6.API.v3

October 27, 2019

1 APIs do GitHub (v3)

Este notebook apresenta os seguintes tópicos:

1.1 APIs do GitHub

Como o GitHub oferece APIs para obter informações de repositórios, usá-las em geral é melhor do que fazer crawling.

O GitHub possui duas versões estáveis de APIs:

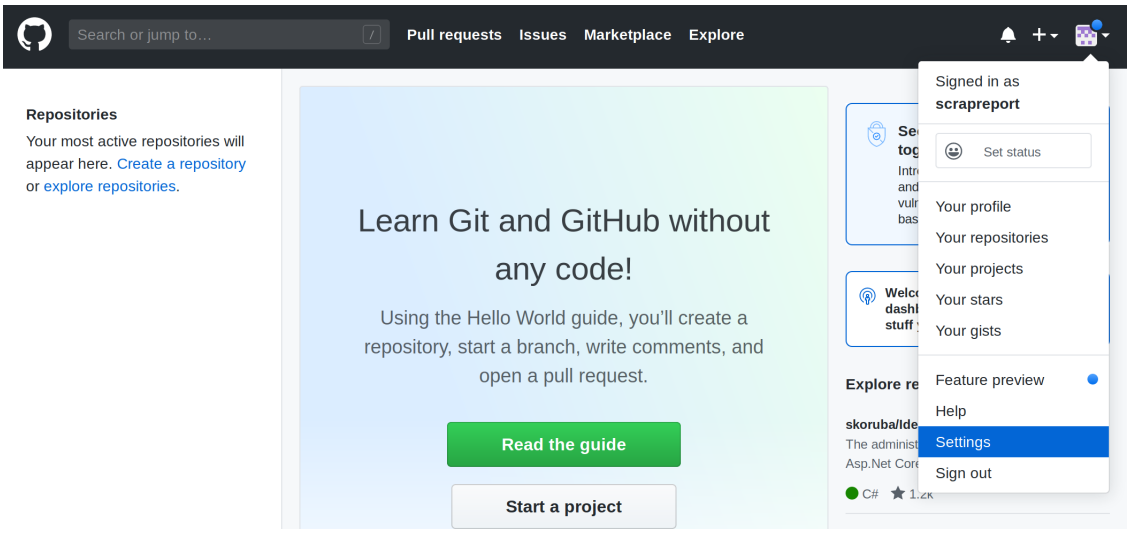
- REST API v3: https://developer.github.com/v3/
- GraphQL API v4: https://developer.github.com/v4/

A forma de usar cada API é diferente e a taxa de requisições permitidas também é. Neste minicurso, usaremos requests para acessar ambas as APIs, mas existem bibliotecas prontas (como a PyGitHub para a v3) que fazem o acesso.

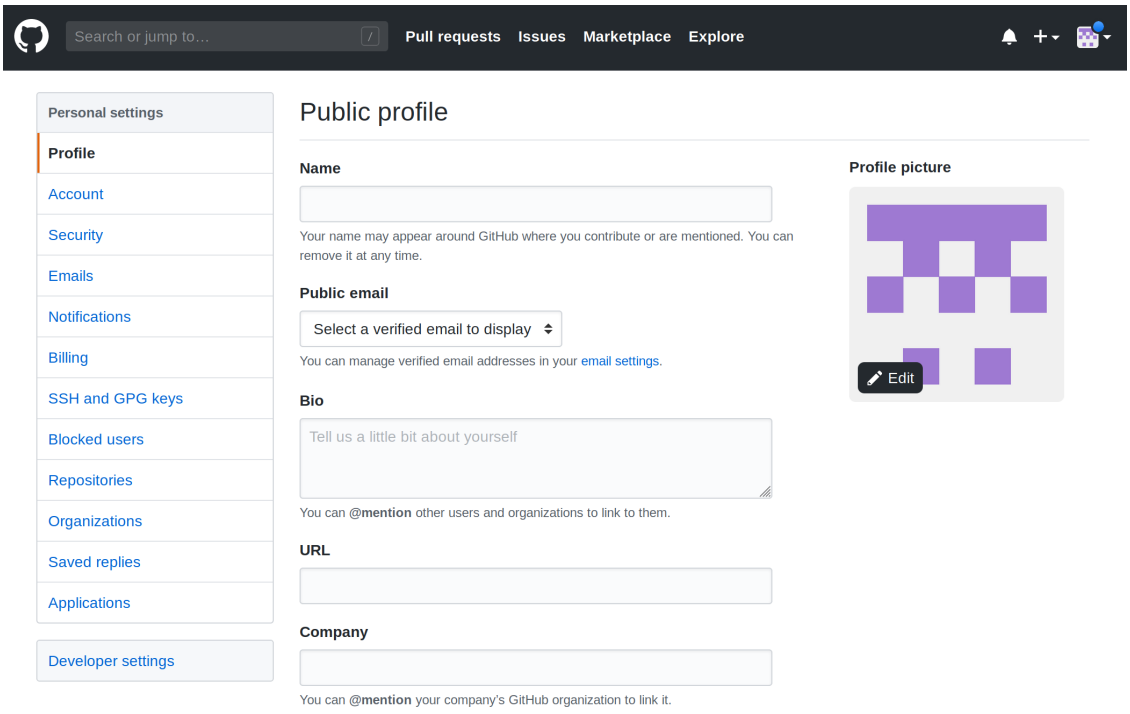
1.1.1 Autenticacao

Para usar qualquer uma das APIs, é necessário gerar um token de autenticação no GitHub seguindo os seguintes passos.

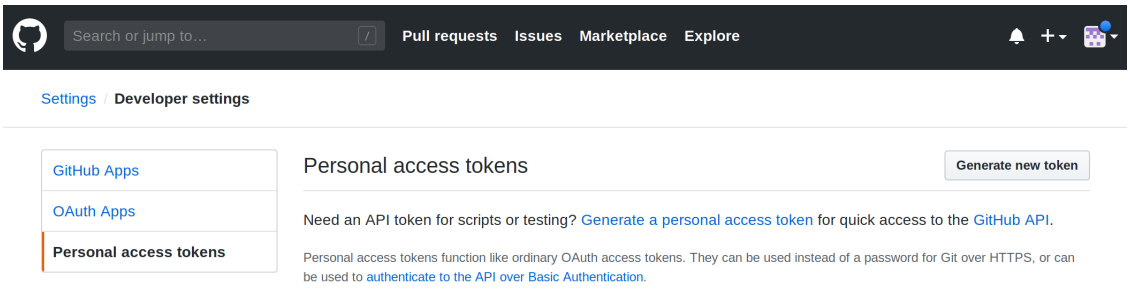
Primeiro, vá em configurações da conta.



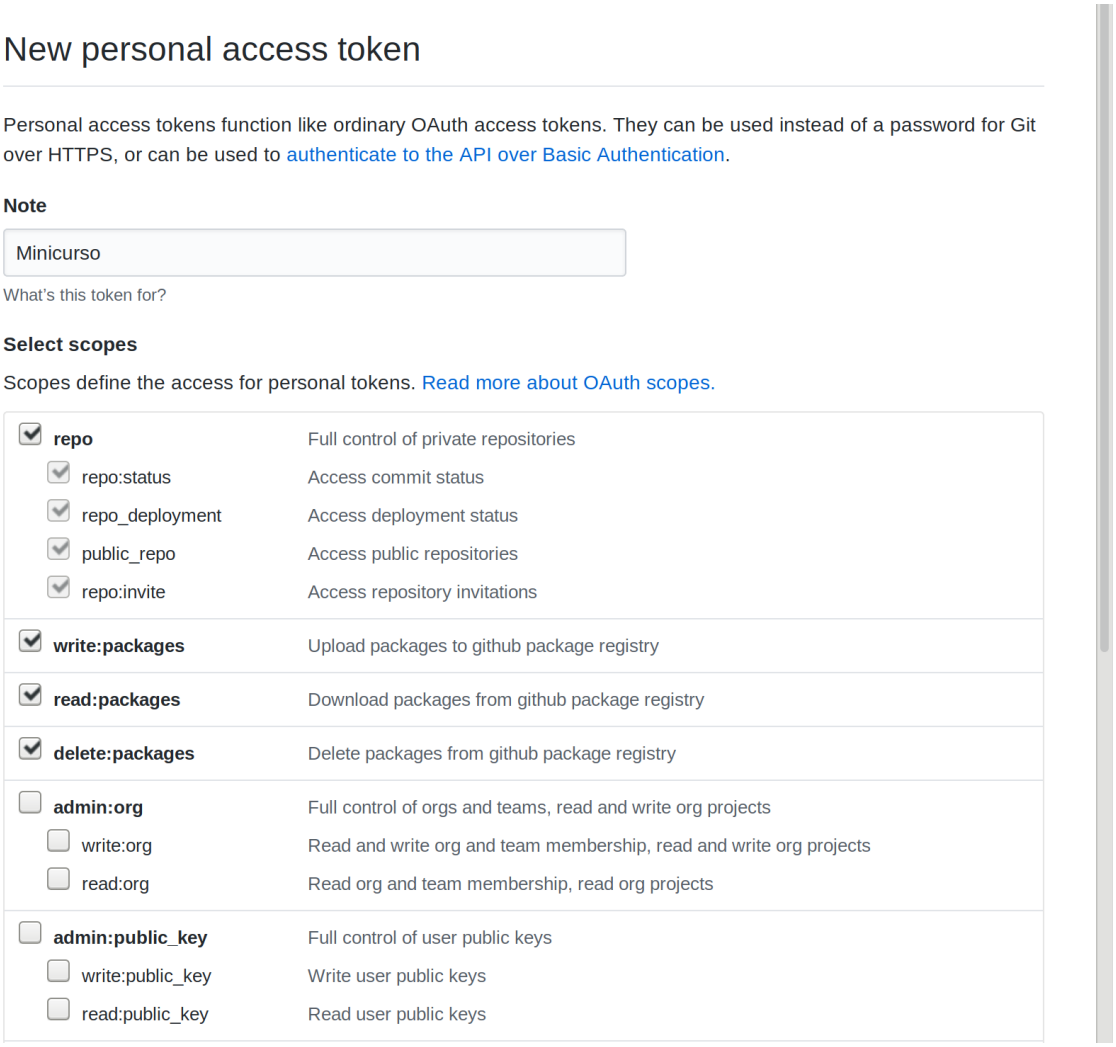
Em seguida, abra configurações de desenvolvedor.



Abra "Personal access tokens" e clique em "Generate new token".



Escolha as permissões que você deseja no token.



Copie o token gerado para algum lugar seguro. Para o minicurso, eu copiei o meu token para `~/githubtoken.txt` e vou carregá-lo para a variável `token` a seguir.

[1]:
```
from ipywidgets import FileUpload, interact
@interact(files=FileUpload())
```

```
def set_token(files={}):
    global token
    if files:
        for key, values in files.items():
            token = values['content'].decode("utf-8").strip()
            print("Token Loaded!")
```

```
interactive(children=(FileUpload(value={}, description='Upload'), Output()), _dom_classes=('wi
```

1.2 API v3

Com o token em mãos, podemos começar a usa a API v3. O acesso a API do GitHub é feito a https://api.github.com. Portanto, precisamos mudar o site de nosso servidor de proxy. Para isso, podemos fechar e reiniciar da seguinte forma:

```
python proxy.py https://api.github.com/
```

Inicialmente, vamos fazer uma requisição para verificar se a autenticação funciona e para vermos nosso limite de requisições.

[2]:
```
import requests
SITE = "http://localhost:5000/" # ou https://api.github.com
def token_auth(request):
    request.headers["User-Agent"] = "Minicurso" # Necessário
    request.headers["Authorization"] = "token {}".format(token)
    return request
response = requests.get(SITE, auth=token_auth)
response.status_code
```

[2]:
```
200
```

Resultado 200 - a autenticação funcionou.

O limite de acesso vem definido no header.

[3]:
```
response.headers["X-RateLimit-Limit"]
```

[3]:
```
'5000'
```

[4]:
```
response.headers["X-RateLimit-Remaining"]
```

[4]:
```
'4999'
```

[5]:
```
response.headers["X-RateLimit-Reset"]
```

[5]:
```
'1571985661'
```

O retorno da API v3 é sempre um JSON. O acesso a https://api.github.com retorna as URLS válidas da API.

[6]:
```
import pdffallback
result = response.json()
pdffallback.show(result)
```

```
{'authorizations_url': 'https://api.github.com/authorizations',
 'code_search_url':
'https://api.github.com/search/code?q={query}{&page,per_page,sort,order}',
 'commit_search_url':
'https://api.github.com/search/commits?q={query}{&page,per_page,sort,order}',
 'current_user_authorizations_html_url':
'https://github.com/settings/connections/applications{/client_id}',
 'current_user_repositories_url':
'https://api.github.com/user/repos{?type,page,per_page,sort}',
…
```

Vamos ver o que a API tem sobre algum repositório.

Primeiro precisamos ver qual URL usar.

[7]:
```
result['repository_url']
```

[7]:
```
'https://api.github.com/repos/{owner}/{repo}'
```

Em seguida, fazemos a requisição para saber o que tem no repositorio `gems-uff/sapos`.

[8]:
```
response = requests.get(SITE + "repos/gems-uff/sapos", auth=token_auth)
response.status_code
```

[8]:
```
200
```

[9]:
```
data = response.json()
pdffallback.show(data)
```

```
{'archive_url': 'https://api.github.com/repos/gems-
uff/sapos/{archive_format}{/ref}',
 'archived': False,
 'assignees_url': 'https://api.github.com/repos/gems-
uff/sapos/assignees{/user}',
 'blobs_url': 'https://api.github.com/repos/gems-uff/sapos/git/blobs{/sha}',
 'branches_url': 'https://api.github.com/repos/gems-
uff/sapos/branches{/branch}',
…
```

O resultado tem diversos resultados e URLs para pegar mais informações. Vamos pegar algumas informações diretas interessantes.

[10]:
```
print("Estrelas:", data["stargazers_count"])
print("Forks:", data["forks"])
print("Watchers:", data["subscribers_count"])
print("Issues abertas:", data["open_issues"])
print("Linguagem:", data["language"])
```

```
Estrelas: 18
Forks: 11
Watchers: 6
Issues abertas: 41
Linguagem: Ruby
```

Se quisermos saber quem são os colaboradores do projeto e quais são as issues existentes, podemos obter as respectivas URLs.

[11]:
```
print("Colaboradores:", data["contributors_url"])
print("Issues:", data["issues_url"])
```

```
Colaboradores: https://api.github.com/repos/gems-uff/sapos/contributors
Issues: https://api.github.com/repos/gems-uff/sapos/issues{/number}
```

Agora podemos obter a lista de colaboradores.

[12]:
```
response = requests.get(SITE + "repos/gems-uff/sapos/contributors",␣
↪auth=token_auth)
response.status_code
```

[12]:
```
200
```

[13]:
```
data = response.json()
pdffallback.show(data, convert=True)
```

```
[{'avatar_url': 'https://avatars1.githubusercontent.com/u/327789?v=4',
  'contributions': 347,
  'events_url': 'https://api.github.com/users/JoaoFelipe/events{/privacy}',
  'followers_url': 'https://api.github.com/users/JoaoFelipe/followers',
  'following_url':
'https://api.github.com/users/JoaoFelipe/following{/other_user}',
…
```

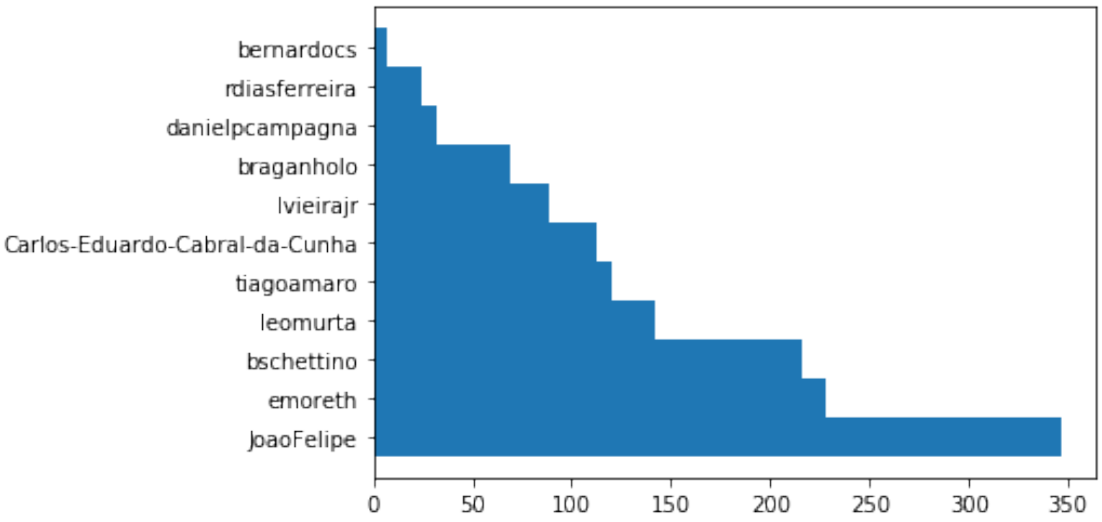
1.2.1 Gráfico de Barras

A partir desta lista, podemos fazer um gráfico de barras de contribuições.

[14]:
```
contributions = {x["login"]: x["contributions"] for x in data}
contributions
```

[14]:
```
{'JoaoFelipe': 347,
 'emoreth': 228,
 'bschettino': 216,
 'leomurta': 142,
 'tiagoamaro': 120,
 'Carlos-Eduardo-Cabral-da-Cunha': 113,
 'lvieirajr': 89,
 'braganholo': 69,
 'danielpcampagna': 32,
 'rdiasferreira': 24,
 'bernardocs': 7}
```

[15]:
```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
labels, values = zip(*contributions.items())
indexes = np.arange(len(labels))
width = 1
plt.barh(indexes, values, width)
plt.yticks(indexes, labels)
plt.show()
```



Nesse código:

- Importamos `matplotlib` e `numpy` para gerar o gráfico
- Chamamos `%matplotlib inline` para permitir a visualização da figura
- Separamos o dicionário `contributions` em duas listas de `labels` e `values`
- Criamos um `arange` de índices
- Criamos o gráfico de barras horizontal, usando `barh`
- Definimos os ticks de y como sendo os `labels` extraídos anteriormente

Podemos acessar também a url de issues.

[16]:
```
response = requests.get(SITE + "repos/gems-uff/sapos/issues", auth=token_auth)
response.status_code
```

[16]:
```
200
```

[17]:
```
data = response.json()
pdffallback.show(data, convert=True)
```

```
[{'assignee': None,
  'assignees': [],
  'author_association': 'CONTRIBUTOR',
  'body': 'Deveria ser possível emitir um relatório com as prorrogações '
          'concedidas em um determinado mês, e o SAPOS não tem essa '
…
```

[18]:
```
len(data)
```

[18]:
```
30
```

Por padrão, a API retorna 30 itens por página. Dessa forma. a lista retornou apenas a primeira página de issues.

Podemos acessar a segunda página com o parâmetro `?page=2`.

[19]:
```
response = requests.get(SITE + "repos/gems-uff/sapos/issues?page=2",␣
↪auth=token_auth)
response.status_code
```

[19]:
```
200
```

[20]:
```
data2 = response.json()
pdffallback.show(data2, convert=True)
```

```
[{'assignee': {'avatar_url':
'https://avatars2.githubusercontent.com/u/7855757?v=4',
               'events_url':
'https://api.github.com/users/danielpcampagna/events{/privacy}',
               'followers_url':
'https://api.github.com/users/danielpcampagna/followers',
               'following_url':
'https://api.github.com/users/danielpcampagna/following{/other_user}',
               'gists_url':
'https://api.github.com/users/danielpcampagna/gists{/gist_id}',
…
```

[21]:
```
len(data2)
```

[21]:
```
11
```

Podemos formar uma lista com todas as issues abertas.

[22]:
```
open_issues = data + data2
```

Essas são apenas as issues abertas. Para pegarmos as issues fechadas, precisamos definir `state=closed`. Podemos aproveitar e definir também `per_page=100` (limite máximo) e fazer um código para pegar todas as páginas.

[23]:
```
should_continue = True
page = 1
closed_issues = []
while should_continue:
    response = requests.get(SITE + "repos/gems-uff/sapos/issues?
 ↪page={}&per_page=100&state=closed".format(page), auth=token_auth)
    if response.status_code != 200:
        print("Fail:", response.status_code)
        break
    data = response.json()
    closed_issues += data
    if len(data) < 100:
        should_continue = False
    page += 1
len(closed_issues), page - 1
```

[23]:
```
(262, 3)
```

Foram encontradas 262 issues em 3 páginas.

Agora podemos fazer um gráfico que mostre a evolução de issues abertas ao longo do tempo.

Para fazer esse gráfico, primeiro precisamos combinar as issues e descobrir qual foi a data da issue mais antiga.

[24]:
```
import dateutil.parser

all_issues = open_issues + closed_issues
oldest_issue = min(
    all_issues,
    key=lambda x: dateutil.parser.parse(x["created_at"])
)

oldest_date = dateutil.parser.parse(oldest_issue["created_at"])
oldest_date
```

[24]:
```
datetime.datetime(2013, 6, 29, 15, 23, 48, tzinfo=tzutc())
```

A partir desta data, podemos criar um range de dias até hoje para ser o nosso índice do gráfico e um array de zeros do `numpy` para acumularmos a quantidade de issues abertas.

[25]:
```
from datetime import datetime, timezone
today = datetime.now(timezone.utc)
delta = today - oldest_date
days = delta.days
print(days)
indexes = np.arange(days)
values = np.zeros(days)
```

```
2310
```

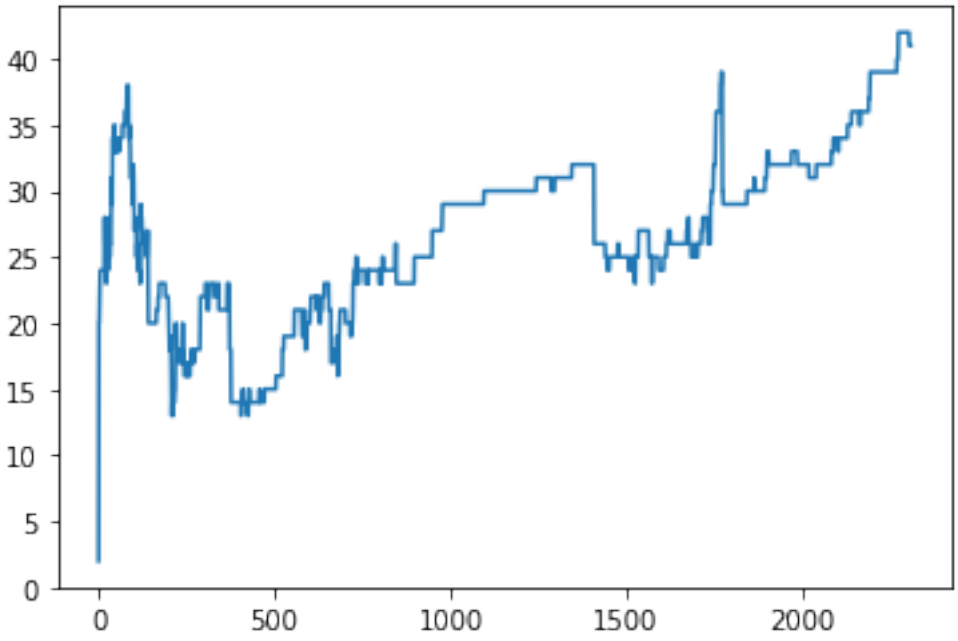
Podemos percorrer todas as issues abertas, incrementando `values` do período em que elas foram abertas até hoje. E podemos percorrer todas as issues fechadas incrementando `values` do período em que elas foram abertas até o período em que elas foram fechadas.

[26]:
```
for issue in open_issues:
    created_at = dateutil.parser.parse(issue["created_at"])
    created_at_index = (created_at - oldest_date).days
    values[created_at_index:] += 1

for issue in closed_issues:
    created_at = dateutil.parser.parse(issue["created_at"])
    created_at_index = (created_at - oldest_date).days

    closed_at = dateutil.parser.parse(issue["closed_at"])
    closed_at_index = (closed_at - oldest_date).days
    values[created_at_index:closed_at_index] += 1
```

Já é possível plotar o gráfico desta forma, mas o entendimento dos eixos ainda não é o ideal.

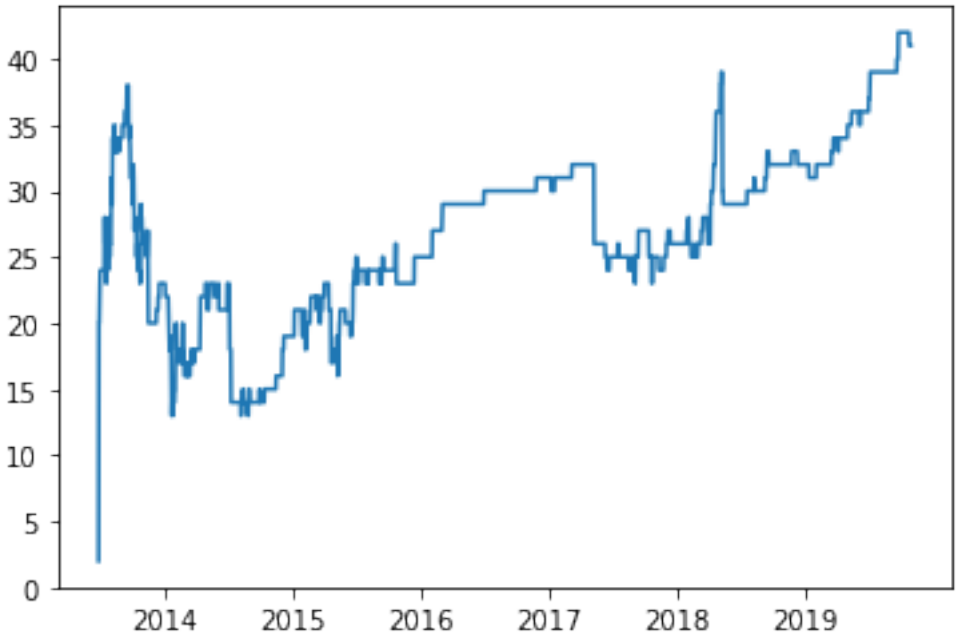
[27]:
```
plt.plot(indexes, values)
```

[27]:
```
[<matplotlib.lines.Line2D at 0x7ff520c36128>]
```



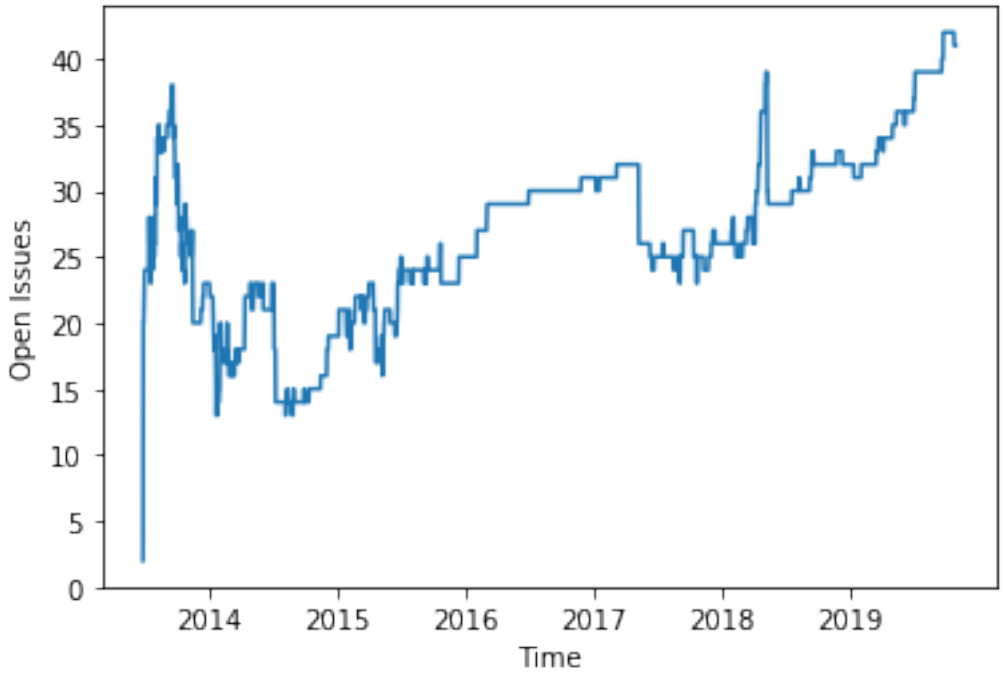
Precisamos definir quais são os anos no eixo x.

[28]:
```
from math import ceil
labels = [datetime(2013 + i, 1, 1, tzinfo=timezone.utc) for i in␣
 ↪range(ceil(delta.days / 365))]
label_indexes = [(label - oldest_date).days for label in labels]
label_years = [label.year for label in labels]
plt.xticks(label_indexes, label_years)
plt.plot(indexes, values)
plt.show()
```



Também podemos definir o que é cada eixo.

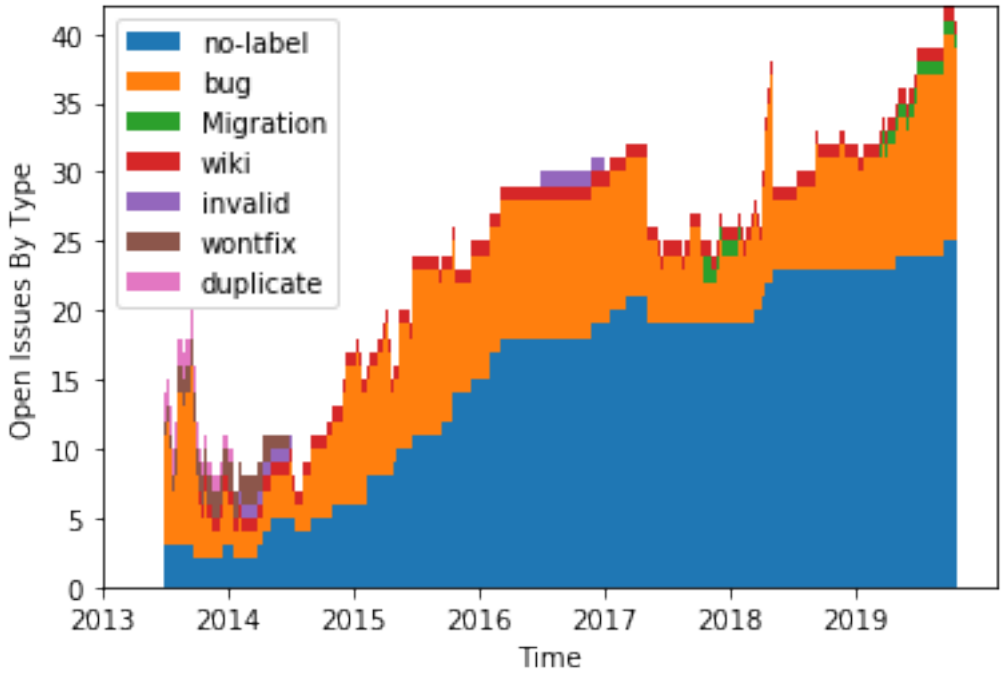
[29]:
```
plt.xticks(label_indexes, label_years)
plt.xlabel("Time")
plt.ylabel("Open Issues")
plt.plot(indexes, values)
plt.show()
```



Issues podem ter diversos labels. Agora vamos fazer um gráfico que mostre barras estacadas com a evolução de cada tipo de issue.

[30]:
```
from collections import defaultdict
values = defaultdict(lambda: np.zeros(days))

for issue in open_issues:
    created_at = dateutil.parser.parse(issue["created_at"])
    created_at_index = (created_at - oldest_date).days
    for label in issue["labels"]:
        values[label["name"]][created_at_index:] += 1
    if not issue["labels"]:
        values["no-label"][created_at_index:] += 1

for issue in closed_issues:
    created_at = dateutil.parser.parse(issue["created_at"])
    created_at_index = (created_at - oldest_date).days

    closed_at = dateutil.parser.parse(issue["closed_at"])
    closed_at_index = (closed_at - oldest_date).days
    for label in issue["labels"]:
        values[label["name"]][created_at_index:closed_at_index] += 1
    if not issue["labels"]:
        values["no-label"][created_at_index:closed_at_index] += 1
```

[31]:
```
bottom = np.zeros(days)
legend_color = []
legend_text = []
for label, yvalues in values.items():
    if not label[0].isdigit(): # Exclui tags de versões
        ax = plt.bar(indexes, yvalues, 1,
                     bottom=bottom)
        legend_color.append(ax[0])
        bottom += yvalues
        legend_text.append(label)

plt.xticks(label_indexes, label_years)
plt.xlabel("Time")
plt.ylabel("Open Issues By Type")
plt.legend(legend_color, legend_text)
plt.show()
```



1.3 Exercicio 6

Crie um gráfico de linhas que mostre apenas issues do tipo bug.

[ ]:
```
...

plt.xlabel("Time")
```

```
plt.ylabel("Open Bug Issues")
plt.show()
```

1.4 Exercicio 7

Crie um gráfico de barras para mostrar a participação de usuários em cada issue. Considere o atributo `user`.

[ ]:
```
...
```

[ ]:
```
...

plt.xlabel("Time")
plt.ylabel("Open Issues By User")
plt.legend(
    legend_color, legend_text,
    bbox_to_anchor=(0,1.02,1,0.2), loc="lower left",
    mode="expand", borderaxespad=0, ncol=2
)
plt.show()
```

1.5 Exercicio 8

Filtre o gráfico do total de issues abertas para mostrar apenas o ano 2014.

[ ]:
```
yfirst = datetime(2014, 1, 1, tzinfo=timezone.utc)
ylast = datetime(2015, 1, 1, tzinfo=timezone.utc)

deltadays = (ylast - yfirst).days
values = np.zeros(deltadays)
indexes = np.arange(deltadays)

...
```

[ ]:
```
labels = [datetime(2014, i + 1, 1, tzinfo=timezone.utc) for i in range(12)]
label_indexes = [(label - yfirst).days for label in labels]
label_years = [label.month for label in labels]
plt.xticks(label_indexes, label_years)
plt.plot(indexes, values)
plt.show()
```

Continua: 7.API.v4.pdf