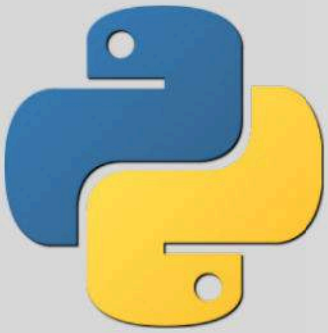
Python Fundamentals



Operadores e Estrutura de Condição

3.1

Anotações

Operadores e Estrutura de condição

Objetivos da Aula

- ✔ Conhecer os operadores.
- ✔ Conhecer estrutura de condição.

Anotações

Operadores e Estrutura de condição

Tipos de Operadores

No Python temos três tipos de operadores:

Aritméticos: utilizados para realizar algum tipo de cálculo.

Relacionais: empregados para comparações, retornando true ou false em seus resultados.

Lógicos: usados para testes condicionais.

Anotações

Operadores e Estrutura de condição

Operadores aritméticos

```
#!/usr/bin/python3
num1 = 5;
num2 = 4;
mult = num1 * num2; # Para a Multiplicação(*)
adic = num1 + num2; # Para a Adição(+)
subt = num1 - num2; # Para a Subtração(-)
divi = num1 / num2; # Para a Divisão(/)
modu = num1 % num2; # Para o Módulo(%)
```

Anotações

Operadores e Estrutura de condição

Operadores aritméticos — Forma abreviada

```
#!/usr/bin/python3
number = 1;  # A variável number recebe 1
number += 2; # Somamos 2 a variável
number -= 2; # Subtraimos 2 a variável
number *= 2; # Multiplicamos a 2 a variável
number /= 2; # Dividimos por 2 a variável
number %= 2; # Resto da divisão por 2
```

Anotações

Operadores e Estrutura de condição

Operadores relacionais

==	=	Igual
!=	=	Não igual ou Diferente
<	=	Menor
<=	=	Menor ou igual
>	=	Maior
>=	=	Maior ou igual
is	=	Compara Booleano

Anotações

Operadores e Estrutura de condição

Os operadores lógicos são:

```
var1 and var2:   # retorna True se var1 e var2
                   forem TRUE;
var1 or var2:    # retorna True se num1 ou num2
                   forem TRUE;
not var:         # retorna TRUE se var for False;
num1 == num2:    # retorna True se num1 e num2
                   forem iguais;
num1 != num2:    # True se num1 for diferente de
                   num2 forem TRUE;
```

Operadores

Constituem meios para manipular dois valores (operandos), como por exemplo: 5 + 10 = 15.
Neste caso, 5 e 10 são chamados de operandos, o sinal + é chamado de operador.
Os operadores nos ajudam principalmente quando trabalhamos com estruturas de decisão.

Anotações

Operadores e Estrutura de condição

Estruturas de Decisão

Estruturas de decisão, permitem manipular o fluxo de execução de código em uma aplicação, tendo como base um teste lógico, que espera um valor verdadeiro ou falso.

```
se condição faça
   comandos;
caso contrário faça
   comandos;
```

```
if condição:
   print("verdadeiro")
else:
   print("falso")
```

Estruturas de decisão

As estruturas de controle, provarão uma determinada condição, caso esta seja verdadeira, executará o bloco do "if", se for falsa executará o bloco do "else".
Os comandos a serem executados, caso a condição seja verdadeira ou falsa, devem estar identados "dentro da condição".

Maiores informações sobre o if/else podem ser consultadas em:
https://docs.python.org/3/reference/compound_stmts.html#if

Anotações

Operadores e Estrutura de condição

Estruturas de Decisão

A estrutura de decisão **IF** testará uma condição, caso seja verdadeira, o que estiver dentro do seu bloco de código será executado. Caso seja Falso, o bloco **ELSE** é executado.

```
#!/usr/bin/python3
idade = 18
if idade == 18:
   print('Você é maior de idade')
else:
   print('Você é menor de idade')
```

Anotações

Operadores e Estrutura de condição

Utilizando operadores

Podemos atender mais de um requisito em uma estrutura de decisão utilizando operadores **and** e **or**.

```
#!/usr/bin/python3
idade = 18
habilitacao = True
if idade >= 18 and habilitacao == True:
   print('Você pode dirigir')
else:
   print('Você não pode dirigir')
```

Estruturas de decisão com operadores

Podemos criar outras estruturas de decisão combinando mais de um tipo. Por exemplo: pense, para dirigir você precisa ter no mínimo 18 anos precisa ter carteira de habilitação ou, estar acompanhado dos pais. O código para esta situação é semelhante ao que segue:

```
#!/usr/bin/python3
 idade = 22
 carteira = False
 pais = True
 if idade >= 18 and carteira == True or pais == True:
    print('Você pode dirigir')
 else:
    print('Você não pode dirigir')
```

Operadores e Estrutura de condição

Encadeamento de condição

Em programação, existe o encadeamento de condições, ou seja, podemos ter diversos **ifs** dentro de uma estrutura.

```
se(condição) faça
   comandos;
Senão
se(condiçao)
   comandos;
senão
   comandos
fim
```

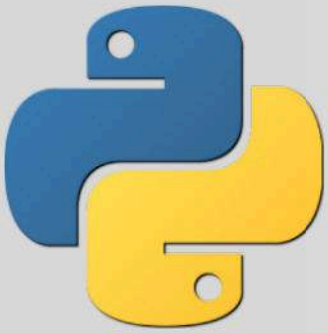
```
if condição:
   print('verdadeiro')
elif condicao:
   print('falso')
else:
   print('Faça isso')
```

Encadeamento de condição

Usamos elif para verificar mais de uma condição, precisamos deste comando quando é necessário entrar somente em um dos ifs encadeados. Caso precise mais de um, é preciso fazer um if dentro do bloco de código do outro.

```
#!/usr/bin/python3
  idade = 17
  carteira = False
  pais = True
  if idade >= 18 and carteira == True:
       print ('Você pode dirigir')
  elif idade == 17 and pais == True:
       print ('Você pode dirigir em emergencias')
  else:
       print ('Você não pode dirigir')
```

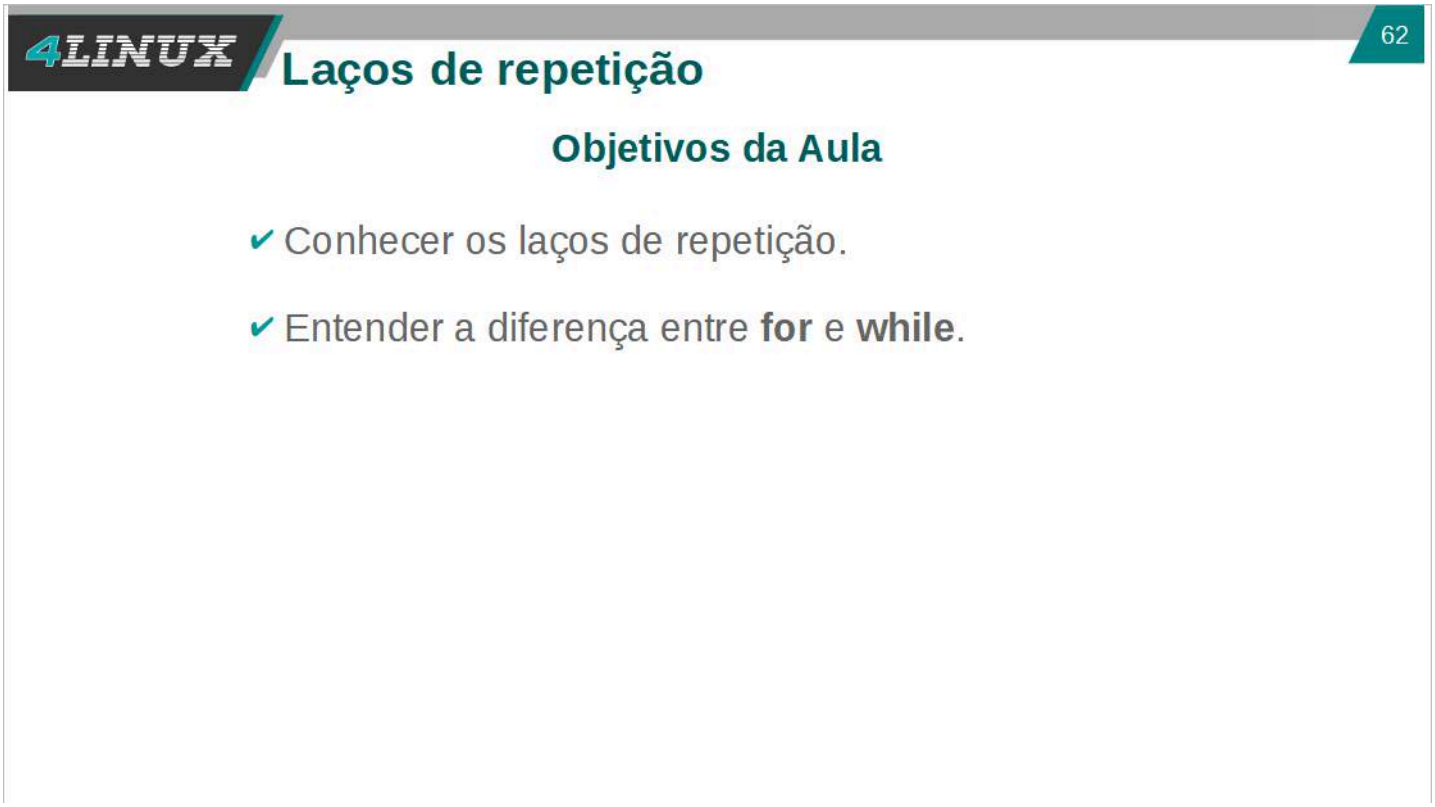
Python Fundamentals



Laços de Repetição For e While

3.2

Anotações



Conhecer os laços de repetição for e while

Nesta aula apresentaremos os laços de repetição for e while, entenderemos as diferenças entre eles e criaremos alguns exemplos práticos.

Anotações

Laços de repetição

while

While permite a execução de um bloco de código, desde que a expressão que foi passada como parâmetro seja verdadeira.

```
enquanto(condição) faça
   comandos;
fimenquanto
```

```
while condiçao:
   comandos
```

Atenção: é muito comum não atentarmos acerca de como o laço irá terminar. Para que termine, precisamos fazer com que a condição se torne false em algum momento.

Laço de repetição while

Nos laços de repetição while, devemos evitar a situação de loops infinitos, estes ocorrem quando em algum momento uma condição torna-se falsa. Em alguns casos, estes loops fazem parte da lógica do nosso script, aprenderemos a utilização de outras formas de condições que interrompem o loop.

Anotações

Laço de repetição

Utilizando o while

Podemos utilizar o while da seguinte forma:

```
#!/usr/bin/python3
x = 1
while x < 10:
   print("Número: {}".format(x))
   x += 1
print("O while acabou !")
```

Operadores

Ainda conseguimos definir o while e passar um True para que execute, porém. não é o mais indicado. Exemplo:

```
#!/usr/bin/python3
  X = 1
  while True:
      print ("Número: {}!".format(x))
      x += 1
```

Em Python não existe o laço de repetição, DO/WHILE, como em outras linguagens, na PEP 0315 o autor explica o porquê da não implementação: https://www.python.org/dev/peps/pep-0315/

Anotações

Laço de repetição

for

A instrução **for** é perfeita ao trabalhar-se com listas no python, pode ser utilizada de maneira tradicional, partindo de um valor inicial para outro valor final.

```
para(valor/condicao)
   comandos;
fimpara
```

```
for item in lista:
   print(item)
```

Atenção: for é utilizado para executar um montante fixo. Por exemplo: percorrer uma lista ou executar até certa quantidade de vezes.

Anotações

Laço de repetição

Utilizando o for

Podemos utilizar **for** da seguinte maneira:

```
#!/usr/bin/python3
fruta = ["Laranja", "Melancia", "Uva" ]

for f in fruta:
   print(f)
```

Anotações

Laço de repetição

Com **for** podemos utilizar uma função chamada **"range"**, retornará uma sequencia:

```
#!/usr/bin/python3
for i in range(10, 0, -1):
        print (i)
```

Sua sintaxe será a seguinte:

```
for i in range(inicio, fim, incremento)
```

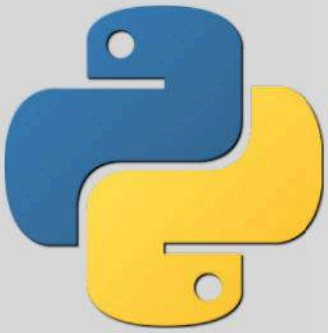
Operadores

A função range() consiste uma maneira, muito mais fácil, para gerar uma sequência de números sem utilizar o while.
Assim como no while, os valores são gerados um por um.
Ainda com o for, podemos utilizar a função enumerate(), para trazer as informações como um indice:

```
#!/usr/bin/python3
  fruta = ["Laranja", "Abacaxi", "Uva" ]
  for num,item in enumerate(fruta):
       print ("{} esta na posicao {}".format(item,num))
```

Anotações

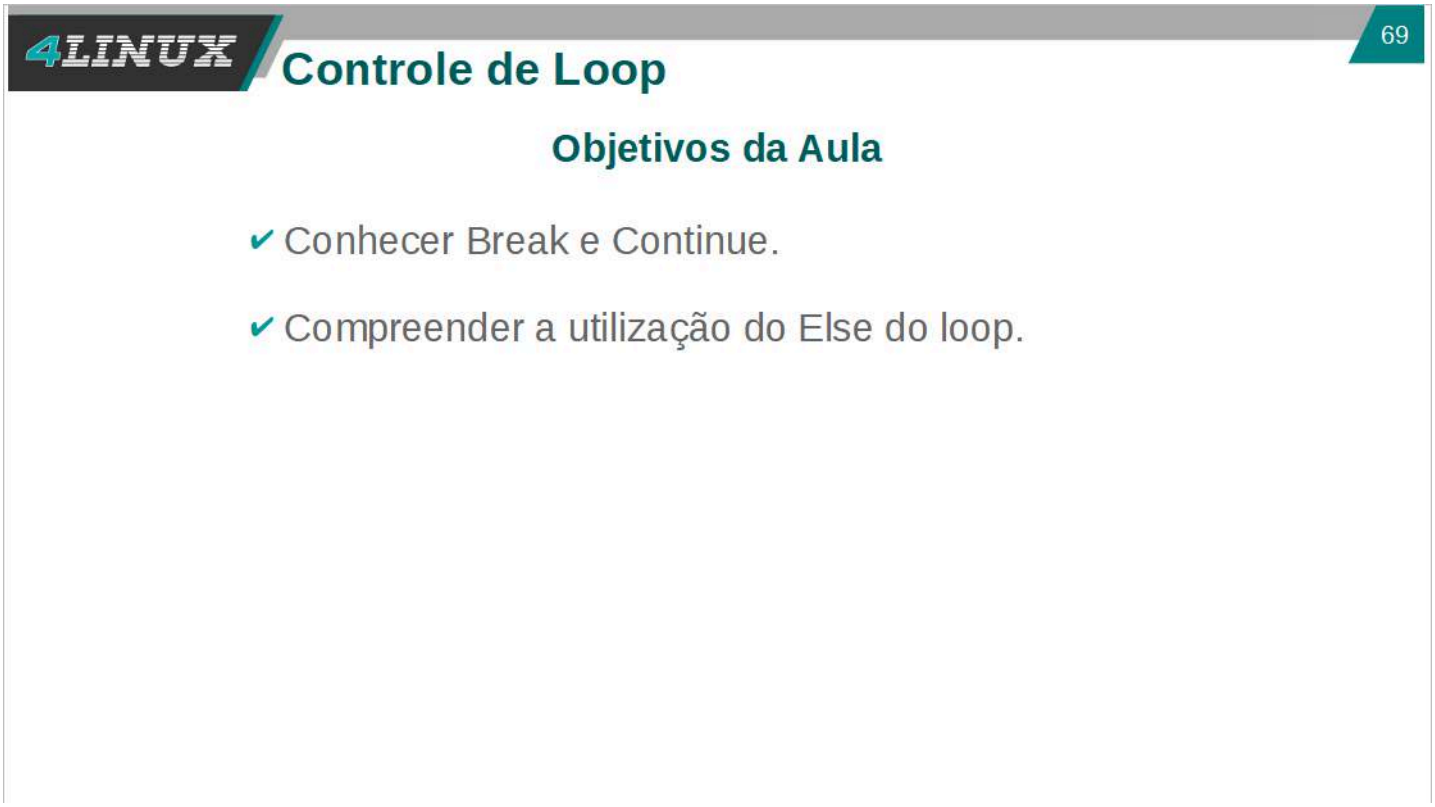
Python Fundamentals



Controle de Loop

3.3

Anotações



Controle de Loop

Nesta aula conheceremos o controle de loop com break e continue, compreendendo também o else do loop.

Anotações

Controle de Loop

Break e Continue

✔ Você pode enfrentar situação na qual precise sair de um loop quando uma condição externa é acionada.

✔ Também pode ocorrer circunstância, em que deseja ignorar uma parte do loop e, iniciar a próxima execução.

✔ Python fornece as instruções break e continue para lidar com tais situações, mantendo bom controle em seu loop.

Anotações

Controle de Loop

Break

- ✔ No Python esta instrução encerra o loop atual e, retoma a execução na próxima instrução, semelhante a quebra tradicional encontrada em C.
- ✔ O uso mais comum de quebra, ocorre quando alguma condição externa é acionada, exigindo a saída rápida de um loop.
- ✔ A instrução break pode ser usada em loops while e for.

Interrompendo um loop

In [1]: Definimos uma variável que será utilizada como contador, para o bloco de repetição.
In [2]: Definimos um bloco de repetição, com uma condição de contator menor que 10 e, o contator com incremento de + 1 por execução do loop, exibindo uma mensagem para visualizar quantidade de execuções.
Definimos um bloco de condição no interior do bloco de repetição, assim, quando o valor do contador for igual a 2, o loop será interrompido, deste modo, o loop será executado 3 vezes ao invés de 10, primeira execução para contator igual a 0 , em seguida 1, 2 e interrompe.

Anotações

Controle de Loop

Break

```
In [1]: cont = 0

In [2]: while cont < 10:
   ...:     print('vezes de execução {}'.format(cont + 1))
   ...:     if cont == 2:
   ...:         print('bloco de condição que interrompe o loop')
   ...:         break
   ...:     cont += 1
   ...:
vezes de execução 1
vezes de execução 2
vezes de execução 3
bloco de condição que interrompe o loop
```

Search

Interrompendo um loop

In [1]: Definimos uma variável que será utilizada como contador, para o bloco de repetição.
In [2]: Definimos um bloco de repetição, com uma condição de contator menor que 10 e, o contator com incremento de + 1 por execução do loop, exibindo uma mensagem para visualizar quantidade de execuções.
Definimos um bloco de condição no interior do bloco de repetição, assim, quando o valor do contador for igual a 2, o loop será interrompido, deste modo, o loop será executado 3 vezes ao invés de 10, primeira execução para contator igual a 0 , em seguida 1, 2 e interrompe.

Anotações

Controle de Loop

Continue

- ✔ No Python, continue, retorna o controle para o início do loop while.
- ✔ Esta instrução, rejeita todas as demais restantes na iteração atual do loop e, move o controle de volta para o início do loop.
- ✔ Continue pode ser usada em loops while e for.

Ignorando instruções do loop

In [1]: Atribuímos uma lista vazia na variável.
In [2]: Definimos um loop numérico de 0 a 20, colocamos uma estrutura de condição na qual, o resto da divisão do número por 2, igual a zero, retorna ao início do loop, ignorando a instrução abaixo que insere o número na lista vazia, deste modo, serão inclusos os números impares.
In [3]: Arrolamos a lista com a alteração realizada no loop do passo anterior.

Anotações

Controle de Loop

Continue

```
In [1]: impar = []

In [2]: for num in range(20):
   ...:     if num % 2 == 0:
   ...:         continue
   ...:     impar.append(num)
   ...:

In [3]: impar
Out[3]: [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

Ignorando instruções do loop

In [1]: Atribuímos uma lista vazia na variável.
In [2]: Definimos um loop numérico de 0 a 20, colocamos uma estrutura de condição na qual, o resto da divisão do número por 2, igual a zero, retorna ao início do loop, ignorando a instrução abaixo que insere o número na lista vazia, deste modo, serão inclusos os números impares.
In [3]: Arrolamos a lista com a alteração realizada no loop do passo anterior.

Anotações

Controle de Loop

Else do loop

- ✔ Python suporta uso de uma instrução else associada a uma instrução de loop.
- ✔ Se for usada com um loop for, else será executada quando o loop tiver esgotado a iteração da lista.
- ✔ Se for usada com um loop while, else será executada quando a condição se tornar-se falsa.

Anotações

Controle de Loop

```
In [1]: nomes = ['maria', 'joao', 'pedro']

In [2]: busca = input('qual nome deseja procurar? ')
qual nome deseja procurar? maria

In [3]: for nome in nomes:
   ...:     if busca  == nome:
   ...:         print('Está na lista')
   ...:         break
   ...: else:
   ...:     print('Não está na Lista')
   ...:
Está na lista
```

Else do loop

In [1]: Definimos uma lista de nomes (strings).
In[2]: Interagimos uma entrada com a função input, atribuindo a entrada em uma variável.
In[3]: Percorremos a lista de nomes, efetuando uma condição, se a busca for igual ao nome, exibir uma mensagem que encontrou, caso o loop não seja interrompido, executar o bloco ELSE com a mensagem não encontrou.

Anotações