# Python para análise de dados

Semana 4

# Q Cronograma estimado



09h00 - 12h00

0

0

13h30 - 14h30

14h30-16h30

16h30 - 17h00

Iteradores e o laço while

Função range() e o laço for

Estruturas de dados

0

Comprehensions\*

#### Recapitulando a aula 3

- Condicionais
- Operadores lógicos
- Erros
- Debugging

#### Iteração - Definição

- Acto ou efeito de iterar ou de repetir. = REITERAÇÃO, REPETIÇÃO\*
- Iterar é uma palavra pouco comum no dia a dia, mas muito comum no mundo da programação.
- Ela é usada quando queremos dizer que algo deve ser repetido, feito mais de uma.

#### Iteração - Definição

- Vamos pensar nos exemplos do professor fechando notas da aula passada. Um professor tem vários alunos (n) e portanto, teria que executar nosso programa várias e várias vezes para poder fechar a nota de todos eles.
- Ao invés disso, ele pode dizer ao programa que execute n vezes.
- Usar a notação "n" para um número que não sabemos qual é muito comum e vocês já devem ter visto ela nos exercícios das aulas anteriores
   :)

#### **Laço While**

- Vamos começar a usar iteração com o laço while, que em português significa enquanto
- Tal qual o if, a expressão while avalia se uma condição é verdadeira
- Leia como:

Enquanto a condição for verdadeira, imprime "Uhul!"

- Aqui nós precisamos ter cuidado! Um laço de repetição cuja condição de parada nunca será verdadeira vai criar um loop infinito. Isso é, nosso programa nunca vai sair do bloco interno de execução.
- Vamos mudar nossa condição para usar uma variável na comparação:

Enquanto num1 é menor que 0, imprime "Uhul!"

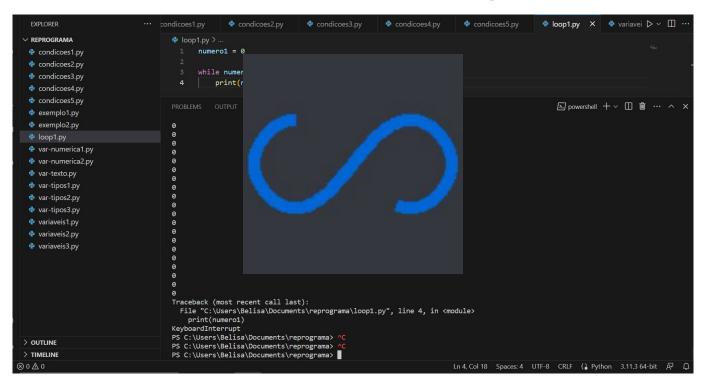
 E agora vamos trabalhar essa variável para garantir que nosso loop seja encerrado em algum ponto

• Vamos criar uma laço que imprime os números de 1 a 5:

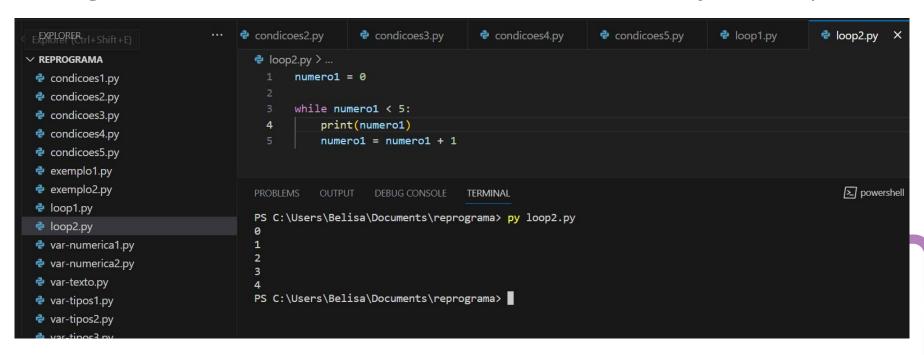
Enquanto numero1 é menor que 5, imprime numero1

 Nós precisamos iniciar a variável num1 antes do laço, senão o python não saberá seu valor:

num1 = 0



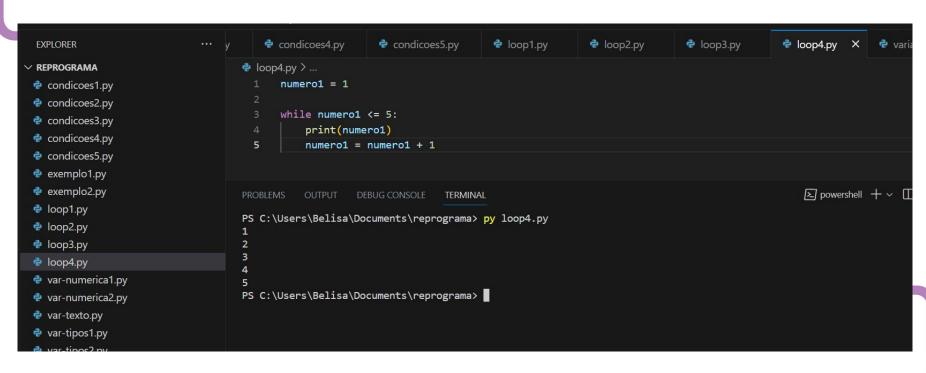
Agora vamos incrementar a variável num1 a cada execução do loop

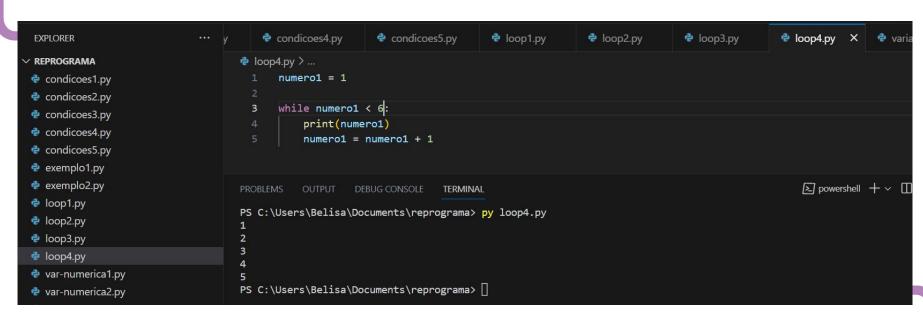


#### Ferramentas de controle de fluxo - While - Perguntas

- Quantas vezes o loop foi executado?
- Nós conseguimos imprimir os números de 1 a 5?
- Se não conseguimos, porque não?







#### **Laço While**

• Vamos mudar nosso laço para imprimir um intervalo iniciado em 0 com n números informado pelo usuário.

```
S03 > exercícios > ♥ while-input.py > ...
       inicio = 0
       intervalo = int(input())
      while inicio < intervalo:
           print(inicio)
           inicio += 1
PROBLEMS
                    TERMINAL
PS C:\Users\Belisa\Documents\reprograma\ementa-python-analise-dados\S03\exercícios> py .\while-input.py
PS C:\Users\Belisa\Documents\reprograma\ementa-python-analise-dados\S03\exercícios> \
```

# 5 Minutinhos pra resfriar o cérebro Tomem água!



- Existe outra forma de fazermos laços sobre uma sequência de números sem que seja necessário incrementar valores dentro do laço.
- Para isso, nós usaremos a função range()
- A palavra range em inglês significa intervalo e é isso que construiremos: um intervalo de números.
- Essa função cria um sequência imutável de números inteiros seguindo alguns parâmetros obrigatórios e outros opcionais.

- Parâmetro obrigatroio: parada, ou seja, o número de itens a serem criados após o início (start).
- Parâmetros opcionais:
  - Start: é possível criar um intervalo que começa em qualquer número desejado. Se um início não for informado, o construtor inicia em 0.
  - Step: também é possível criar intervalos não contínuos. Por exemplo, se quisermos um intervalo somente com números positivos, podemos definir o step como 2 e o início em um número parar, de forma que somente números pares serão criados. Se não for informado, o step é 1.

• A função range é definida de duas formas:

```
class range(stop)
class range(start, stop[, step])
```

 Isso quer dizer que ao passarmos mais de um parâmetro, é necessário prestar atenção na ordem em que eles serão passados. Range é uma função um pouco específica e sua implementação não permite o uso de argumentos nomeados, como os que vimos na aula de funções.

- Há ainda outros pontos que devemos ter cuidado.
- Se um start não for definido, o primeiro inteiro será zero. Iniciar índices em 0 é bastante comum na programação e veremos como é importante ter isso em mente em estruturas de dados.
- Iniciar em zero leva a outro ponto importante: o valor do argumento stop nunca fará parte da sequência, ela será encerrada sempre em até stop-1.
   Por exemplo: um range(0,4) terá os valores: 0,1,2,3,4
- Também é necessário prestar atenção para não criarmos intervalos com stop menor do que start, ou com steps que "não cabem" no intervalo. Isso não resultará erro, mas em um sequência vazia.

### A expressão range() - Exemplos

```
print("range(5)")
      print(range(5))
      print("range(1, 10)")
      print(range(1, 10))
      print("range(25, 89, 3)")
      print(range(25, 89, 3))
      print("range(5, 1)")
      print(range(5, 1))
          OUTPUT DEBUG CONSOLE TERMINAL GITLENS
belisa@belisa-Vostro-3520:~/Documents/reprograma/ementa-python-analise-dados/S04/exercicios/para-sala/laços$ python3 range.py
range(5)
range(0, 5)
range(1, 10)
range(1, 10)
range(25, 89, 3)
range(25, 89, 3)
range(5, 1)
range(5, 1)
```

- Ok, mas o retorno da função é a própria chamada da função????
- Mais ou menos.
- Como dito, a função range é toda diferentona. Ela é bastante otimizada e tem alguns comportamentos um pouco diferentes do que o comum.
- Para realmente podermos entender como ela pode ser útil, vamos ver outro tipo de loop: o for!

#### **Laço For**

- For é um laço que itera sobre um objeto iterável. Assustador, eu sei. Mas vamos com calma.
- For sempre será acompanhado da palavra reservada in e podemos traduzir essas palavras mais ou meno como "para cada" e "dentro" (nesse contexto).
- Vamos ver um exemplo de for usando range para entendermos o que significa um objeto iterável.

```
para cada
        for numero in range(5):
            print(numero)
 PROBLEMS
            OUTPUT
                     DEBUG CONSOLE
belisa@belisa-Vostro-3520:~/Do
 0
 3
```

```
dentro
        for numero in range(5):
            print(numero)
 PROBLEMS
            OUTPUT
                    DEBUG CONSOLE
belisa@belisa-Vostro-3520:~/Do
 0
```

```
objeto iterável
        for numero in range(5):
            print(numero)
 PROBLEMS
            OUTPUT
                     DEBUG CONSOLE
belisa@belisa-Vostro-3520:~/Do
 0
```

- Objeto iterável continua parecendo grego, né?
- Esse tipo de objeto é capaz de retornar cada item seu individualmente.
- Então a nossa função range(5), que tinha aquele retorno mega estranho, agora está retornando um por um dos itens da sua sequência.
- Para entender o que que tá rolando aqui, bora ver a execução do nosso exemplo linha a linha usando o debugger.

#### **Laço For - Outros objetos iteráveis**

O que acontece aqui?

```
var_texto = "A prof Be não tem criatividade para valores"
for item in var_texto:
    print(item)
```

#### Laço for - Mão na massa

- Crie um código que imprima todos os número inteiros entre 0 e 10
- Crie um código que imprima todos os número inteiros entre 10 e 0
- Crie um código que imprima todos os número pares maiores do que 22 e menores do que 68
- Crie um código que imprima a soma de todos os números entre 0 e 10
- Crie um código que conte as letras do seu nome

#### Laço for - Mão na massa - Desafio

- Crie um código que calcula o fatorial de um número.
- O que é um fatorial de um número n? A multiplicação de n por todos os números anteriores a ela:

```
3! = 3 * 2 * 1
6! = 6 * 5 * 4 * 3 * 2 * 1
```

# 5 Minutinhos de alegria - Tomem água!



#### **Estruturas de Dados**

- Agora que já estamos familiar com tipos de dados básicos, como int e str, podemos avançar para tipos mais complexos.
- Estruturas de dados são formas de armazenar e organizar nossos dados.
- Pensando de novo no pobre do professor tentando fechar as notas da sua turma, não seria mais fácil para ele ter as notas organizadas de cada aluno em uma só variável, ao invés de várias variáveis de notas "soltas"?

#### **Estruturas de Dados - Lista**

- Uma lista é uma sequência de valores que pode ter itens adicionados, removidos e alterados através de métodos desse tipo de dado.
- Os tipos de valores dentro de uma lista podem ser variados e eles são acessados pelo index.
- Existem algumas formas de criar uma lista, mas começaremos com a mais simples: usando colchetes.
- Vamos organizar as três notas da aluna em uma lista pra ajudar nosso querido professor.

#### **Estruturas de Dados - Lista**

```
notas-antes.py > ...

def calcula_media(nota1, nota2, nota3):
    media = (nota1 + nota2 + nota3)/3
    return media

nota1 = 15
    nota2 = 98
    nota3 = 75

print(calcula_media(nota1, nota2, nota3))
```

```
notas-lista.py > ...

def calcula_media(nota1, nota2, nota3):
    media = (nota1 + nota2 + nota3)/3
    return media

notas = [15, 98, 75]

#ops, e agora, como acessamos as notas?
print(calcula_media(nota1, nota2, nota3))
```

#### Estruturas de Dados - Lista - Acessando valores

# Estruturas de Dados - Lista - Operações

Operação		Resultado	Notas
x in s		True caso um item de s seja igual a $x$ , caso contrário False	(1)
x not in	s	False caso um item de $s$ for igual a $x$ , caso contrário True	(1)
s + t		a concatenação de s e t	(6)(7)
s * n ou i	1 * s	equivalente a adicionar $s$ a si mesmo $n$ vezes	(2)(7)
s[i]		i-ésimo item de s, origem 0	(3)
s[i:j]		fatia de s de <i>i</i> até <i>j</i>	(3)(4)
s[i:j:k]		fatia de s de $i$ até $j$ com passo $k$	(3)(5)
len(s)		comprimento de s	
min(s)		menor item de s	
max(s)		maior item de s	
s.index()	<[, i[, j]])	índice da primeira ocorrência de $x$ em $s$ (no ou após o índice $i$ , e antes do índice $j$ )	(8)
s.count()	()	numero total de ocorrência de x em s	

# Estruturas de Dados - Lista - Mais operações



#### Combinando listas e fors!

```
notas-for.py > ...
      def calcula_media(notas):
          soma_notas = 0
          for nota in notas:
              soma_notas = soma_notas + nota
          return soma_notas/3
      notas = [15, 98, 75]
      #acessando notas por index
 10
      print(calcula_media(notas[0], notas[1], notas[2]))
```

## Combinando listas e fors! - Usando len()

### Estruturas de Dados - Dicionários

- O tipo dicionário é um tipo de dado que permite nomear seus dados.
- Isso quer dizer que além de valores, nós podemos dar nomes ao nossos valores dentro dos dicionários.
- Eles também são mutáveis, porém os métodos que podemos usar em dicionários são diferentes dos métodos de dicionários. Também acessamos seus valores de forma diferente.
- Dicts são construídos usando chaves, mas sua construção é um pouco diferente, pois cada valor precisa ter também um nome.
- Vamos alterar o exemplo das notas mais uma vez, agora identificando cada nota por seu nome:

#### **Estruturas de Dados - Dicionários**

```
notas-dict.py > ...

def calcula_media(nota1, nota2, nota3):
    media = (nota1 + nota2 + nota3)/3
    return media

notas = {
    "nota1": 15,
    "nota2": 98,
    "nota3": 75
}
```

#### Estruturas de Dados - Dicionários - Acessando dados

```
notas-dict.py > \( \partial \) calcula_media
> VARIABLES
                                                         def calcula_media(notas):

∨ WATCH
                                                              for nota in notas:
 > notas: {'nota1': 15, 'nota2': 98, 'nota3': 75}
                                                                   soma notas = soma notas + nota
   notas[0]: KeyError: 0
                                                              return soma notas/3
   notas[1]: KeyError: 1
   notas[2]: KeyError: 2
                                                         notas = {
   len(notas): 3
                                                              "nota1": 15,
                                                              "nota2": 98,
                                                              "nota3": 75
                                                    10
                                                   12
                                                         #acessando notas por index
                                                         print(calcula_media(notas))
                                                    13
```

## Estruturas de Dados - Dicionários - Acessando dados

```
notas-dict.py > 🕅 calcula_media
> VARIABLES
                                                                         def calcula_media(notas):

∨ WATCH

                                                                              soma notas = 0
                                                                for nota in notas.values():
  > special variables
                                                                                  soma_notas = soma_notas + nota
  > function variables
                                                                              return soma notas/len(notas)
    'nota1': 15
    'nota2': 98
                                                                         notas = {
                                                                              "nota1": 15,
    'nota3': 75
                                                                              "nota2": 98,
    len(): 3
                                                                              "nota3": 75
  notas.values(): dict_values([15, 98, 75])
   len(notas): 3
                                                                         print(calcula_media(notas))
```

## **Estruturas de Dados - Tuplas**

- Existe mais uma estrutura bastante usada, a tupla.
- Ela é bastante parecida com nossas listas, porém são imutáveis.
- Isso quer dizer que depois de criada, ela n\u00e3o pode ser alterada. N\u00e3o \u00e9
  poss\u00edvel adicionar, remover nem mudar valores em uma tupla.
- Elas são criadas usando parênteses ao invés de colchetes:

```
notas-tupla.py > ...

def calcula_media(nota1, nota2, nota3):

media = (nota1 + nota2 + nota3)/3

return media

notas = (15, 98, 75)

#acessando notas por index
print(calcula_media(notas[0], notas[1], notas[2]))
```

# 5 minutinhos pra comemorar - Tomem água!



## **Dict - Aumentando a complexidade**

Vamos complicar nosso exemplo do professor mais uma vez.

• Agora, não queremos calcular a nota de somente uma aluna, mas sim da

turma toda.



## Dict - Aumentando a complexidade - Aninhando dicts

```
notas-dict-nested.py > ...
      def calcula_media(notas):
          soma_notas = 0
          for nota in notas.values():
              soma_notas = soma_notas + nota
          return soma_notas/len(notas)
      notas = {
          "Belisa":
          "nota2": 98,
          "nota1": 98,
          "nota2": 95,
          "nota3": 100
          "nota1": 100,
          "nota2": 98,
          "nota3": 99
      print(calcula_media(notas))
```

#### Dict - Acessando dicts aninhados

```
notas-dict-nested.py > ...
     def calcula media(notas):
         soma notas = 0
         for aluna in notas.values():
              for nota in aluna.values():
                  soma_notas = soma_notas + nota
         return soma_notas/len(notas)
     notas = {
          "nota2": 98.
         "nota1": 98,
          "nota2": 95.
         "nota3": 100
          "nota1": 100,
          "nota2": 98,
     print(calcula_media(notas))
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS
belisa@belisa-Vostro-3520:~/Documents/reprograma/aula4$ python3 notas-dict-nested.py
259.3333333333333
belisa@belisa-Vostro-3520:~/Documents/reprograma/aula4$
```

```
def calcula_media(notas):
         for aluna in notas.values():
             soma_notas = 0
             for nota in aluna.values():
                 soma_notas = soma_notas + nota
             aluna.update({"media": (soma_notas/len(aluna))})
         return notas
     notas = {
         "Belisa":
             "nota1": 15,
             "nota2": 98,
             "nota3": 75
             "nota1": 98,
             "nota3": 100
             "nota1": 100,
             "nota2": 98,
25
             "nota3": 99
     notas = calcula_media(notas)
     for nome, notas in notas.items():
         print(f"A aluna {nome} obteve média {notas['media']}")
```

# Laços de repetição - Comandos extra

- As vezes, é necessário controlar o fluxo de execução dentro de um for.
- Por exemplo, podemos desejar encerrar a execução do for se algo específico acontecer.
- Ou então podemos querer pular um valor iterado dependendo de alguma condição.
- Também podemos desejar executar alguma coisa específica quando nosso laço acabar.
- Para isso, vamos usar os comandos break, continue e else (sim, o mesmo else do if)

## Material extra e exercícios

- Material sobre namespaces e entendendo melhor o que acontece dentro de uma função:
  - https://www.youtube.com/watch?v=nWmPEgTwGMM
  - O vídeo todo é excelente, mas a parte sobre funções começa a partir de 1:08:40

