



UNIVERSIDADE FEDERAL
DE SERGIPE

Introdução à programação com Python

Igor Terriaga Santos

Agenda

1. Introdução
2. Fundamentos
3. Constantes Variáveis
4. Comandos de Atribuição
5. I/O
6. Tipos de Dados
7. Expressões Lógicas
8. Comandos Condicionais

Agenda

9. Comentários

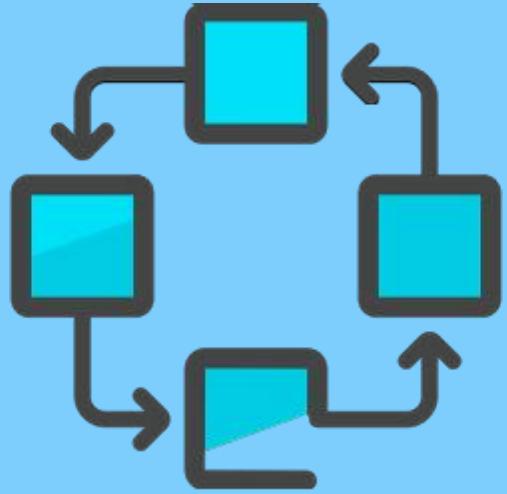
10. While

11. For

12. Estruturas de dados (Listas, Tuplas,
Dicionários).



Por que programar?



Por onde começar?

Primeiro temos que saber o que é um algoritmo!

Algoritmo - Definição

Dicionário

algoritmo



algoritmo

substantivo masculino

1. *mat* sequência finita de regras, raciocínios ou operações que, aplicada a um número finito de dados, permite solucionar classes semelhantes de problemas.
2. *inf* conjunto das regras e procedimentos lógicos perfeitamente definidos que levam à solução de um problema em um número finito de etapas.



Traduções, origem das palavras e mais definições

Algoritmo - Representações

Modo de Preparo

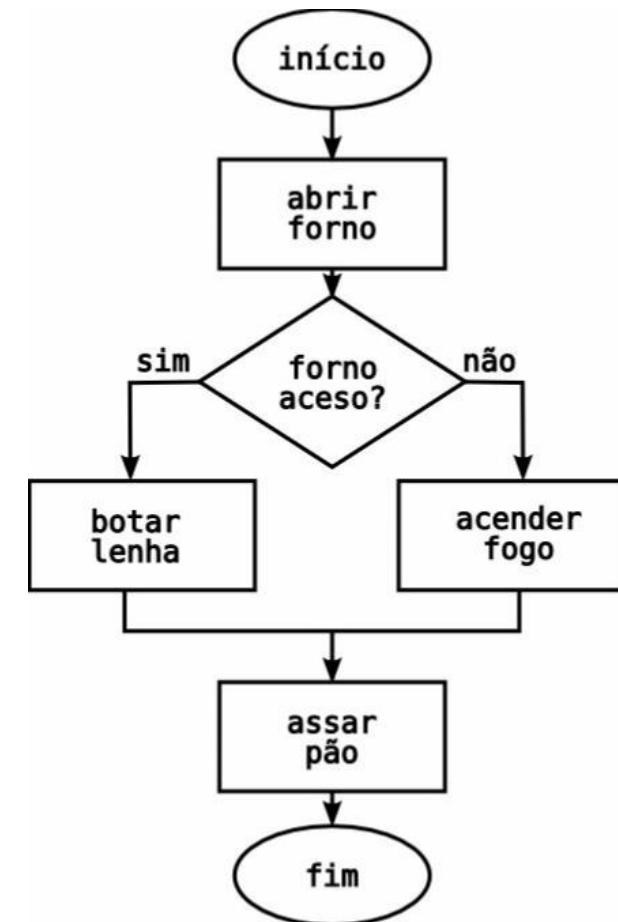
1. MODO DE FAZER:
2. Coloque o fermento na água morna quase fria, deixe uns 5 minutos,
3. depois acrescente o açúcar, o sal, o ovo e o óleo, vai acrescentando a.
4. farinha de trigo, até soltar das mãos.
5. Sove a massa até ficar macia, deixe descansar por uns 30 a 40 minutos,
6. na bacia coberta com um pano.

Mais itens...

PÃO CASEIRO DA NÉIA | Pães e salgados > Receita de Pão Caseiro ...
gshow.globo.com/receitas.../receita/pao-caseiro-da-neia-55ee98c84d3885377400004a.h...



Receita Culinária



Fluxograma

Algoritmo - Representações

```
inicio
    inteiro numero, r
    escrever " introduza um numero: "
    ler numero
    r <- numero % 2
    se r = 0 entao
        escrever "numero par"
    senao
        escrever "numero impar"
    fimse
fim
```

Pseudocódigo (Portugol)

```
1   print "Digite seu nome: "
2   nome = raw_input()
3   if nome != "":
4       print "Ola, " + nome + "!"
5   else :
6       while nome == "":
7           print "Digite seu nome: "
8           nome = raw_input()
9       print "Oii, " + nome + "!"
10      print "Finalmente nos conhecemos!"
11      print "Tchau! o/"
```

Linguagem de Programação Python

Programas, Softwares ou Aplicativos

- São algoritmos escritos em uma linguagem de programação (Python, Java, Ruby, C, Javascript, PHP, etc) e que são interpretados e executados por uma máquina.
- Projetados para resolver problemas do cotidiano e automatizar tarefas muitas vezes repetitivas.
- Muito do que fazemos hoje depende de softwares:
 - Sua matrícula pelo SIGAA
 - As aplicações usadas no seu smartphone
 - Funcionamento de eletrodomésticos, como o microondas
 - Sistema de semáforos de uma cidade
 - Entre muitas outras coisas



“
*A arte de programar consiste na
arte de organizar e dominar a
complexidade.*

-Edgar Dijkstra

“

*Eu escolho uma pessoa preguiçosa
para fazer um trabalho duro.
Porque uma pessoa preguiçosa irá
encontrar uma forma fácil de o
fazer.*

-Bill Gates

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec consequat, diam sed condimentum dapibus, quam risus porttitor eros, id pellentesque mauris tortor at orci. Quisque venenatis accumsan nulla eget scelerisque. Cras interdum enim ac finibus condimentum. Pellentesque tempor turpis vel lectus vestibulum, a sollicitudin est congue. Sed placerat ornare nisl, ut tempus magna eleifend sit amet. Integer ut nisi diam. Proin arcu tortor, auctor in lectus a, consectetur suscipit justo.

Curabitur a mi tempus, euismod mi in, finibus ipsum. Fusce finibus erat at nisi malesuada molestie at ac urna. Proin pharetra libero at justo laoreet, in ullamcorper eros ornare. Nunc non mollis ipsum, quis efficitur ex. Morbi non sapien quis justo mollis lobortis. Vivamus a lorem ut mauris ultrices vestibulum. Nulla dolor sapien, maximus eget augue vitae, pharetra interdum tortor. Praesent sit amet neque risus. Sed vehicula libero in mattis fringilla. Etiam rutrum massa non urna aliquam sollicitudin. Aenean enim erat, aliquet sed nisl et, malesuada vulputate purus. Fusce blandit, sapien nec eleifend accumsan, metus diam laoreet odio, vitae auctor ex felis non velit. Mauris convallis mattis leo vitae tempor.

Você poderia responder essas simples perguntas?

Quantas vogais tem no texto?

Quantas consoantes tem no texto?

Simples, usando Python!

Aplicação:

```
'''  
Desenvolvedor: Jefferson  
Versão: 1.0.0  
Descrição: Aplicação para o cálculo da quantidade de  
vogais e consoantes em um texto.  
nome = input('Nome do arquivo:')  
arquivo = open(nome, 'r')  
texto = arquivo.read()  
print('Conta Letras - Vogais e Consoantes \n')  
texto = texto.lower() # Converte para minúsculas  
  
# Retira os pontos e espaços  
texto = texto.replace(' ', '')  
texto = texto.replace('\n', '')  
texto = texto.replace('.', '')  
texto = texto.replace(',', '')  
texto = texto.replace(';', '')  
  
vogais = 0  
consoantes = 0  
  
# Conta a quantidade de vogais e consoantes  
for caracter in texto:  
    if caracter in 'aeiou':  
        vogais = vogais + 1  
    else:  
        consoantes = consoantes + 1  
  
# Imprime o resultado  
print('Vogais: %d' %vogais)  
print('Consoantes: %d' %consoantes)
```

Resultado:

```
Nome do arquivo: exemplo.txt  
Conta Letras - Vogais e Consoantes  
  
Vogais: 430  
Consoantes: 559  
>>>
```



Fácil, não?



Se não entendeu, fique tranquilo!

Você verá que não é nada de mais e nem um
'bicho de sete cabeças' essa tal de programação.

Programação, pra quê?

- Hoje, a computação está em todos lugares e somos altamente dependentes de software.
- A programação hoje é o novo ‘aprender inglês’.
- Compreender programação te ajudará a entender melhor o novo mundo que nos cerca, onde estamos cada vez mais conectados.
- Melhora seu raciocínio lógico na resolução de problemas.
- Ajudará na sua produtividade.
- Te ajudará a conseguir melhores empregos.
- Mercado de trabalho amplo.



Python: Fundamentos

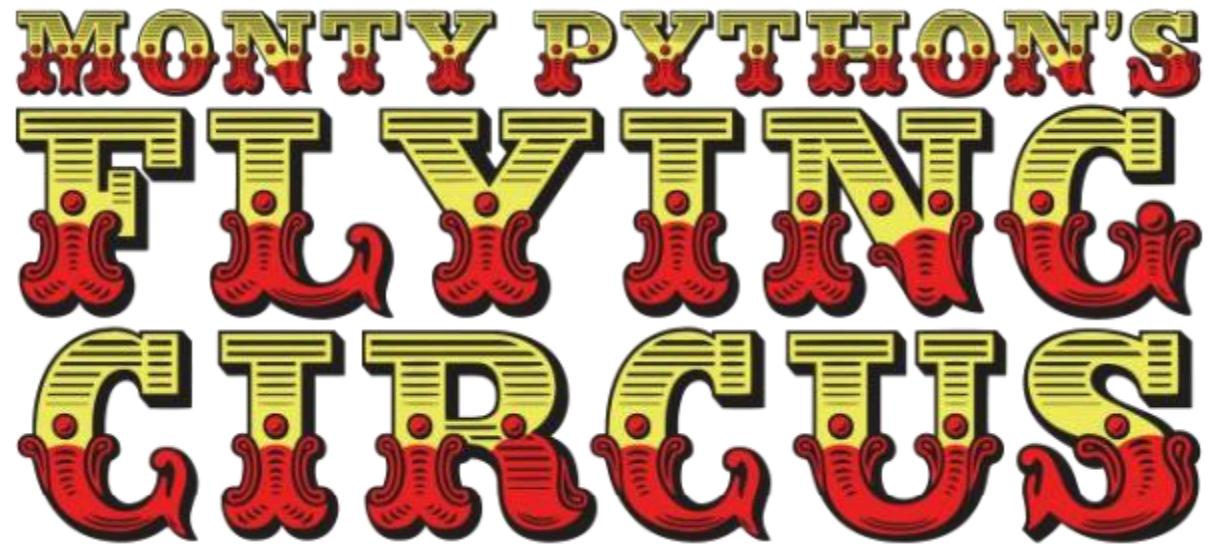
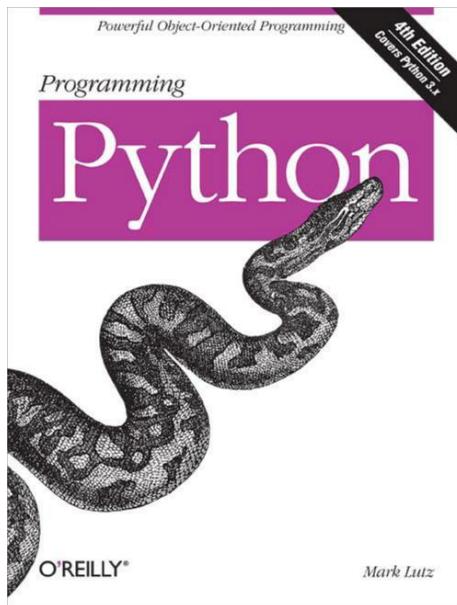
História

- Criada em 1991 pelo holandês Guido von Rossum no Centrum Wiskunde & Informatica(CWI) em Amsterdã, Holanda.
- Concebida a partir da linguagem ABC.
- Implementação oficial mantida pela PSF em C, conhecida como CPython.
- Outras implementações:
 - Python para .NET(IronPython)
 - JVM(Jython)
 - Python(PyPy)



História

- Nome inspirado no grupo humorístico britânico Monty Python Flying Circus.
- Ao contrário que muita gente pensa o nome não faz referência a cobra Python, o símbolo da cobra foi adotado graças a editora O'reilly.



Características

- Linguagem de programação livre, de altíssimo nível, interpretada, orientada a objetos, de tipagem forte e dinâmica.
- Linguagem simples, focada para priorizar o esforço do programador ao invés do esforço computacional, a ênfase da linguagem está em sua legibilidade.
- Uso de identação para marcar o código.
- Linguagem case sensitive
 - Ou seja: variavel, Variavel, VARIABEL são diferentes!
- Multi-paradíigma:
 - Imperativa
 - Funcional
 - Orientação a objetos

Características

- Multiplataforma
 - Pode ser usada no Linux, Mac, Windows, Web, e até para dispositivos móveis, como Android e IOS.
- Baterias inclusas, muito do que você precisa já pode estar incluso na instalação básica do python.

Versões

- **Python 1.0 - 1994**
 - Python 1.2 - 1995
 - Python 1.6. - 2000
- **Python 2.0 - 2000**
 - Python 2.1 - 2001
 - Python 2.2 - 2001
 - Python 2.3 - 2003
 - Python 2.4 - 2004
 - Python 2.5 - 2006
 - Python 2.6 - 2006
 - Python 2.7 - 2010 (Atual)
- **Python 3.0 - 2008**
 - Python 3.1 - 2001
 - Python 3.2 - 2011
 - Python 3.3 - 2012
 - Python 3.4 - 2014
 - Python 3.5 - 2015
 - Python 3.6 - 2016
 - Python 3.6.4 - 2017
 - Pytthon 3.7.2 - 2018
 - **Python 3.7.2 - 2019 (atual)**

Neste curso utilizaremos a partir da versão 3.7.x

O Zen do Python



- Bonito é melhor que feio.
- Explícito é melhor que implícito.
- Simples é melhor que complexo.
- Complexo é melhor que complicado.
- Linear é melhor que aninhado.
- Esparsos é melhor que denso.
- Legibilidade conta.
- Casos especiais não são especiais o bastante para quebrar as regras.
- Ainda que a preticidade vença a pureza.
- Erros nunca devem passar silenciosamente.
- A menos que sejam explicitamente silenciados.
- Diante da ambiguidade, recuse a tentação de adivinhar.
- Deveria haver um, e somente um, modo óbvio de fazer algo.
- Embora esse modo possa não ser óbvio a princípio a menos que você seja holandês.

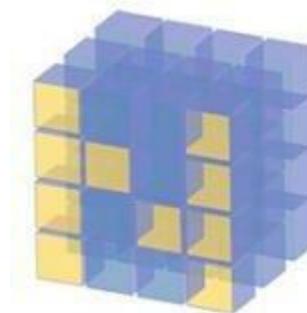
O Zen do Python



- Agora é melhor que nunca.
- Embora nunca frequentemente seja melhor que “já”.
- Se a implementação é difícil de explicar, é uma má ideia.
- Se a implementação é fácil de explicar, pode ser uma boa ideia.
- Namespaces são uma grande ideia, vamos ter mais dessas!

Usos do Python - Frameworks

django



NumPy

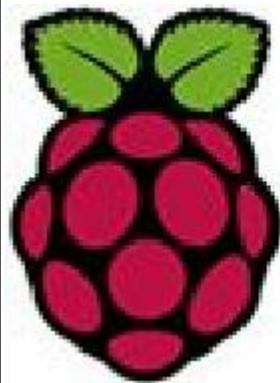


kivy

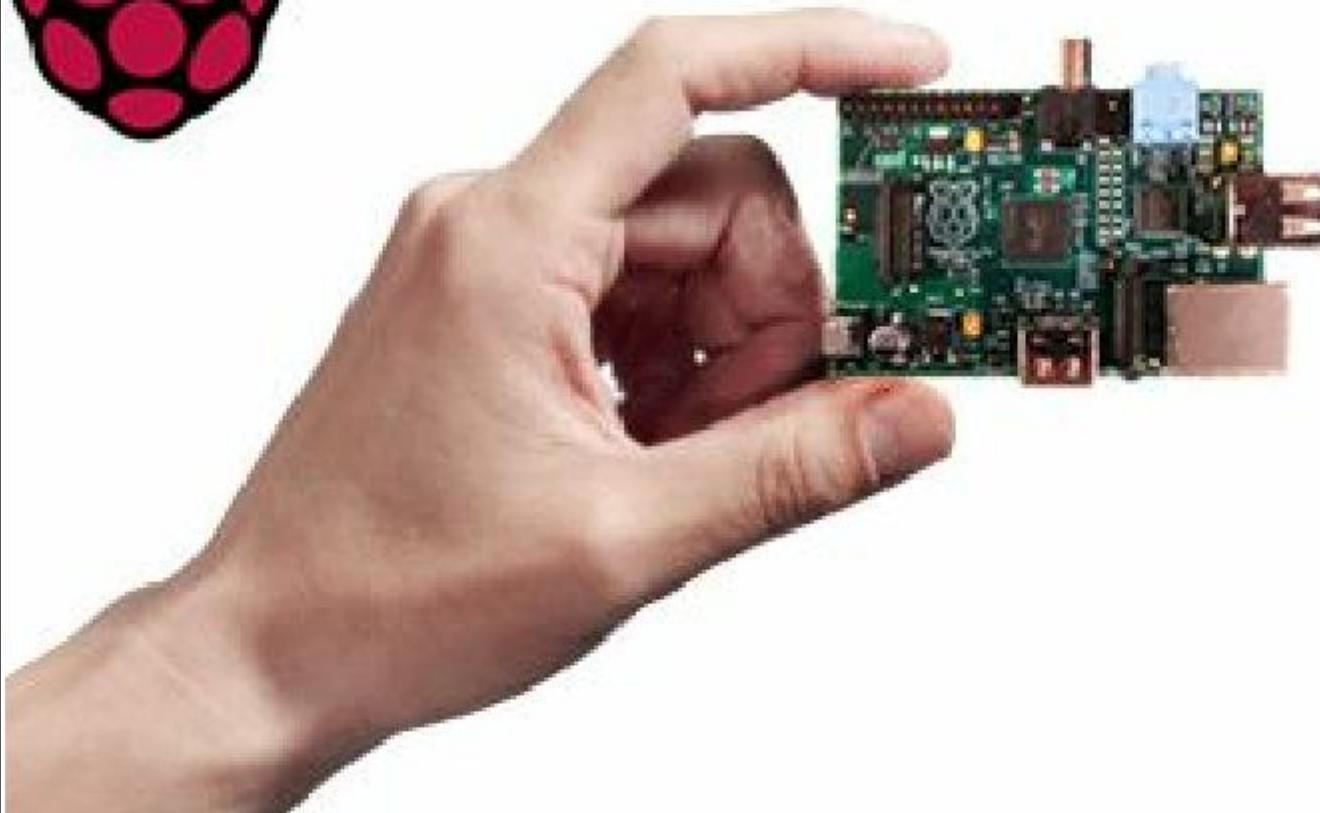


Plone®

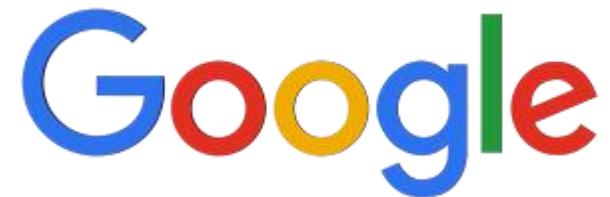
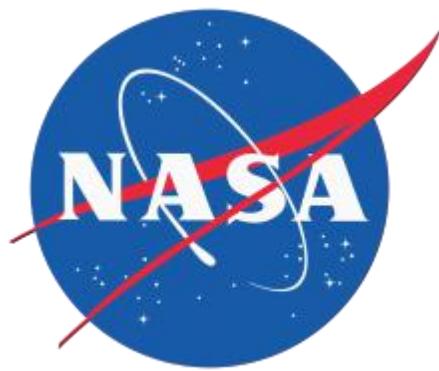
Outros usos do python



Raspberry Pi™



Onde o Python é usado



Downloads

- Python: <https://www.python.org/downloads/>
 - Baixem a última versão
- Notepad++:
<https://notepad-plus-plus.org/download/v7.5.4.html>
- Visual Studio Code:
<https://code.visualstudio.com/download>
- Pycharm:
<https://www.jetbrains.com/pycharm/download/#section=windows>

Material complementar

- Texto: PEP 8 - Guia de boas práticas para escrita de código em Python
 - <http://pep8.org/>
- Vídeo: Curso em Vídeo - Curso Python #02 - Para que serve Python?
 - <https://www.youtube.com/watch?v=Mp0vhMDI7fA>
- Livro: Python in Education
 - <http://www.oreilly.com/programming/free/files/python-in-education.pdf>
- Site: Python Brasil
 - <http://python.org.br/>



Constantes e Variáveis

Constantes

- São espaços reservados na memória do computador para armazenar uma valor fixo, que não será possível modificá-lo ao passar do tempo.
- Podem ser letras, nomes, números, entre outros tipos.
 - **Constantes numéricas:** números com valor fixo, podem ser inteiros ou reais(valores reais usam ponto decimal (.)).
 - **Constantes string:** nomes ou caracteres com valor fixo, são delimitados por apóstrofo(') ou aspas(").

```
>>> print(1)
1
>>> print(5.6)
5.6
>>> print('a')
a
>>> print("Olá mundo")
Olá mundo
>>> |
```

Variáveis

- Posição na memória que através de um nome armazena valores, dados.
- Estes valores podem ser recuperados através dos nomes das variáveis.
- Conteúdo pode ser modificado através de um comando de atribuição (=).
- Em Python os tipos das variáveis não precisam ser declaradas, portanto, o tipo da variável pode variar durante a execução do programa.

```
>>> x = 4          >>> print(x)
4
>>> y = 20         >>> print(x)
>>> x = 12.5       12.5
                           >>> print(y)
                           20
```

Variáveis - Regras

- Devem começar com uma letra ou sublinhado _
- Não devem começar com números
- Podem conter **letras, números ou sublinhados**
- Python diferencia letras minúsculas de minúsculas, o que chamamos de *case sensitive*
- **Variáveis válidas:**
 - nome _nome nome2 sobre_nome
- **Variáveis inválidas:**
 - 1nome nome.1 #nome
- **Variáveis diferentes:**
 - nome Nome NOME

Variáveis - Boas Práticas

Como os nomes das variáveis são definidas pelo programador, procure seguir essas dicas:

- Usar nomes que expressam o significado da variável.
- Usar mneumônicos, palavras que facilitam a memoriização.
- Evitar usar caracteres especiais em nomes de variáveis (! \$ % ?).
- Evitar usar acentuação
 - **Errado:** endereço = “Rua A”
 - **Certo:** endereco = “Rua A”
- Evitar nomear as variáveis apenas com letras,(i, j, x, y, z, a), exceto em contadores de laços de repetição e em coordenadas.
- Obedecer as regras vistas anteriormente citadas.

Variáveis - Legibilidade

exemplo.py x

```
1 xyuasyausan = 20
2 shaibxaисbc = 200
3 hsaushaihs = xyuasyausan * shaibxaисbc
4 print (hsaushaihs)
```

exemplo.py x

```
1 a = 20
2 b = 200
3 c = a * b
4 print (c)
```

O que cada código faz?

Qual é mais legível?

exemplo.py x

```
1 valor_hora = 20
2 horas_trabalhadas = 200
3 salario = valor_hora * horas_trabalhadas
4 print (salario)
```

Variáveis - Palavras Reservadas

O Python possui palavras que não podem ser usadas como nome de variáveis, pois elas desempenham determinadas funções dentro da linguagem.

Estas são:

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Fonte: <https://www.programiz.com/python-programming/keyword-list>



Comando de atribuição

Atribuindo valores

- Para atribuir um valor a uma variável é utilizado o comando de atribuição (=)
 - **variavel = valor** (Onde lê-se variável recebe valor)
- Não confundir o comando de atribuição (=) com o sinal de igualdade(==)
 - **variavel == valor** (Diferente do caso acima, aqui lê-se variável é igual a valor).
- Um comando de atribuição é formado por uma **expressão do lado direito** atribuída a uma **variável** do lado esquerdo, esta que armazena o resultado.
- Um comando de atribuição pode ser formado por:
 - **Variável, constantes e operadores**

$$x = 35 * + (2 - x)$$

Trocando valores de duas variáveis

x = 10

y = 5

x = y

y = x

print(x)

print(y)

Quais os resultados das variáveis
x e y respectivamente?

5 e 10?

Trocando valores de duas variáveis

x = 10

y = 5

x = y

y = x

print(x)

print(y)

Quais os resultados das variáveis
x e y respectivamente?

5 e 10?



O RESULTADO É 5 e 5!

Trocando valores de duas variáveis

x = 10



x 10

y = 5



y 5

x = y



x ~~10~~ 5

y = x

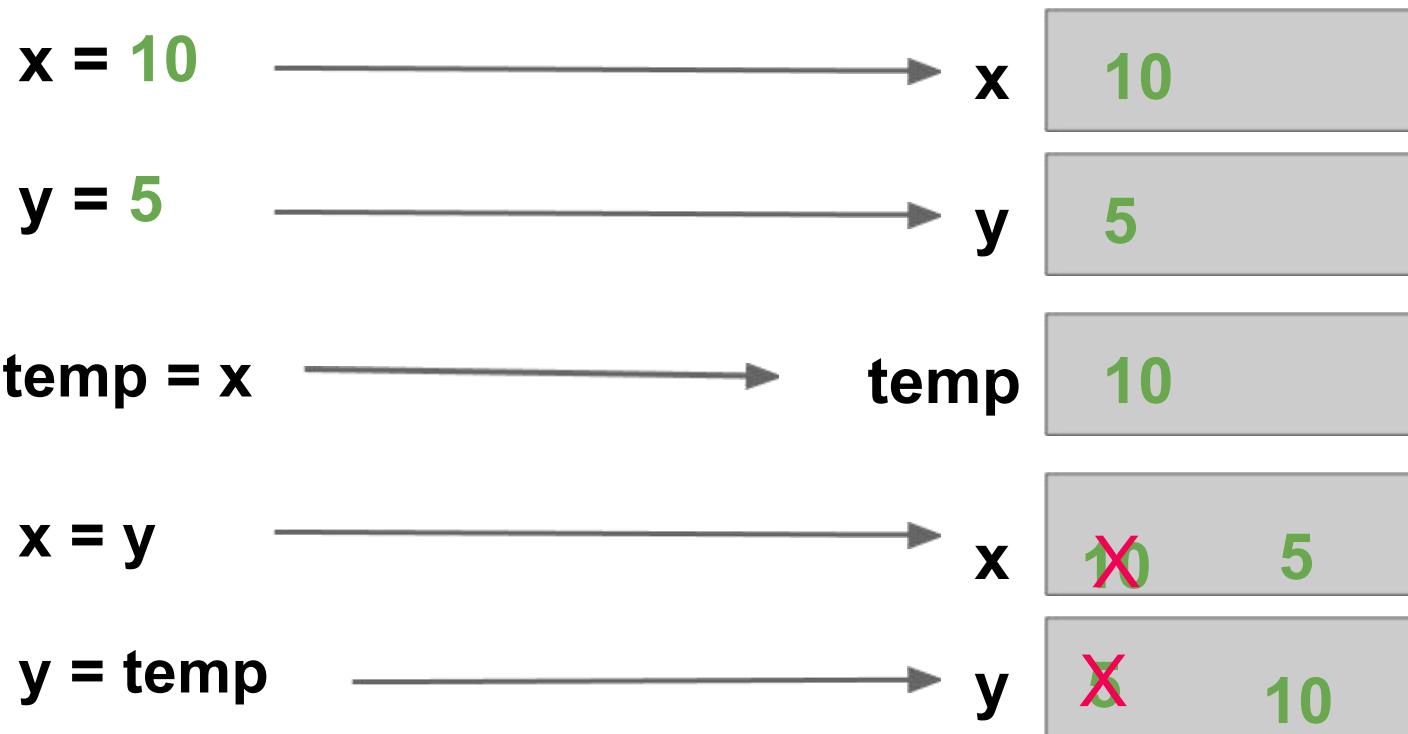


y ~~5~~ 5

print(x)

print(y)

Trocando valores de duas variáveis



`print(x)`

`print(y)`

Agora está certo, os valores foram trocados!
Serão impressos respectivamente 5 e 10

Atribuição múltipla

Com a atribuição múltipla é possível definir vários valores para variáveis diferentes de uma só vez, como o exemplo abaixo:

```
>>> a, b, c = 1, 'a', "olá"
>>> print(a,b,c)
1 a olá
>>> |
```

Trocando os valores de duas variáveis com a atribuição Múltipla

Usando a atribuição múltipla o exemplo anterior da troca de valores das variáveis ficaria bem mais simples, veja:

```
>>> y = 10
>>> x = 5
>>> print(x, y)
5 10
>>> x, y = y, x
>>> print(x, y)
10 5
>>> |
```



Entrada e Saída de Dados

Entrada de Dados

- Uso da função `input()`
 - Maneira mais básica de entrada de dados em Python, ela recebe um string e armazena em uma variável:

```
>>> nome = input('Digite seu nome: ')
```

- O comando acima irá gerar a seguinte mensagem:

```
>>> Digite seu nome: _
```

- O cursor `_` está esperando que algo seja digitado

```
>>> Digite seu nome: Antônio
```

- Após digitado, o valor é armazenado na variável `nome`, então podemos imprimir o resultado

```
>>> print('Olá', nome)
```

- E o resultado final será:

```
>>> Olá Antônio
```

Entrada de Dados - Strings

- Ao usar a função `input()`, o que foi digitado retorna uma String, mesmo que sejam digitados números.

- Veja o exemplo:

```
>>> Idade = input('Digite sua idade: ')
```

- Então digita-se a idade:

Digite sua idade: 20

- Podemos verificar o tipo de dado retornado através da função `type()`

```
>>> type(Idade)
```

- Logo, podemos verificar que ela é um tipo String

<class 'str'>

Entrada de Dados - Números Inteiros

- Já que a função `input()` só retorna String, como eu poderia ler e armazenar um número inteiro?
 - É só transformar o valor em inteiro (`int`):

```
>>> valor = input()
>>> valor_inteiro = int(valor)
```
- A variável `valor` continua sendo uma String, porém a variável `valor_inteiro` contém o inteiro correspondente a String da variável `valor`. Aqui está uma maneira resumida de se escrever o mesmo comando:

```
>>> valor_inteiro = int(input())
```
- Do mesmo modo podemos converter o valor retornado para um tipo `float`

```
>>> valor_float = float(input())
```

Saída de dados

- Já temos todos os dados armazenados em variáveis, agora teremos que mostrar isto ao usuário, e é para este propósito que usaremos a função `print()`
- Ela recebe um ou mais valores separados por vírgula e os imprime na tela.

```
>>> nome = input('Digite seu nome: ')
Digite seu nome: José
>>> cidade = input('Digite a cidade que você mora: ')
Digite a cidade que você mora: Aracaju
>>> print('Seu nome é ', nome, ' e mora em ', cidade, '.')
Seu nome é José e mora em Aracaju .
>>> |
```

Saída de dados

- Outra forma de imprimir usando a função `print()` e a função `format()`

```
>>> nome = input('Digite seu nome: ')
Digite seu nome: José
>>> cidade = input('Digite a cidade que você mora: ')
Digite a cidade que você mora: Aracaju
>>> print('Seu nome é {} e mora em {}'.format(nome, cidade))
Seu nome é José e mora em Aracaju
>>> |
```

Saída de dados

- Mas se você quisesse imprimir uma saída deste jeito:

Dados do Usuário

Nome: José

Cidade: Aracaju

Saída de dados

- Através do uso de aspas triplas “”” com a função `format()` é possível ter uma saída assim:

```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Inte
1)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> nome = input('Digite seu nome: ')
Digite seu nome: José
>>> cidade = input('Digite a cidade que você mora: ')
Digite a cidade que você mora: Itabaiana
>>> print(""""
Dados do Usuário
Nome: {}
Cidade: {}""".format(nome, cidade))

Dados do Usuário
Nome: José
Cidade: Itabaiana
>>>
```

Então, vamos exercitar um pouco?

1. Escreva um algoritmo em Python que leia o ano de nascimento de uma pessoa e imprima sua idade.
2. Você foi contratado para desenvolver um sistema de uma locadora de carros, seu trabalho é desenvolver um sistema que calcule o dia em que o usuário terá que devolver o carro à locadora. O formato de entrada será o seguinte:

Nome:

Data atual:

Quantidade de dias de aluguel:

Já a tela de saída terá que conter os seguintes dados:

Nome:

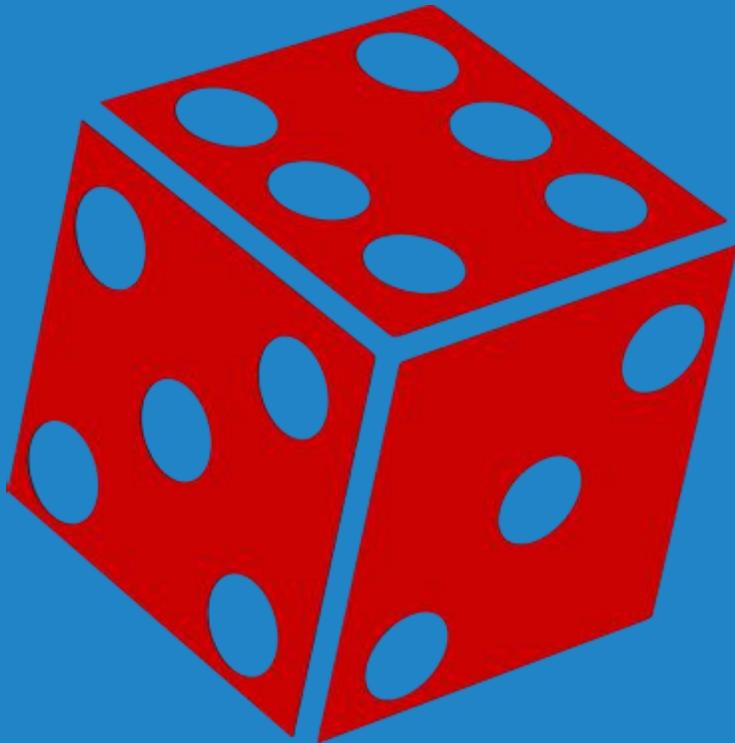
Data de entrega do veículo:

Então, vamos exercitar um pouco?

3. Escreva um programa para calcular a redução do tempo de vida de um fumante. Pergunte a quantidade de cigarros fumados por dia e quantos anos ele já fumou.

Considere que um fumante perde 10 minutos de vida a cada cigarro, calcule quantos dias de vida um fumante perderá. Exiba o total de dias.

4. Programa que converte uma temperatura em digitada em °C para °F. A formula para conversão será: $f = ((9 * \text{celsius})/5 + 32)$.



Tipos de Dados

Tipos de Dados

- **Tipo** é uma maneira de classificar as coisas, sejam objetos ou informações.
- **Dado** é um valor em sua forma bruta.
- Logo, os tipos de dados em programação seriam como estariam classificados os valores, estes que podem ser armazenados em variáveis.

Tipos de dados - String

- É um conjunto de caracteres que formam palavras.
- Operações básicas com String:
 - Existem dois operadores que podem ser usados em Strings:
 - +
 - *
 - Ambos servem para concatenar .

```
>>> nome = 'João'  
>>> sobrenome = ' Nascimento'  
>>> nome_completo = nome + sobrenome  
>>> print(nome_completo)  
João Nascimento  
>>> |
```

```
>>> texto = 'hue'  
>>> texto_multiplicado = texto * 6  
>>> print(texto_multiplicado)  
huehuehuehuehue  
>>>
```

Tipos de dados - Números

- Existem 4 tipos numéricos em Python:
 - Inteiro (int)
 - Ponto Flutuante (float)
 - Booleano (bool)
 - Complexo (complex)*

*Neste curso não usaremos números complexos.

Tipos de dados - Números

- Operadores usados:

Operador	Operação
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
//	Divisão Inteira
**	Exponenciação
%	Resto da Divisão

```
>>> print(1+1)
2
>>> print(3-1)
2
>>> print(2*4)
8
>>> print(6/2)
3
>>> print(5/2)
2
>>> print(5.0/2)
2.5
>>> print(5//2)
2
>>> print(5.0//2)
2.0
>>> print(2**4)
16
>>> print(10%3)
1
>>> |
```

Tipos de dados - Booleanos

- É também um tipo de dado numérico, porém só armazena os valores 0 e 1.
- Utilizado para fazer comparações entre expressões, resultando em verdadeiro ou falso, onde:
 - 0: falso
 - 1: verdadeiro

```
>>> falso = False
>>> type(falso)
<type 'bool'>
>>> print(falso)
False
>>> verdadeiro = True
>>> type(verdadeiro)
<type 'bool'>
>>> print(verdadeiro)
True
>>>
```

Números Inteiros e Reais Misturados

- Quando é feita uma operação entre números inteiros(int) e reais (float) o valor sempre será um número real(float).

```
>>> operacao = 20.0 + 1
>>> type(operacao)
<type 'float'>
>>> print(operacao)
21.0
>>> operacao = ((20 + 4) * 3.5)/2
>>> type(operacao)
<type 'float'>
>>> print(operacao)
42.0
>>> |
```

Precedência de Operadores

- Ao usar vários operadores juntos em um expressão, devemos saber qual tem prioridade, ou seja, qual será executado primeiro.
- Em Python essa é a ordem com que serão feitas as operações:

Parênteses
Exponenciação
Multiplicação/Divisão
Soma/Subtração
Esquerda → Direita



Precedência de Operadores

$$21 - 50 + (10 * 20) / 2^{**2}$$

```
>>> print(21 - 50 + (10 * 20) / 2**2)  
21  
>>> |
```

$$21 - 50 + 200 / 2^{**2}$$

$$21 - 50 + 200 / 4$$

$$21 - 50 + 50$$

$$-29 + 50$$

$$21$$

Parênteses

Exponenciação

Multiplicação/Divisão

Soma/Subtração

Esquerda → Direita

Conversão de tipos - Números

- Através de funções **built-in** é possível fazer a conversão entre tipos.
 - **float()** - Transforma para float
 - **int()** - Transforma para inteiro

```
>>> numero_inteiro = 5
>>> type(numero_inteiro)
<type 'int'>
>>> numero_float = 5.0
>>> type(numero_float)
<type 'float'>
>>> numero_inteiro_convertido = float(numero_inteiro)
>>> type(numero_inteiro_convertido)
<type 'float'>
>>> numero_float_convertido = int(numero_float)
>>> type(numero_float_convertido)
<type 'int'>
```

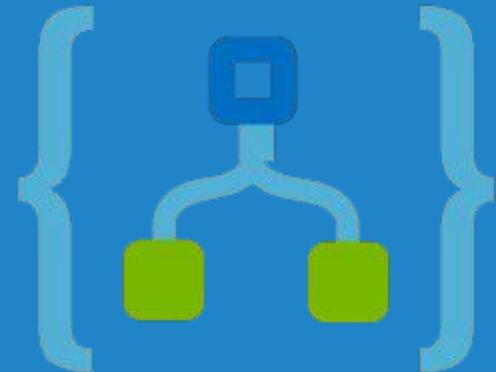
Conversão de tipos - String

- Podemos também converter String em n[umeros com as funções `int()` e `float()`, desde que a **String só contenha números.**

```
>>> valor_string = '123'  
>>> type(valor_string)  
<type 'str'>  
>>> soma = valor_string + 1  
  
Traceback (most recent call last):  
  File "<pyshell#2>", line 1, in <module>  
    soma = valor_string + 1  
TypeError: cannot concatenate 'str' and 'int' objects  
>>> soma = int(valor_string) + 1  
>>> print(soma)  
124  
>>>
```

Exercício

1. Faça um programa em python que calcule a área de cubo.
2. Faça um prorama em python que leia um valor em metros e o transforme em milimetros.
3. Escreva um programa que calcule o tempo de uma viagem de carro. Pergunte a distância a percorrer e a velocidade média esperada para a viagem.



Expressões Lógicas

Expressões Lógicas

- Existem dois tipos de expressões lógicas:
 - Expressões relacionais
 - Expressões booleanas
- São comumente utilizadas para controlar o fluxo de aplicações.

Expressões Relacionais

- **Expressões relacionais** comparam valores entre variáveis ou constantes e retornam **true**(verdadeiro) ou **false**(falso).
- **Operadores relacionais** são usados para fazer a comparação dentro das expressões relacionais.

Operador	Significado
<	Menor que
<=	Menor ou Igual que
==	Igual a
>	Maior que
>=	Maior ou Igual que
!=	Diferente

*Não confundir o operador de atribuição = com o de comparação ==

Expressões Relacionais

```
>>> idade = 18
>>> idade > 18
False
>>> idade >= 18
True
>>> idade < 18
False
>>> idade <= 18
True
>>> idade == 18
True
>>> idade != 18
False
>>> maior_idade = idade >= 18
>>> maior_idade
True
>>> |
```

Operador	Significado
<	Menor que
<=	Menor ou Igual que
==	Igual a
>	Maior que
>=	Maior ou Igual que
!=	Diferente

Expressões Booleanas

- Do mesmo modo que nas expressões relacionais, as expressões booleanas retornam valores **True (verdadeiro)**, ou **False (falso)** resultantes da comparação entre sentenças, variáveis ou constantes.
- São usadas usadas para controlar o fluxo de aplicações.
- É comum usar operadores booleanos e relacionais juntos.

Operador	Significado
not	True se o operador for False e False se o operador for True
or	True se ao menos um dos operadores forem True
and	True se ambos operadores forem True

Operador not (não - negação)

```
>>> verdadeiro = True
>>> falso = False
>>> not verdadeiro
False
>>> not falso
True
>>> not not verdadeiro
True
>>> not not falso
False
>>> |
```

A	not A
True	False
False	True

Operador or (ou - disjunção)

```
>>> bike = True
>>> onibus = True
>>> bike or onibus
True
>>> bike = True
>>> onibus = False
>>> bike or onibus
True
>>> bike = False
>>> onibus = True
>>> bike or onibus
True
>>> bike = False
>>> onibus = False
>>> bike or onibus
False
>>> |
```

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

Operador and (e - conjunção)

```
>>> frequencia = 80
>>> media = 8
>>> frequencia >= 75 and media >= 5
True
>>> frequencia = 90
>>> media = 4
>>> frequencia >= 75 and media >= 5
False
>>> frequencia = 60
>>> media = 9
>>> frequencia >= 75 and media >= 5
False
>>> frequencia = 40
>>> media = 4
>>> frequencia >= 75 and media >= 5
False
>>>
```

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False



Comandos Condicionais

Como tomar decisões?

- Programas de computador têm que tomar decisões o tempo todo, e é para isso que usamos comandos condicionais.
- São comandos que dependendo da entrada dos dados tomam decisões diferentes.
- E em Python usamos o comando `if/else/elif` para tomar decisões.

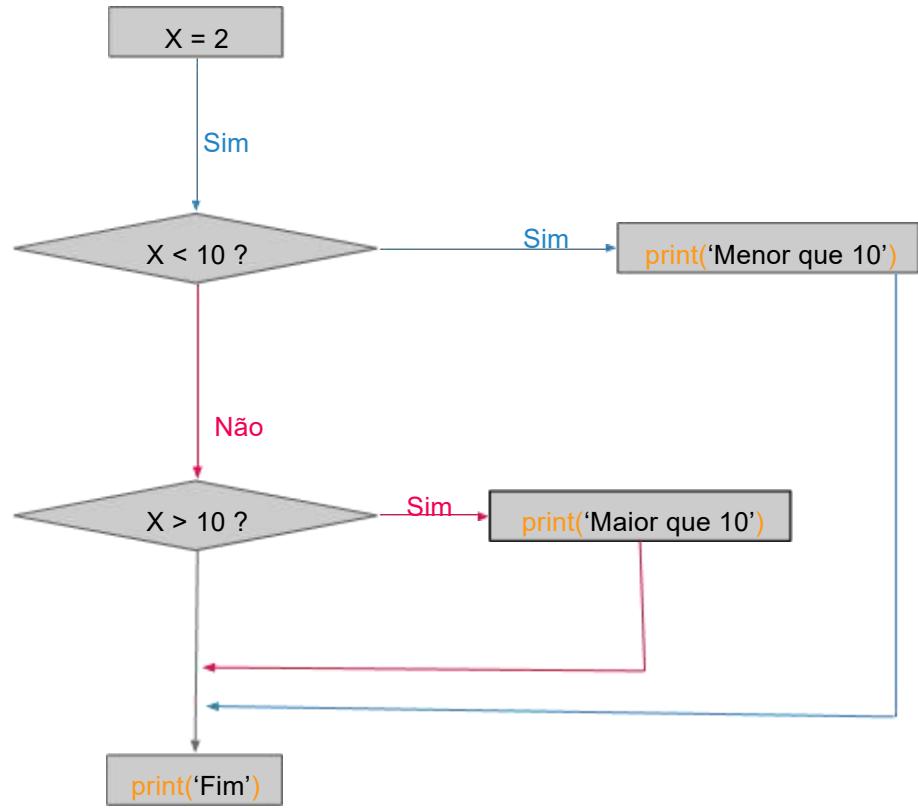
E se?

Entrada:

```
x = 2
if x < 10:
    print('Menor que 10')
if x > 10:
    print('Maior que 10')
print('Fim')
```

Saída:

```
Menor que 10
Fim
>>>
```



Blocos de códigos (Indentação)

- A linguagem Python usa indentação para saber onde um bloco de código começa e termina.
- Blocos de códigos são conjuntos de códigos dentro de um comando (**if/for/while**), método ou classe.



Blocos de códigos (Indentação)

- Para indentar seu código siga os seguintes passos:
 - Use ‘tab’ ou ‘espaço’ (**Não use os dois, ou um ou outro!**)
 - Após os comandos if/for/while aumente a indentação
 - Mantenha a indentação para continuar dentro do escopo do comando
 - Para sair do escopo do comando reduza a indentação
 - Espaços com a tecla ‘enter’ e nem comentários não são considerados para a indentação
 - E mais uma vez, (**não misture ‘tab’ com ‘espaço’!**)

Blocos de códigos (Indentação)

```
numero = 2
if numero % 2 == 0:
    print('O número é par')
    print('.....')
    print('O número continua sendo par')
print('Fim da parte 1 da aplicação')
print('-----')
```

```
for i in range(3):
    print(i)

    if i % 2 == 0 and i / 2 != 0:
        print('Número {} par'.format(i))
        print('-----')

    print('Ainda está dentro do comando for...')
    print('-----')

print('Fim do for')
print('Fim da aplicação')
```

Condições compostas

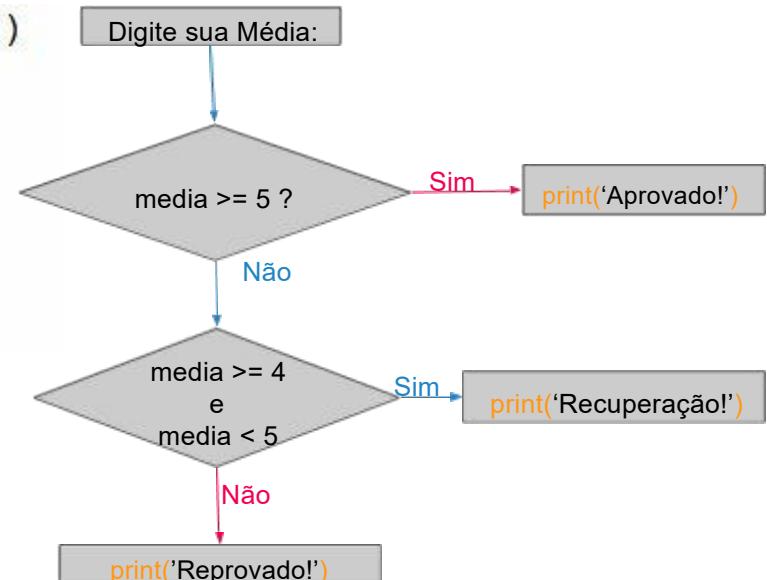
Entrada:

```
media = float(input('Digite sua Média:'))  
if media >= 5:  
    print('Aprovado!')  
elif media >= 4 and media < 5:  
    print('Recuperação!')  
else:  
    print('Reprovado!')
```

Saída:

```
Digite sua Média:4  
Recuperação!
```

```
>>> |
```



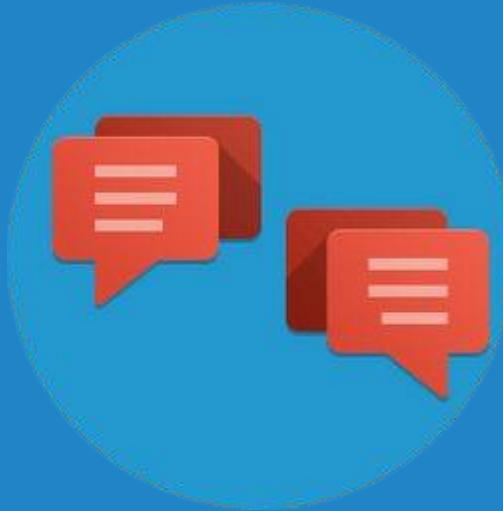
Exercícios

1. Faça um programa que leia três notas, calcule a média do aluno e por fim imprima sua situação, como segue:
 - a. $\text{media} = (\text{nota1} + \text{nota2} + \text{nota3})/3$
 - b. Situação:
 - i. Aprovado: $\text{media} > 7$
 - ii. Recuperação: $\text{media} < 7$ and $\text{media} \geq 5$
 - iii. Reprovado: $\text{media} < 5$
2. Faça um algoritmo que leia 2 números e informe qual é o maior dentre eles, ou se são iguais.
3. Faça um algoritmo que leia o comprimento de três retas e diga ao usuário se elas podem ou não formar um triângulo.

Exercícios

4. Escreva um programa que verifique se um ano é bissexto.
5. Faça um programa que leia um numero de 0 a 9999 e mostre na tela cada um dos dígitos separados.





Comentários

Para quê comentar?

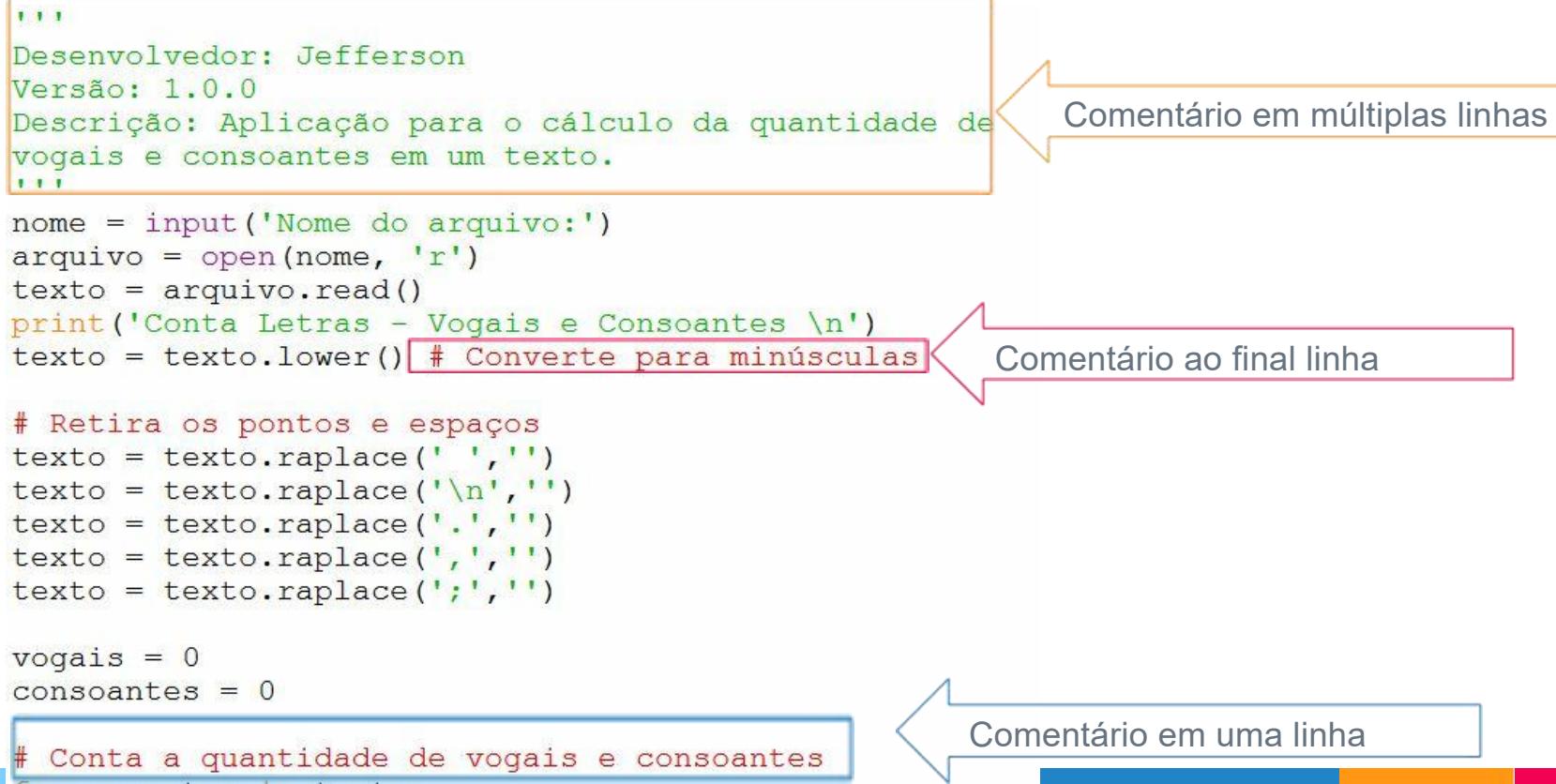
- Com o desenvolvimento de aplicações mais complexas, nem sempre é fácil de entendê-las.
- Temos vários motivos para usar comentários em nossos códigos, entre eles:
 - **Descrever** o que determinado comando faz.
 - **Documentar** o código com os dados de quem o desenvolveu, entre outras informações.
 - **Desabilitar** uma linha ou bloco de código temporariamente.

Tipos de comentários em Python

- **#** - o caractere ‘sharp’ é usado quando se quer comentar apenas uma linha.
- **"""** ou **””** - quando se quer comentar mais de uma linha é usado três aspas no começo e no final do que se quer comentar, podem ser tanto aspas simples “”, como aspas duplas “”.
 - Neste tipo de comentário, começa e termina com o mesmo tipo de aspas.
- Qualquer coisa entre estes comentários será ignorado pela aplicação.

Exemplo de código comentado

```
'''  
Desenvolvedor: Jefferson  
Versão: 1.0.0  
Descrição: Aplicação para o cálculo da quantidade de  
vogais e consoantes em um texto.  
'''  
  
nome = input('Nome do arquivo:')  
arquivo = open(nome, 'r')  
texto = arquivo.read()  
print('Conta Letras - Vogais e Consoantes \n')  
texto = texto.lower() # Converte para minúsculas  
  
# Retira os pontos e espaços  
texto = texto.replace(' ', '')  
texto = texto.replace('\n', '')  
texto = texto.replace('.', '')  
texto = texto.replace(',', '')  
texto = texto.replace(';', '')  
  
vogais = 0  
consoantes = 0  
  
# Conta a quantidade de vogais e consoantes  
for caracter in texto:  
    if caracter in 'aeiou':  
        vogais = vogais + 1  
    else:  
        consoantes = consoantes + 1  
  
# Imprime o resultado  
print('Vogais: %d' %vogais)  
print('Consoantes: %d' %consoantes)
```



Considerações sobre comentários

- Comente apenas o necessário.
 - Não é preciso comentar o que já está explícito.
- Use comentários que “expliquem”.
 - O objetivo dos comentários é ajudar na leitura e entendimento do código.
- Os comentários não afetam o desempenho do programa.



Comando de Repetição while

Quando usar um laço de repetição?

```
numero = int(input('Digite o número: '))

print('Tabuada: ')
print(numero, ' x 1 = ', numero )
print(numero, ' x 2 = ', numero * 2)
print(numero, ' x 3 = ', numero * 3)
print(numero, ' x 4 = ', numero * 4)
print(numero, ' x 5 = ', numero * 5)
print(numero, ' x 6 = ', numero * 6)
print(numero, ' x 7 = ', numero * 7)
print(numero, ' x 8 = ', numero * 8)
print(numero, ' x 9 = ', numero * 9)
```

- O que acham deste código?
- Alguma coisa errada?
- Muito repetitivo?

Quando usar um laço de repetição?

```
numero = int(input('Digite o número: '))
```

```
print('Tabuada: ')
print(numero, ' x 1 = ', numero )
print(numero, ' x 2 = ', numero * 2)
print(numero, ' x 3 = ', numero * 3)
print(numero, ' x 4 = ', numero * 4)
print(numero, ' x 5 = ', numero * 5)
print(numero, ' x 6 = ', numero * 6)
print(numero, ' x 7 = ', numero * 7)
print(numero, ' x 8 = ', numero * 8)
print(numero, ' x 9 = ', numero * 9)
```

- Problemas:

- Código repetitivo
- Difícil de manutenção
- Há poucas mudanças

Usando o while

- Para resolver este problema e melhorar nosso código podemos usar estruturas de repetição.
- Em Python existem 2 tipos destas estruturas: **while** e **for**.
- Veja como nosso algoritmo de tabuada ficaria usando o **while**:

```
numero = int(input('Digite o número: '))
n = 1
print('Tabuada: ')
while n <= 9:
    print(numero, ' x ', n, ' = ', numero * n )
    n = n + 1
```

Fluxo do while

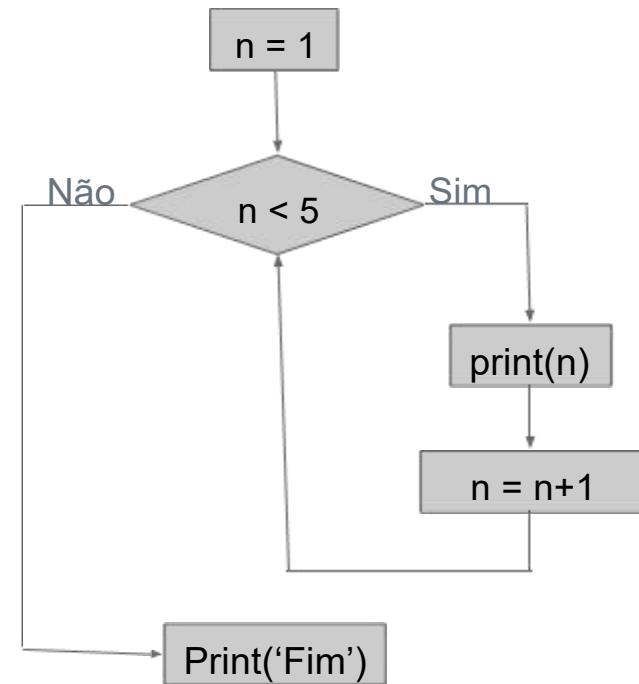
Entrada:

```
n = 1
while n < 5:
    print(n)
    n = n + 1
print('Fim')
>>>
```

The code initializes `n` to 1. It then enters a `while` loop where it prints the value of `n` (which is 1) and increments `n` by 1. This loop repeats until `n` is no longer less than 5. Finally, it prints 'Fim'.

1
2
3
4
Fim
>>>

Saída:



Cuidados com laços de repetição

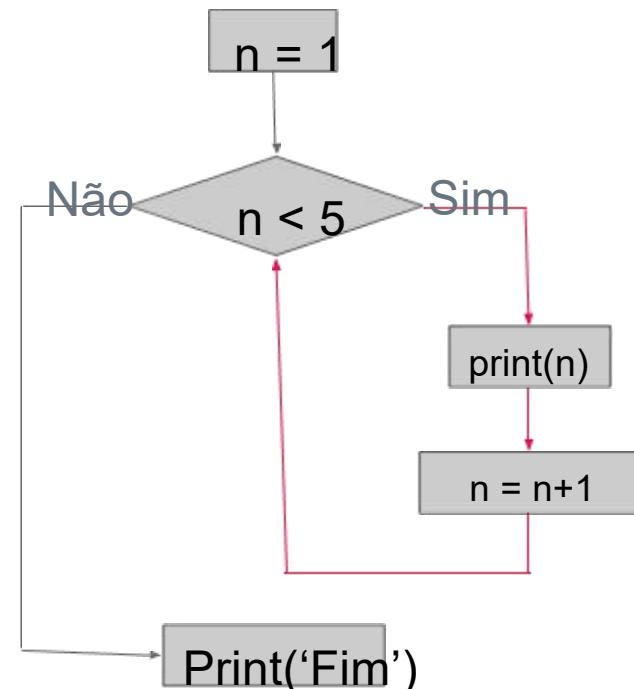
- Note que este algoritmo entra em um laço de repetição.
 - O que aconteceria se tirassemos o incremento $n = n + 1$?
 - Um laço infinito!

Entrada:

```
n = 1
while n < 5:
    print(n)
    #n = n + 1
print('Fim')
```

Saída:

1 1 1 1 1 1 1 1 ■ ■



Comando break

- Quando foi encontrado o que queria, e se quer sair do laço de repetição pode-se usar o comando **break**.
- Assim que o comando **break** é acionado, então o laço é encerrado e o algoritmo parte para a próxima instrução logo após o laço.

Entrada:

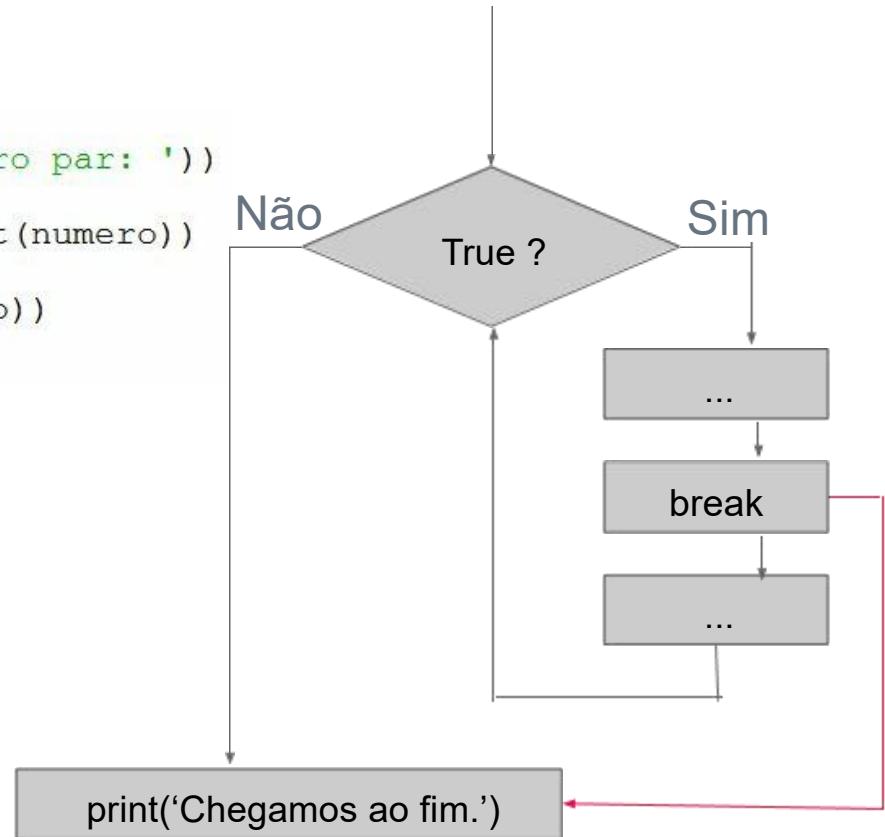
```
while True :  
    numero = int(input('Digite um número par: '))  
    if numero % 2 == 0:  
        print('Certo! {} é par!'.format(numero))  
        break  
    print('{} não é par!'.format(numero))  
print('Chegamos ao fim.')
```

Saída:

```
Digite um número par: 1  
1 não é par!  
Digite um número par: 2  
Certo! 2 é par!  
Chegamos ao fim.  
>>> |
```

Comando break

```
while True :  
    numero = int(input('Digite um número par: '))  
    if numero % 2 == 0 and numero != 0:  
        print('Certo! {} é par!'.format(numero))  
        break  
    print('{} não é par!'.format(numero))  
print('Chegamos ao fim.')
```



Comando continue

- Ao invés de encerrar o laço de repetição atual e ir ao próximo comando logo abaixo, o comando **continue** retorna ao topo do laço corrente. Caso seja True, o laço é executado novamente.

Entrada:

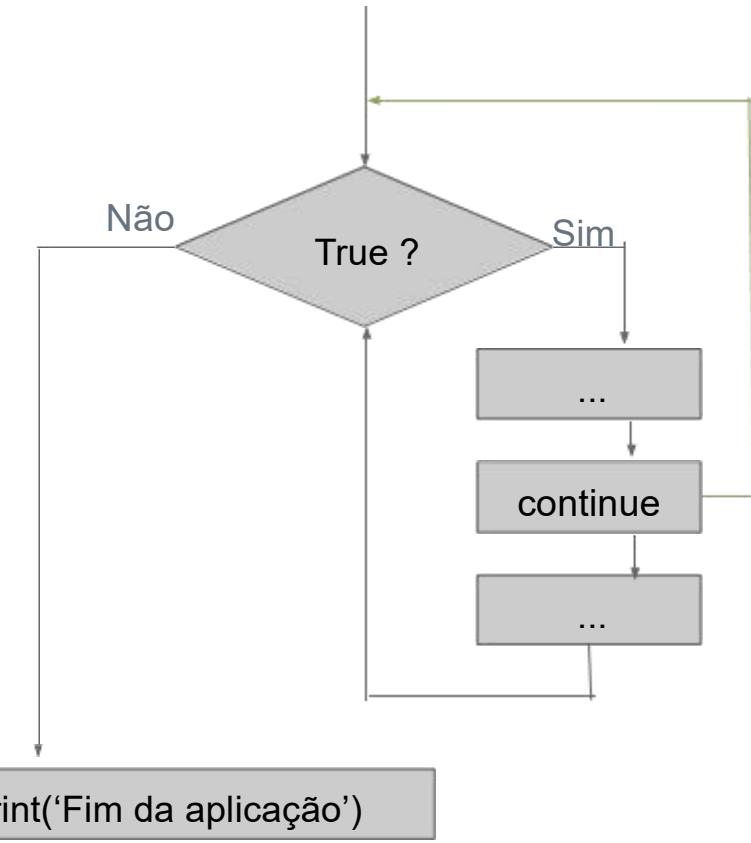
```
while True :  
    print('1 - Cadastrar')  
    print('2 - Listar')  
    print('0 - Sair')  
    numero = int(input('Digite uma Opção: '))  
    if numero == 1:  
        print('Produto cadastrado!')  
        continue  
    if numero == 2:  
        print('Lista de todos Produtos')  
        continue  
    if numero == 0:  
        break  
print('Fim da Aplicação')
```

Saída:

```
1 - Cadastrar  
2 - Listar  
0 - Sair  
Digite uma Opção: 1  
Produto cadastrado!  
1 - Cadastrar  
2 - Listar  
0 - Sair  
Digite uma Opção: 0  
Fim da Aplicação  
>>> |
```

Comando continue

```
while True :  
    print('1 - Cadastrar')  
    print('2 - Listar')  
    print('0 - Sair')  
    numero = int(input('Digite uma Opção: '))  
    if numero == 1:  
        print('Produto cadastrado!')  
        continue  
    if numero == 2:  
        print('Lista de todos Produtos')  
        continue  
    if numero == 0:  
        break  
print('Fim da Aplicação')
```



Exercícios

1. A sequência de Fibonacci é definida pela seguinte sequência: 0 1 1 2 3 5 8 13 21... Em que cada termo da sequência é obtido somando os dois termos anteriores. Escreva um programa que imprima os n primeiros termos da sequência dos números Fibonacci.

2. Escreva um programa que leia um número e verifique se é ou não primo. Para fazer essa verificação, calcule o resto da divisão do número por 2 e depois por todos os número impares até o número lido. Se o resto de uma dessas divisões for igual a zero, o numero não é primo. Observe que 0 e 1 não são primos e que 2 é o único número primo que é par.



Comando de Repetição **for**

Quando usar o for?

- As vezes temos que listar uma gama específica de valores, ou seja, estes valores têm que estar em um determinado intervalo definido.
- Diferente do **while**, onde corremos o perigo de entrar em um laço infinito, podemos definir o número de vezes e como será feita a iteração para cada item do conjunto usando o **for**.
- Então podemos dizer que o **for** é um **laço definido**, pois ele é executado um número de vezes exata.
- Já o **while** seria um **laço indefinido**, pois ele só é parado quando a condição de entrada é **False**.

Usando o for

- A maneira mais simples do uso do `for` é fazendo uma iteração em um intervalo definido dentro de uma lista.

Entrada:

```
for pessoa in ['Jõao', 'Maria', 'Carlos']:  
    print('Olá {}.'.format(pessoa))  
print('Até mais!')
```

Saída:

```
Olá Jõao.  
Olá Maria.  
Olá Carlos.  
Até mais!  
>>> |
```

Usando o for

- Outro modo de usar o **for** é definindo um intervalo explícito através da função **range**.

Entrada:

```
for numero in range(0, 5):
    print(numero)
print('Fim')|
```

Saída:

```
0
1
2
3
4
Fim
>>> |
```

Função range

- Define uma lista com um intervalo inteiro para o laço de repetição.
- Permite controlar a quantidade de iterações em um laço.
- Oferece um contador que funciona como um iterador dentro do laço.

Função range

- Pode receber até 3 parâmetros:
 - **1 parâmetro:** gera uma lista começando do zero até o antecessor do que foi definido.
 - **2 parâmetros:** gera uma lista começando do primeiro parâmetro até o antecessor segundo parâmetro, sempre iterando de um em um.
 - **3 parâmetros:** gera uma lista igual a anterior com 2 parâmetros, porém o terceiro parâmetro define o valor do incremento da lista.

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(1, 5)
[1, 2, 3, 4]
>>> range(1, 5, 2)
[1, 3]
>>> range(0, -6, -1)
[0, -1, -2, -3, -4, -5]
>>>
```

Exercícios

1. Faça um algoritmo que calcule o fatorial de um número.
2. Faça um programa que imprima todos os números ímpares entre dois números dados.
3. Faça um programa que receba a quantidade de termos e a razão de uma progressão aritmética, então imprima a soma de todos os seus termos.
4. Escreva um programa que leia um número e verifique se é ou não primo.
Para fazer essa verificação, calcule o resto da divisão do número por 2 e depois por todos os números ímpares até o número lido.
Se o resto de uma dessas divisões for igual a zero, o número não é primo.
Observe que 0 e 1 não são primos e que 2 é o único número primo que é par.

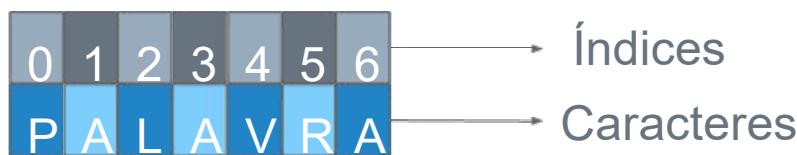


 Lorem
 ipsum dolor sit
 amet, consectetuer
 adipiscing elit, sed
 diam nonummy nibh
 euismod tincidunt ut
 laoreet dolore magna
 aliquam erat
 volutpat. Ut

Strings

Strings

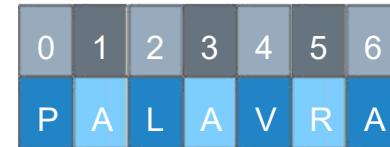
- String é um conjunto de caracteres delimitados por ‘(aspas simples) ou “ (aspas duplas) que podem formar palavras ou textos.
- Uma string pode ser entendida como uma lista de caracteres, e assim como as listas, cada elemento (caractere) possui um índice.
- Cada caractere é uma string, não existe o tipo char em Python.



```
>>> type('PALAVRA')
<class 'str'>
>>> 'PALAVRA'[0]
'P'
>>> type('PALAVRA'[0])
<class 'str'>
>>> |
```

Tamanho de uma string

- Assim como as listas, as strings possuem um comprimento.
- Para saber seu tamanho é usado a função `len()`



```
>>> string = 'PALAVRA'  
>>> len(string)  
7  
>>> |
```

Índices

- É possível acessar um determinado elemento em uma **string** através de seu **índice**.
- Os índices podem ser **positivos** ou **negativos**:
 - o Índices **positivos** começam a contagem da esquerda para direita.
 - o Índices **negativos** começam a contagem da direita para a esquerda.



```
>>> 'PALAVRA'[0]
'P'
>>> 'PALAVRA'[-1]
'A'
>>> |
```

Percorrendo uma string

- Como uma **string** é uma lista de caracteres, é possível percorrer-la usando os laços de repetição `while` e `for`.

```
>>> texto = 'Palavra'  
>>> indice = 0  
>>> while indice < len(texto):  
    print(texto[indice])  
    indice += 1
```

P
a
l
a
v
r
a
>>> |

```
>>> texto = 'Palavra'  
>>> for indice in range(len(texto)):  
    print(texto[indice])
```

P
a
l
a
v
r
a
>>> |

Operações com strings

- **+** : Junta duas ou mais strings.
- **==** : verifica se determinadas strings são iguais.
- **>** : Verifica se uma string vem depois da outra em ordem alfabética.
- **<** : Verifica se uma string vem antes da outra em ordem alfabética.

```
>>> 'Olá' + ' Mundo'  
'Olá Mundo'  
>>> 'Palavra' == 'Texto'  
False  
>>> 'Banana' > 'Caju'  
False  
>>> 'Caju' > 'Banana'  
True  
>>> '2' > '100'  
True  
>>>
```

Operador in

- Verifica se uma determinada string está contida em outra, assim retornando como resultado True ou False.
- Pode ser usado em um comando condicional.

```
>>> texto = 'O rato roeu a roda do rei de Roma.'  
>>> 'rato' in texto  
True  
>>> if 'roda' in texto:  
    print('Nesta frase contém um veículo!')
```

```
Nesta frase contém um veículo!  
>>>
```

Conversão de strings

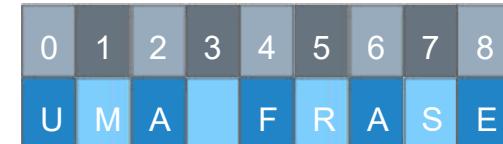
- **int()**: Converter uma string* em um número inteiro.
- **float()**: Converter uma string* em um número real.

```
>>> numero_texto = '10'  
>>> type(numero_texto)  
<class 'str'>  
>>> numero_int = int(numero_texto)  
>>> print(numero_int)  
10  
>>> type(numero_int)  
<class 'int'>  
>>> numero_float = float(numero_texto)  
>>> print(numero_float)  
10.0  
>>> type(numero_float)  
<class 'float'>  
>>> |
```

* Somente possível a conversão se a string for em formato de número. Ex: '10'.

Fatiando uma string

- É possível obter um pedaço de uma string através de um comando chamado de **slice**.
- A sintaxe do comando é a seguinte: **string[x: y]**
- O **x** corresponde ao início de onde será gerada a substring.
- O **y** compreende até onde a substring será gerada.
- Se o **y** for maior que a string a substring termina no último caractere.



```
>>> string = 'UMA FRASE'  
>>> string[0: 3]  
'UMA'  
>>> string[4: 8]  
'FRAS'  
>>> string[4: 40]  
'FRASE'  
>>> |
```

Métodos de uma string

- Toda **string** possui um conjunto de métodos ou funções já inclusos na linguagem que nos ajudam a desempenhar determinadas tarefas.

```
>>> dir('string')
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
 '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__ge__
tnewargs__',
 '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__',
 '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__re_
duce_ex__',
 '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__',
 '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count',
 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index',
 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'is_
numeric',
 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust',
 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'r_
just',
 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith',
 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
>>> |
```

Métodos de uma string

- `lower()`: Transforma todos os caracteres da string em minúsculos.
- `upper()`: Transforma todos os caracteres da string em maiúsculos.
- `capitalize()`: Transforma a primeira letra da string em maiúscula e o resto em minúsculas.
- `title()`: Transforma a primeira letra de cada palavra da string em maiúscula.

```
>>> texto = 'um texto QualQuer'  
>>> texto.lower()  
'um texto qualquer'  
>>> texto.upper()  
'UM TEXTO QUALQUER'  
>>> texto.capitalize()  
'Um texto qualquer'  
>>> texto.title()  
'Um Texto Qualquer'  
>>>
```

Métodos de uma string

- **find()**: Procura da esquerda para direita a ocorrência da primeira string desejada, então retorna sua posição.
- **rfind()**: Procura da direita para esquerda a primeira ocorrência da string desejada, então retorna sua posição.

0	1	2	3	4	5	6
P	A	L	A	V	R	A

```
>>> texto = 'PALAVRA'  
>>> texto.find('A')  
1  
>>> texto.rfind('A')  
6  
>>> |
```

Métodos de uma string

- **replace()**: Busca e substitui uma string pela outra.
- **lstrip()**: Remove espaços em branco à esquerda.
- **rstrip()**: Remove espaços em branco à direita.
- **strip()**: Remove os espaços ambos da esquerda como da direita.

```
>>> fruta = 'Pêra'  
>>> fruta_sem_acento = fruta.replace('ê', 'e')  
>>> print(fruta_sem_acento)  
Pera  
>>> com_espaco = ' texto'  
>>> com_espaco.lstrip()  
'texto'  
>>> com_espaco.rstrip()  
' texto'  
>>> com_espaco.strip()  
'texto'  
>>> |
```

Exercícios

1. Escreva um programa que leia duas Strings. Verifique se a segunda ocorre dentro da primeira e imprima a posição de início.
2. Escreva um programa que leia duas Strings e gere uma terceira com os caracteres comuns às duas strings lidas.
3. Escreva um programa que leia duas strings e gere uma terceira apenas com os caracteres que aparecem em umas delas.

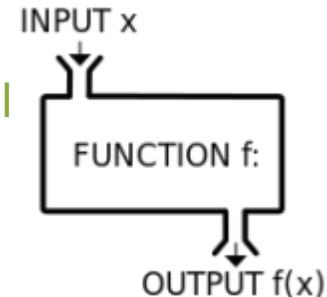




Funções

Funções

- Uma função em Python é uma maneira de organizar sua aplicação, onde através de um conjunto de código **reusável** é possível receber um ou mais argumentos como entrada, computar algo, então retornar o resultado esperado.
- Existem dois tipos de funções em Python.
 - **Funções Built-in**, que são aquelas que já estão feitas e disponíveis para uso dentro da linguagem, ex: `print()`, `int()`, `float()`...
 - **Funções construídas pelo próprio programador**, que são aquelas definidas de acordo com a necessidade do mesmo.



Construindo funções

- Para definir uma função é usada a palavra chave reservada **def** seguida pelo nome da função e **()**.
- O corpo da função deve ser indentada após sua definição.
- O que está no corpo da função é o que será executado por ela.
- Após construir a função é preciso chamá-la para que a mesma seja executada.
- Uma vez definida é possível chamá-la quando quiser.

Definindo a função

```
>>> def ola_mundo():  
     print('Ola Mundo')
```

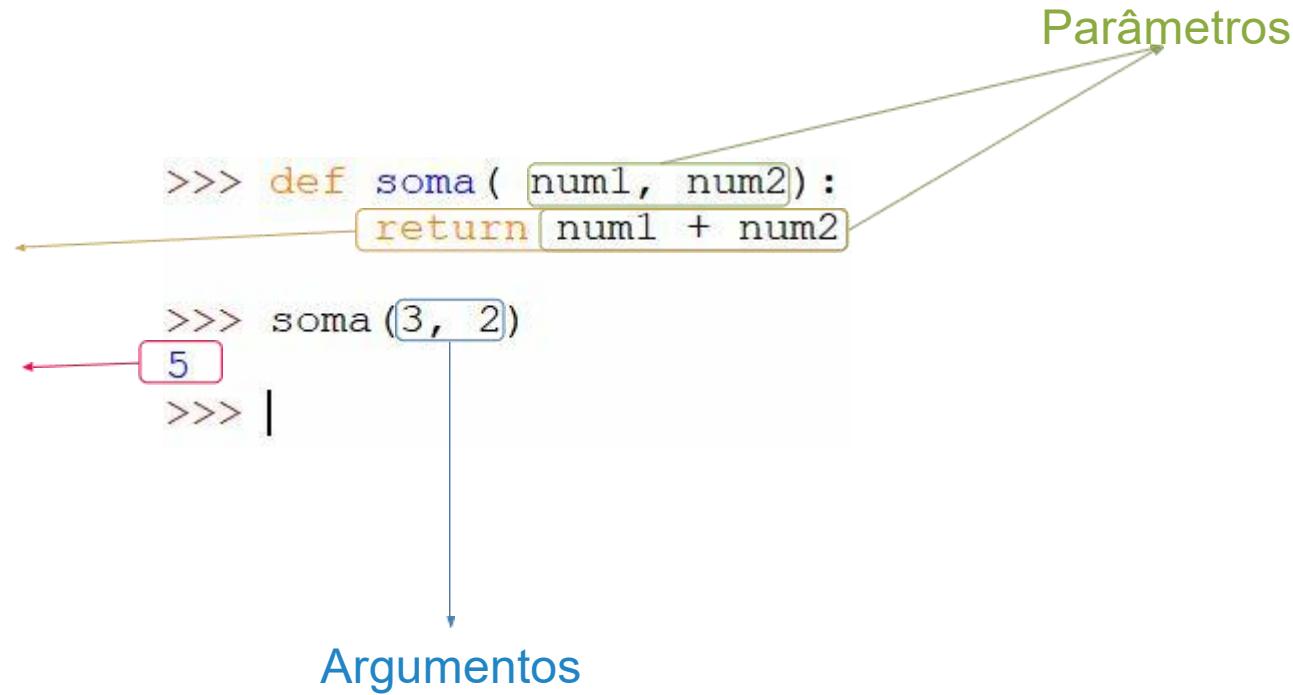
```
>>> ola_mundo()  
Ola Mundo  
>>> |
```

Chamando a função

Esqueleto de uma função

- Uma função pode conter **argumentos**, **parâmetros**, retornar ou não algo.
- **Argumento** é um valor de entrada, que dependendo dele a mesma função pode executar tipos diferentes de trabalho.
- **Parâmetro** é uma variável que pode ser manipulada dentro da função através de um argumento passado pela mesma.
- Se a função tiver mais de um argumento ou parâmetro, eles são separados por **vírgulas**.
- Para retornar um valor pode-se usar a palavra reservada **return**.
- Funções que não retornam um valor são chamadas de ‘void’.

Esqueleto de uma função



Funções Built-in

- **min()**: Recebe uma lista como argumento e retorna o menor valor dentro de uma lista.
- **max()**: Recebe uma lista como argumento e retorna o maior valor dentro de uma lista.
- **sum()**: Recebe uma lista como argumento então soma todos os valores dentro dela.
- **abs()**: Recebe um número como argumento e retorna seu módulo.

```
>>> lista = [2, 6, 8, 10]
>>> min(lista)
2
>>> max(lista)
10
>>> sum(lista)
26
>>> abs(-4)
4
>>> |
```

Funções Recursivas

- Funções podem chamar outras funções dentro delas.
- Quando uma função chama ela mesma isto é chamado de recursividade.

Fatorial

$n! = 1,$ se $n = 0$
 $n * (n - 1)!,$ se $n > 0$

```
>>> def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

```
>>> fatorial(0)
1
>>> fatorial(4)
24
>>> |
```

Bibliotecas de funções em Python

- Biblioteca é um conjunto de funções que podem ser usadas pelo programador para facilitar seu trabalho.
- Em Python já existe uma série de bibliotecas pré definidas.
- Para usá-las é preciso usar o comando **import**, então especificar o nome, ou caminho da biblioteca.
- Se quisermos usar apenas uma função específica de uma biblioteca é usado o comando **from**

Biblioteca math

- Biblioteca com diversas funções matemáticas:
 - sin, cos, tan, asin, acos, atan
 - log, log2, log10
 - pow, sqrt
 - floor, ceil
 - factorial

```
>>> import math
>>> math.pow(2, 10)
1024.0
>>> math.sqrt(4)
2.0
>>> math.factorial(4)
24
>>> |
```



```
>>> from math import pow
>>> pow(10, 2)
100.0
>>> |
```

*Mais informações sobre a biblioteca math:

<https://docs.python.org/3.4/library/math.html>

Importando módulos próprios

- É possível também usar módulos próprios em Python, assim sua aplicação ficará mais enxuta, legível e com uma reusabilidade maior.

```
def minha_funcao():
    return 'Aqui está minha função!'
```

arquivo.py

```
Aqui está minha função!
>>> |
```

Área de Transferência		Organizar	Novo
↑	Este Computador > Área de Trabalho > python		
	Nome	Data de modificaç...	Tipo
áido	arquivo.py	27/02/2018 17:05	Python File
e Traba	principal.py	27/02/2018 17:13	Python File
oads			
mentos			

Saída
do
principal.py

```
import arquivo
print(arquivo.minha_funcao())|
```

principal.py

Importando bibliotecas

SorteandoOrdem.py x

```
1 # Sortear a ordem de apresentação de trabalhos dos alunos.  
2 # Faça um programa que Leia o nome dos quatro alunos e mostre a ordem sorteada.  
3 import math  
4 from random import shuffle  
5 n1 = str(input('Primeiro aluno: '))  
6 n2 = str(input('Segundo aluno: '))  
7 n3 = str(input('Terceiro aluno: '))  
8 n4 = str(input('Quarto aluno: '))  
9 lista = [n1, n2, n3, n4]  
10 shuffle(lista) # shuffle serve para embaralhar  
11 print('A ordem de apresentação será: ')  
12 print(lista)
```

Exercícios

1. Escreva uma função que valide se o numero lido está entre o mínimo e o máximo passados como parâmetros.
2. Escreva uma função que receba uma lista e um valor como parâmetros, e retorne um boolean indicando se o valor está contido nessa lista.
3. Escreva uma função para verificar se um número é divisível por outro.



Listas

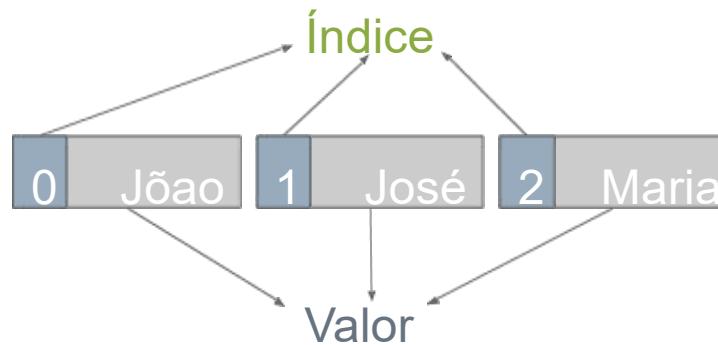
Listas

- Uma **lista** é uma **coleção** de alguma coisa.
- Em Python podemos guardar vários valores em uma única variável com o uso de **listas**.

```
numeros = [1, 4, 6, 7, 8]
amigos = ['Jão', 'Maria', 'José']|
```

Listas

- As listas são delimitadas por colchetes []
- Podemos acessar um determinado elemento da lista através de seu **índice**, este dentro dos colchetes.
- Os **índices** sempre começam do zero.



```
>>> amigos = ['Jôao', 'José', 'Maria']
>>> print(amigos[1])
José
>>> |
```

Percorrendo uma lista

- Podemos percorrer e listar todos os elementos de uma lista através do laço de repetição **for**.

Entrada:

```
amigos = ['José', 'Jão', 'Maria']

for amigo in amigos:
    print('{} é meu amigo!'.format(amigo))

print('Até mais.')|
```

Saída:

```
José é meu amigo!
Jão é meu amigo!
Maria é meu amigo!
Até mais.
>>> |
```

Operações de uma lista

- Dentre as operações que podemos utilizar nas listas estão as seguintes:
 - **Concatenação(+)** junta os elementos de duas ou mais listas.
 - **Concatenação Múltipla(*)** gera várias cópias, assim concatenando todos os elementos em uma nova lista (**somente possível com a multiplicação por um número inteiro**).
 - **Existência(in:)** verifica se determinado elemento está contido na lista, então retorna um valor booleano, True caso verdadeiro e False caso falso.

Operações de uma lista

```
>>> lista = ['Carro', 'Moto', 200, 100]
>>> lista2 = ['Ônibus', 'Bike']
>>> lista + lista2
['Carro', 'Moto', 200, 100, 'Ônibus', 'Bike']
>>> lista * 2
['Carro', 'Moto', 200, 100, 'Carro', 'Moto', 200, 100]
>>> 'Moto' in lista
True
>>> |
```

Tamanho de uma lista

- Se quisermos saber o tamanho de uma lista é possível utilizar a função `len()`.
- Ela recebe uma lista como parâmetro, então retorna o valor com o número de itens.
- Também é possível percorrer uma lista utilizando a função `len()`.

```
>>> carros = ['Civic', 'Gol', 'Corsa']
>>> len(carros)
3
>>> for i in range(len(carros)):
        print(carros[i])

Civic
Gol
Corsa
>>>
```

Métodos de uma lista

- Métodos, ou funções, são um conjunto de código que desempenham uma determinada ação.
- Para auxiliar com determinados tipos de operações, as listas em Python possuem um conjunto de métodos já pré-definidos pela linguagem.
- Podemos verificar os métodos disponíveis em uma lista através da função `dir()`.

```
>>> lista = [1, 2, 3]
>>> dir(lista)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__delslice__',
 '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
 '__getslice__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__',
 '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattribute__', '__setitem__',
 '__setslice__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'count',
 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>> |
```

Métodos de uma lista

- **count()**
 - Através do método count podemos saber a ocorrência de determinado valor dentro da lista.

```
>>> numeros = [1, 3, 4, 54, 2, 64, 2]
>>> numeros.count(3)
1
>>> numeros.count(2)
2
>>> |
```

Métodos de uma lista

- **index()**
 - Retorna a posição, índice, de um elemento em uma lista.
 - Caso se tenha elementos iguais, o método retorna a posição da primeira ocorrência do elemento na lista.

```
>>> numeros = [1, 3, 4, 54, 2, 64, 2]
>>> numeros.index(3)
1
>>> numeros.index(2)
4
>>>
```

Métodos de uma lista

- Inserindo e deletando elementos de uma lista

```
>>> numeros = [1, 3, 6, 5]
>>> print(numeros)
[1, 3, 6, 5] → 1 | 3 | 6 | 5
>>> # insert(2, 8) - Insere na posição 2 o número 8
>>> numeros.insert(2, 8)
>>> print(numeros)
[1, 3, 8, 6, 5] → 1 | 3 | 6 | 8 | 5
>>> # append(10) - Insere o número 10 ao final da lista
>>> numeros.append(10)
>>> print(numeros)
[1, 3, 8, 6, 5, 10] → 1 | 3 | 6 | 8 | 5 | 10
>>> # pop(2) - Remove da lista o elemento da posição 2 e o retorna
>>> numeros.pop(2)
8
>>> print(numeros)
[1, 3, 6, 5, 10] → 1 | 3 | 6 | 5 | 10
>>> # pop() - Remove o último elemento da lista e o retorna
>>> numeros.pop()
10
>>> print(numeros)
[1, 3, 6, 5] → 1 | 3 | 6 | 5
>>> # remove(3) - Remove o elemento 3 da lista
>>> numeros.remove(3)
>>> print(numeros)
[1, 6, 5] → 1 | 6 | 5
>>>
```

Métodos de uma lista

- Ordenação
 - `sort()`- Ordena de forma crescente.
 - `sort(reverse = True)`- Ordena de forma decrescente.
 - `reverse()`- Inverte a ordem dos valores.

```
>>> numeros = [2, 5, 1, 6]      >>> amigos = ['Carlos', 'Maria', 'Alvaro'] >>> misto = [2, 4, 'c', 'a']
>>> print(numeros)            >>> print(amigos)           >>> print(misto)
[2, 5, 1, 6]                  ['Carlos', 'Maria', 'Alvaro'] [2, 4, 'c', 'a']
>>> numeros.sort()          >>> amigos.sort()         >>> misto.sort()
>>> print(numeros)          >>> print(amigos)          >>> print(misto)
[1, 2, 5, 6]                  ['Alvaro', 'Carlos', 'Maria'] [2, 4, 'a', 'c']
>>> numeros.sort(reverse = True) >>> amigos.sort(reverse = True) >>> misto.sort(reverse = True)
>>> print(numeros)          >>> print(amigos)          >>> print(misto)
[6, 5, 2, 1]                  ['Maria', 'Carlos', 'Alvaro'] ['c', 'a', 4, 2]
>>> numeros.reverse()       >>> amigos.reverse()       >>> misto.reverse()
>>> print(numeros)          >>> print(amigos)          >>> print(misto)
[1, 2, 5, 6]                  ['Alvaro', 'Carlos', 'Maria'] [2, 4, 'a', 'c']
>>>
```

Compreensão de listas

- Maneira compactada de gerar uma lista que obedece a seguinte sintaxe:
 - [saída|laço de repetição|filtro]

```
>>> print([ pares for pares in range(11) if pares % 2 == 0 and pares != 0])
[2, 4, 6, 8, 10]
>>>
```

Exercícios

1. Escreva um programa que adicione em uma lista todos os números lidos até ser digitado 0. Ao final, exiba o conteúdo da lista, bem como os índices. Utilize a função enumerate para mostrar os índices.
2. Faça um programa que leia 10 números inteiros e os armazene em uma lista. Em seguida, imprima a lista.
3. Crie um programa que leia 10 nomes de pessoas e os armazene em uma lista. Em seguida, imprima a lista.
4. Construa um menu com opções de 1 - Inserir, 2 - Remover, 3 - Consultar, bem como suas funcionalidades.



Tuplas

Tuplas

- Assim como as listas, **tupla** é um conjunto de valores agrupados, estes que podem ser textos ou números.
- Diferente das listas, a **tupla** é uma estrutura de dados estática, ou seja, os valores contidos nela não podem ser modificados

```
>>> tupla_numeros = (1, 4, 6)
>>> tupla_numeros
(1, 4, 6)
>>> tupla_nomes = ('Maria', 'Carlos')
>>> tupla_nomes
('Maria', 'Carlos')
>>> |
```

Tuplas x Listas

- Comparada com as listas, as **tuplas** têm um conjunto de operações bem mais reduzido.

```
>>> dir(list)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__',
 '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
 '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__',
 '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__',
 '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__',
 '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear',
 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
]
>>> dir(tuple)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
 '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
 '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__',
 '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__rmul__', '__setattr__', '__sizeof__',
 '__str__', '__subclasshook__', 'count', 'index']
>>> |
```

Definindo uma tupla

- Uma tupla pode ser definida com seus elementos sendo separados por **vírgulas** e estando, ou não, delimitados entre **parênteses ()**.
- Para inicializar uma tupla vazia é só preciso usar parênteses ().
- Pode-se ter uma lista dentro de uma tupla, e vice-versa.
 - As listas dentro das tuplas podem ser modificadas.

```
>>> t1 = ()  
>>> t1  
()  
>>> t2 = ('Carro', 38, 'Moto', 43, [1, 2, 3])  
>>> t2  
('Carro', 38, 'Moto', 43, [1, 2, 3])  
>>> t3 = 2, 5, 4  
>>> t3  
(2, 5, 4)
```

Operadores um tupla

- É possível também usar os operadores maior(>) ou menor(<) para comparar os elementos de uma tupla.

```
>>> (1, 3, 5) > (3, 1, 4)
False
>>> (1, 2, 3) < (2, 4, 5)
True
>>> ('Carro', 'Lancha', 'Avião') > ('Bola', 'Sapato', 'Zero')
True
>>> |
```

Percorrendo um tupla

- É possível acessar os elementos de uma tupla do mesmo modo que uma lista, **elemento[índice]**.
- Para percorrer uma tupla é usado o mesmo método que em listas, com laços de repetição.

```
>>> lista_notas = (8, 4, 7.8, 9)
>>> lista_notas[3]
9
>>> for elementos in lista_notas:
        print(elementos)
```

```
8
4
7.8
9
>>>
```

Usando tuplas

- Uma função que retorna vários valores na verdade retorna uma tupla.
- É possível armazenar estes valores em uma variável, assim atribuindo a ela esta tupla.
- O acesso aos elementos de uma tupla é bem mais rápido do que de uma lista.
- Além disto, a atribuição múltipla é feita usando tuplas.

```
>>> def operacoes(x, y):
    return x + y, x - y, x * y, x / y

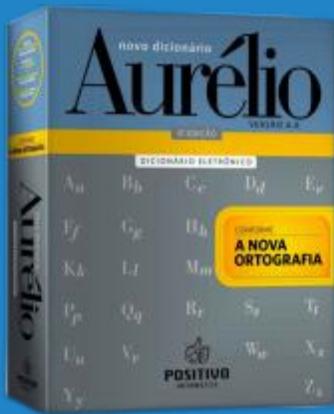
>>> resultado = operacoes(3, 1)
>>> type(resultado)
<class 'tuple'>
>>> resultado
(4, 2, 3, 3.0)
>>>

>>> x, y = 34, 4
>>> x
34
>>> y
4
>>> |
```

Exercícios

1. Crie uma tupla preenchida com os 20 primeiros colocados da Tabela do Campeonato Brasileiro de Futebol, na ordem de colocação. Depois mostre:
 - a) Os 5 primeiros times.
 - b) Os últimos 4 colocados.
 - c) Times em ordem alfabética.
 - d) Em que posição está o time do São Paulo.

2. Desenvolva um programa que leia quatro valores pelo teclado e guarde-os em uma tupla. No final, mostre:
 - A) Quantas vezes apareceu o valor 9.
 - B) Em que posição foi digitado o primeiro valor 3.
 - C) Quais foram os números pares.



Dicionários

Dicionários

- Dicionários são estruturas de dados que possuem dois campos, chave e valor.
- Dicionários são mutáveis, e seu uso se dá através de colchetes {} e com a seguinte sintaxe: dicionário = {chave: valor, chave: valor} com seus elementos separados por vírgula.
- Para adicionar um elemento em um dicionário é só especificar sua chave e valor.

```
>>> pessoa = {'nome': 'Carlos', 'sobrenome': 'Santos'}
>>> pessoa
{'sobrenome': 'Santos', 'nome': 'Carlos'}
>>> pessoa['nome_do_meio'] = 'Silva'
>>> pessoa
{'sobrenome': 'Santos', 'nome_do_meio': 'Silva', 'nome': 'Carlos'}
>>>
```

Definindo dicionários

- Para se inicializa um dicionário vazio só é preciso usar **colchetes{}**.
- Elementos que se podem usar dentro de um dicionário:
 - **Chave**: qualquer um, exceto listas e os dicionários.
 - **Valor**: Qualquer um, incluindo dicionários.

```
>>> dicionario_vazio = {}
>>> dicionario_vazio
{}
>>> dicionario = {'lista': [1, 2], (1, 2): 'tupla', 'numero': 32, 1: 'inteiro',
'dicionario': {'chave': 'valor'}}
>>> dicionario
{(1, 2): 'tupla', 1: 'inteiro', 'lista': [1, 2], 'dicionario': {'chave': 'valor'},
'numero': 32}
>>>
```

Métodos de um dicionário

- Assim como as outras estruturas de dados, já existe uma série de funções prontas que podemos usar.

```
>>> dir(dict)
['__class__', '__cmp__', '__contains__', '__delattr__', '__delitem__', '__doc__',
 '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__',
 '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__ne__',
 '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setitem__',
 '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get',
 'has_key', 'items', 'iteritems', 'iterkeys', 'itervalues', 'keys', 'pop', 'po
pitem', 'setdefault', 'update', 'values', 'viewitems', 'viewkeys', 'viewvalues']
>>>
```

Métodos de um dicionário

- **pop()**: Recebe uma chave como parâmetro, retorna seu valor e remove este item do dicionário.
- **popitem()**: Retorna e remove um elemento aleatório, chave e valor, do dicionário.
- **clear()**: Remove todos os dados de um dicionário.

```
>>> dicionario = {'chave1': 'valor1', 'chave2':'valor2',  
'chave3':'valor3'}  
>>> dicionario  
{'chave1': 'valor1', 'chave3': 'valor3', 'chave2': 'valo  
r2'}  
>>> dicionario.pop('chave1')  
'valor1'  
>>> dicionario  
{'chave3': 'valor3', 'chave2': 'valor2'}  
>>> dicionario.popitem()  
('chave3', 'valor3')  
>>> dicionario  
{'chave2': 'valor2'}  
>>> dicionario.clear()  
>>> dicionario  
{}  
>>>
```

Métodos de um dicionário

- **get()**: Recebe uma chave como parâmetro, retorna seu valor.
- **keys()**: Retorna uma lista contendo todas as chaves do dicionário.
- **values()**: Retorna uma lista contendo todos os valores do dicionário.
- **items()**: Retorna uma lista com tuplas contendo a chave e o valor de cada item do dicionário.

```
>>> dicionario = {'chave1': 'valor1', 'chave2': 'valor2', 'chave3': 'valor3'}
>>> dicionario.get('chave2')
'valor2'
>>> dicionario.keys()
['chave1', 'chave3', 'chave2']
>>> dicionario.values()
['valor1', 'valor3', 'valor2']
>>> dicionario.items()
[('chave1', 'valor1'), ('chave3', 'valor3'), ('chave2', 'valor2')]
>>> |
```

Percorrendo um dicionário

- Para se percorrer um dicionário é usado o mesmo método que em listas e tuplas, com a única diferença de retornar sua chave e valor.

```
>>> dicionario = {'chave1': 'valor1', 'chave2': 'valor2', 'chave3': 'valor3'}  
>>> for chave in dicionario:  
    print(chave, '=', dicionario[chave])  
  
('chave1', '=', 'valor1')  
('chave3', '=', 'valor3')  
('chave2', '=', 'valor2')  
>>> |
```

Ordenando elementos em um dicionário

- Como dicionários possuem chave e valor, podemos ordená-los ou por suas chaves, ou pelos seus valores.
- Porém, diferente das listas e tuplas, não existe um método já pronto para ordenar dicionários.

Ordenando elementos em um dicionário

- Ordenando pelas chaves:

```
>>> dicionario = {'a': 30, 'c': 29, 'b': 2, 'e': 10}
>>> dicionario_ordenado_chaves = dicionario.keys()
>>> dicionario_ordenado_chaves.sort()
>>> for chave in dicionario_ordenado_chaves:
    print(chave, '=', dicionario[chave])

('a', '=', 30)
('b', '=', 2)
('c', '=', 29)
('e', '=', 10)
>>>
```

Ordenando elementos em um dicionário

- Ordenando pelos valores:

```
>>> dicionario = {'a': 30, 'c': 29, 'b': 2, 'e': 10}
>>> for item in sorted(dicionario, key = dicionario.get):
    print (item, '= ', dicionario[item])

('b', '= ', 2)
('e', '= ', 10)
('c', '= ', 29)
('a', '= ', 30)
>>> |
```

Material complementar

- Texto: Por que programar é o novo ‘aprender inglês’
 - <https://www.nexojornal.com.br/expresso/2017/04/02/Por-que-programar-%C3%A9-o-novo-aprender-ingl%C3%AAs>
- Vídeo: Por que todos deveriam aprender a programar?
 - <https://www.youtube.com/watch?v=mHW1Hsqlp6A>
- Vídeo: Curso em Vídeo: Curso Python #01 - Seja um Programador
 - <https://www.youtube.com/watch?v=S9uPNppGsGo>



Obrigado e até a próxima!



igsantos1996@gmail.com