
pyladies

Lynn Root

@roguelynn

roguelynn.com

lynn@lynnroot.com

Lynn Root

Software Engineer at Red Hat

PyLadies of San Francisco

Python Software Foundation Board Member

Today's Plan

Part 1:

Intro to PyLadies

Part 2:

Intro to Python with Data Visualization

Part 3:

PyLadies Cocktail!

International mentorship group

Women + friends

Python + Open Source community

Supported by sponsors, donors, and the PSF

pyladies

Developers or Aspiring

Full time or hobby

Just in ♥ with Python

pyladies

Workshops

Development sprints

Speaker series

Hack nights

pyladies

- Atlanta
- Seattle
- Portland
- San Diego
- San Francisco
- NYC
- Wash, DC
- Los Angeles
- Austin
- Boston
- Nashville

pyladies

- Berlin
- Brno
- Tunisia
- Toronto
- India
- Taiwan
- Montreal
- Stockholm
- Vienna

What does it take to lead a PyLadies group?

My story!

pyladies

Ulterior motive:

find an interested PyLady to start a local PyLadies chapter

Questions? Interested folks?

pyladies.com

#pyladies on Freenode

github.com/pyladies

Intro to Python

with Data Visualization

Adapted from newcoder.io/dataviz

Follow along

rogue.ly/dataviz-slides

rogue.ly/dataviz-pdf

Workshop Plan

Introduction

Setup your Machine

Part 1: Parsing Data

Part 2: Plotting on Google Maps

Introduction

Project

Goals

Python Libraries

Project

Parse data from a CSV file

Plot on Google Maps

Goals

Run & import a Python file

Python's data structures

Make a simple maps

Python Libraries

csv

xml + Google Mapping

Setup your Machine

Python
git
pip + virtualenv
virtualenvwrapper (Mac/Linux)
Text Editor

Part 0: Setup

Setup for Data Visualization

Setup the Project

1. Make project directory

2. Clone my repository

3. Install dependencies

Questions?

Part 1: parse.py

Parsing our sample data

Part 1: parse.py

1. Module Setup
2. Attacking the Parse Function
3. Using the Parse Function
4. Putting it into Action
5. Explore it further

Part 1.1: Module Setup

1. import
2. contants

Part 1.2: Attacking

1. Scaffolding
2. Docstrings
3. Comments
4. Code

Part 1.2.1: Scaffolding

```
def parse(raw_file, delimiter):


    return parsed_data
```

Part 1.2.2: Doc Strings

Documentation strings, or "docstrings", are denoted with triple quotes:

```
"""This function returns x."""
```

Part 1.2.2: Doc Strings

```
def parse(raw_file, delimiter):
    """Parses a raw CSV file to a
       JSON-like object."""

  return parsed_data
```

Part 1.2.3: Comments

```
def parse(raw_file, delimiter):
    ...

    # Open CSV file

    # Read CSV file

    # Close CSV file

    # Build a data structure to return parsed_data

    return parsed_data
```

Part 1.2.4: Code

```
def parse(raw_file, delimiter):
    ...

    # Open CSV file
    opened_file = open(raw_file)

    ...

    return parsed_data
```

Part 1.2.4: Code

```
def parse(raw_file, delimiter):
    ...

    # Read CSV file
    csv_data = csv.reader(opened_file,
                          delimiter=delimiter)

    ...

    return parsed_data
```

Part 1.2.4: Code

```
def parse(raw_file, delimiter):
    ...

    # Setup an empty list
    parsed_data = []

    ...

    return parsed_data
```

Part 1.2.4: Code

```
def parse(raw_file, delimiter):
    ...

    # Skip over first line for headers
    fields = csv_data.next()

    ...

    return parsed_data
```

Part 1.2.4: Code

```
def parse(raw_file, delimiter):
    ...

    # Iterate over each row, zip field -> value
    for row in csv_data:
        parsed_data.append(dict(zip(fields, row)))
    ...

    return parsed_data
```

Part 1.2.4: Code

```
def parse(raw_file, delimiter):
    ...

    # Close the CSV file
    opened_file.close()

    return parsed_data
```

Part 1.3: Using parse()

```
def main():
    # Call parse() and give parameters
    new_data = parse(MY_FILE, ",")

    # Let's see what the data looks like!
    print new_data
```

Part 1.3: Using parse()

```
def main():
  # Call parse() and give parameters
  new_data = parse(MY_FILE, ",")

  # Let's see what the data looks like!
  print new_data

if __name__ == "__main__":
  main()
```

Part 1.4: Action

Follow along with me in my terminal

Questions so far?

Part 2: map.py

Plotting our sample data on Google Maps

Part 2: map.py

1. Module Setup

2. Helper Functions

3. Create G-Map

Part 3.1: Module Setup

MOAR import statements

Part 2.2.1: Create Doc

```
def create_document(title, description=""):
  """Create Overall KML Document"""



  return doc
```

Part 2.2.1: Create Doc

```
def create_document(title, description=""):
    ...

    # Initialize XML doc
    doc = xml.dom.minidom.Document()
    ...
```

Part 2.2.1: Create Doc

```
def create_document(title, description=""):
    ...

    # Define as a KML-type XML doc
    kml = doc.createElement("kml")
    ...
```

Part 2.2.1: Create Doc

```
def create_document(title, description=""):
   ...

   # Pull in common attributes
   kml.setAttrebutes("xmlns",
                     "http://www.opengist.net/kml/2.2")

   doc.appendChild(kml)
   ...
```

Part 2.2.1: Create Doc

```
def create_document(title, description=""):
    ...

    # Pull in common attributes
    document = doc.createElement("Document")
    kml.appendChild(document)
    docName = doc.createElement("title")
    document.appendChild(docName)
    ...
```

Part 2.2.1: Create Doc

```
def create_document(title, description=""):
    ...

    # Pull in common attributes (con't)
    docName_text = doc.createTextNode(title)
    docName.appendChild(docName_text)
    docDesc = doc.createElement("description")
    document.appendChild(docDesc)
    docDesc_text = doc.createTextNode(description)
    docDesc.appendChild(docDesc_text)

    ...
```

Part 2.2.1: Create Doc

```
def create_document(title, description=""):
    ...

    return doc
```

Part 2.2.2: Create Place

```
def create_placemark(address):
  """Generate KML Placemark for given addr"""

  return doc
```

Part 2.2.2: Create Place

```
def create_placemark(address):
    ...

    # Initialize XML doc
    doc = xml.dom.minidom.Document()
    ...
```

Part 2.2.2: Create Place

```
def create_placemark(address):
   ...
   # Create elements for Placemark and add to doc
   pm = doc.createElement("Placemark")
   doc.appendChild(pm)
   name = doc.createElement("name")
   pm.appendChild(name)
   ...
```

Part 2.2.2: Create Place

```
def create_placemark(address):
   ...
   # con't
   name_text = doc.createTextNode("%(name)s",
                                  % address)
   name.appendChild(name_text)
   desc = doc.createElement("description")
   pm.appendChild(desc)
   ...
```

Part 2.2.2: Create Place

```
def create_placemark(address):
   ...

   # con't
   desc_text = doc.CreateTextNode("Date:
               %(date)s, %(description)s",
               address)
   desc.appendChild(desc_text)
   pt = doc.createElement("Point")
   pm.appendChild(pt)
   ...
```

Part 2.2.2: Create Place

```
def create_placemark(address):
   ...

   # con't
   coords = doc.createElement("coordinates")
   pt.appendChild(coords)
   coords_text = doc.createTextNode(
                 "%(longitude)s, %(latitude)s"
                  % address)
   coords.appendChild(coords_text)
   ...
```

Part 2.2.2: Create Place

```
def create_placemark(address):
   ...

   return doc
```

Part 2.3 Create G-Map!

```
def create_gmap(data_file):
 """Create G-Map-readable doc"""
```

Part 2.3 Create G-Map!

```
def create_gmap(data_file):
  ...

  # Create new KML doc
  kml_doc = create_document("Crime map",
                            "Plots of Recent SF Crime")

  ...
```

Part 2.3 Create G-Map!

```
def create_gmap(data_file):
    ...

    # grab specific DOM element (all one line)
    document = kml_doc.documentElement.
               getElementByTagName("Document")[0]
    ...
```

Part 2.3 Create G-Map!

```
def create_gmap(data_file):
    ...

    # iterate over data to create KML doc
    for line in data_file:

    ...
```

Part 2.3 Create G-Map!

```
def create_gmap(data_file):
    ...
    # loop continued
    for line in data_file:
        # Parse the data into a dict
        placemark_info = {
                        "longitude": line["X"],
                        "latitude": line["Y"],
                        "name": line["Category"],
                        "description": line["Descript"],
                        "date": line["Date"]
    ...
```

Part 2.3 Create G-Map!

```
def create_gmap(data_file):
    ...
    # loop continued
    for line in data_file:
        ...
        # Avoid null values for lat/long
        if placemark_info["longitude"] == "0":
            continue
    ...
```

Part 2.3 Create G-Map!

```
def create_gmap(data_file):
    ...
    # loop continued
    for line in data_file:
        ...
        # parse line of data into KML-format
        placemark = create_placemark(placemark_info)
    ...
```

Part 2.3 Create G-Map!

```
def create_gmap(data_file):
   ...
   # loop continued
   for line in data_file:
      ...
      # Adds the placemark to KML doc
      document.appendChild(
                           placemark.documentElement)
   ...
```

Part 2.3 Create G-Map!

```
def create_gmap(data_file):
  ...
  # write parsed KML data to file
  with open("file_sf.kml", "w") as f:
    f.write(kml_doc.toprettyxml(
                  intent="  ", encoding="UTF-8"))
```

Part 2.3 Create G-Map!

```
def main():
  data = p.parse(p.my_file, ",")

  return create_gmap(data)


if __name__ == "__main__":
  main()
```

Part 2.3 Create G-Map!

Follow me on uploading to Google Maps

Congrats!

@roguelynn | roguelynn.com

lynn@lynnroot.com

newcoder.io