python, cartographie, carte, code, em

Rédaction démarrée le 31 mai 2021

Cartographie en python

Quelques notes pour réaliser des cartes statiques en téléchargeant des «tuiles» d'OpenStreetMap sur lesquelles sont ajoutées des données saisies avec umap.
Avec pour but de les imprimer.

nb : soyez indulgent, je débute en python

Deux pistes

- Static Map : https://github.com/komoot/staticmap
- py-staticmaps : https://github.com/flopp/py-staticmaps

Tests

Static Map

Installation de Static Map

```
pip3 install staticmap
pip3 show staticmap # permet de savoir quelle version installée
```

Premier script

```
# test avec https://github.com/komoot/staticmap
# Python 3.5.3 / pip 9.0.1 / staticmap 0.5.5
# Debian 9.5 @ kirin / 20210531

from staticmap import StaticMap, Line

m = StaticMap(3000, 4000, 10)
m.add_line(Line(((13.4, 52.5), (2.3, 48.9)), 'blue', 3))
image = m.render()
image.save('map.png')
```

Et ça produit bien une carte en haute définition, c'est encourageant!

On peut choisir le fournisseur de «tuiles» à la création de la carte (ex. en noir et blanc)

```
m = StaticMap(3000, 4000, 10, url_template='http://a.tile.stamen.com/toner/{z}/{x}/{y}.png')
```

J'en ai trouvé une liste ici : https://wiki.openstreetmap.org/wiki/Tile_servers

py-staticmaps

Je tente l'installation avec

```
pip3 install py-staticmaps
```

Mais ça bloque sur une erreur, et quand j'essaie de la résoudre, c'est la réaction en chaîne de messages d'erreur que je ne comprends qu'à moitié (la moitié vide). Alors, on verra plus tard...

Réalisation

Étape 1 : le fond de carte

Un fond de carte de Quimper, en noir et blanc, en A4 paysage 300 dpi (avec une petite marge de 5 mm sur chaque bord)

- définition x : (29.7 - 1.0) / 2.54 * 300 = 3390 pixels
- définition y : (21 - 1.0) / 2.54 * 300 = 2362 pixels

Pour trouver le niveau de zoom et centrer la carte sur un point, on peut utiliser openstreetmap . Dans l'URL, on peut y lire les coordonnées lat/lon et le niveau de zoom, exemple : https://www.openstreetmap.org/#map=13/47.9968/-4.1043

- zoom : 13
- lat. : 47.9968
- lon. : -4.1043

Au sujet des niveaux de zoom : https://wiki.openstreetmap.org/wiki/Zoom_levels

Pour le fond de carte :

```
# test avec https://github.com/komoot/staticmap
# Python 3.5.3 / pip 9.0.1 / staticmap 0.5.5
# Debian 9.5 @ kirin / 20210531

from staticmap import StaticMap, CircleMarker

m = StaticMap(3390, 2362, url_template='http://a.tile.stamen.com/toner-lite/{z}/{x}/{y}.png')

marker_outline = CircleMarker((-4.1043, 47.9968), 'white', 18)
marker = CircleMarker((-4.1043, 47.9968), '#0036FF', 12)

m.add_marker(marker_outline)
m.add_marker(marker)

image = m.render(zoom=16)
image.save('fond_de_carte_quimper.png')
```

Le jeu de tuile est toner-lite de Stamen (voir http://maps.stamen.com/)
Pour une raison qui m'échappe, le niveau de zoom n'est pas celui que j'avais trouvé avec OSM ?

Ce qui donne (extrait seulement, le fichier complet fait 4.8 MO)

Étape 2 : les données

Une carte de test : https://umap.openstreetmap.fr/fr/map/quimper_test_data_620279 (edit) / Les données sont exportées en geojson depuis l'onglet «partager/exporter» de umap.

Les marqueurs sont numérotés, ils devront être reliés comme suit : (1,2), (2,3), (3,4), (4,5), (5,6), (6,7) mais par la suite ça devra fonctionner avec n'importe quelle paire de nombre.

Installation de geojson

```
pip3 install geojson
```

Lire les données avec geojson

Les données sont extraites du fichier geojson et placées dans un dictionnaire

```
# Lecture de données geojson
# Python 3.5.3 / pip 9.0.1 / geojson 2.5.0
# Debian 9.5 @ kirin / 20210531

import geojson
with open("./quimper.geojson") as f:
    data = geojson.load(f)
points = {}    # créer un dictionnaire contenant les points
for feature in data['features']:
    nom = feature['properties']['name']
    lon = feature['geometry']['coordinates'][0]
```

```
    lat = feature['geometry']['coordinates'][1]
    points[nom] = [lon, lat]     # placer le point dans le dictionnaire

for key, value in points.items():
    print("nom :", key, "lon :", value[0], "lat :", value[1])
```

Chemins

Les chemins qui relient les points sont indiqués dans une liste de liste

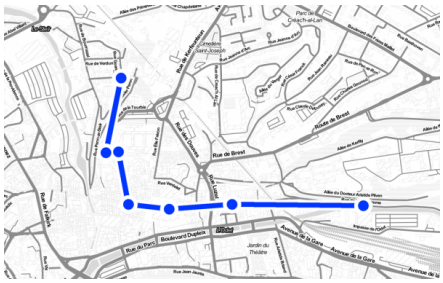
```
chemins =   [
                [1, 2],
                [2, 3],
                [3, 4],
                [4, 5],
                [5, 6],
                [6, 7]
            ]
for chemin in chemins:
    print("Tracer un chemin entre", chemin[0], " et ", chemin[1])
```

Étape 3 : tout rassembler

Procédure pour le script complet :

- créer le fond de carte
- définir les chemins dans une structure de données
- charger le fichier geojson
- tracer les contours des chemins et les contours des points
- tracer les couleurs pour les points et les chemins

Résultat : ça fonctionne (maintenant il restera à l'appliquer aux données réelles!), extrait :



Code complet

fond_de_carte_quimper.py (cliquer pour afficher le code)

fond_de_carte_quimper.py

```
        # Plan de circulation d'après des données geojson
        # Python 3.5.3 / pip 9.0.1 / geojson 2.5.0 / staticmap 0.5.5
        # Debian 9.5 @ kirin / 20210531

        import geojson
        from staticmap import StaticMap, CircleMarker, Line

        m = StaticMap(3390, 2362, url_template='http://a.tile.stamen.com/toner-lite/{z}/{x}/{y}.png')

        with open("./quimper.geojson") as f:
            data = geojson.load(f)

        points = {} # créer un dictionnaire contenant les points

        for feature in data['features']:
            nom = feature['properties']['name']
            lon = feature['geometry']['coordinates'][0]
            lat = feature['geometry']['coordinates'][1]
            points[nom] = [lon, lat] # placer le point dans le dictionnaire

        # Les chemins relient les points
        chemins =   [
                        [1, 2],
                        [2, 3],
                        [3, 4],
                        [4, 5],
                        [5, 6],
                        [6, 7]
                    ]
```

```
# Tracer les contours de marqueurs et de chemins
for key, value in points.items():
    marker_outline = CircleMarker((value[0], value[1]), 'white', 42)
    m.add_marker(marker_outline)
    for chemin in chemins:
        if chemin[0] == int(key) or chemin[1] == int(key):
            point1 = str(chemin[0])
            point2 = str(chemin[1])
            coordinates = [points[point1], points[point2]]
            line_outline = Line(coordinates, 'white', 24)
            m.add_line(line_outline)

# Tracer les chemins
for key, value in points.items():
    for chemin in chemins:
        if chemin[0] == int(key) or chemin[1] == int(key):
            point1 = str(chemin[0])
            point2 = str(chemin[1])
            coordinates = [points[point1], points[point2]]
            line = Line(coordinates, '#0036FF', 12)
            m.add_line(line)

# Tracer les marqueurs
for key, value in points.items():
    marker = CircleMarker((value[0], value[1]), '#0036FF', 24)
    m.add_marker(marker)

image = m.render(zoom=16)
image.save('fond_de_carte_quimper.png')
```

Une autre carte

Cette fois, ce sont des tracés de ligne dans umap qui sont utilisés pour définir une carte de trajets. Les couleurs utilisés dans umap définissent les différents segments des trajets.
J'ai essayé sans succès de réaliser avec Static Map une carte vide avec les noms de rues pour superposer aux trajets, mais ça ne fonctionne pas, le cadrage de la carte se faisant en fonction des éléments placés (et du niveau de zoom), les deux cartes ne se superposent pas.
Peut-être que de meilleurs résultats pourraient être obtenus avec py-staticmaps ?

fond_de_carte_quimper_trajets.py (cliquer pour afficher le code)

fond_de_carte_quimper_trajets.py

```
# Plan de pédibus d'après des données geojson
# Python 3.5.3 / pip 9.0.1 / geojson 2.5.0 / staticmap 0.5.5
# Debian 9.5 @ kirin / 20210601

import geojson
from staticmap import StaticMap, CircleMarker, Line

# définition choisie pour un A4 avec 5mm de marge sur chaque bord
m = StaticMap(3390, 2362, 0, 0, url_template='http://a.tile.stamen.com/toner-lite/{z}/{x}/{y}.png')

with open("./mobilite_douce_quimper.geojson") as f:
    data = geojson.load(f)

pedibus_corniguel = [] # contiendra les segments du premier trajet
pedibus_caphorn = []   # contiendra les segments du second trajet

# Trier les trajets selon la couleur choisie dans umap
for feature in data['features']:
    if feature['geometry']['type'] == "LineString":
        color = feature['properties']['_umap_options']['color']
        coordinates = feature['geometry']['coordinates']
        if color == "Red":
            pedibus_caphorn.append(coordinates)
        if color == "MediumVioletRed":
            pedibus_corniguel.append(coordinates)

# Debug
print("pedibus_corniguel", pedibus_corniguel, "\n")
print("pedibus_caphorn", pedibus_caphorn, "\n")

# Dans un premier temps, tracer les contours
for trajet in pedibus_corniguel:
    ltraj = len(trajet)
    for x in range(0, ltraj-1):
        #print(trajet[x][0], trajet[x][1], "->", trajet[x+1][0], trajet[x+1][1])
        point1 = [ trajet[x][0], trajet[x][1] ]
        point2 = [ trajet[x+1][0], trajet[x+1][1] ]
        coordinates = [point1, point2]
        line_outline = Line(coordinates, 'white', 24)
        m.add_line(line_outline)
```

```
for trajet in pedibus_caphorn:
    ltraj = len(trajet)
    for x in range(0, ltraj-1):
        #print(trajet[x][0], trajet[x][1], "->", trajet[x+1][0], trajet[x+1][1])
        point1 = [ trajet[x][0], trajet[x][1] ]
        point2 = [ trajet[x+1][0], trajet[x+1][1] ]
        coordinates = [point1, point2]
        line_outline = Line(coordinates, 'white', 24)
        m.add_line(line_outline)

# Dans un second temps, trajet les chemins
for trajet in pedibus_corniguel:
    ltraj = len(trajet)
    for x in range(0, ltraj-1):
        #print(trajet[x][0], trajet[x][1], "->", trajet[x+1][0], trajet[x+1][1])
        point1 = [ trajet[x][0], trajet[x][1] ]
        point2 = [ trajet[x+1][0], trajet[x+1][1] ]
        coordinates = [point1, point2]
        line = Line(coordinates, '#e77214', 12)
        m.add_line(line)

for trajet in pedibus_caphorn:
    ltraj = len(trajet)
    for x in range(0, ltraj-1):
        #print(trajet[x][0], trajet[x][1], "->", trajet[x+1][0], trajet[x+1][1])
        point1 = [ trajet[x][0], trajet[x][1] ]
        point2 = [ trajet[x+1][0], trajet[x+1][1] ]
        coordinates = [point1, point2]
        line_outline = Line(coordinates, '#62c92b', 12)
        m.add_line(line_outline)

image = m.render(zoom=16)
image.save('fond_de_carte_quimper_pedibus.png')

# Et une autre couche avec uniquement les noms de rues à superposer
# malheureusement ça ne marche pas, la carte est cadrée différemment...
"""
carte_rues = StaticMap(3390, 2362, 0, 0, url_template='http://a.tile.stamen.com/toner-labels/{z}/{x}/{y}.png')
marker = CircleMarker((-4.118553, 47.985771), '#0036FF', 1)
carte_rues.add_marker(marker) # nécessaire, on ne peut pas faire le rendu d'une carte vide
image_rues = carte_rues.render(zoom=16)
image_rues.save('fond_de_carte_quimper_pedibus_labels.png')
"""
# Une alternative pourrait être de recréer la page avec les trajets
# en les mettant en couleur transparente
```

Ressources

Des alternatives / services alternatifs : https://wiki.openstreetmap.org/wiki/Static_map_images
Différents fournisseurs de tuiles (fonds de carte) : https://wiki.openstreetmap.org/wiki/Tile_servers
Les dictionnaires en python : https://www.mfitzp.com/tutorials/python-dictionaries/

Article extrait de : http://lesporteslogiques.net/wiki/ - **WIKI Les Portes Logiques**
Adresse : **http://lesporteslogiques.net/wiki/ressource/code/cartographie/start?rev=1622591248**
Article mis à jour: **2021/06/02 01:47**