Simple Python webservice client

Code examples are valid for both Python versions 2 and 3.

Data delivery Web service (API for getting time series data)

```
import requests

if __name__ == '__main__':
    request_xml = '''<ws:dataDeliveryRequest dateFrom="2014-04-28" dateTo="
2014-04-28"
    xmlns="http://geomodel.eu/schema/data/request"
    xmlns:ws="http://geomodel.eu/schema/ws/data"
    xmlns:geo="http://geomodel.eu/schema/common/geo"
    xmlns:pv="http://geomodel.eu/schema/common/pv"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <site id="demo_site" name="Demo site" lat="48.61259" lng="20.827079">
    </site>
    <processing key="GHI" summarization="HOURLY" terrainShading="true">
    </processing>
    </ws:dataDeliveryRequest>'''
    api_key = 'demo'
    url = 'https://solargis.info/ws/rest/datadelivery/request?key=%s' %
api_key
    headers = {'Content-Type': 'application/xml'}
    with requests.post(url, data=request_xml.encode('utf8'),
headers=headers) as response:
        print(response.text)
        # parse and consume successful response, or inspect error code and
a message from the server
```

In real production environment you will automatically modify XML request in run-time (e.g. changing location, period etc.). You can do this by using of XML request templates when only particular data will by replaced (e.g. lat, lng, name, dateFrom). In such case the python native The ElementTree XML API can be helpful for XML manipulation. Creating new XML requests from scratch can be easier by using some "XML data binding" technology. First you generate python objects from Solargis XSD schema documents. Then you can use the python objects for marshaling (serializing python objects into XML text) and unmarshalling (deserializing XML text into python objects) either for the request or response. The PyXB package can be used (http://pyxb.sourceforge.net/).

PvPlanner web service (API for getting long-term average data)

```
import requests

if __name__ == '__main__':
    request_xml = '''<calculateRequest xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance"
          xmlns:geo="http://geomodel.eu/schema/common/geo"
          xmlns:pv="http://geomodel.eu/schema/common/pv"
          xmlns="http://geomodel.eu/schema/ws/pvplanner">
        <site lat="48.612590" lng="20.827079">
            <!-- optional terrain data -->
            <geo:terrain elevation="246" azimuth="176" tilt="3.1" />
            <!-- optional custom horizon data can replace the natural
terrain horizon -->
            <!--<geo:horizon>11.11:18.0 7.5:15.53 15.0:10.94 22.5:10.59
30.0:13.06 37.5:14.47 45.0:14.47 52.5:13.76 60.0:12.35 67.5:11.29 75.0:
8.12 82.5:4.59 90.0:1.41 97.5:0.35 105.0:0.35 112.5:0.35 120.0:0.35 127.5:
0.35 135.0:0.0 142.5:0.0 150.0:0.35 157.5:1.41 165.0:2.47 172.5:2.47 180.0:
2.82 187.5:3.18 195.0:2.82 202.5:2.47 210.0:2.47 217.5:2.47 225.0:3.18
232.5:3.18 240.0:2.47 247.5:2.12 255.0:2.12 262.5:2.82 270.0:3.88 277.5:
6.71 285.0:8.47 292.5:10.24 300.0:11.29 307.5:12.71 315.0:14.12 322.5:
15.53 330.0:16.24 337.5:16.94 345.0:17.29 352.5:17.29</geo:horizon>-->
            <pv:geometry xsi:type="pv:GeometryFixedOneAngle" azimuth="175"
tilt="45"/>
            <pv:system installedPower="1" installationType="ROOF_MOUNTED"
availability="99">
                <pv:module type="CSI">
                </pv:module>
                <pv:inverter>
                    <pv:efficiency xsi:type="pv:EfficiencyConstant"
percent="97.5"/>
                </pv:inverter>
                <pv:losses dc="5.5" ac="1.5"/>
            </pv:system>
        </site>
    </calculateRequest>'''
    api_key = 'demo'
    url = 'https://solargis.info/ws/rest/pvplanner/calculate?key=%s' %
api_key
    headers = {'Content-Type': 'application/xml'}
    with requests.post(url, data=request_xml.encode('utf8'),
headers=headers) as response:
        print(response.text)
        # parse and consume successful response, or inspect error code and
a message from the server
```