

Aula 04 – Estruturas de repetição em Python 3

For e While

Parte 1 – Revisão Teórica

Introdução (Estrutura de controle, Loop ou laços)

As estruturas de repetição (também conhecidas como laços ou loops) geralmente são utilizadas para processar uma coleção de dados, linhas de um arquivo, registros de um banco de dados que precisam ser processados por um mesmo bloco de código.

Essas coleções de dados podem ser linhas de um arquivo de texto ou uma busca de clientes que você buscou em um banco de dados.

Bloco de instrução

O Bloco de Instrução é o conjunto de instrução que SEMPRE está na sequência das estruturas e sempre estarão na indentação, um nível hierárquico a frente da definição da estrutura.

Ciclo ou laço

O ciclo é o nome chamado a uma única repetição do bloco de instrução. Do dicionário Aurélio, temos que ciclo significa:

"Série de fenômenos que se sucedem numa ordem determinada"

Então, temos que o bloco de instrução executado por definição, sempre será o mesmo. Assim, devemos pensar no conceito de ciclo para as linguagens de programação, como a execução de um mesmo bloco de código. Talvez, o bloco tenha condições que determinem, conforme a expressão definida, blocos de instruções diferentes, porém, no geral, esse bloco sempre estará contido dentro do bloco de instrução da estrutura de repetição.

Estruturas de Repetição

As linguagens de programação atuais fornecem meios para a repetição de um bloco de instrução de forma mais simples e menos verbosas do que suas antecessoras. Essas estruturas são tecnicamente chamadas de **Iteradores**, isto é, **repetidores**.

Normalmente, as linguagens disponibilizam um iterador que repetira uma quantia finita de vezes, e outro que repetirá, enquanto uma condição seja verdadeira, ou seja, um repetidor condicional.

Iteradores em Python

O Python disponibiliza 2 iteradores:

Iterador/Estrutura de controle	Descrição
for..in.	✓ é um iterador finito, isto é, que repetira por uma quantidade de vezes conhecido previamente

while	✓ o iterador condicional, que repetira um determinado bloco de código enquanto a condição definida no cabeçalho da estrutura for verdadeiro.
--------------	--

Estrutura de controle (For....in)

A instrução **for** se caracteriza por obrigar o programador a definir, explicitamente em seu cabeçalho, a quantidade de vezes [ciclos] que será executado.

A quantidade de ciclos é determinada pela quantidade de elementos contido na lista declarada junto com a instrução **for**.

Dessa forma, será executado um ciclo para cada elemento **isoladamente**.

O **Laço de Repetição for** do Python se assemelha ao **for each** encontrado linguagens como o Java, PHP, C# e etc.

Inclusive, a característica encontrada em todas as linguagens é a mesma que temos aqui em Python: uma estrutura simples e compacta para percorrer todos elementos das coleções ou estruturas que contenham listas de objetos.

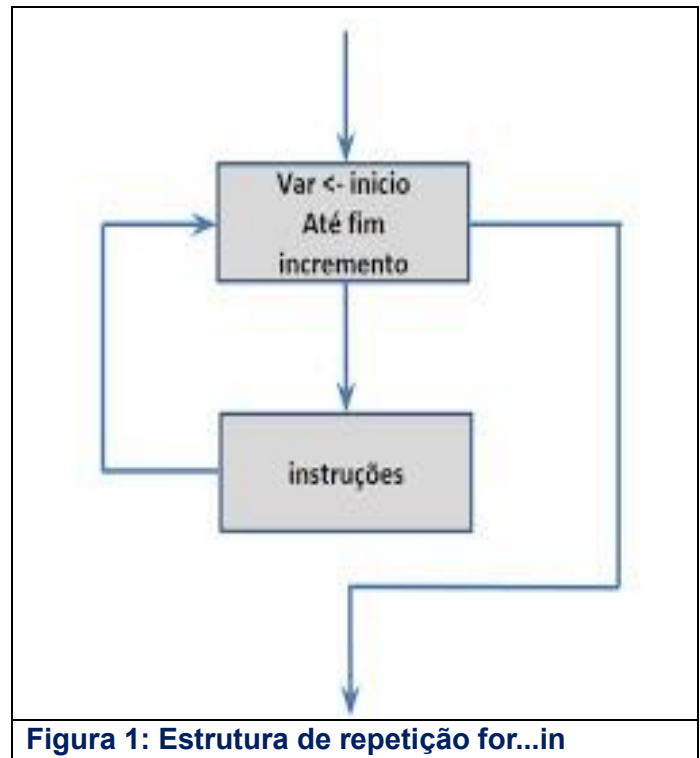


Figura 1: Estrutura de repetição for...in

A sintaxe do Python não possui a estrutura de repetição **for** tradicional, onde define-se uma variável, condição e incremento no cabeçalho da estrutura.

A ideia, é o trabalho e manipulação de estrutura iteráveis, isto é, a definição de iteradores, ou melhor: objetos iteráveis - iterators.

Sendo a estrutura de repetição mais utilizada na linguagem Python, ela aceita

- a) **sequências estáticas** e
- b) **geradas por iteradores**.

Iteradores são estruturas que permitem acesso aos itens de uma **coleção de elementos** de forma **sequencial**.

Quando um laço **for** é executado, a referência aponta para um determinado elemento da sequência. Esta referência é sempre atualizada a cada nova iteração, de forma que o bloco de código do laço processe o elemento correspondente.

A DEFINIÇÃO DA INSTRUÇÃO for

for <variável> in <objeto iterável>: bloco de instrução	for {referência} in {sequência} : {código do bloco}
Listagem 1	Listagem 2

A estrutura **for** exige, inicialmente, a definição de uma variável e, em seguida, a lista que será iterada. A seguir, temos o esquema para o uso da instrução **for**.

A variável a ser declarada na primeira parte da estrutura, receberá, a cada ciclo, um elemento contido na lista que está sendo iterada. Ao término, todos elementos terão sido percorridos e, a cada ciclo, o elemento seguinte contido no objeto iterável terá sido passado pela variável definida inicialmente.

Exemplo de uma lista numérica

```
print("Exemplo de uma lista numerica")
for contador in [0,1,2,3,4,5,6,7]:
    print(contador)
```

Listagem 3

Nós podemos ler o laço de repetição definido acima `for item in [0,1,2,3,4,5,6,7]:` da seguinte maneira. Para cada elemento contido na lista `[0,1,2,3,4,5,6,7]` execute o bloco de código a seguir e a cada execução, atribua à variável `item` `for item in` um item da lista.

Ou então, podemos ler como estamos estudando... Acima, definimos a variável `item` e uma lista com 5 elementos `[0,1,2,3,4,5,6,7]`.

A cada ciclo, o elemento seguinte é atribuído à variável `item` e ao término, teremos executado um ciclo individual para cada elemento contido na lista.

Exemplo de uma tabuada

```
print("Tabuada do 4")

for contador in [0,1,2,3,4,5,6,7,8,9,10]:
    tabuada = contador * 4
    print("4 x " +str(contador) + " = "+str(tabuada))
```

Listagem 4

Exemplo de uma lista de string

```
print("Lista de animais da fazenda")
print("Inicio do loop for....in")

for i in ["Cachorro", "Gato", "Cavalo", "Galinha", "porco", "Canario"] :
    print(i)
print("Fim do loop for....in")
```

Listagem 5

Cláusula `break`, `continue` e `else`

Caso necessário, podemos utilizar a

- a) cláusula **break** para interromper o laço e
- b) **continue** para pular para a próxima iteração.
- c) Podemos, também, colocar um bloco de código dentro de um **else**, para que este seja executado ao final do laço caso ele não for interrompido por um **break**.

```
for {referência} in {sequência} :  
    {código do bloco}  
    continue  
    break
```

Listagem 7

```
for {referência} in {sequência} :  
    {bloco de código}  
    continue  
    break  
else:  
    {bloco de código}
```

Listagem 8

A INSTRUÇÃO else

```
print("Exemplo de for com a clausula break e else")  
  
for x in [0,1,2,3,4,5,6,7,8,9,10]:  
    print(x)  
    #break  
else:  
    print("Saida do else, executou o bloco da instrução else")  
  
print("Estrutura for executou todos os ciclos definidos e no final, executou o bloco da instrução else")
```

Listagem 9

Acima, temos que a estrutura for executou todos os ciclos definidos e no final, executou o bloco da instrução else.

```
status = False  
for cont in [0,1,2,3,4,5,6,7,8,9,10]:  
    print(cont)  
    if(True):  
        status = True  
        #break  
  
if(status):  
    print("Imprime o valor da variavel status", status)
```

Listagem 10

Acima, temos que a estrutura for o primeiro ciclo e neste a instrução break foi invocada, portanto, o laço foi finalizado e o bloco else não foi executado.

EXEMPLO 11: IMPRIMINDO TODAS AS LETRAS DE UMA STRING

Uma String é um conjunto de caracteres e assim, podemos iterar qualquer texto, por exemplo.

```
print()
print("Exemplo de uma lista de palavras")
for letras in "Python":
    print("ciclo: ", letras)
```

Listagem 11

EXEMPLO 12: IMPRIMINDO TODOS OS ITENS DE UMA LISTA

```
print("Lista de animais da fazenda") print("Inicio do loop for....in")
for i in ["Cachorro", "Gato", "Cavalo", "Galinha", "porco", "Canario"] :
    print(i)
print("Fim do loop for....in")
```

Listagem 12

EXEMPLO 13: IMPRIMINDO TODOS OS ITENS DE UMA LISTA

```
print("Lista de Linguagens de Programação")
lingProgramacao = ["Java", "C", "Python", "Lua", "Cobol", "Pascal", "C++"]

for lingua in lingProgramacao:
    print("Linguagem contida na variável 'lingua' neste ciclo: ", lingua)
```

Listagem 13

DEFINIÇÃO DA FUNÇÃO range()

A função **range()** retorna uma série numérica no intervalo enviado como argumento. A série retornada é um objeto iterável tipo range e os elementos contidos serão gerados sob demanda.

É comum o uso da função **range()** com a estrutura for loop. Desta forma temos que a cada ciclo o próximo elemento da sequência será utilizado de tal forma que é possível partirmos de um ponto e ir incrementando, decrementando x unidades.

Aqui tem a [definição da função range\(\) encontrada na documentação oficial do Python](#):

INTERVALO ABERTO E FECHADO

intervalo aberto] x [<p>Quando os colchetes se opõem ao número, temos um intervalo aberto. Assim, todos os números no intervalo estarão contidos na sequência numérica, menos os extremos.</p> <p>Intervalo:]a, b[Conjunto: $\{x \in \mathbb{R} \mid a < x < b\}$ Tradução: números reais maiores do que a e menores do que b (quando o intervalo está aberto, entende-se que todos os números do intervalo estão contidos, menos o(s) extremo(s)).</p>
------------------------	--

intervalo fechado [x]	<p>Quando o colchete estiver de "acordo" (no sentido normal) ao número, tem-se um intervalo fechado.</p> <p>Assim, todos os números contidos neste intervalo estarão contidos inclusive os extremos da sequência. Intervalo: [a, b] Conjunto: $\{x \in \mathbb{R} \mid a \leq x \leq b\}$ Tradução: números reais maiores ou igual a a e menores ou igual a b (quando o intervalo está fechado, entende-se que todos os números do intervalo estão contidos, inclusive o(s) extremo(s)).</p>
--------------------------------	--

VOLTANDO AO PYTHON

Agora que já relembramos alguns conceitos da **matemática**, vamos rever a notação da função **range()**:

range(stop)	<p>é possível invocar a função definindo somente, o intervalo final, ou seja, definindo somente o parâmetro <i>stop</i>.</p> <p>Por default, que o início da sequência é igual a 0 e o final (stop), igual ao valor que está sendo enviado como argumento.</p>
range([start], stop[, step])	<p>Notação completa, temos a capacidade de definir</p> <ul style="list-style-type: none"> (1) Start: o início da sequência (2) Stop: o último elemento e (3) Step o passo, isto é, o intervalo entre cada elemento. <p>A função range() exige a definição do último elemento da sequência numérica. Por padrão, o parâmetro start será igual a 0 e o step igual a 1.</p>

EXEMPLO DE USO DA PRIMEIRA NOTAÇÃO

Vamos desenvolver um simples exemplo demonstrando o uso da primeira notação estudada, assim, vamos invocar a função **range()** e passar como argumento o últimos elemento da sequência numérica que desejamos.

É importante observar que na versão **3x do Python** será retornado, pela função **range()**, um objeto iterável, e não uma lista contendo os elementos propriamente dito. Por essa razão, iremos forçar a conversão do objeto iterável para o **objeto list**.

DEFINIÇÃO: `range(stop[])` DEFINIÇÃO MATEMÁTICA: $[0,5[$

print(list(range(10)))
Listagem 14

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

A função retornou uma lista contendo 5 elementos, onde o primeiro é o 0 e o último 4.

O primeiro elemento é igual a 0, até porque, quando não definido, assume-se que **start = 0**. O último elemento é o 4, até porque, o parâmetro stop possui o intervalo aberto, isto é, o número definido não estará contido na sequência numérica.

EXEMPLOS DE USO DA SEGUNDA NOTAÇÃO

Novamente, iremos converter o objeto iterável retornado pela função **range()** para o tipo **list**, para que o mesmo, tenha seus elementos impressos na saída padrão.

```
print(list(range(2, 6)))  
print("Lista de numeros com passo 1 no intervalo [2 ate 6]")
```

Listagem 15

Na listagem 15, foi definido o parâmetro **start=2** e **stop=6**. Onde o primeiro elemento foi o 2, enquanto o último elemento, igual a 5 passo (**step**) = 1.

Neste exemplo é gerado uma lista que contenha 4 objetos numéricos e que o primeiro seja igual a 2 e o último igual a 5 ou **[2, 3, 4, 5]**

PASSO OU INTERVALO ENTRE OS ELEMENTOS

O step é o intervalo entre cada elemento numérico. Se por exemplo, gerarmos uma sequência numérica entre 0 e 10 e definirmos, será assumido por padrão, que o **step** é igual a 1 e por essa razão, o elemento seguinte estará a 1 unidade do elemento anterior, e a lista será igual a: **[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]**. Agora, se nós gerarmos uma lista entre 0 e 10, porém, definirmos o passo como sendo igual a 2, a seguinte lista será retornada: **[0, 2, 4, 6, 8]**.

Em ambos os casos os estamos trabalhando com uma sequência numérica num mesmo intervalo, porém, no primeiro exemplo o intervalo entre os elementos é igual a 1 unidade, enquanto que no segundo exemplo, o espaço entre os elementos é de 2 unidades.

No código a seguir, vamos gerar uma lista no intervalo 2 à 16 e o step será igual a 2 unidades.

```
print(list(range(2, 16, 2)))
```

Listagem 16

[2, 4, 6, 8, 10, 12, 14]

Em seguida, novamente com a função **range()** definimos uma segunda sequência numérica no intervalo de 0 à 30 e o step igual a 3. Logo, foi impresso uma lista, contendo 10 elementos, onde o primeiro é o 0 e o último o 27. Também é possível observar, que a sequência gerada é a tabuada de 3, faltando somente o último valor e para isso, bastava somente definir o intervalo final como sendo igual a 31.

```
print(list(range(0, 30, 3)))
```

Listagem 17

[0, 3, 6, 9, 12, 15, 18, 21, 24, 27]

EXEMPLOS DA DOCUMENTAÇÃO OFICIAL DO PYTHON

Os exemplos a seguir, foram extraídos da documentação oficial do Python e demonstram situações interessantes com o uso da função **range()**. Vale a pena o estudo de cada situação detalhadamente.

```
#gerando uma sequência de 0 à 9
list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

#gerando uma sequência de 1 à 11 list(range(1, 11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

#gerando uma sequência de 0 à 30 com
step=5 list(range(0, 30, 5)) [0, 5, 10, 15, 20, 25]

#gerando uma sequência os 5 primeiros números da tabuada de 3 list(range(0, 10, 3))
[0, 3, 6, 9]

#gerando uma sequência numérica negativa list(range(0, -10, -1))
```

```
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]

#gerando uma sequência numérica vazia list(range(0))
[]

#gerando uma sequência numérica vazia,
#onde o intervalo inicial é maior do que o final
#por essa razão, há lista é nula
list(range(1, 0))
[]
```

Listagem 18

Veja agora um exemplo prático onde somamos de 0 a 20:

```
soma = 0 print("Soma dos numeros de 1 ate 20")
for i in range(1, 20):
    soma = soma + i
print("Valor da total da soma = " +str(soma))
```

Listagem 18

Exemplo de uma tabuada do 4

```
print("Tabuada do 4")
for contador in 0,1,2,3,4,5,6,7,8,9,10]:
    tabuada = contador * 4
    print("4 x " +str(contador) + " = "+str(tabuada))
```

Listagem 19

While

O **Laço de Repetição (loop) while** repete um bloco de instrução enquanto a condição definida em seu cabeçalho for verdadeira.

O funcionamento da **estrutura if**, pense na **estrutura while** como sendo a **estrutura if** mas que ao invés de executar o seu bloco de instrução uma única vez, executará enquanto a expressão definida for igual a **True**.

O que diferencia o **if** do **while** é só e somente só a quantidade de vezes que o seu bloco de instrução será executado!

O **While** é utilizado quando não é possível definir a quantidade de iterações que podem ocorrer e quando não há uma sequência a seguir.

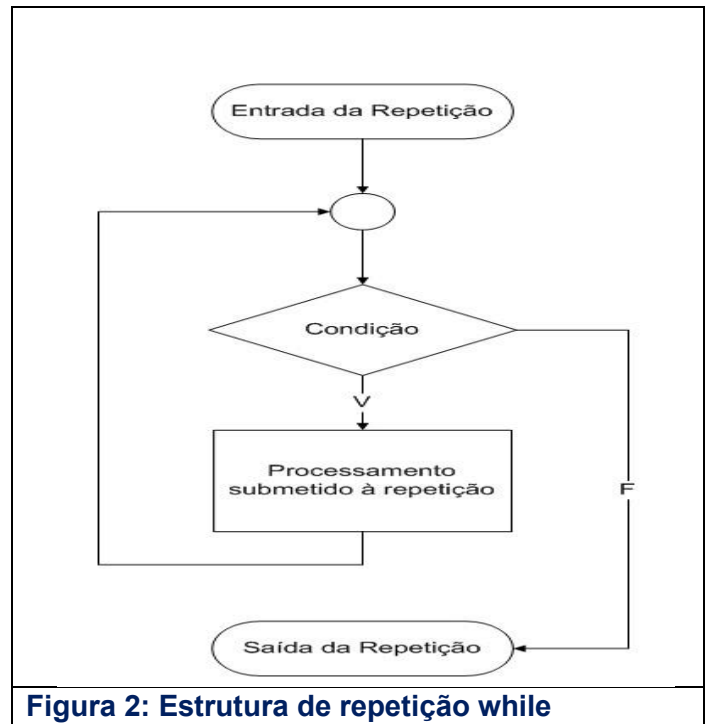


Figura 2: Estrutura de repetição while

Sintaxe:

```

while {condição}:
    {bloco de código}
    continue
    break
else:
    {bloco de código}
  
```

Listagem 20

Assim como o **For**, as cláusulas **break**, **continue** e **else** podem ser utilizadas da mesma maneira.

Exemplo

```

cont = 0

while (cont <=5):
    print("Valor do contador i = " +str(cont))
    cont = cont + 1
  
```

Listagem 21

É importante entender que a expressão **deverá**, obrigatoriamente, deixar de ser verdadeira para que a repetição seja interrompida. Do contrário, a aplicação irá **congelar**, isto é, a aplicação irá travar.

A listagem 22 é um exemplo utilizando os comandos em Português:

```
cond = True enquanto(cond):
    print("Bloco de instrução da estrutura while()")
```

Listagem 22

O bloco de instruções da estrutura enquanto acima será executado até que a variável que chamamos de **cond** tenha seu valor igual a **True**, situação que jamais ocorrerá, pois, nada está sendo feito para que o valor contido seja alterado.

ESTRUTURA else

A linguagem Python define a instrução else como uma estrutura dependente da instrução while cujo funcionamento novamente é análogo ao estudado na instrução if.

Desta forma, em Python, há 4 estruturas em que a instrução else pode ser utilizado para definirmos o bloco de instrução a ser executando quando a condição verificada **deixar de ser verdadeiro**.

No caso, temos a **instrução if**, a **instrução while** a instrução for que será estudado na sequência e a estrutura de tratamento de exceção **try except** que será estudado futuramente.

Vamos analisar agora uma situação que inicialmente será verdadeiro e após executar uma quantidade de vezes o bloco de código a expressão quando avaliada será igual a False.

```
condicao = True
while(condicao): #<1>
    print("BLOCO while() e condicao==True") #<2>
    condicao = False
else:
    print("BLOCO ELSE e condicao==False")
```

A execução do código acima envia o seguinte resultado pra saída padrão:
BLOCO while() e condicao==True
BLOCO ELSE e condicao==False

Listagem 23

O que podemos concluir é que:

#<1>: O bloco verdadeiro da estrutura while foi executado, pois, a lógica é repetir até deixar de ser verdadeiro

#<2>: Condição False

#<3>: e então, a variável condição é definida como

False - #<1>: o cursor de execução volta para o cabeçalho da estrutura while, analisa a condição definida e, como o resultado é igual a False o bloco de instrução else será executado, POIS, igualmente ocorre com a instrução if, o bloco else é executado nas vezes em que a condição NÃO for verdadeira.

Agora, se por alguma razão a instrução break for invocada, o bloco else não será executado, pois, a lógica estabelecida é que o bloco else só será executado ass vezes em que a condição for False e no caso, isso não aconteceu.

A seguir, temos o mesmo exemplo, porém agora, a instrução break será invocada.

```
condicao = True while(condicao):
    print("BLOCO while() e
    condicao==True")    condicao = False
    break
else:
    print("BLOCO ELSE e condicao==False")
```

Listagem 24

A execução do código acima envia o seguinte resultado pra saída padrão:
BLOCO while() e condicao==True

Ou seja, o bloco True da instrução while foi executado, porém, como a instrução break interrompeu a execução de todo o ciclo, o bloco else, isto é, o bloco que é executado nas vezes em que a condição não é verdadeira não foi executado.

Assim, o bloco else pode ser executado uma única vez, nos casos em que a expressão definida no cabeçalho da instrução while, deixar de ser verdadeira.

```
conta = 0 while(conta <= 10):
    conta += 1
    print(conta)
else:
    print("Valor da variável conta é: ", conta)
```

Listagem 25

No código acima, o Bloco de Instrução para quando a expressão for verdadeira será executado até que o valor da variável conta seja igual a 11. Nesse momento, quando a expressão passar a ser falsa, o Bloco da Instrução else será executado e o Laço de Repetição terá chego ao fim.

É importante observar que a instrução else funciona da mesma maneira àquela estudada na Tomada de Decisão.

INTERRUPÇÃO com break

A instrução break finaliza abruptamente a execução das estruturas de repetição e assim, quando essa instrução for executada, o cursor de execução irá interromper a execução das instruções definidas pela Estrutura de Repetição e irá saltar para a linha seguinte após o Laço de Repetição.

Em outras palavras, temos que quando a instrução break for utilizada, o cursor de execução não irá passar por dentro do bloco definido pela instrução else, até porque, a instrução break encerra, **imediatamente** a execução da Estrutura de Repetição.

EXEMPLO

```
x = 0
print("while")
while(x<10):
    print(x)
    x=x+1
else:
    print("else")
print("fim")
```

Listagem 26

Glossário

atribuição (*assignment*)

Comando que atribui um valor a uma variável.

avaliar (*evaluate*)

Simplificar uma expressão através da realização de operações, para produzir um valor único.

comando (*statement*)

Trecho de código que representa uma instrução ou ação. Até agora, os comandos vistos foram de atribuição e exibição.

comentário (*comment*)

Informação em um programa dirigida a outros programadores (ou qualquer pessoa que esteja lendo o código fonte) e que não tem efeito na execução do programa.

composição (*composition*)

Habilidade de combinar expressões e comandos simples em expressões e comandos compostos, de forma a representar operações complexas de forma concisa.

concatenar (*concatenate*)

Juntar dois operandos lado a lado.

diagrama de estado (*state diagram*)

Representação gráfica de um conjunto de variáveis e os valores aos quais elas se referem.

divisão inteira (*integer division*)

Operação que divide um inteiro por outro e resulta em um inteiro. A divisão inteira resulta no número de vezes que o numerador é divisível pelo denominador e descarta qualquer resto.

expressão (*expression*)

Combinação de variáveis, operadores e valores, que representa um resultado único.

operando (*operand*)

Um dos valores sobre o qual o operador opera.

operador (*operator*)

Símbolo especial que representa uma computação simples, como adição, multiplicação ou concatenação de strings.

palavra-chave (*keyword*)

Palavra reservada usada pelo compilador/interpretador para analisar o programa; você não pode usar palavras-chave como `if`, `def`, e `while` como nomes de variáveis. **ponto-flutuante (*floating-point*)**

Formato para representar números que possuem partes fracionárias.

regras de precedência (*rules of precedence*)

O conjunto de regras que governa a ordem em que expressões envolvendo múltiplos operadores e operandos são avaliadas. **tipo (*type*)**

Um conjunto de valores. O tipo de um valor determina como ele pode ser usado em expressões. Até agora, os tipos vistos são: inteiros (tipo `int`), números em ponto-flutuante (tipo `float`) e strings (tipo `string`)

valor (*value*)

Um número ou string (ou outra coisa que ainda vamos conhecer) que pode ser atribuída a uma variável ou computada em uma expressão. **variável (*variable*)**

Nome que se refere a um valor.

Bibliografia

Python, Escreva seus primeiros programas, Felipe Cruz, Casa do Código.

Curso Introdutório de Python, Grupy-Sanca, 09 de maio de 2019

Tutorial Python, Release 2.4.2. Guido van Rossum, Fred L. Drake, Jr., editor. Tradução: Python Brasil

Curso Python 3. Juracy Filho e Leonardo Leitão. Cod3r.com.br

https://aprendacompy.readthedocs.io/pt/latest/capitulo_01.html

<https://www.devmedia.com.br/python-trabalhando-com-variaveis/38644>

<https://cadernodelaboratorio.com.br/2017/12/01/variaveis-em-python3/>

Parte 2 – Prática (mão na massa)!!!

Para as atividades práticas vamos utilizar o VS Code, que deve estar previamente configurado para executar o python)

Para a criação dos arquivos (com extensão .py) vamos utilizar o editor de texto

Para executar, deve utilizar o interpretador do VS Code, temos 2 opções

- Apertar o Botão direito e selecionar a opção **RUN PYTHON TERMINAL**
- No menu superior, selecionar Debug -> **START WITHOUT DEBUGGING**

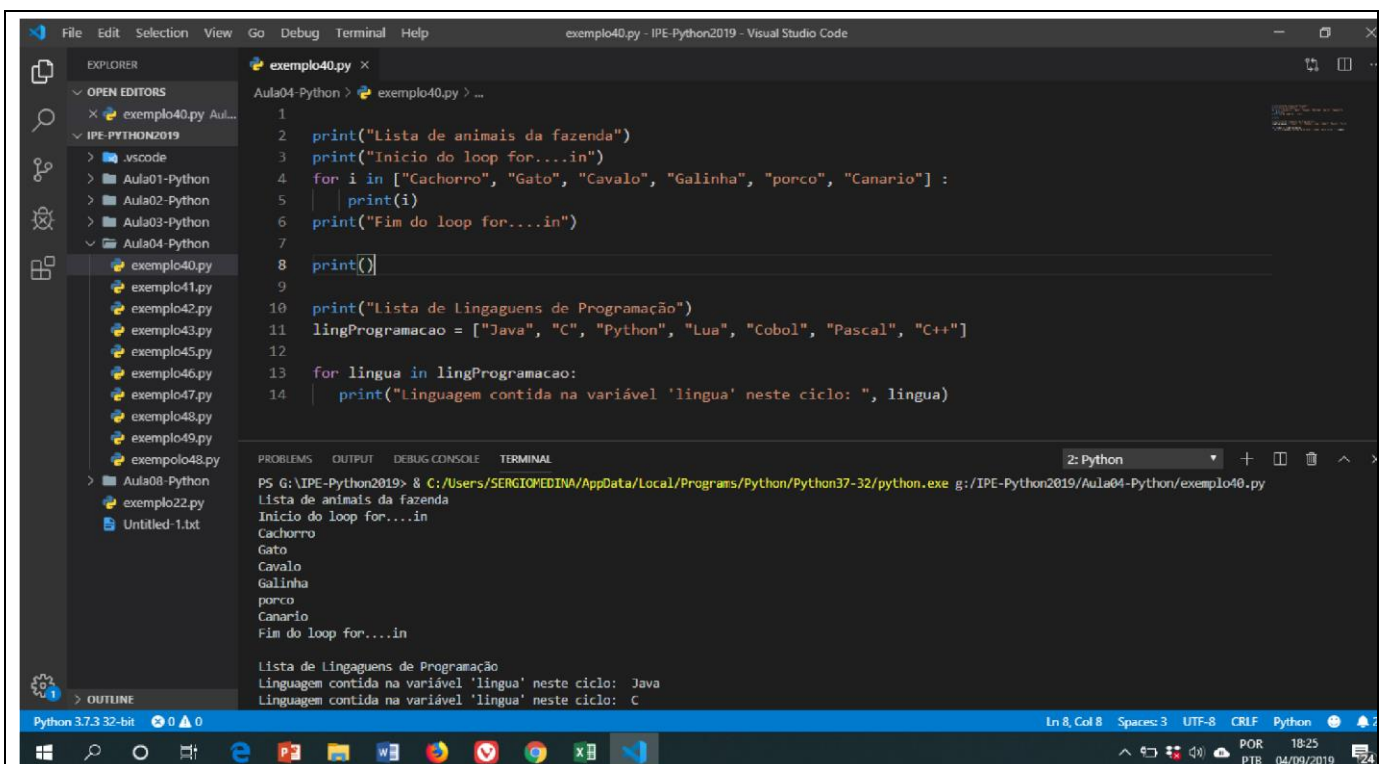
Para limpar o console, utilizar o comando **clear** ou **cls**

2) Criação do Projeto no VS CODE

- Para isto crie uma pasta no seu pendrive com o nome **ALP-Python-2023**.
- Dentro do **ALP-Python-2023**, criar a pasta **Aula04-Python**
- Utilizando o VS CODE acesse a pasta criada (**ALP-Python-2023/ Aula04-Python**)
- Para isso, acione a opção **File > New > File** e crie os seguintes (exemplo40.py, exemplo41.py exemplo42.py,..... exemplo49.py), para melhor organização (**BOA PRÁTICA!!!**)

Exemplo 40

PASTA: ALP-Python-2023/ Aula04-Python
arquivo: exemplo40.py



```
1
2 print("Lista de animais da fazenda")
3 print("Inicio do loop for....in")
4 for i in ["Cachorro", "Gato", "Cavalo", "Galinha", "porco", "Canario"] :
5     print(i)
6 print("Fim do loop for....in")
7
8 print()
9
10 print("Lista de linguagens de Programação")
11 lingProgramacao = ["Java", "C", "Python", "Lua", "Cobol", "Pascal", "C++"]
12
13 for lingua in lingProgramacao:
14     print("Linguagem contida na variável 'lingua' neste ciclo: ", lingua)
```

PS G:\IPE-Python2019> & C:\Users\SERGIOMEDINA\AppData\Local\Programs\Python\Python37-32\python.exe g:/IPE-Python2019/Aula04-Python/exemplo40.py

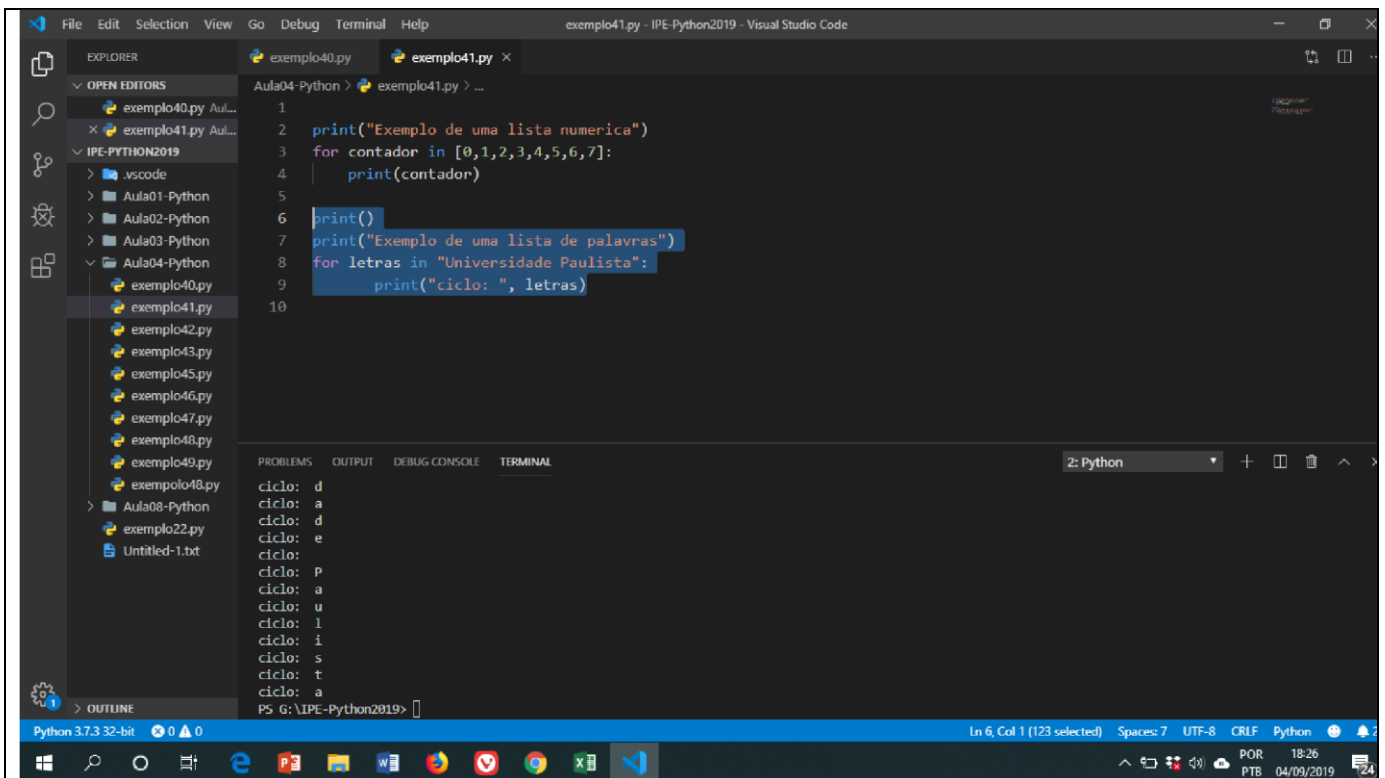
Lista de animais da fazenda
Inicio do loop for....in
Cachorro
Gato
Cavalo
Galinha
porco
Canario
Fim do loop for....in

Lista de Linguagens de Programação
Linguagem contida na variável 'lingua' neste ciclo: Java
Linguagem contida na variável 'lingua' neste ciclo: C

Figura A: exemplo40.py

Exemplo 41

PASTA: ALP-Python-2023/ Aula04-Python
arquivo: exemplo41.py



```
File Edit Selection View Go Debug Terminal Help
exemplo41.py - IPE-Python2019 - Visual Studio Code

EXPLORER
  OPEN EDITORS
    exemplo40.py Aul...
    exemplo41.py Aul...
  IPE-PYTHON2019
    .vscode
    Aula01-Python
    Aula02-Python
    Aula03-Python
    Aula04-Python
      exemplo40.py
      exemplo41.py
      exemplo42.py
      exemplo43.py
      exemplo45.py
      exemplo46.py
      exemplo47.py
      exemplo48.py
      exemplo49.py
      exemplo48.py
    Aula08-Python
      exemplo22.py
      Untitled-1.txt

  OUTLINE

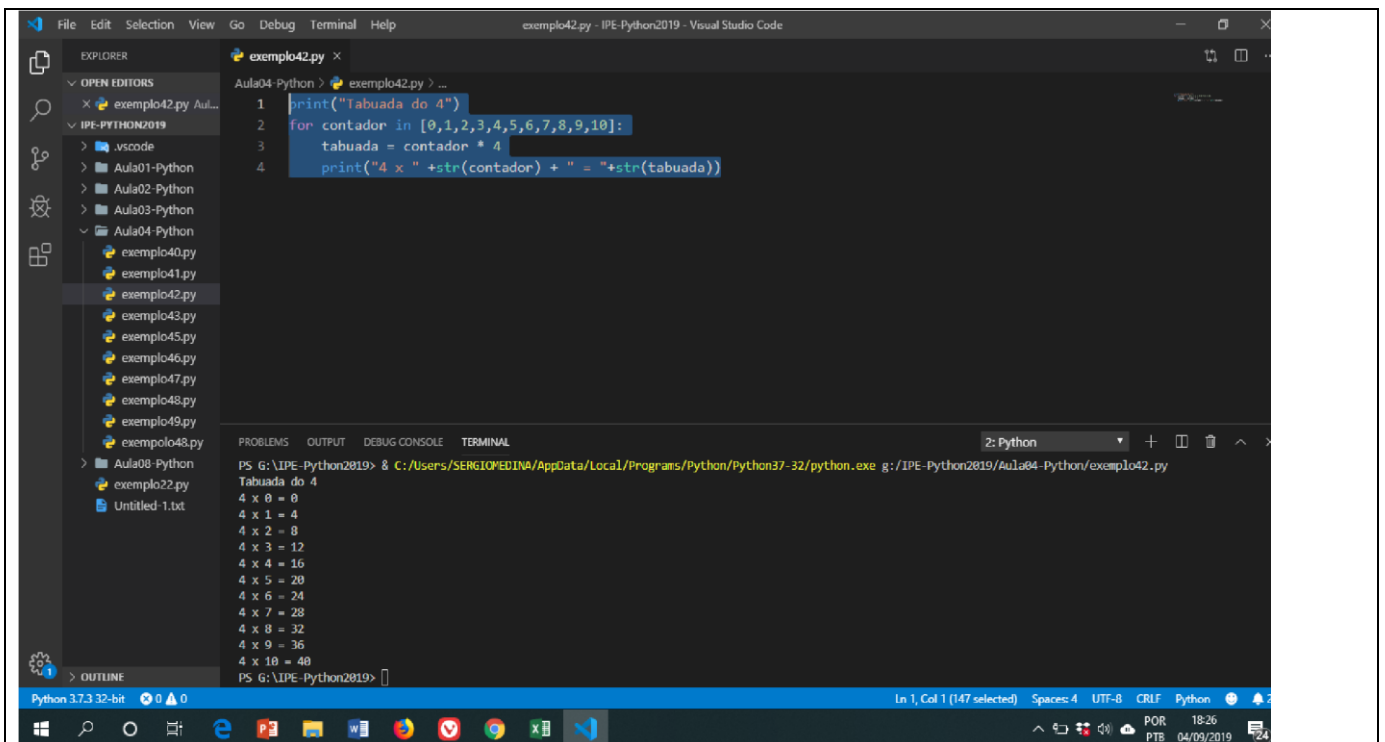
Aula04-Python > exemplo41.py > ...
1
2 print("Exemplo de uma lista numerica")
3 for contador in [0,1,2,3,4,5,6,7]:
4     print(contador)
5
6 print()
7 print("Exemplo de uma lista de palavras")
8 for letras in "Universidade Paulista":
9     print("ciclo: ", letras)
10

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: Python
ciclo: d
ciclo: a
ciclo: d
ciclo: e
ciclo: c
ciclo: P
ciclo: a
ciclo: u
ciclo: l
ciclo: i
ciclo: s
ciclo: t
ciclo: a
PS G:\IPE-Python2019>
```

Figura A: exemplo41.py

Exemplo 42

PASTA: ALP-Python-2023/ Aula04-Python
arquivo: exemplo42.py



```
File Edit Selection View Go Debug Terminal Help
exemplo42.py - IPE-Python2019 - Visual Studio Code

EXPLORER
  OPEN EDITORS
    exemplo42.py Aul...
  IPE-PYTHON2019
    .vscode
    Aula01-Python
    Aula02-Python
    Aula03-Python
    Aula04-Python
      exemplo40.py
      exemplo41.py
      exemplo42.py
      exemplo43.py
      exemplo45.py
      exemplo46.py
      exemplo47.py
      exemplo48.py
      exemplo49.py
      exemplo48.py
    Aula08-Python
      exemplo22.py
      Untitled-1.txt

  OUTLINE

Aula04-Python > exemplo42.py > ...
1 print("Tabuada do 4")
2 for contador in [0,1,2,3,4,5,6,7,8,9,10]:
3     tabuada = contador * 4
4     print("4 x " + str(contador) + " = " + str(tabuada))

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: Python
PS G:\IPE-Python2019> & C:\Users\SERGIOMEDEIRA\AppData\Local\Programs\Python\Python37-32\python.exe g:/IPE-Python2019/Aula04-Python/exemplo42.py
Tabuada do 4
4 x 0 = 0
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
4 x 10 = 40
PS G:\IPE-Python2019>
```

Figura B: exemplo42.py

PASTA: ALP-Python-2023/ Aula04-Python
arquivo: exemplo43.py



PASTA: ALP-Python-2023/ Aula04-Python
arquivo: exemplo44.py



Exemplo 45

PASTA: ALP-Python-2023/ Aula04-Python
arquivo: **exemplo45.py**

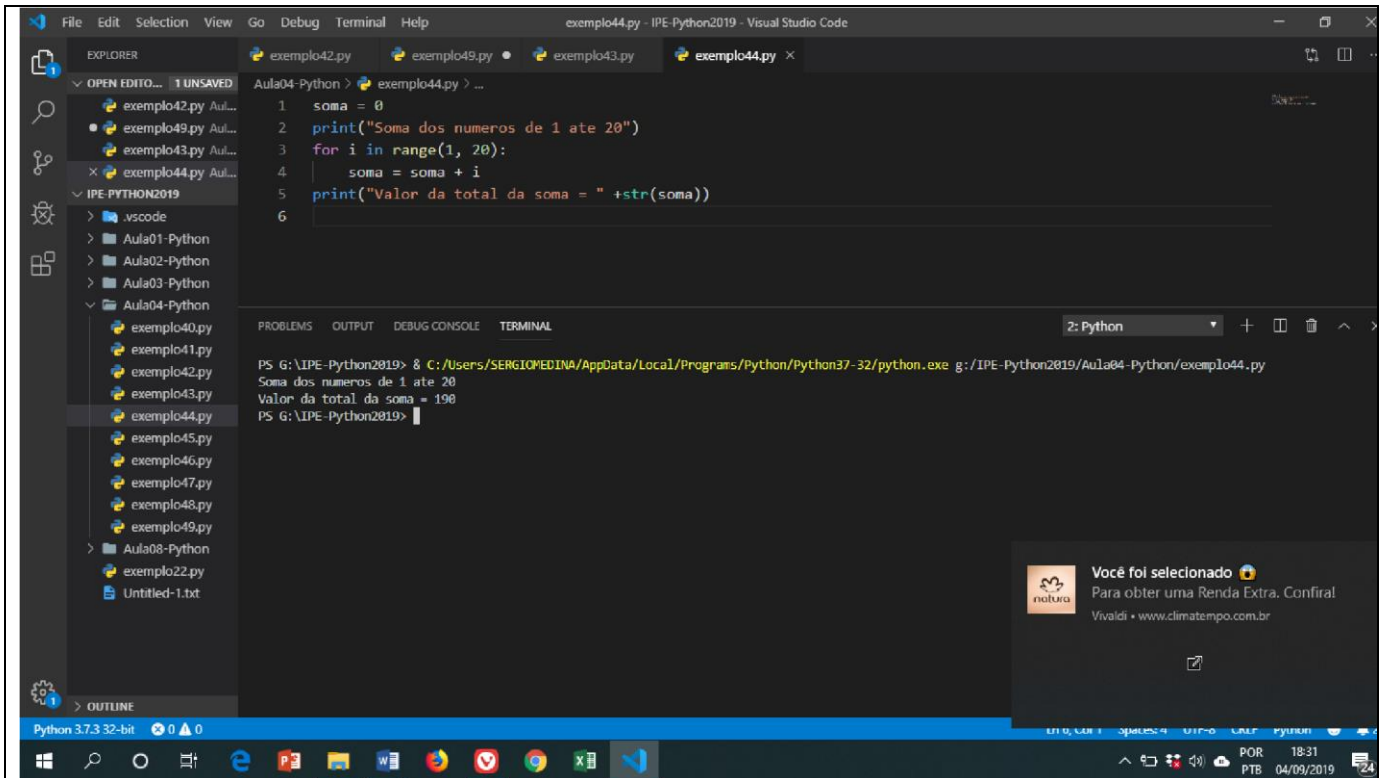


Figura F: exemplo45.py

Exemplo 46

PASTA: ALP-Python-2023/ Aula04-Python
arquivo: **exemplo46.py**

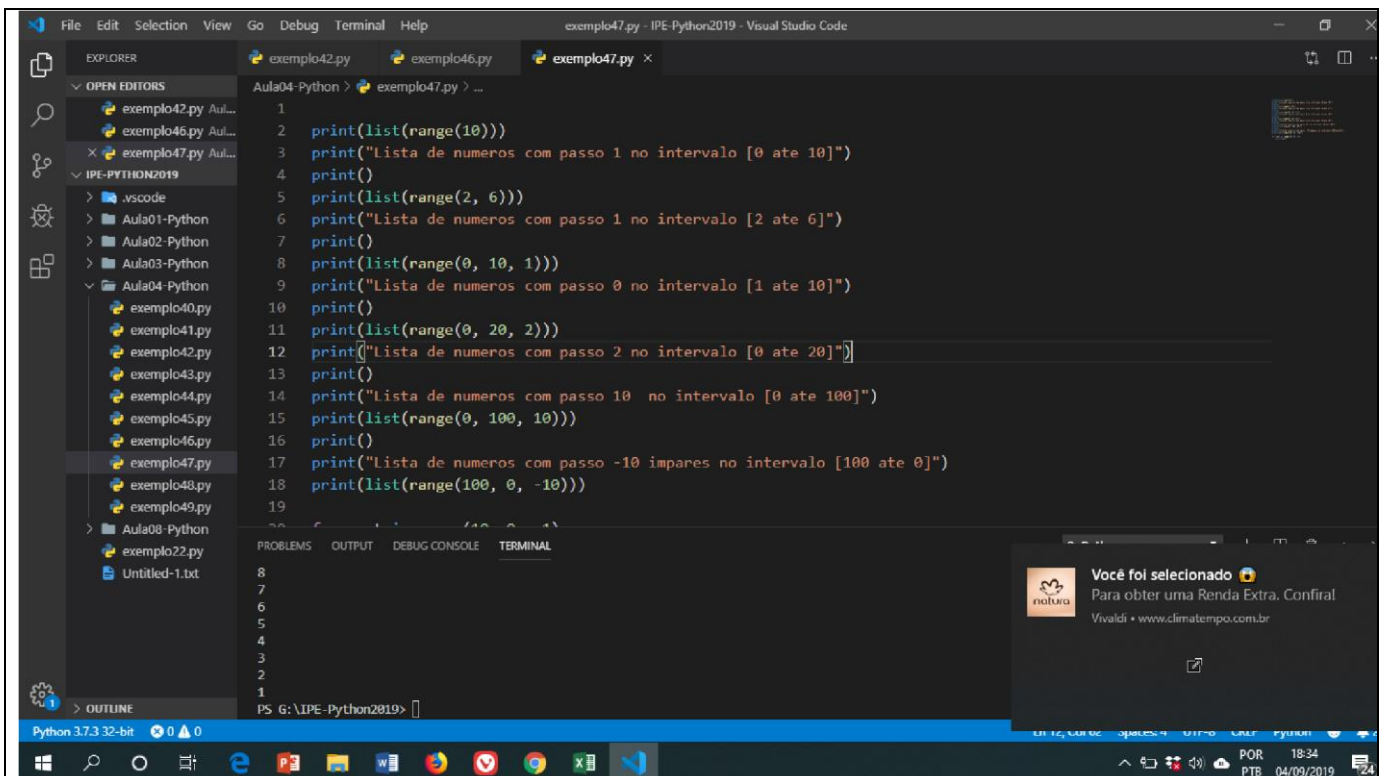
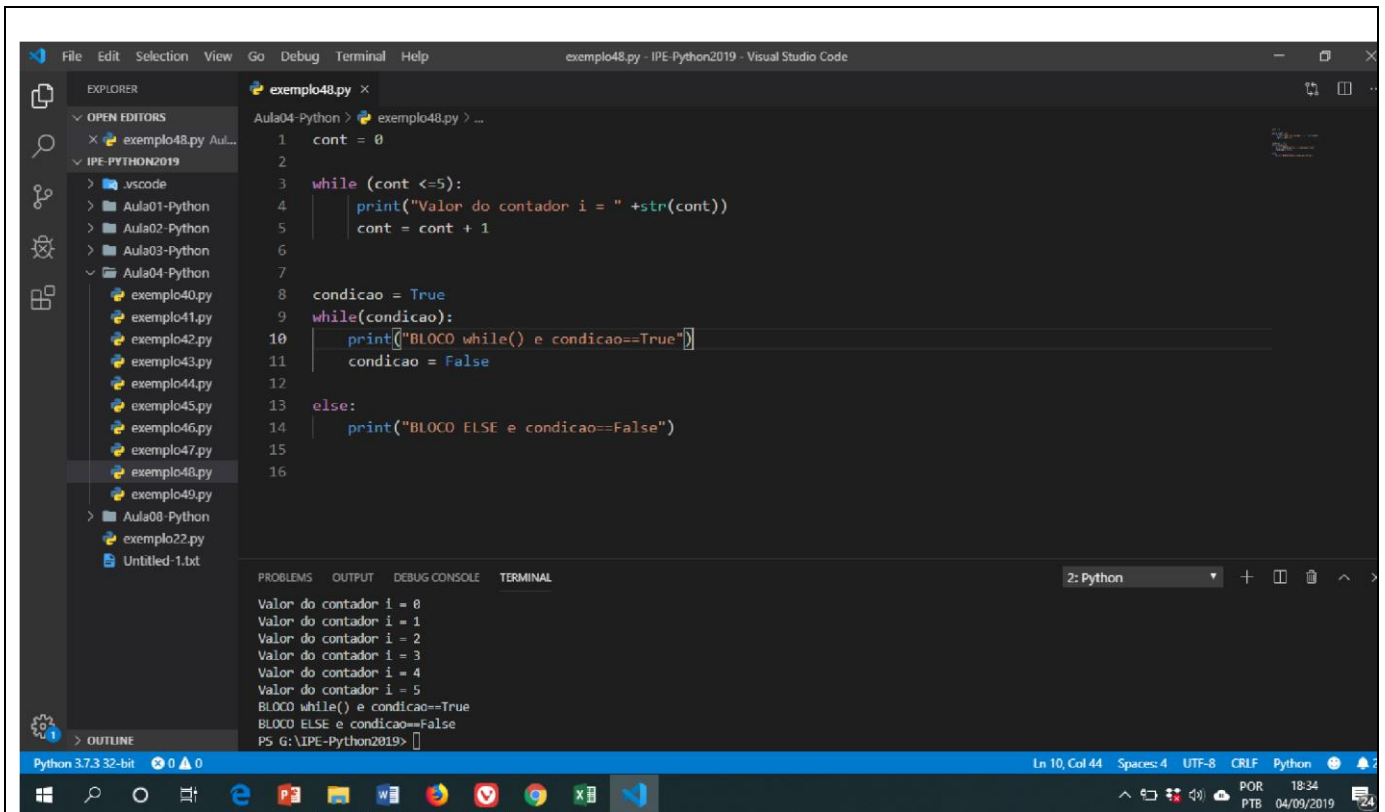


Figura G: exemplo46.py

Exemplo 47

PASTA: ALP-Python-2023/ Aula04-Python
arquivo: exemplo47.py



The screenshot shows the Visual Studio Code editor with the file `exemplo48.py` open. The code defines a variable `cont` and a `while` loop that prints the value of `cont` from 0 to 5. Below the loop, there is an `if` statement that prints a message based on the value of `condicao`. The terminal output shows the execution of the code, displaying the values of `cont` and the messages from the `if` statement.

```
1 cont = 0
2
3 while (cont <=5):
4     print("Valor do contador i = "+str(cont))
5     cont = cont + 1
6
7
8 condicao = True
9 while(condicao):
10    print("BLOCO while() e condicao==True")
11    condicao = False
12
13 else:
14    print("BLOCO ELSE e condicao==False")
15
16
```

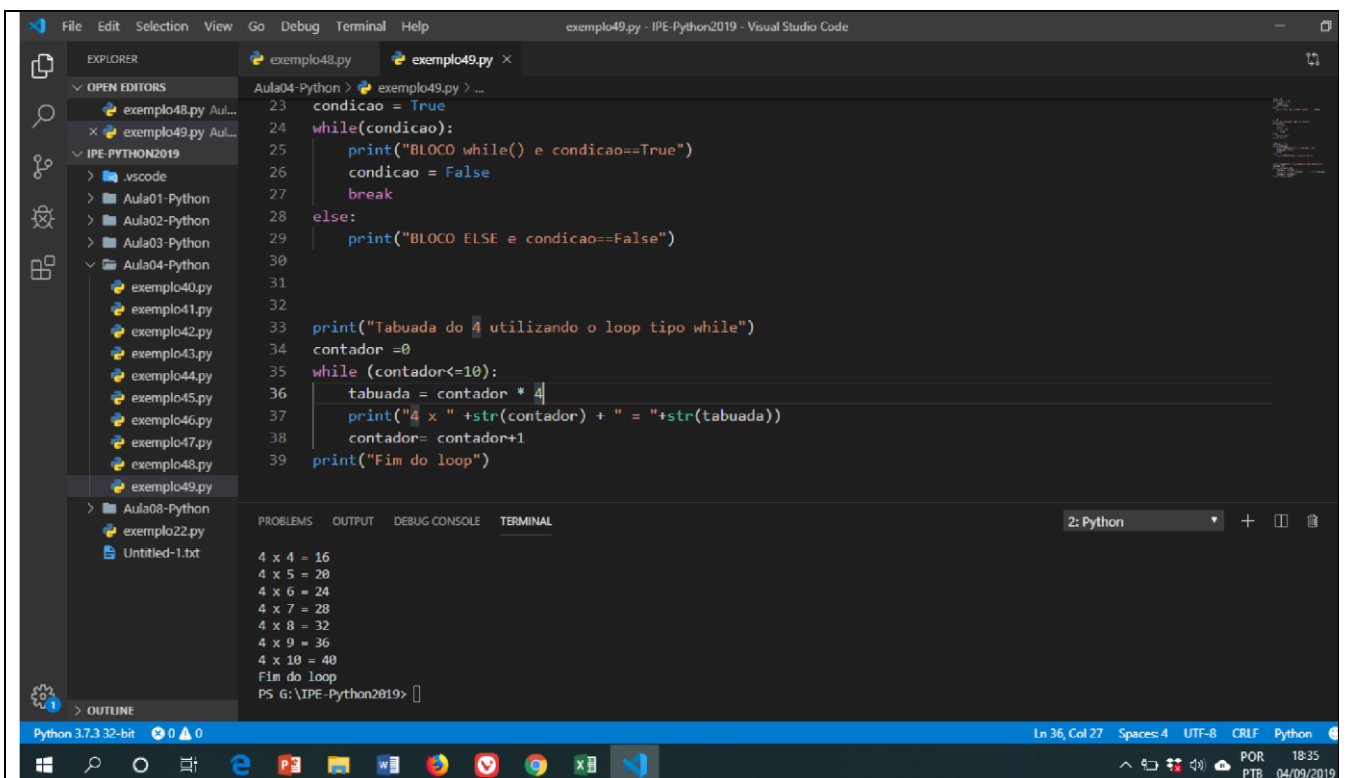
Terminal Output:

```
Valor do contador i = 0
Valor do contador i = 1
Valor do contador i = 2
Valor do contador i = 3
Valor do contador i = 4
Valor do contador i = 5
BLOCO while() e condicao==True
BLOCO ELSE e condicao==False
PS G:\IPE-Python2019>
```

Figura H: exemplo47.py

Exemplo 48

PASTA: ALP-Python-2023/ Aula04-Python
arquivo: exemplo48.py



The screenshot shows the Visual Studio Code editor with the file `exemplo49.py` open. The code defines a variable `condicao` and a `while` loop that prints the value of `condicao` and then sets it to `False` and breaks the loop. Below the loop, there is a `print` statement and a `while` loop that calculates the multiplication table for the number 4. The terminal output shows the execution of the code, displaying the value of `condicao` and the multiplication table.

```
23 condicao = True
24 while(condicao):
25     print("BLOCO while() e condicao==True")
26     condicao = False
27     break
28
29 else:
30     print("BLOCO ELSE e condicao==False")
31
32
33 print("Tabuada do 4 utilizando o loop tipo while")
34 contador =0
35 while (contador<=10):
36     tabuada = contador * 4
37     print("4 x " +str(contador) + " = "+str(tabuada))
38     contador= contador+1
39 print("Fim do loop")
```

Terminal Output:

```
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
4 x 10 = 40
Fim do loop
PS G:\IPE-Python2019>
```

Figura i: exemplo48.py