

GEOCODING WITH THE GOOGLE MAPS API: USING PROC FCMP

TO CALL USER-DEFINED SAS[®] AND PYTHON FUNCTIONS
THAT GEOCODE COORDINATES INTO ADDRESSES,
CALCULATE ROUTES, AND MORE!

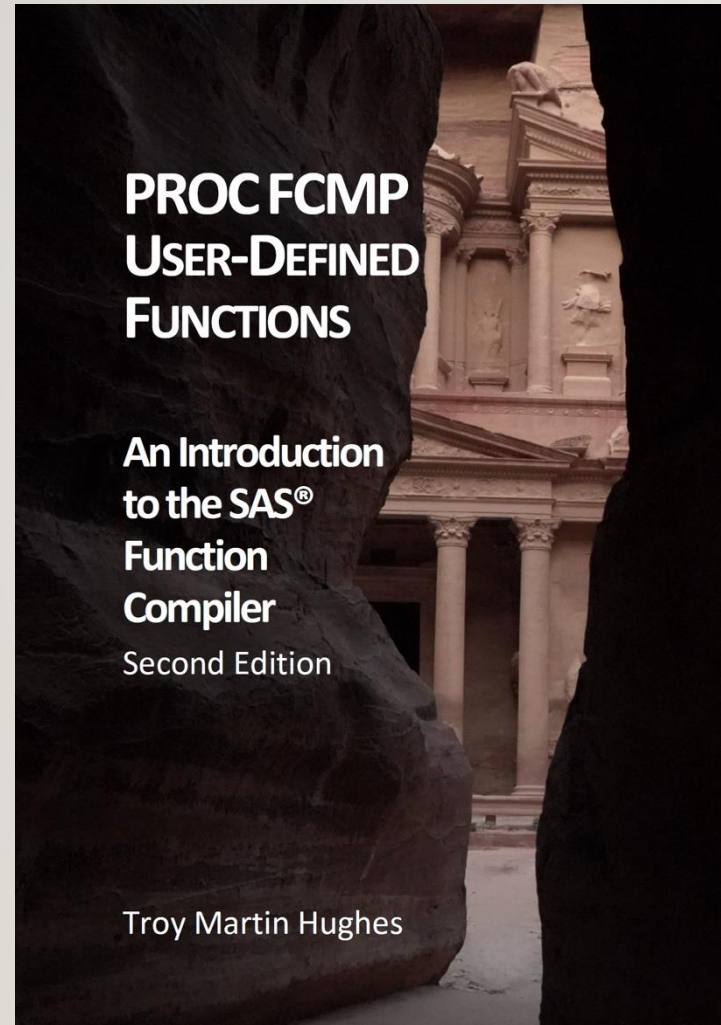
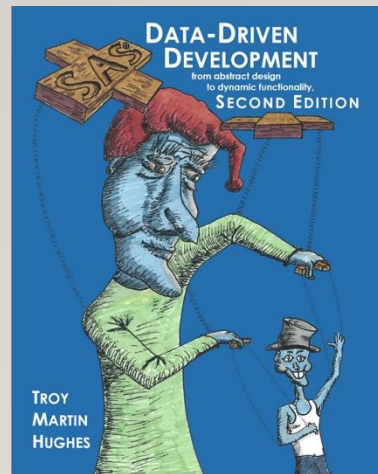
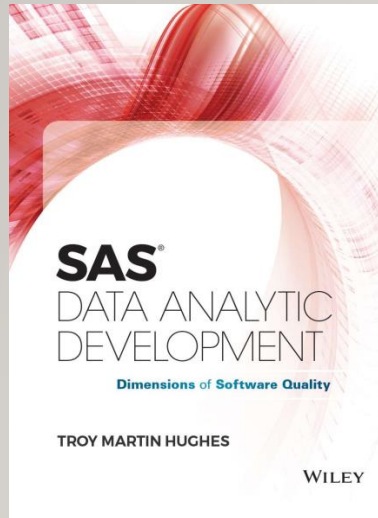
TROY MARTIN HUGHES

MAR 2026

PHUSE



BIOGRAPHY



Troy has been a SAS practitioner for more than 20 years, has managed SAS projects in support of federal, state, and local government initiatives, and is a SAS Certified Base, Advanced, V8, and Clinical Trials Programmer. Since 2013, he has presented more than 180 talks, trainings, and posters at SGF, SAS Analytics Experience, SAS Explore, WUSS, SCSUG, MWSUG, SESUG, PharmaSUG, PHUSE, BASAS, BASUG, MSUG, and SASensei. He has an MBA in Information Systems Management and certifications including: PMP, PMI-RMP, PMI-PBA, PMI-ACP, SSCP, CSSLP, CISSP, CRISC, CISA, CGEIT, CISM, CSM, CSD, A-CSD, CSPO, CSP-SM, CSP-PO, CSP, SAFe Government Practitioner, Network+, Security+, CySA+, CASP+, Cloud+, and ITIL. Troy is a US Navy veteran with two deployments to Afghanistan.

OVERVIEW

Business Case

- Clinical Trials Scenario
- Consideration of Solutions
- Google Maps API

Technical Solution

- PROC FCMP, the SAS Function Compiler
- Python Component Object

Software Quality—maintainability, modularity, readability, reusability, configurability, interoperability

BUSINESS CASE: THE SAS-LOVING CRO

THE SAS-LOVING CRO VS. OPEN SOURCE

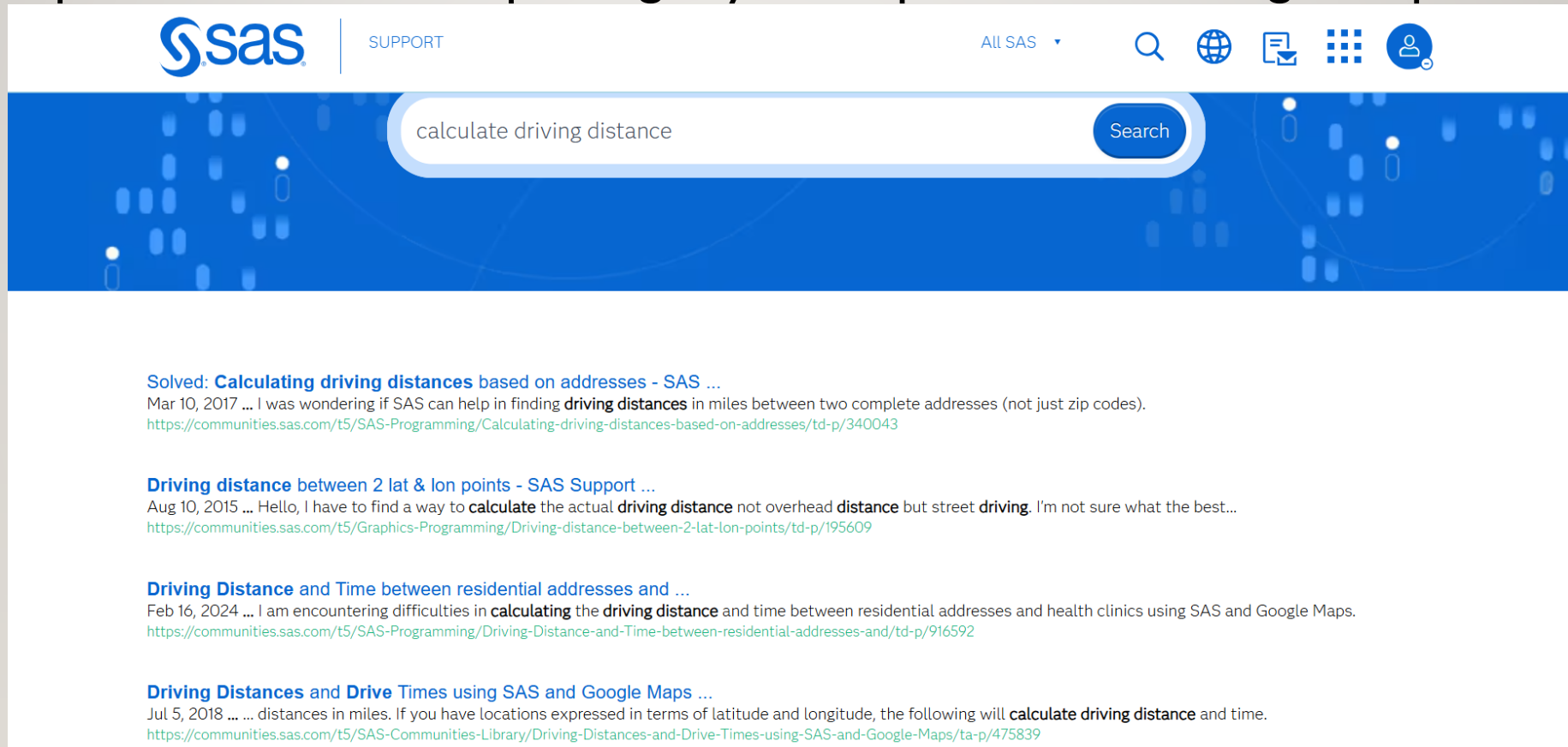
- Consider a study to assess the efficacy of topical ointment indicated for rashes
- And a CRO that operates exclusively in SAS:
 - All data are maintained in SAS data sets
 - ETL, analysis, reporting, and submission uses Base SAS and SAS macro language
 - Staff are predominantly SAS practitioners
- A Principal Investigator hypothesizes that a subject's location (i.e., residence) might influence disease severity and/or treatment outcomes:
 - Could local environmental factors correlate with treatment or outcomes?
 - Could distance from treatment centers correlate with treatment or outcomes?
- What tools exist to support these geospatial analyses?

THE SAS-LOVING CRO VS. OPEN SOURCE

- What geospatial tools should be leveraged to support the required geospatial analysis?
 - SAS GEOCODE procedure – geocodes address into latitude/longitude coordinates
 - SAS GEODIST function – calculates geodetic distance between two sets of coordinates
 - SAS ZIPCITYDISTANCE function – calculates geodetic distance between the centroids of two ZIPCODEs
- Is SAS the industry leader for this type of geospatial analysis? Arguably, no.

THE SAS-LOVING CRO VS. OPEN SOURCE

- Even a search on support.sas.com yields more "attempts" than "wins," and each of the top four hits relies on passing key-value pairs to the Google Maps URL:



THE SAS-LOVING CRO ~~VS.~~ AND OPEN SOURCE

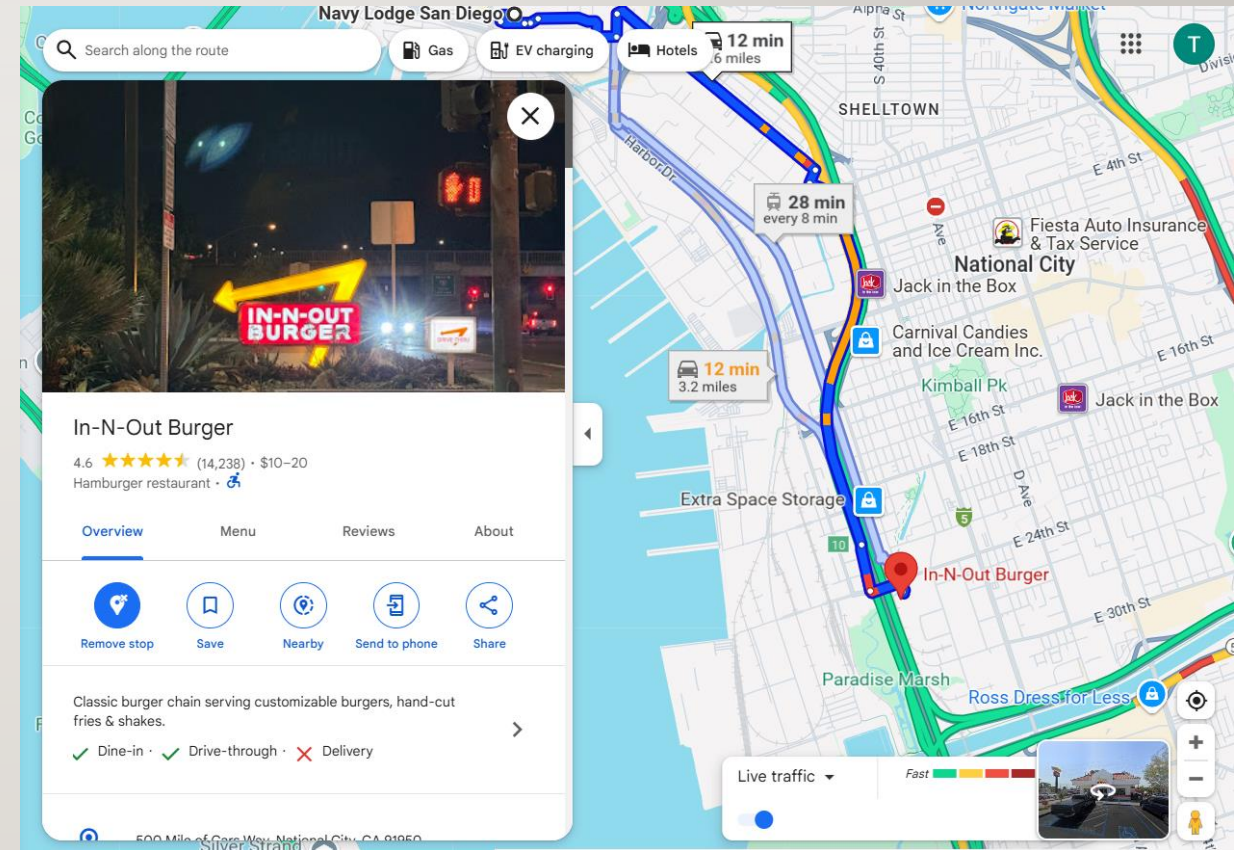
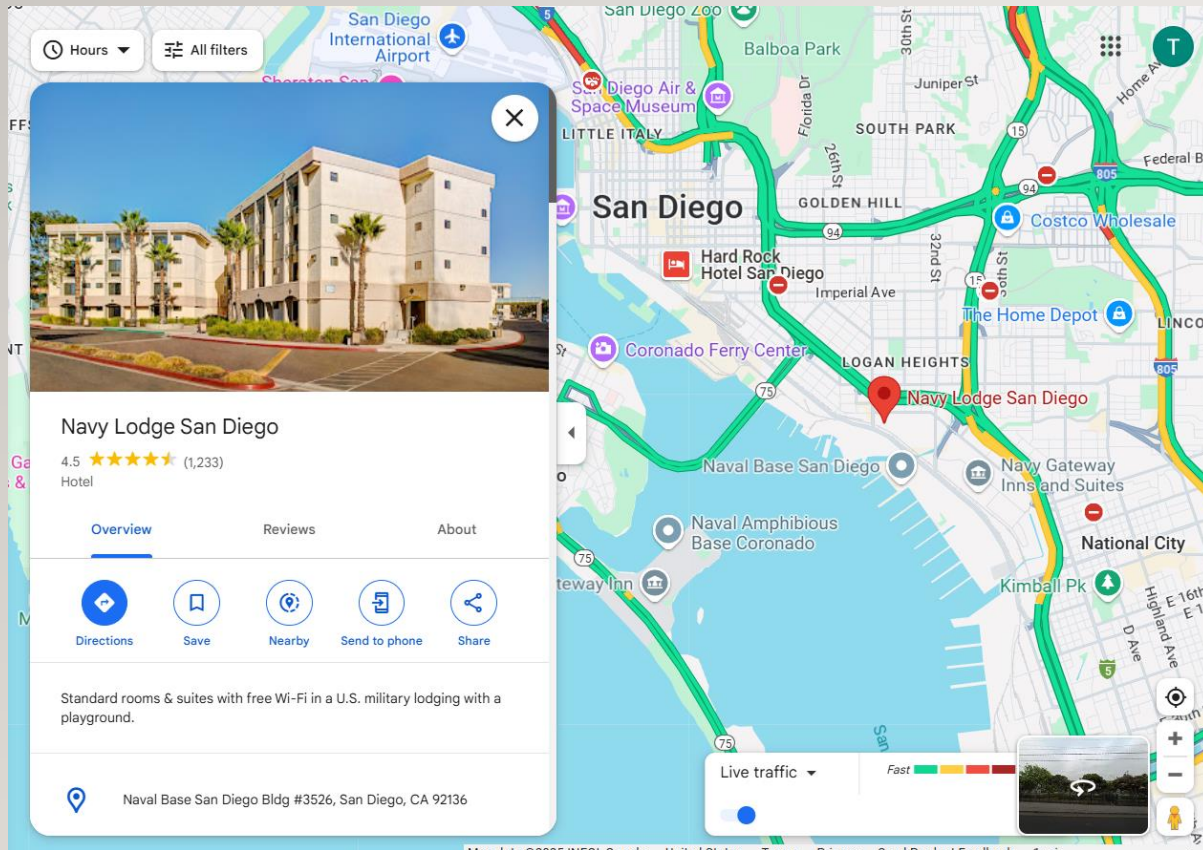
- There's no shame in relying on Google Maps for geospatial analysis, but let's do it the right way:
 - Design (or reuse) Python user-defined functions that call the Google Maps API Python methods
 - Call these user-defined Python functions natively in the Python interpreter *of your choice* using PROC FCMP user-defined functions and subroutines
 - No need for SAS macro language; all data are maintained in only built-in data structures (SAS data set, Python tuple, etc.)

SOFTWARE QUALITY...DELIVERED THROUGH THIS SOLUTION

- Modularity – the ability to make changes to one software component (like a function) without adversely affecting the functionality of other components
- Maintainability – the ease with which software can be modified when necessary—to expand functionality, improve performance, or correct defects
- Reusability – the ability of software components to be reused in the future—for diverse use cases and a diverse user base (not only the original developer(s))
- Readability – the ability to understand (through visual inspection of code) the software being executed
- Transferability – the ability of data to speak across platforms, apps, languages
- Configurability – the ability of end users to pass arguments to alter functionality

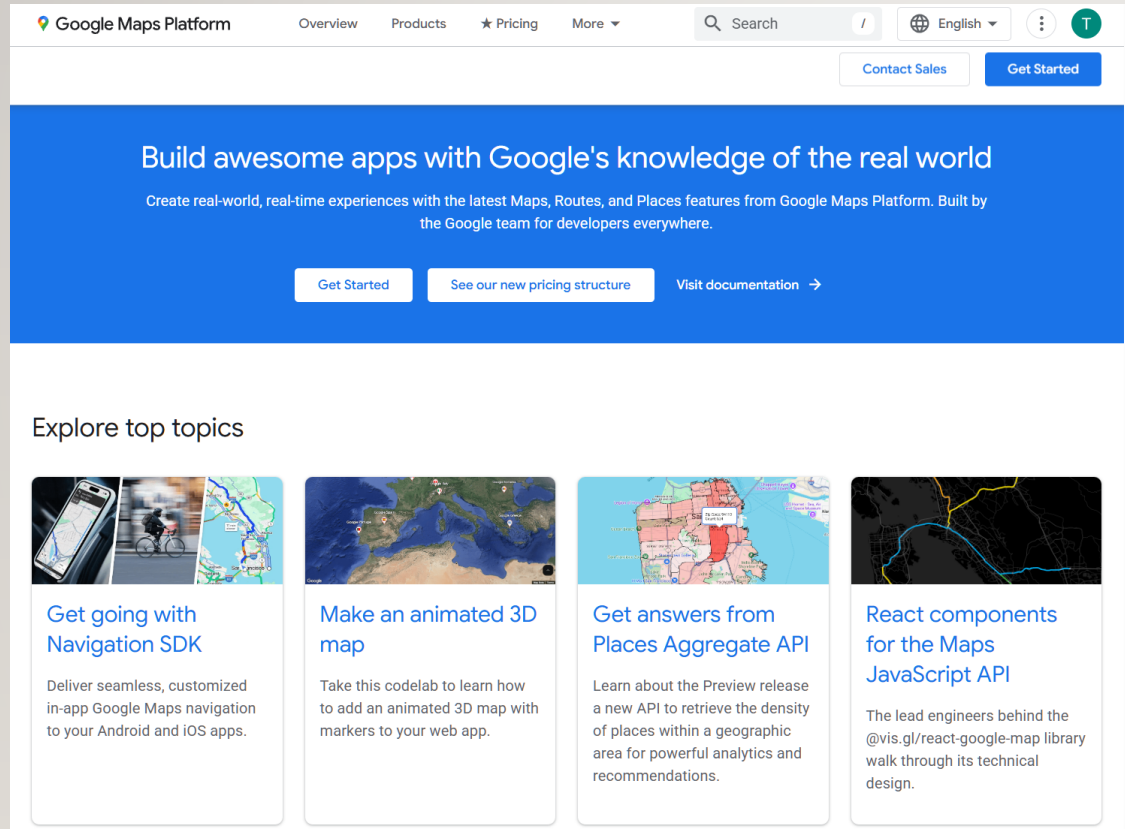
WHY ALL THE GOOGLE MAPS HYPE?

- Many users are only familiar with the manual web interface



WHY ALL THE GOOGLE MAPS HYPE?

- Far more functionality can be accessed programmatically from the Google Maps Developer Platform, including open-source Python, Java, Go, and Node.js



WHY ALL THE GOOGLE MAPS HYPE?

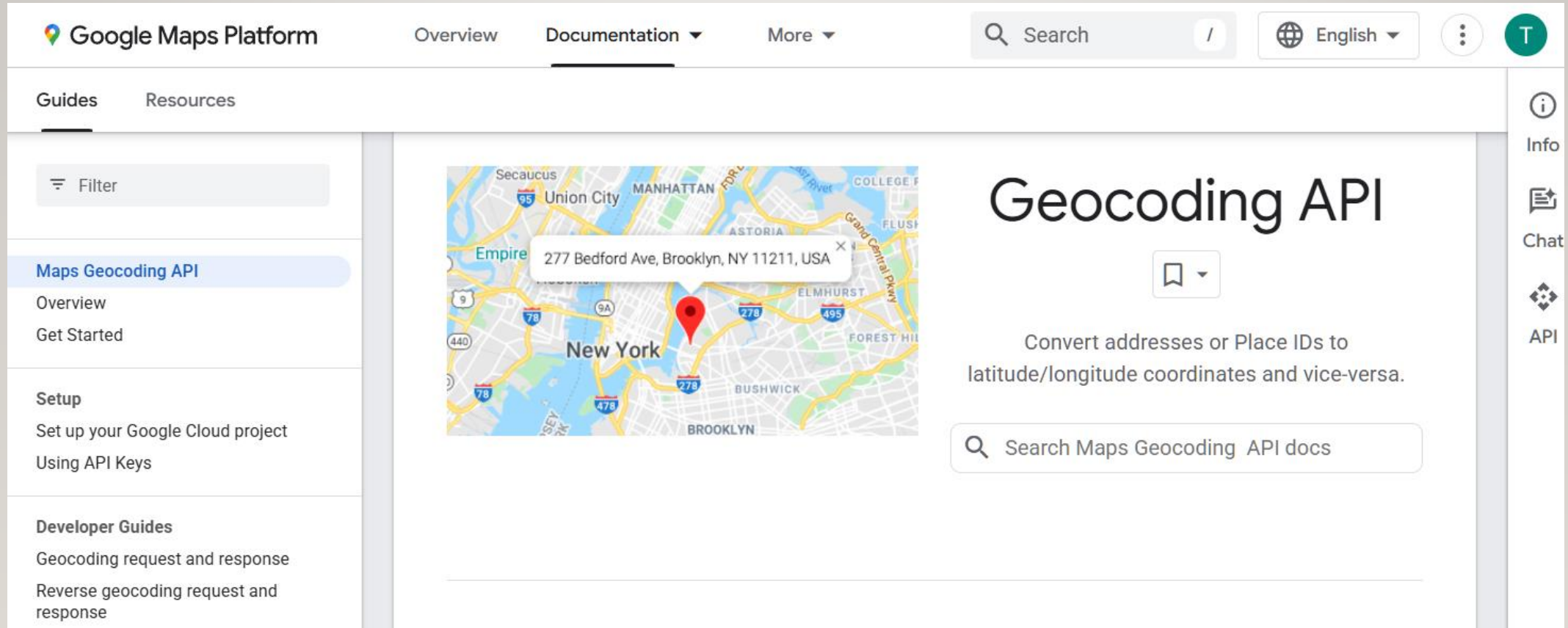
- A Google Cloud account is required (to track API webservice usage)
- A user (or organization) can enable or disable specific API functionality

The screenshot displays the Google Cloud console interface for the Google Maps Platform. The top navigation bar includes the Google Cloud logo, the project name 'My First Project', a search bar, and user account information. The left sidebar lists navigation options: Overview, APIs & Services (selected), Metrics, Quotas, Keys & Credentials, Support, Solution Library, Map Management, Map Styles, and Datasets. The main content area, titled 'APIs & Services', shows four API cards:

- Places UI Kit**: Visualize Places on your front-end. Status: **Enable**. Includes a 'Places' button and a 'Guides' link.
- Geocoding API**: Convert between addresses and geographic coordinates. Status: **Disable**. Includes 'Places', 'Keys', 'Metrics', and 'Guides' links.
- Geolocation API**: Location data from cell towers and WiFi nodes. Status: **Disable**. Includes 'Places', 'Keys', 'Metrics', and 'Guides' links.
- Places API (New)**: Next generation of the Places API with access to more than 200 million places. Status: **Enable**. Includes a 'Places' button and a 'Guides' link.

WHY ALL THE GOOGLE MAPS HYPE?

- Each API has published documentation, best practices, and sample code



The screenshot shows the Google Maps Platform documentation page for the Geocoding API. The page layout includes a top navigation bar with 'Google Maps Platform', 'Overview', 'Documentation' (selected), and 'More'. A search bar and language selector (English) are on the right. A left sidebar contains 'Guides' and 'Resources' tabs, with a 'Filter' button and a list of links including 'Maps Geocoding API', 'Overview', 'Get Started', 'Setup', and 'Developer Guides'. The main content area features a map of New York with a red pin at '277 Bedford Ave, Brooklyn, NY 11211, USA'. To the right of the map, the title 'Geocoding API' is displayed with a bookmark icon. Below the title, a description states: 'Convert addresses or Place IDs to latitude/longitude coordinates and vice-versa.' A search bar for 'Search Maps Geocoding API docs' is located at the bottom of the main content area. A right sidebar contains links for 'Info', 'Chat', and 'API'.

WHY ALL THE GOOGLE MAPS HYPE?

- Client Libraries detail language support for APIs

The screenshot shows the Google Maps Platform documentation page for 'Client Libraries for Google Maps Web Services'. The page is part of the 'Geocoding API' documentation under 'Web Services'. The breadcrumb trail is: Home > Products > Google Maps Platform > Documentation > Web Services > Geocoding API > Resources. The main heading is 'Client Libraries for Google Maps Web Services'. Below the heading, it states: 'The Java Client, Python Client, Go Client and Node.js Client for Google Maps Services are community supported client libraries, open sourced under the [Apache 2.0 License](#). They are available for download and contributions on GitHub, where you will also find installation instructions and sample code:'. The page includes a 'Get Started' button, a 'Contact sales' button, and a 'Send feedback' button. The left sidebar contains links for 'Support' (Support Options, Maps FAQ, Geocoding FAQ, Release Notes, Stay Informed, Terms of Use) and 'Best Practices' (Web Service Best Practices). The right sidebar contains links for 'Info', 'Chat', and 'API'.

Google Maps Platform Overview Documentation More Search / English T

Web Services > Geocoding API Get Started Contact sales

Guides Resources

Filter

Support
Support Options
Maps FAQ
Geocoding FAQ
Release Notes
Stay Informed
Terms of Use

Best Practices
Web Service Best Practices

Home > Products > Google Maps Platform > Documentation > Web Services
> Geocoding API > Resources

Was this helpful?

Client Libraries for Google Maps Web Services

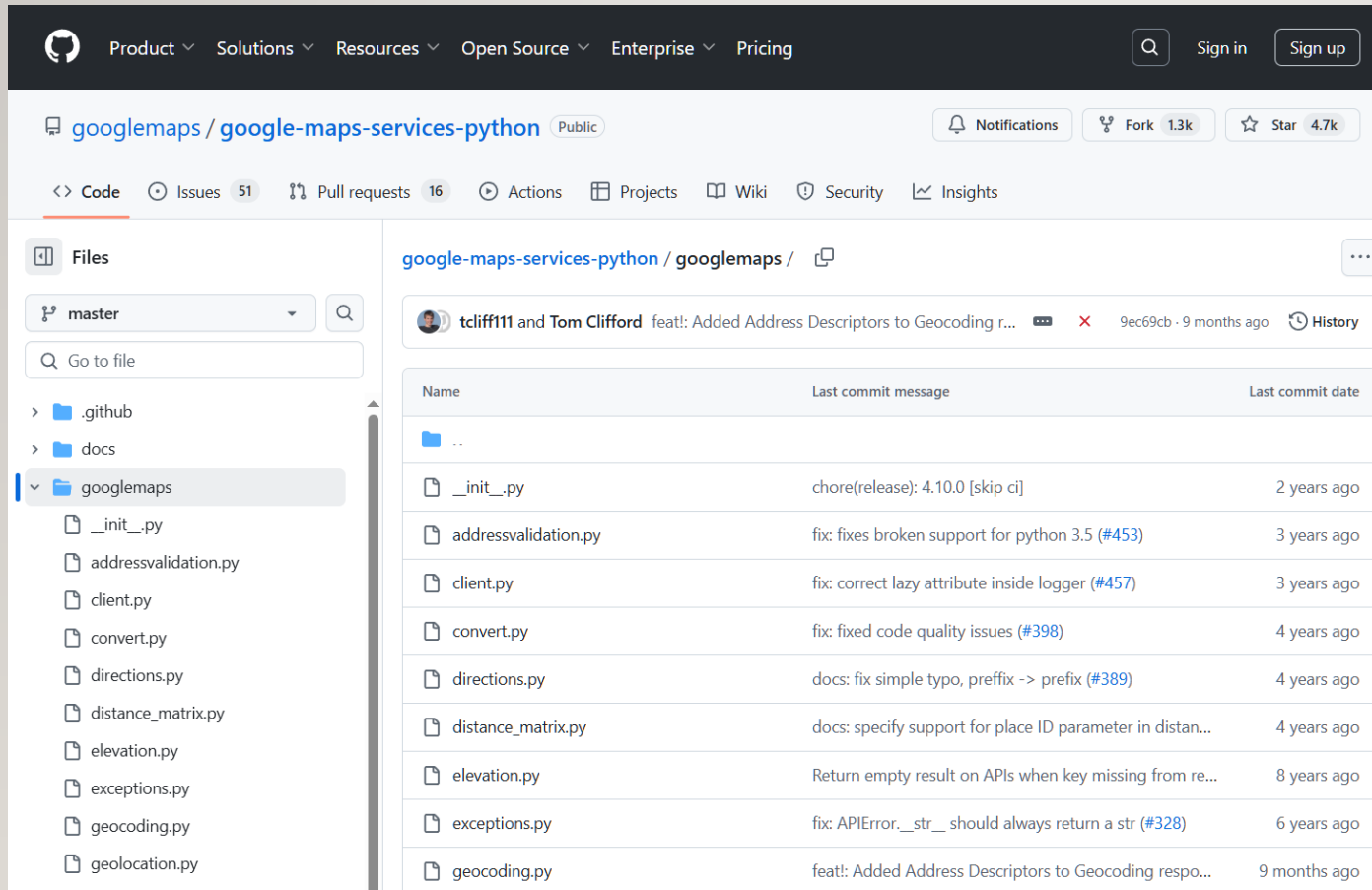
The Java Client, Python Client, Go Client and Node.js Client for Google Maps Services are community supported client libraries, open sourced under the [Apache 2.0 License](#). They are available for download and contributions on GitHub, where you will also find installation instructions and sample code:

Send feedback

Info
Chat
API

WHY ALL THE GOOGLE MAPS HYPE?

- Open-source methods and other functionality are hosted on GitHub



Product Solutions Resources Open Source Enterprise Pricing

googlemaps / google-maps-services-python Public

Notifications Fork 1.3k Star 4.7k

Code Issues 51 Pull requests 16 Actions Projects Wiki Security Insights

Files

master

Go to file

> .github

> docs

▼ googlemaps

- __init__.py
- addressvalidation.py
- client.py
- convert.py
- directions.py
- distance_matrix.py
- elevation.py
- exceptions.py
- geocoding.py
- geolocation.py

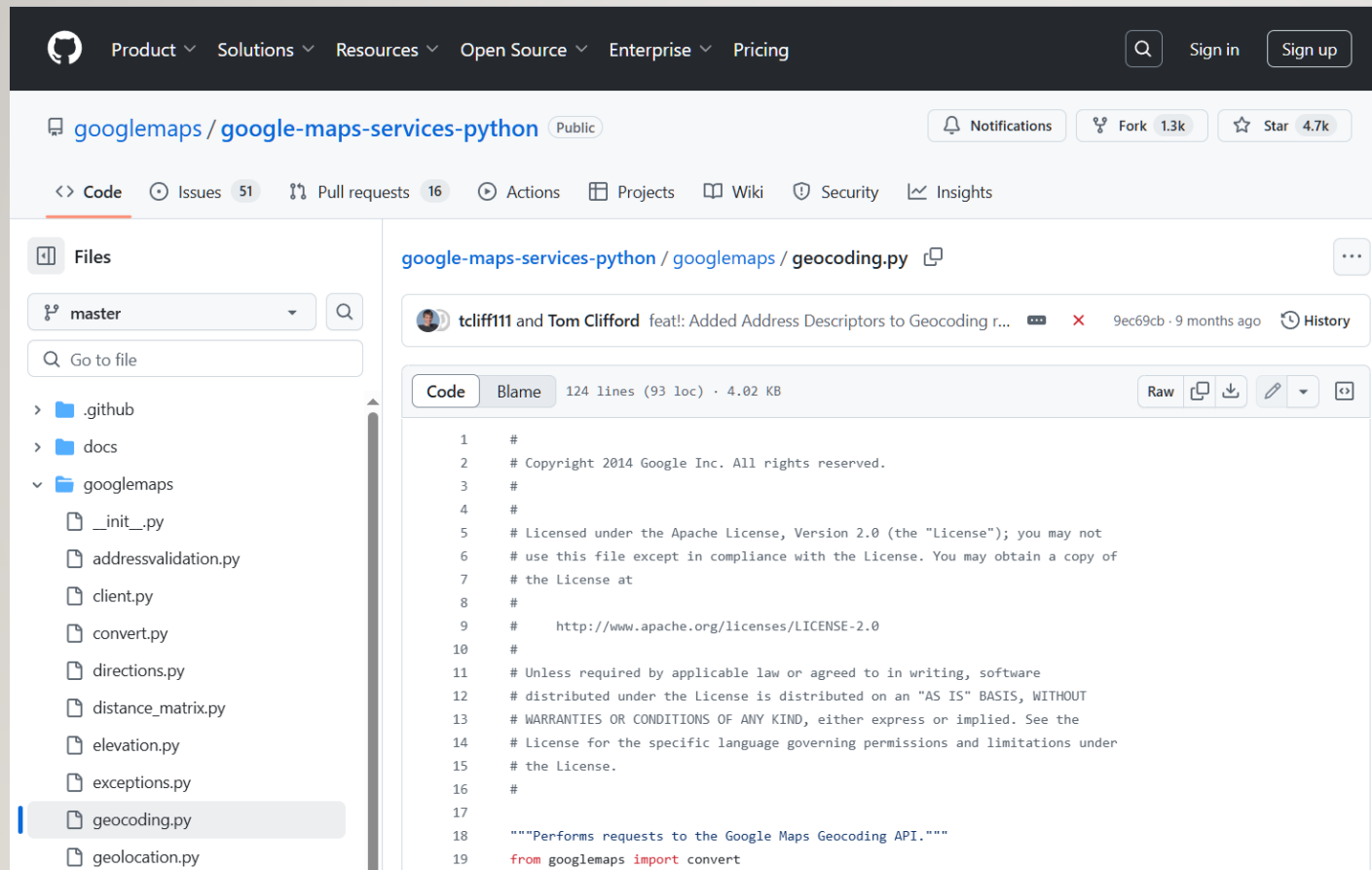
google-maps-services-python / googlemaps /

tlcliff11 and Tom Clifford feat!: Added Address Descriptors to Geocoding r... 9ec69cb · 9 months ago History

Name	Last commit message	Last commit date
..		
__init__.py	chore(release): 4.10.0 [skip ci]	2 years ago
addressvalidation.py	fix: fixes broken support for python 3.5 (#453)	3 years ago
client.py	fix: correct lazy attribute inside logger (#457)	3 years ago
convert.py	fix: fixed code quality issues (#398)	4 years ago
directions.py	docs: fix simple typo, prefix -> prefix (#389)	4 years ago
distance_matrix.py	docs: specify support for place ID parameter in distan...	4 years ago
elevation.py	Return empty result on APIs when key missing from re...	8 years ago
exceptions.py	fix: APIError.__str__ should always return a str (#328)	6 years ago
geocoding.py	feat!: Added Address Descriptors to Geocoding respo...	9 months ago

WHY ALL THE GOOGLE MAPS HYPE?

- So yes, Google Maps and its Developer Platform deserve all the hype!



The screenshot displays the GitHub interface for the `googlemaps/google-maps-services-python` repository. The repository is public and has 1.3k forks and 4.7k stars. The `Code` tab is selected, showing the `geocoding.py` file. The file is 124 lines (93 loc) and 4.02 KB. The code is a Python module for the Google Maps Geocoding API. It includes a license header for Apache License, Version 2.0, and a docstring describing the module's purpose. The code imports the `convert` function from the `googlemaps` module.

```
1 #
2 # Copyright 2014 Google Inc. All rights reserved.
3 #
4 #
5 # Licensed under the Apache License, Version 2.0 (the "License"); you may not
6 # use this file except in compliance with the License. You may obtain a copy of
7 # the License at
8 #
9 #     http://www.apache.org/licenses/LICENSE-2.0
10 #
11 # Unless required by applicable law or agreed to in writing, software
12 # distributed under the License is distributed on an "AS IS" BASIS, WITHOUT
13 # WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
14 # License for the specific language governing permissions and limitations under
15 # the License.
16 #
17
18 """Performs requests to the Google Maps Geocoding API."""
19 from googlemaps import convert
```

BUSINESS SOLUTION: SAS + GOOGLE MAPS

LEVELS OF ABSTRACTION (GEOCODING)

Business Need: Geocode street address into lat/lon coordinates (for subject residences AND treatment centers)—a bottom-up path through the levels:

- `geocode` method passes address to Geocoding API, returns coordinates to Python session
- `get_coords` user-defined Python function calls `geocode`, returns coordinates to Python function
- `sas_get_coords` user-defined FCMP subroutine calls `get_coords`, returns coordinates to SAS subroutine
- DATA step calls the `sas_get_coords` FCMP subroutine, returns coordinates to DATA step

LEVELS OF ABSTRACTION (GEOCODING IN CODE)

```
# get_coords.py
```

```
# import the Google Maps package
import googlemaps
```

```
* sas_get_coords.sas
```

```
# ini * define the folder location of Python program files;
gmaps %let loc_file=c:\goo\
```

```
* analysis.sas
```

```
# fun * call
coord proc options cmplib=work.funcs;
def g su
```

```
"
g
l
l
r
data analyze_pts;
    set pts;
    call sas_get_coords(pt_addy, lat, lon);
    put lat= lon=;
run;
```

```
lat=32.6848432 lon=-117.130015
```

```
quit; NOTE: There were 1 observations read from the data set
WORK.PTS.
```

```
NOTE: The data set WORK.ANALYZE_PTS has 1 observations and 4
variables.
```

LEVELS OF ABSTRACTION (DISTANCE CALCULATION)

Business Need: Calculate driving distance between two locations—a bottom-up path through the levels:

- `distance_matrix` method passes two addresses to Google Maps API, returns driving distance (in meters) to Python session
- `get_distance` user-defined Python function calls `distance_matrix`, returns driving distance (in miles) to Python function
- `sas_get_dist` user-defined FCMP function calls `get_distance`, returns driving distance (in miles) to SAS function
- DATA step calls the `sas_get_dist` FCMP function, returns driving distance (in miles) to DATA step

A CLINICAL TRIALS SCENARIO

CLINICAL TRIALS SCENARIO (WITH DATA)

- A subject—let's call him "Q"—enrolls in the study—because Q has "Q-goo"
- Q's residence is anonymized to the Navy Lodge:
 - Naval Base San Diego Bldg #3526, San Diego, CA 92136
- Various In-N-Out restaurant parking lots are being used as treatment centers at which the topical ointment is administered to subjects:
 - 1900 E Plaza Blvd, National City, CA 91950
 - 2910 Damon Ave, San Diego, CA 92109
 - 10370 Friars Rd, San Diego, CA 92120

GET_COORDS.PY USER-DEFINED PYTHON FUNCTION

```
# import the Google Maps package
import googlemaps

# initialize the Google Maps key
gmaps_key = googlemaps.Client(key="YOUR_SECRET_KEY_GOES_HERE")

# function evaluates street address and extracts lat/lon coords
def get_coords(addr):
    "Output: lat_key, lon_key"
    geo = gmaps_key.geocode(addr)
    lat = geo[0]['geometry']['location']['lat']
    lon = geo[0]['geometry']['location']['lng']
    return lat, lon
```

GEOCODE METHOD DATA STRUCTURE

```
geo = gmaps_key.geocode(addy)
[{'address_components': [{'long_name': '3455',
    'short_name': '3455',
    'types': ['street_number']}],
 {'long_name': 'Senn Street', 'short_name': 'Senn St', 'types': ['route']},
 {'long_name': 'Barrio Logan',
    'short_name': 'Barrio Logan',
    'types': ['neighborhood', 'political']},
 {'long_name': 'San Diego',
    'short_name': 'San Diego',
    'types': ['locality', 'political']},
 {'long_name': 'San Diego County',
    'short_name': 'San Diego County',
    'types': ['administrative_area_level_2', 'political']},
 {'long_name': 'California',
    'short_name': 'CA',
    'types': ['administrative_area_level_1', 'political']}
```

GEOCODE METHOD DATA STRUCTURE CONT.

```
{'long_name': 'United States',
  'short_name': 'US',
  'types': ['country', 'political']},
{'long_name': '92136', 'short_name': '92136', 'types': ['postal_code']},
'formatted_address': '3455 Senn St, San Diego, CA 92136, USA',
'geometry': {'location': {'lat': 32.6848432, 'lng': -117.1300151},
  'location_type': 'ROOFTOP',
  'viewport': {'northeast': {'lat': 32.6861158302915,
    'lng': -117.1287738197085},
    'southwest': {'lat': 32.6834178697085, 'lng': -117.1314717802915}}}},
'navigation_points': [{'location': {'latitude': 32.684517,
  'longitude': -117.130137},
  'restricted_travel_modes': ['WALK']},
  {'location': {'latitude': 32.6849949, 'longitude': -117.13018},
  'restricted_travel_modes': ['DRIVE']}],
'partial_match': True,
'place_id': 'ChIJ402zjVZS2YARgJhRq5k14dI',
'plus_code': {'compound_code': 'MVM9+WX San Diego, CA',
  'global_code': '8544MVM9+WX'},
'types': ['establishment', 'point_of_interest']]
```

SAS_GET_COORDS SUBROUTINE DEFINITION

- PROC FCMP compiles and saves one or more functions or subroutines

```
* define the folder location of Python program files;
%let loc_file=c:\goo\

* call get_coords Python function;
proc fcmp outlib = work.funcs.py;
    subroutine sas_get_coords(sas_addy $, sas_lat, sas_lon);
        outargs sas_lat, sas_lon;
        declare object py(python);
        rc = py.infile("&loc_file.get_coords.py");
        rc = py.publish();
        rc = py.call("get_coords", sas_addy);
        sas_lat = py.results["lat_key"];
        sas_lon = py.results["lon_key"];
    endsub;
quit;
```

SAS_GET_COORDS SUBROUTINE CALL

- Subject data are compiled, and `sas_get_coords` geocodes addresses

```
data subjects;
  length name $20 pt_addy $100 lat lon 8;
  name = 'Q';
  pt_addy = 'Naval Base San Diego Bldg #3526, San Diego, CA 92136';
run;

options cmplib=work.funcs;

data analyze_pts;
  set pts;
  call sas_get_coords(pt_addy, lat, lon);
  put lat= lon=;
run;
```

```
lat=32.6848432 lon=-117.130015
```

CALCULATING DISTANCE

- In-N-Out treatment center locations are initialized

```
data treatment_centers;  
  length center_name $50 center_addy $100;  
  call missing(lat, lon);  
  center_name = 'National City In-N-Out';  
  center_addy = '1900 E Plaza Blvd, National City, CA 91950';  
  output;  
  center_name = 'Pacific Beach In-N-Out';  
  center_addy = '2910 Damon Ave, San Diego, CA 92109';  
  output;  
  center_name = 'Mission Valley In-N-Out';  
  center_addy = '10370 Friars Rd, San Diego, CA 92120';  
  output;  
run;
```

CALCULATING DISTANCE

- **DATA** step geocodes both addresses, and calculates distance between them

```
data test_distance;
  length pt_addy center_addy $100 pt_lat pt_lon center_lat center_lon
         dist_miles 8;
  pt_addy = 'Naval Base San Diego Bldg #3526, San Diego, CA 92136';
  center_addy = '1900 E Plaza Blvd, National City, CA 91950';
  call missing(pt_lat, pt_lon, center_lat, center_lon, dist_miles);
  * geocode subject address;
  call sas_get_coords(pt_addy, pt_lat, pt_lon);
  * geocode center address;
  call sas_get_coords(center_addy, center_lat, center_lon);
  * calculate distance between origin and destination coords;
  dist_miles = sas_get_dist(pt_lat, pt_lon, center_lat, center_lon);
  put pt_lat= pt_lon= center_lat= center_lon= dist_miles=;
run;
```

CALCULATING DISTANCE

- DATA step geocodes both addresses, and calculates distance between them

```
data test_distance;
  length pt_addy center_addy $100 pt_lat pt_lon center_lat center_lon
         dist_miles 8;
  pt_addy = 'Naval Base San Diego Bldg #3526, San Diego, CA 92136';
  center_addy = '1900 E Plaza Blvd, National City, CA 91950';
  call missing(pt_lat, pt_lon, center_lat, center_lon, dist_miles);
  * geocode subject address;
  call sas_get_coords(pt_addy, pt_lat, pt_lon);
  * geocode center address;
  call sas_get_coords(center_addy, center_lat, center_lon);
  * calculate distance between origin and destination coords;
  dist_miles = sas_get_dist(pt_lat, pt_lon, center_lat, center_lon);
  put pt_lat= pt_lon= center_lat= center_lon= dist_miles=;
run;
```

```
pt_lat=32.6848432 pt_lon=-117.1300151 center_lat=32.6763009
center_lon=-117.085868
dist_miles=3.2
```

SAS_GET_DIST USER-DEFINED SAS FUNCTION

- The function calls the `get_distance` user-defined Python function

```
proc fcmp outlib = work.funcs.py;  
    function sas_get_dist(origin_lat, origin_lon, dest_lat, dest_lon);  
        declare object py2(python);  
        rc = py2.infile("&loc_file.get_distance.py");  
        rc = py2.publish();  
        rc = py2.call("get_distance", origin_lat, origin_lon, dest_lat,  
            dest_lon);  
        dist_miles = py2.results["dist_key"];  
        return(dist_miles);  
    endfunc;  
quit;
```

GET_DISTANCE USER-DEFINED PYTHON FUNCTION

- The Python function calls the `distance_matrix` Python method

```
import googlemaps

gmaps_key = googlemaps.Client(key="YOUR_SECRET_KEY_GOES_HERE")

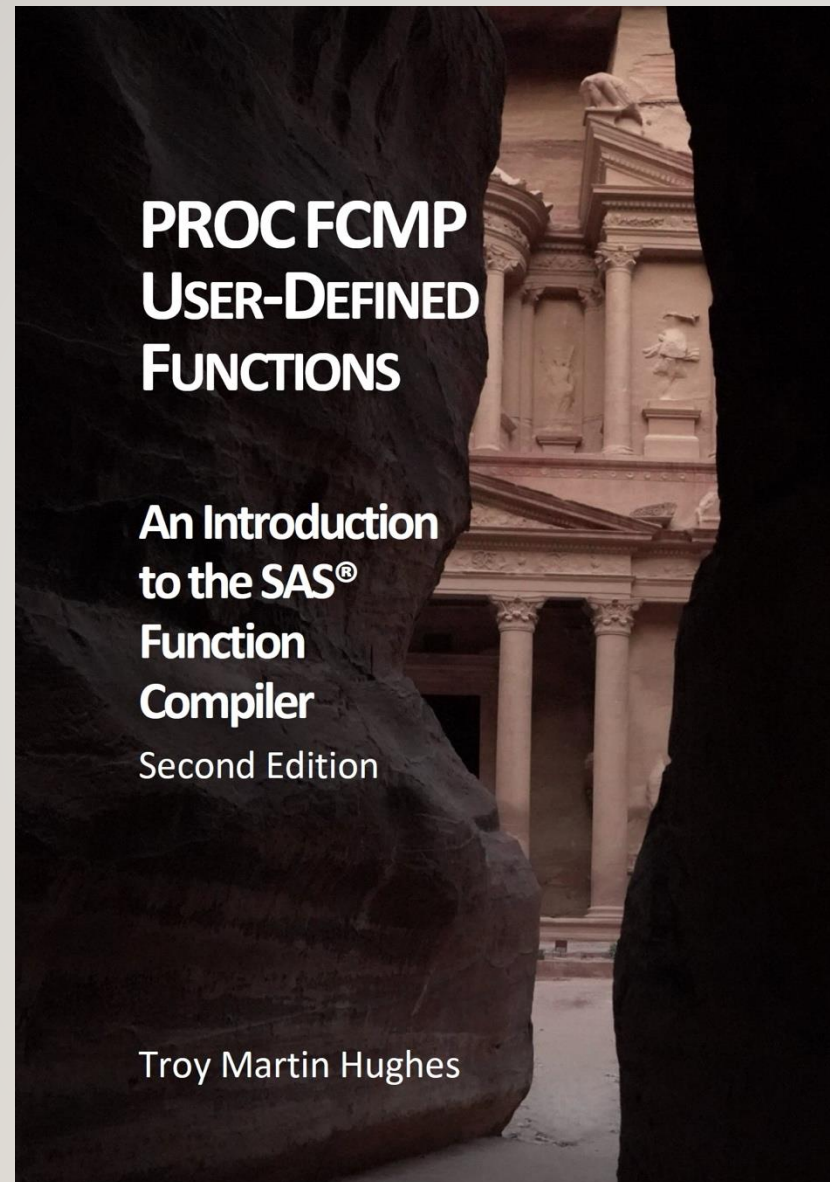
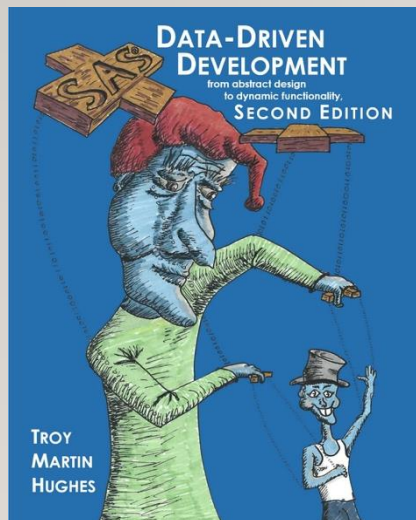
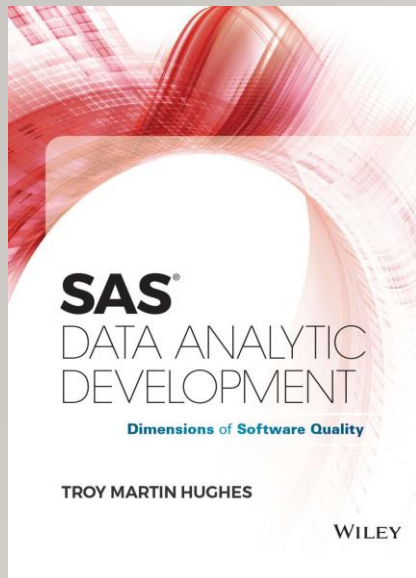
# calculate driving distance between two locations
def get_distance(lat_start, lon_start, lat_end, lon_end):
    "Output: dist_key"
    coords_origin = (lat_start, lon_start)
    coords_dest = (lat_end, lon_end)
    # generates distances in meters
    dist = gmaps_key.distance_matrix(coords_origin, coords_dest,
                                     mode='driving')['rows'][0]['elements'][0]['distance']['value']
    dist = round(dist * 0.000621371, 1) # converts meters to miles
    return dist
```

NEXT STEPS

- Google Maps API webservice calls can become expensive (including I/O resources and pay-per-call webservice fees after free usage threshold is surpassed); saving geocoding results in a SAS hash object (or Python dictionary) can facilitate *memoization*, in which the results of past resource-intensive calls are stored in memory.
 - demonstrated in “Python Component Object” and “Memoization” chapters of *PROC FCMP User-Defined Functions, Second Edition* (Hughes, 2026)
- With lat/lon coordinates extracted from subject residential addresses, point-in-polygon geospatial analysis could geospatially join census-tract-level, neighborhood-level, zip-level, city-level, and other-level data sources (like CDC).
 - demonstrated in *GIS Challenges of Cataloging Catastrophes* (Hughes, 2022)
- Provide customers the highest-caliber software, and support study participants!

CONCLUSION

- PROC FCMP facilitates user-defined functions and subroutines that can call user-defined Python functions via the Python Component Object
- Users should leverage Python solutions where they are a natural fit, and where existing functions can be called natively from their Python IDE of choice
- The flexibility and configurability of PROC FCMP often replaces messier, unnecessary SAS macro language, and uses built-in data structures
- The inherent modular software design of user-defined functions facilitates chunks of code that are more maintainable, readable, testable, and reusable
- Like all other tools, users should understand the high-level functionality of PROC FCMP sufficiently well to know when to use (and not to use) it



troymartinhughes@gmail.com
<https://www.linkedin.com/in/troy-hughes-27a998a8>

