Introdução a Algoritmos

Conceitos iniciais

O QUE É UM ALGORITMO?

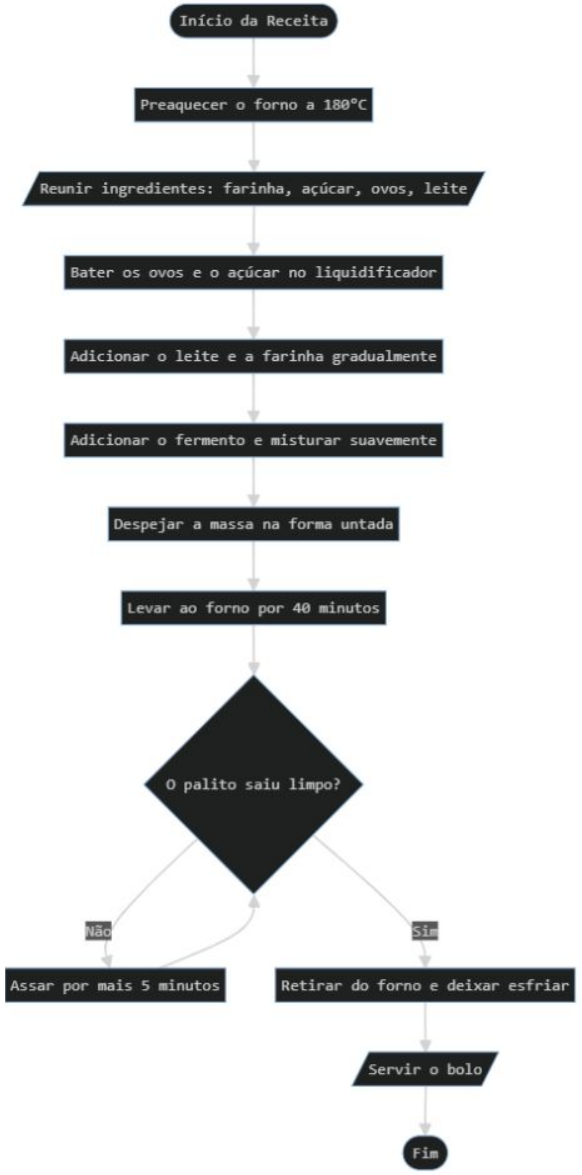
Definição Lógica

Uma sequência finita de instruções bem definidas e não ambíguas, cada uma das quais pode ser executada mecanicamente num período de tempo finito.

No Cotidiano

Estamos cercados por algoritmos: desde uma receita de bolo até o GPS do seu celular. É o "passo a passo" para atingir um objetivo.

O QUE É UM ALGORITMO?



O FLUXO DO ALGORITMO

Entrada: Os dados iniciais necessários para resolver o problema.

Processamento: As operações lógicas e matemáticas aplicadas aos dados.

Saída: O resultado final esperado após a execução.

ALGORITMO VS CÓDIGO

Algoritmo

É a sequência lógica e finita de passos para resolver um problema, independente de linguagem.

Código

É a tradução do algoritmo para uma **SINTAXE** específica que o computador entende. Enquanto o algoritmo é a ideia, o código é a execução real usando comandos de alguma linguagem de programação

EXEMPLO DE SINTAXE NO PYTHON


```python
# Entrada de dados
numero = int(input("Digite um número: "))

# Lógica do Algoritmo (Decisão)
if numero % 2 == 0:
    print(f"O número {numero} é PAR")
else:
    print(f"O número {numero} é ÍMPAR")
```

EXEMPLO DE SINTAXE NO JAVA


```java
import java.util.Scanner;

public class VerificadorParImpar {
    public static void main(String[] args) {
        Scanner leitor = new Scanner(System.in);

        // Entrada de dados
        System.out.print("Digite um número: ");
        int numero = leitor.nextInt();

        // Lógica do Algoritmo (Decisão)
        if (numero % 2 == 0) {
            System.out.println("O número é PAR");
        } else {
            System.out.println("O número é ÍMPAR");
        }

        leitor.close();
    }
}
```

POR QUE TER MAIS DE UMA LINGUAGEM DE PROGRAMAÇÃO?

Especialização por Domínio

Diferentes problemas exigem ferramentas diferentes. O JavaScript foi criado para rodar dentro de navegadores e tornar sites interativos; o R foi feito para estatística; e o C para sistemas que precisam de velocidade extrema e controle de hardware

Equilíbrio entre Produtividade e Performance

Existe uma **troca (trade-off)** constante. Linguagens como Python priorizam a velocidade de escrita e facilidade humana (sintaxe limpa), enquanto linguagens como C++ ou Rust priorizam a velocidade de execução do computador, mesmo que a sintaxe seja mais complexa para o programador.

Python

Python

INTRODUÇÃO AO PYTHON

Criador: Programador holandês Guido van Rossum

Ano de Lançamento: O desenvolvimento começou no final dos anos 80, mas a versão 0.9.0 foi publicada oficialmente em fevereiro de 1991.

O Nome: Ao contrário do que muitos pensam, o nome não foi inspirado na cobra (Embora ela seja o símbolo hoje), mas sim no grupo de comédia britânico Monty Python, do qual Guido era um grande fã.

Disponível em: Python,Tipe Data dan Variabel

CARACTERÍSTICAS

Sintaxe Limpa e Intuitiva: O código Python parece muito com o inglês escrito. Ele utiliza a identação (espaçamentos) para organizar o código, o que obriga o programador a escrever um código visualmente organizado.

Linguagem Interpretada: O código não precisa ser transformado em um arquivo executável pesado antes de rodar; ele é lido e executado linha por linha por um "interpretador", o que facilita o teste rápido de ideias.

CARACTERÍSTICAS

Multiparadigma: Você pode programar de várias formas: orientada a objetos, funcional ou procedural. Ela se adapta ao seu estilo de resolução de problemas.

Tipagem Dinâmica e Forte: Você não precisa dizer que uma variável é um número; o Python descobre sozinho (dinâmica). No entanto, ele não permite que você some um texto com um número sem conversão prévia (forte), o que evita erros bobos.

POR QUE PYTHON?

Simplicidade

Sintaxe próxima à linguagem humana, facilitando o aprendizado e a leitura.

Ciência de Dados

Ecossistema robusto com Pandas, NumPy, Scikit-learn e Matplotlib.

Comunidade

Vasta documentação e suporte global para qualquer desafio técnico.

POR QUE PYTHON?

Disponível em: [20+ Python Libraries for Data Science Professionals [2025 Edition] – Quantum™ Ai Labs]

Primeiros Comandos No Python

Entendendo como manipular os comandos fundamentais do Python

OLÁ MUNDO!

Vamos escrever o primeiro comando Python. O **print(‘’)** serve para você imprimir coisas na tela.

Código

```
[1]: print('olá mundo!')
```

Saída

```
olá mundo!
```

O “Olá mundo” é o passo inicial quando estamos programando, já que é o primeiro comando que é aprendido.

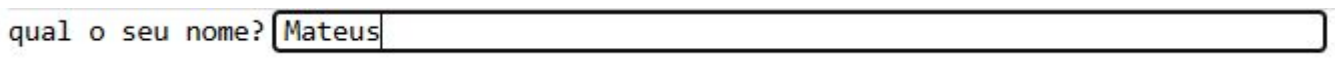
RECEBENDO INFORMAÇÕES

Para que o Python receba informações, utiliza-se o comando **input(‘’)**

Código

```
[2]: input('qual o seu nome?')
```

Saída



Note que o texto que passamos ‘Qual o seu nome?’ irá aparecer do lado da saída e haverá uma caixinha de diálogo para você digitar qualquer coisa.

TIPOS PRIMITIVOS

Tipos primitivos são as categorias mais básicas de dados que uma linguagem de programação pode manipular, eles podem ser:

Inteiros (int): Números sem casas decimais. Exemplo: estoque = 50 | temperatura = -10

- **Ponto Flutuante (float):** Números reais (com casas decimais). Importante notar o uso do ponto como separador. Exemplo: valor = 99.90 | altura = 1.82
- **Strings (str):** Dados alfanuméricos (textos). Sempre delimitados por aspas. Exemplo: usuario = "Ana_2024" | email = 'contato@empresa.com'
- **Booleanos (bool):** Valores lógicos de verdadeiro ou falso. Importante: em Python, começam com letra maiúscula. Exemplo: esta_chovendo = True | fim_de_jogo = False

TIPOS PRIMITIVOS

No Python, podemos realizar as manipulações de cada tipo primitivo, por exemplo com os tipos inteiros (int) e ponto flutuante (float), podemos realizar operações aritméticas.

Código

```
[3]: print('soma de 1 + 1 = ',1 + 1)
     print('diferença de 2 - 1 = ',2 - 1)
     print('diferença de 4 - 5 = ',4 - 5)
     print('multiplicação de 2 x 4 = ',2 * 4)
     print('divisão de 2/4 = ', 2/4)
     print('resto de divisão de 4/2 = ', 4%2)
     print('potenciação de 2 elevado a 8 = ', 2**8)
```

Saída

```
soma de 1 + 1 =  2
diferença de 2 - 1 =  1
diferença de 4 - 5 =  -1
multiplicação de 2 x 4 =  8
divisão de 2/4 =  0.5
resto de divisão de 4/2 =  0
potenciação de 2 elevado a 8 =  256
```

Note que no comando print(‘’) podemos colocar mais um comando para imprimir na tela, nesse caso usamos a combinação de uma string (por exemplo ‘soma de 1 + 1 =’) com um inteiro (int).

VARIÁVEIS

Mas como armazenar esses tipos de forma que eu possa usar o resultado em outros comandos?

Imagine que queira saber o nome e o sobrenome de uma pessoa para mandar uma mensagem de boas. Como eu faria isso no Python? A resposta simples é: **Variáveis**.

- **O que são variáveis?** Uma variável é um espaço na memória do computador reservado para armazenar um dado que pode ser alterado durante a execução do programa.

VARIÁVEIS

Para criar uma variável no Python basta criar um nome, colocar o símbolo de = e no lado direito, adicionar o valor que você quer armazenar.

Código

```
[12]: nome = 'Mateus'
      sobrenome = 'Silva'

      print('Seja bem vindo', nome, sobrenome)
```

Saída

```
Seja bem vindo Mateus Silva
```

Perceba que foram criados duas variáveis: nome e sobrenome. A primeira armazena o nome da pessoa e a segunda seu sobrenome, para que no fim possamos colocar a mensagem de boas vindas.

VARIÁVEIS

Podemos realizar operações aritméticas com as variáveis.

Código

```
[14]: idade_1 = 18
      idade_2 = 20

      print('A soma das idades são', idade_1 + idade_2)
      print('A diferença das idades são', idade_1 - idade_2)
      print('A multiplicação das idades são', idade_1 * idade_2)
      print('A divisão das idades são', idade_1 / idade_2)
```

Saída

```
A soma das idades são 38
A diferença das idades são -2
A multiplicação das idades são 360
A divisão das idades são 0.9
```

Note que ainda valem as mesmas propriedades aritméticas para as variáveis.

VARIÁVEIS

Mas o que acontece se eu somar um inteiro ou float com uma string?

Código

```
[15]: idade = 18
      nome = 'Mateus'

      print('O resultado será...', idade + nome)
```

Saída

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[15], line 4
      1 idade = 18
      2 nome = 'Mateus'
----> 4 print('O resultado será...', idade + nome)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Um erro!! Basicamente o Python é uma linguagem de tipagem forte, ou seja ele não permite somar tipos diferentes de dados!

VARIÁVEIS

E o que podemos fazer? Podemos converter a variável **idade** para String!

Código

```
[16]: idade = 18
      nome = 'Mateus'

      print('O resultado será...', str(idade) + nome)
```

Saída

```
O resultado será... 18Mateus
```

Veja que o comando **str()** realiza uma converção de tipo, estou pegando a minha variável idade, adicionando dentro do comando **str(idade),** somando com o nome (chamamos de concatenar quando somamos duas Strings), para que no fim pudéssemos imprimir o resultado na tela.

VARIÁVEIS

Como o Python tem a característica de ser uma linguagem dinâmica, conseguimos alterar o tipo das variáveis, porém com algumas ressalvas:

- O comando **int()** converte os tipos primitivos **bool** e **float** para inteiro. Mas converte apenas Strings numéricas como por exemplo '10', '-2'.
- O comando **float()** converte os tipos primitivos **bool** e **int** para float. Mas converte apenas Strings numéricas como por exemplo '-22.3', '11.4'.
- O comando **str()** converte todos os tipos para **String**
- O comando **bool()** converte todos os tipos para **booleano**

VARIÁVEIS

Cuidado! Temos algumas limitações quanto ao nome das variáveis enquanto estamos a definindo.

- **Não pode começar com números:** Use nota1, mas nunca 1nota.
- **Não pode ter espaços:** Use o padrão **snake_case** (letras minúsculas separadas por underline). Exemplo: nome_do_usuario.
- **Sensível a maiúsculas:** Idade e idade são consideradas duas variáveis diferentes (Case Sensitive).
- **Sem caracteres especiais:** Não use acentos, cedilha ou símbolos como @, !, $.

OPERADORES LÓGICOS

E se quiséssemos testar condições? Por exemplo, saber se uma pessoa é de maior ou se trabalha como cientista de dados? Como faríamos isso no Python? A resposta é **expressões booleanas**!

- Uma expressão booleana é qualquer afirmação ou comparação que resulta em apenas dois valores possíveis: Verdadeiro (True) ou Falso (False).

OPERADORES LÓGICOS

São formadas por um conjunto de operadores de comparação e operadores lógicos:

- Operadores de comparação:
  - `==` : Igual a.
  - `!=` : Diferente de.
  - `>` e `<` : Maior que e Menor que.
  - `>=` e `<=` : Maior ou igual e Menor ou igual.

OPERADORES LÓGICOS

São formadas por um conjunto de operadores de comparação e operadores lógicos:

- Operadores lógicos:
  - and: Une duas comparações. Só é verdade se as duas forem True.
  - or: Une duas comparações. É verdade se pelo menos uma for True.
  - not: Inverte o resultado.

OPERADORES LÓGICOS

Vamos ver na prática como funciona os operadores booleanos

Código

```
[9]: idade = 18

print('A idade é maior que 18 anos?', idade > 18)
print('A idade é menor que 18 anos?', idade < 18)
print('A idade é igual a 18 anos?', idade == 18)
print('A idade é maior ou igual a 18 anos?', idade >= 18)
print('A idade é menor ou igual a 18 anos?', idade <= 18)
```

Saída

```
A idade é maior que 18 anos? False
A idade é menor que 18 anos? False
A idade é igual a 18 anos? True
A idade é maior ou igual a 18 anos? True
A idade é menor ou igual a 18 anos? True
```

Perceba que cada condição retornará apenas dois valores: True ou False, ou seja o tipo booleano.

OPERADORES LÓGICOS

E se quiséssemos testar mais de uma condição? E se quiséssemos testar se ele tem 18 anos E se ele é cientista de dados? Como faríamos? Com os operadores lógicos!

Código

```
[13]: idade = 18
cargo = 'cientista de dados'


print('A idade é igual a 18 anos E é um cientista de dados?', idade == 18 and cargo == 'cientista de dados')
print('A idade é igual a 18 anos OU é um cientista de dados?', idade == 18 or cargo == 'cientista de dados')
print('NÃO é um cientista de dados?', not cargo == 'cientista de dados')
```

Saída

```
A idade é igual a 18 anos E é um cientista de dados? True
A idade é igual a 18 anos OU é um cientista de dados? True
NÃO é um cientista de dados? False
```

Note que com o operador lógico **and**, uma sentença será verdadeira apenas se as duas condições a serem testadas forem de fato verdadeiras.

OPERADORES LÓGICOS

E se quiséssemos testar mais de uma condição? E se quiséssemos testar se ele tem 18 anos E se ele é cientista de dados? Como faríamos? Com os operadores lógicos!

Código

```
[13]: idade = 18
cargo = 'cientista de dados'


print('A idade é igual a 18 anos E é um cientista de dados?', idade == 18 and cargo == 'cientista de dados')
print('A idade é igual a 18 anos OU é um cientista de dados?', idade == 18 or cargo == 'cientista de dados')
print('NÃO é um cientista de dados?', not cargo == 'cientista de dados')
```

Saída

```
A idade é igual a 18 anos E é um cientista de dados? True
A idade é igual a 18 anos OU é um cientista de dados? True
NÃO é um cientista de dados? False
```

Com o operador **or** é diferente, pelo menos uma das condições precisam ser verdadeiras para que a sentença retorne **True.**

OPERADORES LÓGICOS

E se quiséssemos testar mais de uma condição? E se quiséssemos testar se ele tem 18 anos E se ele é cientista de dados? Como faríamos? Com os operadores lógicos!

Código

```
[13]: idade = 18
cargo = 'cientista de dados'


print('A idade é igual a 18 anos E é um cientista de dados?', idade == 18 and cargo == 'cientista de dados')
print('A idade é igual a 18 anos OU é um cientista de dados?', idade == 18 or cargo == 'cientista de dados')
print('NÃO é um cientista de dados?', not cargo == 'cientista de dados')
```

Saída

```
A idade é igual a 18 anos E é um cientista de dados? True
A idade é igual a 18 anos OU é um cientista de dados? True
NÃO é um cientista de dados? False
```

O operador **not** tem uma funcionalidade diferente, ele converte qualquer resultado de uma condição em seu inverso, por exemplo no ultimo condicional, era pra estar **True**, mas com o **not** ele retornou **False.**

ESTRUTURAS DE CONDIÇÃO

Mas o que eu posso fazer com esses operadores lógicos? E se quiséssemos mandar uma mensagem de boas vindas só para os cientistas de dados com 18 anos de idade? a resposta é estruturas de condição.

Código

```
[14]: idade = 18
cargo = 'cientista de dados'

if idade == 18 and cargo ==  'cientista de dados':
    print("Seja bem vindo!!")
else:
    print("ERRO!!")
```

Saída

```
Seja bem vindo!!
```

Para fazer com que um comando específico seja rodado basta usar o comando **if** para testar uma condição e se ela for falsa, o comando **else** rodará outro comando.

ESTRUTURAS DE CONDIÇÃO

Mas o que eu posso fazer com esses operadores lógicos? E se quiséssemos mandar uma mensagem de boas vindas só para os cientistas de dados com 18 anos de idade? a resposta é estruturas de condição.

Código

```
[14]: idade = 18
cargo = 'cientista de dados'

if idade == 18 and cargo ==  'cientista de dados':
    print("Seja bem vindo!!")
else:
    print("ERRO!!")
```

Saída

```
Seja bem vindo!!
```

Note que o **print("Seja bem vindo!!")** está dentro do comando **if**, isso é chamado de **indentação**, se você quer que um comando só execute se a condição do **if** for verdadeira, basta adicionar espaços vazios para que o comando fique dentro do **if**

ESTRUTURAS DE CONDIÇÃO

Podemos adicionar mais condicionais para testar, usando o comando **elif**.

Código

```
[17]: idade = 18
cargo = 'cientista de dados'

if idade == 18 and cargo ==  'cientista de dados':
    print("Seja bem vindo!!")
elif idade > 18 and cargo == 'analista de dados':
    print('olá tudo bem?')
elif idade > 18 and cargo == 'analista de bi':
    print('bom dia')
else:
    print("ERRO!!")
```

Saída

```
Seja bem vindo!!
```

perceba que podemos fazer vários **elif** de forma que, caso uma condição não seja satisfeita, o python irá testar as demais, caso não encontre, o **else** irá sempre ser executado, por isso que ele **DEVE** ser o último comando dos condicionais aninhados.

ESTRUTURAS DE REPETIÇÃO

Podemos repetir comandos mais de uma vez? A resposta é sim, utilizando o for.

Código

```
[18]: for i in range(1,10,1):
          print(i)
```

Saída

```
1
2
3
4
5
6
7
8
9
```

Vamos analisar com calma a sintaxe do for.

ESTRUTURAS DE REPETIÇÃO

Podemos repetir comandos mais de uma vez? A resposta é sim, utilizando o for.

Código

```
[18]: for i in range(1,10,1):
          print(i)
```

Saída

```
1
2
3
4
5
6
7
8
9
```

O comando range() irá criar um intervalo numérico que começa do 1 (primeiro parâmetro passado) até o 10 (segundo parâmetro passado), de forma que ele vá sempre acrescentando 1 unidade(terceiro parâmetro passado), isto é caminhando do 1 até o 10 de um a um.

ESTRUTURAS DE REPETIÇÃO

Podemos repetir comandos mais de uma vez? A resposta é sim, utilizando o for.

Código

```
[18]: for i in range(1,10,1):
          print(i)
```

Saída

```
1
2
3
4
5
6
7
8
9
```

O i serve como uma variável que armazenará cada valor do intervalo gerado pelo range, de forma que a cada repetição (loop ou iteração) ele mudará esse valor com base no terceiro parâmetro (caminhando de 1 a 1).

ESTRUTURAS DE REPETIÇÃO

Podemos repetir comandos mais de uma vez? A resposta é sim, utilizando o for.

Código

```
[18]: for i in range(1,10,1):
          print(i)
```

Saída

```
1
2
3
4
5
6
7
8
9
```

Todo o comando que estiver “dentro” do comando **for**, irá ser repetido conforme especificado.

Nesse caso o comando **print(i)** irá se repetir 9 vezes (intervalo fechado no início e fechado no fim).

ESTRUTURAS DE REPETIÇÃO

Agora note o seguinte:

Código

```
[19]: for i in range(5):
          print(i)
```

Saída

```
0
1
2
3
4
```

Se eu colocar um único valor, como por exemplo o 5, o for irá fazer 5 repetições onde começa do 0 até 4.

ESTRUTURAS DE REPETIÇÃO

Será que tem como realizar um loop de forma que fique repetindo até uma condição ser satisfeita? a resposta é sim, com o **while**.

Código

```
[20]: contador = 0

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

Saída

```
0
1
2
3
4
5
6
7
8
9
```

note que foi criado uma variável chamada contador, quando ela entra no loop, é testado a condição: contador é menor que 10? se não for, o comando print vai ser executado seguido pelo comando **contador = contador + 1**, esse comando irá acrescentar uma unidade na variável, até que a condição seja satisfeita e o loop seja quebrado.

LISTAS

Como eu poderia fazer para armazenar mais de um valor em uma variável? Com as listas!

Código

```
[21]: lista_idades = [18, 20 ,25, 17, 34, 45, 50]
      print(lista_idades)
```

Saída

```
[18, 20, 25, 17, 34, 45, 50]
```

Uma lista é uma coleção de valores que pode conter tipos distintos (String, float, booleano, etc). Basta usar a sintaxe de colchetes ([]) separando cada elemento com vígula (,)

LISTAS

Agora podemos realizar manipulações mais avançadas com as listas, por exemplo percorrer com o loop for.

Código

```
[23]: lista_idades = [18, 20 ,25, 17, 34, 45, 50]

for i in lista_idades:
    if i > 18 and i <= 30:
        print("Você é mais velho")
    elif i < 18:
        print("você é mais novo")
    else:
        print("você está ficando velho")
```

Saída

```
você está ficando velho
Você é mais velho
Você é mais velho
você é mais novo
você está ficando velho
você está ficando velho
você está ficando velho
```

Para fazer o **for** percorrer elemento a elemento da lista, basta trocar o comando **range** para o **in nome_da_lista**. Dentro do **for** podemos realizar mais manipulações, com o **if** podemos testar cada idade e executar cada comando dependendo do condicional.

LISTAS

Podemos realizar mais manipulações nas listas, mas usando comando internos.

Código

```
[24]: lista_nomes = ["Mateus","Marcus","Lucas", "Judas"] # cria uma lista de nomes
nova_lista_nomes = ["Paulo","Pedro"] # cria mais uma lista de nomes

lista_nomes.append("João") # comando para inserir um novo elemento no final da lista
lista_nomes.remove("Judas") # comando para remover um elemento
lista_nomes.extend(nova_lista_nomes) # comando para juntar duas listas

print(lista_nomes)
print(len(lista_nomes))
```

Saída

```
['Mateus', 'Marcus', 'Lucas', 'João', 'Paulo', 'Pedro']
6
```

o comando com # são comentários no código que podemos utilizar para nos auxiliar na leitura.

LISTAS

Podemos realizar mais manipulações nas listas, mas usando comando internos.

Código

```
[24]: lista_nomes = ["Mateus","Marcus","Lucas", "Judas"] # cria uma lista de nomes
nova_lista_nomes = ["Paulo","Pedro"] # cria mais uma lista de nomes

lista_nomes.append("João") # comando para inserir um novo elemento no final da lista
lista_nomes.remove("Judas") # comando para remover um elemento
lista_nomes.extend(nova_lista_nomes) # comando para juntar duas listas

print(lista_nomes)
print(len(lista_nomes))
```

Saída

```
['Mateus', 'Marcus', 'Lucas', 'João', 'Paulo', 'Pedro']
6
```

O comando .append() e os demais são feitos nas próprias listas. O append irá adicionar um novo elemento a lista.

LISTAS

Podemos realizar mais manipulações nas listas, mas usando comando internos.

Código

```
[24]: lista_nomes = ["Mateus","Marcus","Lucas", "Judas"] # cria uma lista de nomes
nova_lista_nomes = ["Paulo","Pedro"] # cria mais uma lista de nomes

lista_nomes.append("João") # comando para inserir um novo elemento no final da lista
lista_nomes.remove("Judas") # comando para remover um elemento
lista_nomes.extend(nova_lista_nomes) # comando para juntar duas listas

print(lista_nomes)
print(len(lista_nomes))
```

Saída

```
['Mateus', 'Marcus', 'Lucas', 'João', 'Paulo', 'Pedro']
6
```

O comando .remove() irá remover um elemento em específico da lista.

LISTAS

Podemos realizar mais manipulações nas listas, mas usando comando internos.

Código

```
[24]: lista_nomes = ["Mateus","Marcus","Lucas", "Judas"] # cria uma lista de nomes
nova_lista_nomes = ["Paulo","Pedro"] # cria mais uma lista de nomes

lista_nomes.append("João") # comando para inserir um novo elemento no final da lista
lista_nomes.remove("Judas") # comando para remover um elemento
lista_nomes.extend(nova_lista_nomes) # comando para juntar duas listas

print(lista_nomes)
print(len(lista_nomes))
```

Saída

```
['Mateus', 'Marcus', 'Lucas', 'João', 'Paulo', 'Pedro']
6
```

O comando .extend vai juntar os elementos de uma lista na outra e o comando len(lista_nomes) irá nos dar a quantidade de elementos em uma lista.

LISTAS

Como fazer para pegar algum elemento em específico da lista? Utilizaremos o fatiamento (slicing)

Código

```
[28]: lista_nomes = ["Mateus","Marcus","Lucas", "Judas"] # cria uma lista de nomes

print(lista_nomes[0]) # pega o primeiro elemento da lista
print(lista_nomes[1]) # pega o segundo elemento da lista
print(lista_nomes[2]) # pega o terceiro elemento da lista
print(lista_nomes[3]) # pega o quarto elemento da lista
print(lista_nomes[-1]) # pega o último elemento da lista
print(lista_nomes[0:2]) # pega do primeiro elemento até o segundo
```

Saída

```
Mateus
Marcus
Lucas
Judas
Judas
['Mateus', 'Marcus']
```

O processo de indexação do python sempre começa a partir do 0, então se quisermos o primeiro elemento “Mateus” precisamos colocar a sintaxe lista_nomes[0].

LISTAS

Como fazer para pegar algum elemento em específico da lista? Utilizaremos o fatiamento (slicing)

Código

```
[28]: lista_nomes = ["Mateus","Marcus","Lucas", "Judas"] # cria uma lista de nomes

print(lista_nomes[0]) # pega o primeiro elemento da lista
print(lista_nomes[1]) # pega o segundo elemento da lista
print(lista_nomes[2]) # pega o terceiro elemento da lista
print(lista_nomes[3]) # pega o quarto elemento da lista
print(lista_nomes[-1]) # pega o último elemento da lista
print(lista_nomes[0:2]) # pega do primeiro elemento até o segundo
```

Saída

```
Mateus
Marcus
Lucas
Judas
Judas
['Mateus', 'Marcus']
```

Note que há atalhos também, podemos pegar o último elemento colocando o fatiamento [-1].

LISTAS

Como fazer para pegar algum elemento em específico da lista? Utilizaremos o fatiamento (slicing)

Código

```
[28]: lista_nomes = ["Mateus","Marcus","Lucas", "Judas"] # cria uma lista de nomes

print(lista_nomes[0]) # pega o primeiro elemento da lista
print(lista_nomes[1]) # pega o segundo elemento da lista
print(lista_nomes[2]) # pega o terceiro elemento da lista
print(lista_nomes[3]) # pega o quarto elemento da lista
print(lista_nomes[-1]) # pega o último elemento da lista
print(lista_nomes[0:2]) # pega do primeiro elemento até o segundo
```

Saída

```
Mateus
Marcus
Lucas
Judas
Judas
['Mateus', 'Marcus']
```

Podemos também selecionar um pedaço da lista, especificando o começo da posição (0) até o fim que queremos (lembrando que trata-se de um intervalo aberto no final). O comando [0:2] irá selecionar o elemento de posição 0 e 1, mas não o 2.

DICIONÁRIOS

Dicionários são uma outra estrutura de dados para armazenamento, mas dessa vez podemos armazenar nossos dados de forma que se assemelhe a um dicionário real, onde temos uma chave para identificação da coleção de dados em específico

Código

```
[34]: funcionario = {'nome' : 'Mateus', # sempre criar com chaves o dicionário ({})
                     'idade': 24, # separar a chave do valor com ( : )
                     'cargo': 'professor'}


print(funcionario) # imprimindo o dicionario
print(funcionario['nome']) # imprimindo o valor associado a chave nome
print(funcionario['cargo']) # imprimindo o valor associado a chave cargo
print(funcionario.keys()) # imprimindo uma lista das chaves
print(funcionario.values()) # imprimindo uma lista dos valores
```

Saída

```
{'nome': 'Mateus', 'idade': 24, 'cargo': 'professor'}
Mateus
professor
dict_keys(['nome', 'idade', 'cargo'])
dict_values(['Mateus', 24, 'professor'])
```

DICIONÁRIOS

Para criar um dicionário, basta usar chaves ({}) e separar cada chave e valor com (:). Note que para pesquisar o nome do funcionário no dicionário, utilizamos quase a mesma sintaxe de pesquisa nas listas, mas utilizando a chave associada ['nome'].

Código

```
[34]: funcionario = {'nome' : 'Mateus', # sempre criar com chaves o dicionário ({})
                     'idade': 24, # separar a chave do valor com ( : )
                     'cargo': 'professor'}


print(funcionario) # imprimindo o dicionario
print(funcionario['nome']) # imprimindo o valor associado a chave nome
print(funcionario['cargo']) # imprimindo o valor associado a chave cargo
print(funcionario.keys()) # imprimindo uma lista das chaves
print(funcionario.values()) # imprimindo uma lista dos valores
```

Saída

```
{'nome': 'Mateus', 'idade': 24, 'cargo': 'professor'}
Mateus
professor
dict_keys(['nome', 'idade', 'cargo'])
dict_values(['Mateus', 24, 'professor'])
```

DICIONÁRIOS

Podemos listar a quantidade de chaves do dicionário utilizando o comando .keys() e também a quantidade de valores pelo comando .values()

Código

```
[34]: funcionario = {'nome' : 'Mateus', # sempre criar com chaves o dicionário ({})
                     'idade': 24, # separar a chave do valor com ( : )
                     'cargo': 'professor'}


print(funcionario) # imprimindo o dicionario
print(funcionario['nome']) # imprimindo o valor associado a chave nome
print(funcionario['cargo']) # imprimindo o valor associado a chave cargo
print(funcionario.keys()) # imprimindo uma lista das chaves
print(funcionario.values()) # imprimindo uma lista dos valores
```

Saída

```
{'nome': 'Mateus', 'idade': 24, 'cargo': 'professor'}
Mateus
professor
dict_keys(['nome', 'idade', 'cargo'])
dict_values(['Mateus', 24, 'professor'])
```

DICIONÁRIOS

Podemos remover e adicionar chaves também.

Código

```
[36]: funcionario = {'nome' : 'Mateus', # sempre criar com chaves o dicionário ({})
                     'idade': 24, # separar a chave do valor com ( : )
                     'cargo': 'professor'}


funcionario["sexo"] = "masculino" # criando uma nova chave

del funcionario["idade"] # deletando uma chave já existente

print(funcionario)
```

Saída

```
{'nome': 'Mateus', 'cargo': 'professor', 'sexo': 'masculino'}
```

Para criar uma nova chave, basta utilizar a sintaxe de pesquisa da chave nova e inserir o valor como se fosse uma variável **funcionario["sexo"] = "masculino"** e para remover basta utilizar o comando **del nome_dicionario['chave'].**

Vamos por a mão na massa

Realização de problemas práticos para fixação do conteúdo

Exercícios

1. **Peça o nome do usuário e mostre a mensagem:**

Olá, <nome>! Bem-vindo ao Python.

2. **Soma de dois números**

Solicite dois números ao usuário e exiba a soma deles.

3. **Par ou ímpar**

Peça um número inteiro e informe se ele é par ou ímpar.

Exercícios

4. Média de notas

Receba 3 notas e calcule a média.

Mostre:

"Aprovado" se média ≥ 7

"Recuperação" se média entre 5 e 6.9

"Reprovado" se média < 5

Exercícios

5. Tabuada

Peça um número e mostre a tabuada dele de 1 a 10.

6. Maior número da lista

Dada a lista: [10, 45, 2, 78, 23, 89, 1]

encontre o maior número

Exercícios

Faça um programa que leia o nome de um vendedor, o seu salário fixo e o total de vendas efetuadas por ele no mês (em dinheiro). Sabendo que este vendedor ganha 15% de comissão sobre suas vendas efetuadas, informar o total a receber no final do mês, com duas casas decimais.

Entrada

O arquivo de entrada contém um texto (primeiro nome do vendedor) e 2 valores de dupla precisão (double) com duas casas decimais, representando o salário fixo do vendedor e montante total das vendas efetuadas por este vendedor, respectivamente.

Saída

Imprima o total que o funcionário deverá receber, conforme exemplo fornecido.

Exemplos de Entrada	Exemplos de Saída
JOAO 500.00 1230.30	TOTAL = R$ 684.54
PEDRO 700.00 0.00	TOTAL = R$ 700.00
MANGOJATA 1700.00 1230.50	TOTAL = R$ 1884.58

Disponível em: [beecrowd | Dashboard - beecrowd]

Exercícios

Leia 10 valores inteiros. Apresente então o maior valor lido e a posição dentre os 10 valores lidos.

Entrada

O arquivo de entrada contém 10 números inteiros, positivos e distintos.

Saída

Apresente o maior valor lido e a posição de entrada, conforme exemplo abaixo.

Exemplo de Entrada	Exemplo de Saída
2 113 45 34565 6 ... 8	34565 4

Adaptado do: [beecrowd | Dashboard - beecrowd]

Exercícios

Maria acabou de iniciar seu curso de graduação na faculdade de medicina e precisa de sua ajuda para organizar os experimentos de um laboratório o qual ela é responsável. Ela quer saber no final do ano, quantas cobaias foram utilizadas no laboratório e o percentual de cada tipo de cobaia utilizada.

Este laboratório em especial utiliza três tipos de cobaias: sapos, ratos e coelhos. Para obter estas informações, ela sabe exatamente o número de experimentos que foram realizados, o tipo de cobaia utilizada e a quantidade de cobaias utilizadas em cada experimento.

Entrada

A primeira linha de entrada contém um valor inteiro **N** que indica os vários casos de teste que vem a seguir. Cada caso de teste contém um inteiro **Quantia** ($1 \leq$ **Quantia** ≤ 15) que representa a quantidade de cobaias utilizadas e um caractere **Tipo** ('C', 'R' ou 'S'), indicando o tipo de cobaia (*R*:Rato *S*:Sapo *C*:Coelho).

Saída

Apresente o total de cobaias utilizadas, o total de cada tipo de cobaia utilizada e o percentual de cada uma em relação ao total de cobaias utilizadas, sendo que o percentual deve ser apresentado com dois dígitos após o ponto.

Exemplo de Entrada	Exemplo de Saída
10 10 C 6 R 15 S 5 C 14 R 9 C 6 R 8 S 5 C 14 R	Total: 92 cobaias Total de coelhos: 29 Total de ratos: 40 Total de sapos: 23 Percentual de coelhos: 31.52 % Percentual de ratos: 43.48 % Percentual de sapos: 25.00 %

Disponível em: [beecrowd | Dashboard - beecrowd]

Exercícios

Leia 2 valores inteiros **X** e **Y**. A seguir, calcule e mostre a soma dos números impares entre eles.

Entrada

O arquivo de entrada contém dois valores inteiros.

Saída

O programa deve imprimir um valor inteiro. Este valor é a soma dos valores ímpares que estão entre os valores fornecidos na entrada que deverá caber em um inteiro.

Exemplo de Entrada	Exemplo de Saída
6 -5	5
15 12	13
12 12	0

Disponível em: [beecrowd | Dashboard - beecrowd]

Exercícios

1. **Calculadora de Idade:**

Peça ao usuário o ano de nascimento e o ano atual. Calcule e exiba a idade da pessoa.

2. Conversor de Unidades:

Crie um programa que receba um valor em metros e o converta para centímetros.

3. Par ou Ímpar:

Peça um número inteiro ao usuário. O programa deve dizer se o número é par ou ímpar.

Exercícios

4. Média de Notas:

Crie uma lista com 4 notas de um aluno. O programa deve calcular a média e exibir:

5. Tabuada Inteligente:

Peça um número ao usuário e gere a tabuada dele de 1 a 10, mas pare a execução se o resultado da multiplicação for maior que 50.

6. Jogo de Adivinhação

O computador deve "pensar" em um número entre 1 e 10. O usuário tenta adivinhar. Dica: Use import random e random.randint(1, 10) para gerar o número secreto.

Exercícios

7. O Termômetro da Semana

Crie uma lista com as temperaturas médias de 7 dias (ex: [25, 30, 22, 18, 24, 28, 21]). O programa deve:

- Exibir a maior e a menor temperatura da lista.
- Calcular a média das temperaturas.
- Dizer em quantos dias a temperatura ficou acima de 25°C.

Exercícios

A fórmula para calcular a área de uma circunferência é: **area = π . raio²**. Considerando para este problema que **π =** 3.14159:

- Efetue o cálculo da área, elevando o valor de **raio** ao quadrado e multiplicando por **π**.

Entrada

A entrada contém um valor de ponto flutuante (dupla precisão), no caso, a variável **raio**.

Saída

Apresentar a mensagem "A=" seguido pelo valor da variável **area**, conforme exemplo abaixo, com 4 casas após o ponto decimal. Utilize variáveis de dupla precisão (double). Como todos os problemas, não esqueça de imprimir o fim de linha após o resultado, caso contrário, você receberá "Presentation Error".

Exemplos de Entrada	Exemplos de Saída
2.00	A=12.5664
100.64	A=31819.3103
150.00	A=70685.7750

Disponível em: beecrowd | Dashboard - beecrowd

Exercícios

Leia um número inteiro que representa um código de DDD para discagem interurbana. Em seguida, informe à qual cidade o DDD pertence, considerando a tabela abaixo:

DDD	Destination
61	Brasilia
71	Salvador
11	Sao Paulo
21	Rio de Janeiro
32	Juiz de Fora
19	Campinas
27	Vitoria
31	Belo Horizonte

Se a entrada for qualquer outro DDD que não esteja presente na tabela acima, o programa deverá informar:
DDD nao cadastrado

Entrada

A entrada consiste de um único valor inteiro.

Saída

Imprima o nome da cidade correspondente ao DDD existente na entrada. Imprima *DDD nao cadastrado* caso não existir DDD correspondente ao número digitado.

Exemplo de Entrada	Exemplo de Saída
11	Sao Paulo

Disponível em: [beecrowd | Dashboard - beecrowd]

Dúvidas?

Obrigado pela atenção!

Mateus Rocha

Cientista de Dados | Estatístico