Introdução à Programação com Python – Fundamentos para Pentesters

Objetivo da Sala (TryHackMe)

Esta sala tem como propósito ensinar os **fundamentos da linguagem de programação Python**, com foco prático para segurança ofensiva. Embora programar não seja estritamente necessário para atuar em segurança da informação, é uma habilidade valiosa que permite:

- Criar **ferramentas personalizadas** de segurança;
- Automatizar tarefas de análise e ataque;
- Entender scripts maliciosos ou defensivos com maior profundidade.

Você aprenderá os conceitos essenciais da linguagem:

- Variáveis
- Laços de repetição (Loops)
- Funções
- Estruturas de dados
- Condicionais (If statements)
- Manipulação de arquivos

Editor de Código

Durante os exercícios, você utilizará um **editor de código integrado** (à direita da tela). O curso cobre o básico, o suficiente para criar scripts simples. Para programar fora da plataforma, você pode instalar o Python no site oficial, que também fornece um **IDE (Ambiente de Desenvolvimento Integrado)**.

Sintaxe em Python

A **sintaxe** define como um código deve ser estruturado para ser interpretado corretamente pelo computador. Veja um exemplo simples de programa:

```
# This is an example comment

print("Hello World")
```

- A linha com # é um **comentário**, não executado, usado para explicar o código.
- A função print() exibe na tela o conteúdo entre parênteses.
- Quando estamos imprimindo um **texto (string)**, usamos **aspas duplas** "".

Observação: Os exemplos seguem o padrão **Python 3**.

Operadores Matemáticos

Você pode usar Python como uma calculadora. Veja os principais operadores:

Operação	Símbolo	Exemplo	Resultado
Adição	+	1 + 1	2
Subtração	-	5 - 1	4
Multiplicação	*	10 * 10	100
Divisão	/	10 / 2	5.0
Módulo (resto)	%	10 % 2	0
Exponenciação	**	5 ** 2	25

Operadores de Comparação

Usados em condicionais e loops para **avaliar condições**:

Operação	Símbolo
Maior que	>
Menor que	<
Igual a	==
Diferente de	!=
Maior ou igual a	>=
Menor ou igual a	<=

Variáveis

Variáveis armazenam dados que podem ser usados ou modificados. Exemplo:

```
food = "ice cream"
money = 2000
```

- food guarda uma **string** ("ice cream");
- money guarda um **número inteiro** (2000).

Atualizando uma variável

```
age = 30
age = age + 1
print(age)
```

- age começa em 30;
- Depois é incrementado para 31;
- O comando print(age) exibirá **31**.

Tipos de Dados (Data Types)

Cada variável pode armazenar diferentes **tipos de dados**, como:

- **String**: Texto, como "Hello" ou "abc123!"
- **Integer**: Números inteiros, como 5, -10
- **Float**: Números decimais, como 3.14, 0.5
- **Boolean**: Verdadeiro (True) ou Falso (False)
- **List**: Coleção de valores, como [1, 2, "a", True]

Considerações Finais

Esta introdução à linguagem Python visa dar a você o **conhecimento necessário para criar scripts básicos**, úteis em contextos de **pentest**, automação de tarefas de segurança e análise. Com o tempo e prática, você poderá:

- Criar **ferramentas personalizadas**;
- Automatizar análises de vulnerabilidades;
- Modificar e interpretar exploits existentes;
- Contribuir para ambientes defensivos mais seguros.

Python para Pentesters – Fundamentos: Condicionais, Laços, Funções, Arquivos e Bibliotecas

Este material aborda os principais fundamentos da linguagem Python voltados para a criação de scripts úteis em **segurança ofensiva (pentest)**, tais como:

- Verificação de condições (if statements)
- Laços de repetição (loops)
- Funções reutilizáveis
- Manipulação de arquivos
- Uso de bibliotecas externas

🔷 **Condicionais: if statements**

Permitem que o programa **tome decisões com base em condições**.

if age < 17:

```
    print('You are NOT old enough to drive')
else:
    print('You are old enough to drive')
```

- A palavra-chave if inicia uma verificação condicional.
- else define o que ocorre caso a condição do if **não seja verdadeira**.
- Dois-pontos (:) marcam o fim da linha de condição.
- **Identação** (espaço à esquerda) indica o bloco de código pertencente ao if ou else.

Neste exemplo, se age (idade) for menor que 17, imprime que a pessoa **não tem idade suficiente para dirigir**. Caso contrário, imprime que **tem idade suficiente**.

🔷 **Laços de repetição: while e for**

▶️ **Laço while – executa enquanto uma condição for verdadeira.**

```
i = 1
while i <= 10:
    print(i)
    i = i + 1
```

 Inicia i com valor 1;

 Executa o bloco até i ser maior que 10;

 A cada repetição, imprime i e incrementa +1.

▶️ **Laço for** – usado para iterar sobre **listas** ou sequências.

```
websites = ["facebook.com", "google.com", "amazon.com"]
for site in websites:
    print(site)
```

- A lista websites contém três elementos.
- O laço for percorre cada um, imprimindo seu valor.
- Útil para verificar disponibilidade de sites, status de serviços, etc.

🌀 **Iterando com range():**

```
for i in range(5):
    print(i)
```

Imprime os números de 0 a 4. A função range(5) gera 5 valores: 0, 1, 2, 3, 4.

🔷 **Funções: Reutilização de Código**

Funções evitam repetição e tornam o código modular.

📌 **Exemplo de função com parâmetro:**

```
def sayHello(name):
    print("Hello " + name + "! Nice to meet you.")


sayHello("ben")
```

- def inicia a definição da função;
- name é o parâmetro recebido;
- Bloco indentado é o corpo da função;
- Ao chamar sayHello("ben"), a saída será:
  Hello ben! Nice to meet you.

📌 **Função que retorna um valor:**

```
def calcCost(item):
    if(item == "sweets"):
        return 3.99
    elif (item == "oranges"):
        return 1.99
    else:
        return 0.99


spent = 10
spent = spent + calcCost("sweets")
print("You have spent:" + str(spent))
```

- calcCost() retorna diferentes valores com base no item informado;
- Resultado do return é somado à variável spent;
- A saída será: **You have spent:13.99**

🔷 **Leitura e Escrita de Arquivos**

📄 **Leitura de arquivos:**

```
f = open("file_name", "r")
```

```
print(f.read())
```

- open("file_name", "r"): abre arquivo para leitura (r = read);
- read() lê todo o conteúdo do arquivo;
- Arquivo deve estar no mesmo diretório, ou use o caminho completo.

📝 **Escrita em arquivos:**

```
f = open("demofile1.txt", "a")  # Acrescentar (append)
f.write("The file will include more text..")
f.close()
f = open("demofile2.txt", "w")  # Criar ou sobrescrever
f.write("demofile2 file created, with this content in!")
f.close()
```

- "a" adiciona ao final de um arquivo existente;
- "w" cria ou sobrescreve um arquivo;
- Sempre use f.close() ao final da escrita.

🔷 **Importação de Bibliotecas**

Bibliotecas são coleções de funções prontas, reutilizáveis.

📆 **Exemplo com biblioteca datetime:**

```
import datetime
current_time = datetime.datetime.now()
print(current_time)
```

- import importa a biblioteca;
- Acessa now() através da notação biblioteca.funcao().

🧰 **Bibliotecas úteis para pentesters:**

- **Requests** – biblioteca HTTP simples.
- **Scapy** – criação, envio e análise de pacotes de rede.
- **Pwntools** – desenvolvimento de exploits e resolução de CTFs.

📦 **Instalação de bibliotecas externas:**

Para instalar bibliotecas que não vêm por padrão com Python, use o **pip**:

```
pip install scapy
```

Depois, basta importá-la no código com:

import scapy

✅ Resumo Final

Com o que foi abordado, você já possui uma base sólida para:

- **Criar scripts automatizados** de verificação, enumeração e análise;
- **Ler listas** de alvos de arquivos;
- **Iterar sobre dados** com laços e condições;
- **Criar funções reutilizáveis** e usar bibliotecas externas especializadas.