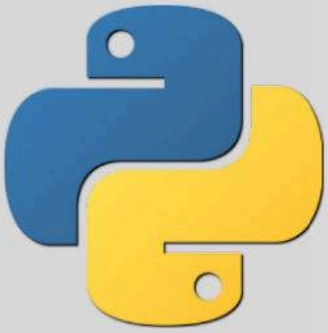
Python Fundamentals



Escopo e Sintaxe

5.1

Anotações

Escopo e sintaxe

Objetivos da Aula

- ✔ Entender funções.
- ✔ Definir funções.
- ✔ Entender escopo de uma função.
- ✔ Definir funções de escopo local e global.

Escopo global e local

Nesta aula apresentaremos o conceito de funções, sua sintaxe, escopo e como trabalhar com variáveis globais e locais.

Anotações

Escopo e sintaxe

Entendendo funções

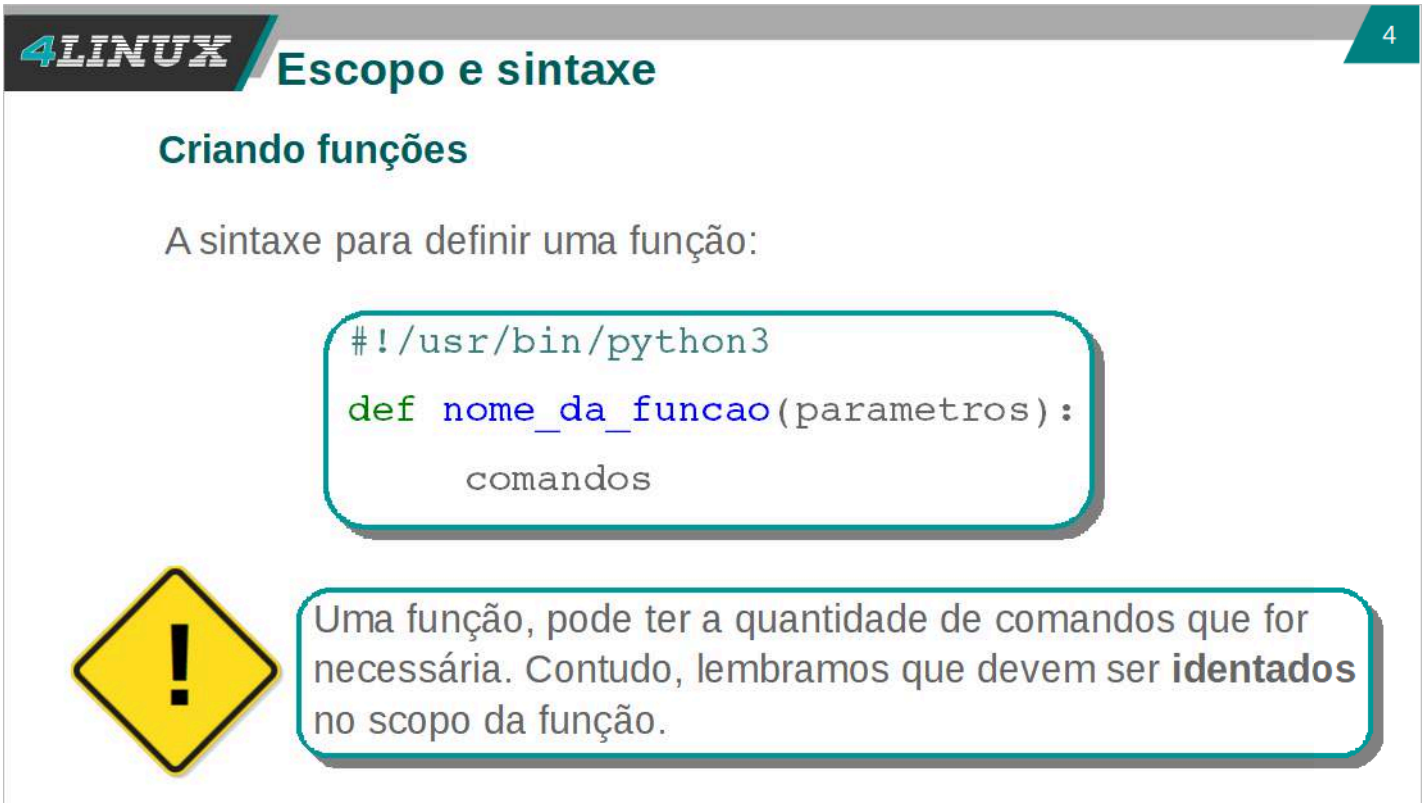
- ✔ Função é uma sequência de instruções que realiza uma operação.
- ✔ Podemos definir nossas próprias funções para resolver um problema em nossa aplicação.
- ✔ É possível utilizar qualquer nome para a nossa função.
- ✔ Porém, PEP8 recomentado que os nomes sejam sempre escritos em minúsculo.
- ✔ Em caso de caso de necessidade, nome separado por underline _ , facilitando a legibilidade.

Entendendo funções

Podemos utilizar as funções para reaproveitamento de código, imagine ter uma aplicação que executará um determinado trecho de código. Há duas opções: reescrever diversas vezes ou criar uma estrutura que execute esta fração de código quando necessário.
Criar funções nos ajuda a deixar o código menor, mais organizado e até mais legível.

Para saber mais sobre funções, pesquise a documentação oficial:
https://docs.python.org/3/tutorial/controlflow.html#defining-functions

Anotações



Criando funções

Ao definir um parâmetro, significa obrigatoriamente que a função espera receber algum dado. Se a função for implementada desta maneira e não passarmos nenhuma informação, ocorrerá erro.
Para definir uma função sem nenhum parâmetro, utilize: def nome_da_funcao()
A seguir um exemplo de programa utilizando funções. A primeira, possui 1 parâmetro obrigatório. A segunda não recebe parâmetros.
Nas funções, **parâmetro** é o valor entre parênteses, o valor esperado. **Argumento** é o valor que passamos como parâmetro para a função. Por exemplo, a variável produto dentro do while.

```
#!/usr/bin/python3
produtos = []

def cadastrarProduto(produto):
     global produtos
     produtos.append(produto)


def listarProdutos():
     global produtos
     for p in produtos:
          print(p)

produto = ""
while produto != "sair":
     produto = input("Digite o produto que deseja cadastrar: ")
     cadastrarProduto(produto)
     print("produtos cadastrados")
     listarProdutos()
```

Escopo e sintaxe

As funções também podem possuir parâmetros opcionais, que podem ser omitidos durante a chamada de função:

```
#/usr/bin/python3
def nome_funcao(parametro=padrao):
    comandos
```

Atenção: para que um parâmetro não seja obrigatório, deve possuir um valor padrão, que será assumido caso o valor não seja passado.

Funções com parâmetro padrão

Quando escrevemos uma aplicação, em geral utilizamos o “pass”. Um exemplo de função com parâmetro opcional: o cálculo de uma compra com cupom de desconto. O valor do desconto será calculado, caso seja informado um cupom válido.

```
#!/usr/bin/python
carrinho = []
produto1 = {"nome":"Tenis","valor":21.70}
produto2 = {"nome":"Meia","valor":10}
produto3 = {"nome":"Camiseta","valor":17.30}
produto4 = {"nome":"Calca","valor":100.00}
carrinho.append(produto1)
carrinho.append(produto2)
carrinho.append(produto3)
carrinho.append(produto4)

def totalCarrinho(carrinho):
    return sum(produto["valor"] for produto in carrinho)

def cupomDesconto(cupom=""):
    if cupom == "xyzgratis":
        return 0.50
    else:
        return 1

print("o total da sua compra e: ",
         (totalCarrinho(carrinho)*cupomDesconto()))
print("utilizando o cupom xyzgratis o valor sera de ",
         (totalCarrinho(carrinho)*cupomDesconto("xyzgratis")))
```

Escopo e sintaxe

Ao definir uma função, é necessário escrever suas instruções. Porém, apenas quando estabelecemos a estrutura da nossa aplicação.
É comum utilizar a declaração **"pass"**, que ignora a função sem causar erros de sintaxe.

```
#!/usr/bin/python3
def nome_da_funcao(parametros):
    pass
```

Código da função

Outro benefício do emprego do "pass", conseguimos dedicação a uma outra parte do código sem esquecer de retornar para implementar a funcionalidade determinada para aquela função. Quando usamos o "pass" o Python interpretará como, "não execute nada". Desta maneira não acontecerá nenhum erro.

Conheça mais sobre o pass na documentação oficial:
https://docs.python.org/3/tutorial/controlflow.html#pass-statements

Anotações

Escopo e sintaxe

Escopo da função

Uma função pode utilizar variáveis de escopo local. Assim podemos determinar a mesma variável em funções diferentes.

```
def sistema():
    nome = 'Linux'
def curso():
    nome = 'Python'
```

Escopo Local

Ao definir variáveis de escopo local, podemos utilizar a mesma variável em outra parte do código, não sendo sobrescrita. Variáveis definidas dentro de uma função, não podem alterar o valor de variáveis estabelecidas fora desta função.
Quando a aplicação está em execução, buscará as variáveis locais, que fazem parte da função e, caso não seja encontrada, pesquisará em variáveis globais.

```
#!/usr/bin/python3

servidor = "192.168.0.2"

def classe_a():
    servidor = '192.168.0.1'
    print(servidor)


def classe_b():
    print(servidor)


if __name__ == '__main__':
    print(servidor)
    classe_a()
    classe_b()
```

Escopo e sintaxe

Escopo global

Funções podem utilizar variáveis de escopo global. Só é possível realizar alterações em uma variável global, ao especificar, na função, o emprego deste tipo de variável.

```
In [1]: var = 10

In [2]: def escopo():
   ...:     global var
   ...:     print(var)
   ...:     var = 5
   ...:     print(var)
   ...:

In [3]: escopo()
10
5

In [4]: var
Out[4]: 5
```

Escopo Global

In [1]: Definimos uma variável global.
In [2]: Estabelecemos uma função que lê uma variável global, exibe seu valor, depois altera seu valor e exibe novamente.
In[3]: Executamos a chamada da função.
In[4]: Exibindo a variável para confirmar os passos anteriores.

Anotações

Escopo e sintaxe

Passando parâmetros para função

Passamos um parâmetro para a função, da seguinte forma:

```
#!/usr/bin/python3
def nome_da_funcao(parametro):
    print(x)
nome_da_funcao('Aqui eu passo o parametro')
```

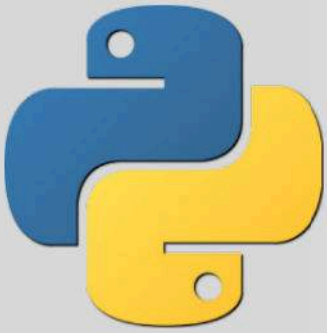
Parâmetros

Ao estabelecer uma função, optamos por receber ou não algum parâmetro, sendo possível passar mais de um parâmetro para a função, veja como:

```
#!/usr/bin/python3

def alterarServidor(atual,novo):
    print("O IP atual é: ",atual)
    print("O novo IP é: ",novo)



alterarServidor("192.168.100.0","10.10.10.1")
```

Anotações

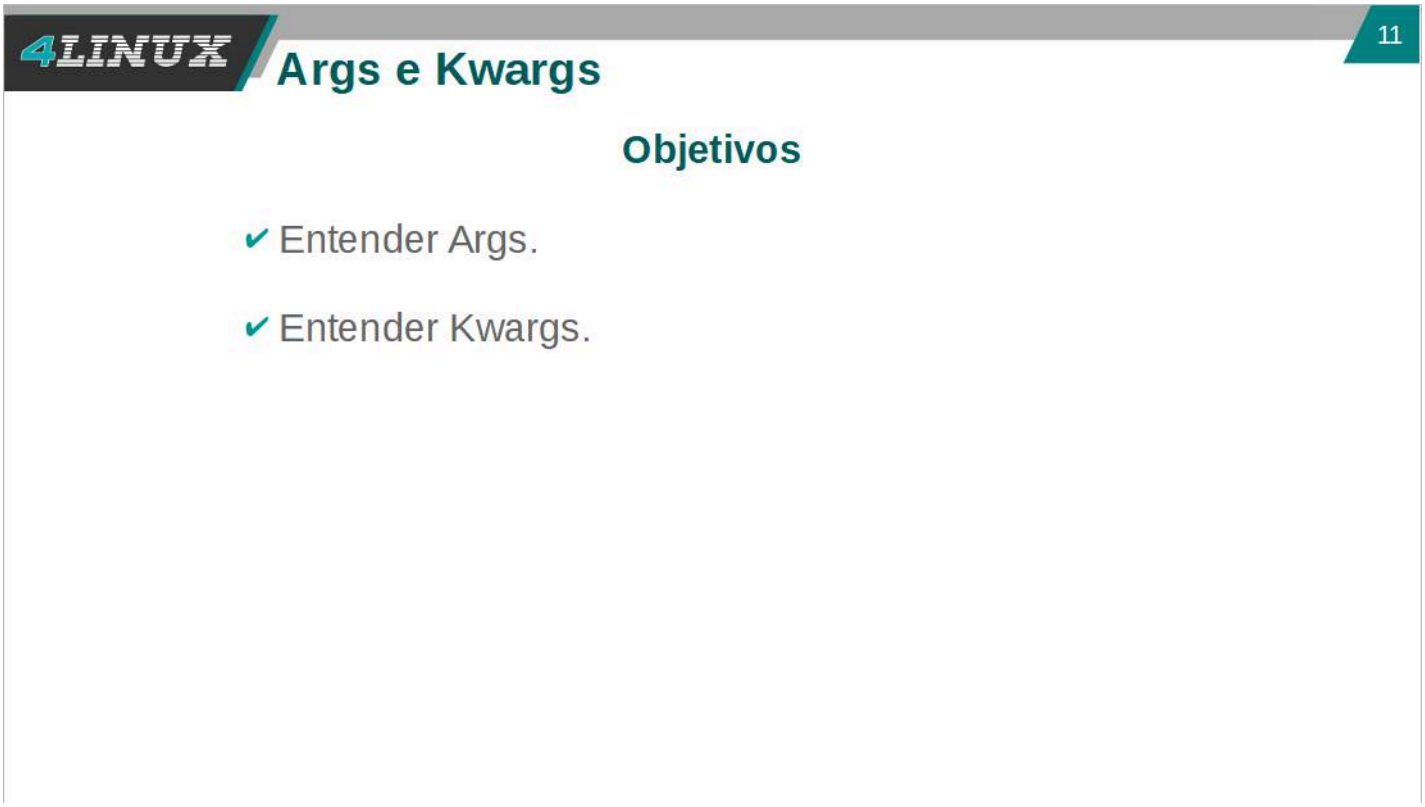
Python Fundamentals



Args e kwargs

5.2

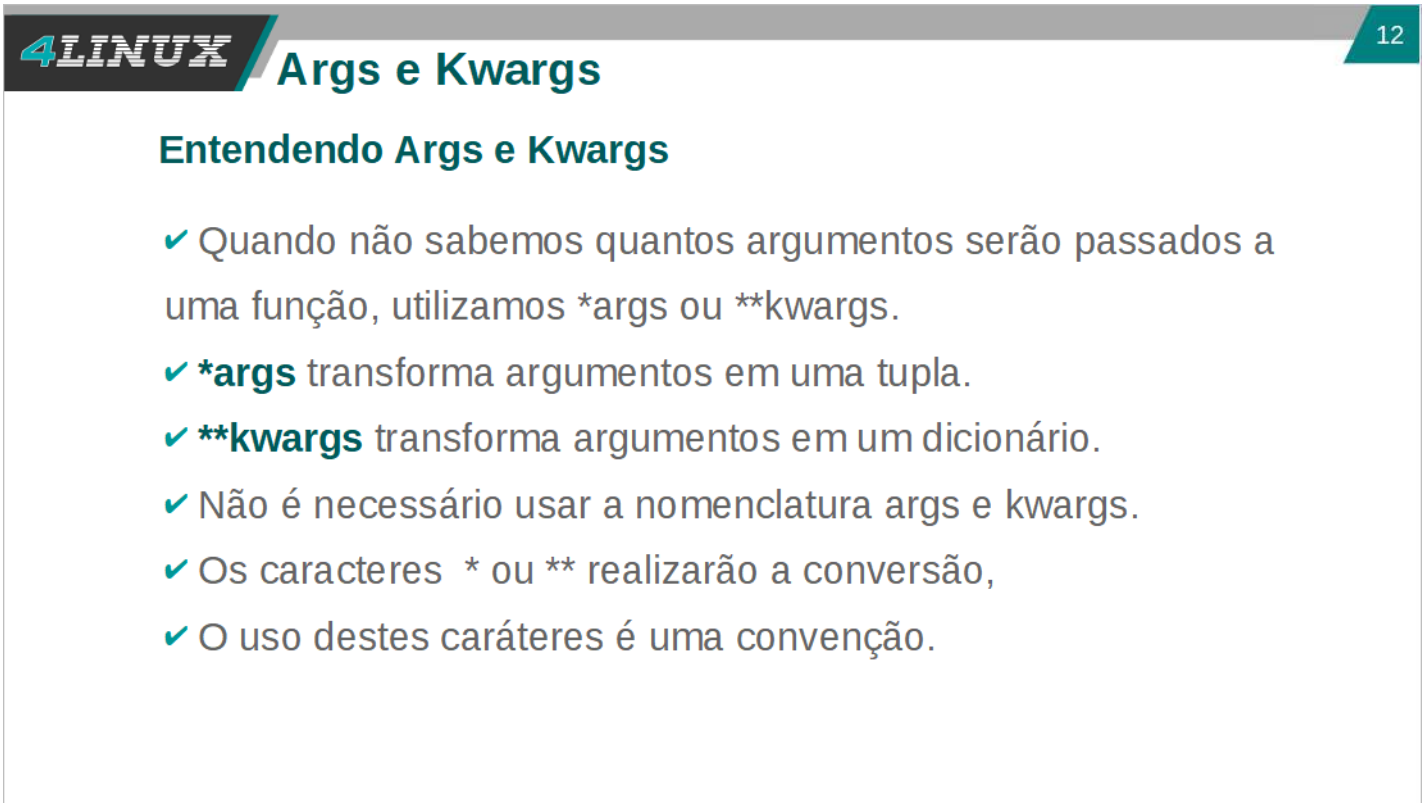
Anotações



Args e Kwargs

Nesta aula apresentaremos Args e Kwargs, seu uso e o entendimento sobre como empregar estes parâmetros em nossas aplicações.

Anotações



Entendendo Args e Kwargs

O uso dos parâmetros *args e **kwargs é comum quando não sabemos a quantidade de argumentos que receberemos. A variação poderá ser de 1 até inúmeros.

Para conhecer mais sobre estes parâmetros, veja a documentação oficial: https://docs.python.org/3/tutorial/controlflow.html#unpacking-argument-lists

Anotações

Args e Kwargs

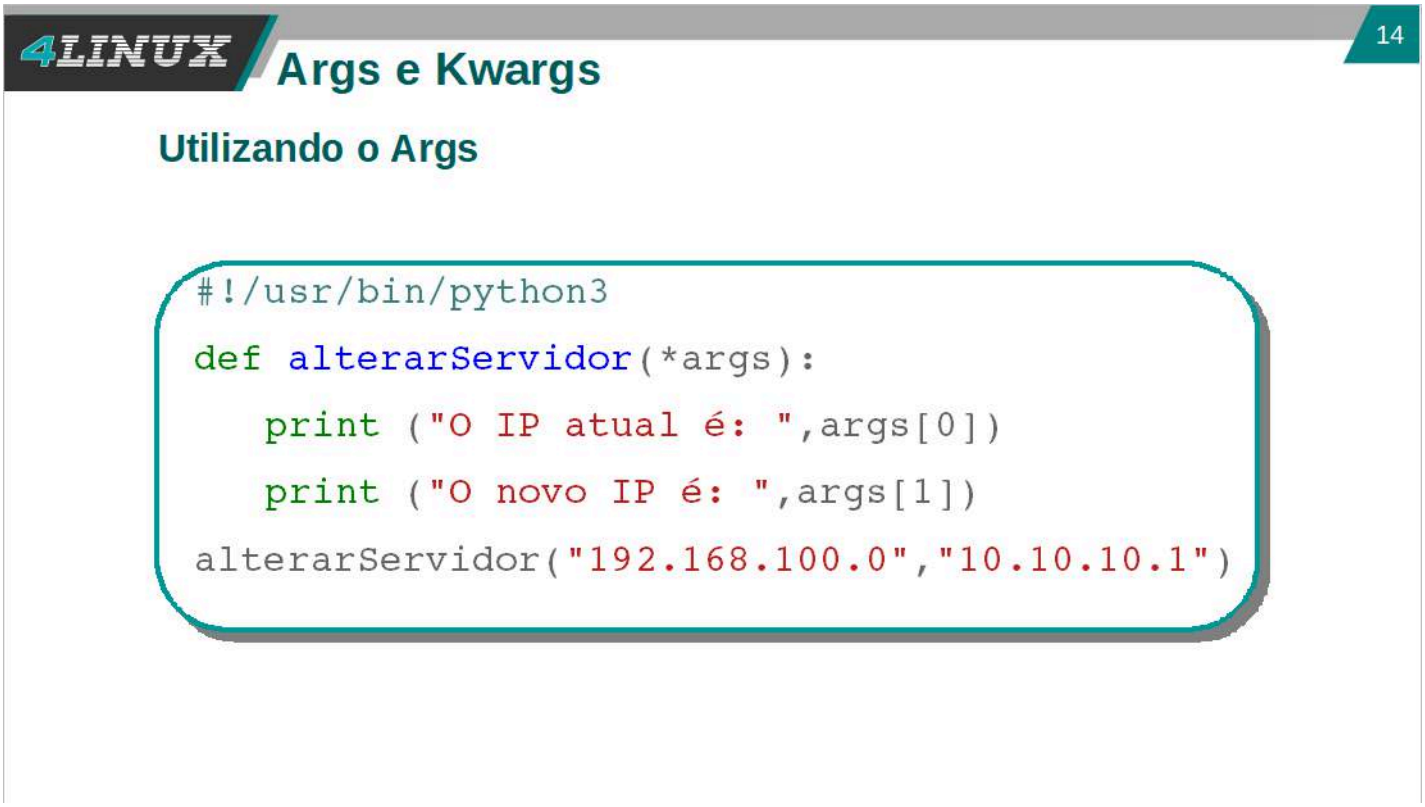
Args

Para criar uma função utilizando args:

```
#!/usr/bin/python3
def nome_da_funcao(*args):
    comandos
```

Desta forma, todos os argumentos passados serão convertidos em uma tupla.

Anotações



Args

Ao empregar Args e receber os argumentos em formato de tupla, devemos trata-la em nossa aplicação para obter o resultado desejado.
Por exemplo: ao chamar uma função, informo os parâmetros: 4linux e python. Utilizo o conceito de tupla, para usar estas informações em minha aplicação da seguinte maneira. Usarei args[0] para exibir 4lilnux e, args[1] para mostrar python, lembre-se, que tuplas são acessadas através de índice.
No exemplo a seguir, apresentamos o calculo da área do quadrado e do retângulo utilizando o parâmetro *args.

```
#!/usr/bin/python3
def calcular(*args):
    if len(args) == 1:
        print ("A area do quadrado e: ",(args[0]*args[0]))
    elif len(args) == 2:
        print ("A area do retangulo e: ",(args[0]*args[1]))
    elif len(args) == 3:
        print ("A area do paralelepipedo e: ",(args[0]*args[1]*args[2]))

calcular(2)
calcular(4,2)
calcular(1,2,3)
```

Args e Kwargs

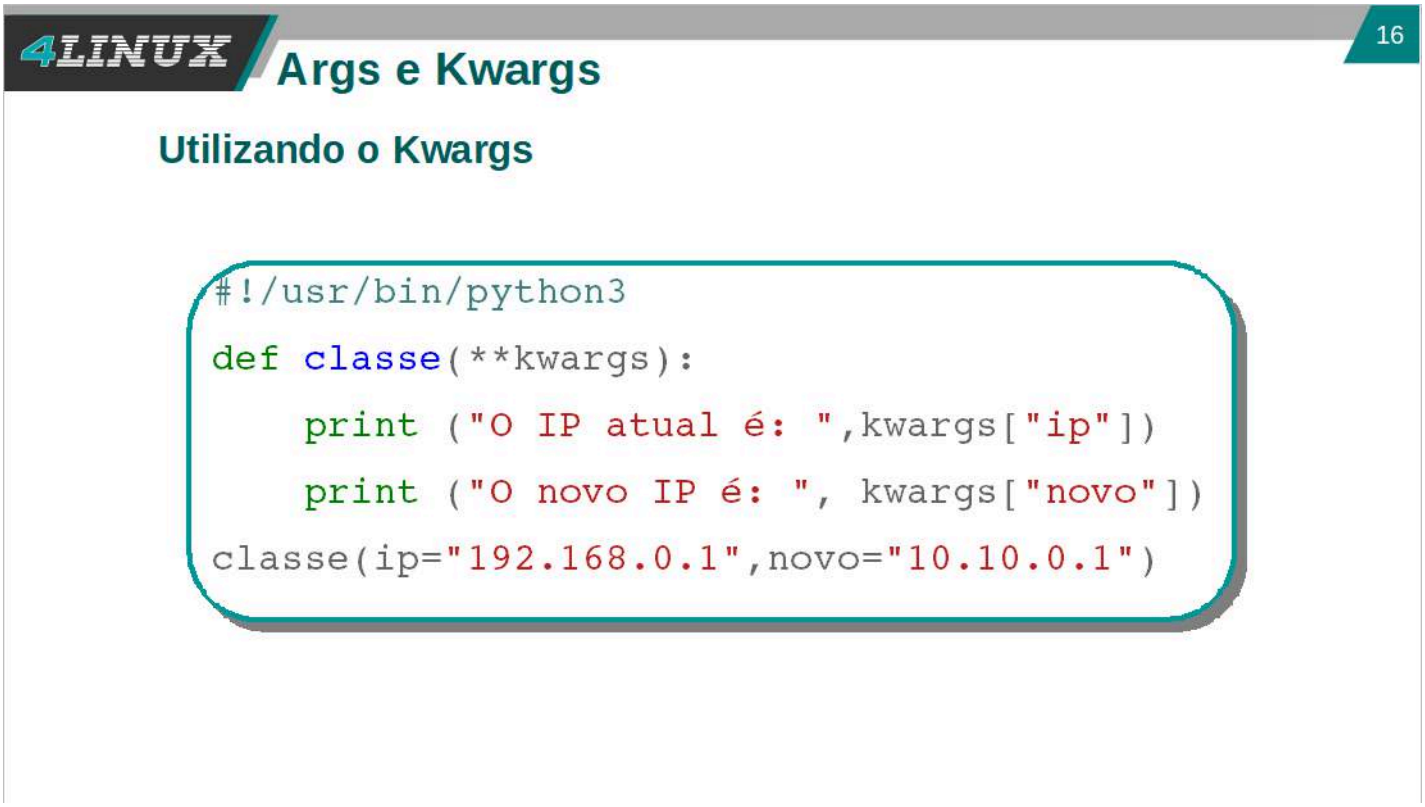
Kwargs

Para criar uma função utilizando Kwargs:

```
#!/usr/bin/python3
def nome_da_funcao(**kwargs):
    comandos
```

Desta forma, todos os argumentos passados serão convertidos em um dicionário.

Anotações

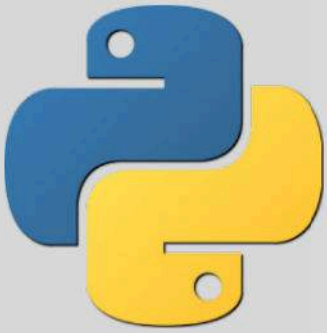


Kwargs

Ao utilizar Kwargs os argumentos serão recebidos em forma de dicionário (chave-valor), para usá-lo na aplicação, devemos acessar a sua chave e assim obter o seu valor.
Por exemplo: ao chamar uma função, passo os seguintes parametros: sistema=linux, linguagem=python. Para utilizar estas informações na aplicação usando o conceito de dicionário, acesso os valores da seguinte forma: kwargs[“linguagem”] e kwargs[“sistema”].
Ainda é possível trabalhar com o dicionário como um todo, para isto, basta usar apenas “kwargs”
No exemplo que segue, veremos todas as chaves e os valores do dicionário informado.

```
#!/usr/bin/python3

def descobreDicionario(**kwargs):
    print (kwargs)
    for k in kwargs.keys():
        print("Chave: {}".format(k))
        print ("Tem o valor: {}".format(kwargs[k]))


descobreDicionario(nome="servidor",
                   ip="192.168.0.1",
                   dominio="4linux.com.br")
```

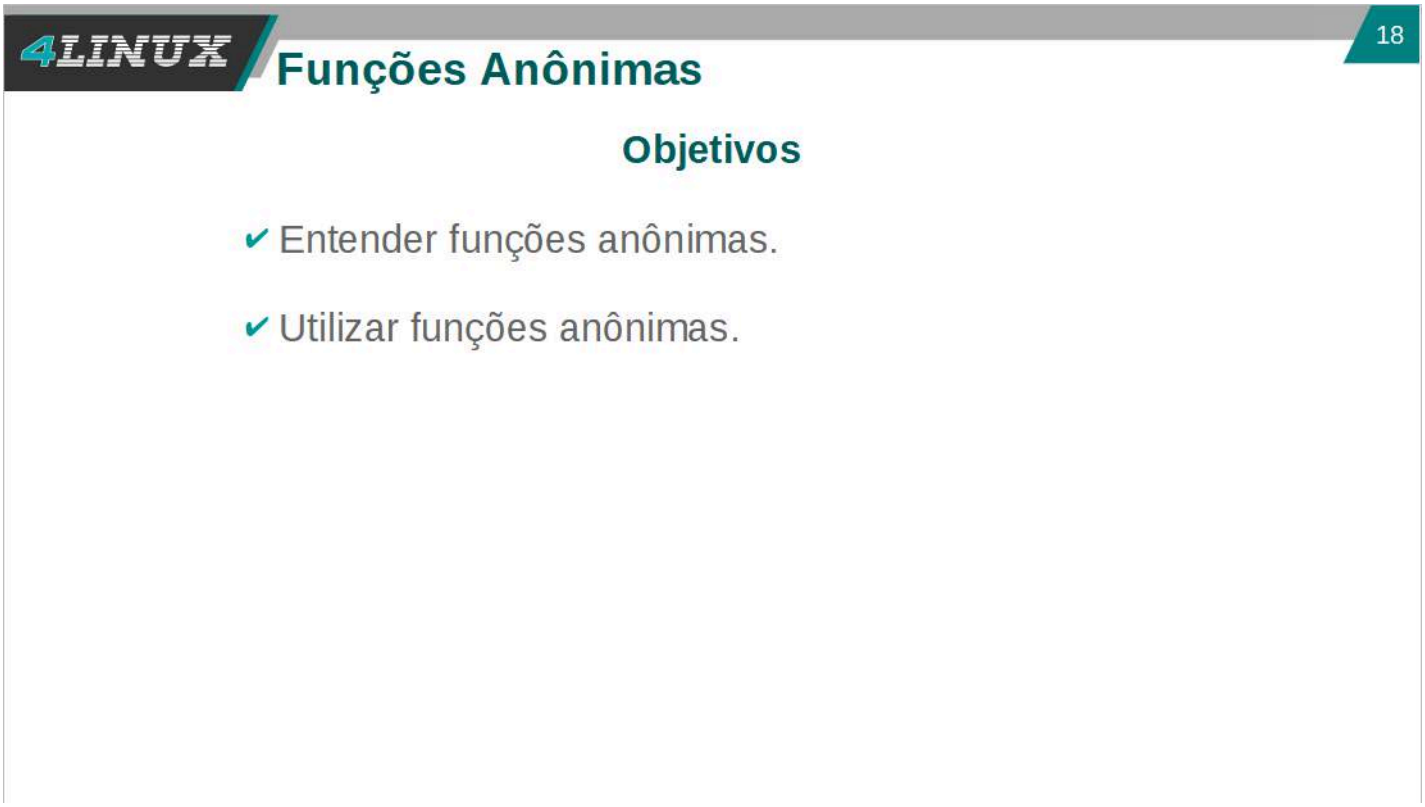
Python Fundamentals



Funções Anônimas

5.3

Anotações



Funções Anônimas

Nesta aula exibiremos as funções anônimas, como funcionam e mostraremos exemplos.

Anotações

Funções Anônimas

Entendendo Funções Anônimas

- ✔ Funções anônimas não estão vinculadas a um nome.
- ✔ Em Python podem ser chamadas como “expressões lambda”.
- ✔ Muito utilizadas no meio acadêmico na resolução de cálculos matemáticos.
- ✔ Funções lambda podem ser definidas dentro de uma função, sendo normalmente atribuídas a uma variável da função principal.

Entendendo Funções Anônimas

Em Python usamos a palavra reservada “lambda” para definir funções anônimas, utilizando esta palavra o retorno da expressão será um objeto de função, por este motivo, são chamadas de funções anônimas.
Podem ser usadas para cálculos matemáticos ou, quando precisamos de uma função que não seja tão complexa, possibilitando execução em uma única linha.

Anotações

Funções Anônimas

Sintaxe

Ao definir uma função anônima, utilizamos a sintaxe:

```
lambda argumentos: expressão
```

Importante lembrar que “lambda” é uma palavra reservada em Python, para evitar erro, não devemos definir nenhuma variável com este nome.

Anotações

Funções Anônimas

Utilizando Funções Anônimas

```
#!/usr/bin/python3

var = lambda x: x*2

print (var(2))
```

No exemplo, a função espera receber um argumento, (valor de x), para depois utilizar este número na multiplicação.

Utilizando Funções Anônimas

Uma função lambda é útil para resolver cálculos matemáticos. Um exemplo do mundo real, quando acontece a Black Friday um produto terá 50% de desconto, o código para essa situação pode ser

```
#!/usr/bin/python3

carrinho = [{"nome":"Tenis","valor":21.70},
            {"nome":"Camiseta","valor":10.33}]

black_friday = lambda x: x / 2

for c in carrinho:
    print ("Nome do produto: ",c["nome"])
    print ("Valor original: ",c["valor"])
    print ("Valor com desconto: ",black_friday(c["valor"]))
    print ("=-=-=-=-=-=-=-=-=-=-=")
```

Funções Anônimas

Passar mais de um número

Podemos passar mais de um número para fazer cálculos:

```
#!/usr/bin/python3

lamb = lambda a,b,c: ((b ** 2) - (4 * a * c))

print (lamb(3,-2,-5))
```

Anotações

Funções Anônimas

Criando mais de uma função

Veja como fazer:

```
#!/usr/bin/python3
anonimas = [lambda x: x** 2,
            lambda x: x** 3,
            lambda x: x** 4]
for a in anonimas:
    print(a(10))
```

Anotações