



Programação em Python para Data Science

Banco de dados NoSQL

O termo "NoSQL" se refere a "Not Only SQL", "não apenas SQL" em nossa língua portuguesa.

Esse tipo de banco de dados não está relacionado a uma arquitetura estruturada de linhas e tabelas (como o tipo SQL) e em relacionamentos fortes via chaves primárias e estrangeiras. A sua arquitetura pode variar conforme a necessidade das aplicações.

Embora o conceito do NoSQL seja antigo (desde a década de 60) seu uso e fama aumentaram consideravelmente com o advento dos grandes sistemas, das redes sociais, sistemas de streaming e a necessidade de se armazenar, gerir e consultar grandes volumes de dados.

O NoSQL veio para suprir as limitações do banco de dados SQL (estruturado), pois o mesmo em nível de arquitetura, desempenho e volume não é tão performático quando se trata de plataformas web, jogos, streamings e redes sociais. O grande volume de informações e a necessidade de baixa latência (resposta rápida do banco de dados) exigem o NoSQL.



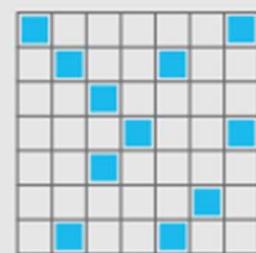
Banco de dados NoSQL

Se os bancos de dados NoSQL não são formados por tabelas relacionadas entre si, quais são os tipos de sua arquitetura de armazenamento e gerenciamento? Por ser um banco não relacional, atualmente encontramos as seguintes arquiteturas:

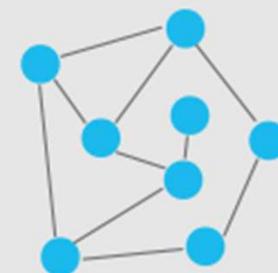
- Colunar
- Orientado a documentos
- Chave-Valor
- Grafos



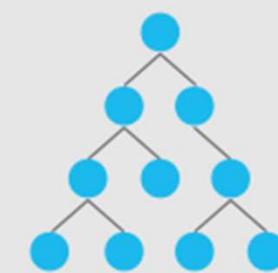
NoSQL
Database



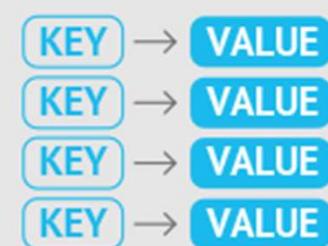
Column-Family



Graph



Document



Key-Value

Banco de dados NoSQL

Colunar: Como o próprio nome sugere, a arquitetura colunar armazena os dados no estilo entidade-coluna, onde para cada entidade (informação), tem uma coluna separada e específica. Vejamos a diferença entre uma tabela SQL e uma colunar NoSQL abaixo:



Row-oriented

ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

Column-oriented

Name	ID
John	001
Karen	002
Bill	003

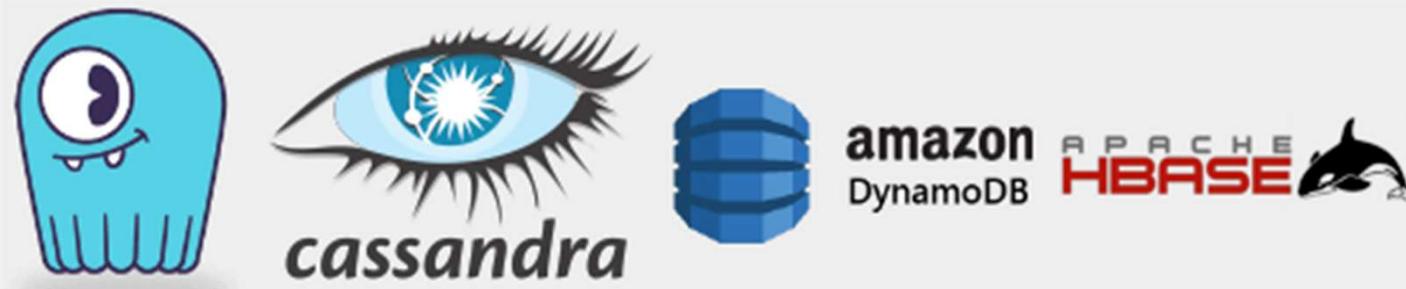
Grade	ID
Senior	001
Freshman	002
Junior	003

GPA	ID
4.00	001
3.67	002
3.33	003

Banco de dados NoSQL

Colunar: Por separar cada informação em tabelas únicas a performance das consultas nesses tipos de bancos de dados é bem rápida.

Alguns exemplos de bancos NoSQL colunares:



ScyllaDB, Cassandra, Amz DynamoDB, HBase

Banco de dados NoSQL

Orientado a documentos: Esses bancos de dados armazenam as informações em documentos, geralmente no formato JSON (JavaScript Object Notation) que são formados pelo conjunto chave-valor. É o tipo de banco de dados ideal para armazenar dados semi-estruturados e não estruturados.

O desenvolvedor não precisa se preocupar antecipadamente com as informações que deseja guardar. É um banco de dados dinâmico. Muito usado por exemplo para armazenar conteúdo de sites: links (urls), artigos, textos, imagens, comentários, etc:

```
Order={  
    "_id": "2934f",  
    "salesDate": "2022-05-02",  
    "customer": {  
        "name": "Jack Beanstalk",  
        "gender": "M",  
        "rewardsMember": "True"  
    },
```



Banco de dados NoSQL



Orientado a documentos: Alguns exemplos de bancos de dados orientados a documentos:



O MongoDB tem sido, de longe, a solução mais utilizada nessa estrutura de dados NoSQL.



Banco de dados NoSQL

Chave-valor: O banco de dados do tipo chave-valor é o mais simples de todos e também o mais rápido. Obviamente deve ser usado para aplicações de armazenamento simples e para consultas simples, tais como dados em cache e sistemas de armazenamento embarcados (ex: dispositivos de IoT).

Como o próprio nome sugere, o banco armazena para cada identidade um valor, muito parecido com a estrutura de dados "dicionário" em Python:

```
"_id": "tomjohnson",
"firstName": "Tom",
"middleName": "William",
"lastName": "Johnson",
"email": "tom.johnson@digitalocean.com",
"department": ["Finance", "Accounting"]
```

```
pedido = {
    "id": "45",
    "cliente": "44411122547",
    "prd1": "4125477",
    "prd2": "45124587"
}
```

Banco de dados NoSQL



Chave-valor: Alguns exemplos de bancos de dados chave-valor:



Banco de dados NoSQL

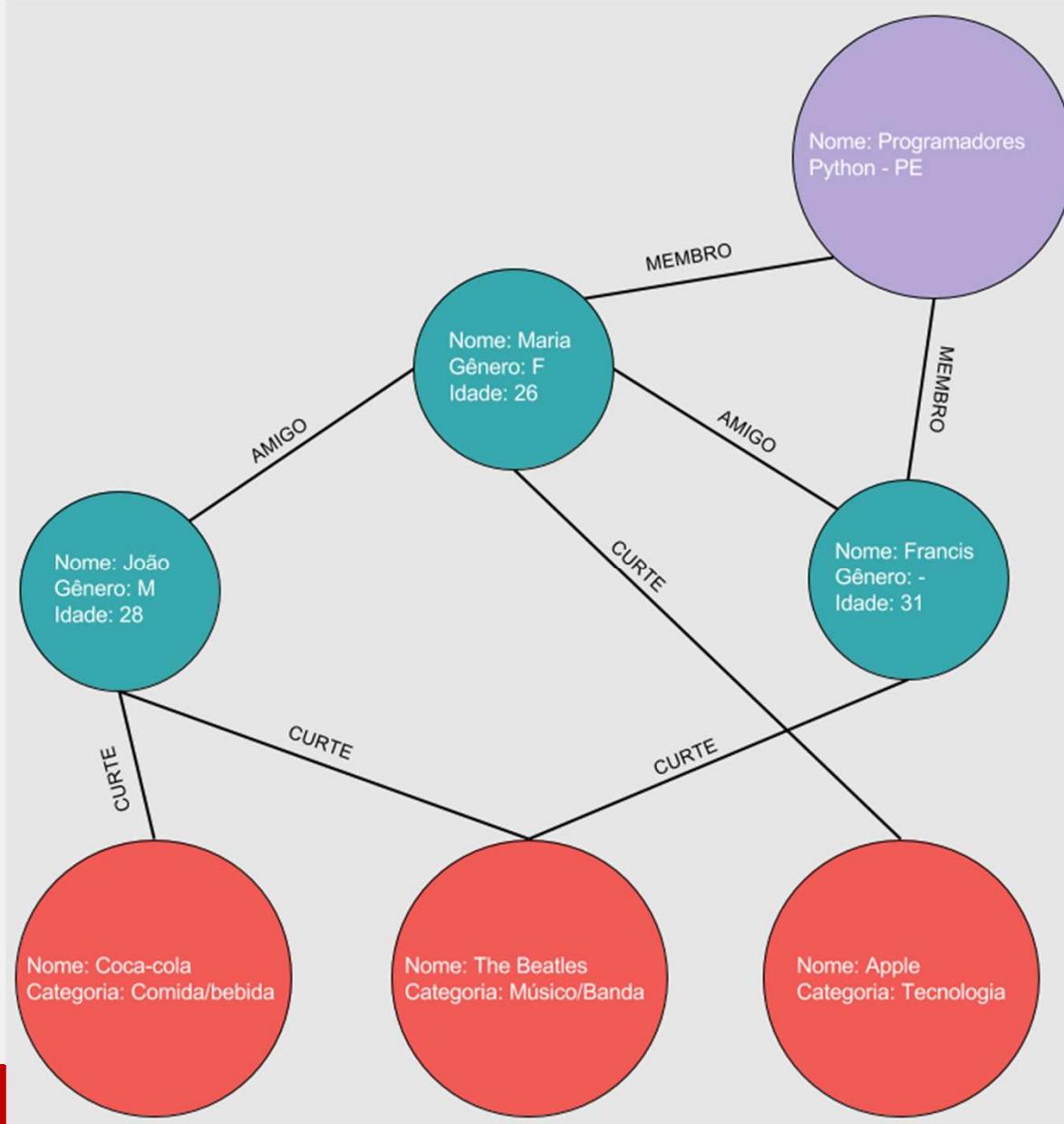


Grafos: É um tipo de banco de dados orientado a relacionamentos extremamente fortes entre diversas entidades e valores. Sua montagem e visualização formam grafos, mostrando visualmente os diversos relacionamentos das entidades.

Banco de dados usado principalmente em redes sociais e na criação de motores de recomendações (por exemplo quando assistimos um vídeo no YouTube e a plataforma nos retorna vários outros vídeos relacionados aos anteriores que assistimos. Alguns exemplos de bancos de dados Grafos:



SENAI



Linguagem de programação Python - Revisão



Antes de iniciarmos nossa prática utilizando uma plataforma de Big Data, precisaremos revisar alguns conceitos da linguagem Python.

Usaremos em breve a plataforma Spark, através de bibliotecas e métodos em Python.

Para aproveitarmos da melhor forma possível esses conceitos, nada melhor que uma boa revisão da linguagem Python.



Introdução ao Python

Histórico

Criado em 1991 por Guido Van Rossum, na Holanda.

Linguagem de alto nível e possibilita vários tipos de implementações (funcional até orientação a objetos).

Linguagem simples e poderosa. Fácil de implementar métodos e funções em comparação a outras linguagens de programação.

Possui várias bibliotecas customizadas, orientadas para vários tipos de aplicações (documentos, imagens, voz, matemática, banco de dados, dados científicos, inteligência artificial, etc).

Possibilita fácil leitura do código. Muito bem estruturada e dinâmica.

Desde 2009 tornou-se a linguagem de programação padrão dos cursos do Massachusetts Institute of Technology (MIT) .

Licença livre. Python pode ser utilizado por qualquer pessoa e de forma gratuita.



Introdução ao Python

Histórico

Por ser uma linguagem de fácil escrita e de possibilitar criação de códigos poderosos e bem estruturados, Python está presente em diversos projetos de grandes empresas, dentre elas:

Instagram (muitos métodos e funções do Insta são feitos em Python)

Pintrest (todo motor dessa rede social é feito em Python)

Dropbox (principalmente os controladores da parte WEB)

Facebook

Spotify



NASA – Em vários projetos de pesquisas

Computação científica e Big Data



Conceitos de programação em Python

Tipos de dados em Python

Tipos básicos de dados

Tipos de dados mais básicos		
Dado	Tipo	Tipo em python
"Juliana"	Texto	str (string)
40	Inteiro	int
"André"	Texto	str (string)
44.8	Decimal	float
False	Booleano	bool
8	Inteiro	int
True	Booleano	bool
"Olá"	Texto	str (string)
45 + 10L	Complexo	complex



Conceitos de programação em Python

Variáveis em Python



Podemos definir as variáveis como reservas de espaços dentro da memória RAM.

Armazenam valores (dados) e atribuem um nome específico para esse espaço dentro da memória.

Os dados ficam armazenados para serem executados juntamente com as instruções de nossos algoritmos (conceito de programa armazenado de Von Neumann).

As linguagens de programação possuem operadores e regras específicas para nomear e atribuir valores a variáveis.



Conceitos de programação em Python

Variáveis em Python

Regras quanto à nomenclatura da variável:

Nome	Válido	Comentários
a1	Sim	Embora contenha um número, o nome a1 inicia com letra.
velocidade	Sim	Nome formado por letras.
velocidade90	Sim	Nome formado por letras e números, mas iniciado por letra.
salário_médio	Sim	O símbolo sublinha (_) é permitido e facilita a leitura de nomes grandes.
salário médio	Não	Nomes de variáveis não podem conter espaços em branco.
b	Sim	O sublinha () é aceito em nomes de variáveis, mesmo no início.
1a	Não	Nomes de variáveis não podem começar com números.



Python - Revisão

Tipos de dados em Python

Os 4 tipos mais básicos de tipos de dados que podemos ter em Python são: strings (textos), nºs inteiros (int), nºs decimais (float) e booleanos (True or False).



```
a = "Maria"
b = 45
c = 10.75
d = 40 > 10
lista = ["Python", "Big Data", "Spark", "Programação", "Bibliotecas"]
tupla = ("SENAI", "Rio Preto", "Alunos", "Cursos", "Rentabilidade")
dicionario = {
    "curso": "Fundamentos de Big Data",
    "alunos": 20,
    "horas": 60,
    "cidade": "São José do Rio Preto"
}
```

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'list'>
<class 'tuple'>
<class 'dict'>
```

Python - Revisão

Operadores relacionais em Python (revisão):

Operador	Operação	Símbolo matemático
<code>==</code>	igualdade	<code>=</code>
<code>></code>	maior que	<code>></code>
<code><</code>	menor que	<code><</code>
<code>!=</code>	diferente	<code>≠</code>
<code>>=</code>	maior ou igual	<code>≥</code>
<code><=</code>	menor ou igual	<code>≤</code>



Python - Revisão

Operadores lógicos



Operador	Definição
and	Retorna True se ambas as afirmações forem verdadeiras
or	Retorna True se uma das afirmações for verdadeira
not	retorna Falso se o resultado for verdadeiro



Python - Revisão

Operadores lógicos

```
a = 80  
b = 150  
r = a - b  
print(r)  
print(a > b and a > r)  
print(a > b or a > r)  
print(not(a > b and a > r))
```

```
-70  
False  
True  
True
```



Python - Revisão



Operadores aritméticos

Operador	Nome	Função
+	Adição	Realiza a soma de ambos operandos.
-	Subtração	Realiza a subtração de ambos operandos.
*	Multiplicação	Realiza a multiplicação de ambos operandos.
/	Divisão	Realiza a Divisão de ambos operandos.
//	Divisão inteira	Realiza a divisão entre operandos e a parte decimal de ambos operandos.
%	Módulo	Retorna o resto da divisão de ambos operandos.
**	Exponenciação	Retorna o resultado da elevação da potência pelo outro.

Python - Revisão



Operadores de atribuição (em variáveis)

Operador	Equivalente a
=	x = 1
+=	x = x + 1
-=	x = x - 1
*=	x = x * 1
/=	x = x / 1
%=	x = x % 1

```
C = 50
C /= 10
print(c)
C += 95
print(c)

5.0
100.0
```



Python - Revisão

Operadores de associação



Operador Funcão

`in` Retorna True caso o valor seja encontrado na sequência

`not in` Retorna True caso o valor não seja encontrado na sequência

```
nomes = ["Ana", "Pedro", "Larissa"]
nome = "João"
print(f"{nome} está na lista? {nome in nomes}")
print(f"{nome} não está na lista? {nome not in nomes}")
```

```
João está na lista? False
João não está na lista? True
```



Python - Revisão

Estruturas condicionais: if, elif, else



```
idade = 37
crianca = (idade <=12)
adolescente = (idade <= 18)
adulto = (idade <= 60)
if crianca:
    r = "criança"
elif adolescente:
    r = "adolescente"
elif adulto:
    r = "adulto"
else:
    r = "idoso"
print(f"Você tem {idade} anos e é {r}.")
```

Você tem 37 anos e é adulto.



Python - Revisão

Entrada de dados através da função `input()`:

```
nome1 = input("Digite o 1º nome:")
idade1 = int(input(f"Digite a idade do(a) {nome1}:"))
nome2 = input("Digite o 2º nome:")
idade2 = int(input(f"Digite a idade do(a) {nome2}:"))
if (idade1 > idade2):
    print(f"{nome1} é mais velho(a) que {nome2}")
elif(idade2 > idade1):
    print(f"{nome2} é mais velho(a) que {nome1}")
else:
    print(f"{nome1} e {nome2} tem a mesma idade!")
```

```
Digite o 1º nome:João
Digite a idade do(a) João:45
Digite o 2º nome:Ana
Digite a idade do(a) Ana:23
João é mais velho(a) que Ana
```



Python - Revisão

Com relação às estruturas de dados, devemos ter muita familiaridade com as listas, tuplas e dicionários.

São variáveis que armazenam vários valores que podem ser de diferentes tipos (int, float, string, outra lista, outro dicionário, etc).

LISTA: Índice numérico sequencial, iniciando pelo índice zero:

```
nomes = ["João", "Larissa", "Ruy"]
nomes.append("Diego")
print(nomes)
print(nomes[2])
```

```
['João', 'Larissa', 'Ruy', 'Diego']
Ruy
```



Python - Revisão

TUPLA: A tupla, diferentemente da lista, é imutável. Em sua criação atribuímos seus valores que depois não podem mais ser alterados. Se tentarmos alterar seus valores, uma exceção (erro) é gerada:

```
nomes = ("João", "Larissa", "Ruy")
# nomes.append("Diego")
nomes[0] = "Diego"
print(nomes)
```

```
Traceback (most recent call last):
  File "C:\Users\Aluno\PycharmProjects\pythonProject\Big\pr
    nomes[0] = "Diego"
TypeError: 'tuple' object does not support item assignment
```



Python - Revisão



DICIONÁRIO: São estruturas de dados muito poderosas em Python. Podem ser usadas em diversas lógicas e situações.



Sua estrutura é basicamente "chave":"valor". Exemplo: "João":15 (nome e idade).

"Ana":[45,1.67] (chave "nome" cujo valor é uma lista contendo idade e altura).

Diferentemente das listas e tuplas que são acessadas pelo índice sequencial e numérico, os dicionários são acessados por sua chave:

```
alunos = {  
    "João":15,  
    "Ana":45,  
    "Lucas":31  
}
```

```
Idade do João: 15.  
Idade da Ana: 45.
```

Python - Revisão

DICIONÁRIO: Abaixo um exemplo de dicionário cujos valores são uma lista:

```
alunos2 = {  
    "João": [15, 1.85],  
    "Ana": [45, 1.67],  
    "Lucas": [31, 1.78]  
}  
  
print(f"Dados do João: Idade: {alunos2['João'][0]} | "  
      f"Altura: {alunos2['João'][1]}")  
  
print(f"Dados do Lucas: Idade: {alunos2['Lucas'][0]} | "  
      f"Altura: {alunos2['Lucas'][1]}")
```

Dados do João: Idade: 15 | Altura: 1.85

Dados do Lucas: Idade: 31 | Altura: 1.78



Python - Revisão

DICIONÁRIO: Inclusão de elementos:

```
# Criando dicionário:  
contatos = {}  
  
# input de dados:  
nome = input("Digite o nome do aluno:")  
idade = int(input(f"Digite a idade do(a) {nome}:"))  
  
# Inserção dos dados no dicionário  
# Atenção à sintaxe com parênteses e chaves:  
contatos.update({nome:idade})  
print(f"Qtd contatos: {len(contatos)}")  
contatos.update({"Larissa":25})  
print(f"Qtd contatos: {len(contatos)}")
```



```
Digite o nome do aluno:Leandro  
Digite a idade do(a) Leandro:35  
Qtd contatos: 1  
Qtd contatos: 2  
Nome: Leandro | Idade: 35  
Nome: Larissa | Idade: 25
```

Python - Revisão - Repetição e fluxo de dados

Como toda linguagem de programação, Python também possui suas próprias estruturas e regras de repetição, cabendo ao programador definir em quais situações elas serão melhor utilizadas. **Comando while:**



```
# contador para auxílio na repetição
cont = 1
# contadores para auxílio na lógica para contar quantos elementos pares e
# quantos ímpares serão digitados:
contp = 0
conti = 0
while (cont <= 7): # código repetirá 7 vezes
    num = int(input(f"Digite o {cont}º valor:"))
    if (num%2 == 0):
        # incrementando contador de pares
        contp += 1 # contp = contp + 1
    else:
        # incrementando contador de ímpares
        conti += 1
    cont += 1 # incrementando contador geral da repetição
# Fim da repetição
print(f"Qtd de pares: {contp}.")
print(f"Qtd de ímpares: {conti}.")
```

```
Digite o 1º valor:1
Digite o 2º valor:2
Digite o 3º valor:3
Digite o 4º valor:3
Digite o 5º valor:3
Digite o 6º valor:4
Digite o 7º valor:1
Qtd de pares: 2.
Qtd de ímpares: 5.
```

Python - Revisão - Repetição e fluxo de dados

Comando while:

```
# importando função randint da
# classe/biblioteca random:
from random import randint
continuar = True
while (continuar):
    # Gerando um número inteiro aleatório entre 1 e 50:
    num = randint(1,50)
    print(f"Número gerado: {num}.")
    r = input("Deseja gerar outro nº? 1 para sim 2 para não:")
    if (r == "2"): # Se a resposta for não:
        continuar = False # continuar será falso
print("Fim de programa!")
```

Número gerado: 11.

Deseja gerar outro nº? 1 para sim 2 para não:**1**

Número gerado: 49.

Deseja gerar outro nº? 1 para sim 2 para não:**1**

Número gerado: 14.

Deseja gerar outro nº? 1 para sim 2 para não:**2**

Fim de programa!



Python - Revisão - Repetição e fluxo de dados

Comando for: É uma estrutura de repetição extremamente poderosa. Em Python é usado principalmente para repetição e manipulação de elementos iteráveis (sequenciais) e que possuem vários elementos: listas, tuplas, dicionários, dentre outros:



```
# importando todas as funções
# da biblioteca math:
import math as mt
# Lógica de exemplo:
x = 15
for i in range(1,6):
    # Usando função pow (exponenciação):
    r = mt.pow(x,i)
    print(f"{x}^{i} = {r:.1f}")
```

```
15^1 = 15.0
15^2 = 225.0
15^3 = 3,375.0
15^4 = 50,625.0
15^5 = 759,375.0
```

Python - Revisão - Repetição e fluxo de dados

Comando for:

```
# Criando uma lista aninhada (listas dentro de listas):
relogios = [
    [1, "CASIO", "MD45", 1500.0],
    [2, "CASIO", "MD79", 380.0],
    [3, "TECHNOS", "SMARTW", 1667.0]
]
total = 0 # acumulador
for elemento in relogios:
    print(f"Cod: {elemento[0]:2} | Marca: {elemento[1]:8} | "
          f"Modelo: {elemento[2]:8} | Preço: R${elemento[3]:,.2f}")
    total += (elemento[3]) # somatório dos preços a cada loop
print(f"Total: R${total:,.2f}")
```

```
Cod: 1| Marca: CASIO      | Modelo: MD45        | Preço: R$1,500.00
Cod: 2| Marca: CASIO      | Modelo: MD79        | Preço: R$380.00
Cod: 3| Marca: TECHNOS    | Modelo: SMARTW     | Preço: R$1,667.00
Total: R$3,547.00
```



Python - Revisão - Repetição e fluxo de dados



Comando for:

```
# Criando um dicionário:  
coordenadas = {  
    "Sao Jose do Rio Preto": [-20.81124607137126, -49.37688729789731],  
    "Sao Paulo": [-23.55533834819181, -46.64457395716928],  
    "Copacabana": [-22.969436264111728, -43.18790100907717],  
    "Suíça": [46.77537359550784, 7.55542862634246],  
    "França": [46.75005969187713, 2.197177745856688]  
}  
  
print(f"Qtd de elementos no dicionário: {len(coordenadas)}")  
for local, coordenada in coordenadas.items(): # uso da função items()  
    print(f"{local:21}| Latitude: {coordenada[0]:20}| Longitude: {coordenada[1]}")
```

Qtd de elementos no dicionário: 5

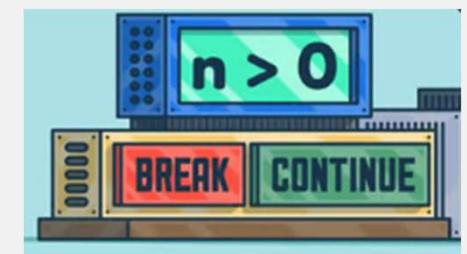
Sao Jose do Rio Preto	Latitude: -20.81124607137126	Longitude: -49.37688729789731
Sao Paulo	Latitude: -23.55533834819181	Longitude: -46.64457395716928
Copacabana	Latitude: -22.969436264111728	Longitude: -43.18790100907717
Suíça	Latitude: 46.77537359550784	Longitude: 7.55542862634246
França	Latitude: 46.75005969187713	Longitude: 2.197177745856688

Python - Revisão - Repetição e fluxo de dados



Interrupção do fluxo de repetição: Comandos break e continue

Durante a execução de uma repetição, pode ser necessário em determinada lógica que a repetição seja interrompida completamente ou que uma estrutura pare de ser executada para voltar ao início do bloco de repetição.



break: Interrompe o loop (repetição) e o encerra:

```
conta_loop = 0
while (conta_loop < 10):
    conta_loop += 1
    print(f"Loop n° {conta_loop}")
    if (conta_loop >= 5):
        break # Estrutura de repetição encerrada!
    print("Repetindo....")
print("Fim da repetição.")
```

```
Loop n° 1
Repetindo....
Loop n° 2
Repetindo....
Loop n° 3
Repetindo....
Loop n° 4
Repetindo....
Loop n° 5
Fim da repetição.
```

Python - Revisão - Repetição e fluxo de dados



continue: Interrompe o loop (repetição) e volta para o início da repetição:

```
conta_loop = 0
while (conta_loop < 10):
    conta_loop += 1
    print(f"Loop n° {conta_loop}")
    if (conta_loop >= 5):
        continue
    print("Repetindo....")
print("Fim da repetição.")
```

```
Loop n° 1
Repetindo....
Loop n° 2
Repetindo....
Loop n° 3
Repetindo....
Loop n° 4
Repetindo....
Loop n° 5
Loop n° 6
Loop n° 7
Loop n° 8
Loop n° 9
Loop n° 10
Fim da repetição.
```

