

## Alinhamento de Expectativas



Objetivo: Dar os primeiros passos no mundo da programação e ser capaz de utilizar lógica para resolver problemas por meio da programação utilizando o Python.

Só é possível Aprender <-> Fazendo

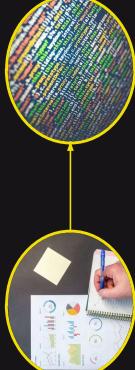
Regra n° 1 do curso: Faça os Exercícios  
Regra n° 2 do curso: Siga a regra n° 1

## O que veremos neste curso:

- 1 Listas e Tuplas
- 2 Tipos de Dados
- 3 Operadores Aritméticos
- 4 Trabalhando com Strings
- 5 Condicionais
- 6 Loops
- 7 Dicionários
- 8 Funções Python e Lambdas
- 9 Clases e Pacotes
- 10

## Introdução à Programação

Como tomar melhores decisões e mais rápido?



## Tomada de Decisão

Complexa: Aonde invisto R\$100 mil?

- Como está a situação financeira da empresa?
- Como estão os indicadores financeiros por fabrica? E por região?
- Produto? E por região?
- Como está o mercado?
- Como está o juros?
- Como está os concorrentes?
- Quais são os outros fatores externos? Previsão do tempo?
- Pandemias?
- ...

Obs.: Seres humanos eram de 30 a 50% das tomadas de decisões.

## Características do Python

- 1 Fácil e Intuitivo
- 2 Multiplatforma
- 3 Código Aberto
- 4 Linguagem Organizada
- 5 Bibliotecas "ilimitadas"
- 6 Orientada a Objetos

## O que veremos neste curso:

- 1 Introdução ao Python 0
- 2 Tipos de Dados
- 3 Operadores Aritméticos
- 4 Trabalhando com Strings
- 5 Condicionais
- 6 Loops
- 7 Dicionários
- 8 Funções Python e Lambdas
- 9 Clases e Pacotes
- 10

## Introdução à Programação

Como fazer ainda mais com muito menos?



## Tomada de Decisão

Simples: O que vou comer?



## Por que Python?

- 1 Basicamente porque é uma linguagem de simples e de propósito geral.
- 2 PHP, HTML, Java Script
- 3 C++, Java
- 4 Fortran
- 5 Desenvolvimento Web (sites)
- 6 Desenvolvimento de Sistemas e Softwares
- 7 Voltado para Análises Numéricas
- 8 Linguagem Organizada
- 9 Bibliotecas "ilimitadas"
- 10 Orientada a Objetos



Mas como assim???



Engenheiro Mecânico  
Bach. Belt em Lean Six Sigma  
Especialista em Projetos, BI, Estatística e Data Analytics  
Experiente em: Python, Fortran, C++, Java Script, Desenvolvedor de Aplicativos de Robot Process Automation, Aplicações de modelos de Algoritmos Genéticos e Redes Neurais

**Everton Menezes**  
Head de Operações  
@dhngroup

## O que veremos neste curso:

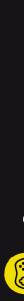
- 1 Introdução ao Python 0
- 2 Tipos de Dados
- 3 Operadores Aritméticos
- 4 Trabalhando com Strings
- 5 Condicionais
- 6 Loops
- 7 Dicionários
- 8 Funções Python e Lambdas
- 9 Clases e Pacotes
- 10

## Módulo 1: Introdução ao Python Zero

Como fazer mais com menos?



## Tríplice da Aplicação da Programação



### Games



### Automações



### Tomadas de Decisões

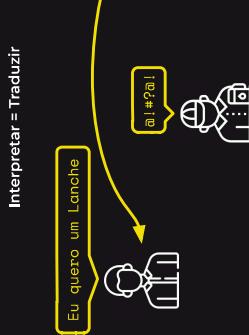
## Bem-vindo ao mundo da Programação

Como fazer mais com menos?

Como tomar decisões melhores e mais rápido?

Como melhorar e facilitar a vida dos seres humanos???

## O que é interpretar?



Interpretar = Traduzir

## Interpretores e IDEs

O Python é uma linguagem interpretada, é um passo a passo!



Mas como assim???

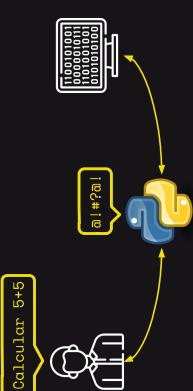
## Python

Uma das linguagens mais **simples** de se aprender.

- É **Flexível**, pois pode ser usada nas mais diversas aplicações, desde Web até Data Science
- Possui "infinitos" **Pacotes** que facilitam nossa vida na programação.

- Games: Battlefield 2
- IA e Machine Learning: Skyscanner
- Casas inteligentes: raspberry e arduino
- Sistemas de recomendações: YouTube, Netflix
- Reconhecimento de imagem: Face ID, Google Car

## O que é interpretar em programação???



## Características de uma Linguagem Interpretada



- Executado linha a linha
- Mais fácil de aciar erros
- Mais devagar a execução
- A interpretação é executada no computador do usuário
- Facilita a segurança
- Menor espaço

## Aplicações

Áreas: Desde Inteligência Artificial até biotecnologia e computação 3D.

- Algunas aplicações:
- Games: Battlefield 2
- IA e Machine Learning: Skyscanner
- Casas inteligentes: raspberry e arduino
- Sistemas de recomendações: YouTube, Netflix
- Reconhecimento de imagem: Face ID, Google Car

## PowerShell

```
Windows PowerShell Copyright (C) Microsoft Corporation. Todos os direitos reservados. Instale o PowerShell[1] mais recente para obter novos recursos e melhoramentos! https://aka.ms/PSWindows
PS C:\Users\leventon> python
Python 3.9.7 (tags/v3.9.7:101ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)]
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Olá, mundo')
Olá, mundo!
```

## Mão na Massa

Agora chegou a hora que vocês estavam esperando...

Let's CODE!



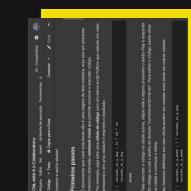
## Interpretadores

Eles interpretam o desejo do programador em código-máquina.

São executados passo a passo, facilitando a detecção e conserto de erros.

## Google Colab

Ambiente de programação em Python da Google para democratizar ainda mais o estudo e a prática de programação.



## Tipos de Dados

Os tipos básicos são: int e float para números, string para textos e booleana para V ou F.

```
#Aula de Introdução
1 print('Leventon') #str
2 print(29.5) #float
3 print(12) #int
4 print(True) #bool
```

## Trabalhando com texto

Textos em Python são colocados entre aspas 'str'

Vamos mostrar a mensagem para o usuário:

Olá, mundo!

'Olá, mundo!'

Olá, mundo!

Olá, mundo! #função imprime!

print(' Olá, mundo! ') #função imprime!

Olá, mundo!



## Antes de tudo

### Como iremos estudar:

- Assista a aula teórica **[Teoria]**
- Abra o arquivo base - **[Próxima aula]**
- Acompanhe a aula prática **[Prática]**
- Code junto comigo durante a aula
- Faça os exercícios

## Módulo 2:

### Tipos de Dados

- 1 Colab
- 2 Tipos Primitivos de Dados
- 3 Funções: print, input e type

## IDEs - Integrated Development Environment



Ambiente de Desenvolvimento Integrado

Editor de código-fonte + Interpretador + Debugger

PyCharm, Sublime, ... Google Colab

## Podemos imprimir uma soma?

```
print(7 + 2)
9
print(7.2 + 2.8)
10.0
print('7' + '2')
72
```

## Trabalhando com número

Números em Python são colocados direto e podem ou não ter casas decimais.

```
7 #int
```

```
7.2 #float
```

Podemos imprimir números?

```
print(7)
7
print(7.2)
7.2
```

## Outros exemplos - str

```
print('Olá, mundo!') #aspas simples
print('Olá, mundo!') #aspas duplas
print('10') #numero
print(" ") #espaço em branco
print() #linha em branco
print('ai! #ão!') #caracten especial
```

Observe as cores!!!

## Entendendo a estrutura

print('Olá, mundo!') #função imprimir  
String [Tipo de Dado] = de cadeia de caracteres 'str'  
↓  
print('Meu nome é ' + 'Everton')  
Meu nome é Everton  
print('Meu nome é ', 'Everton')  
Meu nome é Everton  
print('Eu tenho ', 3, ' cachorros ')
Eu tenho 3 cachorros

Observe as cores!!!

## Podemos juntar strings?

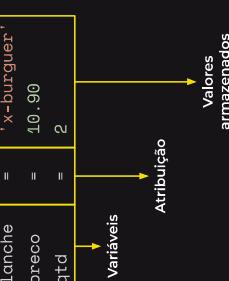
```
print('Meu nome é ' + 'Everton')
Meu nome é Everton
print('Meu nome é ', 'Everton')
Meu nome é Everton
print('Eu tenho ', 3, ' cachorros ')
Eu tenho 3 cachorros
```

## Podemos juntar strings?

```
print('Eu tenho ' + 3 + ' cachorros ')
TypeError: can only concatenate str (not
'int') to str
Na função print() podemos misturar tipos de variáveis desde que elas sejam separadas por []
Já o operador '+' só pode ser usado com tipos de dados compatíveis.
Int + int
Int + float
Str + Str
soma
soma
concatenar
```

## Podemos fazer contas?

```
lanche = 10.9
batata = 5.9
refri = 3
conta = lanche + batata + refri
print(conta)
19.8
```



## Podemos fazer impressão composta?

```
lanche = 10.9
batata = 5.9
pgto = 'cartão'
conta = lanche + batata
print('Você pediu um lanche de: R$', lanche, 'e
uma batata de R$', batata, ', totalizando: R$,
', pgto)
conta = lanche + batata
print('Você pediu um lanche de: R$ 10.9 e uma batata
de: R$ 5.9 Totalizando: R$ 16.8 . O seu
pagamento será feito no: ', pgto)
```

## Tipos de Dado = bool

```
lanche = 10.9
batata = 5.9
refri = 3
conta = lanche + batata + refri
print(conta)
19.8
```



## Tipos de Dado = bool

```
lanche = 10.9
batata = 5.9
refri = 3
conta = lanche + batata + refri
print(conta)
19.8
```



## type()

```
type('Eventon')
str
type(5)
int
type(5.7)
float
```



## type()

```
type('Eventon')
str
type(5)
int
type(5.7)
float
```

## input()

```
lanche = input('Qual lanche você quer? ')
print('Você quer um: ', lanche)
... [Qual lanche você quer? x-burguer]
Você quer um: x-burguer
input( ..... )
Frase descrevendo a informação que o usuário deve inserir.
O dado sempre vem como str
```

## input()

```
lanche = input('Qual lanche você quer? ')
print('Você quer um: ', lanche)
... [Qual lanche você quer? x-burguer]
Você quer um: x-burguer
input( ..... )
Frase descrevendo a informação que o usuário deve inserir.
O dado sempre vem como str
```

## Mão na Massa

Os operadores mais usados são: mais, menos, vezes, dividir e potência



Let's CODE

## Operadores Aritméticos

```
print(7+2) = 9
print(7-2) = 5
print(7*2) = 14
print(7/2) = 3.5
print(7**2) = 49
```



## Módulo 3: Operadores Aritméticos

## Voltando para aritmética

### Ordem de precedência:

```

1. ()          7 + 2 * 4
2. **          #
3. * / % //  #
4. + -        (7 + 2) * 4
  
```

Tome cuidado com as contas e valide os resultados do seu programa!!!

## Mas...

### Qual a aplicação?

Como mostrar para o usuário?

```

print('Bem Vindo')
print('-----')
print('Olá : + mundo !')
print('Olá mundo !')
print('Olá : *5')
print('Olá Olá Olá Olá')
  
```

## str()

```

str(True)
'True'
str(False)
'False'
  
```

## Entendendo a estrutura da Máscara

```

a = 245
b = 2.85895
print(r'[0:10d] e [1:10.4f]')
[245 | e | 2.8595]
  
```

1 → 2<sup>a</sup> variável no format  
: → separador  
i → separar 10 espaços para o número  
d → limitar em 4 floats (algarismos após vírgula)

## Conicionais

Se algo é verdade, faça uma coisa, senão faça outra coisa

```

if True:
    print('É True')
else:
    print('É False')
  
```

## Operadores só funcionam com números?

Dos operadores acima, existem 2 que conseguimos usar com strings:

+	concatena
*	repete n vezes

```

print('Olá : + mundo !')
Olá mundo !
print('Olá : *5')
Olá Olá Olá Olá Olá
  
```

## int()

```

int(1.857)
1
int('4')
4
int('tex34')
ValueError: invalid literal for int() with
base 10: 'tex34'
  
```

## Transformando variáveis

Vamos agora aprender as funções:

```

int() #Converte variável para inteiro
float() #Converte variável para real
str() #Converte variável para string
  
```

Quando iremos usar?  
Quando as variáveis vierem num formato que não conseguimos manipular como queremos.  
Exemplo: o input()

## Operadores

7	+	2	==	9	sona
7	-	2	==	5	subtração
7	*	2	==	14	multiplicação
7	/	2	==	3.5	divisão
7	**	2	==	49	potência
7	//	2	==	3	divisão inteira
7	%	2	==	1	resto da divisão

Operadores → Igualdade

## print() com .format()

```

Vamos agora aprender as funções:
int() #Converte variável para inteiro
float() #Converte variável para real
str() #Converte variável para string
  
```

Quando iremos usar?  
Quando as variáveis vierem num formato que não conseguimos manipular como queremos.  
Exemplo: o input()

## Entendendo a estrutura do print() com .format

```

a = 2
b = 3
print('mais texto [ ] mais texto [ ]', format(a,b))
{'[Máscara]': ficam na posição das variáveis
'format(a,b)': [Método] = separa as variáveis que estão dentro do print()
}
[Texto 2, mais texto 3]
  
```

## print() com .format()

```

# Imprimindo na forma tradicional
print('Você pediu um lanche de: R$ ', 
      lanchePreco, 'e uma batata de: R$', 
      batataPreco, 'totalizando: R$', pedidoValorTotal)

Repare no tanto de aspas e vírgulas
  
```

## Operações aritméticas

### Mão na Massa

Let's CODE



## Conversão de variáveis

int() = converte para inteiro  
float() = converte para real  
str() = converte para string

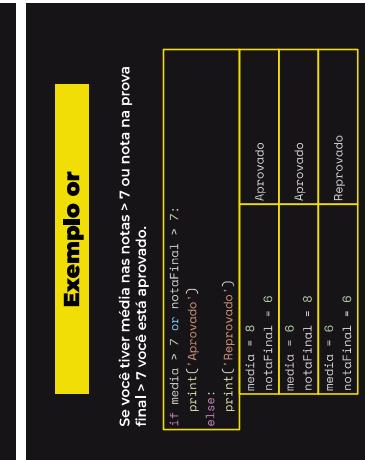
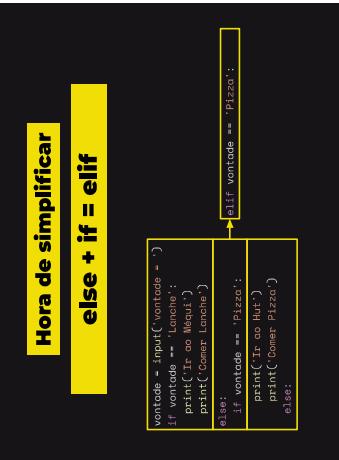
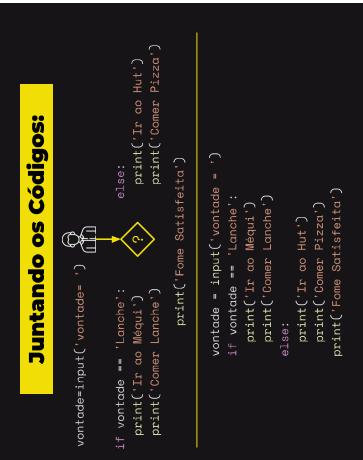
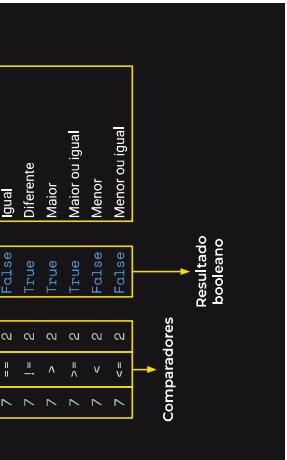
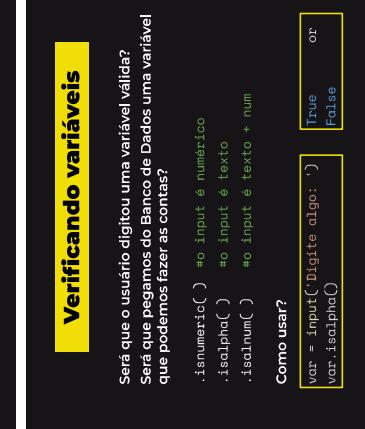
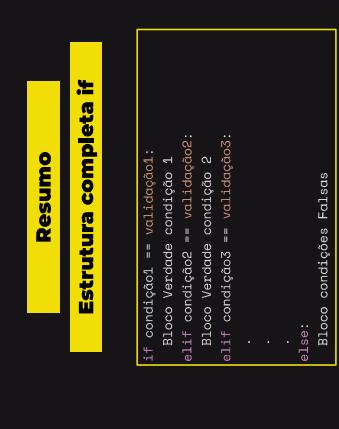
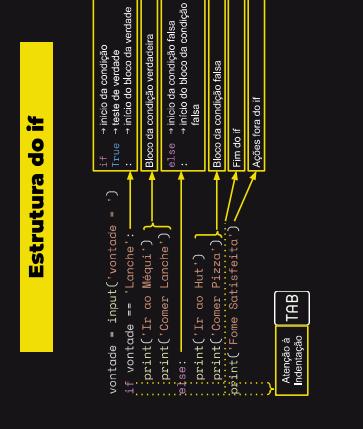
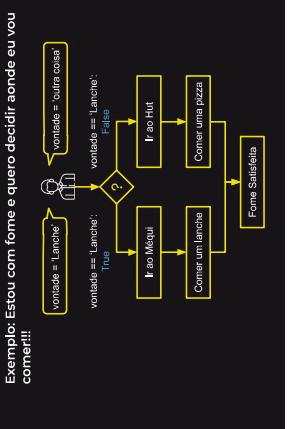
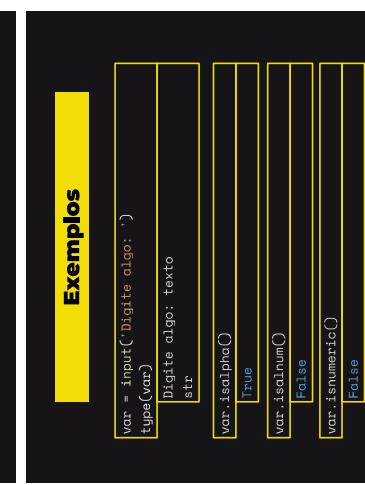
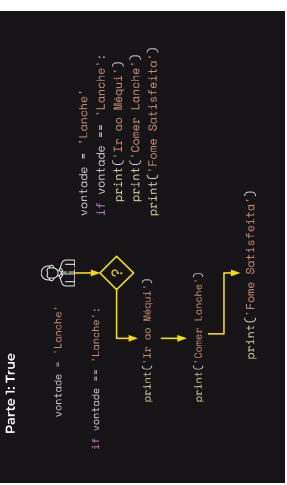
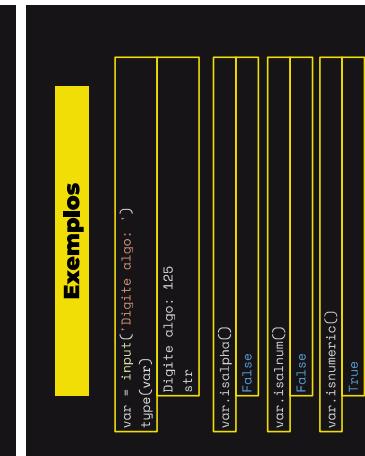
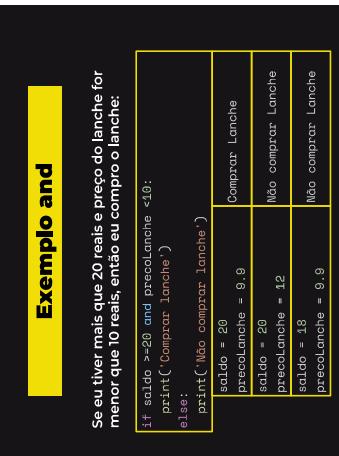
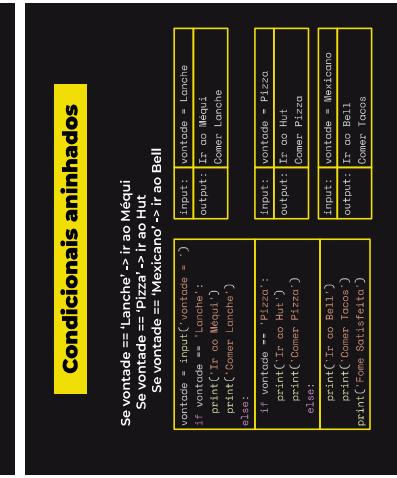
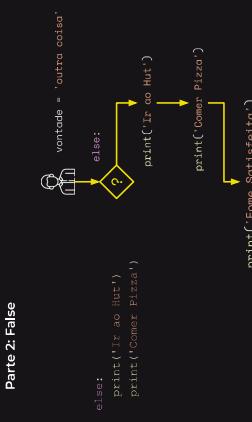
## print() com .format()

print('textos {[}:format()])

De condições para códigos

De condições para código

## Comparadores



## Estrutura if - elif - else

```
if condição1 == validação1:  
    Bloco Verdade condicão 1  
    ...  
elif condição2 == validação2:  
    Bloco Verdade condicão 2  
elif condição3 == validação3:  
    ...  
else:  
    Bloco condições Falsas
```

.isnumeric() → input é número  
.isalpha() → input é texto  
.isalnum() → input é texto + num

## Teste de variável

## Mão na Massa

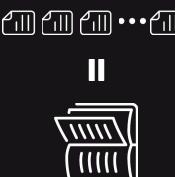
Let's CODE!



## O que é uma repetição?

É algo que precisamos executar várias vezes.

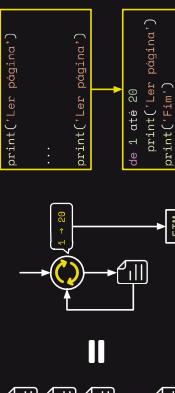
Exemplo: Ler um livro de 20 páginas



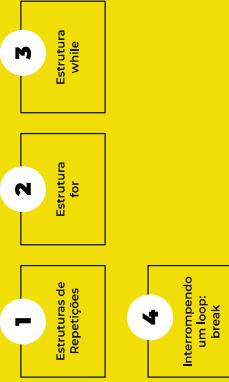
## O que é uma repetição?

É algo que precisamos executar várias vezes.

Exemplo: Ler um livro de 20 páginas



## Módulo 5: Repetições



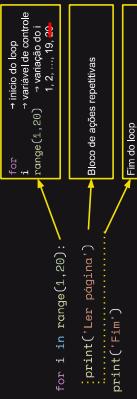
## Repetições

Faça algum processo repetitivo até ele terminar

```
for i in passo:  
    print('For: Execute')  
  
j = True  
while j == True:  
    print('While: Execute')  
    j = False
```

## Entendendo a Estrutura do for

Exemplo: Ler um livro de 20 páginas



## Exemplo 2

Exemplo: Somar 3 números digitados pelo usuário

```
soma = 0  
for i in range(1,3+1):  
    n = int(input('Digite um numero: '))  
    soma = soma + n  
    print('Soma = [ ]'.format(soma))  
  
    Dige um numero: 1  
    n = 1 e soma = 1  
    Dige um numero: 5  
    n = 1 e soma = 6  
    Dige um numero: 4  
    n = 4 e soma = 10  
    soma = 12
```

## Exemplo

Exemplo: Somar os números ímpares consecutivos iniciando em 1 e terminando em 9.

```
soma = 0  
for i in range(1,9+1,2):  
    soma = soma + i  
    print('i = [ ] e soma = [ ]'.format(i,soma))  
    print('Soma = [ ]'.format(soma))  
  
    i = 1 e soma = 1  
    i = 3 e soma = 4  
    i = 5 e soma = 9  
    i = 7 e soma = 16  
    i = 9 e soma = 25  
    soma = 25
```

## Modificando o range()

```
range(início, fim, passo)  
    início → número inteiro inicial  
    fim → número inteiro final  
    passo → quantidade somada em cada loop
```

range(1, 7, 1)	→ 1, 2, 3, 4, 5, 6, 7
range(10, 16, 2)	→ 10, 12, 14, 16
range(5, 1, -1)	→ 5, 4, 3, 2, 1

## Cálculos incrementais:

No mundo da programação é normal pegarmos um valor anterior e atualizarmos ele com alguma soma, subtração ou multiplicação. Como vimos nos 2 últimos exemplos.

Existe uma maneira mais resumida de fazer essa operação.

```
soma = 10  
soma = soma + 1  
soma = 11  
  
soma = 10  
soma = soma - 5  
soma = 5
```

## Cálculos incrementais:

No mundo da programação é normal pegarmos um valor anterior e atualizarmos ele com alguma soma, subtração ou multiplicação. Como vimos nos 2 últimos exemplos.

Existe uma maneira mais resumida de fazer essa operação.

```
soma = 10  
soma = soma * 2  
soma = 20  
  
soma = 10  
soma = soma / 2  
soma = 5
```

## Exemplo e se não soubermos a quantidade de passos?

Quero ler um capítulo de um livro, mas não sei quantas páginas possui este capítulo



## Exemplo

Quero ler um capítulo de um livro, mas não sei quantas páginas possui este capítulo



usaremos o: while



Funções com Tuplas	
<pre>t1 = (0, 1, 2, 3, 4, 5, 6) t2 = (1, 3, 5, 6, 8)</pre>	
Tamanho da Tupla	<code>len(t1)</code>
Ordenação	<code>sorted(t1)</code>
	<code>[0, 1, 2, 3, 4, 5, 6]</code>
	Repare que está dentro de <code>[]</code> então o output não é uma tupla!
Contagem termos	<code>t2.count(3)</code>
Posição de termos	<code>t2.index(7)</code>
	<code>3</code>
	<code>t2.index(3)</code>
	<code>1</code>
	<code>t2.index(3, 3)</code>
	<code>4</code>
Concatenar tuplas	<code>t2 + t4</code>
	<code>[1, 3, 3, 5, 3, 5,</code>
	<code>7, 0, 6, 4, 2, 6]</code>
E se quisermos mostrar o índice da tupla no for?	
Aí vamos usar a função enumerate()	
	<code>t2 = ('Lanche', 'Batata', 'Refrí')</code>
	<code>for ind, valor in enumerate(t2):</code>
	<code>    print(f'Ind: {ind} - Valor: {valor}')</code>
	<code>print('Fim')</code>
	<code>Item 0: Lanche</code>
	<code>Item 1: Batata</code>
	<code>Item 2: Refri</code>
	<code>Fim</code>
Listas	
	As listas são variáveis compostas mutáveis
	<code>lista = ['var1', 'var2', 'var3']</code>
Funções com Listas	
	<code>l1 = [0, 1, 3, 5, 7]</code>
Adicionando termos	<code>l1.append()</code>
	<code>l1.append(7)</code>
	<code>l1 = [0, 1, 3, 5, 7, 7]</code>
Inserindo termos	<code>l1.insert(pos, valor)</code>
	<code>l1.insert(3, 4)</code>
	<code>l1 = [0, 1, 3, 4, 5, 7, 7]</code>

## Tipos de dados

### dentro da Tupla

As variáveis compostas aceitam qualquer tipo de dados:

```
tuplaid = ('Lançone', 'Barata', 'herói')
tuplaz = (2, 6, 4, 5, 7)
tupla3 = (True, False, False)
```

Aceitam também tipos misturados de dados

```
tuplo4d = ('Lançone', 10, 9, False)
print(type(tuplo4d))
print(type(tuplo4d[0]))
print(type(tuplo4d[0:1]))
print(type(tuplo4d[1:2]))
print(type(tuplo4d[2:3]))
```

### Exemplo

Exemplo: Dada a tupla abaixo. Calcule a soma de seus valores.

```
soma = 0
for i in t1:
    soma += i
print('soma = {}' .format(soma))
```

soma = 20

## Módulo 7: Variáveis Compostas - Listas

- 1 O que são Listas
- 2 Funções com Listas
- 3 Listas dentro de Loops
- 4 O que são Listas dentro de Listas
- 5 Trabalhando com Listas Compostas

### Funções com Listas

Como já falamos a maior vantagem das listas é poderem ser modificadas. Vamos ver as principais modificações:

- Modificando um termo

```
11[2] = 3
11[0:2] = [0:1]
11[0:2] = [0:1]
```

```
11 = [2, 6, 4, 5, 7]
```

```
11 = [2, 6, 3, 5, 7]
```

```
11 = [0, 1, 3, 5, 7]
```

<h2>Definição de Tupla</h2> <p>Tuplas são variáveis compostas imutáveis</p> <p>Declarando:</p> <pre>tupla = ('Lanche', 'Batata', 'Refrí1')</pre> <p>Modificando:</p> <pre><del>tupla[1] = 'Milk Shake'</del> <del>tupla = ('Lanche', Milk Shake, 'Refrí1')</del></pre> <p>Após declarada, é impossível mudar uma tupla</p>	<h2>Tuplas dentro de for</h2> <p>E se a tupla só tiver texto?</p> <pre>t2 = ('Lanche', 'Batata', 'Refrí1')  for i in t2:     print('Você quer pedir um(a) {}?'.format(i)) print('Fim')  Você quer pedir um(a) Lanche? Você quer pedir um(a) Batata? Você quer pedir um(a) Refri? Fim</pre>	<h2>Mão na Massa</h2> <p>Let's CODE</p> 	<h2>Listas vs Tuplas</h2> <p>Todas as técnicas que vimos até agora funcionam tanto com <b>Listas</b> quanto com <b>Tuplas</b>:</p> <ul style="list-style-type: none"> <li><input type="checkbox"/> Acessar itens: [0] [0:2] [:3] [3:] [-4]</li> <li><input type="checkbox"/> Métodos: .index() - .count()</li> <li><input type="checkbox"/> Funções: len() - sorted()</li> <li><input type="checkbox"/> Concatenação: lista1 + lista2</li> <li><input type="checkbox"/> Lista dentro de for: for i in lista:       <pre>for i, in enumerate(lista):</pre>       Além de tudo isso, podemos modificar elementos, deletar, adicionar e muito mais!!!</li> </ul>
--	--	--	---

## Fatiamento

```
tupla = [6, 1, 2, 3, 4]
Acessando termos consecutivos
print(Cupido[0:2])
print(Cupido[1:4])
```

Acessando o termo N até o último e vice-versa

```
print(Cupido[1:])
print(Cupido[3:])
print(Cupido[6, 4, 5])
```

Acessando os últimos termos

```
print(Cupido[-1:])
print(Cupido[-3:])
```

## Tuplas dentro de for

Até agora, rodámos nossos loops dentro de um range. Porém podemos também rodar dentro de tuplas e listas:

```
t1 = (2, 6, 4, 5, 7)

for i in t1:
    print('i = {} - Valor da tupla'.format(i))
    i = 2 - Valor de tupla
    i = 6 - Valor de tupla
    i = 4 - Valor de tupla
    i = 5 - Valor de tupla
    i = 7 - Valor de tupla
    Fim
```

## Estrutura da Tupla

Tuplas são variáveis compostas imutáveis

```
tupla = ('var1', 2, True)
tupla[0]  tupla[0:2]  tupla[1:]
```

## Funções

<code>len(t1)</code>	→ qtde de termos
<code>sorted(t1)</code>	→ ordenar
<code>t1.count(valor)</code>	→ contar a qtde de "valor"
<code>t1.index(valor)</code>	→ contar a "índice" de "valor"
<code>t1.index(valor, pos)</code>	→ contar "valor" a partir da "pos"
<code>t1 + t2</code>	→ concatenar tuplas

## Tupla dentro de for

```
for i in t1:
    print('loop:')
```

## Estrutura das Listas

Para declarar seus valores, basta colocar cada um separado por vírgula dentro de colchetes:

```
Pos   [6]
var  = ['Lanche', 'Batata', 'Refrri']
      [4]
      [2]
```

**Tuplas:** tupla = ('Lanche', 'Batata', 'Refrri')

**Listas:** lista = ['Lanche', 'Batata', 'Refrri']



<h2>Acessando os valores</h2> <ul style="list-style-type: none"> <li>❑ Acesso os valores dentro de um for</li> </ul> <pre>for c in sanduiche:     print(f'c = {c}, c[0] = {c[0]}, c[1] = {c[1]}')     c = ['laranja', '10.9'], c[0] = laranja, c[1] = 10.9     c = ['batata', 5.5], c[0] = batata, c[1] = 5.5     c = ['refri', 3.5], c[0] = refri, c[1] = 3.5</pre>	<h2>Dicionários</h2> <p>Dicionários são listas com índices textuais</p> <pre>dic = {'chave1': 'valor',        'chave2': 'Valor2',        'chave3': 'valors'}</pre>	<h2>Manipulando os dados</h2> <ul style="list-style-type: none"> <li>❑ Acessando o dicionário:</li> </ul> <pre>dicio = {'nome': 'Evelton',          'idade': 28,          'cidade': 'São José'}</pre> <ul style="list-style-type: none"> <li>❑ Completando o dicionário</li> </ul> <pre>print(dicio) {'nome': 'Thiago', 'idade': 29, 'cidade': 'São José'}</pre> <ul style="list-style-type: none"> <li>❑ Valores dicionario.values()</li> </ul> <pre>print(dicio.values()) dict_values(['Evelton', 29, 'São José'])</pre> <ul style="list-style-type: none"> <li>❑ Chaves dicionario.keys()</li> </ul> <pre>print(dicio.keys()) dict_keys(['nome', 'idade', 'cidade'])</pre>	<h2>Acessando os dados</h2> <ul style="list-style-type: none"> <li>❑ For dentro de For: Primeiro acessa o mais externo para depois o mais interno.</li> </ul> <pre>pessoas = [{'nome': 'Evelton', 'idade': 28, 'cidade': 'São José'},  {'nome': 'Thiago', 'idade': 34, 'cidade': 'São Paulo'},  {'nome': 'André', 'idade': 22, 'cidade': 'Lorena'}]</pre> <ul style="list-style-type: none"> <li>❑ For dentro de For: Primeiro acessa o mais externo para depois o mais interno.</li> </ul> <pre>for p in pessoas:     for k, v in p.items():         print(f'{k}: {v}')</pre>
--	--	---	--

## Acessando os valores

- ↳ Acesso as listas:
 

```
cardapio[0] == ['Lanche', 10.9]
cardapio[1] == ['Batata', 5.5]
cardapio[2] == ['Refrí', 3.9]
```
- ↳ Acesso aos valores da lista:
 

```
cardapio[0][1] == 10.9
cardapio[1][0] == 'Batata'
cardapio[2][0] == 'Refrí'
```

## Módulo 7: Dicionário

**1**
**2**
**3**
**4**

O que são dicionários
Funções com dicionários
Dicionários dentro de loops

## Manipulando os dados

- ↳ Acessando os valores:
 

```
dicio = []
dicio = {'nome': 'Everton',
        'idade': 29,
        'cidade': 'Sao José'}
```
- ↳ Modificando os valores:
 

```
dicio['nome'] = 'Thiago'
```
- ↳ Criando nova chave:
 

```
dicio['pesso'] = 65
```

## Dicionários dentro de listas

**1**
**2**
**3**

Dicionários dentro de listas
Funções com dicionários
Dicionários dentro de loops

## Declarando

### Listas compostas

Declarando com append e input do usuário:

```
cordapio = []
comida = []
for c in range(0,3):
    comida.append(int(input('Nome da Comida: ')))
    comida.append(float(input('Preço da Comida: ')))
    cardapio.append(comida[:])
    comida.clear() #limpando a memoria da variável
print(cardapio)
```

## Mão na Massa

Let's CODE



## Estrutura dos Dicionários

Existem 3 pontos importantes na estrutura de um dicionário:

```
nomes: {
    'Evertton': {
        'idade': 29,
        'cidade': 'São José',
        'values': [
            'Evertton',
            '29',
            'São José'
        ]
    }
}
```

keys:

```
dicio = { 'nome': 'Evertton', 'idade': 29, 'cidade': 'São José' }
```

values:

```
for v in dicio.values():
    print(f'{v}')
```

items:

```
for k, v in dicio.items():
    print(f'{k}: {v}')
```

Como dicionários são muito similares a listas, também podemos passar por eles dentro de for:

```
for v in dicio.values():
    print(f'{v}')
```

```
for k in dicio.keys():
    print(f'{k}')
```

```
for k, v in dicio.items():
    print(f'{k}: {v}')
```

## Dicionários dentro de for

values: Evertton  
Values: 29  
Keys: nome  
Keys: idade

keys:  
values:  
items:

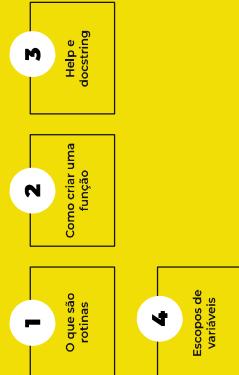
```
for v in dicio.values():
    print(f'{v}')
```

```
for k in dicio.keys():
    print(f'{k}')
```

```
for k, v in dicio.items():
    print(f'{k}: {v}')
```

<h2>Declarando</h2>	<p><b>Declarando com append:</b></p> <pre>[ ] Declarado = [] cardapio = [] comida = [] comida.append('Lanche') comida.append('Ovo') cardapio.append(comida[:])</pre>
<h2>Listas compostas</h2>	<p><b>Estrutura Listas Compostas</b></p> <pre>[ ] lista = [ 'v1', 'v2', 'v3' ] lista[0] == 'v1' lista[1] == 'v2' lista[2] == 'v3'  [ ] lista[0] = 'v0' lista[1] = 'v1' lista[2] = 'v2'  [ ] lista[0] == 'v0' lista[1] == 'v1' lista[2] == 'v2'</pre> <p><b>for c in lista:</b></p> <pre>[ ] for c in range(1, f):     1.append([input[1][0]: ])</pre> <p><b>1.append([input[:1][0]: ])</b></p> <p><b>composta.append([1[:1]: ])</b></p> <p><b>composta.append([1[:1]: ])</b></p> <p><b>1.clear()</b></p> <p><b>for c in composta:</b></p> <pre>[ ] for c in composta:     c[0] == 'v0', 'v1' &amp;gt; c[1] == 'v1', 'v2' ...</pre> <p><b>var.clear() → limpa a memoria da variável</b></p>
<h2>Funções</h2>	<p><b>Dicionários vs Listas</b></p> <p>Dicionários são estruturas de dados, semelhantes a listas, que as posições ao invés de serem numéricas são chaves.</p> <pre>[ ] Chaves = {     nome: 'Eveton',     idade: 29,     Pos: [0] } dicio = [     {'nome': 'Eveton', 'idade': 29, 'Pos': [0]},     {'nome': 'São José', 'idade': 29, 'Pos': [1]},     {'nome': 'São José', 'idade': 29, 'Pos': [2]} ]</pre> <p><b>Lista:</b></p> <pre>[ ] lista = ['Eveton', 29, 'São José']</pre> <p><b>Dicio:</b></p> <pre>[ ] dicio = {     'nome': 'Eveton',     'idade': 29,     'Pos': [0] }</pre>
<h2>Manipulando os dados</h2>	<p><b>Acessando o dicionário:</b></p> <pre>[ ] items_dicionario.items() print(dicio.items()) dicio.items() → {'Eveton': 29, 'São José': 29}</pre> <p><b>Deletando os valores:</b></p> <pre>[ ] del dicio['idade'] print(dicio) → {'nome': 'Eveton', 'idade': 29}</pre>

## Módulo 8: Funções



### Como criar uma função

```
def função():
    print('Função')
    função()
```

Toda função vem no começo do programa e é declarada por `def`. Durante o programa principal, toda vez que você precisa usar essa função você "chama" ela e executa essa subrotina.  
Imagina ter que codar o print toda vez?  
Imagina ter que criar um for toda vez para contar a quantidade de posições em uma lista? [len]

### Algumas funções

2 Funções com parâmetro opcionais:  
Faça uma função some a + b:  
# Função 2:  
def soma(a=0, b=0):  
 print(a + b)

# Programa Principal  
soma(2, 3) → 5  
...  
soma(3, 4) → 7  
...  
soma(-2) → -2

### Help e docstring

E como fazemos com nossas próprias funções? Nós mesmos podemos escrever este "help".  
# Função 2:  
def soma(a=0, b=0):  
 """  
 Função soma: soma 2 números  
 a: parâmetro opcional de entrada  
 b: parâmetro opcional de entrada  
 retorno: s  
 """  
 s = a + b  
 return s

## Mão na Massa

### Let's CODE



### Rotina na programação

Rotina em programação é tudo que é feito várias vezes:

- ❑ No mesmo programa

- ❑ Programas diferentes

### Funções Nativas

As funções nativas do Python nada mais é do que rotinas criadas por outros desenvolvedores que são utilizadas por milhares de outros desenvolvedores:  
❑ `print()`, `range()`, `int()`, `float()`,  
❑ `bibliotecas`

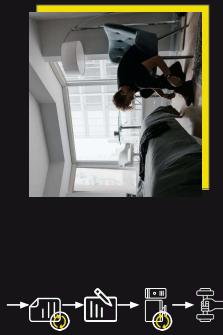
## Estrutura dos Dicionários

Dicionários são listas com índices sextais

```
dicionario = {}  
dicionario = { 'key1': 'value', 'key2': 'value' }  
dicionario[ 'key1' ] == 'value'  
dicionario.items() → valores  
dicionario.keys() → chaves  
dicionario[11][pos] → deleta item da "posição"  
  
lista = []  
lista.append(item)  
for k, v in dicionario.items():  
    print(f'{k}: {v}')  
  
for d in range(ini, fin):  
    dicionario[ 'valor1' ] = input('Valor1: ')  
    dicionario[ 'Valor2' ] = input('Valor2: ')  
    lista.append(dicionario.copy())
```

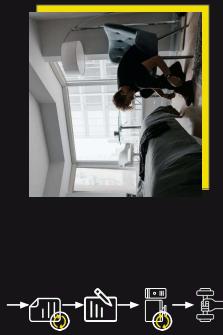
### Lista dentro de for

Rotinas são tarefas repetitivas, que executamos várias vezes, sempre da mesma maneira.



### Rotinas

Rotinas são tarefas repetitivas, que executamos várias vezes, sempre da mesma maneira.



### Funções

Funções são rotinas, ações que são executadas várias vezes nos programas.



### Algumas funções

3 Funções sem parâmetro de entrada:  
Faça uma função que imprima 'Olá Mundo'  
# Função 1:  
def ola():  
 print('Olá Mundo')

# Programa Principal  
ola() → Olá Mundo  
...  
ola() → Olá Mundo  
...  
ola() → Olá Mundo

### Algumas funções

O help do Python nos ajuda a mostrar quais são os parâmetros principais, opcionais, retornos e funcionalidades de qualquer função:  
help(print)  
Help on built-in function print in module builtins:  
print(...)  
 Prints the values to a stream, or to sys.stderr by default.  
 Optional keyword arguments:  
 file: a file-like object (stream); default is the current sys.stdout;  
 sep: string separator between values; default is a space;  
 end: string appended after the last value; default is a newline;  
 flush: whether to forcibly flush the stream;

Funções podem ou não ter parâmetros de entrada e saída.



### Estrutura das funções

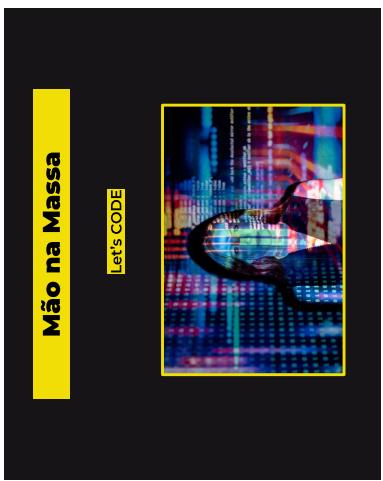
4 Funções com saída (retorno):  
Faça uma função some a + b:  
# Função 2:  
def soma(a=0, b=0):  
 s = a + b  
 return s

# Programa Principal  
s = soma(2, 3) → 5  
...  
s = soma(3, 4) → 7  
...  
s = soma(-2) → -2

### Algumas funções

Funções com saída (retorno):  
Faça uma função some a + b:  
# Função 2:  
def soma(a=0, b=0):  
 s = a + b  
 return s

# Programa Principal  
s = soma(2, 3) → 5  
...  
s = soma(3, 4) → 7  
...  
s = soma(-2) → -2



## Funções

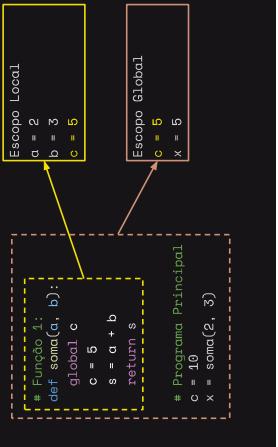
Funções são rotinas, ações que são executadas várias vezes nos programas.

```
def func(a):  
    """  
    docstring  
    """  
    print('Função')  
    func(b)  
    func(c)  
    return var  
func()
```

### Help e docstring

```
help(func)
```

## Escopo de variável



## Escopo de variável

