

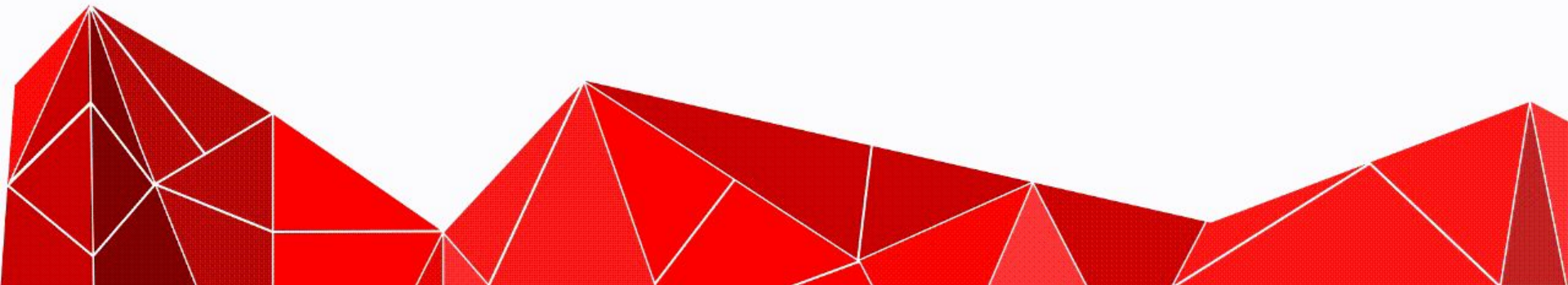


CODING

FUNÇÕES - PARTE 02

Press -> to continue

Powered by PET computação UFC

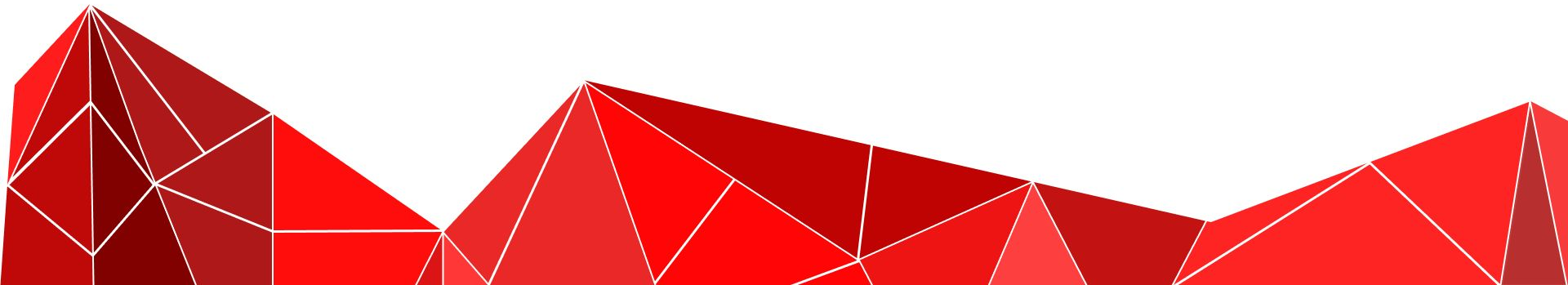


O'Que vamos ver hoje ?

Usando looping

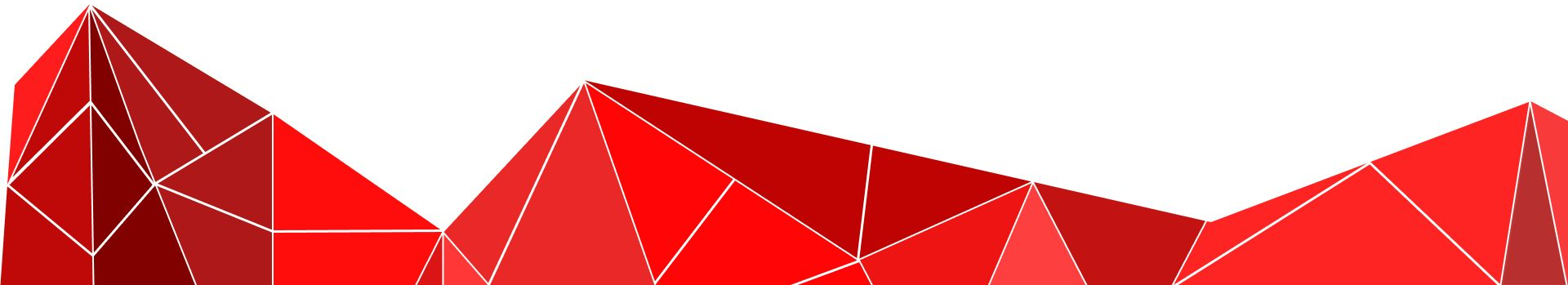
Listas como parâmetro

Número arbitrário de parâmetros



O'Que vamos ver hoje ?

Gravando a função em módulos



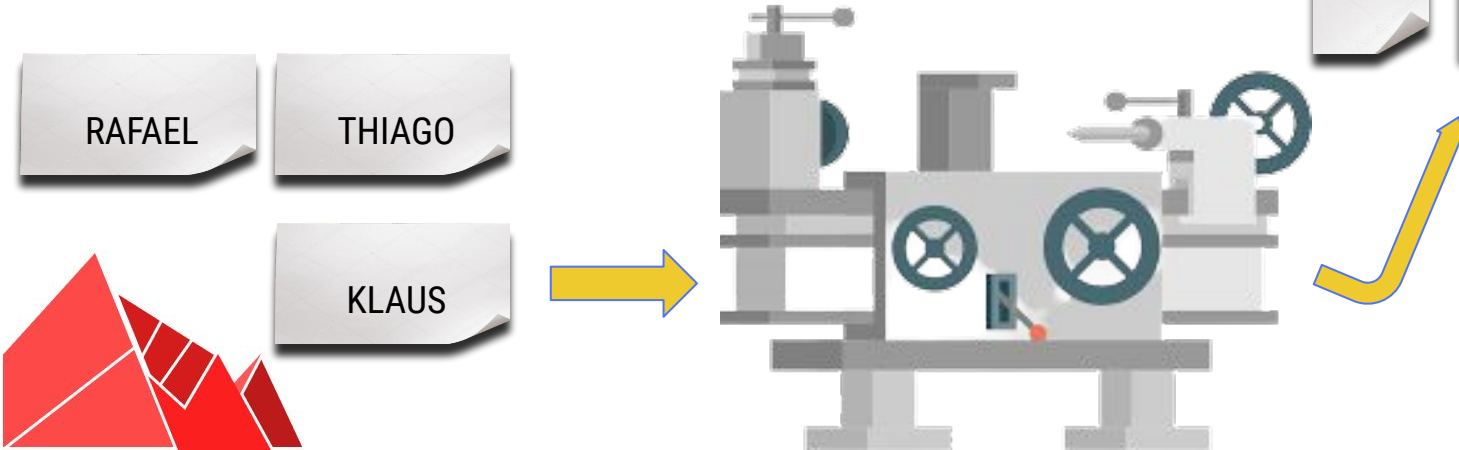
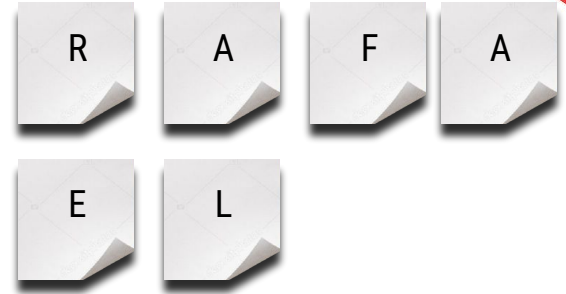


Usando looping



Usando loop com uma função

- Você pode usar funções combinado com tudo que aprendemos
- Podemos usar um loop para chamar uma função várias vezes
- Caso você precise chamar a mesma função para uma sequência
- Vamos relembrar da nossa máquina corta-palavras



Usando loop com uma função

- Agora temos uma sequência de palavras
- Podemos fazer um loop e chamar a função para cada uma

```
firstProgram.py
~/Documents
Open [v] [Python icon] Save [Menu icon] [Window icon] [Close icon]

1 strings_a_serem_cortadas = ["rafael", "thiago", "gabriel", "havillon"]
2
3
4 def wordCutter (string):
5     string_cortada = []
6     for letra in string:
7         string_cortada.append(letra)
8     return string_cortada
9
10 for string in strings_a_serem_cortadas:
11     print(wordCutter(string))
12
13
14
15
16
```



Usando loop com uma função

- O resultado é :

```
rafael@rafael-Lenovo-ideapad-330-15IKB: ~/Documents
rafael@rafael-Lenovo-ideapad-330-15IKB:~/Documents$ python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exec(open('firstProgram.py').read())
['r', 'a', 'f', 'a', 'e', 'l']
['t', 'h', 'i', 'a', 'g', 'o']
['g', 'a', 'b', 'r', 'i', 'e', 'l']
['h', 'a', 'v', 'i', 'l', 'l', 'o', 'n']
>>>
```





Listas como parâmetro



Listas como como parâmetros

- Ao invés de escrever o for fora da função
- Podemos passar a lista como parâmetro
- Assim, colocamos o FOR dentro da função



```
1 strings_a_serem_cortadas = ["rafael", "thiago", "gabriel", "havillon"]
2
3
4 def wordCutter (strings):
5     for string in strings:
6         string_cortada = []
7         for letra in string:
8             string_cortada.append(letra)
9         print(string_cortada)
10
11
12 wordCutter(strings_a_serem_cortadas)
13
```



Listas como parâmetros

- O resultado é o mesmo:

```
rafael@rafael-Lenovo-ideapad-330-15IKB: ~/Documents
rafael@rafael-Lenovo-ideapad-330-15IKB:~/Documents$ python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exec(open('firstProgram.py').read())
['r', 'a', 'f', 'a', 'e', 'l']
['t', 'h', 'i', 'a', 'g', 'o']
['g', 'a', 'b', 'r', 'i', 'e', 'l']
['h', 'a', 'v', 'i', 'l', 'l', 'o', 'n']
>>>
```



Modificando uma lista

- Podemos fazer qualquer coisa após passar a lista para a função
- Modificar ela é uma das coisas que podemos fazer
- Exemplo:

```
*firstProgram.py
~/Documents

Open  [v]  [🔍]  Save  [≡]  [—]  [□]  [✖]

1 clientes = ["rafael", "thiago", "gabriel", "havillon"]
2
3 print(clientes)
4
5 def removerClienteDevedor (nomeDoCliente, listaDeClientesDevedores):
6     if nomeDoCliente in listaDeClientesDevedores:
7         listaDeClientesDevedores.remove(nomeDoCliente)
8     else:
9         print("Cliente não encontrado")
10
11
12 removerClienteDevedor(nomeDoCliente = "rafael", listaDeClientesDevedores = clientes)
13 print(clientes)
14
```



Modificando uma lista

- Ao final a lista não terá mais o nome “rafael”
- Caso o nome não estivesse na lista
- A função retornará uma mensagem de erro

```
rafael@rafael-Lenovo-ideapad-330-15IKB: ~/Documents
rafael@rafael-Lenovo-ideapad-330-15IKB:~/Documents$ python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exec(open('firstProgram.py').read())
['rafael', 'thiago', 'gabriel', 'havillon']
['thiago', 'gabriel', 'havillon']
>>> 
```





**Número arbitrário de
parâmetros**



Número arbitrário de parâmetros

- Às vezes você não tem como prever o número exato de parâmetros
- Indicamos ao Python que o número de parâmetros é “livre”
- Obviamente, precisamos preparar a função para lidar com qualquer número

- Exemplo:



```
1
2
3
4 def printarIngredientes(*ingredientes):
5     print("A pizza vai conter os seguintes ingredientes: \n")
6     print(ingredientes)
7
8 printarIngredientes("queijo", "bacon", "molho de tomate", "frango")
```



Número arbitrário de parâmetros

- O resultado é :

```
rafael@rafael-Lenovo-ideapad-330-15IKB: ~/Documents
rafael@rafael-Lenovo-ideapad-330-15IKB:~/Documents$ python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exec(open('firstProgram.py').read())
['rafael', 'thiago', 'gabriel', 'havillon']
['thiago', 'gabriel', 'havillon']
>>> exec(open('firstProgram.py').read())
A pizza vai conter os seguintes ingredientes:

('queijo', 'bacon', 'molho de tomate', 'frango')
>>>
```



Número arbitrário de parâmetros

- Podemos melhorar o nosso print
- Vamos usar um FOR:

```
1
2
3
4 def printarIngredientes(*ingredientes):
5     print("A pizza vai conter os seguintes ingredientes: \n")
6     for ingrediente in ingredientes:
7         print(f"-{ingrediente}")
8
9 printarIngredientes("queijo", "bacon", "molho de tomate", "frango")
```



Número arbitrário de parâmetros

- A saída ficou um pouco mais organizada :

```
rafael@rafael-Lenovo-ideapad-330-15IKB: ~/Documents
rafael@rafael-Lenovo-ideapad-330-15IKB:~/Documents$ python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exec(open('firstProgram.py').read())
A pizza vai conter os seguintes ingredientes:

-queijo
-bacon
-molho de tomate
-frango
>>> 
```





Misturando os tipos de argumentos

- Podemos misturar os tipos de argumentos aprendidos anteriormente
- Quais ?
 - Posicionais
 - Keywords
 - Padrões
- Podemos usar eles com um número arbitrário de parâmetros
- Vamos ver um exemplo de posicionais
- Mas o conceitos se aplica aos outros tipos também



Misturando os tipos de argumentos

- Podemos adicionar a quantidade de pizzas
- Usando argumentos posicionais:

```
*firstProgram.py
~/Documents

Open  Save  -  □  ×

1
2
3
4 def printarIngredientes(quantidade, *ingredientes):
5     print(f"Fazer {quantidade} pizza(s) com os seguintes ingredientes: \n")
6     for ingrediente in ingredientes:
7         print(f"-{ingrediente}")
8
9 printarIngredientes(12, "queijo", "bacon", "molho de tomate", "frango")
10
```



Misturando os tipos de argumentos

- O resultado é quase o mesmo:

```
rafael@rafael-Lenovo-ideapad-330-15IKB: ~/Documents
rafael@rafael-Lenovo-ideapad-330-15IKB:~/Documents$ python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exec(open('firstProgram.py').read())
Fazer 12 pizza(s) com os seguintes ingredientes:

-queijo
-bacon
-molho de tomate
-frango
>>> 
```



Guardando funções em módulos

- Funções são partes bem específicas do código
- Assim, podemos escrevê-la em um arquivo separado
- Apenas IMPORTANDO para o arquivo principal
- Deixa o código principal mais limpo
- Focado apenas na lógica em um **nível mais alto**



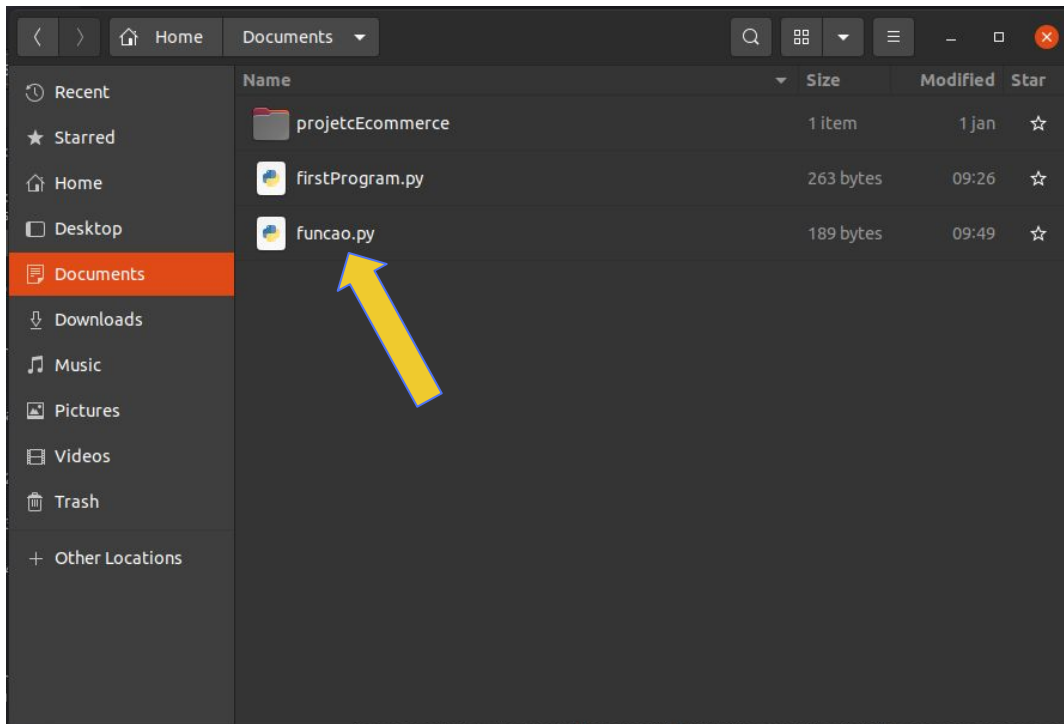
Guardando funções em módulos

- Imagine o código principal como uma fábrica que tem uma esteira principal
- Conforme precisamos
- Adicionamos mais uma máquina



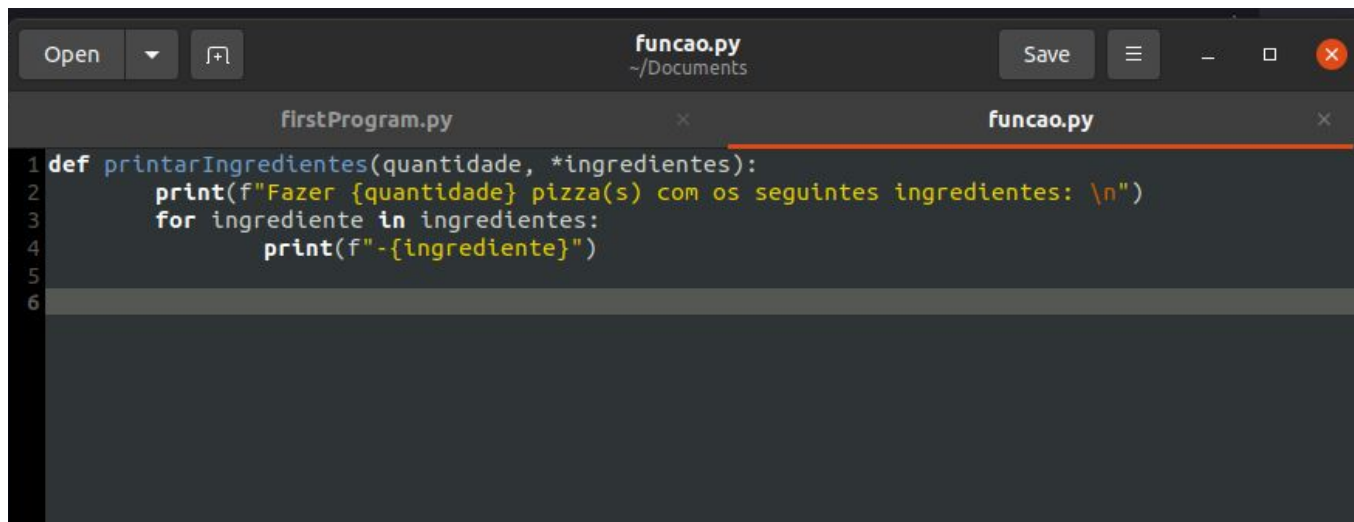
Guardando funções em módulos

- Primeiro criamos o arquivo com a(s) funções



Guardando funções em módulos

- Depois, obviamente, escrevemos a função no arquivo



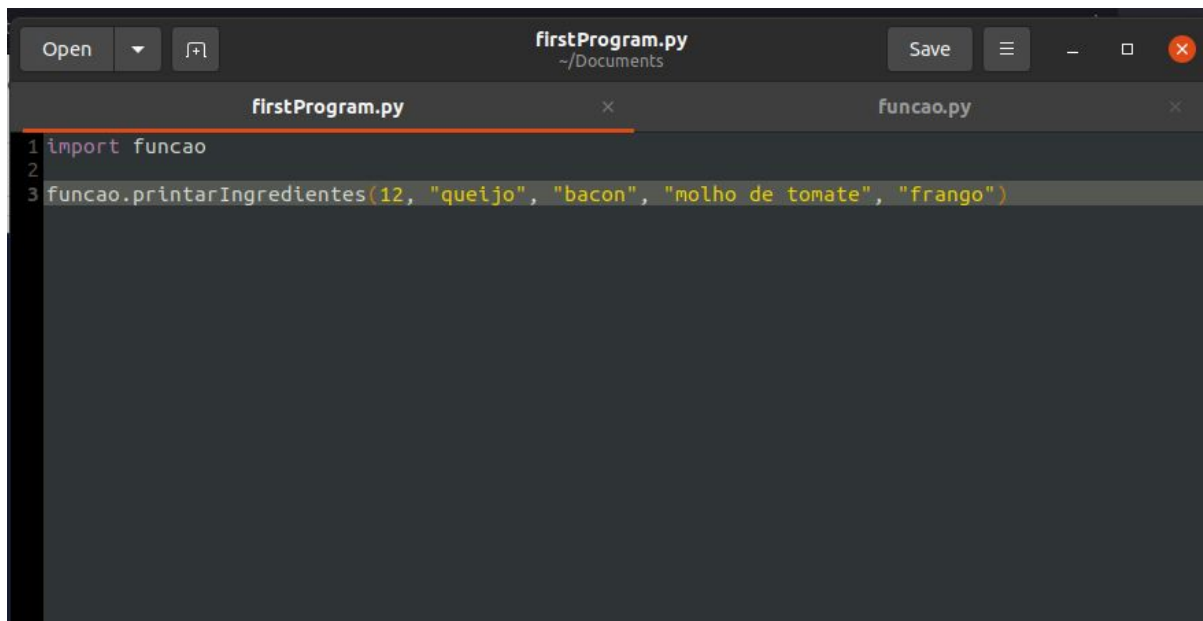
The screenshot shows a code editor window with two tabs: 'firstProgram.py' and 'funcao.py'. The 'funcao.py' tab is active and contains the following Python code:

```
1 def printarIngredientes(quantidade, *ingredientes):
2     print(f"Fazer {quantidade} pizza(s) com os seguintes ingredientes: \n")
3     for ingrediente in ingredientes:
4         print(f"-{ingrediente}")
5
6
```



Guardando funções em módulos

- Por fim, chamamos a função no arquivo principal



```
firstProgram.py
~/Documents
Save
firstProgram.py
funcao.py
1 import funcao
2
3 funcao.printarIngredientes(12, "queijo", "bacon", "molho de tomate", "frango")
```



Guardando funções em módulos

- E o resultado é o mesmo:

```
rafael@rafael-Lenovo-ideapad-330-15IKB: ~/Documents
rafael@rafael-Lenovo-ideapad-330-15IKB:~/Documents$ python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exec(open('firstProgram.py').read())
Fazer 12 pizza(s) com os seguintes ingredientes:

-queijo
-bacon
-molho de tomate
-frango
>>> 
```



Funções Lambda

- O que é isso ?
 - As funções Lambda são chamadas de funções anônimas
 - Nada mais são do que funções que o usuário “não precisa definir”, ou seja, não vai precisar escrever a função e depois utilizá-la dentro do código.
- É como se declararmos elas de uma maneira escondida, dentro de uma variável.
- Vamos analisar a sintaxe:

Funções lambda

- Vamos analisar a sintaxe:

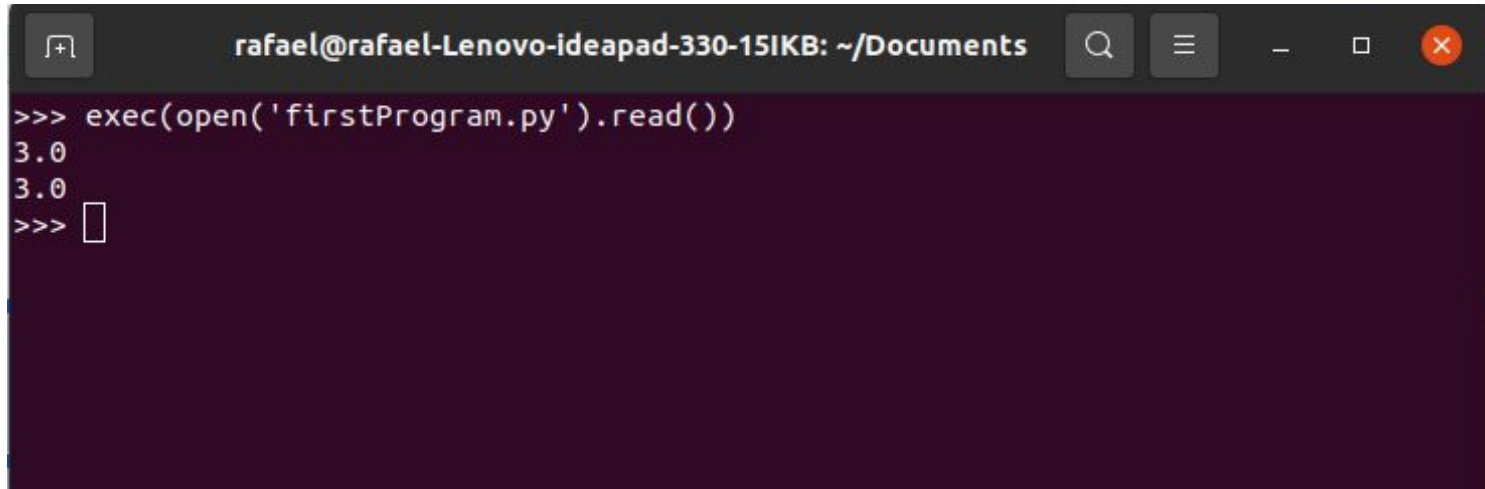


```
1 def calcularPorcentagem (preco):  
2     return preco*0.3  
3  
4 calcularPorcentagem2 = lambda x: x*0.3  
5  
6 print(calcularPorcentagem(10))  
7 print(calcularPorcentagem2(10))
```

- Na função lambda o x é como se fosse o “preço”
- Na função lambda não precisamos colocar a palavra chave “return”, pois ela já retorna o valor do cálculo.

Funções lambda

- E o resultado é o mesmo para as duas funções



```
rafael@rafael-Lenovo-ideapad-330-151KB: ~/Documents
>>> exec(open('firstProgram.py').read())
3.0
3.0
>>> 
```

A terminal window with a dark purple background. The title bar shows the user 'rafael' on a 'rafael-Lenovo-ideapad-330-151KB' machine, in the directory '~/Documents'. The terminal contains three lines of text: a Python command to execute a file, followed by two lines of output, and a blank line with a cursor.

A decorative graphic on the left side of the slide, consisting of a series of overlapping triangles in various shades of red and orange, creating a dynamic, geometric pattern.

**Ufa ! AGORA VAMOS
PRATICAR !!!!!!!**



Exercícios

Crie um programa que tenha uma função que receba uma lista de números e retorne o valor da multiplicação entre todos eles.



Exercícios

Crie um programa que tenha uma função `fatorial()` que receba dois parâmetros: o primeiro que indique o número a calcular e outro chamado `show`, que será um valor lógico (opcional) indicando se será mostrado ou não na tela o processo de cálculo do fatorial.

Exercícios

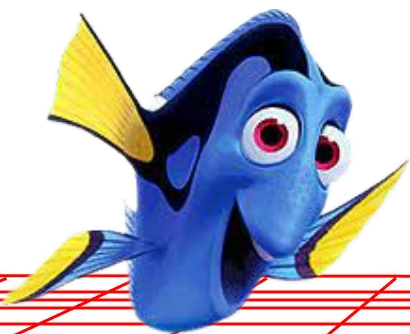
Construa uma função que desenhe um retângulo usando os caracteres '+', '-' e '|', recebendo o número de linhas e colunas.

Exemplo: 5 linhas e 5 colunas:

```
- - - - -  
| + + + + + |  
| + + + + + |  
| + + + + + |  
| + + + + + |  
| + + + + + |  
- - - - -
```

"Apenas continue nadando"

-Dory



The background of the slide features a red grid pattern. The grid is composed of horizontal and vertical lines, with additional diagonal lines extending from the corners towards the center, creating a perspective effect. The text is centered in the middle of the slide.

Obrigado!!
Ainda com dúvida ?
Entre em contato