



Bem-vindas ao curso iniciante  
de Python



# O que é PyLadies?

Além de um grupo de mulheres incríveis?



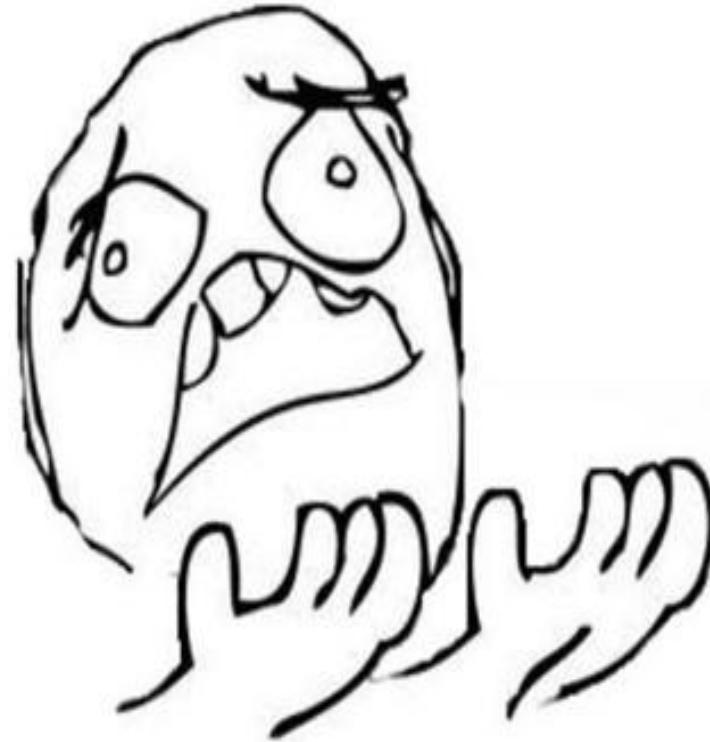
PyLadies é um grupo internacional de mentoria com foco em ajudar mais mulheres e outras minorias a tornarem-se participantes ativas e líderes da comunidade Python





pyladies  
FLORIPA

# MAS POR QUE



## NÃO HOMEM CIS??



# Porque...

**7%**  
da área de T.I é formada  
por mulheres  
(IBGE)

Em 120 anos a USP não  
formou nem **10**  
mulheres negras na área  
(USP)

**3%**  
de fundadores de startups  
tecnológicas são mulheres  
(Google Diversity Report)

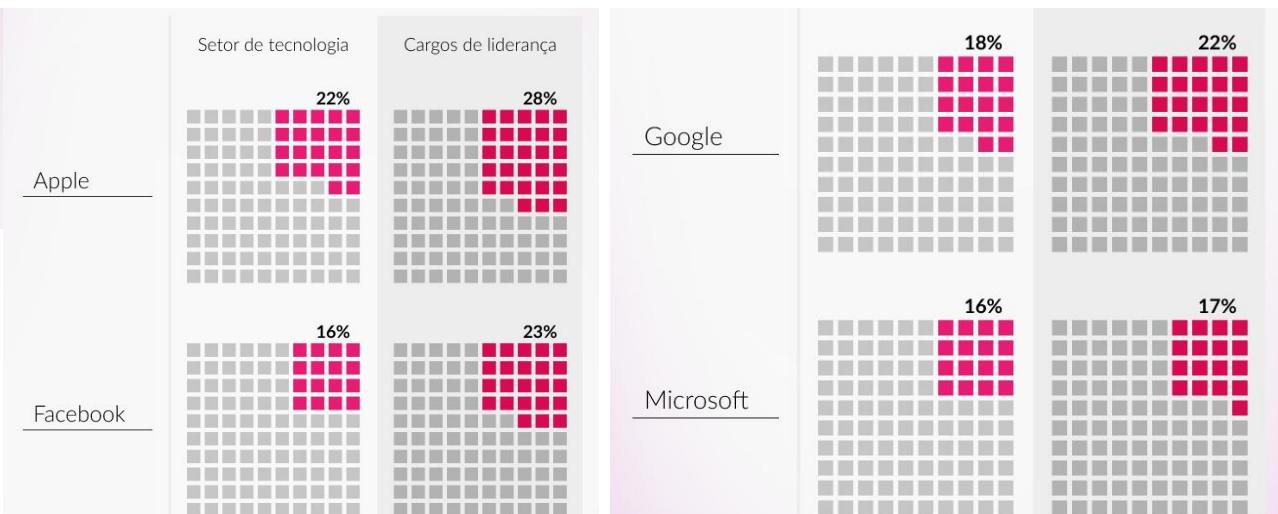
**0.4%**  
calouras pretendem graduar  
em Ciências da Computação  
(Google Diversity Report)

## O XX DA QUESTÃO: AS MULHERES NAS EMPRESAS DE TECNOLOGIA

Em média, 30% dos funcionários das quatro principais empresas de tecnologia do mundo (Apple, Facebook, Google e Microsoft) são mulheres. O número parece baixo, mas espere até você ver a parcela de mulheres trabalhando diretamente com tecnologia ou em cargos de liderança nessas companhias.

**Fonte:** Exame

De pessoas trans nem tem  
estatística :(





# Esse ambiente gera frases como...

“Você veio no evento com seu namorado?”

“Você é web designer?”

“Gostamos de mulheres... Elas embelezam o ambiente”

“Ah, você mudou de área? Você tem um namorado que é da área?”



pyladies  
FLORIPA

Isso sem falar de....

~vários homens explicam~





# MAS TECNOLOGIA É COISA DE MULHER...

## Ada Lovelace

A primeira programadora da história

## Grace Hopper

Criou o primeiro sistema de código de programação, que deu origem ao COBOL

## Dorothy Vaughan

Primeira especialista em Fortran, linguagem que lida com cálculos precisos

## Sheryl Sandberg

Chefe operacional do Facebook





# Minas da tecnologia



Ada Lovelace



Grace Hopper



Naomi Ceder



Radia Perlman



Dorothy Vaughan



Ruchi Sanghvi



Bora falar sobre a parte técnica !



# Antes, um pedido!

**Não deixem o vocabulário assustar vocês!**

Vamos acabar usando vários termos e expressões novas (algumas em inglês), isso pode fazer doer o cérebro.

**Façam perguntas!!**



# O que é algoritmo

Um algoritmo é uma sequência de passos.

Por exemplo:

- fazer compra online
- tomar banho
- lavar o carro
- pintar um quadro



**Explique os passos que eu  
devo realizar para ir até o fim  
da sala pegar um copo de café**



# Mas o que isso tem a ver com programação?

Se prepare para a polêmica....



# Computadores são burros





Mas como assim??

Os computadores não sabem o que devem fazer.

Precisamos **pensar pelo computador** e dizer os passos que queremos que ele dê para fazer o que queremos que ele faça.

**Basicamente, temos que ensinar ele.**



## Mas como ensinamos eles?

Obviamente, nós e o computador não falamos a mesma linguagem, então precisamos de um tradutor: uma linguagem de programação.

E para nos comunicarmos com a linguagem, utilizamos algoritmos.

Estes algoritmos podem ser executados por um humano ou um computador. Logo, programação é a arte de fazer com que o computador execute uma **sequência de instruções**.



# Mas por que Python?

Senta que lá vem textão...



Python é uma linguagem de programação com código aberto, de alto nível  
(mais fácil para a pessoa que está desenvolvendo entender).

Todas as empresas que vocês possam pensar, ligadas a tecnologia, utilizam esta  
linguagem para desenvolver algo.



Além de ter uma comunidade linda <3

# A COMUNIDADE





# O criador





E acima de tudo...



pyladies  
FLORIPA

# A simplicidade comparado a outras linguagens

## Java

```
1 public class Hello
2 {
3     public static void main(String args[]) {
4         java.util.Scanner s = new java.util.Scanner(System.in);
5         System.out.print("Digite seu nome:");
6         String nome = s.nextLine();
7         System.out.println("Olá, " + nome);
8     }
9 }
```



pyladies  
FLORIPA

# A simplicidade comparado a outras linguagens

Java

```
1 public class Hello
2 {
3     public static void main(String args[]) {
4         java.util.Scanner s = new java.util.Scanner(System.in);
5         System.out.print("Digite seu nome:");
6         String nome = s.nextLine();
7         System.out.println("Olá, " + nome);
8     }
9 }
```

C

```
1 #include <stdio.h>
2 int main()
3 {
4     char nome[200];
5     printf("Digite seu nome:");
6     scanf("%s", nome);
7     printf("Olá, %s\n", nome);
8 }
9 }
```



# A simplicidade comparado a outras linguagens

## Java

```
1 public class Hello
2 {
3     public static void main(String args[]) {
4         java.util.Scanner s = new java.util.Scanner(System.in);
5         System.out.print("Digite seu nome:");
6         String nome = s.nextLine();
7         System.out.println("Olá, " + nome);
8     }
9 }
```

## C

```
1 #include <stdio.h>
2 int main()
3 {
4     char nome[200];
5     printf("Digite seu nome:");
6     scanf("%s", nome);
7     printf("Olá, %s\n", nome);
8     return 0;
9 }
```

## Pascal

```
1 program HelloWorld(output);
2 var
3     nome: string;
4 begin
5     writeln('Digite seu nome:');
6     read(nome);
7     writeln('Olá, ', nome);
8 end.
```



pyladies  
FLORIPA

# A simplicidade comparado a outras linguagens

## Java

```
1 public class Hello
2 {
3     public static void main(String args[]) {
4         java.util.Scanner s = new java.util.Scanner(System.in);
5         System.out.print("Digite seu nome:");
6         String nome = s.nextLine();
7         System.out.println("Olá, " + nome);
8     }
9 }
```

## C

```
1 #include <stdio.h>
2 int main()
3 {
4     char nome[200];
5     printf("Digite seu nome:");
6     scanf("%s", nome);
7     printf("Olá, %s\n", nome);
8     return 0;
9 }
```

## Pascal

```
1 program HelloWorld(output);
2 var
3     nome: string;
4 begin
5     writeln('Digite seu nome:');
6     read(nome);
7     writeln('Olá, ', nome);
8 end.
```

# Python

```
1 nome = input('Digite seu nome:')
2 print ('Olá,', nome)
```



Ok... Esse tal Python parece  
ótimo, bora instalar?



# A instalação...

Os slides terão passo a passo via internet, porém sabemos que a internet nem sempre colabora, então passaremos pen drives com o instalador



# Primeiro: entre no site da Python Software Foundation e faça download da versão 3.6.1

The screenshot shows the Python Software Foundation website at https://www.python.org. The top navigation bar includes links for Python, PSF, Docs, PyPI, and Jobs. The main header features the Python logo and a search bar. Below the header, a navigation menu offers links to About, Downloads, Documentation, Community, Success Stories, and News. The Downloads section is currently active, displaying a sidebar with Python code examples and a list of download options: All releases, Source code, Windows, Mac OS X, Other Platforms, License, and Alternative Implementations. A large callout box highlights the "Download for Windows" section, which contains buttons for "Python 3.6.1" and "Python 2.7.13". A note states: "Note that Python 3.5+ cannot be used on Windows XP or earlier." Below this, it says: "Not the OS you are looking for? Python can be used on many operating systems and environments." and "View the full list of downloads."

# Python 3: List comprehension example

```
>>> fruits = ['apple', 'banana', 'orange']
>>> loud_fruits = [fruit.upper() for fruit in fruits]
>>> print(loud_fruits)
['BANANA', 'APPLE', 'ORANGE']

# List and the enumerate function
>>> list(enumerate([(0, 'Banana'), (1, 'Apple'), (2, 'Orange')]))
[(0, 'Banana'), (1, 'Apple'), (2, 'Orange')]
```

All releases

Source code

Windows

Mac OS X

Other Platforms

License

Alternative Implementations

**Download for Windows**

Python 3.6.1    Python 2.7.13

Note that Python 3.5+ cannot be used on Windows XP or earlier.

Not the OS you are looking for? Python can be used on many operating systems and environments.

[View the full list of downloads.](#)

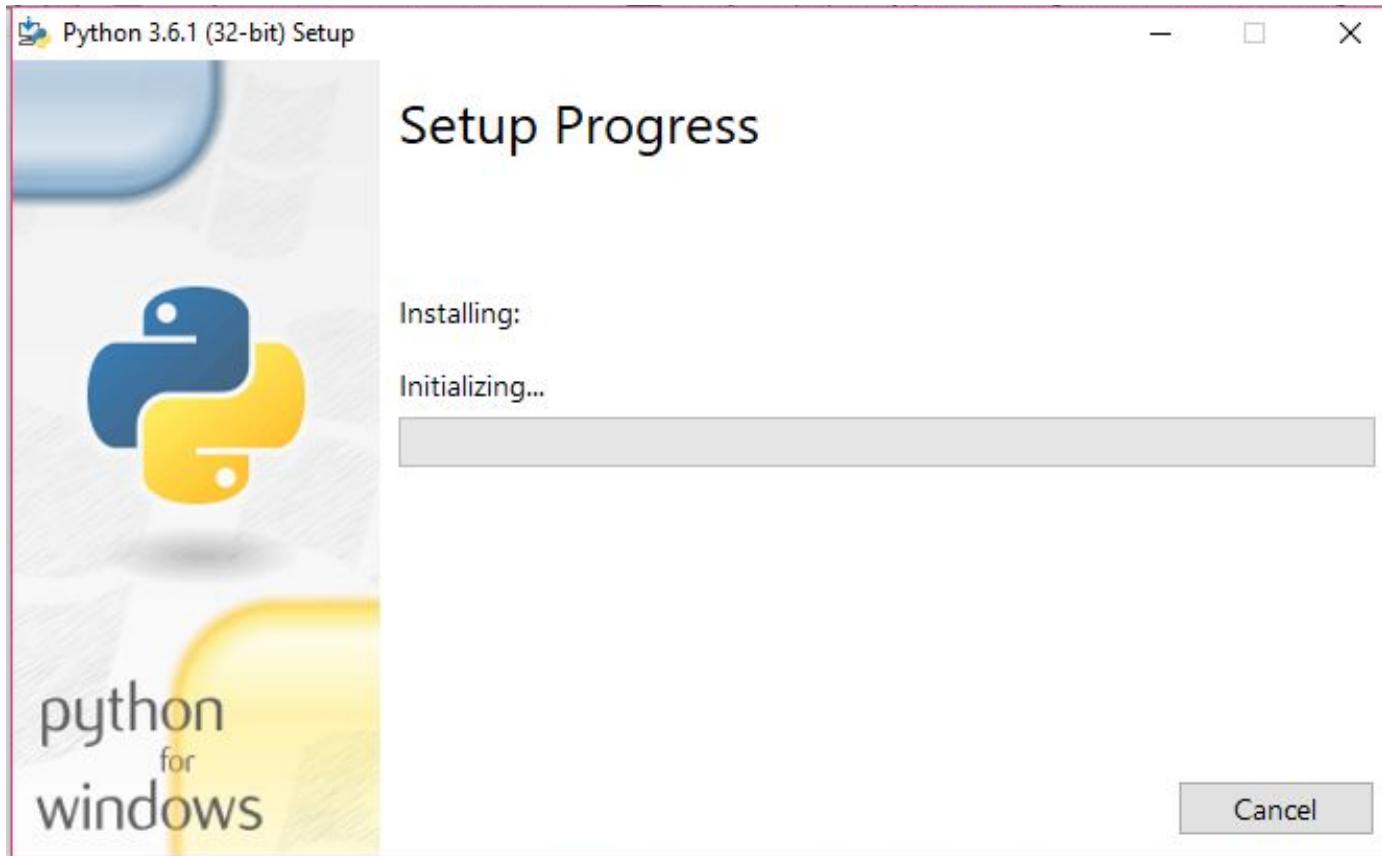


Segundo: Clique duas vezes no instalador, selecione "add Python 3.6 to PATH" e clique em "Install Now"





# Agora, espere...





# Pode rolar isso... Mas é só clicar no Disable path





BORA ABRIR A TELINHA  
PRETA DE HACKER



# Abra o python



UHUUL, VOCÊS ESTÃO NA  
TELINHA PRETA HACKER



## "Printando" na tela

Bom... Essas setas "">>>>" é a indicação que ali vai ser digitado um "comando", que não é nada mais que a ação que você quer que o **Python realize.** (Desculpa essa frase ruim, não achamos jeito melhor de falar)

O primeiro comando que você irá aprender é o "print". Um comando que é responsável por mostrar/escrever/"printar" o que você escrever entre "“”" na tela.

O resultado deste comando é mostrado logo abaixo.

```
>>> print("Essa vai ser a frase que será mostrada na tela")
```



Agora, nós, programadoras,  
temos um rito de passagem:

Mostre na tela um "Olá mundo"



# Saindo dessa telinha como uma hacker

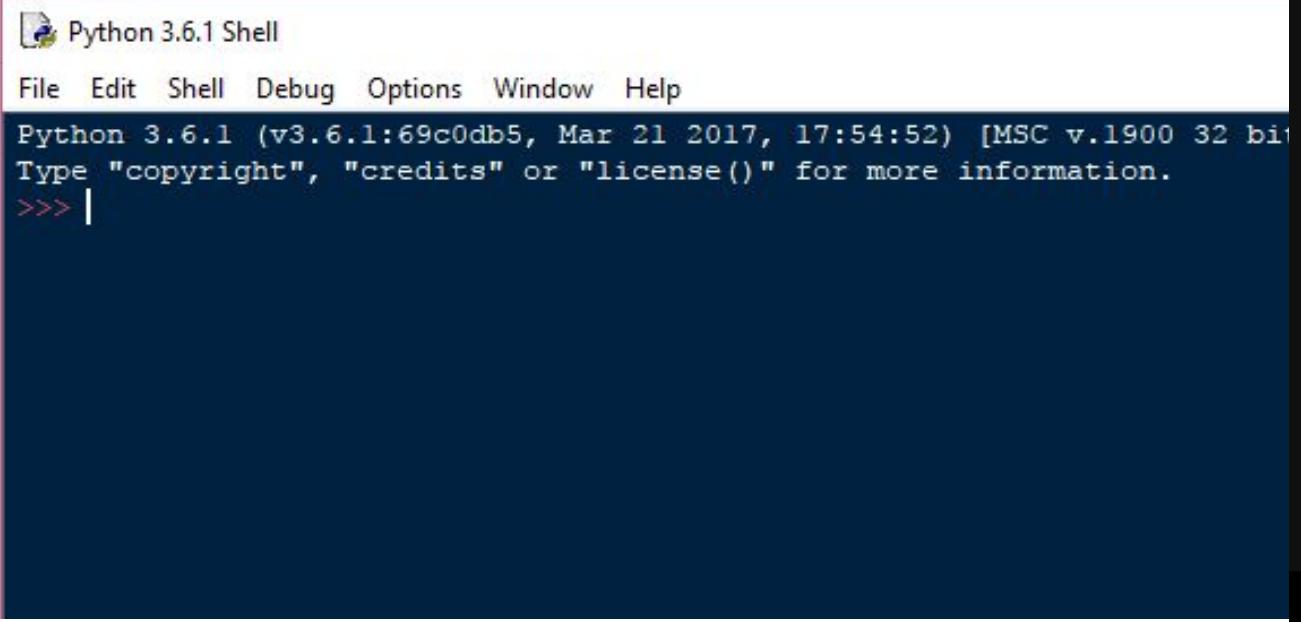
Digite "exit()" na linha de comando



Tá, mas não salvou, como eu  
salvo para mostrar para o/a  
*crush*?

# Editores de código

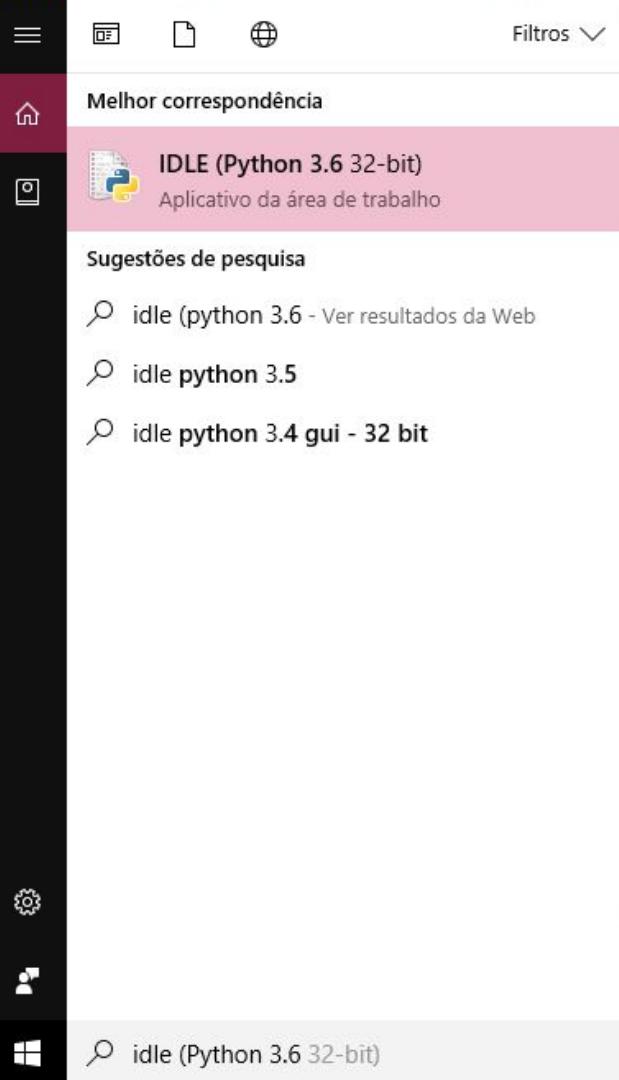
Procure por "IDLE"



Python 3.6.1 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit]
Type "copyright", "credits" or "license()" for more information.
>>> |
```



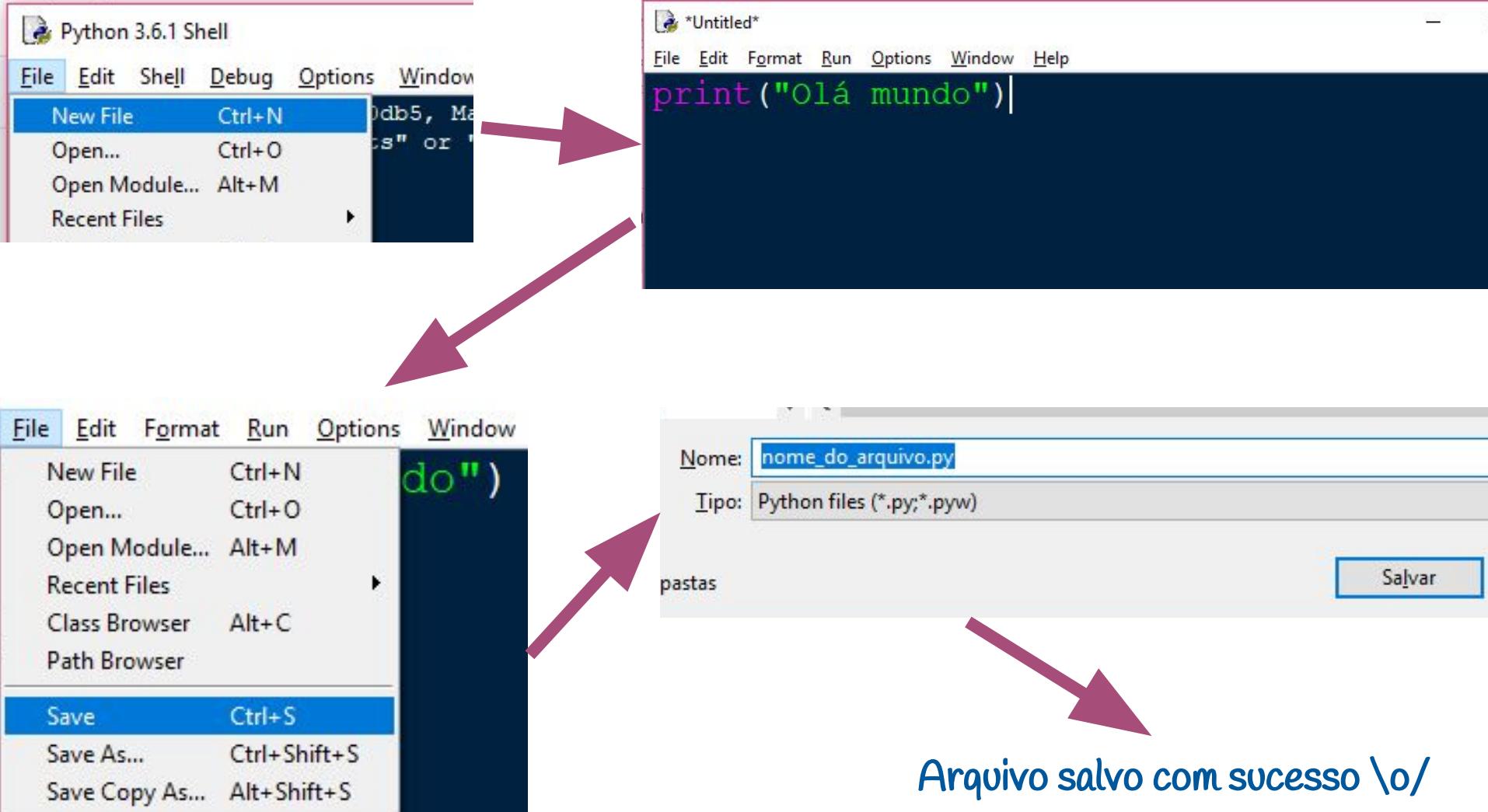


# Editores de texto

Clique em:

- "File" > "New File" (ou ctrl + n)
- Coloque seu "print" no arquivo que você acabou de abrir
- "File" > "Save" (ou ctrl + s)
- Escolha o nome do arquivo

**E... Pronto! Agora tudo o que você escreveu estará salvo!**





Chega de enrolar...  
bora aprender a codar o/



# Variáveis

Variáveis são como **gavetas nomeadas** para o computador, onde ele **guarda as informações** que você dá a ele.

E como é “nomeada”, você não precisa saber onde está a gaveta, só precisa saber o **nome** dela.

Quando você precisa buscar uma informação, você diz o nome da gaveta e o computador vai lá e “abre” esta gaveta, dando a informação que tá dentro dela.



# Variáveis

Mas como tudo na vida... Tem regras!

- Podem ser usados **números, letras ou \_**
- Nunca devem começar com um algarismo

E tem **palavras reservadas** que o Python usa para entender o que queremos que ele faça, como aquele "print" que demos.

Para conseguirmos nos comunicar com o Python, **não podemos dizer que uma dessas palavras reservas é uma gaveta** (variável), se não, em vez do computador "printar", ele vai **abrir** uma gaveta (mostra o valor da variável).



Mas como isso fica no código?



*>>> < nome > = < valor >*

TUDO O QUE TIVER ENTRE "<>" DEVE SER SUBSTITUÍDO PELO O QUE VOCÊS QUEREM



# Tipos de variáveis simples

Para saber como organizar essas variáveis, o Python tem tipos de variáveis

- **Numéricos:** Armazena números. São: inteiros (`int`), de ponto flutuante (`float`)
- **Letras:** Armazena caracteres (qualquer um do teclado). São strings (`string`)
- **Lógicos:** Armazena verdadeiro ou falso. São: booleanos (`bool`)

# Numéricos

- **Tipo inteiro (int):** números inteiros, sem ponto da casa decimal.

```
>>> a = 2  
>>> b = 4  
>>> c = 0
```

- **Tipo ponto flutuante (float):** números que aparece o ponto da casa decimal.

```
>>> d = 2.0  
>>> e = 4.5  
>>> f = 0.0
```



# Letras e Lógicos

- **String (str)**: comprehende todos os caracteres dentro de aspas (simples ou duplas)

```
>>> j = "2"  
>>> k = "2.0"  
>>> l = "12@#nikukyu"  
>>> m = "ççççççç"
```

- **Booleanos (bool)**: comprehende respostas do tipo True ou False (Verdadeiro ou Falso)

```
>>> n = True  
>>> o = False
```



Tá, como eu sei se o Python  
entendeu que a variável L é do  
tipo lógico e não uma string?



# Variáveis

Existe uma forma de pedir para o Python te dizer qual o tipo do valor:

**Sintaxe:**

```
type(<nome_da_variavel>)
```

**Exemplo:**

```
>>> type(m)
<type "bool">
```



# Exercícios \o/

- 1) Teste algumas variáveis definidas anteriormente.
- 2) Tente definir `m = true`

O que acontece? Por que dava certo quando era `True`?  
Tente `m = "true"` e verifique o tipo de `m`.  
Qual sua teoria para explicar o que aconteceu?

Perdeu alguma variável  
pelo caminho?

`a = 2`

`b = 4`

`c = 0`

`d = 2.0`

`e = 4.5`

`f = 0.0`

`g = "2"`

`h = "2.0"`

`i = "12@#nikukyu"`

`j = "§§§§§"`

`k = True`

`l = False`

Uma forma de transformar um tipo de variável em outro é usando os comandos:

- `int()`
- `float()`
- `bool()`
- `str()`

## Exemplos:

```
>>> a = 2
>>> type(a)
<class "int">

>>> b = float(a)
>>> type(b)
<class "float">

>>> c = bool(True)
>>> type(c)
<class "bool">

>>> d = str(a)
>>> d
"2"

>>> e = int(d)
>>> e
2
>>> type(e)
<class "int">
```



Mas o que podemos fazer com  
essas variáveis?



# Podemos fazer operações matemáticas:

Operadores numéricos básicos:

Adição: +

Subtração: -

Divisão: /

Multiplicação: \*

Potenciação: \*\*

Resto de uma divisão: %



## Exemplos:

```
>>> a = 2
>>> b = 3
>>> a + b
5
>>> a * b # multiplicação
6
>>> 10 % 8 # resto da divisão entre números inteiros
2
>>> 5 / 2 # quero que retorne um número “quebrado”
2.5
>>> 5 // 2 # quero que retorne um inteiro
2
>>> (a + b - 2 * a + a ** a) / 5
1
```

# Operadores lógicos

Utilizamos quando queremos ter respostas lógicas, isto é, que retornam **True** ou **False** (verdadeiro/falso):

**Maior que:** `>`

**Menor que:** `<`

**Maior ou igual a:** `>=`

**Menor ou igual a:** `<=`

**Idêntico:** `==`

**Diferente de:** `!=`

**Não:** `not`

**E:** `and` (exige que todas as condições sejam satisfeitas)

**Ou:** `or` (apenas uma das condições precisa ser satisfeita)



# Exemplos

```
>>> a = 2
```

```
>>> b = 3
```

```
>>> a > b
```

False

```
>>> b > a
```

True

```
>>> a == b
```

False

```
>>> a != b
```

True

```
>>> a > b and b > a
```

False

```
>>> a > b or b > a
```

True

```
>>> not (a != b) == False
```

True



# Exercícios \o/

Se  $a = 2$ ,  $b = 3$ ,  $c = 2.0$  e  $d = "2.0"$ , faça  
as operações a seguir:

```
>>> a + b  
>>> b ** a  
>>> a + c  
>>> a + d
```

E teste as condições:

```
>>> a == c  
>>> a <= b  
>>> a < b and b < c  
>>> a < b or b < c  
>>> a > c or a >= c  
>>> not(a != b and b <= (a**2)-1)
```



*Ok, então eu posso fazer essas coisas somente com números e booleanos?*

*Não, aí está a loucura, você também pode fazer essas coisas com strings!*



# Operações com strings

A concatenação é como se você “somasse” palavras.

**Sintaxe:**

“<string>” + “<string>”

**Exemplos:**

```
>>> "a" + "b"
```

“ab”

```
>>> "PyLadies" + " " + "Floripa"
```

“PyLadies Floripa”



# Operações com strings

Você também pode "multiplicar" palavras

Sintaxe:

“<string>” \* <inteiro>

Exemplos:

```
>>> “bla” * 3
```

“blablabla”

```
>>> “k” * 6
```

“kkkkkk”



Legal! O que mais posso fazer  
com strings?



# Mais coisas que você pode fazer com strings

**Uppercase:** todas as letras de uma string em maiúscula (pode fazer com uma variável tipo string também)

Sintaxe:

"<texto>".upper()

ou

variavel\_do\_tipo\_string.upper()

Exemplo:

```
>>> "string".upper()
```

```
"STRING"
```

```
>>> string = "Essa é uma string"
```

```
>>> string.upper()
```

```
>>> "ESSA É UMA STRING"
```



# Mais coisas que você pode fazer com strings

**Lowercase:** todas as letras de uma string em minúscula (pode fazer com uma variável tipo string também)

Sintaxe:

"<texto>".lower()

ou

variavel\_do\_tipo\_string.lower()

Exemplo:

```
>>> "string".lower()
```

```
"string"
```

```
>>> string = "Essa é uma string"
```

```
>>> string.lower()
```

```
>>> "essa é uma string"
```



# Mais coisas que você pode fazer com strings

**Capitalize:** transforma apenas a primeira em letra maiúscula (se for um número, ele não faz nada)

Sintaxe:

"<texto>".capitalize()

ou

variavel\_do\_tipo\_string.capitalize()

Exemplo:

```
>>> "string".capitalize()
```

```
"String"
```

```
>>> string = "essa é uma string"
```

```
>>> string.capitalize()
```

```
>>> "Essa é uma string"
```



# Mais coisas que você pode fazer com strings

**Title:** coloca a primeira letra de cada palavra em maiúscula

**Sintaxe:**

"<texto>".title()

ou

variavel\_do\_tipo\_string.title()

**Exemplo:**

```
>>> "string".title()  
"String"  
>>> string = "essa é uma string"  
>>> string.title()  
>>> "Essa É Uma String"
```



# Mais coisas que você pode fazer com strings

## Testando suas strings:

Começa com (`startswith`) ou Termina com (`endswith`): testa se um texto começa/termina com um elemento (é um teste lógico)

### Sintaxe:

`<string>.startswith("primeiras letras")`

ou

`<string>.endswith("últimas letras")`

### Exemplo:

```
>>> a = "Uma frase bem maneira"
```

```
>>> a.startswith("U")
```

True

```
>>> a.startswith("u")
```

False

```
>>> a.endswith("a")
```

True



# Mais coisas que você pode fazer com strings

Quer mudar algo que escreveu?

**Replace:** troca uma string por outra dentro de um texto.

Sintaxe:

```
<string>.replace("string que quero  
mudar", "nova string")
```

Exemplo:

```
>>> a = "Uma frase bem maneira"  
>>> a.replace("bem", "nada")  
"Uma frase nada maneira"
```



## Exercícios \o/

Defina a variável `a = "Workshop Pyladies as 9hrs da manhã :("` e com apenas 1 (um) comando sobre a variável:

- 1) Escreva WORKSHOP PYLADIES ÀS 9H DA MANHÃ :(
- 2) Escreva Workshop Pyladies Às 9H Da Manhã :(
- 3) Troque o :( por :)
- 4) Some a frase um "Mas pelo menos tem café :)"
- 5) Tire a palavra "Pyladies" da frase.



Mas só tem esses tipos de variáveis?

# Tipos de variáveis compostas

## Por que compostas?

R: Porque elas são compostas por mais de um valor "simples" (mais de um número, mais de uma string).

## Pode ser valores diferentes?

R: No Python sim, mas algumas linguagens não permitem isso.

## Quais são esses tipos?

R: Existem muitos tipos (lista, tupla, dict, etc...), porém vamos focar no principal:

**Listas:** permitem armazenar várias informações em uma mesma variável.



# Listas

Como dissemos anteriormente, as listas permitem que adicionemos várias informações em uma variável só, independente do tipo dessa informação (números, letras e booleano). Inclusive, você pode colocar uma lista dentro de uma lista.

## Sintaxe:

<variável> = [info1, info2, info3]

## Exemplos:

```
>>> a = ["String", 9, True]
```

```
>>> a
```

```
["String", 9, True]
```

```
>>> b = [[“Nome”, 99999], True]
```

```
>>> b
```

```
[[“Nome”, 99999], True]
```

```
>>> type(b)
```

```
<class "list">
```



Mas como eu resgato uma dessas informações?



# Como recuperar um valor de uma Lista

**Índice em uma lista:** número que indica a posição de cada caractere na string. O número será 0 (zero) para o primeiro caractere na string, 1 para o segundo, 2 para o terceiro, etc.

## Sintaxe:

<lista>[número]

["a", 2, True, [1], "Ladies"]

"a"	2	True	[1]	"Pyladies"
0	1	2	3	4

```
>>> a = ["a", 2, True, [1], "Pyladies"]
>>> a[0]
"a"
>>> a[2]
True
>>> a[3]
[1]
```



# Como recuperar partes de uma Lista

**Partes de uma lista:** retorna parte da lista, começando pelo primeiro índice e terminando no anterior ao segundo.

## Sintaxe:

<lista>[índice1:índice2]

**OBS:** Quando omitimos um índice, é mostrado o caractere do extremo correspondente

["a", 2, True, [1], "Pyladies"]

"a"	2	True	[1]	"Pyladies"
0	1	2	3	4

```
>>> a = ["a", 2, True, [1], "Pyladies"]
>>> a[0:2]
["a", 2]
>>> a[2:]
[True, [1], "Pyladies"]
>>> a[:3]
["a", 2, True]
```

# Como recuperar partes de uma Lista

**Incremento de fatia:** permite resgatar índices alternados, e até mesmo inverter uma lista.

## Sintaxe:

<lista>[índice1:índice2:<incremento>]

**OBS:** Quando colocamos um índice negativo no incremento, é invertida a ordem da lista.

["a", 2, True, [1], "Py", 1, 2]

"a"	1	True	[1]	"Py"	1	2
0	1	2	3	4	5	6

```
>>> a = ["a", 2, True, [1], "Py", 1, 2]
>>> a[::-2]
["a", True, "Py", 2]
>>> a[::-1]
[2, 1, "Py", [1], True, 2, "a"]
>>> a[5:0:-2]
[1, [1], 2]
```



# Tamanho da sua Lista

Para saber o tamanho da sua lista, basta pedir para o Python a partir da comando "len".

## Sintaxe:

```
len(<lista>)
```

## Exemplos:

```
>>> a = [1, 2, 3]
>>> len(a)
3
```



Beleza, eu criei minha lista, posso  
colocar mais coisas nela?



# Adicionando mais itens a minha lista

Existem duas formas de fazer isso:  
"concatenando" listas ou adicionando o item  
com a comando **append**.

## Sintaxe:

<lista>.append(<item para adicionar>)  
ou  
<lista1> + <lista2>

## Exemplos:

```
>>> a = [1,2]
>>> a.append(3)
>>> a
[1,2,3]
>>> b = [4, 5]
>>> a + b
[1,2,3,4,5]
```



Vauuu! Só consigo fazer isso com  
listas?

# Listas-De-Letras (vulgo strings)

“Pyladies”							
P	y	I	a	d	i	e	s
0	1	2	3	4	5	6	7

O Python, como a maioria das linguagens, entende que a string é uma lista de letras, logo, você pode aplicar alguns comandos das listas em strings \o/

```
>>> a = "Pyladies"
```

```
>>> a[0]
```

```
"P"
```

```
>>> a[:2]
```

```
"Py"
```

```
>>> a[0:5:2]
```

```
"pld"
```

```
>>> a[::-1]
```

```
"seidalyP"
```

```
>>> a[5:0:-2]
```

```
"iay"
```

```
>>> len(a)
```

8



# Exercícios \o/

Dada as variáveis:

```
>>> string = "Workshop"  
>>> lista = ["Talita", "Alana", "Monalisa"]
```

- 1) Qual o tamanho da **string**?
- 2) Qual o tamanho da **lista**?
- 3) Verifique se a string começa com "**W**"?
- 4) Verifique se a string começa com "**P**"?
- 5) Inverta a string e a lista.
- 6) Acrescente o nome de uma pessoa que você conheceu hoje.
- 7) Acrescente a string na lista.



Como eu "junto" essas variáveis  
em uma string?



# Transformando uma lista em uma string

**Join**: gruda os elementos de uma sequência de strings, usando uma string fornecida, que será usada para "dividir os itens da lista".

## Sintaxe:

"<string>".join(<lista>)

## Exemplos:

```
>>> a = ["banana", "limão", "maçã"]  
>>> ",".join(a)  
"banana,limão,maçã"
```



# Transformando outros tipos de variáveis em strings

Existem duas formas de fazer isso, com o comando `format`, ou "forçando" a transformação através da tipagem `str()`.

**Format**: Permite que você "una" a uma string quaisquer tipos de variáveis. Modifica todo "{}" pela variável passada. Você consegue unir mais de uma variável

## Sintaxe:

`"{}".format(<variável>)`

ou

`str(<variável>)`

## Exemplos:

```
>>> "{}".format(True)  
"True"
```

```
>>> "Um: {}".format(1)  
"Um: 1"
```

```
>>> "{} {}".format(True, 1)  
"True 1"
```

```
>>> str(True)
```

"True"

```
>>> str(1)
```

"1"

```
>>> str(True, 1)
```

**Erro??**



# Quer saber quais comandos você pode explorar?

Tem a comando "dir" do python, que lista todos os comandos válidos para a variável.

**Sintaxe:**

```
>>> dir(<variável>)
```

~ Exemplo muito grande para caber aqui, vamos fazer no python ~



Bom, já aprendemos a "printar" na tela variáveis, mas bora aprender a "pegar" o valor das variáveis pela tela?

# Pegando o valor das variáveis pela tela

Para pegarmos o valor das variáveis pela tela, no Python, usamos o comando "`input`". Existem duas formas de pegar o valor da variável: pegando sempre como string, ou "tipando" ela como um tipo de variável.

## Sintaxe:

```
<variável> = input(<"mensagem para o usuário entrar com o valor">)
```

ou

```
<variável> = <tipo da variável>(input(<"mensagem para o usuário entrar  
com o valor">))
```



# Exemplos

```
>>> input("Qual seu livro favorito?")
Qual seu livro favorito?
```

Mas o `input()` sempre devolve uma string (ele entende que o usuário digitou um texto). Se quero a entrada de um número, então preciso transformar a variável que foi lida como string em uma variável tipo numérico como no exemplo abaixo:

```
>>> int(input("Qual sua idade?"))
Qual sua idade?
```

Tentem colocar uma letra...



Vamos aprender mais coisas \o/



# Funções

**Funções** são pedaços de código que servem para executar um procedimento muitas vezes, evitando que você tenha que reescrevê-lo mais de uma vez.

Uma funcionalidade importante é o fato que, caso precise realizar alguma alteração ou correção, ela vai ser feita neste pedaço de código e não em diversas partes do código



# Funções

## Sintaxe:

Quando a função sempre faz a mesma coisa do mesmo jeito:

```
def <nome da função> ():  
    <pedaço do código que você quer executar>
```

OU

Quando a função faz coisas dependendo do valor dos **parâmetros**:

```
def <nome da função> (<parâmetro(s)>):  
    <pedaço do código que você quer executar>  
    return (caso essa função retorne algum valor)
```



# Funções

O comando `input()`, usado nos exemplos, é na verdade uma função nativa no Python.

Ela solicita uma informação do usuário e nos retornar o valor informado.

Agora vamos criar a nossa própria função :)



# Exemplos

**Exemplo 1:** Uma função que soma.

```
def soma(a, b):  
    return a + b
```

```
>>> print(soma(1, 2))  
3  
#ou  
>>> print(soma("PyLadies ", "Floripa"))  
PyLadies Floripa
```



# Exemplos

**Exemplo 2:** Uma função que multiplica.

```
def multiplica(n1, n2):  
    return n1 * n2
```

```
n1 = float(input("Informe o primeiro número: "))  
n2 = float(input("Informe o segundo número: "))  
print(multiplica(n1, n2))
```

**Resultado:**

```
>>>  
Informe o primeiro número: 6  
Informe o segundo número: 7  
42.0  
>>>
```



# Exercício \o/

Faça uma **função** para pedir ao usuário dois números e calcular a divisão entre eles.

Informe:

- o valor exato da divisão
- o valor inteiro
- o resto



MAIS COISAS, QUEREMOS  
APRENDER MAIS COISAS!!



# Condicionais para tomar decisões

Bom... Na nossa vida nem sempre andamos reto, certo? Pode ter lugares onde temos que tomar uma decisão: direita, esquerda, continuar em frente ou voltar.

Essas decisões são **condicionais** na programação, são ações que tomamos a partir de algum acontecimento.

MAS COMO ASSIM???



Bom, vamos tentar resolver um problema: *estamos com fome, o que devemos fazer?*



Primeiro, vamos na geladeira  
olhar se tem algo. Certo?



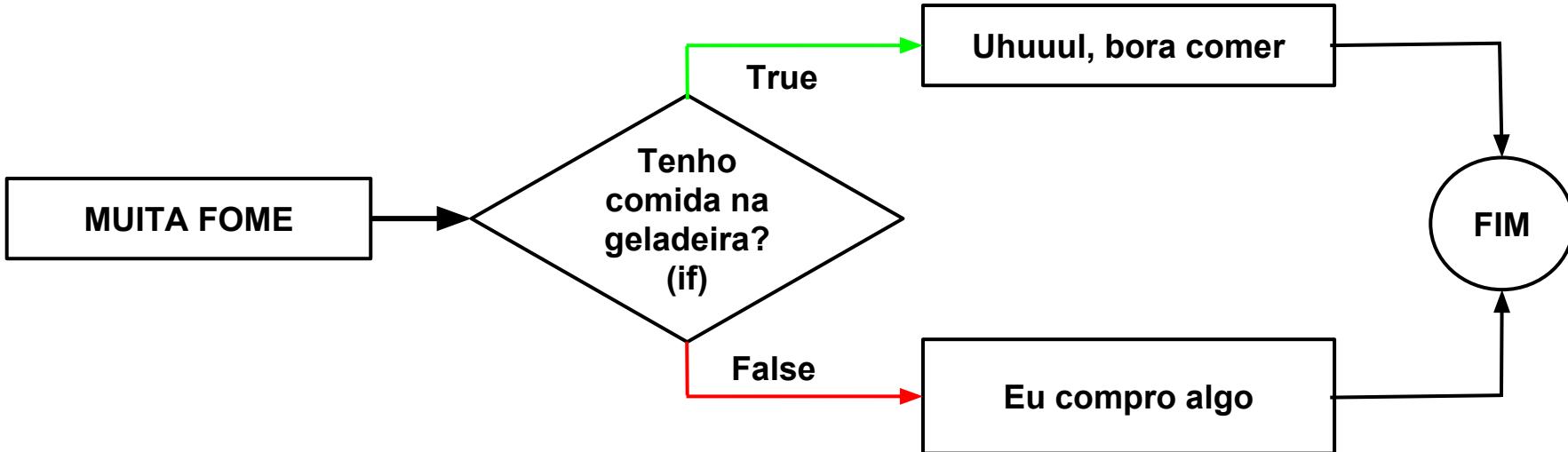
Se não tem nada na geladeira,  
o que faremos?



Bom, vamos comer algo "fora".

# Condicionais

Tudo o que fizemos, foi tomar decisões, isto é: você **condicionou**:  
**se (if)** eu tiver comida na geladeira, eu pego algo para comer.  
Caso contrário (**else**), eu ligo para a pizzaria e compro uma pizza.





Ok, entendemos.  
Agora, como codamos isso?



# Condisional

## Sintaxe:

**if** <condição dada por operador booleano>:

<o que tenho que fazer, caso a condição seja satisfeita>

**else:**

<o que tenho que fazer, caso a condição não seja satisfeita>



# Condisional

**Exemplo:**

```
comida_na_geladeira = input("Tem comida na geladeira?")
```

```
if comida_na_geladeira == "sim":
```

```
    print("Uhul, bora comer")
```

```
else:
```

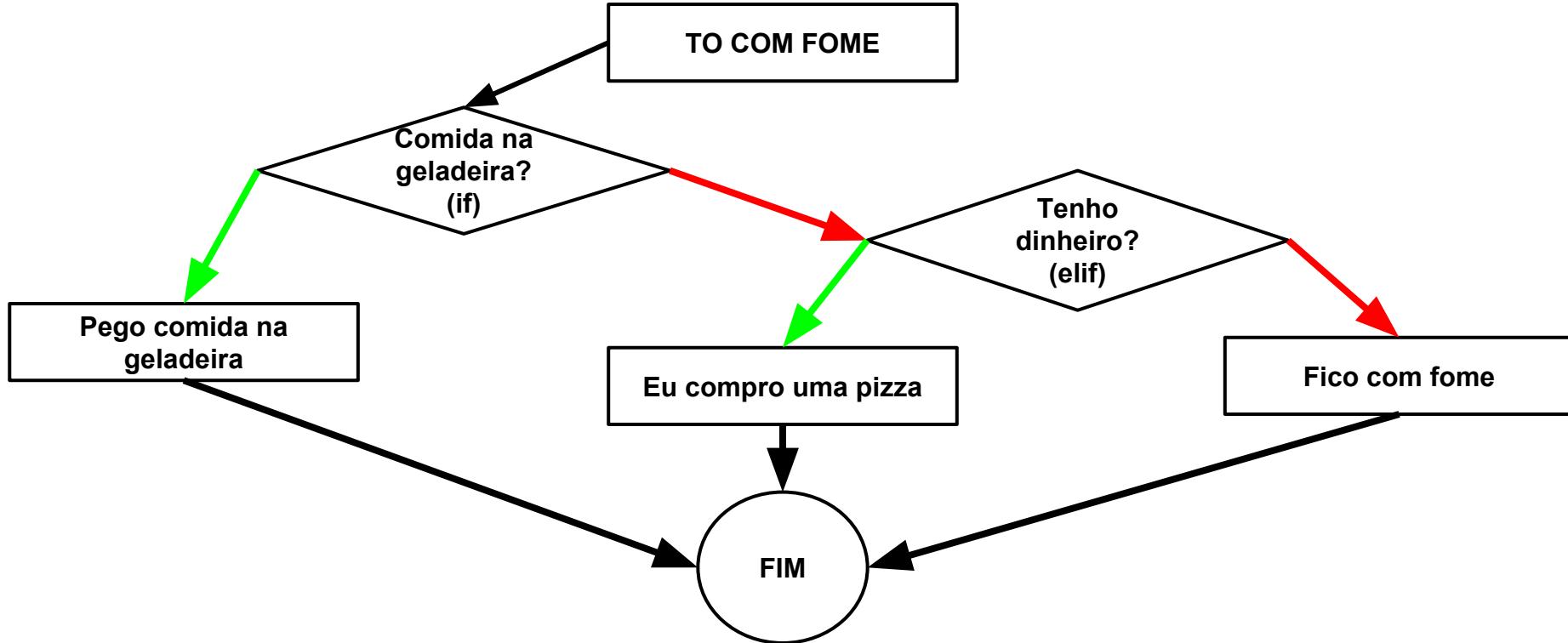
```
    print("Bora comprar pizza!")
```



Tá, mas e se for no fim do mês e  
não tiver dinheiro para comprar  
pizza?

# Condicionais

Se não tiver nada na geladeira, e se eu tiver dinheiro, eu compro uma pizza e fico feliz. Isto é: **elif** (senão se)



# Condicional

## Sintaxe:

**if** <condição dada por operador booleano>:

<o que tenho que fazer, caso a condição seja satisfeita>

*(caso a condição anterior não seja satisfeita, tenho mais uma condição para verificar)*

**elif** <condição dada por operador booleano>:

<o que tenho que fazer se a segunda condição for satisfeita>

**else:**

<o que tenho que fazer, caso nenhuma das condições acima sejam satisfeitas>

# Condisional

## Exemplo:

```
comida_na_geladeira = input("Tem comida na geladeira?")
tem_dinheiro = input("Tenho algum trocado?")

if comida_na_geladeira == "sim":
    print("Pego algo na geladeira para comer")
elif tem_dinheiro == "sim":
    print("Ligo para pizzaria")
else:
    print("Fico com fome :(")
```



## Exercício \o/

Crie um código para uma outra PyLady executar. Pergunte seu nome e a convide para tomar um chá, um café ou um suco. Dê uma resposta para cada opção citada (chá, café ou suco). Deixe a opção de ela não querer tomar nada; caso ela escolha esta opção escreva um "Tudo bem, respeito sua posição :)".

Tempo: 7 minutos.



Ainda estão comigo meninas?  
Tá quase terminando....

# Contadores e Acumuladores

**Contadores:** variáveis auxiliares que usamos para incrementar valores fixos, como por exemplo somar o valor 1 em uma variável. O valor pode ser qualquer valor positivo ou negativo.

**Acumuladores:** Contadores que incrementam valores diferentes.



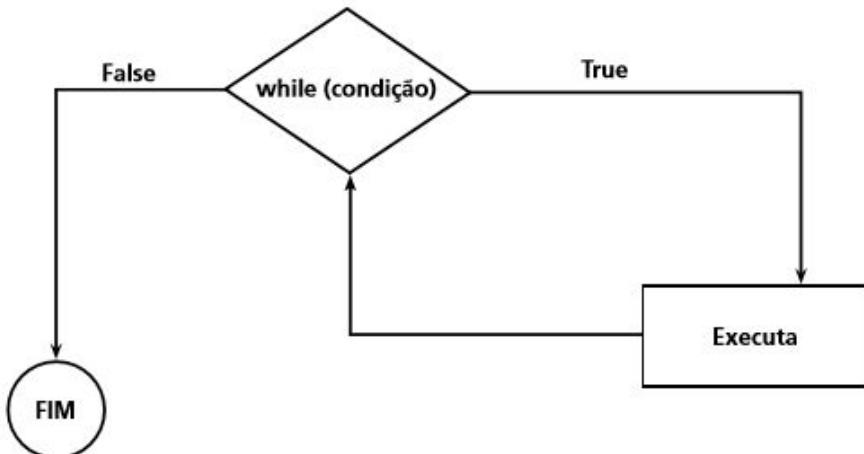
# Exemplo

Soma de 3 números dados pelo usuário.

```
>>> total = 0
>>> num = int(input("entre com um número: "))
>>> total = total + num
>>> num = int(input("entre com um número: "))
>>> total = total + num
>>> num = int(input("entre com um número: "))
>>> total = total + num
>>> print("O total é: {}".format(total))
```

## Loops (Repetições)

**While** é usado quando precisamos repetir uma ação algumas vezes ou fazer uma iteração até confirmar uma condição.



**Sintaxe:**

**while** <condição a ser verificada>:  
<comando que quero executar>



## While (Enquanto)

**Exemplo 1:** soma de 3 números dados pelo usuário.

**contador = 1**

**soma = 0**

**while contador <= 3:**

    num = int(input("entre com um número: "))

    soma = soma + num # acumulador: acumula um valor diferente cada vez que passa

**contador = contador + 1** # contador: somando um valor fixo

print(soma)



## While (Enquanto)

**Exemplo 2:** Criar um código onde o usuário entre com um número e obtenha a tabuada do mesmo.

```
num = int(input("Qual tabuada você quer calcular?"))
print("Tabuada do {}".format(num))
cont = 1
while cont <= 10:
    print("{} x {} = {}".format(cont, num, (cont*num)))
    cont = cont + 1
```



## While (Enquanto)

Às vezes quero interromper uma repetição no meio de um processo, dependendo do que o usuário digita ou outro motivo. Nestes casos, posso usar o **break**.

**Exemplo:** soma de números inteiros até ser digitado **zero**

```
soma = 0
```

```
while True:
```

```
    num = int(input("Digite o número: "))
```

```
    if num == 0:
```

```
        break
```

```
    soma = soma + num
```

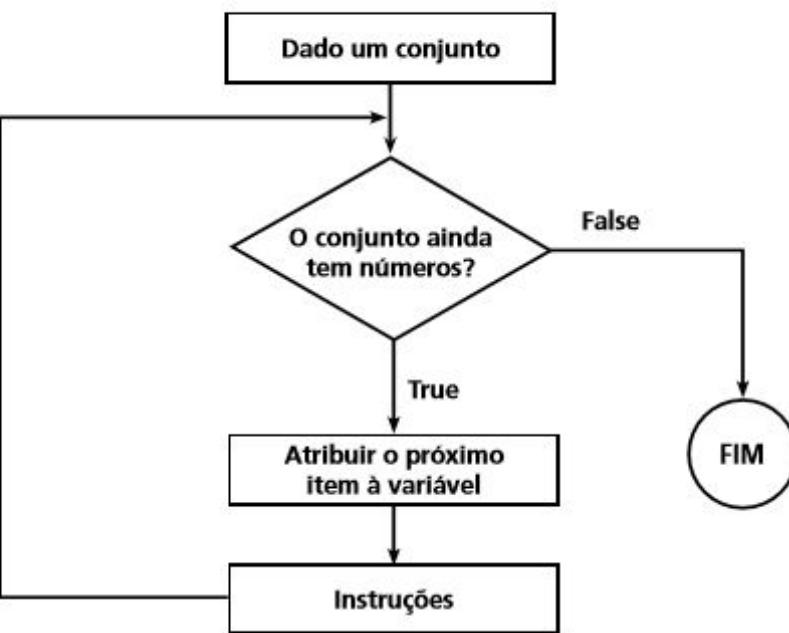
```
print("Soma: {}".format(soma))
```

**Resultado:**

```
Digite o número: 3
Digite o número: 7
Digite o número: 13
Digite o número: 76
Digite o número: 93
Digite o número: 54
Digite o número: 10
Digite o número: 0
Soma: 256
>>>
```

## for (Repetições)

O comando **for** itera sobre os itens de qualquer tipo de sequência (lista ou string), na ordem em que eles aparecem na sequência. A variável que aparece na linha do for se comporta como cada item da **lista**.



### Sintaxe:

```
for <variável> in <lista>:  
<comando que quero executar>
```



## Exemplos

### Exemplo 1:

```
nomes = ["Ariadyne", "Beatrix", "Caroline", "Debora", "Élida", "Jussara",  
"Veronica"]
```

```
for nome in nomes:  
    if nome.startswith("C") and nome.endswith("e"):  
        print(nome)
```



## Exemplos

**Exemplo 2:** Somar todos os números ímpares da lista.

```
print("Soma de números ímpares")

numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
soma_impar= 0
for numero in numeros:
    if (numero % 2) != 0:
        soma_impar = soma_impar+ numero
        print("Somando o número ímpar: {}".format(numero))

print("A soma dos ímpares é: {}".format(soma_impar))
```



# SIMPLIFICANDO

O **while** executa uma repetição até que uma determinada condição seja verdadeira.

O **for** executa uma repetição baseada em um número de vezes pré-determinado.



Vamos fazer um game!

## Game: Adivinhe o número!

No nosso jogo, o computador vai escolher um número secreto aleatoriamente entre 1 e 50 e o jogador deve tentar adivinhar o número.

Para cada tentativa, o computador vai dizer se o jogador acertou, ou se é o número secreto é maior ou menor que a tentativa.



## Vamos fazer um game!

```
import random

secreto = random.randint(1, 50)
tentativa = int(input("Digite um número de 1 a 50: "))
while tentativa != secreto: # enquanto jogador não acerta...
    if tentativa > secreto:
        print("Nope, tente um número menor")
    else:
        print("Hmm, tente um número maior")
    tentativa = int(input("Digite um número de 1 a 50: "))
print("AEEEE, ACERTOU!!")
```



pyladies  
FLORIPA

# import random

“Carrega” o módulo random, que serve para gerar números aleatórios

```
>>> import random
>>> random.randint(1, 6)
3
>>> random.randint(1, 6)
1
>>> random.randint(1, 6)
5
>>> random.randint(1, 1000)
352
>>> random.randint(1, 1000)
596
```

# Exercícios – créditos extras!!

- Altere o game para mostrar o número secreto antes de terminar
- Altere o game para sempre usar o mesmo número
- Faça uma versão troll, que permita o número secreto ser maior que 50
- Altere o game para mostrar “*Você perdeu! Quem sabe na próxima??*” se o jogador esgotar um máximo de 10 tentativas
  - *Dica:* você vai precisar de um contador
- Crie uma função para pedir a tentativa, e chame essa função no lugar de repetir o código `int(input(...))`



# Referências





# Referências

<http://wiki.python.org.br/PrincipiosFuncionais>

[Curso Python para Zumbis](#)

[Curso “An Introduction to Interactive Programming in Python” - Coursera](#)

<http://www.peachpit.com/articles/article.aspx?p=1312792&seqNum=6>

<https://docs.python.org/release/2.3.5/whatsnew/section-slices.html>

<http://www.bbc.co.uk/education/guides/zqh49j6/revision/3>

<https://www.youtube.com/watch?v=SYioCdLPmfw>

[https://pt.wikibooks.org/wiki/Python/Conceitos\\_b%C3%A1sicos/Tipos\\_e\\_operadores](https://pt.wikibooks.org/wiki/Python/Conceitos_b%C3%A1sicos/Tipos_e_operadores)

<http://www.dcc.ufrj.br/~fabiom/mab225/02tipos.pdf>

[www.arquivodecodigos.net/principal/dicas\\_truques\\_categoria2.php?linguagem=12&categoria1=1&categoria2=59](http://www.arquivodecodigos.net/principal/dicas_truques_categoria2.php?linguagem=12&categoria1=1&categoria2=59)



# Referências

[www.arquivodecodigos.net/principal/dicas\\_truques\\_categoria2.php?linguagem=12&categoria1=1&categoria2=51](http://www.arquivodecodigos.net/principal/dicas_truques_categoria2.php?linguagem=12&categoria1=1&categoria2=51)

[www.dotnetperls.com/lower-python](http://www.dotnetperls.com/lower-python)

<https://pt.wikipedia.org/wiki/Algoritmo>

<http://wiki.python.org.br/SoftwarePython>

<http://wiki.python.org.br/EmpresasPython>

<https://powerpython.wordpress.com/2012/03/16/programas-e-jogos-feitos-em-python/>

[http://tutorial.djangogirls.org/pt/python\\_installation/index.html](http://tutorial.djangogirls.org/pt/python_installation/index.html)

<https://powerpython.wordpress.com/2012/03/19/aula-python-17-estrutura-de-decisao/>

<https://under-linux.org/entry.php?b=1371>

<http://aprenda-python.blogspot.com.br/2009/10/nova-formatacao-de-strings.html>

<https://docs.python.org/3/library/string.html#format-spec>

<https://www.python.org/dev/peps/pep-3101/>



# Apêndice

(ou, coisas que a gente num sabia bem se precisava)



## TABELA VERDADE

A	B	A <b>and</b> B	A <b>or</b> B	<b>not</b> (A)	<b>not</b> (B)
True	True	True	True	False	False
True	False	False	True	False	True
False	True	False	True	True	False
False	False	False	False	True	True