f, "test"

            print "Plugins saved to %s" % filepath

        # PRIVATE
        def __get_root(self):
            self.root = ET.parse(self.filepath).getroot()

        def __get_plugins(self):
            plugins_path = self.__build_path(['build', 'pluginManagement', 'plugins'], base=self.bas
            plugins = self.root.find(plugins_path)
            for p in plugins.findall(self.base + 'plugin'):
                self.plugins.append(Plugin(p, self.base))

        def __build_path(self, list_, base = ''):
            path = "."
            for a in list_:
                if base:
                    path += "/" + base + a
                    continue
                path += "/" + a
            return path

    # main function
    if __name__ == '__main__':
        inputFile = 'files/pom.xml'
        outputFile = 'files/pom_maven_plugins.xml'
        base = "{http://maven.apache.org/POM/4.0.0}"

        analyzer = POMPluginExtractor(inputFile, base)
        analyzer.save_plugins(outputFile)
```

```
Plugins saved to files/pom_maven_plugins.xml
```

2 REST API (Django)

2.1 Serialization

In [ ]:

In [ ]:

In [ ]:

In [ ]:

2.2 Requests and Responses

In [ ]:

In [ ]:

In [ ]:

In [ ]:

2.3 Class Based view

In [ ]:

In [ ]:

In [ ]:

In [ ]:

2.4 Authentication and Permission

In [ ]:

In [ ]:

In [ ]:

In [ ]:

2.5 Relationships and Hyperlink APIs

In [ ]:

In [ ]:

In [ ]:

In [ ]:

3 SQL Data Access (MySQL)

In [ ]: `import mysql.connector`

3.0.1 Connecting to MySQL DB

In [ ]:

3.0.2 Creating a table

In [ ]:

3.0.3 Populating a table

In [ ]:

3.0.4 Querying data from a table

In [ ]:

In [ ]:

4 NoSQL Data Access (DynamoDB)

In [ ]: import boto3

4.0.1 Connecting to DynamoDB

In [ ]:

4.0.2 Populating a table

In [ ]:

4.0.3 Querying data from a table

In [ ]:

In [ ]:

5 Exercise: Parse a switch record and output to MySQL

In [ ]:

In [ ]:

In [ ]: