

Pablo Barros

OS SEGREDOS DOS CÓDIGOS

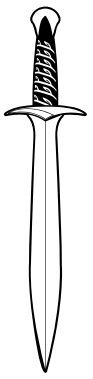
O INÍCIO DA JORNADA EM PYTHON



O I

O CHAMADO DA AVENTURA





Introdução ao Python

Olá, jovens programadores!

Bem-vindos à Terra de Python, um reino mágico onde o conhecimento é a chave para desvendar os segredos da programação. Como Gandalf guia os hobbits, eu estarei aqui para guiá-los através dos mistérios desta linguagem poderosa.

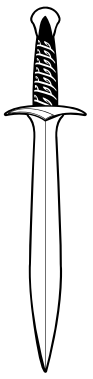
O Que é Python?

Python é uma linguagem de programação de alto nível, conhecida por sua simplicidade e legibilidade. Criada por Guido van Rossum e lançada pela primeira vez em 1991, Python se tornou uma das linguagens mais populares devido à sua sintaxe clara e à vasta gama de bibliotecas disponíveis.

Instalando o Python

Antes de embarcarmos nesta aventura, precisamos nos preparar. Vamos instalar Python em nosso sistema. Visite python.org e faça o download da versão mais recente para seu sistema operacional. Siga as instruções de instalação e, em seguida, abra seu terminal (ou prompt de comando) e digite:

```
python --version
```



Se tudo estiver correto, você verá a versão do Python instalada.

Olá, Mundo!

Nossa jornada começa com o tradicional "Olá, Mundo!". Este é o primeiro feitiço que todo aprendiz de programador deve aprender.

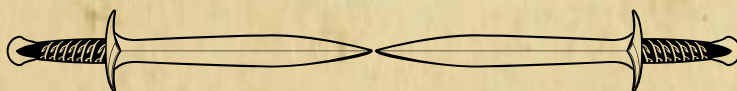
```
olamundo.py  
1 print("Olá, Mundo!")
```

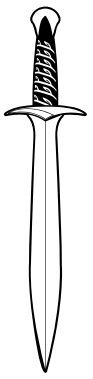
Salve este código em um arquivo chamado `olamundo.py` e execute-o por meio do “Run” da sua IDE ou no terminal com o comando:

```
python olamundo.py
```


02

AS TERRAS ESTRANHAS





Variáveis e Tipos de Dados

Explorando Variáveis

Em nossa jornada, encontraremos diferentes tipos de dados. As variáveis são recipientes que armazenam esses dados para que possamos usá-los posteriormente. Pense nas variáveis como os itens que Frodo e seus amigos carregam em suas mochilas. Vamos ver alguns exemplos:

```
variaveis.py
1 nome = "Frodo"
2 idade = 33
3 altura = 1.25
4 hobbit = True
5
6 print(f"Nome: {nome}, Idade: {idade}, Altura: {altura}, Hobbit: {hobbit}")
```

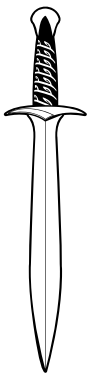
Tipos de Dados

Python nos oferece diversos tipos de dados para lidar com as diferentes informações que encontraremos em nossa jornada.

Inteiros (int)

Números inteiros são utilizados para representar contagens e índices. Pense neles como o número de hobbits em uma festa.

```
tiposdedados.py
1 idade = 33
```



Pontos Flutuantes (float)

Números de ponto flutuante são utilizados para representar valores com casas decimais, como a altura de um personagem.

```
tiposdedados.py
1 altura = 1.25
```

Strings (str)

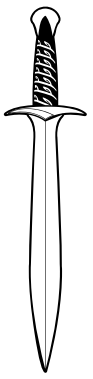
Strings são sequências de caracteres, usadas para representar palavras e frases. Elas são essenciais para guardar nomes e mensagens.

```
tiposdedados.py
1 nome = "Frodo"
```

Booleanos (bool)

Valores booleanos representam verdadeiros (True) ou falsos (False). Eles são úteis para decisões em nossa jornada.

```
tiposdedados.py
1 hobbit = True
```



Listas (list)

Listas são coleções ordenadas de itens que podem ser de tipos diferentes. Elas são como o grupo de personagens que viaja junto.

```
tiposdedados.py
1 amigos = ["Frodo", "Sam", "Merry", "Pippin"]
2
```

Tuplas (tuple)

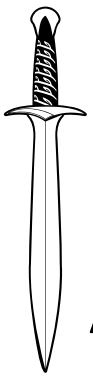
Tuplas são como listas, mas imutáveis, ou seja, não podemos alterar seus valores depois de criadas. São úteis para agrupar dados que não mudam.

```
tiposdedados.py
1 posicao = (100, 200)
```

Dicionários (dict)

Dicionários armazenam pares de chave-valor, permitindo acessar os valores através das chaves. Eles são como mapas que associam locais a descrições.

```
tiposdedados.py
1 personagem = {"nome": "Frodo", "idade": 33, "altura": 1.25}
```

Operadores Matemáticos

Assim como Gandalf usa sua magia, usaremos operadores matemáticos para manipular números. Veja alguns exemplos simples de operações matemáticas:

```
operadores.py
1 soma = 10 + 5
2 subtracao = 10 - 5
3 multiplicacao = 10 * 5
4 divisao = 10 / 5
5
6 print(soma, subtracao, multiplicacao, divisao)
7
```

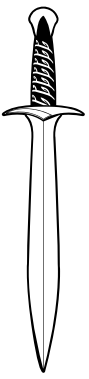
Manipulando Strings

Vamos aprender a manipular palavras e frases, como quando Gandalf recita seus encantamentos. As strings nos permitem trabalhar com texto:

```
strings.py
1 mensagem = "Bem-vindo à Terra de Python"
2 print(mensagem.upper()) # Transformar em maiúsculas
3 print(mensagem.lower()) # Transformar em minúsculas
4 print(mensagem.replace("Python", "Programação")) # Substituir palavras
```

Trabalhando com Listas

Listas nos permitem armazenar e manipular coleções de itens. Imagine que estamos organizando uma festa em Hobbiton e precisamos de uma lista de convidados:



```
lists.py

1 convidados = ["Frodo", "Sam", "Merry", "Pippin"]
2 print(convidados)
3
4 convidados.append("Gandalf") # Adicionando um novo convidado
5 print(convidados)
6
7 convidados.remove("Pippin") # Removendo um convidado
8 print(convidados)
9
10 print(convidados[1]) # Acessando um convidado específico
```

Utilizando Dicionários

Dicionários são úteis para armazenar informações complexas sobre personagens. Vamos criar um dicionário para o nosso amigo Frodo:

```
lists.py

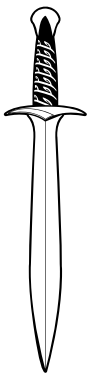
1 frodo = {
2     "nome": "Frodo Baggins",
3     "idade": 33,
4     "altura": 1.25,
5     "hobbit": True
6 }
7
8 print(frodo["nome"])
9 print(frodo.get("idade"))
10
11 frodo["missao"] = "Destruir o Anel" # Adicionando nova informação
12 print(frodo)
```

Agora que exploramos as diferentes terras dos tipos de dados em Python, estamos mais preparados para continuar nossa jornada. Saber como armazenar e manipular dados é essencial para resolver problemas e criar soluções mágicas. No próximo capítulo, enfrentaremos os desafios das estruturas condicionais e loops, ferramentas indispensáveis para qualquer aventureiro.

03

DESAFIOS DO CAMINHO





Estruturas Condicionais e Loops

Estruturas Condicionais

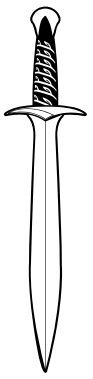
Em nossa jornada, muitas vezes nos encontraremos em situações onde precisamos tomar decisões, assim como Frodo precisou decidir o caminho a seguir na sua aventura. Em Python, usamos estruturas condicionais para tomar essas decisões.

`if`, `elif` e `else`

As estruturas if, elif e else são usadas para executar diferentes blocos de código com base em condições diferentes.

```
condicionais.py
1  idade = 33
2
3  if idade < 18:
4      print("Você é jovem, pequeno hobbit.")
5  elif idade < 100:
6      print("Você está na flor da idade!")
7  else:
8      print("Você é tão velho quanto a Terra Média.")
```

No exemplo acima, o Python verifica cada condição na ordem. Se a condição em if for verdadeira, o código dentro dela será executado. Se não for, ele passa para a próxima condição elif, e assim por diante. Se nenhuma condição for verdadeira, o código dentro do else será executado.



Comparações e Operadores Lógicos

Para fazer essas comparações, usamos operadores de comparação (`==`, `!=`, `<`, `>`, `<=`, `>=`) e operadores lógicos (`and`, `or`, `not`).

```
condicionais.py

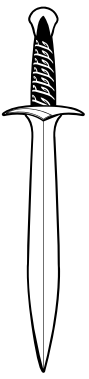
1 # Comparação de valores
2 forca = 80
3 sabedoria = 60
4
5 if forca > 70 and sabedoria > 50:
6     print("Você é um guerreiro sábio!")
7 elif forca > 70 or sabedoria > 50:
8     print("Você tem algumas habilidades impressionantes!")
9 else:
10    print("Você ainda precisa treinar mais.")
```

Repetindo Tarefas com Loops

Para repetir tarefas, usaremos loops, como caminhar incansavelmente por Mordor. Os loops nos permitem executar um bloco de código várias vezes, o que é extremamente útil para automatizar tarefas repetitivas.

Loop for

O loop `for` é ideal para quando sabemos quantas vezes queremos repetir algo. Ele é comumente usado para iterar sobre uma sequência (como uma lista ou uma string).



```
Loops.py
1 amigos = ["Frodo", "Sam", "Merry", "Pippin"]
2
3 for amigo in amigos:
4     print(f"Saudações, {amigo}!")
```

No exemplo acima, estamos saudando cada um dos amigos de Frodo. O loop **for** percorre a lista amigos e atribui cada valor à variável amigo, uma de cada vez, executando o código dentro do loop para cada amigo.

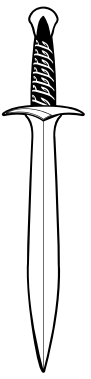
Podemos também usar o **range** para gerar uma sequência de números:

```
Loops.py
1 for i in range(5):
2     print(f"Passo {i + 1} em nossa jornada.")
```

Aqui, `range(5)` gera uma sequência de números de 0 a 4, e o loop **for** percorre essa sequência, imprimindo cada número incrementado de 1.

Loop while

O loop **while** é útil quando não sabemos exatamente quantas vezes queremos repetir algo, mas temos uma condição a ser cumprida. Ele continua executando o bloco de código enquanto a condição for verdadeira.



```
Loops.py

1 passos = 0
2 while passos < 5:
3     print(f"Dando passo {passos + 1}")
4     passos += 1
```

Neste exemplo, o **loop while** continua a execução enquanto passos for menor que 5. Em cada iteração, incrementamos passos em 1 até que a condição `passos < 5` se torne falsa.

Controle de Fluxo com `break` e `continue`

Às vezes, precisamos interromper um loop antes que ele termine todas as iterações ou pular uma iteração específica. Para isso, usamos as declarações `break` e `continue`.

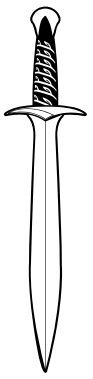
Break

A declaração `break` encerra o loop imediatamente.

```
Loops.py

1 for i in range(10):
2     if i == 5:
3         break
4     print(i)
```

Neste exemplo, o loop será interrompido quando `i` for igual a 5, então a saída será 0, 1, 2, 3, 4.



Continue

A declaração `continue` pula a iteração atual e passa para a próxima iteração do loop.

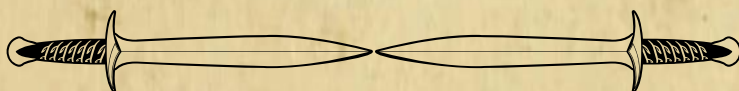
```
Loops.py
1 for i in range(10):
2     if i % 2 == 0:
3         continue
4     print(i)
```

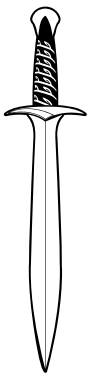
Aqui, o loop imprimirá apenas números ímpares, pois `continue` faz com que o loop pule a impressão de números pares.

As estruturas condicionais e loops são ferramentas poderosas que nos permitem controlar o fluxo do nosso programa, tomando decisões e repetindo tarefas conforme necessário. Agora que dominamos esses conceitos, estamos prontos para enfrentar desafios ainda maiores na nossa jornada através da Terra de Python. No próximo capítulo, vamos desvendar os segredos dos magos com funções e bibliotecas.

04

OS SEGREDOS DOS MAGOS





Funções e Bibliotecas

Invocando Feitiços com Funções

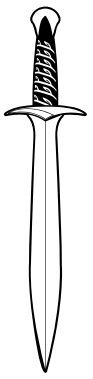
Em nossa jornada, nós usaremos funções. As funções nos permitem organizar nosso código em blocos reutilizáveis, facilitando a manutenção e a leitura.

Criando Funções

Para criar uma função em Python, usamos a palavra-chave **def** seguida pelo nome da função e parênteses. O código dentro da função é executado quando a função é chamada.

```
funcoes.py
1 def saudar(ser):
2     print(f"Olá, {ser}!")
3
4 saudar("Gandalf")
5 saudar("Frodo")
6
```

No exemplo acima, criamos uma função chamada **saudar** que aceita um parâmetro **ser** e imprime uma saudação. Chamamos a função duas vezes, uma vez para Gandalf e outra para Frodo.



Parâmetros e Argumentos

Funções podem aceitar múltiplos parâmetros. Isso nos permite passar diferentes tipos de informações para a função usar.

```
funcoes.py

1 def apresentar_personagem(nome, idade, raça):
2     print(f"Nome: {nome}, Idade: {idade}, Raça: {raça}")
3
4 apresentar_personagem("Frodo", 33, "Hobbit")
5 apresentar_personagem("Gandalf", 2019, "Mago")
6
7
```

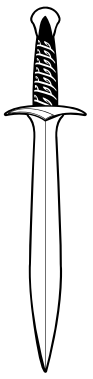
Retornando Valores

Funções também podem retornar valores usando a palavra-chave **return**. Isso é útil quando precisamos que a função calcule algo e nos dê o resultado.

```
funcoes.py

1 def somar(a, b):
2     return a + b
3
4 resultado = somar(10, 5)
5 print(f"O resultado da soma é: {resultado}")
6
7
8
```

No exemplo acima, a função **somar** retorna a soma de **a** e **b**. Armazenamos o resultado em uma variável e o imprimimos.



Funções Aninhadas e Recursão

Funções podem chamar outras funções, incluindo elas mesmas. Isso é chamado de recursão e é útil para resolver problemas que podem ser divididos em subproblemas menores.

```
funcoes.py

1 def fatorial(n):
2     if n == 1:
3         return 1
4     else:
5         return n * fatorial(n - 1)
6
7 print(f"O fatorial de 5 é: {fatorial(5)}")
```

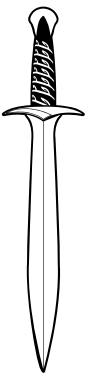
No exemplo acima, a função fatorial calcula o fatorial de um número usando recursão.

Utilizando Bibliotecas

Em nossa jornada por esse mundo maravilhoso e desafiador nós temos bibliotecas em Python para nos ajudar. Bibliotecas são coleções de funções e métodos que podemos usar para facilitar nossas tarefas.

Importando Bibliotecas

Para usar uma biblioteca, precisamos importá-la. Isso é feito com a palavra-chave **import**.



```
bibliotecas.py

1 import math
2
3 print(math.sqrt(25)) # Raiz quadrada
4 print(math.pi)     # Valor de Pi
5
```

No exemplo acima, importamos a biblioteca `math` e usamos suas funções `sqrt` e `pi`.

Bibliotecas Padrão

Python vem com muitas bibliotecas padrão que podemos usar sem precisar instalar nada. Aqui estão algumas das mais úteis:

math: Funções matemáticas avançadas.

datetime: Manipulação de datas e horas.

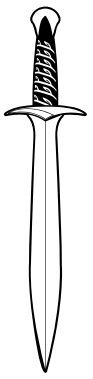
random: Geração de números aleatórios.

os: Interação com o sistema operacional.

Exemplo com **datetime**:

```
bibliotecas.py

1 import datetime
2
3 agora = datetime.datetime.now()
4 print(f"Data e hora atuais: {agora}")
5
6 data_futura = agora + datetime.timedelta(days=10)
7 print(f"Data daqui a 10 dias: {data_futura}")
```



Instalando Bibliotecas Externas

Além das bibliotecas padrão, existem muitas bibliotecas criadas por terceiros que podemos instalar usando o pip. Por exemplo, para trabalhar com requisições HTTP, podemos usar a biblioteca **requests**.

```
import datetime

agora = datetime.datetime.now()
print(f"Data e hora atuais: {agora}")

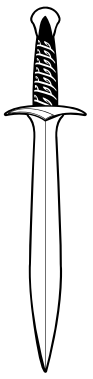
data_futura = agora + datetime.timedelta(days=10)
print(f"Data daqui a 10 dias: {data_futura}")
```

Exemplo com **requests**:

```
bibliotecas.py

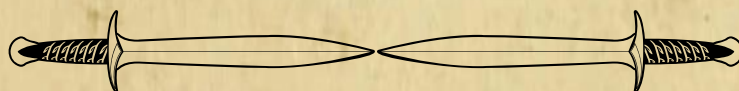
1 import requests
2
3 resposta = requests.get("https://api.github.com")
4 print(resposta.status_code)
5 print(resposta.json())
```

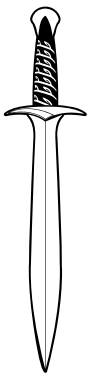
No exemplo acima, instalamos a biblioteca **requests** e a usamos para fazer uma requisição GET para a API do GitHub, imprimindo o status da resposta e o conteúdo em JSON.



As funções e bibliotecas são ferramentas poderosas em Python que nos permitem escrever código modular e reutilizável e aproveitar soluções prontas para resolver problemas comuns. Agora que dominamos esses segredos dos magos, estamos prontos para enfrentar qualquer desafio que apareça em nossa jornada através da Terra de Python. Continue explorando, aprendendo e codificando. Que sua curiosidade e determinação sejam sempre sua melhor guia!

AGRADECIMIENTOS





OBRIGADO POR LER ATÉ AQUI

Este Ebook foi gerado com a ajuda de IA e diagramado por um humano Tolkienista. E A todos os valentes tolkienistas que embarcaram nesta jornada através da Terra de Python, meu sincero agradecimento. Que o conhecimento adquirido aqui seja uma luz em seus caminhos, assim como a estrela de Eärendil. Continuem explorando, aprendendo e criando. Que a sabedoria dos antigos esteja sempre com vocês.