
IN
INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

IN AULA 05 | Python

PY - Funções II

01 Funções

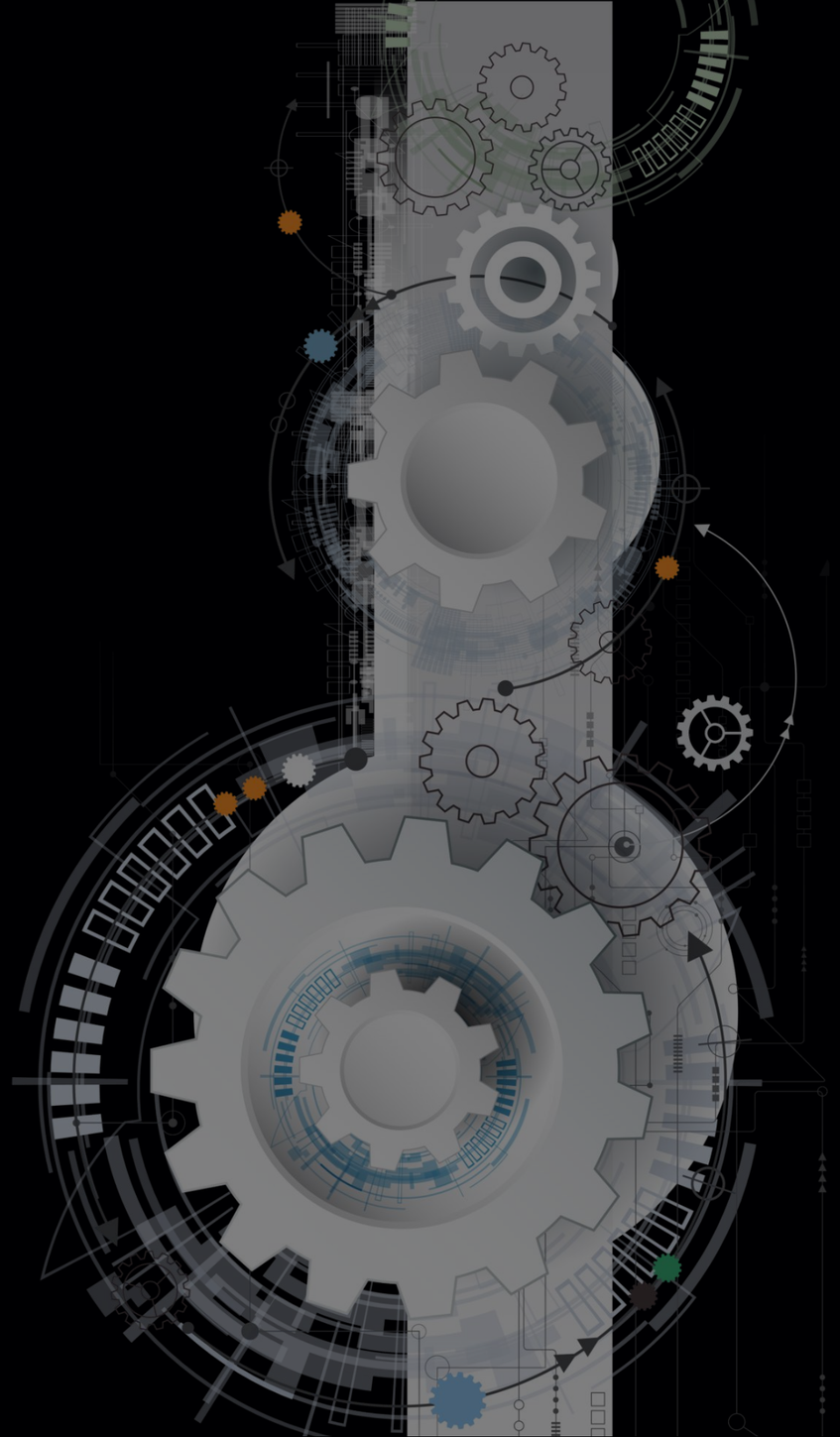
- Revisão
- parâmetros *args e **kwargs
- expressão lambda
- funções agregadoras

IN AULA 05 | Python

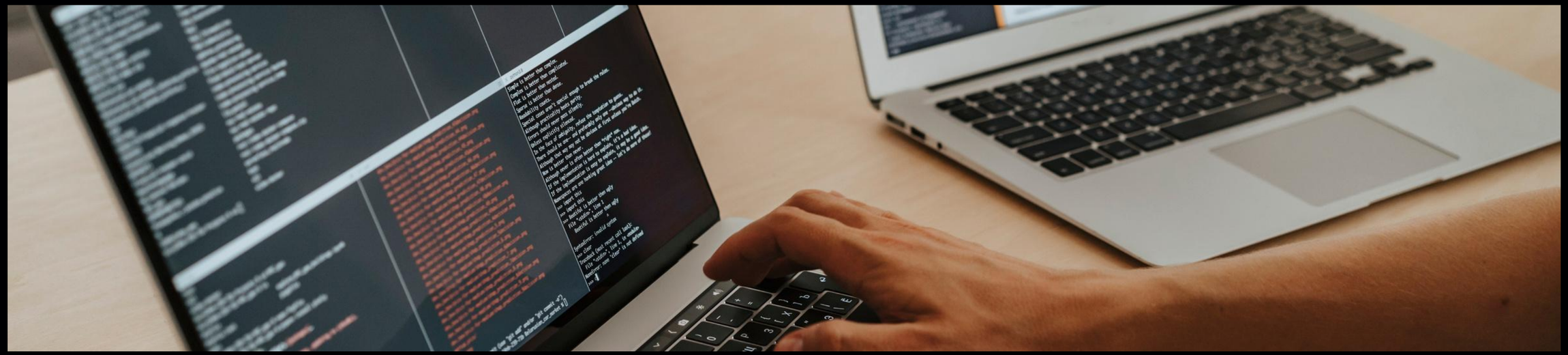
PY - Funções II

Revisão

Uma função é um bloco de código que pode ser chamado repetidamente em um programa. Ele pode aceitar argumentos diferentes e pode retornar algum valor útil. Em todas as linguagens de programação, temos funções integradas e funções definidas pelo usuário.



PY - Funções II



Como inicializamos uma função ?

Iniciamos uma função com a palavra reservada para funções def o nome atribuído à função e os parênteses () utilizado para definição dos dados de entrada da função, também chamados de parâmetros.

Em seguida usa-se dois pontos : e abaixo o bloco de código a ser executado.

PY - Funções II

Revisando na prática

Juntos vamos criar uma função que verifica se um número é maior que o outro:

- Definimos o nome da nossa função, no meu caso será maiorQue, abrimos parênteses e definimos dois parâmetros, a e b que será definidos como números inteiros, finalizamos a primeira linha com dois pontos “:”.

- Vamos fazer uma condicional verificando os parâmetros para sabermos qual valor inseridos neles vão ser maior que o outro, não esqueçam de retornar uma mensagem personalizada informando qual valor inseridos no parâmetro é o maior.

```python
def maiorQue(a:int,b:int)→str:
    if a > b:
        return a,"é maior"
    else:
        return b, "é menor"
```

IN AULA 05 | Python

PY - Funções II

Exercícios de revisão

1. Crie uma função que inverte uma string e a retorna de trás para frente.

2. Crie uma função enigmática que recebe dois números e retorna o maior deles.

3. Crie uma função que recebe uma lista de palavras e retorna a palavra mais longa encontrada.

4. Crie uma função que recebe uma lista de cores e as imprime em sequência, criando um efeito visual de cores.

   use essa lista para inserir como argumento no parâmetro de sua função:

```
cores = ['\033[44m','azul','\033[0;0m',
         '\033[42m''verde','\033[0;0m',
         '\033[43m','amarelo','\033[0;0m']
```

PY - Funções II

*args e **kwargs

Os parâmetros ***args** e ****kwargs** são muito úteis quando você precisa definir funções flexíveis que possam lidar com diferentes números e tipos de argumentos. Eles oferecem grande versatilidade ao projetar suas funções em Python.

Esses parâmetros especiais permitem que você defina funções que podem receber um número variável de argumentos.

Vamos aprender em detalhes sobre cada um deles:

PY - Funções II

***args**

O parâmetro *args permite que uma função receba um número variável de argumentos posicionais. O asterisco (*) antes de args indica que todos os argumentos adicionais passados para a função após os argumentos obrigatórios serão empacotados em uma tupla chamada args. A função pode acessar e iterar sobre essa tupla para trabalhar com os argumentos adicionais.

Neste exemplo, a função minha_funcao() recebe um número variável de argumentos posicionais. Os argumentos adicionais são empacotados em uma tupla chamada args, e a função os imprime usando um loop for.

```
def minha_funcao(*args):
    for arg in args:
        print(arg)

minha_funcao("José","Maria","João")
```

PY - Funções II

**kwargs

O parâmetro **kwargs permite que uma função receba um número variável de argumentos nomeados. O duplo asterisco (**) antes de kwargs indica que todos os argumentos nomeados adicionais passados para a função após os argumentos obrigatórios serão empacotados em um dicionário chamado kwargs. A função pode acessar e manipular esse dicionário para trabalhar com os argumentos nomeados.

Neste exemplo, a função minha_funcao() recebe um número variável de argumentos nomeados. Os argumentos adicionais são empacotados em um dicionário chamado kwargs, e a função itera sobre as chaves e valores desse dicionário, imprimindo-os.

```python
def minha_funcao(**kwargs):
    for chave, valor in kwargs.items():
        print(f"{chave}:{valor}")

minha_funcao(nome="Rafaela", idade=23, cidade="Salvador")
```

PY - Funções II

***args **kwargs**

Você também pode combinar *args e **kwargs na definição de uma função para receber argumentos posicionais e nomeados em conjunto.

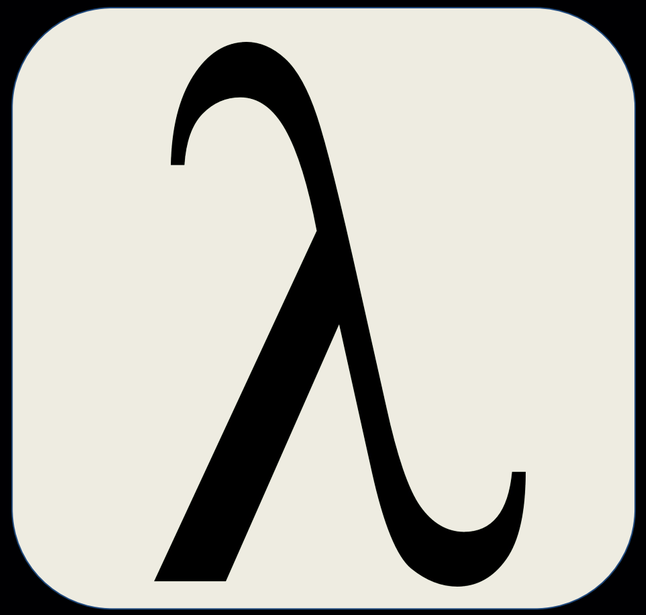
```python
def minha_funcao(*args, **kwargs):
    for arg in args:
        print(arg)
    for chave, valor in kwargs.items():
        print(chave, valor)

minha_funcao("Curriculo", "Desenvolvedor", nome="Alice", idade=25)
```

A função minha_funcao() recebe tanto argumentos posicionais (“Curriculo” e “Desenvolvedor”) quanto argumentos nomeados (nome="Alice" e idade=25), e a função os manipula separadamente.

PY - Funções II

Funções Lambda

As funções lambda, também conhecidas como funções anônimas, são funções pequenas e concisas que podem ser definidas em uma única linha de código. Elas são úteis quando você precisa de uma função simples que será usada apenas em um contexto específico.
As funções lambda não têm um nome definido, pois são anônimas. Elas são usadas principalmente como argumentos de outras funções ou em situações em que você precisa de uma função temporária.



PY - Funções II

Sintaxe:

A sintaxe geral de uma função lambda é a seguinte: **lambda parâmetros: comando**. Elas são definidas usando a palavra-chave lambda, seguida pelos argumentos da função, dois pontos :, e a expressão que será executada e retornada pela função.

```
variável = lambda parâmetro : comando
```

Veremos na prática em seguida.

PY - Funções II

entendendo melhor

Vamos ver alguns exemplos de funções lambda para entender melhor:

```python
quadrado = lambda x : x ** 2    #← Função lambda para calcular o quadrado de um numero.
print(quadrado(5))

par = lambda x: x % 2 == 0     #← Função lambda para verificar se o numero é par.
print(par(10))

name_upperCase = lambda n : n.upper() # ← Função lambada em strings
print(name_upperCase("jose"))
```

PY - Funções II

Expressões condicionais em funções lambda

Você pode usar expressões condicionais em funções lambda para criar lógica condicional dentro da expressão. Nesse exemplo, a função lambda par_impar recebe um número x.

A expressão condicional if x % 2 == 0 else "ímpar" verifica se x módulo de 2 é igual a zero. Se a condição for verdadeira, a função retorna a string "par"; caso contrário, retorna a string "ímpar".

```
# Função lambda usando condicional para verificar se um número é par ou ímpar
par_impar = lambda x: "par" if x % 2 == 0 else "ímpar"

# Exemplos de uso da função lambda
print(par_impar(5)) # Saída: ímpar
print(par_impar(-2)) # Saída: par
```

PY - Funções II

Veja outro exemplo:

Também podemos usar expressões condicionais mais complexas dentro de funções lambda

```
# Função lambda usando condicional para classificar informações em três categorias de menssagens.
valida_usuarios = lambda user: "Erro: usuario precisa ser definido" if user == "" else ("usuario não
pode ter menos de 4 digitos" if len(user) < 4 else "usuario definido com sucesso!")

# Exemplos de uso da função lambda
print(valida_usuarios(""))
print(valida_usuarios("zé"))
print(valida_usuarios("josé"))
```

Nesse caso, a função lambda valida_usuarios recebe uma string user. A expressão condicional verifica se user é igual a “”; se for verdadeiro, retorna “erro: usuário precisa ser definido”. Caso contrário, ela verifica se o comprimento da string atribuída a user é menor que 4; se for verdadeiro, retorna “usuário não pode ter menos de 4 dígitos”. Se nenhuma das condições anteriores for atendida, a função retorna a string "usuário definido com sucesso".

IN AULA 05 | Python

PY - Funções II

Exercícios

1. Escreva uma função lambda que receba um número e verifique se ele é par ou ímpar. A função deve retornar "par" se o número for par e "ímpar" caso contrário.

2. Implemente uma função lambda que receba duas strings e retorne a concatenação das duas, apenas se ambas as strings tiverem mais de 5 caracteres. Caso contrário, a função deve retornar uma mensagem de erro.

3. Escreva uma função lambda que receba um número e verifique se ele é maior que 10. Se for maior, a função deve retornar o próprio número; caso contrário, deve retornar o número dividido por 2.

4. Implemente uma função lambda que receba um número e verifique se ele é divisível por 3 e por 5. A função deve retornar "divisível" se a condição for satisfeita e "não divisível" caso contrário.

PY - Funções II

map, filter, reduce

As funções lambda também são frequentemente usadas em combinações de funções integradas ao Python. As funções map(), filter() e reduce() são muito úteis para manipulação de dados em Python. Elas permitem que você aplique transformações em elementos de uma sequência, filtre elementos com base em condições e reduza uma sequência a um único valor. Ao combinar essas funções com funções lambda, você pode escrever código mais conciso e expressivo.

A seguir vamos aprender como cada uma funciona.

PY - Funções II

map

A função map() recebe uma função e uma sequência (como uma lista) como argumentos e aplica a função a cada elemento da sequência. Ela retorna um objeto map que pode ser convertido em uma lista, se necessário. A função map() é útil quando você deseja aplicar uma determinada operação a todos os elementos de uma sequência.

Exemplo usando map() com uma função lambda:

```
numeros = [1, 2, 3, 4, 5]
quadrados = list(map(lambda x: x ** 2, numeros))
print(quadrados)
```

Neste exemplo, a função lambda x: x ** 2 é aplicada a cada elemento da lista numeros, produzindo uma nova lista quadrados com os valores ao quadrado.

PY - Funções II

filter

A função filter() recebe uma função e uma sequência como argumentos e retorna um objeto filter contendo apenas os elementos da sequência para os quais a função retorna True. Ela é útil quando você deseja filtrar elementos de uma sequência com base em uma determinada condição.

Exemplo usando filter() com uma função lambda:

```
numeros = [1, 2, 3, 4, 5]
pares = list(filter(lambda x: x % 2 == 0, numeros))
print(pares)
```

Neste exemplo, a função lambda x: x % 2 == 0 é aplicada a cada elemento da lista numeros, retornando apenas os elementos pares na lista pares.

PY - Funções II

reduce

A função reduce() está disponível no módulo **functools** e recebe uma função e uma sequência como argumentos. Ela aplica a função cumulativamente aos elementos da sequência, de modo que cada aplicação sucessiva usa o resultado da aplicação anterior. Ela retorna um único valor como resultado final.

```python
from functools import reduce

numeros = [1, 2, 3, 4, 5]
soma = reduce(lambda x, y: x + y, numeros)
print(soma)
```

Neste exemplo, a função lambda x, y: x + y é aplicada cumulativamente aos elementos da lista numeros, resultando em uma única soma de todos os elementos.

PY - Funções II

conclução

Nessa aula aprendemos como usar expressões condicionais em funções lambda para criar lógica condicional simples e também como usar funções agregadoras para manipulação de dados em Python.

Lembre-se de que as funções lambda são mais adequadas para tarefas simples, expressões mais complexas podem se tornar difíceis de ler e entender.

IN AULA 05 | Python

PY - Funções II

Exercícios

1. A partir de uma lista de strings, utilize map() e uma função lambda para converter todas as letras em maiúsculas.

2. A partir de uma lista de palavras, utilize filter() e uma função lambda para filtrar apenas as palavras que possuem mais de 5 letras.

3. Dada uma lista de valores numéricos, utilize reduce() e uma função lambda para obter o valor máximo da lista.

4. A partir de uma lista de dicionários, cada um representando uma pessoa com os campos "nome" e "idade", utilize `map( )` e uma função lambda para obter uma nova lista contendo apenas os nomes das pessoas.


IN
INFINITY SCHOOL
VISUAL ART CREATIVE CENTER