

PYTHON

BIBLIOTECAS EXTERNAS E SISTEMA OPERACIONAL

HUMBERTO D. DE SOUSA



6

LISTA DE FIGURAS

Figura 6.1 – Arquitetura para aplicação Wazeyes.....	7
Figura 6.2 – Baixando Geopy.....	12
Figura 6.3 – Instalando o Geopy	13
Figura 6.4 – Instalação com sucesso do Geopy.....	13
Figura 6.5 – Arquivo JSON – entrada.json	14
Figura 6.6 – Alterando a execução do projeto.....	22
Figura 6.7 – Emular o console.....	22



LISTA DE CÓDIGOS-FONTE

Código-fonte 6.1 – Converter_geolocalizacao.....	10
Código-fonte 6.2 – Geolocalização com o JSON	15
Código-fonte 6.3 – Explorando o Geopy	16
Código-fonte 6.4 – Geolocalizacao reversa	17
Código-fonte 6.5 – Recuperando Informações.....	19
Código-fonte 6.6 – Usuário e Data	20
Código-fonte 6.7 – Capturando usuário e senha.....	21
Código-fonte 6.8 – Capturando usuário com senha oculta	21

SUMÁRIO

6 BIBLIOTECAS EXTERNAS E SISTEMA OPERACIONAL.....	5
6.1 Utilizando pacotes externos	9
6.1.1 Apresentamos... Geo Py – Geopy.....	10
6.1.2 Instalando Geopy	12
6.1.3 Usando Geopy	14
6.1.4 Um pouco mais sobre o Geopy	16
6.2 Conversando com o sistema operacional.....	19
REFERÊNCIAS.....	24

6 BIBLIOTECAS EXTERNAS E SISTEMA OPERACIONAL

Nosso código agora irá alcançar voos mais altos e mais distantes. Chegou a hora de apresentar o mundo exterior ao Python. Muitos recursos virão e as possibilidades de aplicação aumentarão exponencialmente. Prontos para receberem os kits de superpoderes? Vamos lá...

6.1 Utilizando pacotes externos

Seu código, até agora, teve contato apenas com recursos internos e padrões do Python, que já existiam dentro da PVM que reina em seu computador. Comandos e funções que compõem a linguagem, como: `if`, `while`, `for`, `open()`, `split()`, `len()`, `upper()`, entre tantos outros que já abordamos, nós aprendemos a usá-los sem fazer qualquer referência, isso porque esses comandos/funções/recursos já estão liberados para utilização.

Também abordamos referências que já existem no Python, mas que podem estar incluídas na sua PVM (algumas dependem dos recursos instalados em seu computador), como foram os casos dos imports utilizados para “`json`” (que possui as funções para a manipulação de arquivos JSON) e “`os`” (que tem funções que permitem recuperar algumas informações do sistema operacional, como, por exemplo, o caminho do diretório da sua aplicação que estiver em execução).

Entretanto, podemos encontrar muitos recursos que não estão incorporados na linguagem e na PVM, ou seja, necessitam ser instalados para que possam ser reconhecidos, para esses recursos, atribuímos o termo “pacotes externos”. Perceba que não é algo completamente novo, pois você já desenvolveu um pacote de funções, que, se for trabalhado para se tornar portátil para outras aplicações, poderá vir a ser um “pacote externo”, no qual qualquer programador poderá instalar o seu pacote e, assim, utilizar as funções que você previamente programou. Por isso, quando você importa um pacote externo, significa diretamente que você está utilizando um código que alguém, ou alguma empresa, criou e compartilhou para que outros programadores pudessem produzir códigos mais simples e de maneira mais rápida.

Obviamente, seria impossível abordar todos os pacotes externos existentes para Python. Faremos juntos um exemplo com um pacote externo, assim, quando precisar utilizar qualquer pacote externo, o processo de instalação e importação será o mesmo. Por isso, segue uma dica:

Sempre que precisar programar algo que julgue complexo, procure pesquisar se já não existe algo pronto, o que irá economizar muito tempo de programação. Por outro lado, cuidado com a qualidade do que você importa para o seu projeto, isso pode prejudicar o desempenho da sua aplicação.

6.1.1 Apresentamos... Geo Py – Geopy

Como foi abordado anteriormente, sempre que se deparar com a necessidade de desenvolver algo, procure verificar o que já existe pronto, que possa ajudá-lo. Você está na seguinte situação: um grupo de pesquisa, patrocinado pela BitMed, voltado para o desenvolvimento de “wearables” aplicados ao setor da saúde, está desenvolvendo um aplicativo, denominado “Wazeyes”, que irá trocar coordenadas geográficas com um wearable a fim de orientar pessoas com necessidades especiais visuais. O paciente irá digitar o endereço e o aplicativo deverá convertê-lo em coordenadas geográficas (latitude e longitude), para que possa marcar a posição no mapa (GoogleMaps ou OpenStreepMap, por exemplo) e assim orientar, por voz, o deslocamento da pessoa portadora de necessidade especial visual.

Coube a você desenvolver o código (ficará no servidor) que será responsável por capturar o endereço que estará dentro de um arquivo JSON (gerado pelo aplicativo instalado no dispositivo mobile do portador de necessidade especial visual) e, então, passar as coordenadas para outro arquivo JSON que será consumido por outra aplicação que manipula mapas e desenvolve rotas. Estamos afirmando, portanto, uma solução que terá basicamente a seguinte arquitetura:



Figura 6.1 – Arquitetura para aplicação Wazeyes
Fonte: Elaborado pelo autor (2017)

Resumidamente, as etapas 1, 2 e 3 significam:

- No aplicativo mobile, estará a etapa 1, que terá a responsabilidade de receber o endereço, por voz ou texto, e gerar um arquivo JSON no servidor, com o endereço que foi digitado ou falado.
- Na etapa 2, que rodará no servidor, SUA aplicação Python irá realizar a leitura do arquivo JSON (entrada.json), que armazena o endereço e, então, deverá escrever em outro arquivo JSON (saida.json) as coordenadas geográficas referentes ao endereço.
- Na etapa 3, outra aplicação irá consumir o arquivo “saida.json”, dentro de determinado intervalo de tempo, e, então, irá gerar uma rota que será exibida ao portador da necessidade especial, somente quando ele ativar o botão “Iniciar o trajeto”.

Com base na descrição das três etapas básicas, você ficou responsável por auxiliar na segunda etapa. E então? Será que é difícil montar uma rotina em Python, para **ler** um arquivo JSON, **converter** o endereço obtido em coordenada geográfica e **escrever** as coordenadas obtidas em um outro arquivo JSON? Vamos avaliar os verbos negritados! Ler e escrever (arquivos JSON) nós já fizemos, logo, não será o

nosso maior desafio. Converter o endereço para as coordenadas parece ser o mais difícil, não é mesmo?

Pois é, você está enganado, a não ser que queira criar tudo do zero. Apresentamos para você o pacote externo “Geopy”, que possui, entre outras, a função de converter um texto em coordenadas geográficas, assim iremos conseguir ajudar o departamento de inovação tecnológica de maneira simples e rápida. Vamos colocar esse “superpoder” em nossa aplicação.

6.1.2 Instalando Geopy

Como já dissemos anteriormente, o pacote/módulo Geopy é externo, por isso, precisaremos baixá-lo do link: <https://pypi.org/project/geopy/>, conforme a figura a seguir (Salve-o em um local de fácil acesso, por exemplo: [c:\pacotes](#)):

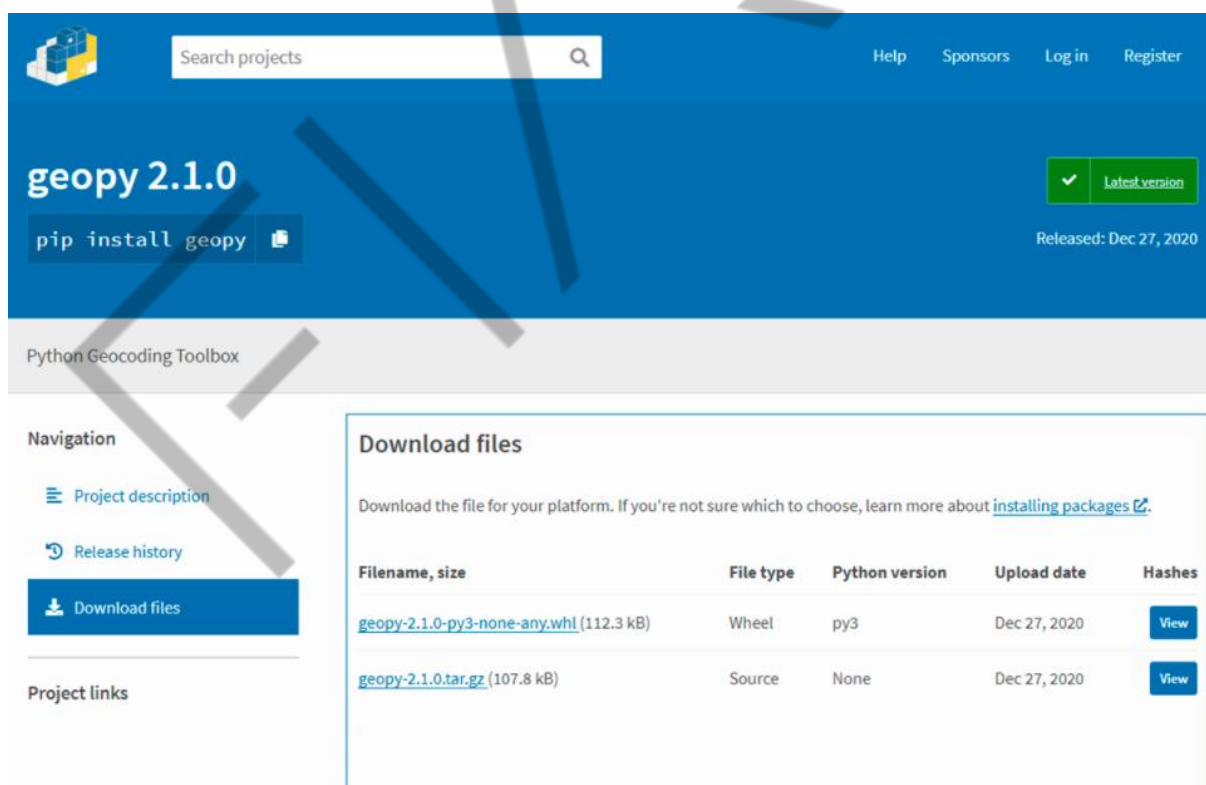


Figura 6.2 – Baixando Geopy
Fonte: Elaborado pelo autor (2020)

Agora, abra o console do seu sistema operacional e direcione até a pasta criada. No Windows, deverá seguir: <Windows>+<R> / cmd <enter> e, na tela de

console, usar o comando “cd” (change directory) para se deslocar entre diretórios, conforme está sublinhado em vermelho na figura seguinte (para quem baixou o arquivo exatamente em c:\pacotes):

```
Microsoft Windows [versão 10.0.18362.1082]
(c) 2019 Microsoft Corporation. Todos os direitos reservados.

(venv) C:\Users\andre\Desktop\Funções>pip install geopy
```

Figura 6.3 – Instalando o Geopy
Fonte: Elaborado pelo autor (2020)

O resultado deverá ser o seguinte:

```
(venv) C:\Users\andre\Desktop\Funções>pip install geopy
Collecting geopy
  Using cached geopy-2.0.0-py3-none-any.whl (111 kB)
Requirement already satisfied: geographiclib<2,>=1.49 in c:\users\andre\desktop\funções\venv\lib\site-packages (from geopy) (1.50)
Installing collected packages: geopy
Successfully installed geopy-2.0.0
```

Figura 6.4 – Instalação com sucesso do Geopy
Fonte: Elaborado pelo autor (2020)

Caso apareça uma mensagem de erro do tipo “não é reconhecido como comando”, significa que você não tem o comando “pip” instalado/configurado. Esse é o comando responsável por instalar pacotes externos no seu Python. Você deve utilizar:

- **pip install** <nomeDoPacote> => para instalar ou
- **pip uninstall** <nomeDoPacote> => para desinstalar o pacote especificado.

Se você precisar instalá-lo, digite no console o seguinte comando:

- **python -m ensurepip --upgrade**

Caso o comando “python” não funcione, é porque você se esqueceu de marcar a opção “Add to path” no momento da instalação do Python. Recomendamos que volte para o material no qual explicamos como instalar o Python e refaça essa instalação de acordo com as instruções do material.

Depois, volte a digitar:

- **pip install geopy**

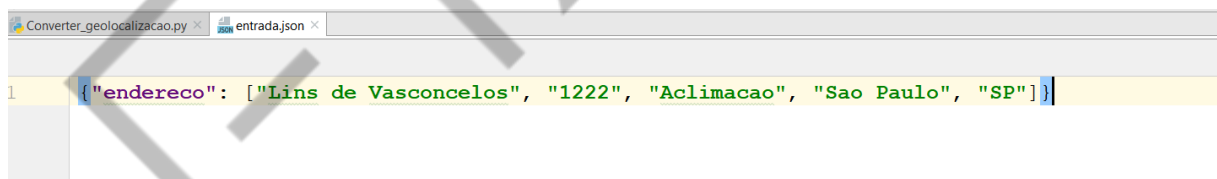
6.1.3 Usando Geopy

Agora que o pacote está devidamente instalado, vamos montar o nosso código. Para isso, abra o nosso projeto no PyCharm, crie um novo “Python Package”, chamado: **“5_1_Wazeyes”**, dentro dele, crie um arquivo “Python File”, chamado **“Converter_geolocalizacao.py”** e digite o seguinte código:

```
from geopy.geocoders import Nominatim
geolocator = Nominatim(user_agent="wazeyes") # Nome do seu aplicativo
location = geolocator.geocode("175 5th Avenue NYC")
print(location.address)
print((location.latitude, location.longitude))
```

Código-fonte 6.1 – Converter_geolocalizacao
Fonte: Elaborado pelo autor (2020)

Com as linhas, já conseguimos a parte do código que deveria ser a mais difícil, isto é, capturamos uma string que representa um endereço e convertemos para as coordenadas geográficas latitude e longitude, para que possam ser aplicadas, por outra aplicação, sobre um mapa. Agora, a missão é criar um arquivo JSON com um endereço, consumi-lo, gerar as coordenadas e gravar em outro arquivo JSON. Nesse momento, antes de seguir a leitura da sequência do código a seguir, aproveite para treinar, crie o arquivo JSON (conforme a figura seguinte) e produza o código que irá ler o arquivo do endereço e gerar outro arquivo com as coordenadas, vamos lá!



```
{ "endereco": ["Lins de Vasconcelos", "1222", "Aclimacao", "Sao Paulo", "SP"] }
```

Figura 6.5 – Arquivo JSON – entrada.json
Fonte: Elaborado pelo autor (2017)

Pronto! Concluiu a programação? Seu código deve estar semelhante às linhas de código abaixo:

```
from geopy.geocoders import Nominatim
from Funcoes.Funcoes_JSON import ler_arquivo, gravar_arquivo

geolocator = Nominatim(user_agent="wazeyes")
dicionario=ler_arquivo("entrada.json")
lista=dicionario["endereco"]
endereco=lista[0] + "," + lista[1] + " " + lista[2] + " " + lista[3]
location = geolocator.geocode(endereco)
saida={"coordenadas": (location.latitude, location.longitude)}
gravar_arquivo(saida,"saida.json")
```

Código-fonte 6.2 – Geolocalização com o JSON
Fonte: Elaborado pelo autor (2020)

E. então, havia pensando em utilizar as funções (`ler_arquivo()` e `gravar_arquivo()`) que criamos anteriormente no arquivo `Funcoes_JSON`? Espero que sim... Essa é a mágica em criarmos módulos flexíveis a ponto de serem reaproveitados em qualquer outro projeto, ou ao menos na maioria dos projetos.

Detalhando um pouco mais o código apresentado: temos duas importações (`Geopy` e do arquivo no qual encontramos as funções que desejamos e que já havíamos criado), capturamos o conteúdo do `ler_arquivo()` em um dicionário, recuperamos somente a lista, com base na lista, montamos o conteúdo da variável “endereço” por meio da concatenação dos itens da lista e obtivemos uma variável do tipo `string`, criamos um dicionário chamado “saida” para armazenar a chave “coordenadas” e, dentro dela, o resultado do método `geocode()` sobre a nossa variável “endereço”, ou seja, as coordenadas e, por fim, gravamos o arquivo “saida.json” com o dicionário “saida”. Um show de conceitos, não é mesmo?

Praticamente, em cada linha, temos um conceito e uma estrutura diferentes. Por isso, é muito importante que você conheça bem as estruturas de dados já abordadas. Se ainda você se sentir desconfortável em utilizar qualquer uma das estruturas, não deixe para depois. Pratique mais (não é decorar – é praticar), refaça exercícios, procure aplicações no seu cotidiano, mesmo que sejam bem simples e que você nem chegue a implementá-las posteriormente, mas não deixe de praticar até dominar as estruturas básicas que o Python disponibiliza.

Dica: o `Geopy` é um pacote que pode se comunicar com o Google Maps, ou seja, um serviço pago do Google. Por isso, não estranhe caso apareça o erro “**OVER_QUERY_LIMIT**” ao executar seguidas vezes o seu código. Para que ele funcione seguidas vezes, deveria comprar o serviço de geolocalização do Google, assim, teria uma chave que permitiria obter recursos adicionais. Para fins acadêmicos, isso não se faz necessário, é só mudar o endereço que deseja pesquisar e executar novamente o código.

6.1.4 Um pouco mais sobre o Geopy

Podemos, ainda, utilizar o Geopy para obter mais informações com base em um endereço fornecido. Crie um arquivo chamado “Geolocalizacao_plus.py” e digite o seguinte código:

```
from geopy.geocoders import Nominatim
geolocator = Nominatim(user_agent="wazeyes")

endereco=input("Digite um endereço com número e cidade. "
               "\nExemplo: avenida paulista, 100 São Paulo:\n")
resultado = str(geolocator.geocode(endereco)).split(",")

if resultado[0]!='None':
    print("Endereço completo.: ", resultado)
    print("Bairro.....: ", resultado[1])
    print("Cidade.....: ", resultado[2])
    print("Regiao.....: ", resultado[3])
```

Código-fonte 6.3 – Explorando o Geopy

Fonte: Elaborado pelo autor (2020)

Vamos analisar esse código:

- Importamos o nosso pacote Geopy e, em seguida, solicitamos ao nosso usuário o endereço que ele deseja pesquisar.
- Atribuímos, para a nossa lista “resultado”, o resultado obtido pela execução do método geocode() junto ao endereço digitado pelo usuário. O que esse método faz exatamente? Ele pesquisa a string recebida (no nosso caso, o conteúdo da variável “endereco”) dentro do GoogleMaps e retorna em forma de string, com vários elementos separados por “vírgula”. Por isso, dividimos a nossa string em elementos de uma lista a cada vírgula encontrada na string.
- Verificamos se o endereço é um endereço válido por meio da primeira posição da lista, que se for diferente de “None”, significa que foi encontrado algum endereço..
- Se o endereço for válido, então, mostraremos o endereço completo, ou seja, mesmo que você tenha digitado “Avenida Paulista, 10”, você verá CEP, bairro, cidade e país. Logo após, mostrará apenas o bairro do endereço localizado, seguido da cidade e da região, somente a fim de demonstrar

como podemos extrair apenas uma informação específica do endereço, caso seja necessário.

Faça algumas experiências, experimente digitar somente um CEP, um endereço menos completo, outro mais completo e aprenda com o comportamento do pacote “Geopy”. Podemos também fazer o inverso, monte um novo arquivo, chamado “Geolocalizacao_reversa.py”, com o seguinte código:

```
from geopy.geocoders import Nominatim
geolocator = Nominatim(user_agent="wazeyes")

latitude=float(input("Digite a latitude...: "))
longitude=float(input("Digite a longitude.: "))

resultado = str(geolocator.reverse(f"{latitude}, {longitude}")).split(",")
if resultado[0]!='None':
    print("Endereço completo.: ", resultado)
    print("Dado 1.....: ", resultado[0])
    print("Dado 2.....: ", resultado[1])
    print("Dado 3.....: ", resultado[2])
```

Código-fonte 6.4 – Geolocalizacao reversa
Fonte: Elaborado pelo autor (2020)

Perceba, agora, que não estamos solicitando o endereço e, sim, a latitude e longitude. O método utilizado na quarta linha não é mais o geocode() e, sim, o reverse (), que recebe dois valores: latitude e longitude, respectivamente, e retorna o endereço encontrado. Utilize como teste as coordenadas:

- Latitude: -23.5740406
- Longitude: -46.623408900000015

Logo obterá o endereço completo, e mais três dados exibidos separadamente.

Observe que, por meio de uma necessidade real, na aplicação “Wazeyes”, tivemos a oportunidade de aprender muito sobre geolocalização e, melhor que isso, percebemos que muitos códigos estão prontos, é só conhecer, pesquisar sobre o pacote externo e aplicar os conhecimentos adquiridos sobre os conceitos apresentados anteriormente. O que isso tem a ver com Defesa Cibernética? Se ainda não caiu a ficha, vou lhe dar uma ajudazinha...

O próprio “Wazeyes” estará na forma de aplicativo e se comunicando com a internet... seguro, isso? Você sabe que não! Caberá a você observar todo o trâmite dos dados e as trocas dos pacotes para que possa garantir que o público-alvo não

seja vítima de ciberataques. E, se não quiser falar de aplicativos para a saúde, já pensou em um aplicativo bancário que, por meio das coordenadas geográficas, pode auxiliar a classificação quanto à veracidade de uma transação bancária? E para todas as IoTs que temos no mercado, consegue perceber a gama de aplicações?

Todos os dias úteis, você termina realizando um caminho, um tanto quanto parecido (exceção a alguns cargos e/ou profissões que exigem viagens mundo afora). Poderíamos delimitar uma faixa nesse mapa, como se fosse uma cerca mesmo, e, caso saíssemos dessa faixa, o aplicativo poderia interpretar como um momento de perigo, um sequestro-relâmpago, por exemplo. E, então, poderia avisar alguns contatos predefinidos, por SMS ou qualquer outra aplicação para a troca de mensagens.

Enfim, são apenas ideias relacionadas à geolocalização. Imagine quantos pacotes externos existem e quanto espaço você tem para criar soluções, corrigir e propor inovações. E, quanto mais “poderes” você for adquirindo, mais “\$uper-herói” você será... E então? Grande o desafio, não é mesmo? Estamos juntos nessa! Vamos avançar para adquirir mais poderes ainda... Que venha o sistema operacional!

6.2 Conversando com o sistema operacional

Iremos abordar, agora, o sistema operacional e como ele pode ser um forte aliado para os desenvolvedores de Python. Como o sistema operacional controla o equipamento, ele é alvo preferido dos cibercriminosos e também o responsável por nos fornecer informações muito importantes sobre o seu comportamento.

Apresentaremos, agora, um primeiro contato e você verá que é bem simples manipular algumas informações do sistema operacional. Crie um novo “Python Package”, chamado “5_2_SistemaOperacional”, e, nele, um novo “Python File”, denominado “RecuperandoDados.py”, em seguida, digite o código a seguir:

```
import platform

print("Distribuição do Sistema Operacional.: ", platform.platform())
print("Nome do Sistema Operacional.....: ", platform.system())
print("Versão do Sistema Operacional.....: ", platform.release())
print("Arquitetura.....: ", platform.architecture())
print("Nome do Computador.....: ", platform.node())
print("Tipo de Máquina.....: ", platform.machine())
```

```
print("Processador.....: ", platform.processor())  
print("Versão do Python.....: ", platform.python_version())
```

Código-fonte 6.5 – Recuperando Informações

Fonte: Elaborado pelo autor (2017)

Esse código aponta que precisamos importar o pacote “platform” para obtermos informações como:

- `platform()`: método que retorna a distribuição exata do sistema operacional que está sendo executado. Muito útil para que possamos desenvolver ferramentas de inventário ou ferramentas que podem servir para apontar computadores que não estão atualizados.
- `system()`: este método retorna simplesmente o sistema operacional que está sendo executado. Isso pode auxiliar sua ferramenta a executar, por exemplo, comandos de Windows no Windows, e comandos do Linux no Linux, tornando-a multiplataforma.
- `release()`: retorna a versão do sistema operacional que está sendo executada.
- `architecture()`: exibe a arquitetura que está em uso pela máquina.
- `node()`: retorna o nome do computador na rede. Mais uma vez, essa é uma informação muito importante para o inventário. Lembra quando nós fizemos uma ferramenta em que o usuário deveria digitar? Pois é, podemos deixar nossas ferramentas mais independentes do usuário e, conseqüentemente, com maior padronização e fidelidade dos dados recuperados.
- `machine()`: retorna o tipo da máquina, e é muito utilizada com o método `processor()`, que retorna o processador que está sendo executado, o que pode ser muito útil, por exemplo, para providenciarmos atualizações sobre falhas de segurança para processadores de um fabricante em específico. Por exemplo, no ano de 2018, tivemos a divulgação das falhas de segurança: Meltdown e Spectre, representando vulnerabilidades que podem ser exploradas de acordo com os fabricantes de processadores, atingindo os maiores players de mercado como: Intel, Apple e AMD. A identificação do processador pode lhe permitir o direcionamento de patches de atualização para cada computador que faz parte da sua rede corporativa.

- `python_version()`: irá retornar a versão do Python que está sendo executada. Esse método também pode ser muito importante para garantir a plena execução da sua ferramenta, uma vez que existe uma grande diferença entre o Python versão 2.x e o 3.x, além de diferenças menores entre as versões intermediárias.

Temos, também, alguns outros pacotes que podem ser úteis para ações comuns, conforme podemos observar no código a seguir, que deve ser digitado no novo arquivo chamado “Usuario_data.py”:

```
import getpass
from datetime import datetime

print("Usuário.....: ", getpass.getuser())
print("Data Completa.: ", datetime.now())
print("Dia.....: ", datetime.now().day)
print("Mês.....: ", datetime.now().month)
print("Ano.....: ", datetime.now().year)
print("Hora.....: ", datetime.now().hour)
print("Minuto.....: ", datetime.now().minute)
print("Segundo.....: ", datetime.now().second)
```

Código-fonte 6.6 – Usuário e Data
Fonte: Elaborado pelo autor (2017)

Podemos observar que o pacote “datetime” irá nos permitir recuperar a data completa de acordo com o sistema operacional, assim como: dia, mês, ano, hora, minuto e segundo. Essa sequência também poderá nos ajudar, por exemplo, na montagem de um inventário. Já o pacote “getpass” nos propicia retornar o usuário logado no momento, o que pode nos auxiliar, por exemplo, no desenvolvimento de uma ferramenta de auditoria, em que saberemos as ações que determinado usuário realizou, assim como estabelecer se o usuário está em um dia/horário que deve possuir ou não acesso. Se a clínica não abre aos domingos, poderemos bloquear todos os usuários, exceção ao administrador, o acesso ao sistema, máquina ou qualquer outro item que faça parte da rede, que desejarmos.

Mas o “getpass” também nos proporciona ter acesso a outro recurso bem interessante. Observe o código seguinte e o considere como parte de uma ferramenta:

```
usuario=input("Digite o usuário: ").upper()
senha= input("Digite a senha: ")

if usuario == "BITMED" and senha == "FiAp1222":
    print("Usuário logado")
else:
    print("Login Negado")
```


Código-fonte 6.7 – Capturando usuário e senha
Fonte: Elaborado pelo autor (2017)

Esse código é bem simples, estamos somente capturando um usuário e senha e verificando se é o usuário (“BITMED”) com a senha válida (“FiAp1222”). Somente se usuário e senha coincidirem, o usuário terá acesso e será logado. Mas observe que, ao digitar a senha, essa será apresentada na tela enquanto o usuário digitar, ou seja, nada seguro, não é mesmo? É aí que entra a outra função do “getpass”. Veja, agora, o código alterado (digite-o em um novo arquivo chamado “Login.py”):

```
import getpass

usuario = input("\nDigite o usuário: ").upper()
senha = getpass.getpass("Digite a senha: ")

if usuario == "BITMED" and senha == "FiAp1222":
    print("Usuário logado")
else:
    print("Login Negado")
```

Código-fonte 6.8 – Capturando usuário com senha oculta
Fonte: Elaborado pelo autor (2017)

Caso você execute esse código dentro do PyCharm, verá que o comportamento do sistema não está adequado (dependendo do seu sistema operacional). Você digita o usuário, ele muda de linha e permanece solicitando o usuário, até que você encerre o código manualmente. Isso ocorrerá porque a IDE (ou até mesmo o ambiente IDLE do Python) não consegue utilizar o método getpass() para a emissão de caracteres. Somente quando o código for executado fora da IDE é que ele irá funcionar perfeitamente. Logo, teremos duas opções: a primeira seria você ir até o console do seu sistema operacional, se deslocar até a pasta do seu arquivo e, então, compilar o código-fonte; a outra, mais simples, é que podemos simular a execução no console dentro do Python, para isso, observe a seguinte figura:

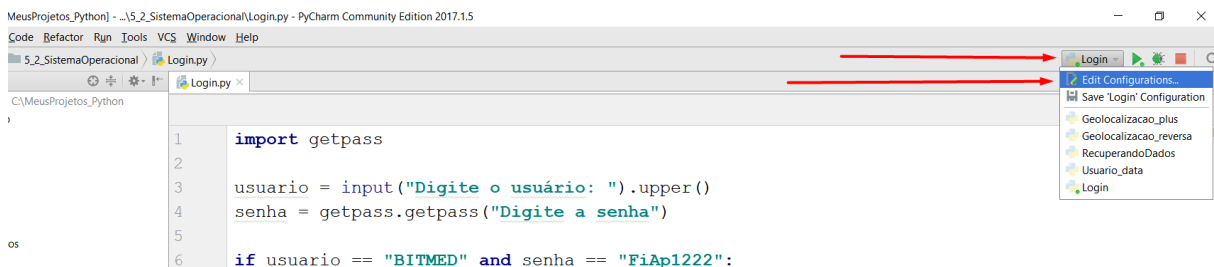


Figura 6.6 – Alterando a execução do projeto
Fonte: Elaborado pelo autor (2017)

De acordo com a imagem, clique sobre o *dropbox*, no qual aparece o nome do seu arquivo (Login), em seguida, clique sobre a opção “Edit Configurations”, surgirá, então, a seguinte caixa de diálogo, marque a opção que está destacada:

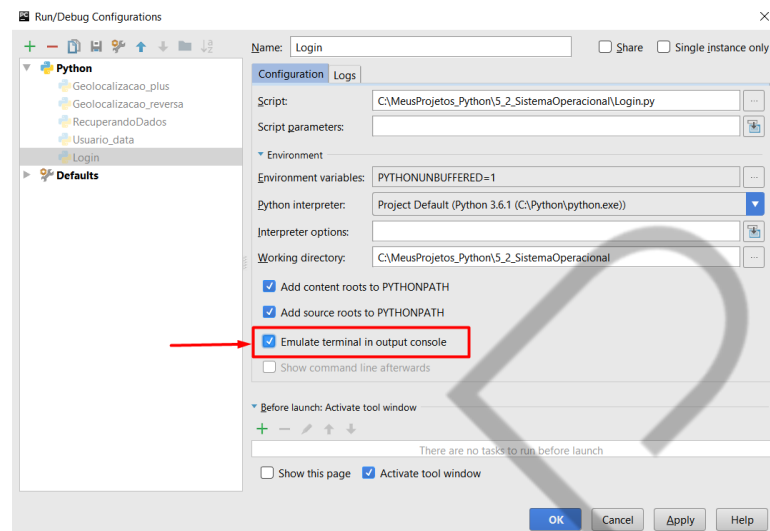


Figura 6.7 – Emular o console
Fonte: Elaborado pelo autor (2017)

Clique sobre o botão “OK” e, em seguida, execute o seu código. Você verá, na parte inferior, que o seu código será executado como se estivesse sendo compilado no console do seu sistema operacional. Agora, no momento da digitação, ele não só oculta os caracteres, mas também a quantidade de caracteres que estão sendo digitados na senha.

REFERÊNCIAS

FORBELLONE, André Luiz Villar. **Lógica de programação**. 3. ed. São Paulo: Prentice Hall, 2005.

JET BRAINS. **PyCharm Community, version 2017.2.4**: IDE para programação. Disponível em: <<https://www.jetbrains.com/pycharm/download/#section=windows>>. Último acesso: nov. 2017.

KUROSE, James F. **Redes de computadores e a Internet**: uma abordagem top-down. 6. ed. São Paulo: Pearson Education do Brasil, 2013.

MAXIMIANO, Antonio Cesar Amaru. **Empreendedorismo**. São Paulo: Pearson Prentice Hall Brasil, 2012.

PIVA, Dilermando Jr. **Algoritmos e programação de computadores**. São Paulo: Elsevier, 2012.

PUGA, Sandra. **Lógica de programação e estruturas de dados**. São Paulo: Prentice Hall, 2003.

RAMEZ, Elmasri; SHAMKANT B. Navathe. **Sistemas de banco de dados**. 6. ed. São Paulo: Pearson Addison Wesley, 2011.

RHODES, Brandon. **Programação de redes com Python**. São Paulo: Novatec, 2015.

STALLINGS, W. **Arquitetura e organização de computadores**. 8. ed. São Paulo: Prentice Hall Brasil, 2010.