Introdução ao Python

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Sobre Mim

- +9 desenvolvedor backend (Python e Java)
- De paraquedas... total!
- Construir sistemas com qualidade.
- Netflix, Games e Futebol.
- @guicarvalho (Linkedin e Github)

Objetivo Geral

Conhecer um pouco sobre a história da linguagem, explorando as ideias de seu criador. Responder perguntas como: Onde eu devo usar Python?

Pré-requisitos

- Gostar de história.

Percurso

Etapa 1 **Oi, eu sou o Python!**

Etapa 2 Onde eu devo usar essa linguagem?

Etapa 1

Oi, eu sou o Python

// Introdução ao Python

A origem

Python nasceu em 1989 como um hobby, do programador Guido Van Rossum. A ideia inicial era dar continuidade a linguagem ABC, que era desenvolvida no Centro de Pesquisa Holandês (CWI).

Os objetivos

Python foi influenciada por ABC, que era uma linguagem pensada para iniciantes, devido a sua facilidade de aprendizagem e utilização.

Os objetivos de Van Rossum para a linguagem Python eram:

- Uma linguagem fácil e intuitiva.
- Código aberto, para que todos possam contribuir.
- Código tão inteligível quanto Inglês.
- Adequada para tarefas diárias, e produtiva!

Linha do tempo

Guido Van Rossum inicia o desenvolvimento em 1989 e em fevereiro de 1991 é lançada a primeira versão pública: 0.9.0.

É tetra!

Brasil é treta, Romário eleito o melhor jogador da competição e o Python tem a versão 1.0 lançada!

Adeus...

Em 1995 Guido lança a versão 1.2, enquanto trabalhava no CWI. Com o vínculo encerrado com o centro de pesquisa, Van Rossum e a equipe principal de desenvolvedores Python mudaram-se para BeOpen.com, nasce a **BeOpen Python Labs**.

Anos 2000

A segunda versão do Python é publicada em Outubro de 2000, nessa versão nasce **List Comprehensions** e uma melhoria no coletor de lixo para remoção de referências cíclicas.

Em 2001 nasce a Python Software Foundation (PSF), que a partir do Python 2.1 possui todo o código, documentação e especificações da linguagem.

Python 3

Em 2008 é lançada a versão 3.0, que resolveu muitos problemas de design da linguagem e melhorou a performance. Algumas mudanças foram muito profundas e dessa forma a versão 3.x não é retrocompatível.

Atualmente estamos na versão **3.10.2** do Python.

Percurso

~~Etapa 1~~ ~~Oi, eu sou o Python!~~

Etapa 2 **Onde eu devo usar essa linguagem?**

Etapa 2

Onde eu devo usar essa linguagem?

// Introdução ao Python

Só não é boa para APP Mobile!

Python é uma linguagem muito versátil!

- Tipagem dinâmica e forte.
- Multiplataforma e multiparadigma.
- Comunidade gigante e ativa.
- Curva de aprendizado baixa.

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Configuração do ambiente de desenvolvimento

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

Instalar e configurar nossa máquina, para desenvolver projetos utilizando Python.

Pré-requisitos

- Conexão com a Internet.

Percurso

Etapa 1 **Instalar o Python**

Etapa 2 Baixar e configurar a IDE

Etapa 1

Instalar o Python

// Configuração do ambiente de desenvolvimento

Linux e MacOS

Provavelmente o Python já está instalado na sua máquina. Para verificar qual a versão entre com o comando:

python -V ou python3 -V

Windows

Como a maioria dos programas Windows, o Python possui um instalador que pode ser baixado acessando: http://www.python.org. Após executar a download, faça a instalação seguindo os passos descritos no tutorial: https://python.org.br/instalacao-windows/.

Percurso

~~Etapa 1~~ ~~Instalar o Python~~

Etapa 2 **Baixar e configurar a IDE**

Etapa 2

Baixar e configurar a IDE

// Introdução ao Python

VSCode ou PyCharm

VSCode é uma ótima ferramenta e será a nossa escolha por alguns motivos:

- Gratuita.
- Suporta múltiplas tecnologias.
- Boa performance.

Acesse o site: https://code.visualstudio.com/.

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Primeiro programa

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

Criar nosso primeiro programa em Python.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1 **Criando nosso primeiro programa**

Etapa 1

Criando nosso primeiro programa

// Exibindo uma mensagem de boas vindas

A receita

Programar consiste em informar ao computador uma sequência de rotinas que devem ser processadas. Imagine uma receita de bolo, precisamos saber os ingredientes e modo de preparo. Seguindo corretamente as instruções ao fim do processo teremos um bolo.

Criando nosso arquivo

Para criar a nossa receita de bolo em Python, precisamos criar um arquivo com extensão ***py***. Com o arquivo criado podemos inserir nossos ingredientes e modo de preparo!

Percurso

~~Etapa 1~~ ~~Criando nosso primeiro programa~~

Hands On!

“Falar é fácil. Mostre-me o código!”

Linus Torvalds

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Tipos de dados

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

Conhecer os tipos de dados em Python.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1 **O que são tipos?**

Etapa 2 Tipos númericos

Etapa 3 Booleanos e Strings

Etapa 1

O que são tipos?

// Espaço alocado e operações

Por que usamos tipos?

Os tipos servem para definir as caracteristicas e comportamentos de um valor (objeto) para o interpretador. Por exemplo:

Com esse tipo eu sou capaz de realizar operações matemáticas.

Esse tipo para ser armazenado em memória irá consumir 24 bytes.

Tipos em Python

Os tipos built-in são:

Texto	str
Númerico	int, float, complex
Sequência	list, tuple, range
Mapa	dict
Coleção	set, fronzenset
Booleano	bool
Binário	bytes, bytearray, memoryview

Percurso

~~Etapa 1~~	~~O que são tipos?~~
Etapa 2	**Tipos númericos**
Etapa 3	Booleanos e Strings

Etapa 2

Tipos númericos

// Trabalhando com números

Números inteiros

Números inteiros são representados pela classe ***int*** e possuem precisão ilimitada. São exemplos válidos de números inteiros:

1, 10, 100, -1, -10, -100...99001823

Números de ponto flutuante

Os números de ponto flutuante são usados para representar os números racionais e sua implementação é feita pela classe ***float***. São exemplos válidos de números de ponto flutuante:

1.5, -10.543, 0.76...999278.002

Percurso

~~Etapa 1~~ ~~O que são tipos?~~

~~Etapa 2~~ ~~Tipos númericos~~

Etapa 3 **Booleanos e Strings**

Etapa 3

Booleanos e Strings

// Trabalhando com booleanos e textos

Booleano

É usado para representar verdadeiro ou falso, e é implementado pela classe ***bool***. Em Python o tipo booleano é uma subclasse de ***int***, uma vez que qualquer número diferente de 0 representa verdadeiro e 0 representa falso. São exemplos válidos de booleanos:

True e False

Strings

Strings ou cadeia de caracteres são usadas para representar valores alfanúmericos, em Python as strings são definidas utilizando a classe ***str***. São exemplos válidos de string:

"Python", 'Python', """Python""", '''Python''', "p"

Percurso

~~Etapa 1~~ ~~O que são tipos?~~

~~Etapa 2~~ ~~Tipos númericos~~

~~Etapa 3~~ ~~Booleanos e Strings~~

Hands On!

“Falar é fácil. Mostre-me o código!”

Linus Torvalds

Links Úteis

- https://github.com/guicarvalho/trilha-python-dio
- Referências:
  - https://docs.python.org/3/library/stdtypes.html

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Modo interativo

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

Como usar o modo interativo do interpretador Python.

Pré-requisitos

- Python 3

Percurso

Etapa 1 **Usando o modo interativo**

Etapa 2 Funções dir e help

Etapa 1

Usando o modo interativo

// Como realizar testes rápidos sem criar um arquivo

O modo interativo

O interpretador Python pode executar em modo que possibilite o desenvolvedor a escrever código, e ver o resultado na hora.

Iniciando o modo interativo

Existem duas formas de iniciar o modo interativo, chamando apenas o interpretador *(python)* ou executando o script com a flag -i *(python -i app.py)*.

Percurso

~~Etapa 1~~ ~~Usando o modo interativo~~

Etapa 2 **Funções dir e help**

Etapa 2

Funções dir e help

// Documentação offline e direto no terminal

dir

Sem argumentos, retorna a lista de nomes no escopo local atual. Com um argumento, retorna uma lista de atributos válidos para o objeto. Exemplo:

dir()

dir(100)

help

Invoca o sistema de ajuda integrado. É possível fazer buscas em modo interativo ou informar por parâmetro qual o nome do módulo, função, classe, método ou variável. Exemplo:

help()

help(100)

Percurso

~~Etapa 1~~ ~~Usando o modo interativo~~

~~Etapa 2~~ ~~Funções dir e help~~

Hands On!

“Falar é fácil. Mostre-me o código!”

Linus Torvalds

Links Úteis

- https://github.com/guicarvalho/trilha-python-dio
- Referências:
  - https://wiki.python.org.br/ModoInterativo

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Variáveis e constantes

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

Entender o que são e como utilizar variáveis e constantes.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1 **O que são variáveis e constantes?**

Etapa 2 Boas práticas

Etapa 1

O que são variáveis e constantes?

// Armazenando valores mutáveis e imutáveis

Variáveis

Em linguagens de programação podemos definir valores que podem sofrer alterações no decorrer da execução do programa. Esses valores recebem o nome de variáveis, pois eles nascem com um valor e não necessariamente devem permanecer com o mesmo durante a execução do programa.

```
age = 23
name = 'Guilherme'
print(f'Meu nome é {name} e eu tenho {age} ano(s) de idade.')
>>> Meu nome é Guilherme e eu tenho 23 ano(s) de idade.


age, name = (23, 'Guilherme')
print(f'Meu nome é {name} e eu tenho {age} ano(s) de idade.')
>>> Meu nome é Guilherme e eu tenho 23 ano(s) de idade.
```

Alterando os valores

Perceba que não precisamos definir o tipo de dados da variável, o Python faz isso automaticamente para nós. Por isso não podemos simplesmente criar uma variável sem atribuir um valor. Para alterar o valor da variável basta fazer uma atribuição de um novo valor:

```
age = 28
name = 'Guilherme'
print(f'Meu nome é {name} e eu tenho {age} ano(s) de idade.')
>>> Meu nome é Guilherme e eu tenho 28 ano(s) de idade.

age = 27
name = 'Giovanna'
print(f'Meu nome é {name} e eu tenho {age} ano(s) de idade.')
>>> Meu nome é Giovanna e eu tenho 27 ano(s) de idade.
```

Constantes

Assim como as variáveis, constantes são utilizadas para armazenar valores. Uma constante nasce com um valor e permanece com ele até o final da execução do programa, ou seja, o valor é imutável.

Python não tem constantes

Não existe uma palavra reservada para informar ao interpretador que o valor é constante. Em algumas linguagens por exemplo: Java e C utilizamos *final* e *const*, respectivamente para declarar uma constante.

Em Python usamos a convenção que diz ao programador que a variável é uma constante. Para fazer isso, você deve criar a variável com o nome todo em letras maíusculas:

```
ABS_PATH = '/home/guilherme/Documents/python_course/'
DEBUG = True
STATES = [
    'SP',
    'RJ',
    'MG',
]
AMOUNT = 30.2
```

Etapa 2

Boas práticas

// Seguindo as convenções

Percurso

~~Etapa 1~~ ~~O que são variáveis e constantes?~~

Etapa 2 **Boas práticas**

Boas práticas

- O padrão de nomes deve ser *snake case*.
- Escolher nomes sugestivos.
- Nome de constantes todo em maiúsculo.

Percurso

~~Etapa 1~~ ~~O que são variáveis e constantes?~~

~~Etapa 2~~ ~~Boas práticas~~

Hands On!

“Falar é fácil. Mostre-me o código!”

Linus Torvalds

Links Úteis

- https://github.com/guicarvalho/trilha-python-dio

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Conversão de tipos

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

Aprender a converter os tipos das variáveis.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1 **Convertendo tipos**

Etapa 1

Convertendo tipos

Convertendo tipos

Em alguns momentos é necessário será necessário converter o tipo de uma variável para manipular de forma diferente. Por exemplo:

Variáveis do tipo *string*, que armazenam números e precisamos fazer alguma operação matemática com esse valor.

Inteiro para float

```
preco = 10
print(preco)
>>> 10

preco = float(preco)
print(preco)
>>> 10.0

preco = 10 / 2
print(preco)
>>> 5.0
```

Float para inteiro

```
preco = 10.30
print(preco)
>>> 10.3

preco = int(preco)
print(preco)
>>> 10
```

Conversão por divisão

```
preco = 10
print(preco)
>>> 10

print(preco / 2)
>>> 5.0

print(preco / 2)
>>> 5
```

Numérico para string

```
preco = 10.50
idade = 28

print(str(preco))
>>> 10.5

print(str(idade))
>>> 28

texto = f"idade {idade} preco {preco}"
print(texto)
>>> idade 28 preco 10.5
```

String para número

```
preco = "10.50"
idade = "28"

print(float(preco))
>>> 10.50

print(int(idade))
>>> 28
```

Erro de conversão

```
preco = "python"
print(float(preco))

>>>
Traceback (most recent call last):
  File "main.py", line 3, in <module>
        print(float(preco))
ValueError: could not convert string to float: 'python'
```

Percurso

~~Etapa 1~~ ~~Convertendo tipos~~

Hands On!

“Falar é fácil. Mostre-me o código!”

Linus Torvalds

Links Úteis

- https://github.com/guicarvalho/trilha-python-dio

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Funções de entrada e saída

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

Aprender como receber e exibir informações para o usuário.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1 **Lendo valores com a função input**

Etapa 2 Exibindo valores com a função print

Etapa 1

Lendo valores com a função input

Função input

A função builtin *input* é utilizada quando queremos ler dados da entrada padrão (teclado). Ela recebe um argumento do tipo string, que é exibido para o usuário na saída padrão (tela). A função lê a entrada, converte para string e retorna o valor.

Exemplo

```
nome = input("Informe o seu nome: ")
>>> Informe o seu nome: |
```

Percurso

~~Etapa 1~~ ~~Lendo valores com a função input~~

Etapa 2 **Exibindo valores com a função print**

Etapa 2

Exibindo valores com a função print

Função print

A função builtin *print* é utilizada quando queremos exibir dados na saída padrão (tela). Ela recebe um argumento obrigatório do tipo varargs de objetos e 4 argumentos opcionais (sep, end, file e flush). Todos os objetos são convertidos para string, separados por *sep* e terminados por *end*. A string final é exibida para o usuário.

Exemplo

```
nome = "Guilherme"
sobrenome = "Carvalho"

print(nome, sobrenome)
print(nome, sobrenome, end="...\n")
print(nome, sobrenome, sep="#")

>>> Guilherme Carvalho
>>> Guilherme Carvalho...
>>> Guilherme#Carvalho
```

Percurso

~~Etapa 1~~ ~~Lendo valores com a função input~~

~~Etapa 2~~ ~~Exibindo valores com a função print~~

Hands On!

“Falar é fácil. Mostre-me o código!”

Linus Torvalds

Links Úteis

- https://github.com/guicarvalho/trilha-python-dio
- https://docs.python.org/3/library/functions.html#input
- https://docs.python.org/3/library/functions.html#print

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Operadores aritméticos

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

O que são operadores aritméticos e como utilizá-los.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1 **Conhecendo os operadores aritméticos**

Etapa 2 Precedência de operadores

Etapa 1

Conhecendo os operadores aritméticos

O que são?

Os operadores aritméticos executam operações matemáticas, como adição, subtração com operandos.

Adição, subtração e multiplicação

```
# Adição
print(1 + 1)
>>> 2

# Subtração
print(10 - 2)
>>> 8

# Multiplicação
print(4 * 3)
>>> 12
```

Divisão e divisão inteira

```
# Divisão
print(12 / 3)
>>> 4.0

# Divisão inteira
print(12 // 2)
>>> 6
```

Módulo e exponenciação

```
# Módulo
print(10 % 3)
>>> 1

# Exponenciação
print(2 ** 3)
>>> 8
```

Percurso

~~Etapa 1~~ ~~Operadores aritméticos~~

Etapa 2 **Precedência de operadores**

Etapa 2

Precedência de operadores

Na matemática

Na matemática existe uma regra que indica quais operações devem ser executadas primeiro. Isso é útil pois ao analisar uma expressão, a depender da ordem das operações o valor pode ser diferente:

x = 10 - 5 * 2

x é igual a 10 ou 0?

Na matemática

A definição indica a seguinte ordem como a correta:

- Parêntesis
- Expoêntes
- Multiplicações e divisões (da esquerda para a direita)
- Somas e subtrações (da esquerda para a direita)

Exemplo

```
print(10 - 5 * 2)
>>> 0

print((10 - 5) * 2)
>>> 10

print(10 ** 2 * 2)
>>> 200

print(10 ** (2 * 2))
>>> 10000

print(10 / 2 * 4)
>>> 20.0
```

Percurso

~~Etapa 1~~ ~~Operadores aritméticos~~

~~Etapa 2~~ ~~Precedência de operadores~~

Hands On!

“Falar é fácil. Mostre-me o código!”

Linus Torvalds

Links Úteis

- https://github.com/guicarvalho/trilha-python-dio

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Operadores de comparação

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

O que são operadores de comparação e como utilizá-los.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1 **Conhecendo os operadores de comparação**

Etapa 1

Conhecendo os operadores de comparação

O que são?

São operadores utilizados para comparar dois valores.

Igualdade

```
saldo = 450
saque = 200

print(saldo == saque)
>>> False
```

Diferença

```
saldo = 450
saque = 200
print(saldo != saque)
>>> True
```

Maior que / maior ou igual

```
saldo = 450
saque = 200
print(saldo > saque)
>>> True

print(saldo >= saque)
>>> True
```

Menor que / menor ou igual

```
saldo = 450
saque = 200
print(saldo < saque)
>>> False

print(saldo <= saque)
>>> False
```

Percurso

~~Etapa 1~~ ~~Conhecendo os operadores de comparação~~

Hands On!

“Falar é fácil. Mostre-me o código!”

Linus Torvalds

Links Úteis

- https://github.com/guicarvalho/trilha-python-dio

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Operadores de atribuição

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

O que são operadores de atribuição e como utilizá-los.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1 **Conhecendo os operadores de atribuição**

Etapa 1

Conhecendo os operadores de atribuição

O que são?

São operadores utilizados para definir o valor inicial ou sobrescrever o valor de uma variável.

Atribuição simples

```
saldo = 500

print(saldo)
>>> 500
```

Atribuição com adição

```
saldo = 500
saldo += 200

print(saldo)
>>> 700
```

Atribuição com subtração

```
saldo = 500
saldo -= 100

print(saldo)
>>> 400
```

Atribuição com multiplicação

```
saldo = 500
saldo *= 2

print(saldo)
>>> 1000
```

Atribuição com divisão

```
saldo = 500
saldo /= 5

print(saldo)
>>> 100.0

saldo = 500
saldo //= 5

print(saldo)
>>> 100
```

Atribuição com módulo

```
saldo = 500
saldo %= 480

print(saldo)
>>> 20
```

Atribuição com exponenciação

```
saldo = 80
saldo **= 2

print(saldo)
>>> 6400
```

Percurso

~~Etapa 1~~ ~~Conhecendo os operadores de atribuição~~

Hands On!

“Falar é fácil. Mostre-me o código!”

Linus Torvalds

Links Úteis

- https://github.com/guicarvalho/trilha-python-dio

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Operadores de lógicos

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

O que são operadores lógicos e como utilizá-los.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1 **Conhecendo os operadores lógicos**

Etapa 1

Conhecendo os operadores lógicos

O que são?

São operadores utilizados em conjunto com os operadores de comparação, para montar uma expressão lógica. Quando um operador de comparação é utilizado, o resultado retornado é um booleano, dessa forma podemos combinar operadores de comparação com os operadores lógicos, exemplo:

op_comparacao + op_logico + op_comparacao... N ...

Exemplo

```
saldo = 1000
saque = 200
limite = 100

saldo >= saque
>>> True

saque <= limite
>>> False
```

Operador E

```
saldo = 1000
saque = 200
limite = 100

saldo >= saque and saque <= limite
>>> False
```

Operador OU

```
saldo = 1000
saque = 200
limite = 100

saldo >= saque or saque <= limite
>>> True
```

Operador Negação

```
contatos_emergencia = []

not 1000 > 1500
>>> True

not contatos_emergencia
>>> True

not "saque 1500;"
>>> False

not ""
>>> True
```

Parênteses

```
saldo = 1000
saque = 250
limite = 200
conta_especial = True

saldo >= saque and saque <= limite or conta_especial and saldo >= saque
>>> True

(saldo >= saque and saque <= limite) or (conta_especial and saldo >= saque)
>>> True
```

Percurso

~~Etapa 1~~ ~~Conhecendo os operadores lógicos~~

Hands On!

“Falar é fácil. Mostre-me o código!”

Linus Torvalds

Links Úteis

- https://github.com/guicarvalho/trilha-python-dio

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Operadores de identidade

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

O que são operadores de identidade e como utilizá-los.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1 **Conhecendo os operadores de identidade**

Etapa 1

Conhecendo os operadores de identidade

O que são?

São operadores utilizados para comparar se os dois objetos testados ocupam a mesma posição na memória.

Exemplo

```
curso = "Curso de Python"
nome_curso = curso
saldo, limite = 200, 200

curso is nome_curso
>>> True

curso is not nome_curso
>>> False

saldo is limite
>>> True
```

Percurso

~~Etapa 1~~ ~~Conhecendo os operadores de identidade~~

Hands On!

“Falar é fácil. Mostre-me o código!”

Linus Torvalds

Links Úteis

- https://github.com/guicarvalho/trilha-python-dio

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Operadores de associação

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

O que são operadores de associação e como utilizá-los.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1 **Conhecendo os operadores de comparação**

Etapa 1

Conhecendo os operadores de associação

O que são?

São operadores utilizados para verificar se um objeto está presente em uma sequência .

Exemplo

```
curso = "Curso de Python"
frutas = ["laranja", "uva", "limão"]
saques = [1500, 100]

"Python" in curso
>>> True

"maçã" not in frutas
>>> True

200 in saques
>>> False
```

Percurso

~~Etapa 1~~ ~~Conhecendo os operadores de associação~~

Hands On!

“Falar é fácil. Mostre-me o código!”

Linus Torvalds

Links Úteis

- https://github.com/guicarvalho/trilha-python-dio

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Indentação e blocos

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

Aprender como o interpretador Python utiliza a indentação do código para delimitar os blocos de comandos.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1 **Indentação e os blocos de comandos**

Etapa 1

O papel da indentação

A estética

Identar código é uma forma de manter o código fonte mais legível e manutenível. Mas em Python ela exerce um segundo papel, através da indentação o interpretador consegue determinar onde um bloco de comando inicia e onde ele termina.

Bloco de comando

As linguagens de programação costumam utilizar caracteres ou palavras reservadas para terminar o início e fim do bloco. Em Java e C por exemplo, utilizamos chaves:

Bloco em Java

```
void sacar(double valor) {  // início do bloco do método

    if (this.saldo >= valor) { // início do bloco do if

        this.saldo -= valor;

    } // fim do bloco do if

}  // fim do bloco do método
```

Bloco em Java sem formatar

```
void sacar(double valor) {  // início do bloco do método
if (this.saldo >= valor) {  // início do bloco do if
this.saldo -= valor;
}  // fim do bloco do if
}  // fim do bloco do método
```

Utilizando espaços

Existe uma convenção em Python, que define as boas práticas para escrita de código na linguagem. Nesse documento é indicado utilizar 4 espaços em branco por nível de indentação, ou seja, a cada novo bloco adicionamos 4 novos espaços em branco.

Bloco em Python

```
def sacar(self, valor: float) -> None:  # início do bloco do método

    if self.saldo >= valor:  # início do bloco do if

        self.saldo -= valor

    # fim do bloco do if

# fim do bloco do método
```

Isso não funciona em Python!

```
def sacar(self, valor: float) -> None:  # início do bloco do método
if self.saldo >= valor:  # início do bloco do if
self.saldo -= valor
# fim do bloco do if
# fim do bloco do método
```

Qual versão é mais fácil de ler?

```
void sacar(double valor) {
if (this.saldo >= valor) {
this.saldo -= valor;}}
```

```
def sacar(self, valor: float) -> None:
    if self.saldo >= valor:
        self.saldo -= valor
```

Percurso

~~Etapa 1~~ ~~Indentação e os blocos de comandos~~

Hands On!

“Falar é fácil. Mostre-me o código!”

Linus Torvalds

Links Úteis

- https://github.com/guicarvalho/trilha-python-dio

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Estruturas condicionais

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

O que são as estruturas condicionais e como utilizá-las.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1	**If / if-else / elif**
Etapa 2	If aninhado
Etapa 3	If ternário

Etapa 1

If / if-else / elif

O que são?

A estrutura condicional permite o desvio de fluxo de controle, quando determinadas expressões lógicas são atendidas.

If

Para criar uma estrutura condicional simples, composta por um único desvio, podemos utilizar a palavra reservada if. O comando irá testar a expressão lógica, e em caso de retorno verdadeiro as ações presentes no bloco de código do if serão executadas.

Exemplo

```
saldo = 2000.0
saque = float(input("Informe o valor do saque: "))

if saldo >= saque:
    print("Realizando saque!")

if saldo <= saque:
        print("Saldo insuficiente!")
```

If/else

Para criar uma estrutura condicional com dois desvios, podemos utilizar as palavras reservadas if e else. Como sabemos se a expressão lógica testada no if for verdadeira, então o bloco de código do if será executado. Caso contrário o bloco de código do else será executado.

Exemplo

```
saldo = 2000.0
saque = float(input("Informe o valor do saque: "))

if saldo >= saque:
    print("Realizando saque!")
else:
        print("Saldo insuficiente!")
```

If/elif/else

Em alguns cenários queremos mais de dois desvios, para isso podemos utilizar a palavra reservada elif. O elif é composto por uma nova expressão lógica, que será testada e caso retorne verdadeiro o bloco de código do elif será executado. Não existe um número máximo de elifs que podemos utilizar, porém evite criar grandes estruturas condicionais, pois elas aumentam a complexidade do código.

Exemplo

```
opcao = int(input("Informe uma opção: [1] Sacar \n[2] Extrato: "))

if opcao == 1:
        valor = float(input("Informe a quantia para o saque: "))
        # ...
elif opcao == 2:
        print("Exibindo o extrato...")
else:
        sys.exit("Opção inválida")
```

Percurso

~~Etapa 1~~ ~~If / if ... else / elif~~

Etapa 2 **If aninhado**

Etapa 3 If ternário

Etapa 2

If aninhado

If aninhado

Podemos criar estruturas condicionais aninhadas, para isso basta adicionar estruturas if/elif/else dentro do bloco de código de estruturas if/elif/else.

Exemplo

```
if conta_normal:
        if saldo >= saque:
                print("Saque realizado com sucesso!")
        elif saque <= (saldo + cheque_especial):
                print("Saque realizado com uso do cheque especial!")
elif conta_universitaria:
        if saldo >= saque:
                print("Saque realizado com sucesso!")
        else:
                print("Saldo insuficiente!")
```

Percurso

~~Etapa 1~~ ~~If / if ... else / elif~~

~~Etapa 2~~ ~~If aninhado~~

Etapa 3 **If ternário**

Etapa 3

If ternário

If ternário

O if ternário permite escrever uma condição em uma única linha. Ele é composto por três partes, a primeira parte é o retorno caso a expressão retorne verdadeiro, a segunda parte é a expressão lógica e a terceira parte é o retorno caso a expressão não seja atendida.

Exemplo

```
status = "Sucesso" if saldo >= saque else "Falha"

print(f"{status} ao realizar o saque!")
```

Percurso

~~Etapa 1~~ ~~If / if ... else / elif~~

~~Etapa 2~~ ~~If aninhado~~

~~Etapa 3~~ ~~If ternário~~

Hands On!

“Falar é fácil. Mostre-me o código!”

Linus Torvalds

Links Úteis

- https://github.com/guicarvalho/trilha-python-dio

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Estruturas de repetição

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

Conhecer as estruturas de repetição for e while e quando utilizá-las.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1	**O que são estruturas de repetição?**
Etapa 2	Comando for e a função built-in range
Etapa 3	Comando while

Etapa 1

O que são estruturas de repetição

O que são estruturas de repetição?

São estruturas utilizadas para repetir um trecho de código um determinado número de vezes. Esse número pode ser conhecido previamente ou determinado através de uma expressão lógica.

Exemplo sem repetição

```
# Receba um número do teclado e exiba os 2 números seguintes

a = int(input("Informe um número inteiro: "))
print(a)

a += 1
print(a)

a += 1
print(a)
```

Exemplo com repetição

```
# Receba um número do teclado e exiba os 2 números seguintes

a = int(input("Informe um número inteiro: "))
print(a)

repita 2 vezes:
    a += 1
    print(a)
```

Percurso

~~Etapa 1~~	~~O que são estruturas de repetição?~~
Etapa 2	**Comando for e a função built-in range**
Etapa 3	Comando while

Etapa 2

Comando for e a função built-in range

Comando for

O comando for é usado para percorrer um objeto iterável. Faz sentido usar for quando sabemos o número exato de vezes que nosso bloco de código deve ser executado, ou quando queremos percorrer um objeto iterável.

for

```
texto = input("Informe um texto: ")
VOGAIS = "AEIOU"

for letra in texto:
        if letra.upper() in VOGAIS:
                print(letra, end="")

print()  # adiciona uma quebra de linha
```

for/else

```
texto = input("Informe um texto: ")
VOGAIS = "AEIOU"

for letra in texto:
        if letra.upper() in VOGAIS:
                print(letra, end="")
else:
    print()  # adiciona uma quebra de linha
```

Função range

Range é uma função built-in do Python, ela é usada para produzir uma sequência de números inteiros a partir de um ínicio (inclusivo) para um fim (exclusivo). Se usarmos range(i, j) será produzido:

i, i+1, i+2, i+3, ..., j-1.

Ela recebe 3 argumentos: stop (obrigatório), start (opcional) e step opcional.

range

```
# range(stop) -> range object
# range(start, stop[, step]) -> range object

list(range(4))
>>> [0, 1, 2, 3]
```

Utilizando range com for

```
for numero in range(0, 11):
    print(numero, end=" ")

>>> 0 1 2 3 4 5 6 7 8 9 10


# exibindo a tabuada do 5
for numero in range(0, 51, 5):
    print(numero, end=" ")

>>> 0 5 10 15 20 25 30 35 40 45 50
```

Percurso

~~Etapa 1~~	~~O que são estruturas de repetição?~~
~~Etapa 2~~	~~Comando for e a função built-in range~~
Etapa 3	**Comando while**

Etapa 3

Comando while

Comando while

O comando while é usado para repetir um bloco de código várias vezes. Faz sentido usar while quando não sabemos o número exato de vezes que nosso bloco de código deve ser executado.

while

```
opcao = -1

while opcao != 0:
    opcao = int(input("[1] Sacar \n[2] Extrato \n[0] Sair \n: "))

    if opcao == 1:
        print("Sacando...")
    elif opcao == 2:
        print("Exibindo o extrato...")
```

while/else

```
opcao = -1

while opcao != 0:
    opcao = int(input("[1] Sacar \n[2] Extrato \n[0] Sair \n: "))

    if opcao == 1:
        print("Sacando...")
    elif opcao == 2:
        print("Exibindo o extrato...")
else:
        print("Obrigado por usar nosso sistema bancário, até logo!")
```

Percurso

~~Etapa 1~~	~~O que são estruturas de repetição?~~
~~Etapa 2~~	~~Comando for e a função built-in range~~
~~Etapa 3~~	~~Comando while~~

Hands On!

“Falar é fácil. Mostre-me o código!”

Linus Torvalds

Links Úteis

- https://github.com/guicarvalho/trilha-python-dio

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

String e fatiamento

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

Conhecer métodos úteis para manipular objetos do tipo string, como interpolar valores de variáveis e entender como funciona o fatiamento.

Pré-requisitos

- Python 3
- VSCode

Percurso

Etapa 1	**Conhecendo métodos úteis da classe string**
Etapa 2	Interpolação de variáveis
Etapa 3	Fatiamento de string
Etapa 4	String múltiplas linhas

Etapa 1

Conhecendo métodos úteis da classe string

Introdução

A classe String do Python é famosa por ser rica em métodos e possuir uma interface muito fácil de trabalhar.

Em algumas linguagens manipular sequências de caracteres não é um trabalho trivial, porém, em Python esse trabalho é muito simples.

Maiúscula, minúscula e título

```
curso = "pYtHon"

print(curso.upper())
>>> PYTHON

print(curso.lower())
>>> python

print(curso.title())
>>> Python
```

Eliminando espaços em branco

```
curso = "   Python "

print(curso.strip())
>>> "Python"

print(curso.lstrip())
>>> "Python "

print(curso.rstrip())
>>> "   Python"
```

Junções e centralização

```
curso = "Python"

print(curso.center(10, "#"))
>>> "##Python##"

print(".".join(curso))
>>> "P.y.t.h.o.n"
```

Percurso

~~Etapa 1~~	~~Conhecendo métodos úteis da classe string~~
Etapa 2	**Interpolação de variáveis**
Etapa 3	Fatiamento de string
Etapa 4	String múltiplas linhas

Etapa 2

Interpolação de variáveis

Introdução

Em Python temos 3 formas de interpolar variáveis em strings, a primeira é usando o sinal %, a segunda é utilizando o método format e a última é utilizando f strings.

A primeira forma não é atualmente recomendada e seu uso em Python 3 é raro, por esse motivo iremos focar nas 2 últimas.

Old style %

```
nome = "Guilherme"
idade = 28
profissao = "Progamador"
linguagem = "Python"


print("Olá, me chamo %s. Eu tenho %d anos de idade, trabalho como %s e
estou matriculado no curso de %s." % (nome, idade, profissao, linguagem))

>>> Olá, me chamo Guilherme. Eu tenho 28 anos de idade, trabalho como
Progamador e utilizo e estou matriculado no curso de Python.
```

Método format

```
nome = "Guilherme"
idade = 28
profissao = "Programador"
linguagem = "Python"

print("Olá, me chamo {}. Eu tenho {} anos de idade, trabalho como {} e
estou matriculado no curso de {}.".format(nome, idade, profissao,
linguagem))

print("Olá, me chamo {3}. Eu tenho {2} anos de idade, trabalho como {1} e
estou matriculado no curso de {0}.".format(linguagem, profissao, idade,
nome))
```

Método format

```
print("Olá, me chamo {nome}. Eu tenho {idade} anos de idade, trabalho como
{profissao} e estou matriculado no curso de
{linguagem}.".format(nome=nome, idade=idade, profissao=profissao,
linguagem=linguagem))

print("Olá, me chamo {nome}. Eu tenho {idade} anos de idade, trabalho como
{profissao} e estou matriculado no curso de
{linguagem}.".format(**pessoa))

>>> Olá, me chamo Guilherme. Eu tenho 28 anos de idade, trabalho como
Progamador e estou matriculado no curso de Python.
```

f-string

```
nome = "Guilherme"
idade = 28
profissao = "Programador"
linguagem = "Python"

print(f"Olá, me chamo {nome}. Eu tenho {idade} anos de idade, trabalho
como {profissao} e estou matriculado no curso de {linguagem}.")

>>> Olá, me chamo Guilherme. Eu tenho 28 anos de idade, trabalho como
Progamador e utilizo e estou matriculado no curso de Python.
```

Formatar strings com f-string

```
PI = 3.14159

print(f"Valor de PI: {PI:.2f}")
>>> "Valor de PI: 3.14"

print(f"Valor de PI: {PI:10.2f}")
>>> "Valor de PI:       3.14"
```

Percurso

~~Etapa 1~~	~~Conhecendo métodos úteis da classe string~~
~~Etapa 2~~	~~Interpolação de variáveis~~
Etapa 3	**Fatiamento de string**
Etapa 4	String múltiplas linhas

Etapa 3

Fatiamento de string

Introdução

Fatiamento de strings é uma técnica utilizada para retornar substrings (partes da string original), informando inicio (start), fim (stop) e passo (step): [start: stop[, step]].

Fatiamento

```
nome = "Guilherme Arthur de Carvalho"

nome[0]
>>> "G"

nome[:9]
>>> "Guilherme"

nome[10:]
>>> "Arthur de Carvalho"

nome[10:16]
>>> "Arthur"

nome[10:16:2]
>>> "Atu"

nome[:]
>>> "Guilherme Arthur de Carvalho"

nome[::-1]
>>> "ohlavraC ed ruhtrA emrehliuG"
```

Percurso

~~Etapa 1~~	~~Conhecendo métodos úteis da classe string~~
~~Etapa 2~~	~~Interpolação de variáveis~~
~~Etapa 3~~	~~Fatiamento de string~~
Etapa 4	**String múltiplas linhas**

Etapa 4

String multiplas linhas

Introdução

Strings de múltiplas linhas são definidas informando 3 aspas simples ou duplas durante a atribuição. Elas podem ocupar várias linhas do código, e todos os espaços em branco são incluídos na string final.

Strings triplas

```
nome = "Guilherme"

mensagem = f"""
Olá meu nome é {nome},
Eu estou aprendendo Python
"""
>>>

Olá meu nome é Guilherme,
Eu estou aprendendo Python
```

Strings triplas

```
nome = "Guilherme"

mensagem = f'''
   Olá meu nome é {nome},
 Eu estou aprendendo Python.
     Essa mensagem tem diferentes recuos.
'''
>>>

   Olá meu nome é Guilherme,
 Eu estou aprendendo Python.
     Essa mensagem tem diferentes recuos.
```

Percurso

~~Etapa 1~~	~~Conhecendo métodos úteis da classe string~~
~~Etapa 2~~	~~Interpolação de variáveis~~
~~Etapa 3~~	~~Fatiamento de string~~
~~Etapa 4~~	~~String múltiplas linhas~~

Hands On!

“Falar é fácil. Mostre-me o código!”

Linus Torvalds

Links Úteis

- https://github.com/guicarvalho/trilha-python-dio
- https://docs.python.org/pt-br/3/library/string.html
- https://docs.python.org/pt-br/3/library/stdtypes.html#textseq

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)

Desafio: Criando um sistema bancário

Guilherme Arthur de Carvalho

Analista de sistemas

@decarvalhogui

Objetivo Geral

Criar um sistema bancário com as operações: sacar, depositar e visualizar extrato.

Desafio

Fomos contratados por um grande banco para desenvolver o seu novo sistema. Esse banco deseja modernizar suas operações e para isso escolheu a linguagem Python. Para a primeira versão do sistema devemos implementar apenas 3 operações: depósito, saque e extrato.

Operação de depósito

Deve ser possível depositar valores positivos para a minha conta bancária. A v1 do projeto trabalha apenas com 1 usuário, dessa forma não precisamos nos preocupar em identificar qual é o número da agência e conta bancária. Todos os depósitos devem ser armazenados em uma variável e exibidos na operação de extrato.

Operação de saque

O sistema deve permitir realizar 3 saques diários com limite máximo de R$ 500,00 por saque. Caso o usuário não tenha saldo em conta, o sistema deve exibir uma mensagem informando que não será possível sacar o dinheiro por falta de saldo. Todos os saques devem ser armazenados em uma variável e exibidos na operação de extrato.

Operação de extrato

Essa operação deve listar todos os depósitos e saques realizados na conta. No fim da listagem deve ser exibido o saldo atual da conta. Se o extrato estiver em branco, exibir a mensagem: Não foram realizadas movimentações.

Os valores devem ser exibidos utilizando o formato R$ xxx.xx, exemplo:

1500.45 = R$ 1500.45

Dúvidas?

> Fórum/Artigos

> Comunidade Online (Discord)