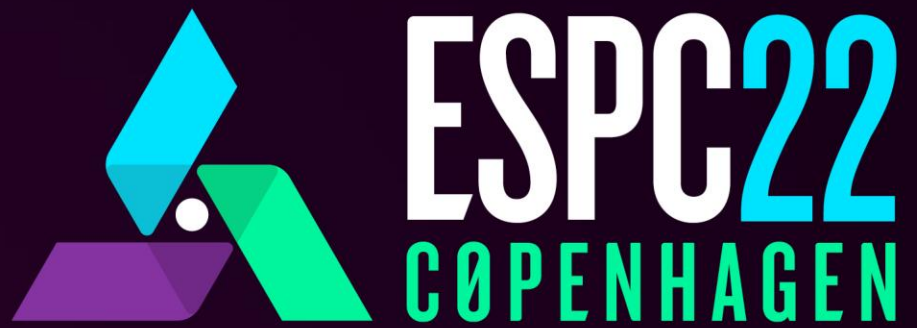

European
SharePoint
Office 365 & Azure
Conference


ESPC22
C0PENHAGEN

Microsoft Graph Development with the Python SDK

Fabian G. Williams

Sr. Product Manager, MICROSOFT, USA

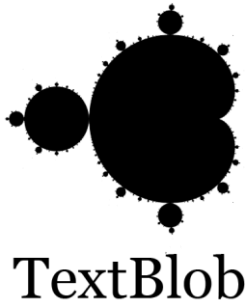
Gabrielle A. Williams

Student, Atholton HS, USA

Scenario

Suppose you want to be able to quickly and efficiently both capture data and reason over it with outputs in a powerful ‘excel-like’ form, perform sentiment analysis on textual content, or see powerful charts and graphs that can also instantly be outputted as images into a report. Well then… Python has all the libraries you need and with the Graph Python SDK, that became that much easier.

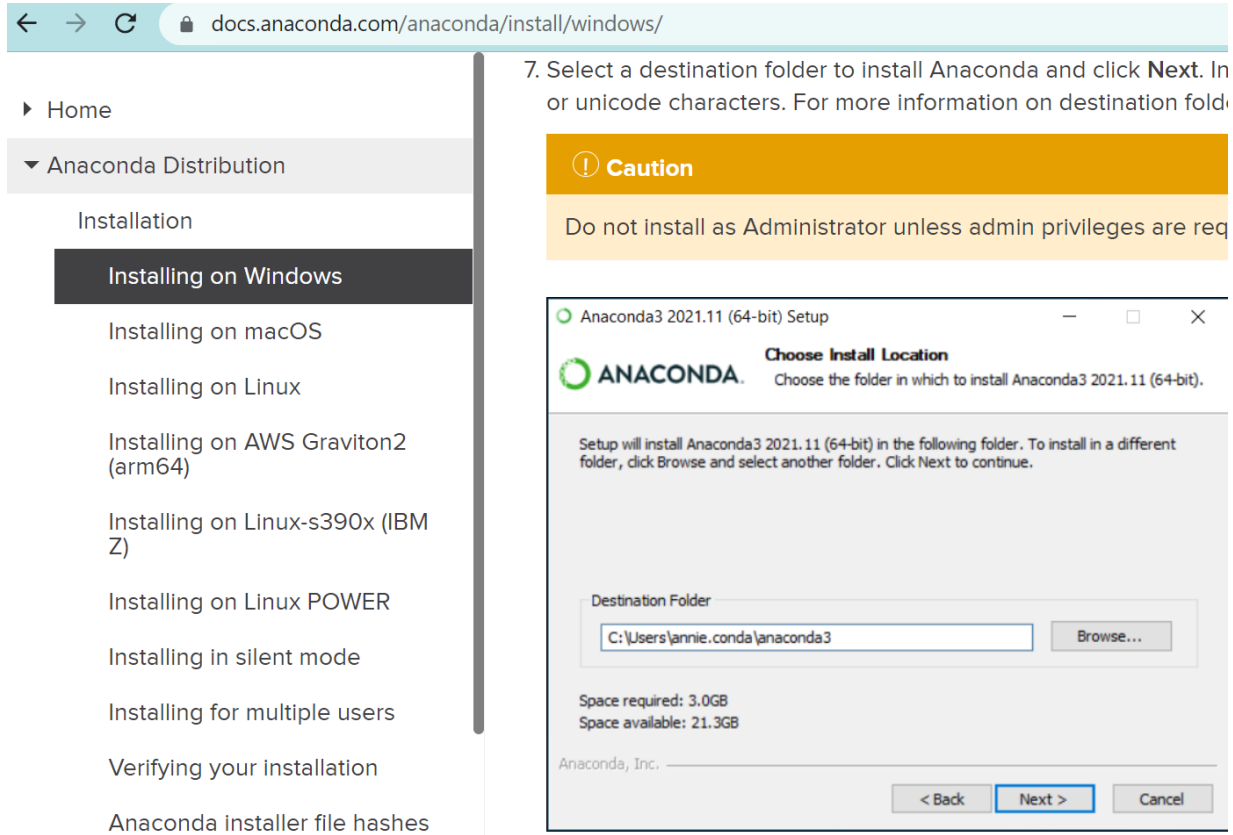
Are you a Python Dev looking to get into Microsoft 365 Dev? Python SDK is right for you!



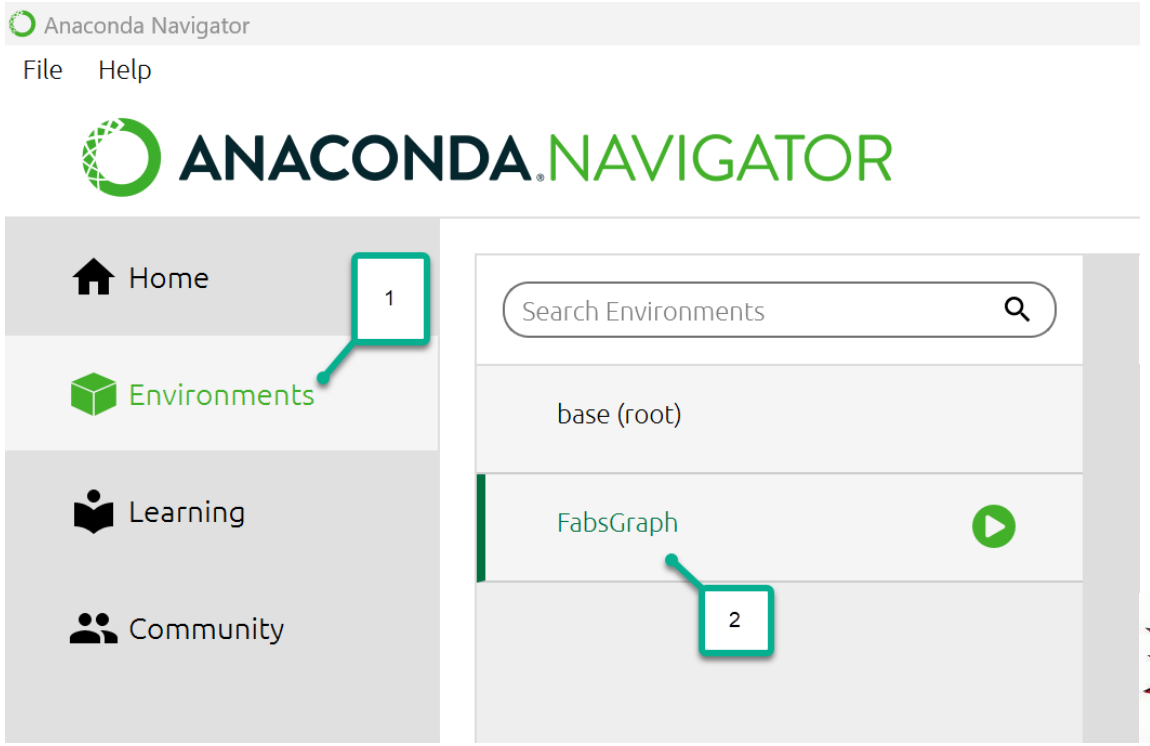
Getting set up in Python?

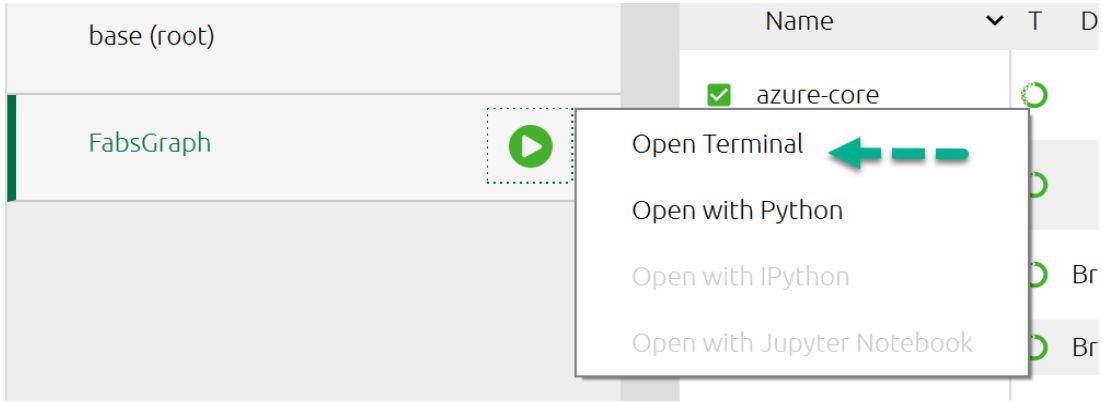
Setting up Python

OR



Setting up Python – Gabbys' Preferred - Anaconda

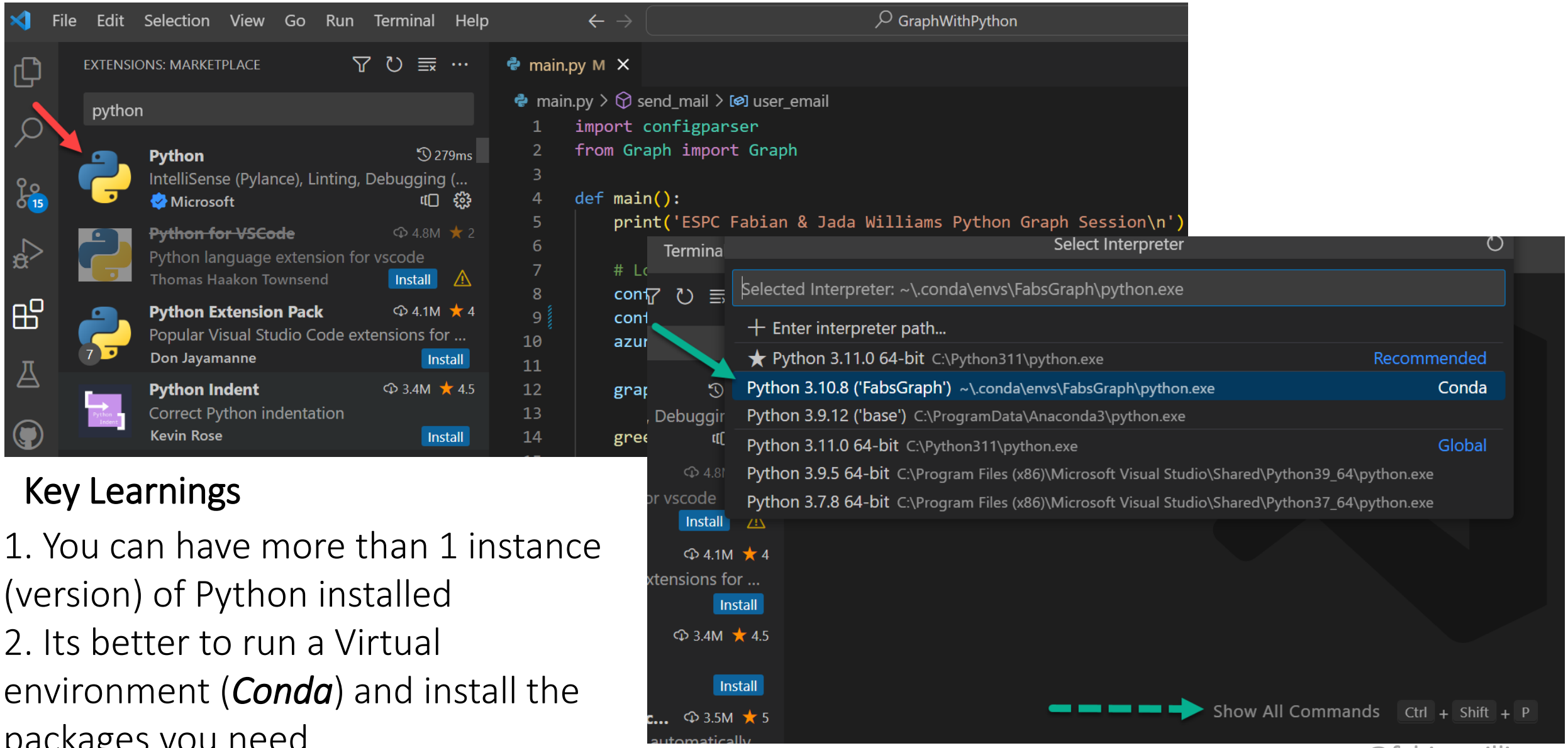




Key Learnings

- Setup a virtual environment
- Pull down the additional libraries you need
- Open the terminal from anaconda (conda) Then open up VSCode from there once you navigate to your working directory
- -- that will give you a shortcut to the "env" environment you set up w/out issuing a conda command to switch environment

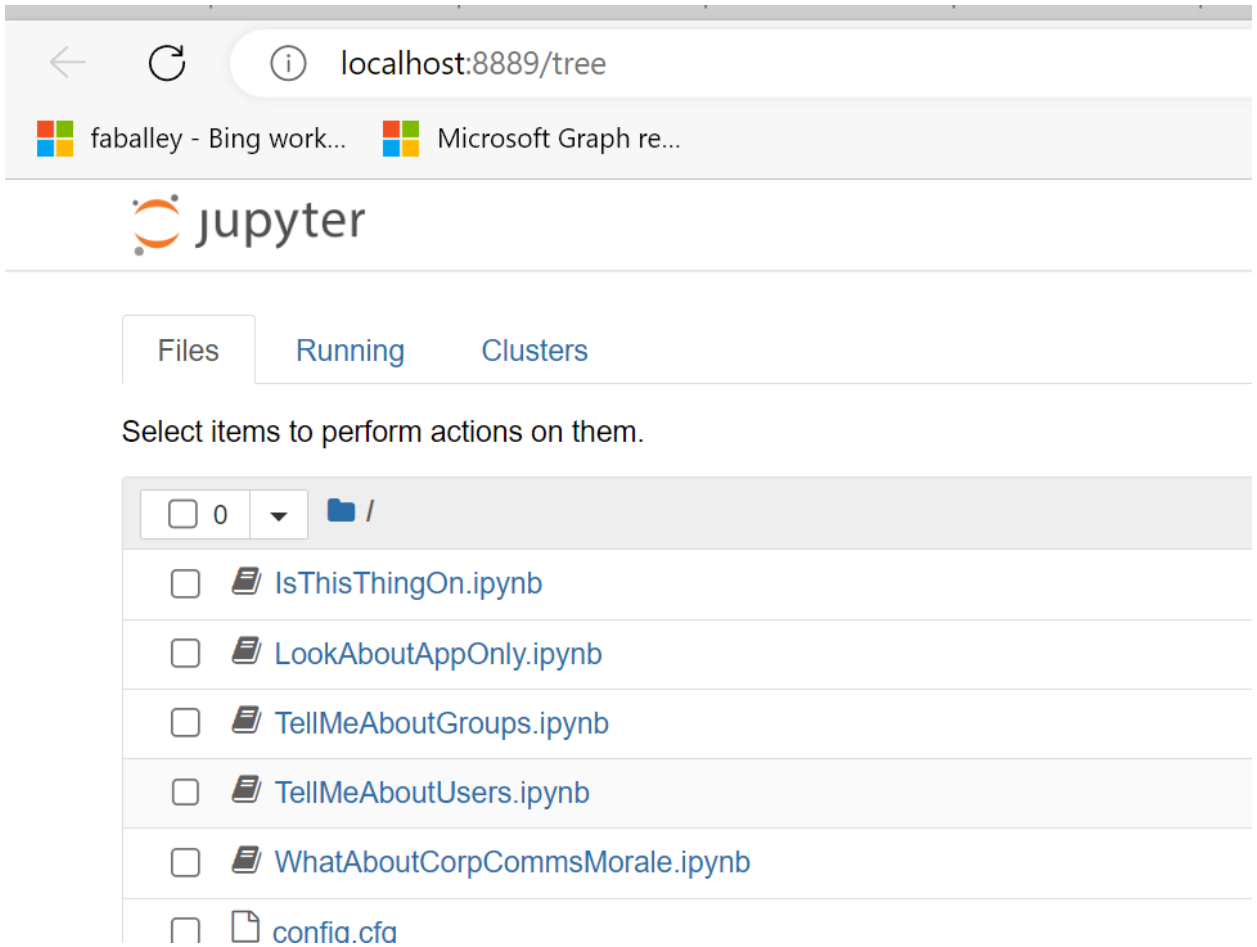
What about the IDE ?



Key Learnings

1. You can have more than 1 instance (version) of Python installed
2. Its better to run a Virtual environment (***Conda***) and install the packages you need

What about the IDE ? - … its fluid



Key Learnings

- If you need visuals for the work you will be doing in Python, use Jupyter notebook
- You can still run the commands as if you were in shell but it is step by step and you can also do a “run” all
- If you want to set up a cron job or have a schedule timer.. This is not the option for you however

How do I start to work?

main.py M

main.py > main

```python
import configparser
from Graph import Graph

def main():
    print('ESPC Fabian & Jada Williams Python

    # Load settings
    config = configparser.ConfigParser()
    config.read(['config.cfg'])
    azure_settings = config['azure']

    graph: Graph = Graph(azure_settings)

    greet_user(graph)

    choice = -1

    while choice != 0:
```

jupyter WhatAboutCorpCommsMorale Last Checkpoint: 3 hour

File Edit View Insert Cell Kernel Widgets Help

Run Code

In [7]:

```python
import json
import requests
import pandas as pd
from datetime import datetime
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import re
import configparser
from Graph import Graph
from textblob import TextBlob
from wordcloud import WordCloud
```

In [8]:

```python
# Load settings
config = configparser.ConfigParser()
config.read('config.cfg')
azure_settings = config['azure']

graph: Graph = Graph(azure_settings)
```

Function to my Mailbox

In [19]:

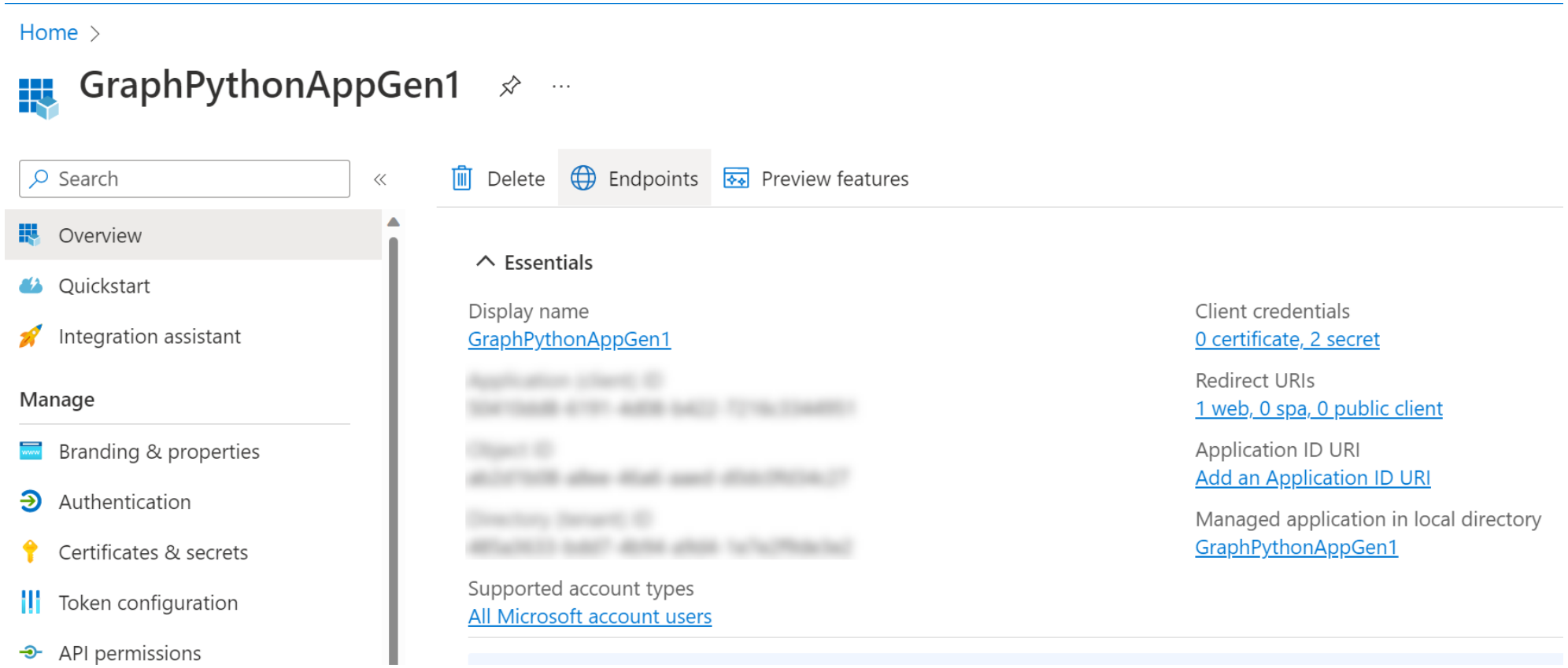
```python
def get_inbox(self):
    endpoint = '/me/mailFolders/inbox/messages'
    # Only request specific properties
    select = 'from,isRead,receivedDateTime,subjec
    # Get at most 25 results
    top = 25
    # Sort by received time, newest first
```

DEMO from VSCode

Console App that does CRUD against Graph Resources

Using the Microsoft Graph Python Core Libraries

Step 1 – Register your Application in Azure AD



Step 2 – Install your Libraries

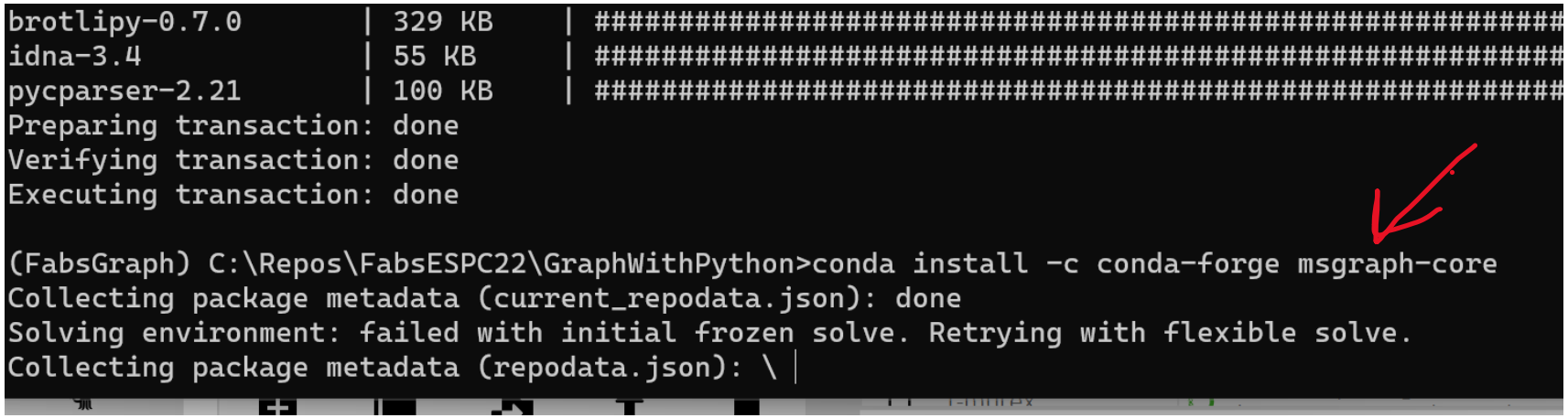
2. Install the required packages

msgraph-core is available on PyPI.

```
python -m pip install msgraph-core
python -m pip install azure-identity
```

If you are using native Python without a Distribution manager

If you are using Conda like I do



Step 3 – Import your Modules from Libraries

3. Import modules

```
from azure.identity import InteractiveBrowserCredential
from msgraph.core import GraphClient
```

The basic/ minimum of what you need IF you are only doing Browser Credential Flow

I set up for both Browser Credential Flow as well as App Only

```
import json
from configparser import SectionProxy
from azure.identity import DeviceCodeCredential, ClientSecretCredential
from msgraph.core import GraphClient

class Graph:
    settings: SectionProxy
    device_code_credential: DeviceCodeCredential
    user_client: GraphClient
    client_credential: ClientSecretCredential
    app_client: GraphClient

```

Step 4 – Configure your Client Credential Object

4. Configure a Credential Object

```
# Using InteractiveBrowserCredential for demonstration purposes.
# There are many other options for getting an access token. See the following for more information.
# https://pypi.org/project/azure-identity/

browser_credential = InteractiveBrowserCredential(client_id='YOUR_CLIENT_ID')
```

Literally 1 line of code

```
13    def __init__(self, config: SectionProxy):
14        self.settings = config
15        client_id = self.settings['clientId']
16        tenant_id = self.settings['authTenant']
17        graph_scopes = self.settings['graphUserScopes'].split(' ')
18
19        self.device_code_credential = DeviceCodeCredential(client_id, tenant_id = tenant_id)
20        self.user_client = GraphClient(credential=self.device_code_credential, scopes=graph_scopes)
21
```

Or set up a function to handle multiple scenarios

```
88    def ensure_graph_for_app_only_auth(self):
89        if not hasattr(self, 'client_credential'):
90            client_id = self.settings['clientId']
91            tenant_id = self.settings['tenantId']
92            client_secret = self.settings['clientSecret']
93
94            self.client_credential = ClientSecretCredential(tenant_id, client_id, client_secret)
95
96        if not hasattr(self, 'app_client'):
97            self.app_client = GraphClient(credential=self.client_credential,
98                                          scopes=['https://graph.microsoft.com/.default'])
99
```

Step 5 – Use the Client and Make your Calls

5. Pass the credential object to the GraphClient constructor.

```
client = GraphClient(credential=browser_credential)
```

Literally 1 line of code

6. Make a requests to the graph API using the client

```
result = client.get('/me')
print(result.json())
```

Or set up a function (pattern) make repetitive calls as needed

```
27      def get_user(self):
28          endpoint = '/me'
29          # Only request specific properties
30          select = 'displayName,mail,userPrincipalName'
31          request_url = f'{endpoint}?$select={select}'
32
33          user_response = self.user_client.get(request_url)
34          return user_response.json()
```

DEMO from Jupyter Notebook

See some App Only Permissions as well as Visuals

Announcing xxxx

Authentication

Continuing our efforts to standardize our tools, this SDK is also generated with Kiota, an **open-source** project for generating an API client to call any OpenAPI described API, assuring quality and consistency across our tools.

```
from azure.identity.aio import ClientSecretCredential
from kiota_authentication_azure.azure_identity_authentication_provider import AzureIdentityAuthenticationProvider
credential=ClientSecretCredential(tenant_id: str, client_id: str, client_secret: str)
scopes = ['User.Read', 'Mail.Read'];
auth_provider = AzureIdentityAuthenticationProvider(credential, scopes=scopes)
```

The authentication provider handles the **fetching**, **caching**, and **refreshing of tokens** automatically, ensuring your requests are always authenticated. The auth provider **verifies the token is always valid by tracking the expiration and automatically refreshing it in the background before it expires**.

https://microsoft.github.io/kiota/

Fluent Interface

```
client = GraphServiceClient(request_adapter)

req = client.users_by_id('userPrincipalName').messages('messageId').get()
msg = asyncio.run(req)
print(msg.subject)
```

- The fluent pattern makes the request building experience more intuitive. Instead of passing raw URLs, this new version provides a fluent experience that enhances discoverability and efficiency, also reducing the time spent in reference docs.
- Contrary to other fluent experiences in Python, we don't map every endpoint and method to a unique function; instead, **we present a framework for constructing any request using method chaining**, providing a clean flow that works seamlessly with Python.
- The fluent pattern contributes to reducing errors, by **displaying only the methods corresponding to operations on that resource**, aided by the IDE's autocomplete, allowing the developer to confidently type code that runs.

https://forms.microsoft.com/r/RHJZce44Ak

European SharePoint Office 365 & Azure Conference

Whova

PLEASE RATE THIS SESSION ON THE APP

GET IT ON Google Play

Download on the App Store