



Powered by  serasa™



# Porque estudar Python???

Conhecendo a  
Linguagem

# O que é Python?

Linguagem de programação criada em 1991  
por um programador pesquisador da  
Holanda chamado Guido Von Rossum



- Objetivo do projeto da linguagem eram **produtividade e legibilidade**
- Produzir código bom e fácil de manter de maneira rápida.



# Nome da linguagem

Guido queria que o nome da linguagem fosse marcante e forte, mas não fazia questão que o nome possuísse um significado profundo.

Foi então que Guido usou a primeira coisa que veio a sua cabeça: Monty Python's Flying Circus, uma série de comédia britânica.



# Símbolo da linguagem

O símbolo da cobra surgiu quando a editora O'Reilly — que possui a tradição de utilizar animais nas capas de seus livros — sugeriu colocar uma cobra píton na capa do seu primeiro livro "Programming Python".



# PEP

Guia de estilo para código Python

<https://www.python.org/dev/peps/pep-0008/>

Poema

<https://www.python.org/dev/peps/pep-0020/#id3>

```
# Correct:  
import os  
import sys
```

```
# Wrong:  
import sys, os
```





# 5 motivos para aprender Python

# Ótimo para iniciantes

O quê faz uma linguagem de programação ser “fácil de aprender”?

Simplicidade e velocidade de aprendizagem:

- Linguagem de alto nível (Não precisa conhecer detalhes do hardware)
- Menos códigos para construir tarefas básicas, quando comparado com outras linguagens
- Por ser simples, quem começa a programar em Python rapidamente já estará desenvolvendo programas complexos e robustos.





# Versatilidade

Python possui mais de 220 mil bibliotecas de terceiros!

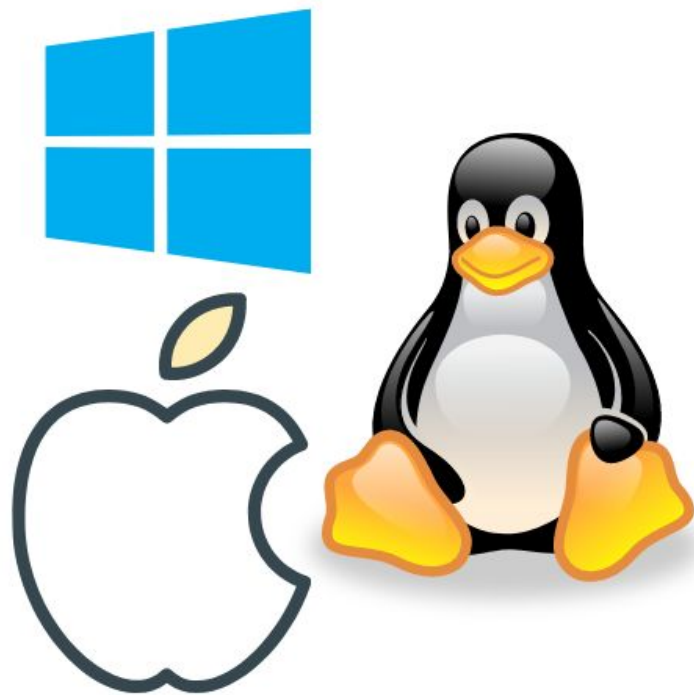
Elas são muito úteis para:

- Desenvolvimento Web, Mobile e Desktop
- Processamento de imagens
- Data Science
- Machine Learning



# Multiplataforma

- Por ser uma linguagem interpretada e não compilada, python roda em diferentes plataformas (Windows, Linux, macOS) sem precisar alteração de código.



# Comunidade

Python possui uma comunidade ativa e vibrante espalhada por todo o mundo!

Como a linguagem é de código aberto, todos os usuários estão dispostos a contribuir.



# Mercado

Quem usa Python?



E aiii, vamos aprender Python????





# Introdução a Python

Primeiros passos na  
linguagem

# Porque Python?

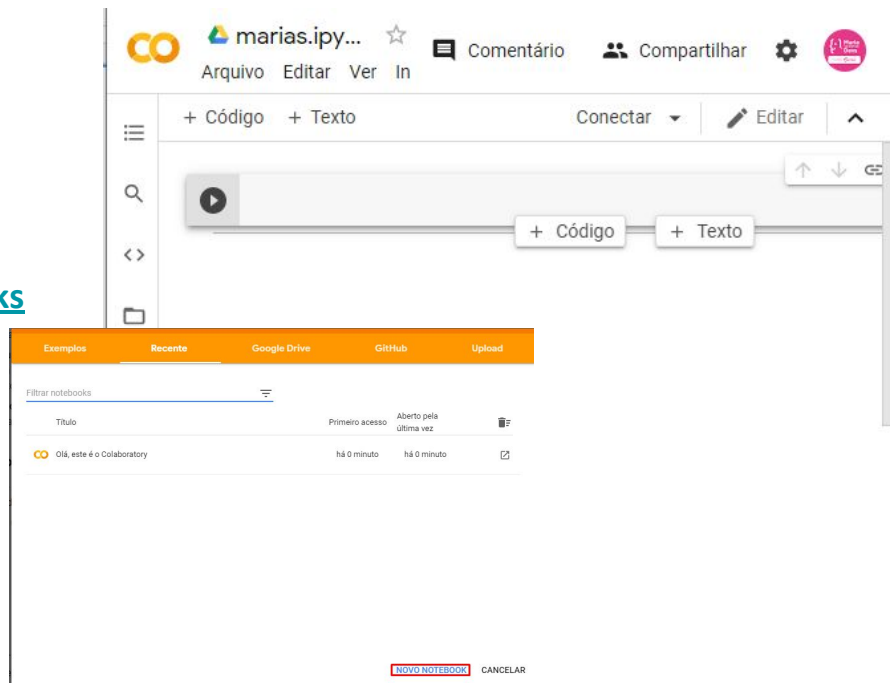
Recapitulando a sessão anterior..

- **Acessível:** fácil de aprender, legível.
- **Open source:** documentação disponível na internet.
- **Expressivo/Produtivo:** pouco código, muito resultado
- **Versátil:** para web, para desktop, aplicações científicas...



# Ferramentas utilizadas neste workshop

- Colab da Google:
  - <https://colab.research.google.com/notebooks/intro.ipynb>
  - Fazer login / Criar um notebook





# Tipado dinamicamente

Não é necessário declarar uma variável e definir o tipo de seu dado: o tipo de dado será associado em tempo de execução de acordo com o valor atribuído para aquela variável.

*Exemplo Python*

```
nome = "Alini"
print(nome)
```

Alini

*Exemplo Java*

```
8
9 public class Main
10 {
11     public static void main(String[] args) {
12         String nome = "Alini";
13         System.out.println("Hello " + nome);
14     }
15 }
16
```



# Tipos básicos em Python

Python	O que significa?	Exemplos
bool	Valores <u>booleanos</u> : Verdadeiro ou Falso	eh_maior = 22 > 10 status = False
float	Valores com ponto flutuante	nota = 7.5 preco = 30.55
int	Valores inteiros: ..., -2, -1, 0, 1, 2, 3, ...	alunos_presentes = 20 numero_de_slides = 50
str	Textos: python considera texto tudo que estiver entre aspas (simples ou duplas)	dia = "sábado" assunto = 'python básico' semana= "5"
list	Sequência de objetos separados por vírgula, e dentro de colchetes.	devs = ["ana", "amanda", "andorinha"] idades = [19, 38, 24]



# Operadores aritméticos

É possível usar o python como  
uma calculadora

Operador	Descrição
+	adição
-	subtração
*	multiplicação
/	divisão
//	divisão inteira
**	exponenciação
%	módulo



# Operadores relacionais

O interpretador também pode ser utilizado para operações relacionais.

Operador	Descrição
==	igual
!=	diferente
<	menor que
>	maior que
<=	menor ou igual
>=	maior ou igual



# Variáveis

Em Python os nomes das variáveis podem ser compostos de letras maiúsculas, minúsculas, underscore e números, iniciando com letras ou underscore (\_).

```
nome = "Alini"
valor = 10
expressao = 1 + valor / 3

print(nome)
print(valor)
print(expressao)
```

Alini  
10  
4.333333333333334



# Variáveis e tipos de informação

Há um comando para descobrir o tipo de uma variável: `type()`.

```
nome = "Alini"
valor = 10
expressao = 1 + valor / 3

print(type(nome))
print(type(valor))
print(type(expressao))

<class 'str'>
<class 'int'>
<class 'float'>
```



# Exercícios em Python [1]

Você possui uma variável `a = 10` e uma variável `b = 5`, e precisa trocar o valor das duas. Como você faria?



# Exercícios em Python [1]

Normalmente você criaria mais uma variável, não?

Python permite que você faça essa operação em uma única linha ->>>

```
a = 10
b = 5
a, b = b, a

print(a)
print(b)
```





# Entradas e saídas

Perceberam que no exercício anterior eu utilizei uma função chamada print, para apresentar a informação da variável na tela?

Print é uma função de saída, e também possuímos uma função de entrada, chamada input.

**\*\*Todos os dados recebidos no input são textos**



Textos devem estar entre aspas, simples ou duplas

```
Entrada:
    input("Texto opcional que será exibido para o usuário")
Saída:
    print(f"Textos a serem exibidos e {variaveis}")
```



Função f-string do python

## f-string

[Facilitador de formatação,  
substituem o que está entre  
{chaves pelo seu valor}]



# Entradas em Python

São consideradas textos

Se desejamos considerar outro tipo, precisamos pedir para que o python faça a conversão.

Exemplo: `variavel = int(variavel)`

Tipo	Comando de conversão
Booleano(bool)	<code>bool(variavel)</code>
Inteiro(int)	<code>int(variavel)</code>
Real(float)	<code>float(variavel)</code>
Texto(str)	<code>str(variavel)</code>



# Entradas em Python

As operações de leitura e conversão de dados podem ser compostas, como por exemplo:

```
valor = float(input("Digite o valor escolhido: "))
```

Há uma restrição:

- A conversão só pode ser aplicada a um valor por vez.
- Em Python, todos os valores digitados em uma mesma linha serão considerados o mesmo texto.



# Exercícios em Python [2]

Faça um programa que pergunte que dia é, e diga quantos faltam até o final do mês. Considere o mês com trinta e um dias.



# Exercícios em Python [2]

```
dia = input("Dígite o dia do mês: ")
dia = int(dia)
restantes = 31 - dia
print(f"faltam {restantes} para o fim do mês!")
```

```
Dígite o dia do mês: 20
faltam 11 para o fim do mês!
```



# Exercícios em Python [3]

Faça um programa que receba o número de brindes do Maria vai com as Devs que a Jéssica vai mandar fazer, e imprima quanto de entrada ela irá que pagar para o fabricante.

Considere que cada brinde custa R\$25,00, e a entrada para o fabricante é de 50% do valor total.



# Exercícios em Python [3]

```
valor = 25
quantidade = int(input("Digite o número de brindes a serem confeccionados:"))
entrada = (valor * quantidade)/2
print(f"O valor da entrada do pagamento a ser pago para o fabricante é R$ {entrada:.2f}")
```

```
Digite o número de brindes a serem confeccionados:60
O valor da entrada do pagamento a ser pago para o fabricante é R$ 750.00
```



# Exercícios em Python [4]

A Serasa irá gerar um minicurso de aprendizagem financeira e está planejando um coffee break às 15 horas.

Leia o número de participantes do minicurso e imprima: quantos litros de refrigerante teremos que comprar, quantas coxinhas e o valor que iremos gastar.

Considere que cada participante toma 400ml de refrigerante, come 10 coxinhas, que cada litro de refrigerante custa R\$4,00 e cada coxinha custa R\$0,50.





# Exercícios em Python [4]

```
participantes = int(input("Digite o número de participantes do minicurso:"))
litros_refri = 0.4 * participantes
preco_refri = litros_refri * 4

coxinhas = 10 * participantes
preco_coxinha = coxinhas * 0.50

custo_do_pedido_geral = preco_coxinha + preco_refri

print(f"Serão necessários comprar {litros_refri} litros de refrigerante.")
print(f"Serão necessários comprar {coxinhas} unidades de coxinhas.")
print(f"Iremos gastar R${custo_do_pedido_geral:.2f} ")
```

```
Digite o número de participantes do minicurso:10
Serão necessários comprar 4.0 litros de refrigerante.
Serão necessários comprar 100 unidades de coxinhas.
Iremos gastar R$66.00
```



# Comentários

Em Python, um comentário é tudo o que há numa linha depois de uma "#", assim:

```
print("Exemplo de código que não vai ser ignorado pelo Python")  
#Comentário que o Python não vai processar ^-^
```



# Dicas

No Python você pode incrementar ou decrementar um valor e salvar o resultado na mesma variável.

```
x = 10
x += 1 # Mesma coisa que x = x + 1
x -= 3 # Mesma coisa que x = x - 3
print(x)
```

8



# Dicas

## Separando texto com split()

Python trata a entrada de dados por linha, e portanto que assume que cada linha conterà uma única variável.

Para contornar este problema, utilizamos o comando `split()`, com ele podemos determinar um caractere de separação para os nossos dados.



# Dicas

## Separando texto com split()

- Se você não determinar nenhum caractere de espaçamento para o split(), ele assumirá que é um espaço em branco.
- Como você dividirá sua entrada em duas, poderá armazená-la em duas variáveis diferentes! Se você não colocar variáveis suficientes, python criará uma **lista\***.

```
primeiro_nome, segundo_nome = input("Dígite dois nomes:").split()

print(f"Primeiro nome {primeiro_nome}")
print(f"Segundo nome {segundo_nome}")

Dígite dois nomes: Pri Alini
Primeiro nome Pri
Segundo nome Alini
```



# Dicas

## Separando texto com split()

Na conversão de tipos de dados de entrada, para dividir a entrada em mais de uma variável e já transformar ela para o tipo desejado, utilizamos um comando auxiliar chamado map()

```
p1, p2 = map(float, input("Digite as suas notas, separadas por vírgula:").split(","))  
print(f"Tipo da variável 1 após receber o valor {type(p1)}")  
print(f"Tipo da variável 2 após receber o valor {type(p2)}")
```

```
Digite as suas notas, separadas por vírgula:10, 5.8  
Tipo da variável 1 após receber o valor <class 'float'>  
Tipo da variável 2 após receber o valor <class 'float'>
```

Você poderá convertê-las separadamente se achar muito complicado assim, mas map() facilita muito!



# Exercícios em Python [5]

Para que a jéssica consiga enviar os brindes e não errar na preferência de cores de todas as participantes, ela precisa saber a quantidade de pessoas que preferem a cor rosa, roxo ou azul. Faça um programa que leia a quantidade de pessoas que preferem cada cor e retorne a porcentagem de cada uma destas cores.



# Exercícios em Python [5]

Para que a jéssica consiga enviar os brindes e não errar na preferência de cores de todas as participantes, ela precisa saber a quantidade de pessoas que preferem a cor rosa, roxo ou azul.

Faça um programa que leia a quantidade de pessoas que preferem cada cor e retorne a porcentagem de cada uma destas cores.





# Exercícios em Python [5]

```
rosa, roxo, azul = map(int, input("Rosa, roxo e azul [Separados por vírgula]:").split(","))
total_mulheres = rosa + roxo + azul

porcentagem_rosa = rosa / total_mulheres * 100
porcentagem_roxo = roxo / total_mulheres * 100
porcentagem_azul = azul / total_mulheres * 100

print(f"{porcentagem_rosa} % de pessoas preferem rosa.")
print(f"{porcentagem_roxo} % de pessoas preferem roxo.")
print(f"{porcentagem_azul} % de pessoas preferem azul.")

Rosa, roxo e azul [Separados por vírgula]:10, 10, 10
33.33333333333333 % de pessoas preferem rosa.
33.33333333333333 % de pessoas preferem roxo.
33.33333333333333 % de pessoas preferem azul.
```





# Introdução a Python

Dia 2 <3



# Recapitulando:

- Conhecemos Colab
- Tipos básicos
- Operadores
- Variáveis
- Entradas e saídas
- Comentários
- Split



# Estruturas condicionais

Uma Estrutura de Condição, como o próprio nome já diz, verifica a condição dos argumentos passados e, executa um comando caso a condição seja verdadeira.



## Algoritmo

[ Uma "receita" para executarmos  
uma tarefa ou resolver algum  
problema]

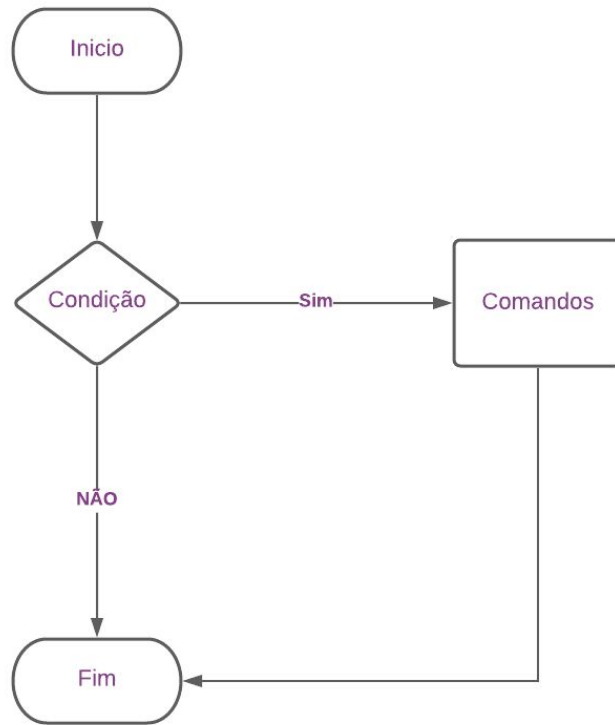


# Estruturas condicionais

Com as estruturas condicionais, podemos fazer com que algumas linhas do nosso algoritmo não sejam executadas.

Sintaxe de um **if simples**:

```
se condicao:  
    comando1  
    comando2
```

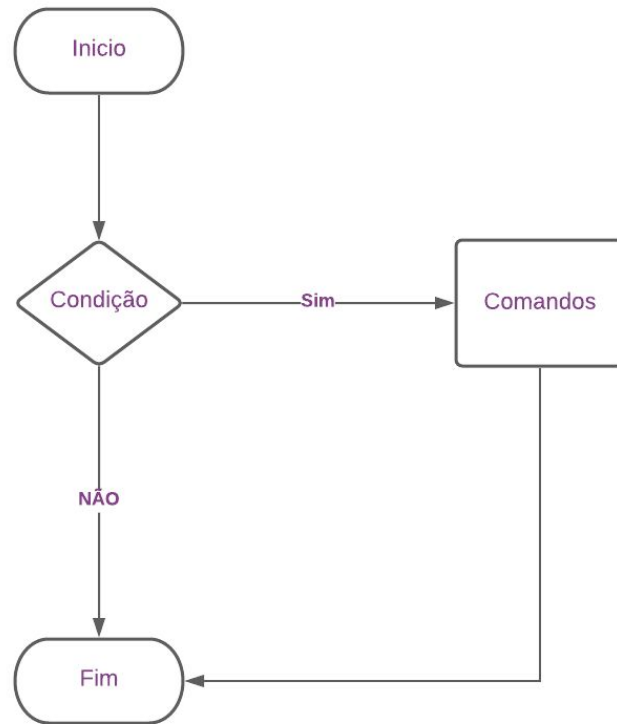


# Estruturas condicionais

Exemplo de um **if simples**:

Emitir um alerta de multa quando a velocidade de um carro estiver acima do permitido (70 km/h)

```
velocidade = float(input("Informe a velocidade: "))  
if(velocidade > 70):  
    print("você será multado")  
print("fim")
```



# Álgebra booleana em Python

Para facilitar a estruturação das condições, podemos utilizar os operadores booleanos (Lembram da tabela da verdade?)

Operador	Python	Exemplo	Descrição
E	and	P and Q	Se P e Q forem verdadeiros retorna True, se não retorna False
OU	or	P or Q	Se P ou Q forem verdadeiros retorna True, se não retorna False
Não	not	not Q	Se Q é verdadeiro retorna False, se não retorna True



# Estruturas condicionais



Exemplos de um **if simples** em situações compostas:

```
# Descubra a situação da massa corporal
massa = float(input("Informe o peso:"))
altura = float(input("Informe altura:"))
imc = massa / (altura*altura)
if (imc < 18):
    print("Abaixo do recomendado")
if (imc >= 18 and imc < 26):
    print ("Recomendado")
if (imc >=26):
    print("Acima do recomendado")
print("Fim!! PS: você é linda em qualquer imc!")
```

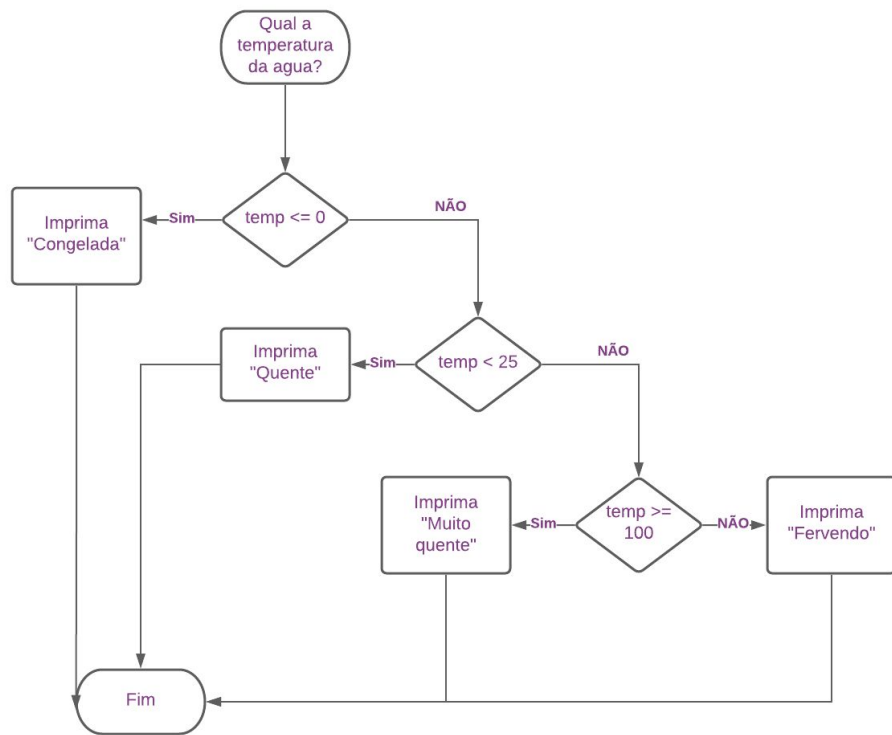




# Estruturas condicionais

Sintaxe de um **if composto**:

```
# Sintaxe de um if composto  
se condição  
    comando1  
    comando2  
senao  
    comando3
```



# Estruturas condicionais

Exemplo de um **if composto** (que tem o else):

```
# Imprimir o maior de dois números
n1 = float(input('Informe o primeiro numero:'))
n2 = float(input('Informe o segundo número:'))
if(n1 > n2):
    print(n1)
else:
    print(n2)
```



# Encadeamento de comandos condicionais

```
# Descubra a massa corporal, utilizando encadeamento de comandos
# Encadeamento de comandos seria, colocar um if dentro de um outro if

massa = float(input("Informe o peso:"))
altura = float(input("Informe altura:"))
imc = massa / (altura*altura)
if (imc < 18):
    print("Abaixo do peso!")
else:
    if(imc >= 18 and imc < 26):
        print ("Peso normal!")
    else:
        print("Acima do peso!")
print (imc)
print ("Fim")
```



# Encadeamento de comandos condicionais, com “elif”

```
# Descubra a massa corporal, utilizando encadeamento de comandos
# utilizando o elif

massa = float(input("Informe o peso:"))
altura = float(input("Informe altura:"))
imc = massa / (altura*altura)
if (imc < 18):
    print("Abaixo do peso!")
elif (imc >= 18 and imc < 26):
    print ("Peso normal!")
else:
    print("Acima do peso!")
print (imc)
print ("Fim")
```



# Indentação e blocos de código

- No Python, tudo que vem depois de ":" é um bloco de código;
- É como se existisse uma hierarquia dentro do código;
- Se os blocos não forem indentados, irá ocorrer erro no console.

Tudo dentro do while é um bloco

```
x = 0
while x < 20:
    print(x)
    x += 2
```

Indentação é o espaçamento em relação ao primeiro nível hierárquico

```
# Exemplo de código não indentado:
y = 20
if y % 2 == 0:
    print('y é par')
else:
    print('y é ímpar')

# O console retorna:
#   print('y é par')
#   ^
# IndentationError: expected an indented block
```

No Python, os Blocos precisam ser indentados



# Exercícios em Python [6]

Dez pessoas assinalaram que iriam participar do encontro do Maria vai com as Devs hoje. Faça um programa que leia o número de pessoas presentes e escreva “Sucesso!” se todas as pessoas compareceram, e “Alguém faltou :/” caso contrário. Se houverem mais de dez pessoas imprima “Temos um intruso entre nós :v”.



# Exercícios em Python [6]

```
marias = int(input("Digite o numero de mulheres presentes:"))
if marias == 10:
    print("Sucesso")
elif marias > 10:
    print("Temos um intruso entre nós :v")
else:
    print("Alguém faltou :/")
```

```
Digite o numero de mulheres presentes:2
Alguém faltou :/
```



# Exercícios em Python [7]

Receba duas notas, uma é de uma prova e outra é de um trabalho, por fim retorne:

- Se a nota da prova for maior ou igual a 5 e nota do trabalho maior que 6: “Aprovado”;
- Senão: “Reprovado”.





# Exercícios em Python [7]

```
nota_prova = float(input("Dígite a nota de sua prova:"))
nota_trabalho = float(input("Dígite a nota de seu trabalho:"))
if nota_prova >= 5 and nota_trabalho > 6:
    print ("Aprovado")
else:
    print ("Reprovado")
```

```
Dígite a nota de sua prova:5
Dígite a nota de seu trabalho:3
Reprovado
```



# Exercícios em Python [8]

A Serasa anualmente promove a ida dos desenvolvedores para a Python Brasil, mas como temos um número limitado de ingressos disponíveis, pode ser necessário decidir quem irá. Como critério de desempate sobre os ingressos, **quem nunca participou do evento terá prioridade**.

Receba o número de ingressos disponíveis, o número total de interessados e o número de interessados que já participaram do evento antes, e diga se é possível distribuir os ingressos apenas com essas regras, ou se será necessário um sorteio.



# Exercícios em python [8]

```
vagas = float(input("Número de vagas disponíveis:"))
ja_participaram = float(input("Número de pessoas que já participaram:"))
interessados = float(input("Número de pessoas interessadas:"))

if interessados == vagas:
    #há vagas o suficiente para todos os interessados
    print("Todos irão")
elif interessados - ja_participaram <= vagas:
    #De acordo com a regra de que, quem nunca participou tem prioridade, todos que nunca foram irão.
    print("Todos que nunca foram irão")
else:
    # Inica que há mais interessados que nunca foram para a Python Brasil do que vagas.
    print("Haverá sorteio")
```

```
Número de vagas disponíveis:10
Número de pessoas que já participaram:2
Número de pessoas interessadas:10
Todos irão
```



# Exercícios em Python [9]

Nos últimos anos o Brasil está com muita variação de temperatura, e uma coisa louca acontecendo é que você pode passar por várias estações do ano num mesmo dia!

Para ajudar as pessoas a não pegar uma gripe ou uma insolação, você vai ler a temperatura e a umidade do ar da tabela do próximo slide, e criar um programa que classifica quais itens de sobrevivência são necessários para nossos dias de “verão”.



# Exercícios em Python [9]

Temperatura às 8h - °C	Umidade do ar - %	Itens de sobrevivência
10°C < 15°C	0 < 40	Blusa de frio e regata
10°C < 15°C	40 <= 100	Blusa de frio, regata e guarda-chuva
15°C < 20°C	0 < 40	Blusa de frio e regata
15°C < 20°C	40 <= 100	Blusa de frio, regata e guarda-chuva
20°C < 25°C	0 < 40	Regata
20°C < 25°C	40 <= 100	Regata e guarda-chuva



# Estruturas de Repetição

As estruturas padrões para repetição em Python são:

```
while (condição):  
    # código
```

```
for item in iterável:  
    # código
```

Observações:

- **Lembre-se:** Python utiliza indentação para definir blocos de código.
- O laço while executar um bloco de código quando determinada condição é atendida.
- O laço for, nos permite percorrer os itens de um iterável e para cada um deles, executar um bloco de código.



# Estruturas de Repetição

Exemplo:

```
▶ # Aqui repetimos o print 3 vezes
  for n in range(0, 3):
      print(n)
  # Output:
  # 0
  # 1
  # 2

  # Aqui iniciamos o n em 0, e repetimos o print até que
  # seu valor seja maior ou igual a 3

  n = 0
  while n < 3:
      print(n)
      n += 1
  # Output:
  # 0
  # 1
  # 2
```



# Estruturas de Repetição

O loop for em Python itera sobre os itens de um conjunto, sendo assim, o range(0, 3) precisa ser um conjunto de elementos. E na verdade ele é:

```
list(range(0, 3))  
[0, 1, 2]
```





Iterando uma lista:

```
lista = [1, 2, 3, 4, 10]
for numero in lista:
    print(numero ** 2)
```

```
1
4
9
16
100
```

Se aplica a strings também:

```
palavra = "casa"
for letra in palavra:
    print(letra)
```

```
c
a
s
a
```



# Auxiliares dos repetidores

- **break:** É usado para sair de um loop, não importando o estado em que se encontra.
- **continue:** Funciona de maneira parecida com a do break, porém no lugar de encerrar o loop, ele faz com que todo o código que esteja abaixo (porém ainda dentro do loop) seja ignorado e avança para a próxima iteração.

```
"""
```

```
Esse código deve rodar até que a palavra "sair" seja digitada.
```

```
* Caso uma palavra com 2 ou menos caracteres seja digitada, um aviso deve ser exibido e o loop será executado do início (devido ao continue), pedindo uma nova palavra ao usuário.
```

```
* Caso qualquer outra palavra diferente de "sair" seja digitada, um aviso deve ser exibido.
```

```
* Por fim, caso a palavra seja "sair", uma mensagem deve ser exibida e o loop deve ser encerrado (break).
```

```
Ou seja: continue, vai para a proxima iteração e o break para tudo.
```

```
"""
```

```
while True:
```

```
    string_digitada = input("Digite uma palavra: ")
```

```
    if string_digitada.lower() == "sair":
```

```
        print("Fim!")
```

```
        break
```

```
    if len(string_digitada) < 2:
```

```
        print("String muito pequena")
```

```
        continue
```

```
    print("Tente digitar \"sair\"")
```

```
Digite uma palavra: a
```

```
String muito pequena
```

```
Digite uma palavra: b
```

```
String muito pequena
```

```
Digite uma palavra: slaodkd
```

```
Tente digitar "sair"
```

```
Digite uma palavra: sair
```

```
Fim!
```

# Exercícios em Python [11]

Leia números enquanto a soma dos números lidos seja menor que 20.



# Exercícios em Python [11]

```
soma = 0

while soma < 20:
    soma += int(input("Digite o próximo número:"))

print(soma)
```

```
Digite o próximo número:3
Digite o próximo número:5
Digite o próximo número:3
Digite o próximo número:12
23
```





Link do material de  
estruturas  
condicionais



# Instalando o Python e a nossa IDE (Pycharm Community)

[Link para o passo-a-passo](#)



# Manipulação de Strings

# Exercícios em Python [9]

```
temperatura = float(input("Temperatura:"))
umidade = float(input("Umidade:"))

if temperatura <= 10 or temperatura > 25:
    #Como não há instruções do que vestir nestas temperaturas, fizemos um tratamento para retornar algo para o usuario.
    print("Não saia de casa, você ficará doente")
else:
    if temperatura < 15:
        # Entra no loop se a temperatura for menor que 15
        if umidade <= 40:
            print("Blusa de frio e regata")
        else:
            print("Blusa de frio, regata e guarda-chuva")
    elif temperatura < 20:
        # Senão, entra no loop se a temperatura for menor que 20
        if umidade <= 40:
            print("Blusa de frio e regata")
        else:
            print("Blusa de frio, regata e guarda-chuva")
    elif temperatura < 25:
        # Senão, entra no loop se a temperatura for menor que 25
        if umidade <= 40:
            print("Regata")
        else:
            print("Regata e guarda-chuva")
```

```
Temperatura:20
Umidade:40
Regata
```





# Strings

Símbolo	Significado	Exemplo	Resultado
+	Concatenação	"bolinho de" + "chuva" + "quero"	?
*	Repetição	"Maria vai com as devs"*4	?
[]	Indexação	"chuva"[1]	?
[:]	Fatiamento	"chuva"[0:1]	?
{}	Substituição	"chuv{}".format("inha")	?
in	Verificação	"chuv" in "chuvinha"	?
not in	Verificação	"chuva" in "chuvinha"	?



# Strings - Símbolos

Símbolo	Significado	Exemplo	Resultado
+	Concatenação	"bolinho de " + "chuva " + "quero"	"bolinho de chuva quero"
*	Repetição	"Maria"*4	"MariaMariaMariaMaria"
[]	Indexação	"chuva"[1]	"h"
[:]	Fatiamento	"chuva"[0:1]	"c"
{}	Substituição	"chuv{}".format("inha")	"chuvinha"
in	Verificação	"chuv" in "chuvinha"	True
not in	Verificação	"chuva" not in "chuvinha"	True



# Strings - Funções

Função	Parâmetro	Aplicação	Resultado
upper	nenhum	"bolinho de chuva".upper()	BOLINHO DE CHUVA
lower	nenhum	"BOLINHO DE CHUVA".lower()	"bolinho de chuva"
capitalize	nenhum	"bolinho de chuva".capitalize()	"Bolinho de chuva"
count	item	"bolinho de chuva".count("h")	2
replace	(old, new)	"bolinho de chuva".replace("a", "inha")	"bolinho de chuvinha"
find	item	"bolinho de chuva".find("chu")	1
len	item	len("bolinho de chuva")	16



# Exercícios em Python [10]

Você é o responsável por gerar os certificados dos da trilha financeira da serasa, porém, sabemos que eventualmente os participantes não cooperam e escrevem seus nomes sem seguir a norma padrão, apenas com a primeira letra maiúscula. Para evitar certificados despadronizados, você resolveu escrever um programinha em Python para normalizar os nomes para você!



# Exercícios em Python [10]

```
nome = input("Digíte o nome do participante:")  
print(f"{nome.capitalize()}")
```

```
Digíte o nome do participante:aliniiiiii  
Aliniiiii
```



# Exercícios em Python [12]

Possuímos uma lista de cinco desenvolvedoras que participam do programa Maria vai com as devs.

Construa um programa que descubra quantas delas possuem o nome começando com a letra “A”.

As desenvolvedoras se chamam: Andorinha, Azaléia, Amélia, Margarida e Rosa.



# Exercícios em Python [12]

```
devs = ['amelia', 'azaleia', 'andorinha', 'margarida', 'rosa']
nome_com_a = 0

for nome in devs:
    if nome[0] == 'a':
        nome_com_a += 1

print(f"Foram encontradas {nome_com_a} desenvolvedoras que começam com a letra A")
```

Foram encontradas 3 desenvolvedoras que começam com a letra A





Powered by  serasa™





# Aprofundamento no Python - Aula 1

- Conhecendo o projeto que vamos construir
- Criar o projeto e utilizar versionamento
- Conhecer tipos complexos: Listas e Dicionários
- Manipulação de strings e arquivos

# Conhecendo o projeto

Estamos abrindo novas vagas para cursos online de programação, houveram inscrições de pessoas de todo o brasil.

Recebemos arquivos com os dados dos inscritos de todo o brasil e para facilitar o gerenciamento de todos os inscritos **precisamos desenvolver um sistema que irá auxiliar os na gestão do curso.**



# Especificações

- Receberemos um arquivo de extensão .csv com os dados dos inscritos.
- Precisamos armazenar os dados em um banco de dados relacional
- Precisamos disponibilizar API's para que nosso sistema se comunique com o mundo externo.

API's disponibilizadas:

- **Listar todos os participantes**
- Deletar um participante
- Alterar dados de um participante
- Adicionar um novo participante
- Detalhar um participante



# E agora?

- Criar projeto no github
- Fazer o clone do projeto no seu computador
- Abrir o pycharm com o projeto



# E agora?

Analisar o arquivo CSV que contém os dados de matrícula e responder:

- Os dados encontrados na planilha são referentes a que? Como é o formato do arquivo?
- Para armazenar estes dados, precisamos criar quantas tabelas?
- O que vamos precisar conhecer para conseguir colocar esse arquivo pra dentro da nossa aplicação python?



# E agora?

Analisar o arquivo CSV que contém os dados de matrícula e responder:

- Os dados encontrados na planilha são referentes a que? Como é o formato do arquivo?

Resposta: Planilha contém dados dos alunos inscritos no curso, possui um cabeçalho e muitas linhas com os registros dos alunos

- Para armazenar estes dados, precisamos criar quantas tabelas?

Resposta: Pessoa, Endereço e Curso

- O que vamos precisar conhecer para conseguir colocar esse arquivo pra dentro da nossa aplicação python?

Resposta: Manipulação Strings, Listas, Dicionários, Manipulação de arquivos, Conectar python com banco de dados, API



Sim! Ainda vamos precisar aprender vários conteúdos até conseguir de fato implementar tudo o que precisamos.





# Manipulação de Strings



# Strings - Símbolos

Símbolo	Significado	Exemplo	Resultado
+	Concatenação	"bolinho de " + "chuva " + "quero"	"bolinho de chuva quero"
*	Repetição	"Maria"*4	"MariaMariaMariaMaria"
[]	Indexação	"chuva"[1]	"h"
[:]	Fatiamento	"chuva"[0:1]	"c"
{}	Substituição	"chuv{}".format("inha")	"chuvinha"
in	Verificação	"chuv" in "chuvinha"	True
not in	Verificação	"chuva" not in "chuvinha"	True



# Strings - Funções

Função	Parâmetro	Aplicação	Resultado
upper	nenhum	"bolinho de chuva".upper()	BOLINHO DE CHUVA
lower	nenhum	"BOLINHO DE CHUVA".lower()	"bolinho de chuva"
capitalize	nenhum	"bolinho de chuva".capitalize()	"Bolinho de chuva"
count	item	"bolinho de chuva".count("h")	2
replace	(old, new)	"bolinho de chuva".replace("a", "inha")	"bolinho de chuvinha"
find	item	"bolinho de chuva".find("chu")	1
len	item	len("bolinho de chuva")	16



# Exercícios em Python [Strings-1]

Você é o responsável por gerar os certificados dos da trilha financeira da serasa, porém, sabemos que eventualmente os participantes não cooperam e escrevem seus nomes sem seguir a norma padrão, apenas com a primeira letra maiúscula. Para evitar certificados despadronizados, você resolveu escrever um programinha em Python para normalizar os nomes para você!



# Exercícios em Python [Strings-1]

```
nome = input("Digíte o nome do participante:")  
print(f"{nome.capitalize()}")
```

```
Digíte o nome do participante:aliniiii  
Aliniiiii
```





Fim aula 1



# Listas

# Listas

É uma sequência de valores organizados entre colchetes [ ].

Esses dados podem ser de diferentes tipos: inteiros, floats, strings e inclusive outras listas.

Exemplos:

```
dezenas = [10, 20, 30, 40]
```

```
pessoas= ['Alini', 'Pri', 'Lu', 'Jessica','Suzi']
```

```
lista_vazia = []
```



# Listas

Os elementos de uma lista podem ser acessados pelo índice:

```
cores = ['amarelo', 'azul', 'branco', 'dourado']
```

Lembrando que a primeira posição de uma lista em Python é a posição 0:

```
>>> cores = ['amarelo', 'azul', 'branco', 'dourado']
>>> cores[0]
'amarelo'
>>> cores[2]
'branco'
>>>
```





# Listas

Além de ser possível **acessar** cada elemento da lista pelo índice, é possível, **inserir, alterar e deletar**.

Listas são **mutáveis**, ou seja, é possível alterar seus elementos: **adicionando, removendo e/ou substituindo-os**.



# Listas

Formato da função	Explicação	Exemplo
<code>lista.append(exemplo)</code>	Adiciona um novo elemento ao final da lista.	<code>inteiros = [1, 2, 3]</code> <code>inteiros.append(10)</code>
<code>lista.insert(posição, elemento)</code>	Adiciona um novo elemento numa determinada posição da lista. Todos os elementos daquela posição em diante são deslocados uma posição para frente.	<code>disciplinas = ["SO", "IA"]</code> <code>disciplinas.insert(0, "CAP")</code>
<code>lista.remove(elemento)</code>	Remove o primeiro elemento especificado encontrado na lista.	<code>nomes = ["Ana", "Bia"]</code> <code>nomes.remove("Ana")</code>
<code>lista.count(elemento)</code>	Conta quantas vezes o elemento especificado aparece na lista. Retorna zero se o elemento não pertencer à lista.	<code>frutas = ["maçã", "uva"]</code> <code>frutas.count("uva")</code>



# Listas

Formato da função	Explicação	Exemplo
<code>lista.index(elemento)</code>	Retorna a posição da primeira ocorrência do elemento especificado. Caso não seja encontrado, um erro será lançado.	<code>tamanhos = ["M", "GG"]</code> <code>tamanhos.index("GG")</code>
<code>lista.pop(posição)</code>	Remove e retorna o elemento da posição especificada. Caso não seja especificada uma posição, removerá o último elemento da lista.	<code>Inteiros = [5, 6, 7, 8]</code> <code>inteiros.pop(2)</code>
<code>lista.sort()</code>	Caso os elementos tenham implementação do operador menor (<) e sejam comparáveis (normalmente, de mesmo tipo), Python oferece uma função pronta para ordenar seu conteúdo. Caso os elementos não sejam comparáveis, um erro será lançado.	<code>cores = ["lilás", "azul"]</code> <code>cores.sort()</code>



# Listas

Formato da função	Explicação	Exemplo
<code>lista.reverse()</code>	Inverte as posições dos elementos de uma lista. O último passa a ser o primeiro, o penúltimo passa a ser o segundo e assim por diante.	<code>coffee = ["bolo", "refri"] coffee.reverse()</code>
<code>list("objeto iterável")</code>	Cria uma lista a partir de um objeto iterável, por exemplo, uma string. Se não for objeto iterável, um erro será lançado	<code>palavra = "PyLadies" list(palavra)</code>
<code>len(lista)</code>	Retorna o tamanho de uma lista. Funciona para qualquer objeto iterável. Se não for um objeto iterável, um erro será lançado.	<code>ladies = ["Nath", "Karol"] len(ladies)</code>
<code>elemento in lista</code>	Verifica se um elemento está presente em uma lista.	<code>trilhas = ["IA", "web"] "IA" in trilhas</code>



# Listas

Exemplo de como utilizar algumas das funções apresentadas:

```
coffee = ['refri', 'bolinha de queijo', 'bolo', 'coxinha']  
coffee.append('café')  
coffee.insert(0, 'água')  
coffee.remove('refri')
```

Resultado esperado:

```
>>['água', 'bolinha de queijo', 'bolo', 'coxinha', 'café']
```



Iterando uma lista:

```
lista = [1, 2, 3, 4, 10]
for numero in lista:
    print(numero ** 2)
```

```
1
4
9
16
100
```

Se aplica a strings também:

```
palavra = "casa"
for letra in palavra:
    print(letra)
```

```
c
a
s
a
```



# Exercícios em Python [Listas-1]

Possuímos uma lista de cinco desenvolvedoras que participam do programa Maria vai com as devs.

Construa um programa que descubra quantas delas possuem o nome começando com a letra “A”.

As desenvolvedoras se chamam: Andorinha, Azaléia, Amélia, Margarida e Rosa.



# Exercícios em Python [Listas-1]

```
devs = ['amelia', 'azaleia', 'andorinha', 'margarida', 'rosa']
nome_com_a = 0

for nome in devs:
    if nome[0] == 'a':
        nome_com_a += 1

print(f"Foram encontradas {nome_com_a} desenvolvedoras que começam com a letra A")
```

Foram encontradas 3 desenvolvedoras que começam com a letra A





# Exercícios em Python [Listas-2]

Leia um número N e, em seguida, leia N números inteiros. Imprima o maior e o menor entre os N inteiros lidos.

Exemplo de entrada: 10

10 -5 1 2 3 100 4 5 -17 1

Saída esperada: Maior: 100

Menor: -17



# Exercícios em Python [Listas-2]

```
n = int(input("Digite a quantidade de números que deseja processar:"))
maior = -1000000
menor = 1000000
entrada = map(int, input(f"Digite {n} números, separado por vírgula:").split(","))

for atual in entrada:
    menor = min(menor, atual)
    maior = max(maior, atual)

print(f"Maior: {maior}")
print(f"Menor: {menor}")
```

Digite a quantidade de números que deseja processar:10

Digite 10 números, separado por vírgula:10, -5, 1, 2, 3, 100, 4, 5, -17,1

Maior: 100

Menor: -17



# Exercícios em Python

## [Listas-2-segunda solução <3 mais linda também]

```
quantidade = int(input("Quantos números você quer escolher? "))
index = 0
lista = []
while (index < quantidade):
    proximo = int(input("Digite o proximo número: "))
    index += 1
    lista.append(proximo)

lista.sort()
print(lista)
print(f"Menor: {lista[0]} , Maior: {lista[-1]}")
```

exemplos x

```
C:\Users\alini\Documents\repository\Virtualenvs\mvcad-cursos\Scripts\python.exe
Quantos números você quer escolher? 5
Digite o proximo número: 10
Digite o proximo número: -10
Digite o proximo número: 100
Digite o proximo número: 8
Digite o proximo número: 5
[-10, 5, 8, 10, 100]
Menor: -10 , Maior: 100

Process finished with exit code 0
```



# Exercícios em Python [Listas-3]

O RH da Serasa precisa controlar a presença dos membros nas atividades do MVCAD, isto é, precisa anotar na planilha de frequência todos os que participaram.

Como você está estudando Python, se voluntariou a ajudar a tornar essa tarefa um pouco mais fácil, você pretende entregar a lista de presentes já ordenada alfabeticamente para o RH!

Dado um número N, que representa quantas pessoas participaram, e N nomes, imprima os N nomes ordenados alfabeticamente.

Exemplo de entrada:

4  
Suzi  
Pri  
Alini  
Lu

Exemplo de entrada:

Alini  
Lu  
Pri  
Suzi



# Exercícios em Python [Listas-3]

```
presentes = int(input("Quantas pessoas estão presentes?"))
pessoas = []
for i in range(presentes):
    pessoas.append(input("Digite o nome do participante:"))
pessoas.sort()
print("Participantes por ordem alfabética:")
print("\n".join(pessoas))
```

```
Quantas pessoas estão presentes?4
Digite o nome do participante:Suzi
Digite o nome do participante:Alini
Digite o nome do participante:Lu
Digite o nome do participante:Pri
Participantes por ordem alfabética:
Alini
Lu
Pri
Suzi
```





# Dicionários

# Dicionários

Dicionário é um objeto que relaciona uma chave a um valor.

Um dicionário se parece com uma lista, mas é mais geral. Em uma lista, os índices têm que ser números inteiros; em um dicionário, eles podem ser de (quase) qualquer tipo.

Um dicionário contém uma coleção de índices, que se chamam chaves e uma coleção de valores. Cada chave é associada com um único valor. A associação de uma chave e um valor chama-se par chave-valor ou item.

Em linguagem matemática, um dicionário representa um mapeamento de chaves a valores, para que você possa dizer que cada chave “mostra o mapa a” um valor.



# Dicionários

A função `dict` cria um novo dicionário sem itens.

Como `dict` é o nome de uma função integrada, você deve evitar usá-lo como nome de variável.

```
>>> marias = dict()
>>> marias
{}

```

As chaves `{}` representam um dicionário vazio. Para acrescentar itens ao dicionário, você pode usar colchetes (a linha abaixo cria um item que mapeia da chave `nome` ao valor `'Maria vai com as devs'`. ):

```
>>> marias['nome'] = 'Maria vai com as devs'
>>> marias
{'nome': 'Maria vai com as devs'}

```





# Dicionários

```
dicionario_vazio = {}  
camiseta = {"M":12, "G": 10, "GG":10}  
  
dados = {  
    "grupo": "PyLadies São Carlos" ,  
    "fundação":2014,  
    "linguagem": "Python",  
    "membras": 37  
}  
  
alunas_curso = {"Ana": "BCC", "Amanda": "EnC"}  
alunas_curso["Lúcia"] = "BCC"
```



# Dicionários

Para verificar se uma chave já existe, utilize o operador in:

```
>>> dic = {'chave': 'valor'}
>>> 'chave' in dic
True
>>> 'chave_inexistente' in dic
False
>>>
```

Para deletar uma chave e seu valor de um dicionário, utilize o operador del:

```
>>> del(dic['chave'])
>>> dic
{}
>>> _
```



# Dicionários

Podemos obter todas as chaves de um dicionário com o método `keys()`:

```
dicionario.keys()
```

E também podemos obter todos os valores de um dicionário com o método `values()`:

```
dicionario.values()
```



# Dicionários - Funções

Formato da função	Explicação	Exemplo
<code>dicionario.clear()</code>	Deleta todo o conteúdo do dicionário e retorna um dicionário vazio.	<code>mvcad= {"nome": "Ana"}</code> <code>mvcad.clear()</code>
<code>dicionario.copy()</code>	Retorna um novo dicionário, com uma cópia real do objeto. Se você fizer <code>dict1 = dict2</code> , ele criará apenas uma referência.	<code>mvcad= {"nome": "Ana"}</code> <code>mvcad.copy()</code>
<code>dicionario.fromkeys( objeto_iterável, valor_default)</code>	Cria um novo dicionário com as chaves contidas no objeto_iterável e valores baseadas no valor_default.	<code>dict().fromkeys([1, 2, 3, 4], None)</code>



# Dicionários - Funções

Formato da função	Explicação	Exemplo
<code>dicionario.get(chave)</code>	Retorna o valor de uma dada chave, ou None caso ela não exista. É aconselhável o uso em vez de acesso direto em casos onde uma chave possa não existir, pois no acesso direto causaria erro.	<pre>mvcad= {"nome": "Alini", "estado": "SC"} mvcad.get('nome')</pre>
<code>dicionario.items()</code>	Retorna um objeto <code>dict_items</code> , que é composto por uma tupla que contém as chaves e outra que contém os valores. Útil em iterações para pegar chave e valor.	<pre>mvcad= {"nome": "Alini", "estado": "SC"} mvcad.items()</pre>
<code>dicionario.keys()</code>	Retorna um objeto <code>dict_keys</code> , que é composto por uma lista com as chaves.	<pre>mvcad= {"nome": "Alini", "estado": "SC"} mvcad.keys()</pre>



# Dicionários - Funções

Formato da função	Explicação	Exemplo
dicionario.pop(chave )	<b>Retorna o valor de uma dada chave e exclui todo o elemento do dicionário.</b>	<pre>mvcad= {"nome": "Ana"} mvcad.pop('nome')</pre>
dicionario.popitem()	<b>Retorna um elemento do dicionário e exclui ele do dicionário. Não é possível escolher o elemento que será retornado.</b>	<pre>mvcad= {"nome": "Ana"} mvcad.popitem()</pre>
dicionario.update(dicionario)	<b>Atualiza elementos existentes ou adiciona elementos caso estes não existam num dicionário base.</b>	<pre>mvcad= {"nome": "Ana"} mvcad.update()</pre>



# Dicionários

Para iterar sobre dicionários usando a chave como acesso:

```
for chave in dicionario:
```

```
    dicionario[chave]
```





Intervalo:  
Volta às 11:20





# Recapitulando:

- Conhecemos o projeto que vamos construir
- Manipulação de Strings
- Listas
- [Dicionários](#)



# E agora?

Analisar o arquivo CSV que contém os dados de matrícula e responder:

- Os dados encontrados na planilha são referentes a que? Como é o formato do arquivo?

Resposta: Planilha contém dados dos alunos inscritos no curso, possui um cabeçalho e muitas linhas com os registros dos alunos

- Para armazenar estes dados, precisamos criar quantas tabelas?

Resposta: Pessoa, Endereço e Curso

- O que vamos precisar conhecer para conseguir colocar esse arquivo pra dentro da nossa aplicação python?

Resposta: ~~Manipulação Strings, Listas, Dicionários,~~

**Manipulação de arquivos, Conectar python com banco de dados, API**



# Manipulando arquivos de texto

Podemos abrir um arquivo de duas maneiras:

- Para somente leitura ('r')
- Com permissão de escrita ('w')

```
#  
# leitura  
#  
f = open('nome-do-arquivo', 'r')  
  
#  
# escrita  
#  
f = open('nome-do-arquivo', 'w')
```



# Manipulando arquivos de texto

Se não especificarmos o segundo parâmetro, a forma padrão leitura ('r') será utilizada:

```
>>> arquivo = open('nome-do-arquivo')
>>> arquivo
<_io.TextIOWrapper name='nome-do-arquivo.text' mode='r' encoding='UTF-8'>
```

O terceiro parâmetro é opcional e nele especificamos a codificação do arquivo:

```
arquivo = open(nome-do-arquivo, 'r', encoding="utf8")
```



# Manipulando arquivos de texto

Se tentarmos abrir um arquivo para leitura que não existe, um erro será lançado:

```
>>> f = open('nome-errado.text', 'r')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'nome-errado.text'
```

Se tentarmos abrir um arquivo para escrita que não existe, então ele será criado, porém, se ele já existir, todo seu conteúdo será apagado no momento em que abrirmos o arquivo.

Devemos sempre fechar o arquivo aberto: **arquivo.close()**



# Exemplos

Como exemplo utilizaremos o arquivo de texto

**seu-arquivo.text** que possui o seguinte

conteúdo:

primeira linha  
segunda linha  
terceira linha  
quarta linha  
quinta linha

Podemos abrir um arquivo e iterar por cada linha conforme exemplo abaixo:

```
1 f = open('arquivo.txt', 'r')
2
3
4 for line in f:
5     print(line)
6
7
8 for line in f:
```

Run: handling\_files x

C:\Users\alini\Documents\repository\Virtualenvs\teste

primeira linha

segunda linha

terceira linha

quarta linha

quinta linha

Process finished with exit code 0



# Exemplos

Se quisermos ler todo o conteúdo do arquivo em uma única string podemos utilizar a função `read()`:

```
>>> f = open('seu-arquivo.text', 'r')
>>> f.read()
'primeira linha\nsegunda linha\nterceira linha\nquarta linha\nquinta linha'
```

A função retornará uma lista vazia `[]` quando encontrar o final do arquivo (após a última linha ter sido lida).

```
>>> f = open('seu-arquivo.text', 'r')
>>> f.readline()
'primeira linha\n'
>>> f.readline()
'segunda linha\n'
>>> f.readline()
'terceira linha\n'
>>> f.readline()
'quarta linha\n'
>>> f.readline()
'quinta linha'
>>> f.readline()
''
```



# Exemplos

Se quisermos ler todas linhas restantes em uma lista podemos utilizar a função `readlines` (estamos no plural).

```
>>> f = open('seu-arquivo.text', 'r')
>>> f.readlines()
['primeira linha\n', 'segunda linha\n', 'terceira linha\n', 'quarta linha\n', 'quinta linha']
>>> f.readlines()
[]
```





# Exemplos

Repare que ao chamarmos pela segunda vez a função retornar uma lista vazia pois ela, na verdade, retorna as linhas restantes. Como, ao abrir o arquivo, restavam todas as linhas então ela retornou todas as linhas.

Confundi? Veja se este exemplo clareia as coisas.

```
>>> f = open('seu-arquivo.text', 'r')
>>> f.readline()
'primeira linha\n'
>>> f.readline()
'segunda linha\n'
>>> f.readlines()
['terceira linha\n', 'quarta linha\n', 'quinta linha']
```



# Exemplos

Para escrever em um arquivo sem apagar seu conteúdo, ou seja, adicionando

(incluído) novo conteúdo seguimos 3 passos:

- Ler todo o conteúdo do arquivo (Cursor vai para o final do arquivo)
- efetuar a adição do conteúdo
- escrever as linhas

```
# Abra o arquivo (leitura)
arquivo = open('musica.txt', 'r')
conteudo = arquivo.readlines()

# insira seu conteúdo
# obs: o método append() é proveniente de uma lista
conteudo.append('Nova linha')

# Abre novamente o arquivo (escrita)
# e escreva o conteúdo criado anteriormente nele.
arquivo = open('musica.txt', 'w')
arquivo.writelines(conteudo)
arquivo.close()
```





# Leitura e escrita de arquivos CSV

# Arquivos CSV

**CSV** (Comma Separated Values, ou valores separados por vírgulas) correspondem ao idioma comum de troca de informações entre planilhas de cálculo ou base de dados.

Embora cada aplicativo tenha o seu método específico de **codificar seus dados**, todos possuem uma forma de **exportar e importar** dados em formato csv.

- Cada linha em um arquivo csv é um registro de dados.
- Cada registro consiste de um ou mais campos, separados por vírgulas.



# Biblioteca CSV

A linguagem Python possui um módulo, não por acaso denominado csv, que facilita em muito o trabalho com este tipo de arquivo em Python.

Este módulo implementa funções que realizam a leitura e escrita sobre arquivos csv.

<https://cadernodelaboratorio.com.br/arquivos-csv-em-python/>



# Biblioteca CSV

As funções que iremos conhecer são:

- **csv.reader()**: Retorna um objeto “reader” que permite a iteração sobre as linhas do arquivo csv
- **csv.writer()**: Retorna um objeto “writer” que converte o dado do usuário numa linha do arquivo csv.
- **csv.DictReader()**: Com esta classe trabalhamos com um dicionário no qual as chaves são dadas pelos nomes das colunas. Estes nomes podem ser dados através do parâmetro “fieldnames” ou obtidos diretamente da primeira linha do arquivo csv.
- **csv.DictWriter()**: Cria um objeto que permite a gravação de linhas no arquivo csv a partir de um dicionário passado como parâmetro.



## Exemplo CSV.reader()

```
with open('teste.csv') as csfile:  
    reader = csv.reader(csfile)
```

## Exemplo CSV.writer()

```
with open('teste.csv', 'w') as csfile:  
    writer = csv.writer(csfile)  
    writer.writerow("nova linha")
```



# Exemplo CSV.DictWriter()

```
# w -> write, a -> append
with open('meu_arquivo.csv', 'a', newline='') as csvfile:
    fieldnames = ['nome_coluna1', 'nome_coluna2']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames, delimiter=";")

    writer.writeheader()
    writer.writerow({'nome_coluna1': 'boeing 737', 'nome_coluna2': 'Ferrari'})
    writer.writerow({'nome_coluna1': 'airbus4', 'nome_coluna2': 'Volks6'})
```





# Exemplo

## CSV.DictReader()

```
with open('meu_arquivo.csv') as csvfile:
    reader = csv.DictReader(csvfile, delimiter=";")
    for row in reader:
        print(row['nome_coluna1'], row['nome_coluna2'])
```





# Recapitulando:

- Conhecemos como funciona a manipulação de arquivos de texto no Python
- Conhecemos leitura e escrita de CSV no Python





Importando  
arquivo .csv para  
nosso projeto

# E agora?

- Vamos voltar para nosso projeto
- Importar o arquivo CSV
- Separar os dados dos alunos em uma lista





Aula dia 21 de  
novembro



# Conectando com banco de dados

- Utilizar a biblioteca [PyGreSQL](#) (Exemplo da instalação a seguir)
- Analizar e replicar o [exemplo do arquivo indicado](#).

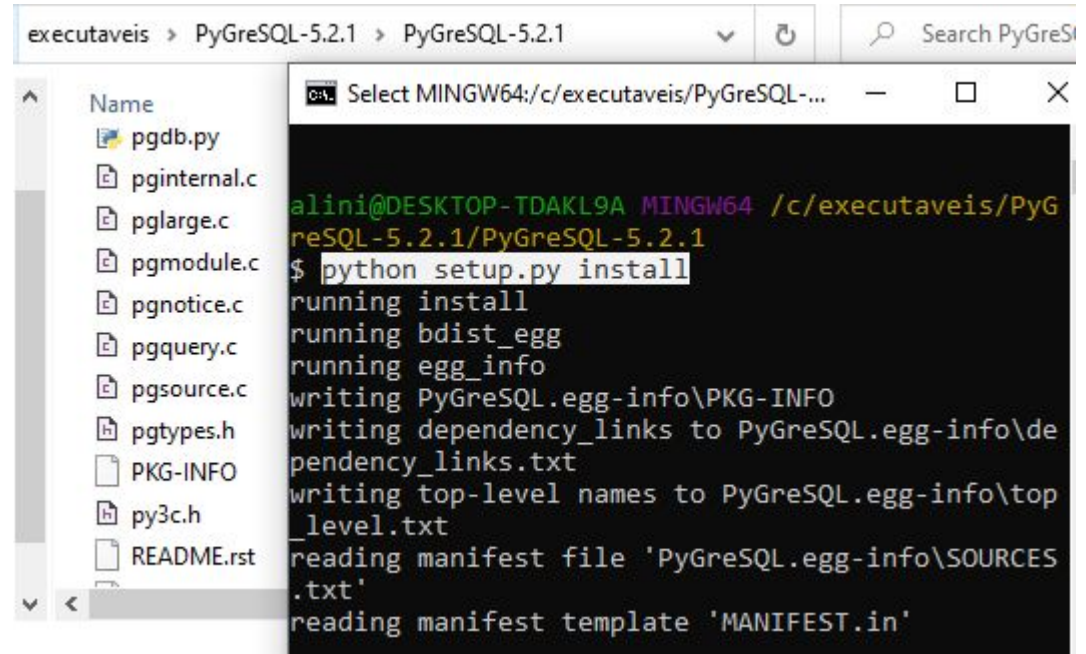
# Instalando a biblioteca

Página do PyGreSQL na pypi

comando-> `pip install pyGreSQL`

**OU**

Baixar arquivo zip, descompactar e dentro da pasta, executar o comando `python setup.py install`



The image shows a Windows file explorer window and a terminal window side-by-side. The file explorer window is titled 'executaveis > PyGreSQL-5.2.1 > PyGreSQL-5.2.1' and displays a list of files: pgdb.py, pginternal.c, pglarge.c, pgmodule.c, pgnotice.c, pgquery.c, pgsource.c, pgtypes.h, PKG-INFO, py3c.h, and README.rst. The terminal window is titled 'Select MINGW64:/c/executaveis/PyGreSQL-...' and shows the following commands and output:

```
alini@DESKTOP-TDAKL9A MINGW64 /c/executaveis/PyGreSQL-5.2.1
$ python setup.py install
running install
running bdist_egg
running egg_info
writing PyGreSQL.egg-info\PKG-INFO
writing dependency_links to PyGreSQL.egg-info\dependency_links.txt
writing top-level names to PyGreSQL.egg-info\top_level.txt
reading manifest file 'PyGreSQL.egg-info\SOURCES.txt'
reading manifest template 'MANIFEST.in'
```



# Modelar nosso banco



- Quantas tabelas precisamos criar?

Volte para as especificações e verifique qual dado vamos precisar disponibilizar e faça uma comparação com os dados que o arquivo possui.





# Especificações

- Receberemos um arquivo de extensão .csv com os dados dos inscritos.
- Precisamos armazenar os dados em um banco de dados relacional
- Precisamos disponibilizar API's para que nosso sistema se comunique com o mundo externo.

API's disponibilizadas:

- **Listar todos os participantes**
- Deletar um participante
- Alterar dados de um participante
- Adicionar um novo participante
- Detalhar um participante



# Modelar nosso banco

- De acordo com as especificações, apenas vamos construir uma tabela chamada pessoa (**make it simple**).
- Esta tabela irá possuir os campos: Nome, endereço, cpf, estado, turma, período, módulo
- Vã para o pgAdmin e crie um **banco de dados** específico para nosso projeto, chamado **mvcad-cursos** e nele, crie uma **tabela** chamada **pessoa**, com todos os campos necessários.
- Comando: **xxxxxxx**



# Conexão com o banco

- Estabelecer conexão com banco ->

```
db = DB(dbname='mvcad-cursos', host='localhost', port=5432, user='postgres', passwd='postgres')
```

- Buscar todas as tabelas -> `db.get_tables()`

- Inserir dados na tabela -> `db.insert('aluno', id=uuid4(), name='Alini')`

- Fazer select do banco -> 

```
query = db.query('select * from aluno')
resultato = query.getresult()
```



# Criando métodos que manipulam dados do banco

- Criar função que salva uma pessoa
- Criar função que busca uma pessoa
- Criar função que retorna todas as pessoas salvas
- Criar função que deleta uma pessoa



# Ligar as funções criadas com o arquivo CSV importado

- Criar função que lê todos os dados do arquivo csv e envia para a função criada anteriormente que salva uma pessoa no banco de dados. (Lembrando que você pode encontrar o projeto da última [aula aqui](#).)
- Esta função deve estar dentro do arquivo `leitor.csv`, percorrer todos os dados da `lista_pessoas` e chamar a função criada anteriormente para criar uma pessoa no banco.





# Recapitulando:

- Conhecemos como criar uma conexão entre o banco de dados e a aplicação python
- Criamos funções que enviam os dados para o banco
- Salvamos os dados do nosso arquivo .csv no banco de dados.



# E agora?

Analisar o arquivo CSV que contém os dados de matrícula e responder:

- Os dados encontrados na planilha são referentes a que? Como é o formato do arquivo?

Resposta: Planilha contém dados dos alunos inscritos no curso, possui um cabeçalho e muitas linhas com os registros dos alunos

- Para armazenar estes dados, precisamos criar quantas tabelas?

Resposta: Pessoa, Endereço e Curso

- O que vamos precisar conhecer para conseguir colocar esse arquivo pra dentro da nossa aplicação python?

Resposta: ~~Manipulação Strings, Listas, Dicionários, Manipulação~~  
~~de arquivos, Conectar python com banco de dados,~~ **API**





Volta do  
Intervalo: 11:20

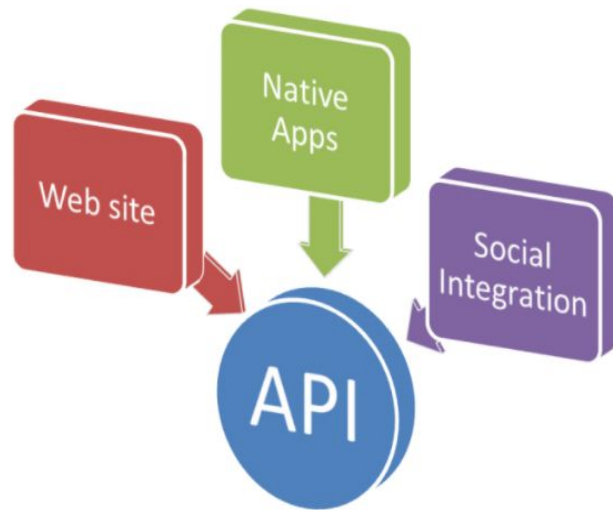




APIs

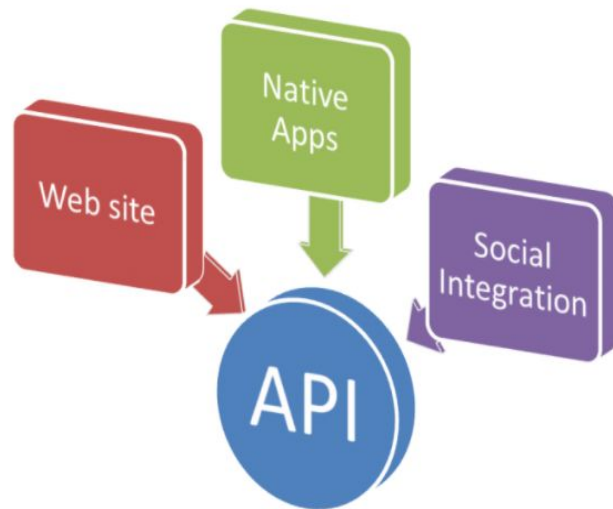
# O que é uma API?

O acrônimo API que provém do inglês Application Programming Interface (Em português, significa Interface de Programação de Aplicações), trata-se de um conjunto de rotinas e padrões estabelecidos e documentados por uma aplicação **A**, para que outras aplicações consigam utilizar as funcionalidades desta aplicação **A**, sem precisar conhecer detalhes da implementação do software.



# O que é REST?

REST significa **Representational State Transfer**. Em português, Transferência de Estado Representacional. Trata-se de uma abstração da arquitetura da Web. Resumidamente, o REST consiste em princípios e regras que, quando seguidas, permitem a criação de um projeto com interfaces bem definidas. Desta forma, permitindo, por exemplo, que aplicações se comuniquem.



# REST x RESTful

Existe uma certa confusão quanto aos termos REST e RESTful.

Entretanto, ambos representam os mesmo princípios. A diferença é apenas gramatical. Em outras palavras, sistemas que utilizam os princípios REST são chamados de RESTful.

- REST: conjunto de princípios de arquitetura
- RESTful: capacidade de determinado sistema aplicar os princípios de REST.



# Como vamos construir a API em nosso projeto Python?

- Utilizando a biblioteca [flask-restful](#)
- Flask é um microframework muito utilizado para a construção de aplicações ([Documentação](#)).
- Imagem ao lado é o mínimo para colocar a aplicação em pé.

```
from flask import Flask
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

class HelloWorld(Resource):
    def get(self):
        return {'hello': 'world'}

api.add_resource(HelloWorld, '/')

if __name__ == '__main__':
    app.run(debug=True)
```



# Criando APIs que chamam as funções criadas anteriormente, de manipulação dos dados do banco

- API que lista todas as pessoas
- API que lista todos os dados de uma pessoa (A partir de um ID)
- API que deleta uma pessoa (A partir de um ID)
- API que atualiza os dados de uma pessoa (A partir de um ID)
- API que cria um novo participante



# Links complementares para estudar

- [Flask-RESTful](#)
- Validação de dados com [Schematics](#)
- ORM [SQLAlchemy](#)





Docker





# Docker

Quem nunca ouviu a frase: "**Funciona na minha máquina**"??????

No mundo do desenvolvimento sempre temos vários ambientes e em todos eles, esperamos que nossos softwares funcionem perfeitamente!

Muitas vezes as coisas não acontecem como planejamos e para nos ajudar a colocar nossos softwares em várias máquinas diferentes e fazer com que ele se comporte igual em todas, existe o docker!



# O que era antes do docker?

Antigamente, quando queríamos montar o nosso sistema, com vários serviços (bancos de dados, proxy, etc) e aplicações, acabávamos tendo vários servidores físicos, cada um com um serviço ou aplicação do nosso sistema, por exemplo:



# O que era antes do docker?

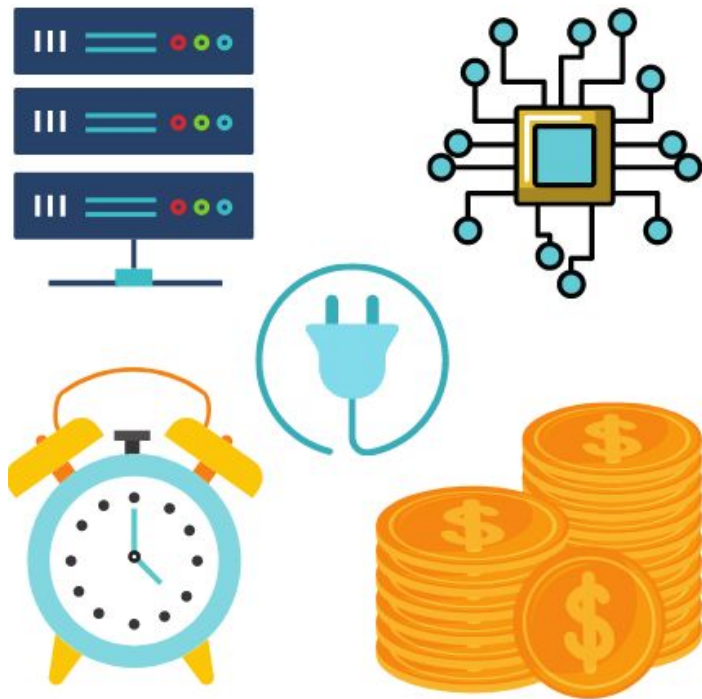
E claro, não conseguimos instalar os serviços diretamente no hardware do servidor, ou seja, precisamos de um intermediário entre as aplicações e o hardware, que é o sistema operacional. Ou seja, devemos instalar sistemas operacionais em cada servidor, e os sistemas poderiam ser diferentes:



# O que era antes do docker?

E se quisermos que uma aplicação se comunique com outra ou faça qualquer comunicação externa, devemos conectar os servidores à rede. Além disso, para eles funcionarem, precisamos ligá-los à eletricidade. **Logo, havia diversos custos de eletricidade, rede e configuração dos servidores.**

Além disso, o processo era lento, já que a cada nova aplicação, deveríamos comprar/montar o servidor físico, instalar o sistema operacional, configurá-lo e subir a aplicação.



# Capacidade pouco aproveitada

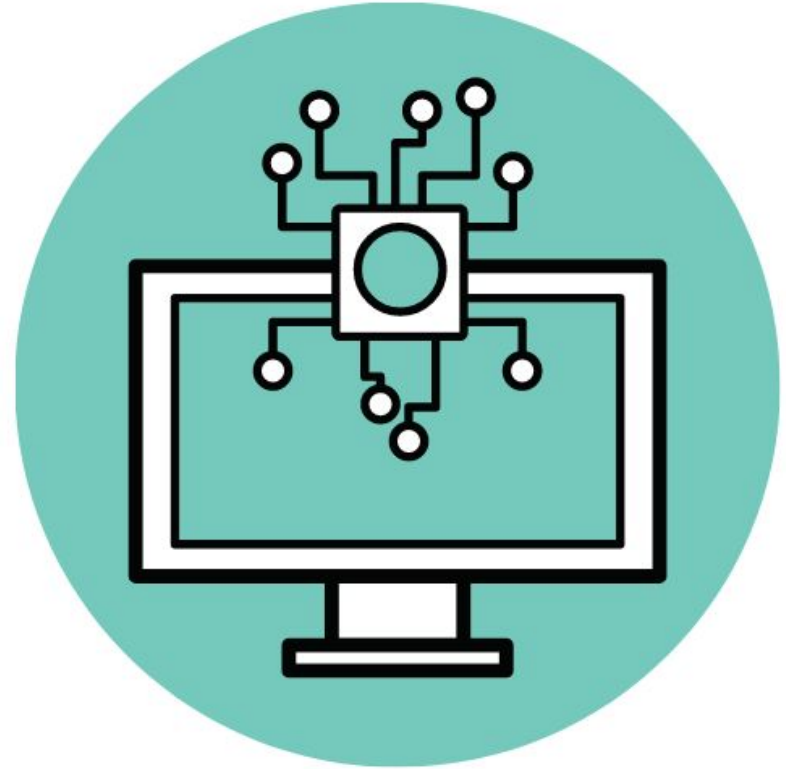
O único problema dessa arquitetura de vários servidores físicos não era apenas o custo, era muito comum termos servidores parrudos, com uma única aplicação sendo executada, para normalmente ficarem funcionando abaixo da sua capacidade, para quando for necessário, o servidor aguentar uma grande quantidade de acessos.

Isso resultava em muita capacidade ociosa nos servidores, com muitos recursos desperdiçados.



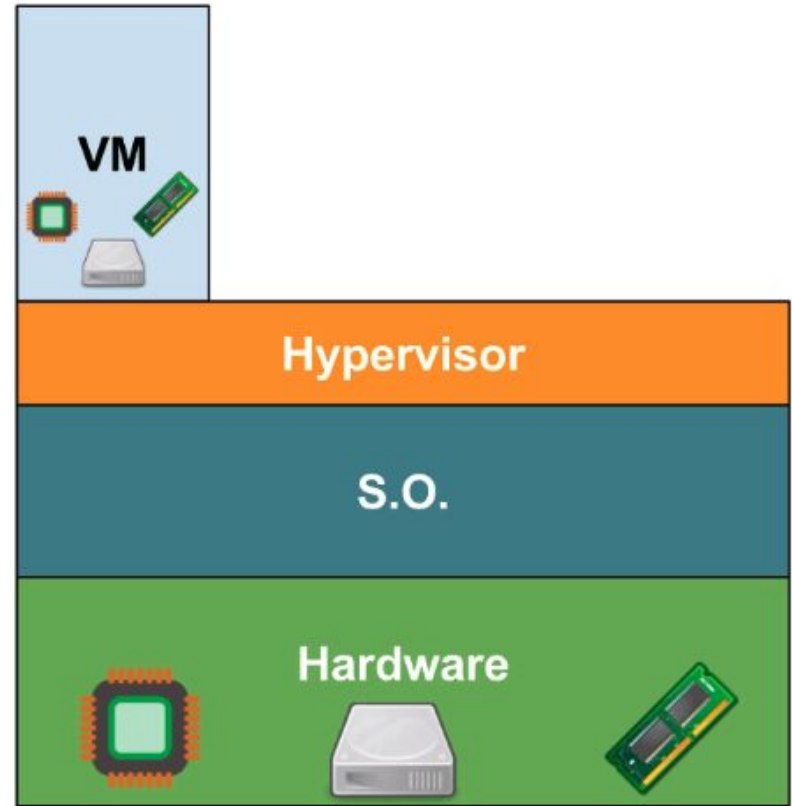
# Virtualização

Para fugir desses problemas de servidores ociosos e alto tempo e custo de subir e manter aplicações em servidores físicos, surgiu como solução a virtualização, surgindo assim as **máquinas virtuais**.



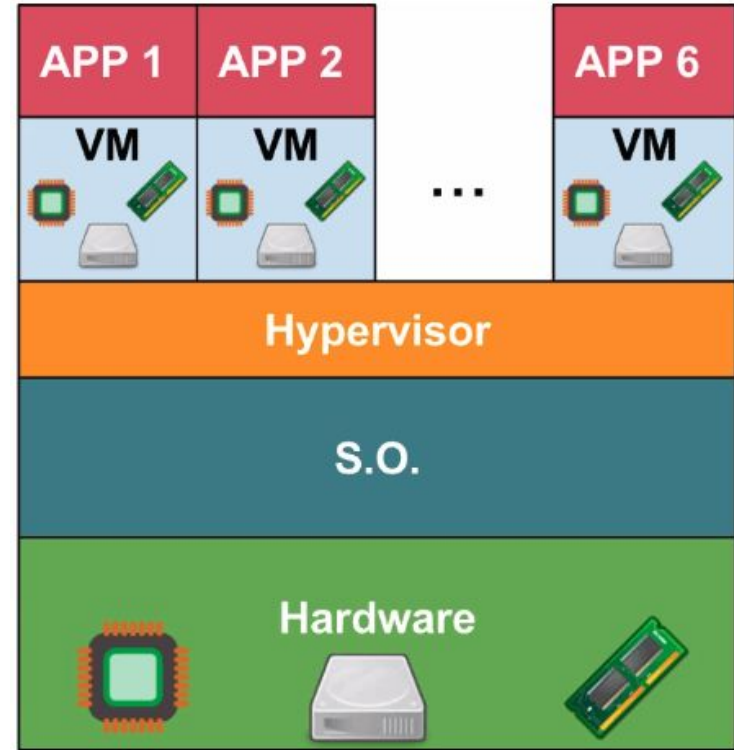
# Virtualização

As máquinas virtuais foram possíveis de ser criadas graças a uma tecnologia chamada Hypervisor, que funciona em cima do sistema operacional, permitindo a virtualização dos recursos físicos do nosso sistema. Assim, criamos uma máquina virtual que tem acesso a uma parte do nosso HD, memória RAM e CPU, criando um computador dentro de outro:



# Virtualização

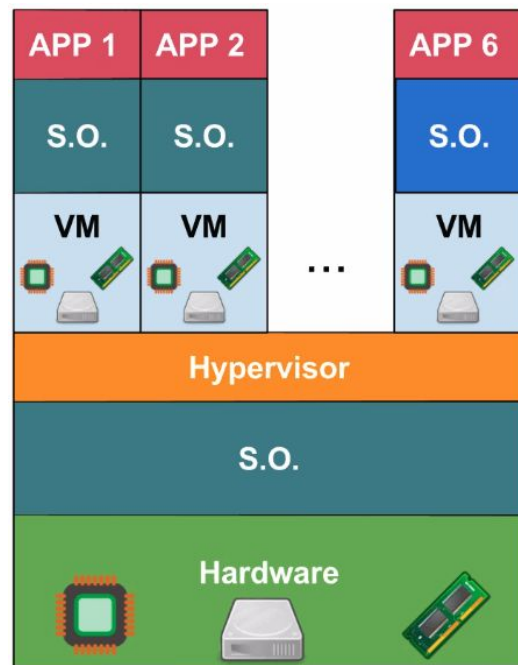
E se temos uma máquina virtual que está acessando uma parte do nosso hardware como um todo, conseguimos colocar dentro dela uma das nossas aplicações. E replicar isso, criando mais máquinas virtuais com outras aplicações, **desta forma reduzimos o custo de energia e rede, aproveitamos melhor os recursos e diminuimos a ociosidade do servidor.**





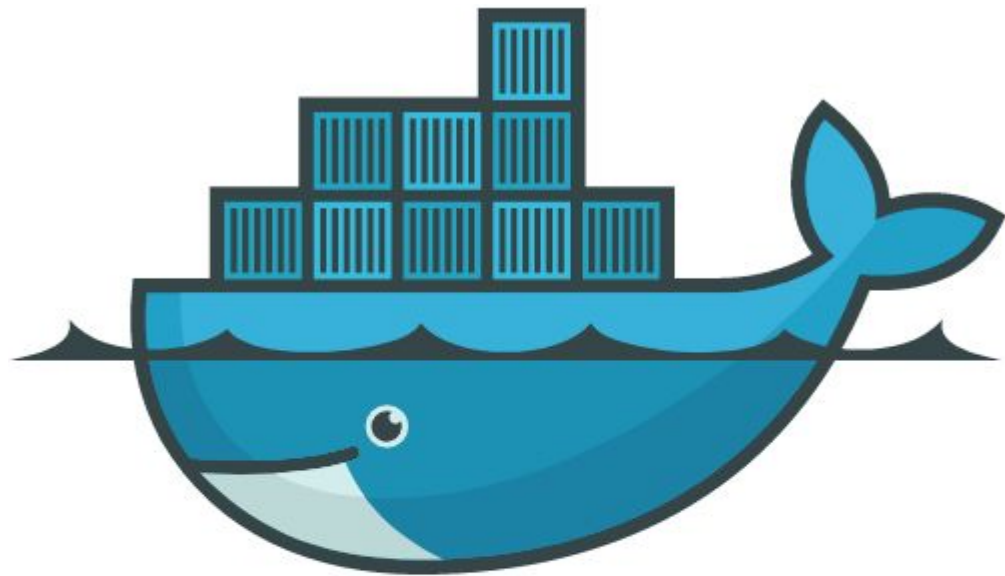
# Problema da virtualização

- Cada VM precisa ter um Sistema Operacional
- Cada SO consome memória RAM, disco e processamento
- Cada SO é preciso instalar softwares, liberar portas e cuidar da segurança
- O tempo de manutenção dessas VM's era muito grande
- Cada VM então tem um custo mínimo para seu processamento



Solução para estes  
problemas!!!???

Containers!



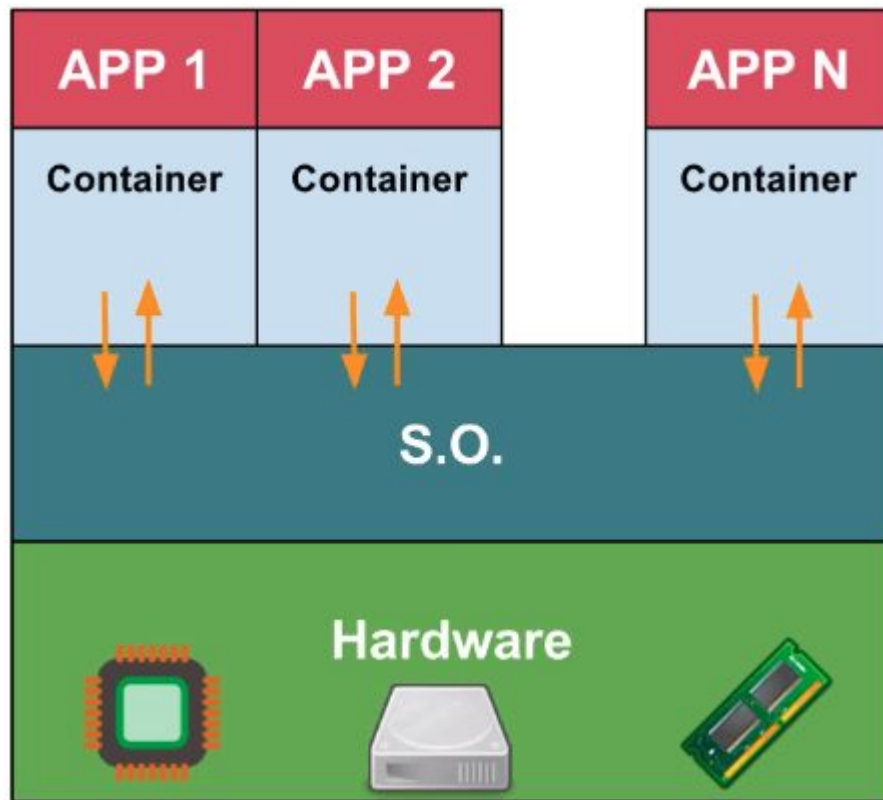
docker



# Containers

Um container funcionará junto do nosso sistema operacional base, e conterá a nossa aplicação, ou seja, a aplicação será executada dentro dele. Criamos um container para cada aplicação, e esses containers vão dividir as funcionalidades do sistema operacional.

**Somente um sistema operacional, reduzimos os custos de manutenção e de infraestrutura como um todo.**



# Vantagens dos containers

Por não possuir um sistema operacional, o container é muito mais leve e não possui o custo de manter múltiplos sistemas operacionais, já que só teremos um sistema operacional, que será dividido entre os containers.

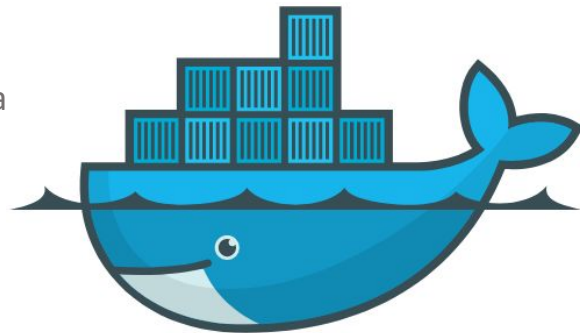
Além disso, por ser mais leve, o container é muito rápido de subir, subindo em questão de segundos.

**O container é uma solução para suprir o problema de múltiplas máquinas virtuais em um hardware físico, já que com o container, nós dividimos o sistema operacional entre as nossas aplicações.**



# Necessidade do uso de Containers

Mas por que precisamos dos containers, não podemos simplesmente instalar a aplicações no nosso próprio sistema operacional?



- E se dois aplicativos utilizarem a mesma porta de rede?
- E se uma aplicação consumir toda a CPU e prejudicar as demais?
- E se duas aplicações precisarem de versões diferentes de uma biblioteca?

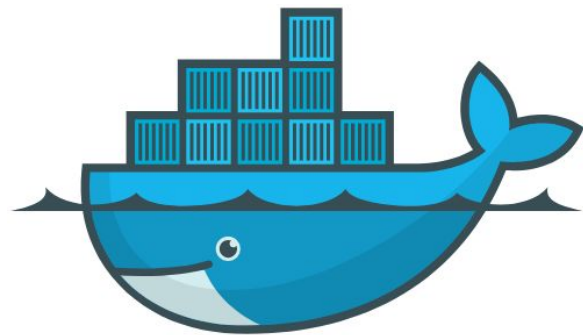
Ex: Java 7 ou Jac

Por isso é bom ter essa separação das aplicações, isolar uma da outra, e isso pode ser feito com os **containers**.



# Vantagens do uso de Containers

- Com os containers, conseguimos limitar o consumo de CPU das aplicações
- Melhorando o controle sobre o uso de cada recurso do nosso sistema (CPU, rede, etc).
- Temos uma facilidade maior em trabalhar com versões específicas de linguagens/bibliotecas,
- Temos uma agilidade maior na hora de criar e subir containers, já que eles são mais leves que as máquinas virtuais.



# Mão na massa!



# Instalando o Docker

Instalando docker no windows: <https://docs.docker.com/docker-for-windows/install/>

Instalando o docker no Mac: <https://docs.docker.com/docker-for-mac/install/>

Feito a instalação, execute esse comando no terminal `docker --version`. Se a instalação ocorreu com sucesso deve ser impresso algo semelhante a isso `Docker version 17.03.1-ce, build c6d412e`.

ou

Podemos utilizar o [Play With Docker](#), que é um playground interativo online para brincar e aplicar os comandos que veremos aqui! (Entrar no link e clicar em `ADD NEW INSTANCE`)





# Hello World

- Vamos checar se o docker está ok, com o comando: `docker version`
- Rodar o hello world no docker, com o comando: `docker run hello-world`
- Ao executar o comando, a primeira mensagem impressa é:

```
✖ aliniribeiro@Alinis-MacBook-Pro ~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
```

O QUE ISSO  
SIGNIFICA?



# Hello World

Significa que o Docker não conseguiu achar a imagem localmente, e ele foi em algum lugar e a baixou.

## Como assim?

Quando executamos o comando `docker run hello-world`, estamos dizendo para o Docker criar um **container** com a **imagem** do hello-world. Como não possuímos essa imagem localmente, ele foi buscá-la no **Docker Hub**, repositório do próprio Docker com várias imagens para utilizarmos em nossos projetos.

O QUE SÃO  
ESTAS  
SIGLAS??



# Imagens

A imagem é como se fosse uma receita de bolo, uma série de instruções que o Docker seguirá para criar um container, que irá conter as instruções da imagem, do hello-world.

Depois de buscar a receita do bolo, o container é criado e executado, e isso tudo pelo nosso comando **docker run hello-world**.

Há milhares de imagens disponíveis para executarmos a nossa aplicação, elas ficam disponíveis no [Docker Hub](#).



# Imagens

Também podemos apenas baixar nossa imagem, sem executar nada, utilizando o comando **docker pull** **hello-world**. Há milhares de imagens disponíveis para executarmos a nossa aplicação, elas ficam disponíveis no [Docker Hub](#). Exemplo a [imagem do de um projeto da alini](#): **docker pull aliniribeiroo/mvcad**

```
aliniribeiro@Alinis-MacBook-Pro ~$ docker pull aliniribeiroo/mvcad
Using default tag: latest
latest: Pulling from aliniribeiroo/mvcad
756975cb9c7e: Already exists
c3366ee73d64: Already exists
5225da16ddc9: Pull complete
b72f392fc89a: Pull complete
Digest: sha256:b83a43b0c6f14a07ad9576f844a3a3e6fe5990790834bd0ecd61988ee4d3333c
Status: Downloaded newer image for aliniribeiroo/mvcad:latest
docker.io/aliniribeiroo/mvcad:latest
aliniribeiro@Alinis-MacBook-Pro ~$
```

Aqui o Docker baixou nossa imagem. Percebam que uma imagem é composta de várias camadas, por esse motivo teve que fazer vários Downloads/Pull



# Listando Imagens

- Para listar todas as imagens podemos usar o comando: **docker images**

```
aliniribeiro@Alinis-MacBook-Pro ~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
aliniribeiro/mvcad	latest	6ba824a4f7e3	5 minutes ago	894MB

O nome da imagem é exibido na coluna **REPOSITORY**, cada imagem tem um identificador único que é exibido na coluna **IMAGE ID**. A coluna **TAG** indica a "versão" da imagem do ubuntu. O **latest** quer dizer que é a última "versão" da imagem (a mais recente).



# Executando containers

- Para executar um container podemos usar o comando: **docker run --name mvcad -p 5000:5000**  
**<nome da imagem>**

```
aliniribeiro@Alinis-MacBook-Pro ~$ docker run --name mvcad -p 5000:5000 aliniribeiro/mvcad
>>>>>>>> Estamos muito feliz que você chegou até aqui! Continue estudando que você vai longe <3 <<<<<<<<<
>>>>>>>> Execute um get, para chamar a api disponível neste projeto, a porta é a 5000 <<<<<<<<<
* Serving Flask app "main" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 129-803-226
```



# Listando containers

- Para listar todos os containers que estão em execução, podemos usar o comando: **docker ps**

```
aliribeiro@Alinis-MacBook-Pro ~ % docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
ef89121b54fc	aliribeiro/mvcad	"python main.py"	About a minute ago	Up About a minute	0.0.0.0:5000->5000/tcp	mvcad

Aqui temos informações sobre os containers em execução: ID, imagem base, comando inicial, há quanto tempo foi criado, status, quais portas estão disponíveis e/ou mapeadas para acesso e o nome do mesmo.

Quando não especificamos um nome ao iniciá-lo, será gerado um nome aleatoriamente. (Por isso usamos o comando `--name` quando rodamos o container anteriormente)



# Listando containers

- Para listar todos os containers que não estão em execução, podemos usar o comando: **docker ps -a**

```
aliniribeiro@Alinis-MacBook-Pro ~ % docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
ef89121b54fc	aliniribeiro/mvcad	"python main.py"	7 minutes ago	Up 7 minutes	0.0.0.0:5000->5000/tcp	mvcad

Depois de criado, um container pode ser iniciado e parado com os comandos:

- **docker start <nome container>**
- **docker stop <nome container>**





# Removendo containers e imagens

- Para remover um container, podemos usar o comando: **docker rm <nome ou id do container>**
- Para remover uma imagem, podemos usar o comando: **docker rmi <nome ou id da imagem>**

Dicas:

- Podemos apenas utilizar os três primeiros números dos ids de uma imagem ou container. Exemplo:  
container com o Nome mvcad e o id ef89121b54fc, podemos utilizar o comando:  
**docker <ação> ef8**
- Para forçar a exclusão de uma imagem ou container, podemos utilizar a tag -f no comando:  
**docker rmi -f 8ef**
- Para remover uma imagem, não pode haver nenhum container utilizando a mesma.



# Como são feitas as imagens?

Uma imagem pode ser criada a partir de um arquivo de definição chamado de [Dockerfile](#), nesse arquivo usamos algumas diretivas para declarar o que teremos na nossa imagem.

Lembra da nossa receita de bolo? é no Dockerfile que ela está escrita!

```
# Imagem base que o docker file vai extender, ou seja, ele vai
# Pegar uma imagem já existente do python e irá "complementar"
# ela com os nossos comandos
FROM python:3.8

# Fala qual o diretório do container que iremos trabalhar,
# não é uma boa pratica utilizar na pasta root.
WORKDIR /code

# Copia os arquivos que iremos utilizar para iniciar nosso projeto
COPY requirements.txt .
COPY main.py .

# instala as dependencias
RUN pip install -r requirements.txt

# Coloca a porta 5000 exposta para que a aplicação possa ser acessada
EXPOSE 5000

# Comando que irá rodar quando o container iniciar
CMD [ "python", "main.py" ]
```



# Como são feitas as imagens?

Depois de criar o arquivo Dockerfile, podemos construir nossa imagem com o comando:

**docker build -t <nome da imagem> .**

- "-t" é um parâmetro para informar que a imagem pertence ao meu usuário
- "." Significa o diretório em que você está, pois executamos o comando docker build dentro da pasta onde o Dockerfile se encontra.



Dica: Gere a imagem com o nome do seu usuário do dockerhub nas iniciais.

Exemplo:

`<user_dockerhub>/<nome_da_imagem>`

ou seja:

**aliniribeiroo/mvcad**



# Enviando imagem para DockerHub

Para enviar uma imagem para o dockerhub, podemos construir nossa imagem com o comando:

**docker push <nome da imagem>**



# Links complementares para estudar

- [Blog Mundo Docker](#)

