



- Conceitos iniciais
- Primeiro Programa
- Comentários
- Variáveis
- Leitura de Dados

Python

Lógica de Programação

Principais conceitos da Linguagem Python

01 Conceitos Iniciais

- Python é uma linguagem de programação, como Java, etc.
- IDE (*Integrated Development Environment*) é o ambiente onde você criará os seus sistemas
 - Inicialmente usaremos o Replit como IDE online
 - <https://replit.com/languages/python3>
 - Posteriormente, usaremos o Pycharm

02 Primeiro Programa

```
1 print('Olá, mundo!')
2 print(1)
```

- **print** é um comando (também conhecida por função) que imprime alguma coisa para o cliente
- No exemplo acima, vamos imprimir um texto, que vem entre aspas simples (''). E, logo depois, vamos imprimir o número 1, que não vem entre aspas simples porque é um número, não um texto

! Ao lado de cada uma das linhas do seu código, a IDE apresentará a linha. No exemplo acima, temos duas linhas. Linhas em branco não afetam o funcionamento do sistema.

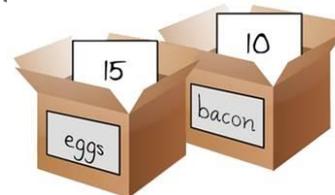
03 Comentários

```
1 # Meu comentário
2 print('Olá, mundo!')
3 #print(1)
```

- Comentários são linhas do código que são ignoradas pelo computador/compilador. Geralmente usamos para ignorar alguma parte do código que não queremos executar ou, para deixar comentários do nosso código
- No exemplo acima, as linhas 1 e 3 não serão executadas.

04 Variáveis

- Também conhecido por Atributos, variáveis representam uma caixinha da memória. Na sequência, temos a representação de duas caixinhas, `eggs` e `bacon`, que armazenam respectivamente os valores 15 e 10.



- A seguir, temos a declaração das duas variáveis supracitadas e a apresentação do valor destas variáveis.

```
1 eggs = 15
2 bacon = 10
3 print('nós temos', eggs, 'ovos e', bacon, 'kg de bacon')
```



Note que, a declaração da variáveis obedece ao formato: `variável = valor`
Não menos importante, verifique que, na linha 3 do código acima, imprimimos o valor das variáveis e texto, sempre **separados por vírgulas**.

- Existem 4 tipos de dados principais no Python, sendo eles: **int** (para números inteiros), **str** (String, para textos), **float** (para números quebradinhos, com casas decimais) e **bool** (Boolean, para guardar se é verdadeiro ou falso).
- Na sequência, criamos 4 variáveis para cada um dos tipos supracitados. Observe que, `salario` (float) possui casas decimais e a variável `nome` (str) possui aspas simples (podem ser duplas, também).

```
1 ano = 1985
2 nome = 'Daniel Abella'
3 salario = 10000.50
4 possuiEmprego = True
```

- As variáveis devem sempre iniciar com minúsculo. Vamos praticar?

04 Variáveis (Continuação)

- Para descobrir o tipo da variável, basta, dentro de um print, chamar o método `type(variável)`

```
6 print(type(ano))
7 print(type(nome))
8 print(type(salario))
9 print(type(possuiEmprego))
```

- Não te aperreie, não!** Vou deixar claro dois conceitos apresentados até aqui. Variáveis, são as caixinhas na memória e guardam valores. Por outro lado, métodos são funções que, você passa algo (ou não) e ele faz algo. Por exemplo, o print é um método que funciona da seguinte maneira:

- print(o que vai ser impresso)

```
1 ano = 1985
2 nome = 'Daniel Abella'
3
4 print('texto')
5 print(1)
6 print(ano)
7 print(nome)
```

- Nos exemplos acima, declaramos duas variáveis. Na sequência, imprimimos:

- O texto 'texto'
- O número 1
- O valor da variável ano, ou seja, 1985
- O valor da variável nome, ou seja, 'Daniel Abella'



Não muito comum, mas podemos criar duas variáveis e, na mesma linha, receber os valores, como no exemplo a seguir.

```
x, y = 10, 20;
```

A variável x receberá o valor 10, enquanto a variável y, receberá o valor 20. E, se é o mesmo valor para duas variáveis, podemos fazer a operação a seguir.

```
ESQUERDA ← x = y = 20; → DIREITA
```

05 Lendo Dados do Teclado

- Apresentamos antes como imprimir dados usando o `print` e, agora faremos como ler dados usando um teclado usando o método `input`.



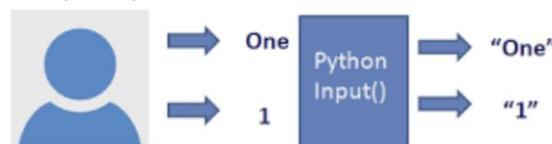
- O exemplo deste método é apresentado a seguir. Dentro do método `input`, passamos o texto que é apresentado ao usuário.
 - Quando o usuário informar o nome e clicar em `enter`, o valor informado é atribuído a variável nome.1

```
1 nome = input('digite o seu nome ')
2 print('o nome lido foi ', nome)
```

- O resultado da execução do código acima é apresentado a seguir.

```
digite o seu nome Daniel
o nome lido foi Daniel
>
```

- Uma limitação do `input`, é que, tudo que é lido, é convertido para texto (`str`, `String`). Ou seja, se eu quiser ler o meu ano de nascimento, precisarei realizar uma conversão de texto (`str`) para `int` (inteiro).



- Na sequência, temos os 3 principais métodos utilizados para a conversão. Os primeiros dois, converte de `str` para `int` e `float`, enquanto que o último converte de `int` ou `float` para `str`.

Método	Detalhes
<code>int(texto)</code>	Converte o <code>str</code> para inteiro
<code>float(texto)</code>	Converte o <code>str</code> para float
<code>str(inteiro, float, etc)</code>	Converte o <code>int</code> ou <code>float</code> para <code>str</code>

- Por fim, agora vamos mostrar na prática. No exemplo a seguir, convertamos o que está destacado em vermelho (ano lido pelo usuário em `str`) para inteiro. "Prestenção" aos parênteses.

- Parece complicado? Talvez, pois realizamos a leitura do teclado (`input`) com a conversão (`int`) ao mesmo tempo.

```
1 ano = int(input('digite o ano de nascimento '))
2 print('o seu ano foi ', ano)
```

- Achou complicado? É possível fazer a mesma operação em duas etapas, caso queiras. Na linha 1 a seguir, lemos o ano em texto (`str`) e, na linha 2, convertamos para `int`.

```
1 ano = input('digite o ano de nascimento ')
2 ano = int(ano)
3 print('o seu ano foi ', ano)
```



- Saída de Dados
- Operadores
- Strings
- Condicionais (if)

Python

Lógica de Programação

Principais conceitos da Linguagem Python

01 Saída de Dados

- Usamos a função print para apresentação dos dados
- No exemplo abaixo:
 - Na linha 1, a variável saldo recebe um valor com 5 casas decimais
 - Na linha 2, exibimos o saldo com as 5 casas decimais
 - Na linha 3, imprimimos uma linha em vazio
 - Por fim, na linha 4, exibimos o mesmo saldo (da linha 2), mas com apenas 2 casas decimais
 - Verifique que, na parte `%.2f`, o `f` se refere a *float*, enquanto que o número 2, se refere a quantidade de casas decimais
 - Se fosse `int`, seria `d` (ao invés de `f`)

```
1 saldo = 1000.50123
2 print(' o saldo é ',saldo)
3 print()
4 print(" o saldo é %.2f " %saldo)
```

```
1 nome = 'Daniel '
2 sobrenome = 'Abella'
3 nomeCompleto = nome + sobrenome
4 print(' o meu nome completo é', nomeCompleto)
```

- No exemplo acima, criamos duas variáveis (nome e sobrenome) e, uma terceira variável que é a junção das duas anteriores. A esta operação de junção, comumente chamamos de concatenação.
 - A concatenação deve sempre ser entre `str` (Strings) e inteiros, sem poder misturar. Ou seja, o exemplo a seguir (misturando `str` e `int`) não funciona em Python.

```
1 nome = 'Daniel '
2 idade = 26
3 nomeComIdade = nome + idade
4 print(' nome com idade é ', nomeCompleto)
```



- No exemplo a seguir, apresentamos as principais operações representadas na tabela anterior.

1	<code>print(3+2)</code>	5
2	<code>print(3-2)</code>	1
3	<code>print(3*2)</code>	6
4	<code>print(3/2)</code>	1.5
5	<code>print(3**2)</code>	9
6	<code>print(3%2)</code>	1
7	<code>print(22//8)</code>	2

02 Operadores Matemáticos

- Na tabela a seguir, apresentamos os principais operadores matemáticos disponíveis no Python.

Operador	Operação	Exemplo
<code>**</code>	Potência	<code>2 ** 3 = 8</code>
<code>%</code>	Módulo	<code>5 % 2 = 1</code>
<code>//</code>	Divisão inteira	<code>22 // 8 = 2</code>
<code>*</code>	Multiplicação	<code>3 * 8 = 24</code>
<code>-</code>	Subtração	<code>8 - 2 = 6</code>
<code>+</code>	Soma	<code>2 + 2 = 4</code>
<code>/</code>	Divisão	<code>33 / 4 = 8.25</code>

- O operador `+`, que se refere obviamente a soma, pode ser utilizado em números (*int* e *float*), bem como em strings (*str*).

- Como sempre, as mesmas operações feitas acima com literais, podem ser feitas com uso de variáveis (lidas ou não pelo usuário)

```
1 membro1 = 3
2 membro2 = 2
3
4 print(membro1+membro2)      5
5 print(membro1-membro2)     1
6 print(membro1*membro2)     6
7 print(membro1/membro2)     1.5
8 print(membro1**2)          9
9 print(membro1%membro2)     1
10 print(membro1//membro2)   1
```

03 Trabalhando com Strings (str)

- Para armazenar caracteres (que ainda assim, podem ser números), usamos o tipo str (String). No exemplo a seguir, apresentamos um exemplo do uso de Strings.
 - Para declarar Strings, basta atribuir um valor entre aspas simples ou duplas a uma variável
- Nas linhas 1 a 3, declaramos 3 variáveis. Verifique que, a variável idade, tende a ser int, mas ainda assim, podemos declarar como string, pois a colocamos entre aspas
- Nas linhas seguintes, apenas apresentamos os textos concatenados

```
1 nome = 'Daniel '  
2 sobrenome = 'Abella'  
3 idade = '36'  
4  
5 print(nome + sobrenome)  
6 print('Daniel' + 'Abella')  
7 print('possui a idade ', idade)
```

- Para finalizar, no exemplo a seguir apresentamos o uso do método **len**, que se refere a length, que é o tamanho da String. Note que, imprimimos o tamanho de duas maneiras o tamanho da String 'Daniel', que possui 6 caracteres.
 - Na primeira (linha 2), chamamos diretamente o método len durante o print
 - A segunda abordagem (linhas 4 e 5), atribui o tamanho da String a uma variável antes de entrar no print.

```
1 nome = 'Daniel'  
2 print('o tamanho do nome tem', len(nome), 'caracteres')  
3  
4 tamanhoTexto = len(nome)  
5 print('o tamanho do nome tem', tamanhoTexto, 'caracteres')
```

- O uso de operadores matemáticos com Strings é funcional
 - O operador soma, funciona para juntar 2 Strings (concatenar)
 - Os operadores subtração e divisão não possuem funcionamento com Strings
 - Como vou fazer 'Daniel'/?? `Daniel`-2?
 - O operador multiplicação possui uso no Python
 - `Daniel`*3 exibe DanielDanielDaniel
 - O exemplo a seguir demonstra o funcionamento do que acabamos de discutir. As linhas 1,2,3 e 6 funcionam, enquanto todas as outras apresentam erro.

```
1 nome = 'Daniel '  
2 sobrenome = 'Abella'  
3 print(nome + sobrenome)  
4 print(nome + 3)  
5 print(nome - 3)  
6 print(nome * 3)  
7 print(nome / 3)
```

- Dentre as operações mais comuns com Strings, são a de tomar uma string em maiúscula, minúscula ou substituir um dado caractere por outro. Um exemplo das 3 operações é apresentado a seguir:
 - Na linha 3, colocamos 'Daniel' em maiúsculo
 - Na linha 6, em minúsculo
 - Na linha 9, substituímos o caractere 'd' de Daniel por 'r', tomando-o 'Raniel'

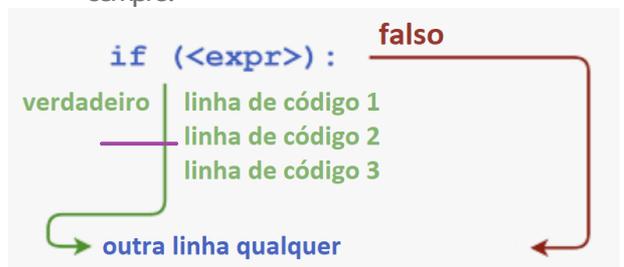
```
1 nome = 'Daniel '  
2  
3 nome = nome.upper()  
4 print(nome)  
5  
6 nome = nome.lower()  
7 print(nome)  
8  
9 nome = nome.replace('d','r')  
10 print(nome)
```

04 Condicionais (If)

- Outra coisa muito importante em programação é o uso de condicionais e aqui vamos aprender o uso do IF (Se, em português). No exemplo a seguir, apresentamos o texto 'é maior de idade' caso a variável idade tenha valor >= (maior ou igual) a 18.

```
1 idade = 18  
2  
3 if idade >= 18 :  
4 |   print('é maior de idade')  
5   print('fim do programa')
```

- Ainda com relação ao exemplo anterior, verifique que, o print da linha 4, possui um recuo para indicar que, esse print só será executado se a condição do if (maior de 18) for verdadeira.
 - E, neste exemplo, é verdadeiro 😊
- A linha 5, independente do if ser executado ou não, será impressa, pois não possui um recuo, de modo que, entende-se que não pertence ao if e será executado sempre.
- Para melhor entendimento do exemplo, apresentamos a seguir o funcionamento do if.
 - Note que, a linha em roxo representa o recuo necessário, indicando que linhas de código 1, 2 e 3 pertencem ao if (em azul)
 - E, a linha "outra linha qualquer" será apresentada sempre.





- Condicionais (If)
- Operadores de Comparação
- Else (Senão)
- Elif
- Elif na Prática

Python

Lógica de Programação

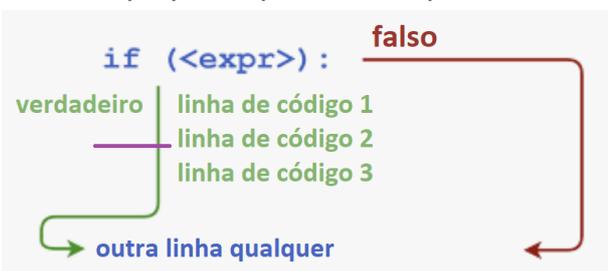
Principais conceitos da Linguagem Python

01 Condicionais (If)

- Uma estrutura muito importante na programação são os condicionais e, neste momento, nos aprofundaremos sobre o comando IF.
- Para entendermos melhor o IF, tomemos como exemplo o código a seguir.

```
1 idade = 67
2
3 if idade >= 65:
4     print('acesso gratuito ao Ônibus')
5     print('fim do programa')
```

- No código acima, temos os seguintes destaques:
 - Na linha 3, temos o nosso condicional IF
 - A condição é se a idade é maior ou igual a 65
 - Ao término da condição, temos um dois pontos (:)
 - A linha 4, verificamos um recuo (espaço), que indica que, esta linha, “pertence” ao IF da linha 3, de modo que, esta linha 4 só será exibida se o IF da linha 3 for verdadeiro (neste exemplo é verdadeiro)
 - A linha 5, como pode-se perceber, não tem espaço, de modo que é executada independente do resultado (verdadeiro ou falso) do IF.
- Para alinhar o entendimento, apresentamos novamente a imagem a seguir. Se a expressão do IF for verdadeira, as linhas de código 1, 2 e 3 são executadas. Enquanto que, a linha “outra linha qualquer” é apresentada sempre.



- Não siga para os próximos conteúdos sem que este conceito esteja armazenado na sua cabeça

02 Operadores de Comparação

- Na seção anterior, usamos o operador de comparação >= para verificar se a idade é maior ou igual a 65, lembram? Na sequencia temos a lista dos principais operadores de comparação existentes no Python.

Operador	Operação	Exemplo
==	Igualdade	If idade == 18:
!=	Diferente	If idade != 18:
>	Maior que	If idade > 18:
<	Menor que	If idade < 18:
>=	Maior ou igual que	If idade >= 18:
<=	Menor ou igual que	If idade <= 18:

- Para consolidar o que foi apresentado anteriormente, no programa a seguir, relacionamos o uso de todos os operadores de comparação:

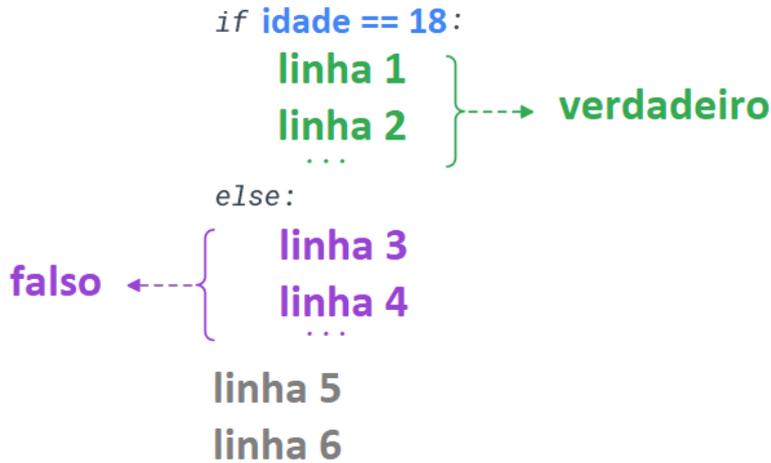
```
1 idade = 67
2
3 if idade == 18:
4     print('você tem 18 anos')
5
6 if idade != 18:
7     print('você não tem 18 anos')
8
9 if idade > 18:
10    print('você tem mais de 18 anos')
11
12 if idade < 18:
13    print('você tem menos de 18 anos')
14
15 if idade >= 18:
16    print('você tem 18 anos ou mais')
17
18 if idade <= 18:
19    print('você tem 18 anos ou menos')
```

Saída

```
você não tem 18 anos
você tem mais de 18 anos
você tem 18 anos ou mais
```

03 Else (Senão)

- Else, em português, significa SENÃO. É usado para executar um bloco de código caso a condição do IF seja falsa. O ELSE tem funcionamento apresentado a seguir



- No exemplo acima, se a idade for EXATAMENTE 18, as linhas 1 e 2 serão apresentadas, bem como as linhas 5 e 6 (não por causa do IF);
- Caso a idade seja 17, 19 ou outra DIFERENTE de 18, as linhas 3, 4, 5 e 6 serão executadas.

As linhas 5 e 6 sempre serão executadas independente do IF e do ELSE, pois eles não estão relacionados a eles, devido a ausência de recuo (espaço). Por outro lado, as linhas 1 e 2 são apresentadas se o IF for verdadeiro, enquanto as linhas 3 e 4 são apresentadas se o IF for falso.

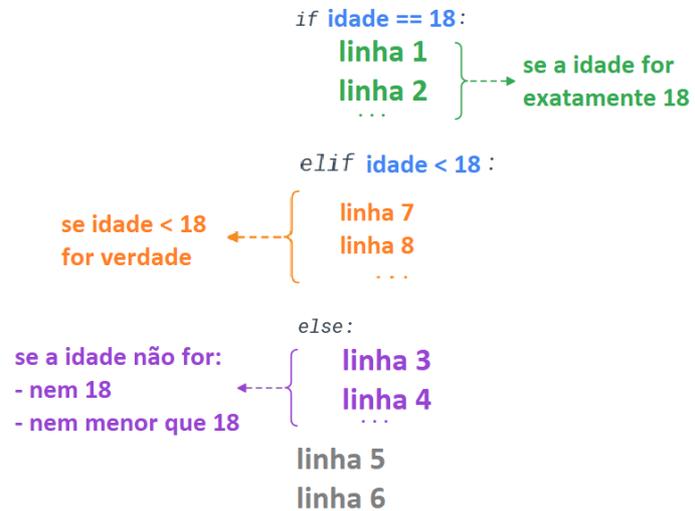
```
1 variavelA = 10
2 variavelB = 20
3
4 if variavelA > variavelB:
5     print('variável A é maior')
6 else:
7     print('variável B é maior')
```

```
1 variavelA = 20
2 variavelB = 10
3
4 if variavelA > variavelB:
5     print('variável A é maior')
6 else:
7     print('variável B é maior')
```

- Acima demonstramos o funcionamento do IF e ELSE em conjunto.
 - Na imagem à esquerda do exemplo acima, a linha 7 é apresentada
 - Na direita, em que os valores são ao contrário (variavelA com 20 e variavelB com 10), a linha 5 é apresentada
 - E, se as DUAS variáveis tivessem o MESMO valor, o que aconteceria? Não exibiria NENHUMA mensagem.

04 Elif

- Usamos o ELIF (Else IF) para informar uma nova condição a ser testada, caso a primeira condição (no IF) for falsa. O funcionamento é apresentado a seguir.



- Com base na imagem acima, caso a idade for EXATAMENTE 18, as linhas 1,2,5 e 6 são executadas
- Caso a idade seja menor que 18 (ou seja, 17, 16, ...), as linhas executadas serão as 7,8,5 e 6
- Por fim, caso a idade seja maior de 18 (uma vez que 18 é atendida no IF, menor que 18 é atendida no ELIF), as linhas executadas serão 3,4,5 e 6

05 Elif na Prática

- Comumente calculamos o Índice de Massa Corpórea (IMC) para verificar se estamos no peso ideal. A fórmula é a seguinte: $IMC = \frac{\text{peso}}{\text{altura} \times \text{altura}}$
- Complementarmente, o resultado do cálculo pode ser interpretado com base na tabela a seguir.

IMC	Classificação
< 16	Magreza grave
16 a < 17	Magreza moderada
17 a < 18,5	Magreza leve
18,5 a < 25	Saudável
25 a < 30	Sobrepeso
30 a < 35	Obesidade Grau I
35 a < 40	Obesidade Grau II (severa)
≥ 40	Obesidade Grau III (mórbida)

- O código para cálculo do IMC com base na tabela supracitada, pode ser encontrado em: <https://danielabella.com/unifacisa/p1/codigos/imc.py>





- If com AND
- Mais sobre IF

Python

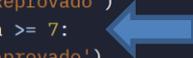
Lógica de Programação

Principais conceitos da Linguagem Python

01 If com AND

- Podemos no mesmo IF ou ELIF avaliar duas ou mais expressões com uso do AND e do OR. Nesta seção, apresentaremos o AND.
- Para melhor entendimento, vamos criar um programa que leia duas notas de um aluno e:
 - Apresente 'Reprovado', caso a média seja inferior a 7
 - Apresente 'Aprovado', caso a média seja maior que 7
 - Em casos de média 10, apresente 'Aprovado com Distinção'
- Teoricamente, o primeiro pensamento é criar algum dos 2 programas abaixo. Entretanto, se você tiver a média 10, não será apresentado 'Aprovado com Distinção', mas 'Aprovado', por que?

```
1 nota1 = float(input('digite a nota 1 '))
2 nota2 = float(input('digite a nota 2 '))
3 media = (nota1+nota2)/2
4
5 if media < 7:
6     print('Reprovado')
7 elif media >= 7:
8     print('Aprovado')
9 else:
10    print('Aprovado com distincao')
```



```
1 nota1 = float(input('digite a nota 1 '))
2 nota2 = float(input('digite a nota 2 '))
3 media = (nota1+nota2)/2
4
5 if media < 7:
6     print('Reprovado')
7 elif media >= 7:
8     print('Aprovado')
9 elif media == 10:
10    print('Aprovado com distincao')
```



```
1 nota1 = float(input('digite a nota 1 '))
2 nota2 = float(input('digite a nota 2 '))
3 media = (nota1+nota2)/2
4
5 if media < 7:
6     print('Reprovado')
7 elif media >= 7 and media < 10:
8     print('Aprovado')
9 elif media == 10:
10    print('Aprovado com distincao')
```

- Verifiquem que, na linha 7, queremos que, seja verdadeiro se a media seja maior ou igual a 7 E media menor que 10.
 - Estão vendo a palavra E, que em Inglês significa And?
 - Você sabe a diferença entre E (And) e OU (Or)?



Imagine que, ao irmos ao supermercado, temos uma parte com maçã e banana.



E, em dado momento, seu colega peça: “Eu quero banana E maçã”. Se você entregar banana apenas, ele não aceitará. Se você entregar maçã apenas, ele também não aceitará. Ele aceitará se você entregar banana e maçã, as duas!

Entretanto, se o seu amigo peça “Quero banana OU maçã”. Isto significa que, ele aceita banana apenas, como também aceita maçã apenas também. E, se você chegar com banana e maçã, ele também aceita.

Toda esta explicação está representada na tabela a seguir.

A	B	A e B	A ou B
F	F	F	F
F	V	F	V
V	F	F	V
V	V	V	V

- A resposta é porque, a média 10 atende a restrição do primeiro ELIF. Como assim? A média é 10, que é maior ou igual (>=) a 7, então entra no ELIF indicado pela seta azul acima.
 - O que queremos é: Se a média for maior ou igual a 7, mas menor que 10, apresente 'Aprovado'
 - E, se for 10, apresente 'Aprovado com Distinção'
 - Na sequencia, apresentaremos a solução!

02 Mais sobre IF

- Nesta seção, discutiremos as principais dificuldades e problemas com relação ao uso do IF, sendo elas:
 - Item 1: Espaço no IF
 - Item 2: Escrever a condição errada com And
 - Item 3: Diferença entre IF, IF, IF e IF, ELIF, ELIF e ELSE
 - Item 4: Diferença entre = e ==

Item 1: Espaço no IF

- Apresentamos a seguir a estrutura do IF, ELIF e ELSE. Verifique que os espaços, destacados em amarelo, vermelho e verde, impactam o funcionamento destas estruturas.
 - Por exemplo, as linhas 2 e 3 possuem um recuo à direita para dizer que só serão executadas se o IF for verdadeiro



- A seguir temos um exemplo de como NÃO funciona (à esquerda, sem espaços) e como funciona (à direita, com espaços).

```
1 idade = 18
2 if idade == 18:
3 print('tem 18 anos')
```



```
1 idade = 18
2 if idade == 18:
3 | print('tem 18 anos')
```



Item 2: Condição Errada com And

- Se eu quiser imprimir algo para adultos (maiores de 18 anos e menores de 65 anos), o IF será o apresentado a seguir.

```
1 idade = 18
2 if idade >= 18 and idade < 65:
3 | print('é adulto, mas não é idoso')
```

- Verificou que, na condição esquerda ($idade \geq 18$) e na condição direita ($idade < 65$), em ambas a variável `idade` é utilizada? E é assim mesmo! Entretanto, alguns fazem da maneira errada a seguir.

```
1 idade = 18
2 if idade >= 18 and < 65:
3 | print('é adulto, mas não é idoso')
```



- Ficou faltando o `idade` na condição a direita. O grande ponto é: no if `<condição> and <condição>`, a condição à esquerda nada tem a ver com a condição à direita. Nada a ver um com o outro!

Item 3: Diferença entre usar ifs e Elifs

- Para criar as nossas condições no código, geralmente pensamos: vamos usar vários IF's OU usar a combinação de um IF com ELIF's acompanhado opcionalmente de um ELSE? No exemplo a seguir, mostramos um exemplo que vai deixar claro a diferença entre eles.

```
1 idade = 18
2
3 if idade == 18:
4 | print('tem 18 anos')
5
6 if idade >= 16:
7 | print('adolescente ou adulto')
8
9 if idade >= 65:
10 | print('idoso')
```

```
1 idade = 18
2
3 if idade == 18:
4 | print('tem 18 anos')
5 elif idade >= 16:
6 | print('adolescente ou adulto')
7 elif idade >= 65:
8 | print('idoso')
```

- Na parte à esquerda do exemplo acima, as linhas 4 e 7 serão impressas, porque os dois ifs (das linhas 3 e 6) deram verdadeiro. Entretanto, à direita, apenas a linha 4 foi apresentada ao usuário.
- Por que isso aconteceu? Porque à esquerda, quando usamos vários ifs, eles não possuem relação um com o outro, de modo que, sempre que der verdade, vai imprimir.
- Por outro lado, à direita, só tem UM IF e vários ELIF'S e, caso quiséssemos, poderíamos ter um ELSE. Desta maneira, entende-se que, os ELIF's e o ELSE (se tivesse) pertencem ao IF.
- Sendo assim, assim que a primeira condição der verdadeiro, seja no IF ou no ELIF, ele o conteúdo do IF e não procura mais nenhuma outra condição.
- No exemplo acima, uma vez que o IF da linha 3 foi verdadeiro, ele imprime a linha 4 e ponto final. Isto é, não executa de forma alguma as linhas 5,6,7 e 8.

Item 4: Diferença entre = e ==

- De maneira objetiva, o operador `=` serve para atribuir valores, como apresentamos a seguir.

```
1 idade = 18
2 idadeProfessor = idade * 2
3 idadeAluno = idade
```

- Por outro lado, o operador `==` se refere a comparação, que não necessariamente precisa estar dentro de um IF ou ELIF, mas que é onde geralmente vamos colocar. Ou seja, em IF é só ==

```
1 idade = 18
2 maiorIdade = idade == 18
3
4 if maiorIdade:
5 | print('maior de idade')
6
7 if idade == 18:
8 | print('maior de idade')
9
10 print('eu sou maior de idade?', idade == 18)
```

- Com relação ao exemplo acima, na linha 2, verificamos se a idade é 18. Desta maneira, a variável `maiorIdade` vai receber `True` (verdadeiro) ou `False` (falso). Deste modo, usamos a variável na linha 4. Caso não quiséssemos usar, faríamos como na linha 7.



- IF com OR
- Negação e Diferença
- IF aninhados
- Função Range

Python

Lógica de Programação

Principais conceitos da Linguagem Python

01 If com Or (Ou)

- Como vimos na *sheet* anterior, os operadores And (E) e Or (OU) funcionam da maneira ilustrada a seguir. O operador And (E), só é verdadeiro se o lado esquerdo representado por A bem como o lado direito representado por B sejam verdadeiros.

A	B	A e B	A ou B
F	F	F	F
F	V	F	V
V	F	F	V
V	V	V	V

- Por outro lado, o operador Or (OU), só é verdadeiro se um dos lados (esquerdo ou direito) forem verdadeiros. Para finalizar, analise o código a seguir e tire suas conclusões.

```

1 print(1==1 and 1==2) False
2 print(2==1 and 1==2) False
3 print(3>=1 and 1<=2) True
4
5 print(1==1 or 1==2) True
6 print(2==1 or 1==2) False
7 print(3>=1 or 1<=2) True

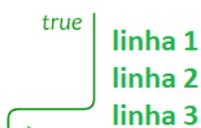
```

Saída

- Considerando a idade = 18, no código a seguir, as linhas 1,2,3,4 e 5 serão executadas, pois:

- O lado esquerdo (destacado em laranja) é verdadeiro, pois idade é maior ou igual a 18
- E, o lado direito (destacado em verde) também é verdadeiro, pois idade é menor que 65.
- Então, como os dois lados são verdadeiros, dá certo, pois o operador And exige que os dois lados (laranja e verde) sejam verdadeiros.

```
if idade >= 18 and idade < 65 :
```



linha 4
linha 5

- E se fosse o OR (Ou)? No exemplo a seguir, se a idade for 70, as linhas 1,2,3,4,5 serão apresentadas, pois a condição à esquerda (laranja) será verdadeira e à direita (verde) será falsa. E, para o operador OR (Ou), apenas um dos lados precisa ser verdadeiro.

```
if idade >= 18 or idade < 65 :
```



linha 4
linha 5

02 Combinando And e Or

- Podemos combinar os operadores And e Or, como veremos no exemplo a seguir, em que a linha 5 é executada.

```

1 idade = 13
2 salary = 10000
3
4 if (idade > 18 and idade < 65) or salary >= 10000:
5     print('adulto ou qualquer um com salario acima de 10k')

```

- A primeira expressão a ser executada é a que está entre os parênteses, destacado em vermelho, devido a precedência. A condição esquerda (idade > 18) é falsa e a direita (idade < 65) é verdadeira, de modo que falso e verdadeiro resulta em falso.
- Uma vez analisado os parênteses, é hora de analisar o lado direito (salary >= 10000), que é verdadeiro. Por fim, devido a estarmos em uma condição de OR, o resultado final é verdadeiro.

```

VERDADEIRO
      FALSO      OU      VERDADEIRO
      FALSO      E      VERDADEIRO      VERDADEIRO
4 if (idade > 18 and idade < 65) or salary >= 10000:
5     print('adulto ou qualquer um com salario acima de 10k')

```

03 Negação e Diferença

- O operador not permite que você inverta o valor de expressões e objetos booleanos. Ou seja, se algo era true, vira false e vice-versa.
- Você pode usar esse operador em contextos booleanos, como if e while, como também não booleanos.

03 Negação e Diferença (Continuação)

- Chamamos o operador NOT como negação e, conforme dito anteriormente, inverte os valores booleanos. Na sequência apresentamos o funcionamento do operador NOT na prática.

```
1 print(True)           True
2 print(not True)       False
3
4 print(False)          False
5 print(not False)      True
6
7 estaEmpregado = True
8 print(estaEmpregado)  True
9 print(not estaEmpregado) False
```

Saída

- O operador NOT pode estar associado a IFs, como no exemplo a seguir. Caso o operador NOT não tivesse sido utilizado, a linha 4 não seria apresentada.

```
1 idade = 17
2
3 if not idade >= 18:
4     print('menor de idade')
```

Saída menor de idade

- Por fim, vamos apresentar o operador de DIFERENTE (!=). Se a idade for exatamente 18, a linha 4 será apresentada. Caso a idade seja diferente (!=) de 18, a linha 6 será apresentada.

```
1 idade = 17
2
3 if idade == 18:
4     print('tem 18 anos')
5 elif idade != 18:
6     print('não tem 18 anos')
```

Saída não tem 18 anos

04 IFs aninhados

- É possível ter uma instrução if/elif/else dentro de outra instrução if/elif/else, o que chamamos de IFs aninhados, em inglês, nested IFs statements. Este fato, pode diminuir a legibilidade do código e aumentar a confusão no entendimento, de modo que deve ser evitado.
- Tomemos como exemplo o código apresentado a seguir.

```
1 num = float(input("Informe um número "))
2
3 if num >= 0:
4     if num == 0:
5         print("Zero")
6     else:
7         print("Número positivo")
8 else:
9     print("Número negativo")
```

- Notem que, dentro do IF num >= 0, temos duas instruções:
 - IF num == 0 e um else
 - Caso na linha 1, informássemos o valor 0 no teclado, teríamos o seguinte funcionamento:
 - **Linha 3:** If num >= 0 será verdadeiro, de forma que entra no IF
 - **Linha 4:** If num == 0 será executado, uma vez que a linha 3 é verdadeira. E, como num é zero, a sentença é verdadeira e a linha 5 é executada.
 - A seguir, destacamos em vermelho o fluxo da execução para o cenário supracitado.

```
1 num = float(input("Informe um número "))
2
3 if num >= 0:
4     if num == 0:
5         print("Zero")
6     else:
7         print("Número positivo")
8 else:
9     print("Número negativo")
```

- Para completar o entendimento, experimente executar o programa acima com valores positivos e negativos.

05 Função Range

- A função range retorna uma sequência de números e é comumente utilizada com o comando FOR, que aprenderemos na sequência. Existe 3 opções (“sabores”) para a função range:
 - range(n) na qual gera números de 0 até n-1
 - Por exemplo, range(3) gera 0,1 e 2
 - range(a,n) na qual gera números de a até n-1
 - Por exemplo, range(1,3) gera 1 e 2
 - range(a,n,s) na qual gera números de a até n-1, pulando de s em s
 - Por exemplo, range(0,5,2) em teoria geraria 0,1,2,3,4. Entretanto, o último parâmetro (s) que tem o valor 2 significa que ele pula de 2 em 2. Como assim? No 0, ele pula diretamente para 2 e do 2 exatamente para o 4, de modo que os 1 e 3 são desconsiderados.

```
1 for n in range(0,5,2):
2     print(n)
3
4 for n in range(0,5):
5     print(n)
6
7 for n in range(5):
8     print(n)
```



- Função Range
- FOR
- FOR com ELSE
- Exemplos com FOR
- WHILE

Python

Lógica de Programação

Principais conceitos da Linguagem Python

01 Função Range

A função range retorna uma sequência de números e é comumente utilizada com o comando FOR, que aprenderemos na sequência. Existe 3 opções (“sabores”) para a função range:

- **range(n)** na qual gera números de 0 até n-1
 - Por exemplo, range(3) gera 0,1 e 2
- **range(a,n)** na qual gera números de a até n-1
 - Por exemplo, range(1,3) gera 1 e 2
- **range(a,n,s)** na qual gera números de a até n-1, pulando de s em s
 - Por exemplo, range(0,5,2) em teoria geraria 0,1,2,3,4. Entretanto, o último parâmetro (s) que tem o valor 2 significa que ele pula de 2 em 2. Como assim? No 0, ele pula diretamente para 2 e do 2 exatamente para o 4, de modo que os 1 e 3 são desconsiderados.

```
1 print(range(5))
2 print(list(range(5)))
3 print(list(range(0,5)))
4 print(list(range(0,5,2)))
```

Saída

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

Note que, no exemplo acima, diferente da folha anterior, não usamos for. A primeira linha não exibe o resultado esperado, que é uma lista. Para isto, usamos o método list(), que converte o range() em uma lista, tornando-o apresentável no print.

02 Operador FOR

O operador FOR é usado para iterar sobre uma sequência (lista, dupla, String), entre outros objetos iteráveis. O formato é o seguinte.

```
for var in sequencia:
    corpo do loop (linha 1)
```

```
1 frutas = ['maça', 'banana', 'abacaxi']
2 for fruta in frutas:
3     print(fruta)
4     print('mais uma linha dentro do FOR')
5     print() #pula linha
6 print('fim do FOR')
```

Saída

```
maça
mais uma linha dentro do FOR
banana
mais uma linha dentro do FOR
abacaxi
mais uma linha dentro do FOR
fim do FOR
```

Acima temos um exemplo comum do uso do FOR. Na linha 1, declaramos uma lista de `frutas`, contendo 3 frutas. Na linha 2, usamos o FOR, onde `fruta` é uma variável temporária, que guarda o valor de cada fruta da lista. Como temos 3 itens na lista `frutas`, o for executará 3 vezes, de modo que as linhas 3,4 e 5 serão executadas 3 vezes.

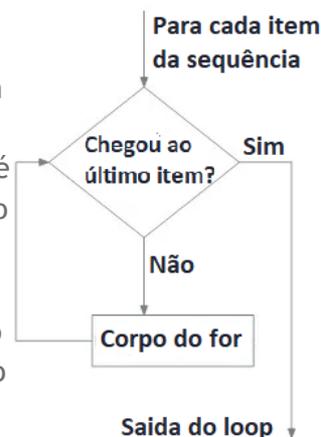
- Na primeira iteração, a linha 3 imprimirá maça, depois imprime a linha 4 e pula a linha (linha 5)
- Na segunda iteração, a linha 3 imprimirá banana, depois imprime a linha 4 e pula a linha (linha 5)
- a linha 3 imprimirá abacaxi, depois imprime a linha 4 e pula a linha (linha 5)

O funcionamento do FOR é apresentado à direita. Para cada item da sequência (frutas), antes de executar os comandos, verifica-se se é o último elemento da sequência. Caso não seja, executa-se o corpo do for (linhas 3,4 e 5).

Quando alcançar o fim da lista, pula o corpo do for e segue o funcionamento do programa (linha 6 e seguintes)

Verifique o código a seguir

```
1 for n in range(0,5,2):
2     print(n)
3
4 for n in range(0,5):
5     print(n)
6
7 for n in range(5):
8     print(n)
```



03 Operador FOR com ELSE

- Em um FOR é possível (mas não muito comum) termos um ELSE, como exemplificado na imagem a seguir. Quando terminamos o FOR das linhas 2 a 5, o ELSE da linha 6 (e consequentemente a linha 7) serão executados.
- Ao término da execução da linha 7, a linha 8 é executada.

```
1 frutas = ['maçã', 'banana', 'abacaxi']
2 for fruta in frutas:
3     print(fruta)
4     print('mais uma linha dentro do FOR')
5     print() #pula linha
6 else:
7     print('else do FOR')
8 print('fim do FOR')
```

04 Mais sobre FOR (Exemplos)

- **Exemplo #1:** Na linha 1, declaramos uma lista com 4 elementos, enquanto na linha seguinte, declaramos uma variável soma, que vai armazenar a soma de todos os elementos de uma lista.

```
1 numeros = [6, 5, 3, 8]
2 soma = 0
3
4 for num in numeros:
5     soma = soma+num
6
7 print('A soma é', soma)
```

- Na linha 4, iniciamos o for e para cada um dos números da lista, executamos a linha `soma = soma+num` que basicamente:
 - Pega o valor que tem em soma e acresce a um número da lista
 - Ao término do for, a variável soma terá o valor da soma de todas as variáveis.

- **Exemplo #2:** No exemplo a seguir, usamos o FOR para imprimir a palavra banana, letra a letra.

```
1 for letra in "banana":
2     print(letra)
```

- **Exemplo #3:** Com o `continue` podemos parar a iteração atual do loop e continuar com a próxima. No exemplo a seguir, quando ele encontrar banana (linha 3), vai pular para o próximo elemento.

```
1 frutas = ['maçã', 'banana', 'abacaxi']
2 for fruta in frutas:
3     if fruta == 'banana':
4         continue
5     print('a fruta é',fruta)
6     print('mais uma linha dentro do FOR')
7     print() #pula linha
```

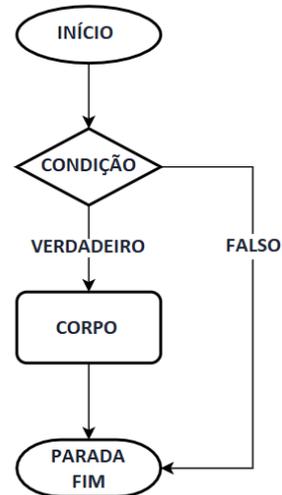
- **Exemplo #4:** Com a instrução `break` podemos interromper o loop antes que ele tenha percorrido todos os itens, como no exemplo a seguir. Exibimos maçã e depois banana. Abacaxi não será exibido.

```
1 frutas = ['maçã', 'banana', 'abacaxi']
2 for fruta in frutas:
3     print('a fruta é',fruta)
4     print('mais uma linha dentro do FOR')
5     print() #pula linha
6     if fruta == 'banana':
7         break
8     print('outra linha')
```

05 Operador WHILE

- WHILE (Enquanto, em português) permite que você execute um bloco de código (corpo) repetidamente enquanto uma condição for True (verdadeira), conforme exemplo a seguir.

```
1 while condicao
2     corpo - linha de código 1
3     corpo - linha de código 2
```



- A direita temos o funcionamento do WHILE, que verifica a condição no início de cada iteração.
- Executará o corpo enquanto a condição for True. No corpo do loop, você precisa fazer algo para interromper o loop em algum momento.
- Senão, você entra em um LOOP infinito e o aplicativo não fecha mais.

- **Exemplo #1:** Enquanto o número for menos que 6, vai imprimir o número (linha 3). Entretanto, na linha 4, incrementamos (somamos 1) o número de modo que, ele vai imprimir até 5 e para (critério de parada - fim).

```
1 numero = 1
2 while numero <6:
3     print(numero)
4     numero = numero+1
5 print("Terminou o loop")
```

1
2
3
4
5
Terminou o loop

- **Exemplo #2:** Este se refere a um dos usos comuns do WHILE, que é de apresentar menus de opções ao usuário. No exemplo a seguir, ele vai apresentar as opções 1) Saque 2) Extrato 3) Sair enquanto o usuário não digitar 3, que é Sair da aplicação.

```
1 print('Bem-vindo do Banco Abella')
2 command = '0'
3 while command != '3':
4     print('1) Saque 2) Extrato 3) Sair')
5     command = input('Digite sua opção ')
6     #funcionamento do programa
```



- Listas
- Tuplas
- Sets (Conjuntos)
- Dictionaries

Python

Lógica de Programação

Estrutura de Dados com Python

01 Introdução

- Estruturas de dados são basicamente um conjunto de dados armazenados na memória de modo a fazer sentido.
- Para que possamos entender melhor, imagine que, eu queira criar uma lista com todos os itens que vou comprar no supermercado. Para isto, o Python tem a estrutura de dados chamada de lista, que é apenas uma das estruturas existentes.

02 Básico sobre Listas

- Imagine que você queira criar uma lista de todos os itens de uma feira. A primeira vista, você criaria uma variável para cada uma dos itens, não é mesmo? Mas, para isso você tem as listas, que aprenderemos agora. No exemplo a seguir, temos uma lista de *strings*, contendo 3 elementos.

```
1 listaFeira = ['maca', 'banana', 'uva']
```

- A lista acima, é uma lista com 3 elementos, na qual a primeira posição temos uma maçã, na segunda uma banana e na terceira uma uva. Entretanto, **em Python, a primeira posição é zero**, de modo que, a maçã está na posição 0, banana na posição 1 e uva na posição 2. A analogia é apresentada na imagem a seguir.



- A lista acima, é uma lista com 3 elementos, na qual a primeira posição temos uma maçã, na segunda uma banana e na terceira uma uva. Entretanto, **em Python, a primeira posição é zero**, de modo que, a maçã está na posição 0, banana na posição 1 e uva na posição 2. A analogia é apresentada na imagem a seguir.

```
1 listaFeira = ['maca', 'banana', 'uva']
2
3 print(listaFeira[0])
4 print(listaFeira[1])
5 print(listaFeira[2])
6
7 primeiroItemLista = listaFeira[0]
8 print('o primeiro item
   é', primeiroItemLista)
```

Saída

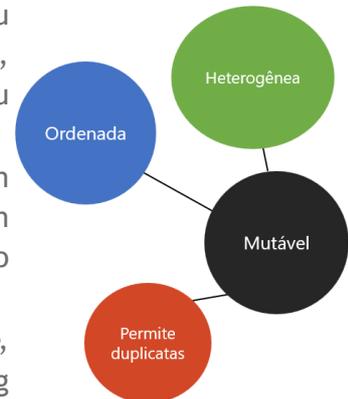
```
maca
banana
uva
o primeiro item é maca
```

- Podemos verificar no código anterior que, listaFeira[0] obtém o primeiro elemento da lista ('macã'), descrito em detalhes na imagem a seguir.
- Pode se tratar de uma lista, podemos criar uma variável para receber uma cópia dos elementos armazenados em alguma das posições da lista, algo como aconteceu na linha 7 do código anterior, na qual criamos a variável primeiroItemLista, que agora salva a 'macã', que está na primeira posição (posição 0).

```
listaFeira = ['maca', 'banana', 'uva']
               ↑         ↑         ↑
listaFeira[0] listaFeira[1] listaFeira[2]
```

03 Propriedades de Listas

- Os atributos que definem uma lista são apresentados a seguir.
- **Mutável** significa "mudável", ou seja, depois de criada a lista, podemos adicionar, remover ou modificar os elementos
- **Ordenada** significa que, cada item tem um índice (index) e seguem uma ordem (os novos são adicionados no final)
- **Permite duplicatas** significa que, se eu adicionar uma String 'Daniel', eu posso adicionar outra String 'Daniel' na mesma lista.
- **Heterogênea** significa que, uma lista pode ter ao mesmo tempo, elementos String, int e float, como o exemplo a seguir.
 - A primeira posição (0) é uma String ('maca')
 - A segunda posição (1) é um int
 - A terceira posição (1) é um float



```
1 lista = ['maca', 1, 1.0]
2
3 print(lista[0])
4 print(lista[1])
5 print(lista[2])
```

04 Criando Listas

- Anteriormente, nos antecipamos um pouco e criamos as nossas primeiras listas, não é mesmo? Agora, retomamos aos estágios preliminares de lista, que é a criação, que pode ser feita de duas formas, sendo elas:

- Usando o construtor chamado list()
- Utilizando colchetes [], como fizemos anteriormente

```
1 lista1 = []
2 lista2 = list()
3
4 lista3 = ['A', 'B']
5 lista4 = list( ('A', 'B') )
```

- No exemplo acima, criamos as lista1 e lista2 vazias, isto é, sem elementos. Complementarmente, as lista3 e lista4 possuem duas Strings ('A' na posição 0 e 'B' na posição 1).
 - Atenção ao modo em que a lista4 é criada, mais especificamente aos parênteses. Os parênteses indicados pela cor azul são relacionados ao construtor list(), enquanto que, os parênteses indicados pela cor vermelha estão associadas aos elementos, que por sua vez estão separados por vírgulas.

05 Acessando Elementos por Índice

- Os elementos da lista podem ser acessados por índice (em inglês, *index*). Como vimos anteriormente, os elementos em uma lista com Python, iniciam no índice 0 até tamanho - 1. Ou seja, em uma lista 3 elementos, vai de 0 a 2.

```
1 listaFeira = ['maca', 'banana', 'uva']
```

0	1	2
		

- Entretanto, o que não falamos é que, podemos ter índices negativos. Loucura, não?

```
1 lista1 = ['A', 'B', 'E', 'L', 'L', 'A']
2
3 print(lista1[1])
4 print(lista1[-5])
```

- No exemplo acima, nas linhas 3 e 4 será apresentado a String 'B'. O entendimento dos índices negativos é apresentado na imagem a seguir.

A	B	E	L	L	A
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

- Os elementos da lista podem ser acessados da direita para a esquerda usando índices negativos. O valor negativo começa de -1 a - tamanho da lista. Ou seja, indica que em Python podemos trabalhar de trás para frente.

06 Tamanho da Lista

- Em inglês, comprimento significa *length*. E vai ser com o método len() que vamos descobrir o tamanho de uma lista ou String.
- No exemplo a seguir, apresentamos como usar o método len.
 - Na linha 3 exibimos diretamente o tamanho
 - Nas linhas 5 e 6 também exibimos o tamanho, mas antes atribuímos o valor a uma variável

```
1 lista = ['A', 'B', 'E', 'L', 'L', 'A']
2
3 print(len(lista))
4
5 tamanhoLista = len(lista)
6 print(tamanhoLista)
```

- E agora, vamos mostrar o uso do método len() com Strings, que basicamente conta a quantidade de caracteres da String.

```
1 nome = 'Daniel'
2
3 print(len(nome))
4
5 tamanhoNome = len(nome)
6 print(tamanhoNome)
```

- Sim, mas quando vou usar esse método? No exemplo a seguir, usamos os métodos:

- **Sum()** que soma todos os elementos da lista
- **Len()** que retorna o tamanho (comprimento) da lista

```
1 notas = [10, 8, 9]
2
3 somaNotas = sum(notas)
4 media = somaNotas / len(notas)
5 print(media)
```

07 Imprimir Elementos da Lista

- Para apresentar os elementos de uma lista, chamamos comumente de iterar. Na sequência, apresentaremos duas formas de iterar uma lista.

```
1 lista = ['maçã', 'uva', 'banana']
2
3 ▼ for item in lista:
4     print(item)
```

```
1 lista = ['maçã', 'uva', 'banana']
2
3 ▼ for i in range(len(lista)):
4     print(lista[i])
```

- No exemplo à esquerda da página anterior, utilizamos o for sem o uso de índices, mas com uso de variável temporária (item). O funcionamento é o seguinte:

- Ao executar o for (da linha 3), a variável item recebe o valor que está na posição 0 da lista (isto é, 'maçã') e executa as linhas que possuem recuo (que estão atreladas ao for), isto é, a linha 4
- Ao terminar a execução das linhas recuadas (neste exemplo, apenas a linha 4), regressamos a linha 3 e a variável item recebe o valor que está na posição 1 da lista (isto é, 'uva') e executa as linhas que possuem recuo
- Ao terminar a execução das linhas recuadas, regressamos a linha 3 e a variável item recebe o valor que está na posição 2 da lista (isto é, 'banana') e executa as linhas que possuem recuo
- Como terminamos todos os elementos da lista, o for é encerrado e seguiremos a execução das linhas após o recuo (neste exemplo, linha 5 em diante)

- No exemplo à direita da página anterior, utilizamos índices. Notem que, o for é diferente do exemplo à esquerda, pois utilizamos o método `range()` associado ao método `len()`. A chamada `len(lista)` retorna o valor 3, de modo que, `range(3)` permite que o for execute de 0 até menor que 3 (isto é, 2). O funcionamento é o seguinte:

- Na primeira vez que o for é executado, a variável i recebe o valor 0 e na sequência ele exibe o trecho recuado (neste caso, linha 4) que é o `lista[i]`, que neste caso (que `i = 0`), exibiremos `lista[0]`, que 'maçã'. Quando terminar o trecho recuado, retornamos à linha 3
- Na segunda vez, exibimos o `lista[1]`
- Na terceira vez, exibimos o `lista[2]`
- Lembra que é até menor que 3 (no caso, 2)? Então neste caso, terminamos.

08 “Fatiar” uma lista

- Imagine que, você tenha uma lista com 100 elementos e precise apenas os dois primeiros elementos? Você precisa fatiar a lista. Fatiar em inglês significa *slice*.

```

1 listaFeira = ['maçã', 'uva', 'banana', 'pêra']
2
3 subLista = listaFeira[0:3]
4 print(subLista)
5
6 print(listaFeira[0:3])

```

- No exemplo acima, apresentamos duas maneiras de criar uma lista dos 2 primeiros elementos (`subLista`) da `listaFeira`.

- Neste exemplo, colocamos `listaFeira[0:3]`, que significa que a `subLista` é criada desde o índice 0 até menor que 3, ou seja, até 2

- Agora, a seguir, vamos apresentar outros exemplos que você pode se deparar.

- Na linha 2, são exibidos os 4 primeiros elementos
- Na linha 3, imprime o primeiro elemento (índice 0) e depois pula de 2 em 2
- Na linha 4, invertemos a lista, ou seja, o último vira primeiro e o primeiro vira último (e assim sucessivamente)
- Na linha 5, exibe a partir do elemento de índice 3 e imprime até o fim da lista

```

1 lista = [2, 4, 8, 10, 12, 14]
2 print(lista[:4])
3 print(lista[::2])
4 print(lista[::-1])
5 print(lista[3:])

```

09 Adicionar Elementos à Lista

- Para adicionar elementos à lista, temos 3 opções: `append()`, `insert()` e `extend()`.

- Vamos de `append()`? O método `append()` adiciona no fim da lista, como no exemplo a seguir, na qual na linha 2 adicionamos a String 'Nathaly' à lista.

```

1 lista = ['Daniel', 'Arthur']
2 lista.append('Nathaly')
3
4 ▼ for item in lista:
5     print(item)

```

- O método `insert()` possibilita a inserção de um novo elemento em uma dada posição da lista.

```

1 lista = ['Daniel', 'Arthur']
2 lista.insert(2, 'Nathaly')
3 lista.insert(0, 'Joselita')

```

- No exemplo acima, na linha 2, inserimos 'Nathaly' na posição 2, que não estava ocupada. Veja a ilustração a seguir.

- Antes da execução da linha 2:

Valores → 'Daniel' 'Arthur'

Índices → 0 1

- Depois da execução da linha 2:

Valores → 'Daniel' 'Arthur' 'Nathaly'

Índices → 0 1 2

```

1 lista = ['Daniel', 'Arthur']
2 lista.insert(2, 'Nathaly')
3 lista.insert(0, 'Joselita')

```

- Agora, ao executar a linha 3, em que vamos tentar adicionar 'Joselita' (minha sogra) temos uma curiosidade: esta posição está ocupada por 'Daniel', o que acontecerá? Veja a ilustração a seguir.

- Antes da execução da linha 3:

Valores →	'Daniel'	'Arthur'	'Nathaly'
Índices →	0	1	2

- Depois da execução da linha 3:

Valores →	'Joselita'	'Daniel'	'Arthur'	'Nathaly'
Índices →	0	1	2	3

- Note que, inserimos 'Joselita' exatamente na posição desejada (isto é, 0) e os itens à direita são deslocados uma posição para direita, ou seja, 'Daniel' que estava antes na posição 0, passa a assumir a posição 1 e assim sucessivamente.
- Por fim, vamos aprender sobre o método `extend()`, que adiciona uma lista de elementos ao fim da lista. Veja o exemplo a seguir.

```

1 lista = ['Daniel', 'Arthur']
2 lista.extend(['Nathaly'])

```

- No exemplo acima, adicionamos na linha 2, uma lista com um elemento (['Nathaly']) à lista anterior (com 2 elementos), de modo que, esta ficou agora com 3 elementos. Sim, mas se tiver mais de um elemento? Mesma coisa, veja no exemplo a seguir.

```

1 lista = ['Daniel', 'Arthur']
2 lista.extend(['Nathaly', 'Elisa'])

```

10 Tuplas (Tuples)

- As tuplas, seguem a mesma filosofia das listas, porém são imutáveis de modo que não podemos modificar os itens.
- Ou seja, quando uma tupla é criada, NÃO é possível adicionar, remover ou alterar os elementos.
- Vamos entender como funciona a sua criação, que pode ser feitas de três maneiras:
 - Na linha 1, criamos a tupla com uso de parênteses
 - Na linha 2, sem uso de parênteses, que são ocopnais
 - Na linha 3, usando o construtor `tuple()`.
 - Todas as 3 tuplas tem Strings, mas poderiam ter tipos variados

```

1 tupla1 = ('Daniel', 'Nathaly', 'Arthur')
2 tupla2 = 'Daniel', 'Nathaly', 'Arthur'
3 tupla3 = tuple(('Daniel', 'Nathaly', 'Arthur'))

```



Se você quiser criar uma tupla de um único elemento, você precisa colocar uma vírgula no fim. Por que? Veja o exemplo a seguir.

```

1 tupla = 'Daniel',
2 print(type(tupla))
3
4 nome = 'Daniel'
5 print(type(nome))

```

No exemplo acima, o que diferencia as linhas 1 e 4 é basicamente a vírgula, correto? Mas é isso que faz a variável da linha 1 ser uma tupla e a variável da linha 4 ser uma String (str).

- Ou seja, é bem parecido com listas. Inclusive, as operações a seguir são IGUAIS em listas e tuplas:
 - Tamanho com `len()`, Imprimir elementos, Acessar elementos por índice, Índices negativos e "Fatiar" uma lista. Então não vou repetir 😊
- Tuplas são usadas para adicionar diferentes tipos de dados com quantidade de elementos previamente definidos

11 Tuplas (Tuples) são imutáveis?

- Imutáveis significa que você não pode adicionar, remover ou alterar os valores. No exemplo a seguir, a execução da linha 1 funciona perfeitamente, pois uma lista podemos modificar os elementos.
 - Em contrapartida, a linha 2 não é possível, pois não é possível alterar o valor de uma elemento de uma tupla.

```

1 tupla1 = ('Daniel', 'Nathaly', 'Arthur')
2 tupla1[0] = 'Daniel Abella'
3
4 lista1 = ['Daniel', 'Nathaly', 'Arthur']
5 lista1[0] = 'Daniel Abella'

```



12 Exclusão de Tuplas e Listas

- Podemos basicamente excluir uma *tupla*, ou seja, apagar ela do programa, de modo que, após a sua exclusão, a variável não pode ser utilizada. Para isto, usamos a operação `del`, cujo exemplo é apresentado a seguir.

```

1 numbers=4,5,6
2 del numbers
3 print(numbers)
4

```

```

Traceback (most recent call last):
  File "main.py", line 3, in <module>
    print(numbers)
NameError: name 'numbers' is not defined

```

- Note que, na linha 2 realizamos a exclusão (em "americanizado" chamamos de deleção). E, na linha 3 em diante não podemos mais usar a variável `numbers`, que dá erro.

13 Exclusão de um Elemento da Lista

- Como vimos anteriormente, podemos excluir uma lista ou tupla INTEIRA. Entretanto, podemos excluir um elemento da lista ou um intervalo de elementos de uma lista. Este tipo de exclusão não funciona com tuplas, como veremos a seguir.

```
1 frutas = ['banana', 'uva', 'maçã']
2 del frutas[0]
3 print(frutas)
4
5 nomes = 'Daniel', 'Arthur', 'Nathaly'
6 del nomes[0]
7 print(nomes)
```



- No exemplo acima podemos verificar a exclusão de um elemento. Na linha 2, eliminamos o primeiro elemento (posição 0) de uma lista. Entretanto, a operação da linha 6 não é possível, vai dar erro, pois **tuplas são imutáveis**.
- Complementarmente, no exemplo abaixo, usamos a operação `del frutas[:2]`, na qual o dois pontos seguido do número 2 nos informa que, exclua da primeira posição, até a posição 2 (ou seja, não inclui a posição 2). Desta maneira, o resultado da linha 3 será `['maçã', 'melancia', 'kiwi']`

```
1 frutas = ['banana', 'uva', 'maçã', 'melancia', 'kiwi']
2 del frutas[:2] #exclui ATÉ a posição 2
3 print(frutas)
```

14 Descobrimo Índice de um Elemento

- Em qual posição de uma lista ou tupla o meu elemento está? Usaremos o método `index`, cujo exemplo está a seguir. Note que, em ambos os casos, nas linhas 2 e 6, buscamos identificar em qual índice está o elemento 20.

```
1 lista1 = [10, 20, 30, 40, 50]
2 posicao = lista1.index(20)
3 print(posicao)
4
5 tupla1 = (10, 20, 30, 40, 50)
6 posicao = tupla1.index(20)
7 print(posicao)
```

1
1
Saída

- Acima é realiza uma busca completa, concordas? Eu posso delimitar o intervalo de busca usando o método `index`, mas da seguinte forma:
 - `index(item, indiceInicioBusca, indiceFimBusca)`
 - No exemplo abaixo, iniciamos uma busca pelo elemento **60**, iniciando pelo índice **2** até o índice 4 (isto é, até **5**).

```
1 tuple1 = (20, 30, 40, 50, 60)
2 posicao = tuple1.index(60, 2, 5)
3 print(posicao)
```



Se você fizer uma busca usando `index` por algum elemento que não existe, uma exceção será lançada. Abaixo vemos um exemplo do problema e a respectiva *exception* que é lançada.

```
1 tuple1 = (20, 30, 40, 50, 60)
2 posicao = tuple1.index(61, 2, 5)
3 print(posicao)
```

ValueError: tuple.index(x): x not in tuple

Não sabe o que é *exception*, né? Mais a frente falamos, mas caso esteja com pressa, *Google It!*

- Ou seja, é bem parecido com listas. Inclusive, **as operações a seguir são IGUAIS em listas e tuplas**:
 - Tamanho com `len()`, Imprimir elementos, Acessar elementos por índice, Índices negativos e “Fatiar” uma lista. Então não vou repetir 😊
- Tuplas são usadas para adicionar diferentes tipos de dados com quantidade de elementos previamente definidos

15 Como saber se um element está na lista ou tupla?

- Usamos basicamente o operador `in`, apresentado no exemplo a seguir, que pode ser usado em tuplas ou listas. Note que, podemos perguntar diretamente se um elemento está na lista/tuple, como nas linhas 3 e 4, bem como podemos colocar a condição em um `if`, a exemplo das linhas 6 e 11.

```
1 tuple1 = (10, 20, 30, 40, 50)
2
3 print(20 in tuple1)
4 print(21 in tuple1)
5
6 ▼ if 20 in tuple1:
7     print('esta na tupla')
8
9 variavel = 20
10
11 ▼ if variavel in tuple1:
12     print('esta na tupla')
```

- Como funciona com listas? Só mudar na linha 1 de parênteses (característica de uma dupla) para colchetes (característica de uma lista). Vai funcionar sem, mas eu ainda faria uma pequena modificaçãozinha: mudaria o nome da variável de `tuple1` para `list1` para ficar redondo! Apesar que vai exigir mudança no código inteiro.

```
1 list1 = [10, 20, 30, 40, 50]
```

16 Gambiarra: Adicionar Elementos em uma Tupla

- Como dissemos anteriormente, não podemos mudar uma tupla, pois ela é imutável. Mas, adoramos uma gambiarra, digo, workaround (sua tradução em inglês), que deixa muito + elegante.



- No workaround a seguir, para adicionar em uma tupla, a convertemos em uma lista, fazemos nossas mudanças (adicionar, remover, alterar um elemento) e por fim, transformamos de volta a lista em uma tupla. Liiiiindo, não? PN.

```
1 tuple1 = (0, 1, 2, 3, 4, 5)
2 listaGambiarra = list(tuple1)
3 listaGambiarra.append(6)
4 tuple1 = tuple(listaGambiarra)
5 print(tuple1)
```

17 Métodos Filé de Listas

- Vamos aprender métodos bem úteis que podem ser aplicados a listas, sendo eles: `append()`, `insert()`, `remove()`, `pop()`, `clear()`, `sort()` e `reverse()`. São métodos *built-in*.
- A seguir temos um exemplo completo. Experimente por um `print` entre cada uma das linhas para entender melhor.
 - Na linha 2, adicionar um elemento ao fim da lista
 - Na linha 3, inserimos o valor 10 na posição 1
 - Na linha 4, removemos o elemento 10
 - Na linha 5, removemos o elemento da posição 2
 - Na linha 7, ordenamos os elementos de ordem crescente (do menor ao maior)
 - Na linha 9, usamos o mesmo método, porém ordenamos de maneira decrescente (do maior ao menor)

```
1 lista1 = [20, 30, 40]
2 lista1.append(50) #insere no final
3 lista1.insert(1,10) #insere 10 na posição 1
4 lista1.remove(10) #remove o elemento 10
5 lista1.pop(2) #remove o elemento da posição 2
6 lista1.insert(0,70) #insere 70 na posição 0
7 lista1.sort() #ordena de modo crescente
8 #vai ficar [20,30,50,70]
9 lista1.sort(reverse=True) #ordena de modo decrescente
10 #vai ficar [70,50,30,20]
11 print(len(lista1))
```

18 Métodos Filé de Listas e Tuplas

- Existem mais métodos bacanas que podem ser usados em listas e tuplas, sendo eles `min`, `max` e `sum`.

```
1 lista1 = [20, 30, 40]
2
3 menorElemento = min(lista1)
4 maiorElemento = max(lista1)
5 somaDosElementos = sum(lista1)
6 mediaDosElementos = sum(lista1) / len(lista1)
7
8 tupla1 = (20, 30, 40)
9
10 menorElemento = min(tupla1)
11 maiorElemento = max(tupla1)
12 somaDosElementos = sum(tupla1)
13 mediaDosElementos = sum(tupla1) / len(tupla1)
```

- No exemplo acima, usamos o método `min` para obter o menor elemento de uma lista/tupla, `max` para obter o maior elemento de uma lista/tupla. Por fim, usamos `sum` para obter a soma de todos os elementos de uma lista.

19 Básico de Conjuntos

- Conjuntos (em Inglês, *Set*) é uma estrutura de dados que não é ordenada e não possui duplicadas, ou seja, se eu adicionei o elemento 'Abella', não pode ter outro elemento com o mesmo valor. Além disso, é heterogêneo, ou seja, a
 - Possivelmente no ensino médio tenhamos discutido conjuntos, mas hoje vamos entender na prática o funcionamento.
- Para criar um conjuntos, temos duas maneiras. A primeira deles é usando o método `set()`.

```
1 livrosAbella = set(('Gestão A3', 'Scrum Arretado'))
2 print(type(livrosAbella))
```

- A segunda é usar chaves, na qual os valores são separados por vírgula.

```
1 livrosAbella = {'Gestão A3', 'Scrum Arretado'}
2 print(type(livrosAbella))
```

- No exemplo a seguir, verificamos que o conjunto é bem parecido com as outras estruturas que vimos. Atenção ao método `add` que adiciona um elemento e ao `update` que adiciona mais de um elemento ao mesmo tempo.

```
1 livrosAbella = {'Gestão A3', 'Scrum Arretado'}
2
3 ▼ for livro in livrosAbella:
4     print(livro)
5
6 ▼ if 'Gestão A3' in livrosAbella:
7     print('existe')
8
9 ▼ if not 'Use a Cabeça PMP' in livrosAbella:
10    print('não está na lista')
11
12 print('Gestão A3' in livrosAbella)
13
14 livrosAbella.add('Novo Livro')
15 livrosAbella.update(['Novo Livro 2', 'Novo Livro 3'])
16
17 print(len(livrosAbella))
```

20 Remoção em Conjuntos

- Existem 3 métodos principais usados para remoção.
- O primeiro deles é o `remove`, que remove o elemento informado como parâmetro e caso este elemento não exista no conjunto, lança uma exceção.

- No exemplo a seguir, a linha 4 lança uma `exception`, pois o 'Livro3' foi excluído na linha 3

```
1 livrosAbella = {'Gestão A3',
2               'Scrum Arretado', 'Livro3'}
3 livrosAbella.remove('Livro3')
4 livrosAbella.remove('Livro3')
```

- O segundo deles, o `discard`, possui a mesma função do `remove`, porém não lança uma exceção.

- No exemplo a seguir, a linha 4 não lança nenhuma `exception`, diferente do exemplo anterior.

```
1 livrosAbella = {'Gestão A3',
2               'Scrum Arretado', 'Livro3'}
3 livrosAbella.discard('Livro3')
4 livrosAbella.discard('Livro3')
```

- O terceiro e último método é o `pop`, que exclui um elemento de maneira aleatória.

- Ou seja, se você não sabe quem vai ser excluído, você não informa nada no parâmetro.

```
1 livrosAbella = {'Gestão A3',
2               'Scrum Arretado', 'Livro3'}
3 livrosAbella.pop()
```

- E, se eu quiser limpar um conjunto? Ou seja, apagar todos os elementos existentes “de uma lapada só”. Usamos o `clear`.

- No exemplo a seguir, a linha 3 imprimirá 0

```
1 livrosAbella = {'Gestão A3', 'Scrum Arretado'}
2 livrosAbella.clear()
3 print(len(livrosAbella))
```

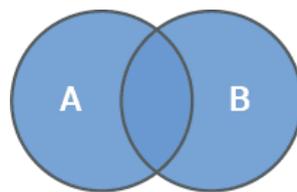
- E, se eu quiser excluir um conjunto, ou seja, torna-lo inutilizável, basta usar o comando `del`, como no exemplo a seguir.

- A linha 3 causará um erro, pois esta foi inutilizada na linha 2

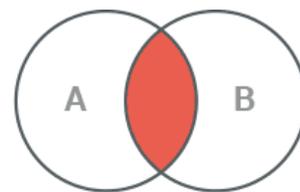
```
1 livrosAbella = {'Gestão A3', 'Scrum Arretado'}
2 del livrosAbella
3 print(len(livrosAbella))
```

21 Operações em Conjuntos

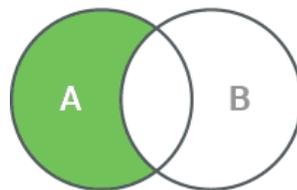
- Vamos voltar ao ensino médio e entender as operações em conjuntos, sendo elas: união (`()`), interseção (`&`), diferença (`-`) e diferença simétrica (`^`).



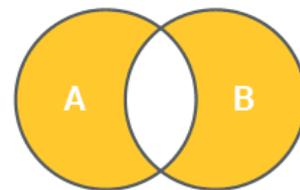
Union



Intersection



Difference



Symmetric Difference

Fonte: learnbyexample.org/python-set/

- Acima, relacionamos as principais operações em conjuntos.

União (Union) de Conjuntos

- A primeira operação que vamos aprender com conjuntos é a União. Relembrando com nostalgia o ensino médio, a união retoma todos os itens dos 2 conjuntos relacionados. Como conjunto, se um elemento estiver presente nos 2 conjuntos relacionados, apenas 1 é incluído no conjunto resultante.

- Existem 2 formas de fazer união com conjuntos. A primeira delas é usando o método `union()` e a segunda é com o operador `|`. O exemplo de uso é apresentado a seguir, cujo resultado é `{'banana', 'maçã', 'uva'}`

```
1 conjuntoA = {'banana', 'maçã'}
2 conjuntoB = {'uva', 'banana'}
3
4 print(conjuntoA | conjuntoB)
5 print(conjuntoA.union(conjuntoB))
```

Interseção (Intersection) de Conjuntos

- A interseção retoma os elementos contidos em ambos os conjuntos. Para isto, usa o método `intersection()` ou o operador `&`, como podemos ver no exemplo a seguir, que retorna `{'banana'}`.

```
1 conjuntoA = {'banana', 'maçã'}
2 conjuntoB = {'uva', 'banana'}
3
4 print(conjuntoA & conjuntoB)
5 print(conjuntoA.intersection(conjuntoB))
```

Diferença (Difference) de Conjuntos

- A diferença retornará os itens que estão apenas no primeiro conjunto (`conjuntoA`). Para isto, usamos o operador `-` ou o método `difference()`. No exemplo a seguir, a saída é `{'maçã'}`.

```

1 conjuntoA = {'banana', 'maçã'}
2 conjuntoB = {'uva', 'banana'}
3
4 print(conjuntoA - conjuntoB)
5 print(conjuntoA.difference(conjuntoB))

```

Diferença Simétrica (*Symmetric Difference*) de Conjuntos

- A diferença simétrica retornará os itens que estão em **ambos** conjuntos, mas desconsiderando os que estiverem nos 2 conjuntos. Para isto, usamos o operador `^` ou o método `symmetric_difference()`. No exemplo a seguir, a saída é `{'maçã', 'uva'}`.

```

1 conjuntoA = {'banana', 'maçã'}
2 conjuntoB = {'uva', 'banana'}
3
4 print(conjuntoA ^ conjuntoB)
5 print(conjuntoA.symmetric_difference(conjuntoB))

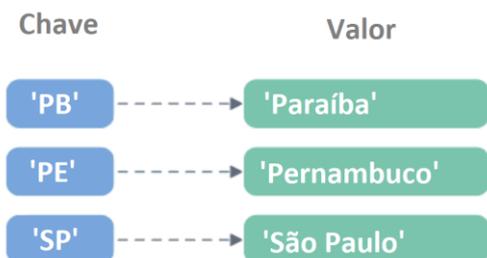
```

Resumo das Operações com Conjuntos

Operação	Método	Operador
União	<code>union()</code>	<code> </code>
Inserseção	<code>Intersection()</code>	<code>&</code>
Diferença	<code>difference()</code>	<code>-</code>
Diferença Simétrica	<code>symmetric_difference()</code>	<code>^</code>

22 Dicionários

- Também conhecido por *Hash Maps*, Dicionários é basicamente um mapeamento entre conjunto de índices (também conhecidos por chaves) a um conjunto de valores, tal como é apresentado na imagem a seguir. Note que, para cada chave (por exemplo, 'PB') é associada um valor (por exemplo, 'Paraíba').



- A associação de uma chave a um valor é conhecido por **chave:valor** ou **item**. O dicionário apresentado na imagem anterior, caso fosse em código Python, seria o seguinte.

- Ou seja, usamos chaves `{}` para criar dicionários

```

1 estados = {
2     'PB': 'Paraíba',
3     'PE': 'Pernambuco',
4     'SP': 'São Paulo',
5 }

```

- Alternativamente podemos criar dicionários com uso do método `dict()`.

```

1 estados = dict({
2     'PB': 'Paraíba',
3     'PE': 'Pernambuco',
4     'SP': 'São Paulo'})

```

- Uma coisa muito importante é que as chaves devem ser únicas, de modo que, se você colocar 2 chaves iguais, a segunda sobrescreverá a primeira, tal como no exemplo a seguir.

```

1 estados = dict({
2     'PB': 'Paraíba',
3     'PE': 'Pernambuco',
4     'PB': 'Parahyba'})
5 print(estados)

```

```
{'PB': 'Parahyba', 'PE': 'Pernambuco'}
```

Adição de Elementos em Dicionários

- No exemplo a seguir, adicionamos o elemento "Paraíba" com chave "PB". Além disso, adicionamos à chave "SP" uma tupla ("São Paulo", "Sampa").

```

1 estados = {} #Dicionário vazio
2 estados['PB'] = 'Paraíba' #Adiciona chave PB e
  valor Paraíba
3 estados['SP'] = 'São Paulo', 'Sampa' #Adiciona
  chave SP e valor que é uma tupla com {'PB':
  'Parahyba', 'SP': ('São Paulo', 'Sampa')}

```

Listagem em um Dicionário

- No exemplo a seguir, apresentamos a chave e o valor associado à chave.

```

1 estados = { 'pb': 'paraíba', 'sp': 'sampa' }
2
3 for i in estados:
4     print(i)
5     print(estados[i])

```

Obter o Elemento de uma Chave

- Se eu quiser saber qual o valor da chave "PB"? Mostramos a seguir duas formas. A primeira delas usando o colchete com o valor da chave (linha 4) e a segunda usando o método `get()` (linha 5).

```

1 estados = {}
2 estados['PB'] = 'Parahyba'
3
4 print(estados['PB']) #Obtém valor da chave 'PB'
5 print(estados.get('PB')) #Obtém valor da chave 'PB'

```

Exclusão de Elementos em Dicionários

- Existem 3 métodos associados à exclusão de elementos, sendo eles `del`, `pop` e `clear`.

Exclusão de Elementos em Dicionários

- O método `popitem` remove um item (chave/valor) de maneira aleatória e retorna como tupla
- O método `pop`, recebe uma chave, e exclui e retorna o item (chave/valor) associado à chave informada.
- O método `clear()` faz um “limpa”, ou seja, apaga todos os itens

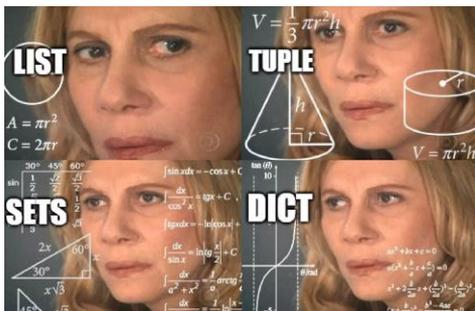
```

1 estados = {}
2 estados['PB'] = 'Paraíba'
3 estados['PE'] = 'Pernambuco'
4 estados['SP'] = 'São Paulo', 'Sampa'
5
6 del estados['PB'] #Apaga o item de chave 'PB'
7
8 elementoEliminado = estados.popitem()
9 print(elementoEliminado)
10
11 elementoEliminado = estados.pop('PE')
12 print(elementoEliminado)
13
14 estados.clear() #limpa o dicionário
15 print(estados)

```

23 Resumo

- Ficou assim, né? Vamos resumir.



- A seguir apresentamos um resumo de cada estrutura de dados.

Estrutura	Ordenado?	Construtor	Exemplo
list	Sim	[] ou list()	['A', 'E']
tuple	Sim	() ou tuple()	('A', 'E')
set	Não	{ } ou set()	{ 'A', 'E' }
dictionary	Sim	{ } ou dict()	{ 'PB': 20, 'PE': 22 }

Resumo de Listas

```

listaFeira = ['maca', 'banana', 'uva']
              ↑       ↑       ↑
listaFeira[0] listaFeira[1] listaFeira[2]

```

Método	Descrição	Exemplo
<code>append(elemento)</code>	Adiciona ao fim da lista	<code>lista.append(12)</code>
<code>clear()</code>	Remove todos elementos	<code>lista.clear()</code>
<code>count(x)</code>	Quantas 'x' tem na lista?	<code>lista.count('x')</code>
<code>index(x)</code>	Posição de X na lista?	<code>lista.index('x')</code>
<code>insert(i, elemento)</code>	Add 'X' na posição i	<code>lista.insert(0, 'X')</code>
<code>pop()</code>	Remove o último elemento	<code>lista.pop()</code>
<code>remove(x)</code>	Remove o elemento 'x'	<code>lista.remove('x')</code>
<code>reverse()</code>	Inverte os elementos	<code>lista.reverse()</code>
<code>sort()</code>	Ordena de modo crescente	<code>lista.sort()</code>
<code>sort()</code>	Ordena de modo decrescente	<code>lista.sort(reverse=True)</code>

Resumo de Tuplas

```

1 tupla1 = ('Daniel', 'Nathaly', 'Arthur')
2 tupla2 = 'Daniel', 'Nathaly', 'Arthur'
3 tupla3 = tuple(('Daniel', 'Nathaly', 'Arthur'))

```

Método	Descrição	Exemplo
<code>del tupla</code>	Exclui a tupla	<code>del tupla</code>
<code>del tupla[0]</code>	Remove um elemento da tupla	<code>del tupla[0]</code>
<code>count(x)</code>	Quantas 'x' tem na tupla?	<code>tupla.count('x')</code>
<code>index(x)</code>	Posição de X na tupla?	<code>tupla.index('x')</code>
<code>in</code>	Se um elemento tá na tupla	<code>If 20 in tupla</code>
<code>max</code>	Maior valor da tupla	<code>max(tupla)</code>
<code>min</code>	Menor valor da tupla	<code>min(tupla)</code>
<code>len</code>	Tamanho da tupla	<code>len(tupla)</code>

Resumo de Conjuntos

```

1 livrosAbella = set(('Gestão A3', 'Scrum Arretado'))
2 print(type(livrosAbella))

1 livrosAbella = {'Gestão A3', 'Scrum Arretado'}
2 print(type(livrosAbella))

```

Método	Descrição	Exemplo
<code>add</code>	Adiciona um elemento	<code>livros.add('A')</code>
<code>update</code>	Adiciona mais de 1 elemento	<code>livros.update('A', 'B')</code>
<code>count(x)</code>	Quantas 'x' tem no set?	<code>livros.count('x')</code>
<code>index(x)</code>	Posição de X no set?	<code>livros.index('x')</code>
<code>in</code>	Se um elemento tá no set	<code>If 20 in livros</code>
<code>max</code>	Maior valor do set	<code>max(livros)</code>
<code>min</code>	Menor valor do set	<code>min(livros)</code>
<code>len</code>	Tamanho do set	<code>len(livros)</code>
<code>remove</code>	Remover do conjunto (set)	<code>livros.remove('A')</code>
<code>discard</code>	Igual remove, mas não lança exceção	<code>livros.discard('A')</code>
<code>pop</code>	Exclui um elemento de modo aleatório	<code>livros.pop()</code>

Resumo das Operações com Conjuntos

Operação	Método	Operador
União	<code>union()</code>	
Inserseção	<code>Intersection()</code>	&
Diferença	<code>difference()</code>	-
Diferença Simétrica	<code>symmetric_difference()</code>	^

Resumo de Dicionários

Método	Descrição	Exemplo
<code>clear()</code>	Limpa o dicionário	<code>dict.clear()</code>
<code>get()</code>	Obtém o valor de uma chave	<code>dict.get(chave))</code>
<code>items()</code>	Retorna a listas contendo uma tupla para cada chave/valor	<code>print(dict.items())</code> #Resposta: (('Física', 88), ('Matemática', 65))
<code>keys()</code>	Retorna a lista de chaves	<code>dict.keys()</code> #Resultado ['name', 'salary', 'age']
<code>pop()</code>	Remove um elemento de uma chave	<code>removido = dict.pop('Chave')</code>
<code>popitem()</code>	Remove o último item (chave/valor)	<code>dict.popitem()</code>
<code>setdefault()</code>	Retorna o valor de uma chave.	<code>valor = dict.setdefault(chave)</code>
<code>update()</code>	Atualiza um item	<code>dict.update({'Chave': 48})</code>
<code>values()</code>	Retorna a lista de todos os valores	<code>marks.values() # Resultado: dict_values([67, 87])</code>



- Conceitos de Métodos
- Criação de Métodos
- Exemplo de IMC
- Ordem dos Parâmetros

Python

Lógica de Programação

Funções em Python

01 Conceito de Métodos

- Qual foi o nosso primeiro código, lembram? Foi o tradicional *Hello World* (Olá Mundo). O print é um método que o Python tem para apresentar dados ao usuário final. E, neste resumo aprenderemos a usar métodos, também conhecidos por funções.

```
main.py x
1 print('Olá mundo!')
```

- Imagine-se desenvolvendo uma aplicação para ser utilizado em academias. E, em diversos locais do aplicativo, realizamos o cálculo do IMC.
- Se, hipoteticamente, o cálculo do IMC mude, o que vai acontecer? Você vai ter que modificar a forma de calcular o IMC em todos os locais.

- E se você esquecer de alterar em algum dos locais? Zebrou, concorda? 🙌

```
1 altura = float(input("Digite sua altura (em metros): "))
2 peso = float(input("Digite seu peso (em kg): "))
3 imc = peso / altura**2
```

- Para resolver isto, precisamos usar métodos, como veremos na seção a seguir.

02 Criação de Métodos

- Para criar as nossas primeiras funções, devemos usar a palavra-chave def, conforme o exemplo a seguir.

```
1 def minhaFuncao():
2     print('Imprime algo')
3     x = 1
4     print(x)
5
6 print('Não tem nada a ver com a função')
7 minhaFuncao()
```

- No exemplo acima, criamos uma função com nome chamado `minhaFuncao`, seguido de `()`. Esta função possui 3 linhas, que são as linhas 2,3 e 4, devido ao recuo. A linha 6 não tem nada a ver com a função. E, na linha 7, chamamos essa função.

- Uma vez criada a função, podemos chamar quantas vezes quiser a função, como fizemos na linha 7.

```
1 def minhaFuncao(parametro1):
2     print(parametro1)
3
4 print('Não tem nada a ver com a função')
5 minhaFuncao('valor1')
```

Não tem nada a ver com a função
valor1 Saída

- No exemplo acima, criamos uma função diferente, pois usamos **parâmetros**. Parâmetros são informações que, quem for chamar (linha 5) passam informações para o método.
 - Na linha 5, passamos para o método `minhaFuncao` a String 'valor1'
 - Na linha 1, entre os parênteses `()` temos o parâmetro chamado `parametro1`, que receberá o 'valor1'. Este valor recebido é apresentado na linha 2.
- Como faremos para criar um método para cálculo do IMC? Vamos a primeira versão, que não é tão boa, mas funciona. E baseado na primeira versão, mostrei a segunda versão.

```
1 def calculoImc():
2     altura = float(input("Digite sua altura (em metros): "))
3     peso = float(input("Digite seu peso (em kg): "))
4     imc = peso / altura**2
5     print(imc)
6
7 calculoImc()
```

- Esse método funciona, mas tem alguns probleminhas.

- #1 É bom fazer leitura (input) das informações para cálculo do IMC fora do método e, dentro do método ter somente o cálculo do IMC
- #2 É bom passar as informações como parâmetro, como fizemos anteriormente.
- #3 É bom retornar o valor ao invés de imprimir com print, dentro do método
- #4 Quero você deixar doidão por Python 😊



03 Cálculo do IMC (Versão 1)

- Abaixo temos a primeira versão. Note que, com relação a:
 - #1, as variáveis são lidas fora do método (linhas 5 e 6).
 - #2 Passamos a altura e peso (lidos nas linhas 5 e 6) como parâmetros (linha 7). Note que, o método recebe os valores na linha 1.

```
1 def calculoImc(alt, pes):
2     imc = pes / alt**2
3     print(imc)
4
5 altura = float(input("Digite sua altura (em metros): "))
6 peso = float(input("Digite seu peso (em kg): "))
7 calculoImc(altura, peso)
```

- Note que, o item #3 ficamos devendo, mas vamos desvendar na seção a seguir.

04 Cálculo do IMC (Versão 2)

- Faltou retornar, mas o que é retornar? Vimos que no código acima fizemos um print dentro do método? Pois bem, isto não é uma boa prática. O ideal é, você devolver (retornar) a quem chamar o valor que ele precisa (neste caso, o valor do IMC) e quem chamou, se quiser, imprime.
- Um bom exemplo de um método que retorna algo é o input, que retorna uma String.
 - No exemplo a seguir, chamamos input, que devolve uma String. E, pegamos esse retorno (devolução) e atribuímos a uma variável (nome).

```
1 nome = input("Digite nome")
```

- E um exemplo de métodos que não tem retorno? Print 😊

```
main.py x
1 print('Olá mundo!')
```

- Para retornar, usamos a palavra-chave return, como veremos no exemplo a seguir. Note que:

- Tiramos o print de dentro do método;
- O método passou a retornar o valor 3
- Ao chamar o método (linha 7), usamos a variável chamada retorno para “pegar” o retorno do método

```
1 def calculoImc(alt, pes):
2     imc = pes / alt**2
3     return imc
4
5 altura = float(input("Digite sua altura (em metros): "))
6 peso = float(input("Digite seu peso (em kg): "))
7 retorno = calculoImc(altura, peso)
8 print(retorno)
```

- Para deixar claro, no exemplo anterior, chamamos o método usando valores lidos do usuário. Mas, podemos chamar os métodos diretamente, passando os valores “na mão”, como podemos verificar no exemplo a seguir.

```
1 def calculoImc(alt, pes):
2     imc = pes / alt**2
3     return imc
4
5 retorno = calculoImc(1.8,85)
6 print(retorno)
```

Possíveis erros que você pode encontrar:

- Chamar método sem parâmetros, quando o método tem parâmetros
- Chamar método passando 1 parâmetro, quando o método “pede” 2 parâmetros
- Em resumo, se o método tem N parâmetros, você tem que passar N parâmetros ao chamar.

Exemplos a seguir:



```
1 def calculoImc(alt, pes):
2     imc = pes / alt**2
3     return imc
4
5 retorno = calculoImc(85)
```

```
1 def calculoImc(alt, pes):
2     imc = pes / alt**2
3     return imc
4
5 retorno = calculoImc()
```

05 Ordem dos Parâmetros

- Note que, abaixo temos um método comum com 2 parâmetros. Entretanto, chamamos o método de 2 maneiras:
 - A primeira forma (linha 4) é como usualmente realizamos, onde os argumentos são posicionados. Neste cenário, passamos 178 e 88, que implica que, parametro1 recebe 178 e parametro2 recebe 88.
 - Neste cenário, a ordem dos argumentos é essencial
 - A segunda forma (linha 5) passamos o nome do parâmetro seguindo de um ‘=’ e o valor. Desta maneira, a ordem não importa, de forma que, no exemplo passamos parametro2 antes do parametro1, tal como acontece no método.

```
1 def metodoTeste(parametro1, parametro2):
2     print(parametro1, parametro2)
3
4 metodoTeste(178, 88)
5 metodoTeste(parametro2=88, parametro1=178)
```



- Definição de *Strings*
- Principais Métodos e Operações
- Exemplos de Uso
- Formatar *String* com *f-string*

Python

Lógica de Programação

Strings

01 Definição de Strings

- Strings são basicamente uma lista/array de caracteres em Unicode.
 - Por exemplo a String 'Daniel', apresentada na imagem a seguir, é uma lista os 6 caracteres do nome
 - E, como uma lista, o primeiro índice começa sempre com zero (0) e termina com o tamanho - 1 (neste caso, 6-1 = 5).
 - Complementarmente, veja que, embora incomum, existe a possibilidade de usar índices negativos.

Índices Positivos

0	1	2	3	4	5
---	---	---	---	---	---

D	A	N	I	E	L
---	---	---	---	---	---

-6	-5	-4	-3	-2	-1
----	----	----	----	----	----

Índices Negativos

- Vamos para o nosso primeiro exemplo a seguir.
 - Na linha 1, declaramos a variável Daniel, que pode estar entre aspas simples ' ou duplas ", desde que seja consistente
 - Na linha 3, apresentamos a variável
 - Na linha 4, apresentamos o primeiro caractere da String ('D'), que está na posição 0
 - Opcionalmente, como vimos anteriormente, podemos usar índices negativos, a exemplo da linha 5.

```

1 nome = 'Daniel'
2
3 print(nome)
4 print(nome[0])
5 print(nome[-6])
    
```

02 Principais Métodos e Operações

- A seguir, relacionamos os principais métodos utilizados em String.

Operação	Descrição	Exemplo
len(nome)	Tamanho da String	len(nome)
for	Lista caracteres de uma String	nome = 'Daniel Abella' for letra in nome: print(letra)
s2 in s1	Verifica se substring s2 é parte da String s1	nome = 'Daniel Abella' print('Da' in nome) if 'Da' in nome: print('meu nome tem da')
nome[i]	Obtém caractere de uma data posição	nome[0]
nome[i:j]	Obtém caracteres entre as posições	nome[0:2]
nome.lower()	Converte para minúsculo	nome.lower()
nome.upper()	Converte para maiúsculo	nome.upper()
nome.count('a')	Verifica quantas vezes 'A' aparece em nome	nome.count('a')
nome.isnumeric()	Verifica se a String é composta por números	nome.isnumeric()
nome.isalpha()	Verifica se a String é composta por letras	nome.isalpha()
nome.split()	Quebra a String por espaço	nome = 'Daniel Abella' nome.split(' ') #Retornará 2 strings (uma 'Daniel' e outra 'Abella')
nome.split('-')	Quebra a String em várias utilizando '-' como delimitador	nome = 'Daniel-Abella' nome.split('-') #Retornará 2 strings (uma 'Daniel' e outra 'Abella')
nome.replace(antigo, novo)	Substitui um texto por outro	nome = 'Daniel' nome.replace('da', 'vo') #Retornará 'voniel'
nome.join(sobre nome)	Retorna uma string com o nome e sobrenome concatenados	nome.join('Abella') #Retornará 'DanielAbella'
nome.startswith('Da')	Verifica se o nome inicia com 'Da'	if nome.startswith('Da'): print('inicia com 'Da')
s1 + s2	Junta as 2 strings	s1 + s2
s1 * 5	Repete a s1 por 5 vezes	print(s1 * 5)
capitalize()	Primeira letra maiúscula e o resto minúscula	nome.capitalize()
find('ni')	Verifica em que posição tá 'ni'	nome.find('ni')

03 Exemplos de Uso

- A seguir, apresentaremos exemplos de cada um dos métodos ou operações com códigos objetivos. Experimente fazer você mesmo 😊

For e Len

```
1 nome = 'Daniel'
2 print(len(nome)) #Saída: 13
3 ▼ for letra in nome:
4     print(letra) #Letra a letra
```

Operador in

```
1 nome = 'Daniel'
2 print('Da' in nome) #Saída: true
3 ▼ if 'Da' in nome:
4     print('Da está dentro do nome')
```

Obter substring (pedaço da String)

```
1 nome = 'Daniel Abella'
2 primeiraLetra = nome[0] #Saída 'D'
3 primeiroNome = nome[0:6] #Saída 'Daniel'
4 print(primeiraLetra, primeiroNome)
```

Converte para Maiúsculo e Minúsculo

```
1 nome = 'Daniel'
2 nomeMaiusculo = nome.upper()
3 nomeMinusculo = nome.lower()
4 print(nomeMaiusculo) #Saída: DANIEL
5 print(nomeMinusculo) #Saída: daniel
```

É maiusculo? É minúsculo?

```
1 nome = 'DANIEL'
2
3 ▼ if nome.isupper():
4     print('maiusculo')
5 ▼ elif nome.islower():
6     print('minusculo')
```

Quantos 'a' tem? A string é número ou letra?

```
1 nome = '1'
2 ▼ if nome.isnumeric():
3     print('é número') #Entra aqui
4 ▼ elif nome.isalpha():
5     print('é letra')
6
7 nomeCompleto = 'Daniel Abella'
8 quantidadeA = nomeCompleto.count('a')
9 print(quantidadeA) #Saída: 2
```

Juntar Strings com join e +

```
1 nome = 'Daniel'
2 sobrenome = 'Abella'
3 sobrenomes = ['Abella', 'Souza']
4
5 nomeCompleto1 = nome + sobrenome
6 nomeCompleto2 = nome.join(sobrenomes)
7
8 print(nomeCompleto1) #Saída DanielAbella
9 print(nomeCompleto2) #Saída AbellaDanielSouza
```

Fatiar Strings com Split

```
1 nome1 = 'Daniel Abella'
2 nome2 = 'Daniel-Abella'
3
4 nomesFatiadosEspaco = nome1.split()
5 nomesFatiadosHifen = nome2.split('-')
6
7 ▼ for nome in nomesFatiadosEspaco:
8     print(nome)
9
10 ▼ for nome in nomesFatiadosHifen:
11     print(nome)
```

Saída
Daniel
Abella
Daniel
Abella

Substituir Caractere com Replace

```
1 nome = 'Daniel'
2 nomeModificado = nome.replace('Da', 'vo')
3 print(nomeModificado) #Retornará 'voniel'
```

Começa com 'Da'

```
1 nome = 'Daniel'
2
3 print(nome.startswith('Da'))
4 ▼ if nome.startswith('Da'):
5     print('Nome inicia com Da')
```

Em que posição tá a String 'ni'

```
1 nome = 'Daniel'
2 print(nome.find('ni')) #saída: 2
```

04 Formatar String com f-string

- Existem várias maneiras de formatar as Strings com Python
- Antes do Python 3.6, era usado `$=formatting` ou `.format()`
- Agora no Python 3.6, temos o f-string que deixa tudo mais fácil
- Exemplo a seguir.

```
1 nome = 'Daniel'
2 idade = 36
3 print(f"meu nome é {nome} e eu tenho {idade} anos")
```



- Conceitos Básicos
- Instalação do MySQL
- Primeiros passos

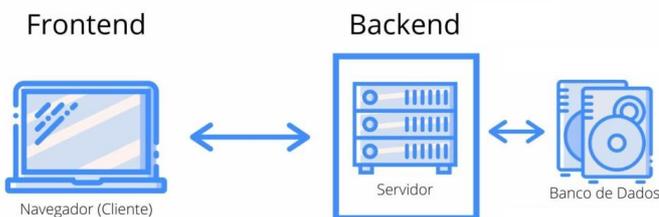
Banco de Dados

Conceitos Básicos

Tópicos Essenciais para Iniciar na Área de Banco de Dados

01 Iniciando

- Antes de iniciar nossa “prosa” sobre banco de dados, vamos entender o que é um banco de dados e onde ele se encaixa em um sistema.
- Como vimos nas folhas anteriores, utilizamos estruturas de dados como listas, tuplas, dicionários ou conjuntos para armazenar os dados. Entretanto, quando paramos o sistema e o executávamos novamente, perdemos todos os dados, concorda? Salvar em arquivos não considero uma opção, pois não entregamos sistemas que salva os dados em arquivos.
- Neste sentido, usamos os bancos de dados, que em uma analogia bem bacana, podem ser pensadas como planilhas do Microsoft Excel, que vai guardar os dados em um determinado formato que, sempre que os dados da aplicação forem manipulados, a planilha é atualizada de modo que, ao reiniciar a aplicação, os dados não são perdidos.
- Mas, em que lugar um banco de dados se encaixa em uma aplicação? A imagem a seguir detalha um pouco do funcionamento.



Fonte: Adaptado de <https://marquesfemendes.com/tecnologia/o-que-e-um-desenvolvedor-backend-e-o-que-de-faz/>

- À esquerda, temos o **frontend**, que é basicamente a interface que o cliente vai ter com o sistema, isto é, a tela, que por sua vez pode ser *web, desktop ou mobile* (smartphones). Nesta tela, preencheremos formulários, como por exemplo os dados para criação de uma nova conta no AbellaBank.
- Por exemplo, ao clicar no botão “Criar Conta”, os dados preenchidos são remetidos ao **backend**, que por sua vez, os armazena em um banco de dados. É assim que este mundo perfeito funciona!

02 “Leriado” Básico

- Primeiramente, o termo correto é **Sistema de Gerenciamento de Banco de Dados**, conhecido por sua sigla SGBD. Entretanto, quase todo mundo fala equivocado e confunde SGBD com **Banco de Dados** (BD).
- SGBD é um conjunto de programas de *software* que permite aos usuários criar, editar, atualizar, armazenar e recuperar dados em tabelas de banco de dados. Por outro lado, BD é uma coleção de dados relacionados.
- Na imagem a seguir creio que podemos tomar as coisas um pouco mais claras.
 - O SGBD é o lugar onde podemos ter vários bancos de dados. Exemplos de SGBD você já pode ter ouvido: Oracle, MySQL, PostgreSQL, Microsoft SQL Server, entre outros.



- Por exemplo, Daniel Abella, utilizo o SGBD MySQL. E, dentro do MySQL, eu crio um banco de dados para o meu sistema de ouvidoria e um banco de dados para cada um das minhas aplicações.
- E, dentro de cada banco de dados, podemos ter várias tabelas, que se parecem em muito com as velhas planilhas do Microsoft Excel. Ficou claro? **Um SGBD tem vários BD e cada BD tem várias tabelas.**
- Vamos omitir muita teoria e na próxima seção já iniciaremos a parte prática!

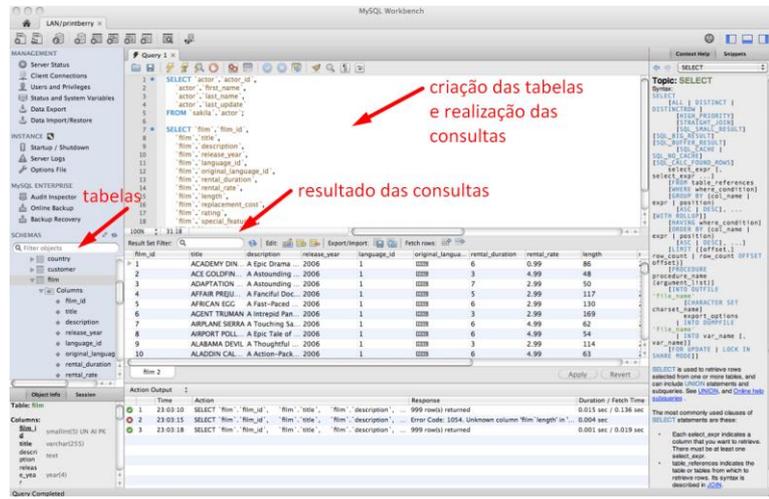
03 Alternativas para usar o MySQL

- Acabooooooooo o leriado! Aqui vamos apresentar duas alternativas de usar o MySQL.

- A primeira delas é utilizando uma ferramenta *online* para os primeiros passos.
- A última alternativa é utilizar as ferramentas apropriadas (e diga-se de passagem, profissionais) para isso.
 - Entenderam que a primeira maneira é só para, digamos, os primeiros dias?
- Cabe destacar que, tudo produzido aqui funciona em qualquer uma das alternativas.

04 Alternativa 1 - BD Fiddle

- **DB Fiddle** é uma ferramenta online que permite você “brincar” com o MySQL, além de outros SGBD como PostgreSQL e SQLite **sem precisar instalar nada**.
 - Lembra que, usamos o Replit para realizar as primeiras “brincadeiras” com Python? Mesma coisa aqui.
- O site é o seguinte <https://www.db-fiddle.com/>
- Uma outra ferramenta online é o **SQL Fiddle**, disponível no endereço <http://sqlfiddle.com/>. Uma boa explicação sobre a ferramenta é o link <https://www.youtube.com/watch?v=ZaOwbRDOqtK> (9 minutos e só).



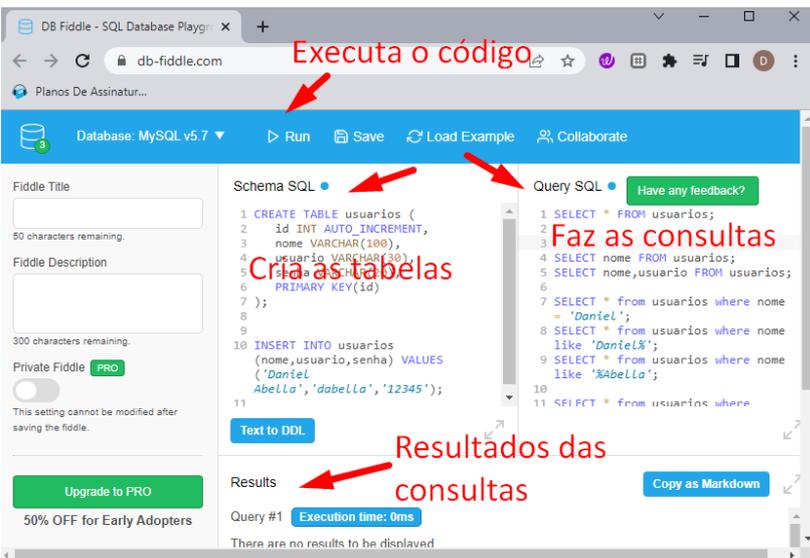
06 Passo 1 - Criar/Excluir o BD

- Conforme discutimos anteriormente, um SGBD possui vários bancos de dados, que por sua vez, possuem várias tabelas. Explicaremos aqui como criar um banco de dados.

! Para quem estiver ainda usando a ferramenta DB Fiddle, não vai ser necessário realizar este passo, pois ele já cria um automaticamente. Entretanto, para os usuários do Workbench, vai ser necessário.

- Antes de criar um banco de dados, sugere-se verificar os BD que estão criados por meio do comando: **SHOW DATABASES**
- Agora, para criar um BD de nome **ouvidoria** o comando é: **CREATE DATABASE ouvidoria;**
- Caso em algum momento precise excluir o BD criado anteriormente, o comando é **DROP DATABASE ouvidoria;** Lembrando que, uma vez excluído, todas as tabelas que estiverem dentro do BD serão excluídas também.

```
1 SHOW DATABASES
2 CREATE DATABASE ouvidoria;
3 DROP DATABASE ouvidoria;
```



05 Alternativa 2 - MySQL e Workbench

- Como dito anteriormente, sugere-se esta segunda alternativa em que o MySQL é instalado na sua máquina. Para isto, siga todos os passos descritos no site <https://www.alura.com.br/artigos/mysql-do-download-e-instalacao-ate-sua-primeira-tabela>
- Além do MySQL, você vai precisar instalar o **Workbench**, que é uma ferramenta para gerenciar os BD do MySQL. Ou seja, do Workbench que vamos criar as tabelas e fazer os comandos necessário para inserir, alterar, excluir ou listar os dados armazenados nas tabelas de um BD.
- Pelo amor de Deus, **ao instalar não esqueça de anotar o usuário e senha que você usou para criar o banco de dados!**

07 Passo 2 - Selecionar o BD Criado

- Na seção anterior criamos um banco de dados, não? Antes de começar a usar, precisamos indicar qual BD vamos utilizar. Para isto, usamos o comando **USE ouvidoria;**

```
1 USE ouvidoria;
```

08 Tipos de Dados em MySQL

- Imagine que vamos precisar criar uma tabela para armazenar todos os dados dos usuários do sistema. Caso não conhecêssemos BD e usássemos o Excel, fariamos algo como o da página a seguir. Note que, temos 4 colunas, sendo elas: ID (identificação), Nome, Usuário e Senha.

- E em nosso “banco de dados” temos 3 usuários, sendo o primeiro deles Daniel (nome) com ID possuindo valor 1, usuário dabella e senha 12345.

	A	B	C	D
1	ID	Nome	Usuário	Senha
2		1 Daniel	dabella	12345
3		2 Arthur	aabella	223344
4		3 Nathaly	nabella	334455

DON'T FORGET!

Utilizei a coluna ID para que, cada vez que um novo usuário seja inserido na tabela, incremente a coluna ID. Por exemplo, Daniel, como primeiro usuário na tabela, recebeu o ID 1, enquanto que, Arthur, como segundo usuário, teve o ID 2 e assim sucessivamente..

- Em Python não tínhamos os tipos de dados str, int, float, bool, etc? Em BD também temos!
- Apresento a seguir os principais tipos dentre os diversos existentes.

Tipo	Descrição	Exemplos
Int	Int do Python	1, 2
Float	Float do Python	1.1, 1.0
Varchar(N)	Str, onde N é o tamanho máximo da Str	“Oi”, “Daniel Abella”
Boolean	Bool do Python	true, false, TRUE, FALSE, True, False

- Agora que sabemos os tipos de dados, na seção a seguir vamos explicar como tirar do “Excel” para uma tabela do BD.

09 Passo 3 - Criar/Excluir uma Tabela

- Nos passos anteriores, criamos um BD chamado ouvidoria e o selecionamos para uso. Agora, vamos criar a nossa primeira tabela.
- Para criar a tabela, precisamos de duas coisas, o nome da tabela e o nome dos campos (e respectivos tipos).
- Baseando-se no Excel anterior, sugiro que a tabela se chame USUARIOS e que possua 4 campos, sendo eles ID, nome, usuário e senha.
- Para criar esta tabela, o comando a seguir deve ser utilizado, que possui os seguintes detalhes:
 - Linha 1:** O nome da tabela é **usuarios**,
 - Entre os parênteses **()**; temos a relação dos campos, neste exemplo, os 4 supracitados
 - Entre os campos, colocamos vírgulas

```

1 CREATE TABLE usuarios (
2   id INT AUTO_INCREMENT,
3   nome VARCHAR(100),
4   usuario VARCHAR(30),
5   senha VARCHAR(20),
6   PRIMARY KEY(id)
7 );

```

- Linha 2:** Criamos um campo chamado **ID** que é um inteiro.
 - Este campo é incrementável, ou seja, a cada registro novo, vai aumentando de 1 em 1
 - Para isso, usamos o termo **AUTO_INCREMENT**
- Linha 3:** Criamos um campo chamado **nome** que é uma String de até 100 caracteres
- Linha 4:** Criamos um campo chamado **usuario** que é uma String de até 30 caracteres
- Linha 5:** Criamos um campo chamado **senha** que é uma String de até 20 caracteres
- Linha 6:** Precisamos definir que coluna não pode repetir valores, ou seja, que é a chave primária (PRIMARY KEY - PK)
 - Neste exemplo, usamos o campo **id** como PK
- Caso queira excluir a tabela acima, basta um **DROP TABLE ouvidoria;**

```

1 DROP
2 TABLE ouvidoria;

```

10 Passo 4 - Inserir Linhas na Tabela

- Uma vez criada a tabela **USUARIOS**, a tabela está vazia, isto é, sem linhas. Para adicionar novos usuários, o comando é o seguinte: **INSERT INTO usuarios (nome,usuario,senha) VALUES ('Daniel Abella','dabella','12345');**

```

1 INSERT INTO usuarios (nome, usuario, senha)
2 VALUES
3 (
4   'Daniel Abella', 'dabella', '12345'
5 );

```

- Note que:
 - #1 Informamos o nome da tabela após o comando INSERT INTO
 - #2 Após o nome da tabela, entre os parênteses temos o nome das colunas que serão preenchidas
 - Se temos 4 colunas, porque não informamos a coluna ID
 - Porque colocamos esta coluna para incrementar (aumentar de 1 em 1) automaticamente
 - Desta maneira, não precisamos especificar este campo, pois “ele se vira sozinho”
 - Caso não tivesse o comando AUTO_INCREMENT, teríamos que especificar

- #3 Na sequência, temos a cláusula VALUES seguida por ();
 - Se em #2 especificamos 3 colunas, dentro do VALUES, colocaremos na mesma ordem os valores correspondentes

Te desafio a inserir também Arthur e Nathaly para ficar igual ao Excel.

	A	B	C	D
1	ID	Nome	Usuário	Senha
2		1 Daniel	dabella	12345
3		2 Arthur	aabella	223344
4		3 Nathaly	nabella	334455

11 Passo 5 - Listar as Linhas da Tabela

- Vamos neste passo realizar um SELECT, para listar todos os registros inseridos na tabela USUARIOS.
- Na sequência apresentamos um exemplo de alguns cenários comuns no uso de SELECT:

```

1 SELECT * FROM usuarios;
2 SELECT nome FROM usuarios;
3 SELECT nome,usuario FROM usuarios;
4
5 SELECT * from usuarios where nome = 'Daniel';
6 SELECT * from usuarios where nome like 'Daniel%';
7 SELECT * from usuarios where nome like '%Abella';
8
9 SELECT * from usuarios where usuario = 'dabella'
10 and nome = 'Daniel Abella';

```

- **Linha 1:** Apresentamos todos (por isso o *) campos de todos os usuários
- **Linha 2:** Apresentamos apenas o campo nome de todos os usuários
- **Linha 3:** Apresentamos os campos nome e usuário de todos os usuários
- **Linha 5:** Apresentamos todos os campos dos usuários que o campo nome é igual a Daniel
- **Linha 6:** Apresentamos todos os campos dos usuários que o campo nome inicia com Daniel
 - Para isso, usamos o like e, dentro da String, usamos o % para dizer que vem coisa depois
 - Se fosse os que terminassem com Abella, observe a linha 7
- **Linhas 9 e 10:** Apresentamos todos os campos dos usuários que o campo usuário seja dabella e o nome seja Daniel Abella
 - Para isso, usamos o and, como em Python
 - Ainda temos or e muitos outros operadores

12 Passo 6 - Atualizar as Linhas da Tabela

- Na seção anterior, aprendemos a inserir uma linha com registros em uma tabela. E, por alguma situação, precisássemos alterar o valor de uma linha? Para isto, utilizaremos o comando UPDATE, cujo exemplos e comentários estão a seguir.

```

1 UPDATE usuarios
2   SET nome = 'Daniel Abella Souza' WHERE id = 1;
3 UPDATE usuarios
4   SET nome = 'Daniel' WHERE id = 1 or usuario='dsouza';
5 UPDATE usuarios
6   SET nome = 'Dan' and usuario = 'ddan'
7     WHERE id = 1 and usuario='dsouza';

```

- **Linha 1:** Alteramos o campo **nome** para o usuário com **ID 1**
- **Linha 2:** Alteramos o campo **nome** para o usuário com **ID 1** ou **usuário dsouza**
- **Linha 3:** Alteramos os campos **nome** e **usuario** para o **usuário** com **ID 1** e **usuário dsouza**

13 Passo 7 - Excluir as Linhas na Tabela

- Para excluir as linhas da tabela **USUARIOS**, usamos o comando **DELETE**. Abaixo apresentamos os exemplos e as considerações:

```

1 DELETE FROM usuarios;
2 DELETE FROM usuarios WHERE id = 1;
3 DELETE FROM usuarios WHERE id = 1 or usuario='dsouza';
4 DELETE FROM usuarios WHERE id = 1 and usuario='dsouza';

```

- **Linha 1:** Excluimos todos os usuários (veja que não temos *, pois não excluimos uma dada coluna)
- **Linha 2:** Excluimos todos os usuários cujo ID seja 1
- **Linha 3:** Excluimos todos os usuários cujo ID seja 1 ou usuário dsouza
- **Linha 4:** Excluimos todos os usuários cujo ID seja 1 e usuário dsouza

14 E agora?

- Seguramente omitimos uma série de detalhes que serão aprofundados posteriormente. Entretanto, o conhecimento suficiente para iniciar nesta área de banco de dados está relacionado aqui. Na folha seguinte, apresentaremos como fazer essas operações (inserção, listagem, atualização e exclusão) a partir de um código em Python.

15 Resumo dos Comandos

Comando	Função	Exemplos
SELECT	Lista as linhas	<pre> 1 SELECT * FROM usuarios; 2 SELECT nome FROM usuarios; 3 SELECT nome,usuario FROM usuarios; 4 SELECT * from usuarios where nome = 'Daniel'; 5 SELECT * from usuarios where nome like 'Daniel%'; 6 SELECT * from usuarios where nome like '%Abella'; 7 SELECT * from usuarios where usuario = 'dabella' 8 and nome = 'Daniel Abella'; </pre>
UPDATE	Atualiza linhas	<pre> 1 UPDATE usuarios 2 SET nome = 'Daniel Abella Souza' WHERE id = 1; 3 UPDATE usuarios 4 SET nome = 'Daniel' WHERE id = 1 or usuario='dsouza'; 5 UPDATE usuarios 6 SET nome = 'Dan' and usuario = 'ddan' 7 WHERE id = 1 and usuario='dsouza'; </pre>
DELETE	Exclui as linhas	<pre> 1 DELETE FROM usuarios; 2 DELETE FROM usuarios WHERE id = 1; 3 DELETE FROM usuarios WHERE id = 1 or usuario='dsouza'; 4 DELETE FROM usuarios WHERE id = 1 and usuario='dsouza'; </pre>
SHOW DATABASES		
CREATE DATABASE db1	Cria o DB db1	
USE db1	Usa o db1	



- Classes na Cozinha
- Primeira Classe
- Classes c/ Método e Var
- Módulos e Import
- Classes e Listas
- Buscar objetos em lista
- Usando índices e listas

Python

Lógica de Programação

Classes, Módulos e Imports com Python

01 Classes na “Cozinha”

- Linguagens como Java são baseadas em Programação Orientada à Objetos (POO)
- A melhor forma que eu consigo explicar o conceito de Classes é usando conceitos da culinária. Classe é como se fosse uma forma de um bolo, como o da imagem a seguir.
- Uma Classe em Python agrupa métodos (funções) e variáveis (atributos). No exemplo a seguir, entre as linhas 1 e 5, temos uma classe chamada Conta, com 4 variáveis. Note que, cada variável eu imagino que é uma parte da forma do bolo (Classe).



```
1 class Conta:
2     agencia = 0
3     conta = 0
4     saldo = 0.0
5     titular = ''
```



```
6 contaDaniel = Conta()
7 contaNathaly = Conta()
```

- Como toda forma de bolo, a partir dela geramos bolos. Traduzindo para programação:
 - Classe = forma do bolo
 - Instância ou Objetos = bolo gerado a partir da forma

- Na linha 6 acima, temos um objeto (bolo) cuja variável se chama contaDaniel. Na linha 7, agora você já sabe o que é.

02 Criando sua Primeira Classe

- Abaixo (linhas 1 e 2) temos a criação de uma classe vazia (ou seja, sem nada) chamada Conta, enquanto que, na linha 4, criamos um objeto chamado novaConta.



Note que, classe inicia sempre com Maiúsculo, enquanto métodos com minúsculos. E, quando a classe ou método estiver vazio, precisamos por a palavra pass. Detalhes em <https://legacy.python.org/dev/peps/pep-0008/>.

```
main.py x
1 class Conta:
2     pass
3
4 novaConta = Conta()
```

03 Classes com Métodos e Variáveis

- Na classe a seguir, temos uma classe Conta com uma variável chamada cotacaoDolar e um método chamado converterDolares, que por sua vez, recebe o valor em reais e dá um print no valor em dólares.
 - O que fez esse self no método? Em métodos dentro de classes, obrigatório!
 - Na linha 9, criamos um objeto chamado novaConta
 - Na linha 10, chamamos o método que converte usando o objeto da linha 9

```
1 class Conta:
2
3     cotacaoDolar = 4.9
4
5 def converterDolares(self, valorReais):
6     print(valorReais * self.cotacaoDolar)
7     print(valorReais * 4.9)
8
9 novaConta = Conta()
10 novaConta.converterDolares(100)
--
```

- No código acima, nas linhas 6 e 7, imprimimos o valor em reais de duas maneiras (sem variável e com variável). Prefiro a opção da linha 6, pois facilita a manutenção
 - Se amanhã o valor do dólar for 5.0, ajusto apenas a variável da linha 3
 - Note que, na linha 6, usamos self.cotacaoDolar
 - Self é obrigatório, lembre-se!
- E se, ao invés de imprimir (print), eu queira retornar (melhor opção), como seria? Código a seguir.
 - Note que, na linha 9, imprimimos o retorno do método
 - Alternativamente, na linha 11, atribuímos o retorno a uma variável e na linha 12, imprimimos a variável

```

1 class Conta:
2
3     cotacaoDolar = 4.9
4
5     def converterDolares(self, valorReais):
6         return valorReais * self.cotacaoDolar
7
8 novaConta = Conta()
9 print(novaConta.converterDolares(100))
10
11 valorEmDolares = novaConta.converterDolares(100)
12 print(valorEmDolares)

```

- Vamos agora criar uma classe mais próxima a realidade? Uma classe geralmente tem, ao menos, número da agência, número da conta, titular e o saldo, concordam? Exemplo abaixo.

```

1 class Conta:
2     agencia = 0
3     conta = 0
4     saldo = 0.0
5     titular = ''
6
7     def __init__(self, a, c, s, t):
8         self.agencia = a
9         self.conta = c
10        self.saldo = s
11        self.titular = t

```

- Mas, o que “danado” é esse método `init`? Ele é um método que comumente chamamos de **inicializador** ou **construtor**. Ou seja, com ele, eu consigo já criar um projeto com os valores, como vou mostrar a seguir.
 - Na página anterior, **sem construtor**, lembram que criamos um objeto da seguinte maneira: `novaConta = Conta()`

! No construtor acima, a linha 7 criamos um construtor, que é um método com nome `__init__`. **Como todo método de classe, o primeiro parâmetro precisa ser o `self`**. Após o `self`, temos 4 parâmetros (`a,c,s,t`) que guardam os valores informados pelo usuário. E, entre as linhas 8 e 11, atribuímos às variáveis das linhas 2 a 5 os valores recebidos pelos 4 parâmetros. Por exemplo, na linha 8, a variável de classe `agencia`, guarda o valor do parâmetro `a`.

- Enfim, agora, **com um construtor**, criamos o objeto da maneira das linhas 13 e 16 abaixo.
 - E, nas linhas 14 e 17, apresentamos respectivamente o saldo de Abella (100) e de Arthur (1000).

```

12
13 contaDaniel = Conta(1,123,100,'Abella')
14 print(contaDaniel.saldo)
15
16 contaArthur = Conta(1,124,1000,'Arthur')
17 print(contaArthur.saldo)

```

04 Módulos e Import

- Até o presente momento, tudo o que fazemos é dentro do arquivo `main.py`. Conforme os sistemas sejam maiores, isto não é escalável. Para isto, usamos o conceito de **módulos**.
 - Em resumo, módulos são arquivos com extensão `py` que podem armazenar métodos, classes e variáveis
 - No exemplo a seguir, criamos um módulo chamado `calculadora`, que basicamente é um arquivo `calculadora.py` e possui um método `soma`.
 - **Não tem `self` aqui porque não está dentro de uma classe.**

```

calculadora.py ×
1 def soma(x,y):
2     return x+y

```

- E, como faço para usar um módulo? No código a seguir, na linha 1, importamos o módulo `calculadora` por meio da palavra-chave **`import`**
 - Para usar, chamamos de uma das 3 seguintes maneira:
 - **`modulo.metodo`**
 - **`modulo.variável`**
 - **`modulo.Classe`**

```

main.py ×
1 import calculadora
2
3 print(calculadora.soma(2,2))

```

- Se o módulo `calculadora` acima tivessem mais outros métodos, chamaríamos da mesma maneira que chamamos a `soma`.

! No exemplo acima importamos o módulo inteiro (ou seja, tudo que tiver dentro do módulo). Mas, se eu quiser importar apenas um método de um módulo? **`from modulo import método`**

```

1 from calculadora import soma
2
3 print(soma(2,2))

```

Se você importa por método, **podemos chamar `soma` diretamente**. Ou seja, não precisa fazer `calculadora.soma(2,2)`, como tínhamos feito anteriormente.

- Uma excelente ideia é, colocar as classes dentro de módulos. Note que, no código a seguir, criamos um módulo chamado `conta`, que possui um construtor / inicializador e um método `sacar`. Que, recebe um montante e diminui do saldo.

conta.py ×

```
1 class Conta:
2     agencia = 0
3     conta = 0
4     saldo = 0.0
5     titular = ''
6     cotacaoDolar = 4.9
7
8 def __init__(self, a, c, s, t):
9     self.agencia = a
10    self.conta = c
11    self.saldo = s
12    self.titular = t
13
14 def sacar(self, montante):
15    self.saldo = self.saldo - montante
```

- No código abaixo, explico como usar o módulo criado anteriormente.
 - **Linha 1:** Importamos o módulo inteiro
 - **Linha 3:** Criamos um objeto para Daniel com saldo 1000
 - **Linha 4:** Chamamos o método sacar na variável novaConta
 - **Linha 5:** Apresentamos o novo saldo (900)

main.py ×

```
1 import conta
2
3 novaConta = conta.Conta(123,442233,1000,'Daniel')
4 novaConta.sacar(100)
5 print(novaConta.saldo)
```

05 Classes e Listas

- Agora que já sei como funcionam as classes, posso adicionar um objeto à uma lista? Por exemplo, termos uma lista de contas de uma dada agência bancária? Sim, no exemplo a seguir apresento este exemplo.

main.py ×

```
1 import conta
2
3 listaContasAgencia = [conta.Conta(123,442232,100,'Daniel')]
4
5 novaConta2 = conta.Conta(123,442233,101,'Nathaly')
6 novaConta3 = conta.Conta(123,442234,102,'Arthur')
7
8 listaContasAgencia.append(novaConta2)
9 listaContasAgencia.append(novaConta3)
10
11 print(len(listaContasAgencia))
12
13 for con in listaContasAgencia:
14     print('Agencia ',con.agencia, ' Conta ', con.conta)
```

- **Linha 1:** Importamos o módulo inteiro
- **Linha 3:** Criamos uma lista normalmente
- **Linha 3:** Adicionamos “de cara” um objeto para Daniel
 - Não era necessário, mas quis mostrar como seria

- **Linhas 5 e 6:** Criamos 2 objetos
 - Não foram inseridos à lista ainda
- **Linhas 8 e 9:** Adicionamos os 2 objetos a lista com append
 - Não mudou nada 😊
- **Linha 11:** Apresentamos o tamanho da lista (3)
- **Linhas 13 e 14:** Apresentamos todos os objetos que estão na listas

06 Buscar objetos em lista

- Quando usamos listas com *int*, *float*, *bool* ou *str*, a busca era bem simples. Agora, quando a lista contém objetos, o negócio “é mais embaixo”. O exemplo completo está a seguir.
 - **Linha 12:** Criamos uma variável vazia (None)
 - **Linha 15:** Verifica se a variável titular do objeto (con) é igual a ‘Daniel’
 - **Linha 16:** Atribuímos o objeto (con) à variável da linha 12
 - **Linha 17:** Se eu achar o objeto que eu preciso, por que eu preciso ir até o final da lista? “Meto” um break para parar a busca.
 - **Linhas 19 e 20:** Apresenta os dados da conta buscada (de Daniel)

main.py ×

```
1 import conta
2
3 listaContasAgencia = []
4
5 novaConta1 = conta.Conta(123,442232,100,'Daniel')
6 novaConta2 = conta.Conta(123,442234,102,'Arthur')
7
8 listaContasAgencia.append(novaConta1)
9 listaContasAgencia.append(novaConta2)
10 print(len(listaContasAgencia))
11
12 contaDaniel = None
13
14 for con in listaContasAgencia:
15     if con.titular == 'Daniel':
16         contaDaniel = con
17         break
18
19 print('agencia ',contaDaniel.agencia)
20 print('saldo ',contaDaniel.saldo)
```

07 Usando Lista de Objetos com Índices

```
1 import conta
2
3 listaContasAgencia = []
4 novaConta1 = conta.Conta(123,442232,100,'Daniel')
5 listaContasAgencia.append(novaConta1)
6 contaDaniel = None
7
8 for i in range(len(listaContasAgencia)):
9     if listaContasAgencia[i].titular == 'Daniel':
10         contaDaniel = listaContasAgencia[i]
11         break
12
13 print('saldo ',contaDaniel.saldo)
```



- Pré-requisitos
- Preparação de Terreno
- Inserindo no BD
- Listagem do BD
- Exclusão do BD
- Atualização do BD
- Resumo das Operações do BD

Python

Lógica de Programação

Conectando Python com o Banco de Dados MySQL

01 Pré-Requisitos

- Neste *cheat sheet*, aprenderemos como interagir com Banco de Dados MySQL usando o Python
- Os pré-requisitos são os seguintes:
 - Instalação do Python 3
 - <https://realpython.com/installing-python/> ou
 - <https://www.python.org/downloads/>
 - Instalação do Conector do MySQL
 - No *prompt*, digite o seguinte comando:
 - `python3 -m pip install mysql-connector-python`
 - Instalação do MySQL e Workbench
 - <https://www.alura.com.br/artigos/mysql-do-download-e-instalacao-ate-sua-primeira-tabela>

02 Preparação do Terreno

- O que precisamos agora é criar as nossas tabelas que vamos usar. Para isto, precisamos realizar os seguintes passos:

Comandos (Procedimentos)	Descrição
<code>CREATE DATABASE ouvidoria;</code>	Cria o BD ouvidoria
<code>USE ouvidoria;</code>	Seleciona o BD ouvidoria para uso
<pre> 1 CREATE TABLE usuarios (2 id INT AUTO_INCREMENT, 3 nome VARCHAR(100), 4 usuario VARCHAR(30), 5 senha VARCHAR(20), 6 PRIMARY KEY(id) 7); </pre>	Cria a tabela usuarios com os campos ID, nome, usuário e senha.

03 Inserindo no BD

- No código a seguir, realizamos uma inserção de elementos no BD.
 - Linha 1: Importamos a biblioteca do MySQL
 - Linha 3 a 8: Criamos uma conexão com BD, que está funcionando na mesma máquina (localhost), com usuário root, senha 2424 e o nome do BD é ouvidoria.
 - Poderia ter feito em uma linha só, mas quebramos em várias para ficar mais entendível

```

1 import mysql.connector
2
3 connection = mysql.connector.connect(
4   host="localhost",
5   user="root",
6   password="2424",
7   database="ouvidoria"
8 )
9
10 cursor = connection.cursor()
11
12 sql = "INSERT INTO usuarios (nome, usuario, senha) VALUES (%s, %s, %s)"
13 data = ('Daniel Abella', 'dabella', '12345')
14
15 cursor.execute(sql, data)
16 connection.commit()
17
18 userid = cursor.lastrowid
19
20 cursor.close()
21 connection.close()
22
23 print("ID do novo usuário:", userid)
                    
```

- **Linha 10:** usamos a conexão criada para criar um cursor, que é quem realiza as consultas no BD.
- **Linha 12:** criamos uma String com a consulta que faremos
 - Note que, os 3 valores de um novo usuário usamos %s
- **Linha 13:** criamos uma tupla com 3 elementos, que são os 3 valores de um novo usuário
- **Linha 15:** O cursor executa o SQL com os valores da tupla
 - A inserção foi feita no BD? Ainda não ☹️ Precisamos chamar a linha 16, que é o commit
- **Linha 16:** O *commit* é quem é o cara! É agora que a inserção solicitada na linha 15 é realizada!



Por que precisa do *commit* da linha 16? No exemplo acima, temos apenas uma operação de *execute* (linha 15). Mas, antes do *commit*, poderíamos ter inúmeras operações *execute* (além da linha 15). E, todas essas operações só seriam enviada para o BD após o *commit*.

- **Linha 18:** Pega o ID gerado automaticamente na inserção
- **Linha 20:** Terminou de usar o cursor? Feche com *close*.
- **Linha 21:** Terminou de usar a conexão? Feche com *close*.
- **Linha 23:** Imprime o ID Gerado na inserção.

Em resumo:

- `Execute()` envia o comando ao BD
- `Commit()` efetiva todos os comandos enviados pelos `Execute()` executados antes do `commit`

04 Listagem do BD

- Nesta seção apresentaremos como realizar um `SELECT` na base de dados.
 - Linha 12: Select que será realizado
 - Linha 14: Executa o comando de Select
 - Linha 15: Executa um comando de listar tudo chamado `fetchall`
 - A variável `listaUsuarios` recebe uma lista com tuplas que representa os resultados
 - Linha 20: Apresenta a `listaUsuarios` sem segredo
 - Linha 21: Apresentamos a tupla com os dados de cada usuário
 - Linha 22: Maneira usual de apresentar os dados.
 - Só apresentei os 2 primeiros campos :P

```
1 import mysql.connector
2
3 connection = mysql.connector.connect(
4     host="localhost",
5     user="root",
6     password="2424",
7     database="ouvidoria"
8 )
9
10 cursor = connection.cursor()
11
12 sql = "SELECT * FROM usuarios"
13
14 cursor.execute(sql)
15 listaUsuarios = cursor.fetchall()
16
17 cursor.close()
18 connection.close()
19
20 for usuario in listaUsuarios:
21     print(usuario)
22     print(usuario[0], usuario[1])
```

05 Exclusão do BD

- Linha 12: Consulta para exclusão (exclui se o ID for igual a 1)
- Linha 13: Informa o valor do ID (neste exemplo, 1) Usamos uma tupla
- Linha 18: Obtemos a quantidade de linhas afetadas pela consulta.
- Linha 23: Imprime a quantidade de linhas afetadas.

```
1 import mysql.connector
2
3 connection = mysql.connector.connect(
4     host="localhost",
5     user="root",
6     password="2424",
7     database="ouvidoria"
8 )
9
10 cursor = connection.cursor()
11
12 sql = "DELETE FROM usuarios WHERE id = %s"
13 data = (2,)
14
15 cursor.execute(sql, data)
16 connection.commit()
17
18 recordsaffected = cursor.rowcount
19
20 cursor.close()
21 connection.close()
22
23 print('linhas afetadas ', recordsaffected)
```

06 Atualização do BD

- Para finalizar, a seguir usamos o comando `UPDATE`. Creio que, nenhuma linha abaixo é novidade, dado os exemplos apresentados anteriormente.

```
1 import mysql.connector
2
3 connection = mysql.connector.connect(
4     host="localhost",
5     user="root",
6     password="2424",
7     database="ouvidoria"
8 )
9
10 cursor = connection.cursor()
11
12 sql = "UPDATE usuarios SET nome = %s, usuario = %s WHERE id = %s"
13 data = ('Abella', 'abellad', 1)
14
15 cursor.execute(sql, data)
16 connection.commit()
17
18 recordsaffected = cursor.rowcount
19
20 cursor.close()
21 connection.close()
22
23 print('linhas afetadas ', recordsaffected)
```

07 Resumo das Operações com BD

- `SELECT ... FROM ... WHERE ...`
- `INSERT INTO ... VALUES ...`
- `UPDATE ... SET ... WHERE ...`
- `DELETE FROM ... WHERE ...`





- Introdução
- Criando seu projeto
- Ambiente PyCharm
- Palavra Bug
- Breakpoints
- Benefício Debugger
- Como usar
- Principais funções
- Atalhos
- Desafio

Python

Lógica de Programação

Ferramentas PyCharm



01 Introdução

- Neste *cheat sheet*, apresentaremos a ferramenta PyCharm, usada para desenvolvimento com Python, provida pela JetBrains
 - <https://www.jetbrains.com/pt-br/pycharm/download/>
- As ferramentas de desenvolvimento, chamamos de *Integrated Development Environment (IDE)*.
- O PyCharm tem 2 versões: uma chamada *Professional*, que é paga e uma intitulada *Community*, que é gratuita e a utilizada neste curso.

Baixar PyCharm

Windows macOS Linux

Professional

Para desenvolvimento Web com Python e desenvolvimento científico. Com suporte para HTML, JS e SQL.

Baixar

Avaliação gratuita por 30 dias disponível

Community

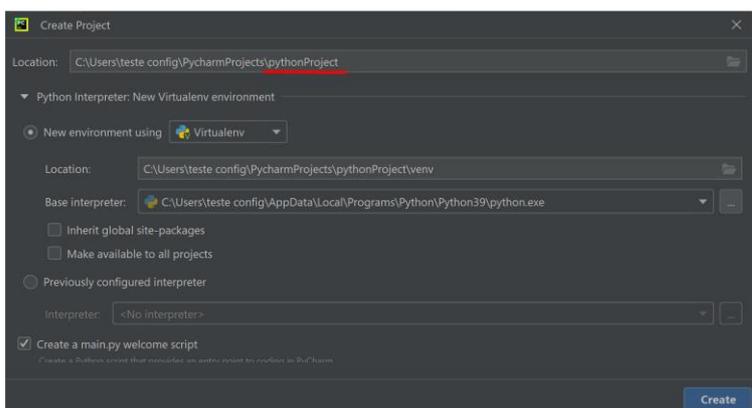
Para o autêntico desenvolvimento Python

Baixar

Gratuito, com base em open source

02 Criando o seu Projeto

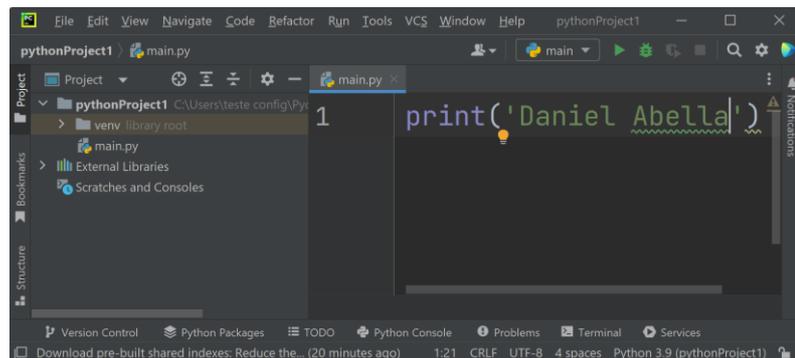
- Pycharm solicita que, sempre que for codificar algo, seja criado um projeto. Ao iniciar o Pycharm pela primeira vez, a janela a seguir é apresentada. Em **vermelho** deve ser inserido o nome do projeto.



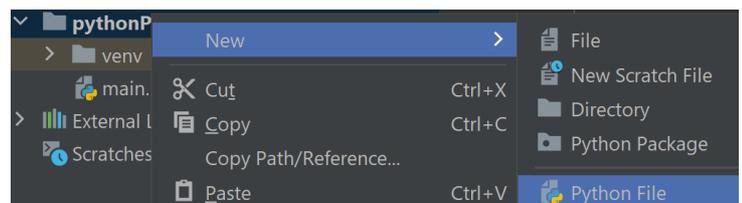
- Caso você já tenha criado algum projeto antes, o último projeto aberto é carregado. Neste caso, para criar um novo projeto, temos que ir em **File** e depois em **New Project...**

03 Ambiente PyCharm

- Uma vez o projeto foi criado, a tela a seguir é apresentada. No lado esquerdo, temos o nome do projeto **pythonProject1** e temos o arquivo onde vamos iniciar o código **main.py**.
- Caso você clique em **main.py** no lado esquerdo, o arquivo é aberto no lado direito, inclusive eu coloquei um `print('Daniel Abella')`.



- Para executar o projeto, clicamos no ícone ou **Shift + F10**
- Para interromper a execução do projeto, clicamos no ícone ou **Ctrl + F12**
- Para criar novos arquivos (módulos), clique com o botão direito no projeto, depois em **New** e logo depois em **Python File**, conforme imagem a seguir.

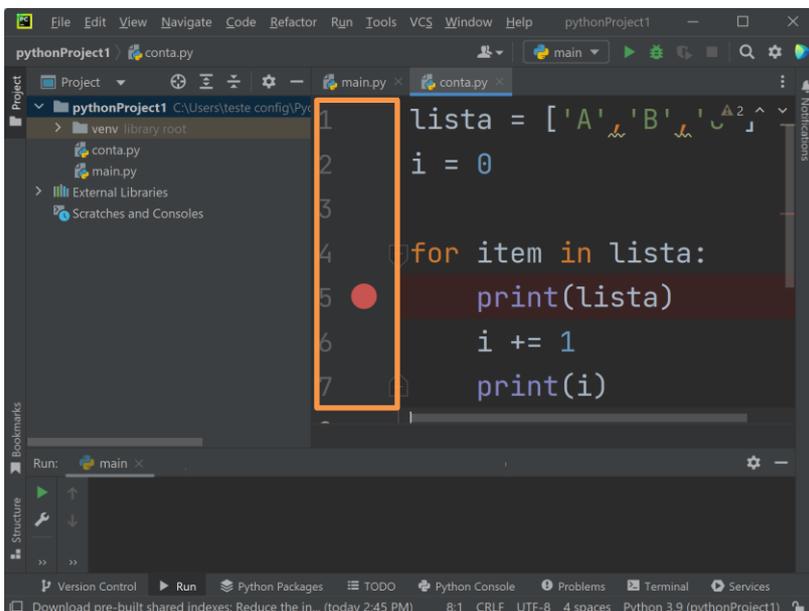


04 Origem da Palavra Bug

- Bug é uma palavra em inglês que significa inseto e foi usada para definir problemas por Grace Hopper, programadora da marinha dos EUA em 1947. Ela usou este termo (Bug) porque um inseto ficou preso nos contatos de um relê, causando mau funcionamento do computador Mark II na época. E até hoje o termo é usado.

05 Debugger e Breakpoints

- *Debugger* (Depurador, em Português) é a ferramenta ideal para você entender como o seu código está funcionando e identificar possíveis bugs.
- Como funciona? Basicamente você vai definir um ou mais pontos onde você gostaria que a execução desse uma pausa e a partir daí, você acompanhar.
 - Ou seja, o código roda normalmente e assim que chegar nesse ponto, ele pausa imediatamente a execução e você controla a execução, isto é, vendo o que acontece linha a linha
 - E, durante isso, você fica sabendo os valores das variáveis
 - Esses pontos de interrupção, chamamos de **breakpoints**
- Para criar um **breakpoint**, clique na linha desejada na área em **laranja** indicada na imagem a seguir.
 - Uma vez selecionada a linha, clique de modo que o ícone seja apresentado. Na imagem a seguir, colocamos o breakpoint na linha 5, isto é, quando a linha 5 for executada, a execução é pausada.

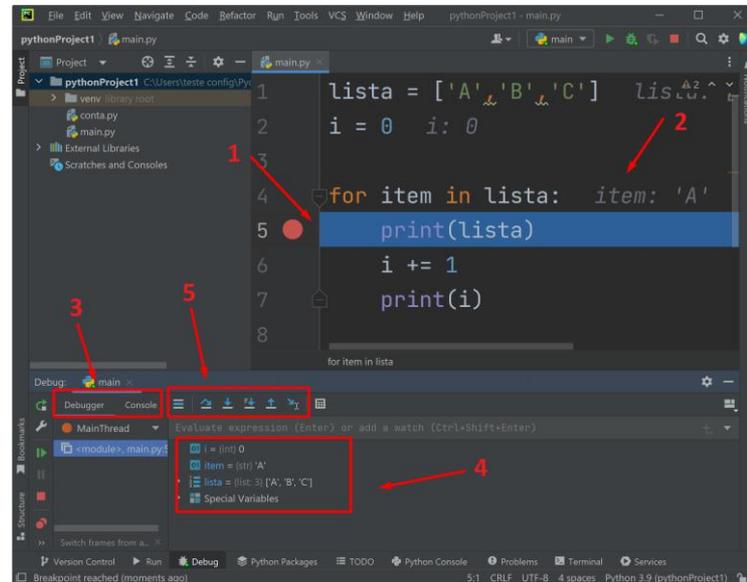


06 Benefícios do Debugger

- Um dos principais benefícios, como dito anteriormente, é a possibilidade de definir **breakpoints** e acompanhar a execução a partir deste ponto
- Entretanto, outros benefícios podem ser listados:
 - Verificar os valores que foram atribuídos as variáveis
 - Acompanhar a execução de métodos
 - E uma coisa muito legal que é, alterar o valor de variáveis enquanto ainda estiver executando.
 - Ou seja, permite alterar o valor em memória de uma variável

07 Como depurar?

- Uma vez que você colocou os **breakpoints**, deve clicar no ícone para iniciar a execução em modo *debug*.
 - Alternativamente, você pode usar **⇧ Shift + F9**
 - Só assim os **breakpoints** serão considerados.



- Na imagem anterior, iniciamos o debug. Destaco 5 pontos (indicados por setas):
 - **Seta Número 1:** A linha 5, indica que, a execução foi interrompida devido ao breakpoint da linha 5
 - **Seta Número 2:** Para ajudar no entendimento do código, o PyCharm coloca em uma fonte com cor bem clara o valor variável. Neste exemplo, a variável `item` está com o valor 'A'
 - **Seta Número 3:** Nesta seção, indicamos o que gostaríamos de visualizar no PyCharm, que pode ser o **Debugger** (que permite a visualização do valor das variáveis, indicado na Seta 4) ou **Console**, que apresenta a saída do programa (geralmente, oriunda de prints)
 - **Seta Número 4:** Se na área indicada na seta de número 3, você selecionou **Debugger**, você vai poder visualizar os valores das variáveis do sistema, como elas estão na memória
 - **Seta Número 5:** Principais funções do depurador para poder acompanhar o código (descritas na seção a seguir).

08 Principais Funções para Acompanhar o Código

- Ao iniciar o Debug, na parte inferior, destacada na Seta 5 da página anterior, temos alguns ícones para as funções de acompanhar o código com Debug.

- Na imagem a seguir, apresentamos as principais funções que vocês precisam conhecer.



- Iniciar o Debug**
 - Opção 1: Clicar em
 - Opção 2:
- Step Over (Executa a Linha de Código onde o Cursor Está)**
 - Opção 1: Clicar em
 - Opção 2:
- Step Into (Entra no Método e Chama o Método)**
 - Opção 1: Clicar em
 - Opção 2:
- Step Out (Termina o Debug da Função e o Cursor Volta ao Local Anterior)**
 - Opção 1: Clicar em
 - Opção 2:

09 Principais Atalhos do PyCharm

- Abaixo relacionamos os principais atalhos do PyCharm. Entretanto, nos links a seguir você pode verificar mais atalhos:
 - Verificar todos os disponíveis no próprio PyCharm
 - Acessar o menu depois
 - Clicamos em na janela apresentada
 - <https://www.shortcutfoo.com/app/dojos/pycharm-win/cheatsheet>

Atalho	Função
	Completa o código
	Comenta ou descomenta com '''
	Comenta ou descomenta com #
	Exclui uma linha
	Seleciona todas (All) linhas do arquivo
e	Pula de um método pra outro
	Mostra os arquivos abertos recentemente
	ctrl + C junto com Ctrl + V do código selecionado

Atalho	Função
	Busca (find)
e	Move as linhas selecionadas
	Importa os pacotes ausentes
	Otimiza seus pacotes importados, remove os não utilizados
	Formata (Identa) uma linha de código

CHALLENGE

10 Desafio

- Acompanhe o código abaixo usando o Debug do PyCharm.

```

main.py x
1 num_list = [500, 600, 700]
2 alpha_list = ['x', 'y', 'z']
3
4
5 def nested_loop():
6     for number in num_list:
7         print(number)
8         for letter in alpha_list:
9             print(letter)
10
11 nested_loop()

```

11 Quer Saber Mais?

- <http://excript.com/python/depuracao-pycharm-python.html>
- <https://medium.com/analytics-vidhya/useful-pycharm-shortcuts-65343a72e6f2>
- <https://www.shortcutfoo.com/app/dojos/pycharm-win/cheatsheet>
- <https://github.com/nareddy/cs3600-pycharm-debugging/blob/master/pycharm-setup-and-debugging.md>
- <https://realpython.com/pycharm-guide/>



- Introdução
- Implementação
- Testes
- Cliente com Python
- Hospedagem no Heroku
- Mais

Python

Criando WebServices com Python

Usando Flask

01 Introdução

- Arthur Abella é um programador que gostaria de criar uma aplicação em Python para conversão de reais a dólares.
- Inicialmente, ele fez o código a seguir.

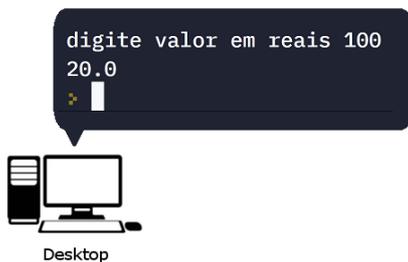


```
main.py ×
1 ▼ def converterParaDolares():
2     reais = float(input('digite valor em reais'))
3     dolares = reais / 5
4     print(dolares)
5
6 converterParaDolares()
```

- Inicialmente, Arthur ficou muito feliz e mostrou o resultado ao seu pai, que fez as seguintes considerações.

Consideração 1

- O aplicativo funciona em computadores (*Desktop*). Ou seja, se amanhã você quiser criar um aplicativo para celular (*mobile*) ou *web* (via *browser*) que chame este método não vai funcionar.



Consideração 2

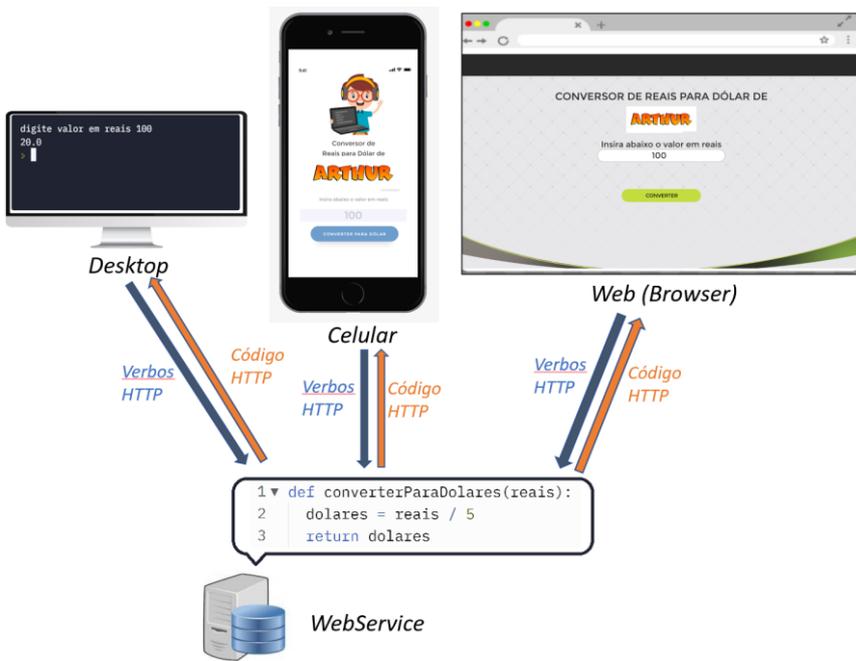
- Tão breve Daniel Abella fez a primeira consideração, seu primogênito o questionou o motivo.
 - O motivo é que, dentro do método não é interessante colocar leitura de dados (com `input`, por exemplo), nem tampouco apresentação de dados (com `print`, por exemplo).
 - Oportunamente, Arthur apresentou uma nova versão do seu código.

```
main.py ×
1 ▼ def converterParaDolares(reais):
2     dolares = reais / 5
3     return dolares
4
5 valor = float(input('digite valor em reais '))
6 valorEmDolares = converterParaDolares(valor)
7 print(valorEmDolares)
```

- Agora o código está elegível para ser chamado por celulares e web, pois a leitura de dados é fora do método (linha 5) e a apresentação da conversão é realizada também fora do método (na linha 7).

Consideração 3

- Arthur ficou empolgado com a ideia de fazer o aplicativo dele funcionar em celulares, web e desktops. E, questionou como isto poderia funcionar. Sendo assim, Daniel definiu alguns passos para fazer.
 - **Passo 1:** A primeira coisa é transformar o seu método `converterParaDolares` em um webservice, que é o que vamos explicar neste resumo.
 - De maneira simplória, **imagine webservice como um método que pode ser chamado a partir de qualquer aplicação**, seja ela para celular, computador ou browser.
 - **Passo 2:** Desenvolver 3 aplicações (um para celular, um para web e uma para desktops) que fazem a conversão de reais para dólares.
 - E, fazer com que, esses 3 aplicativos chamem o webservice desenvolvido no passo 1.
 - Arthur, ligado no 220 como o pai, desenhou a ideia proposta por seu pai, apresentado na folha seguinte.
 - Arthur detalhou a sua proposta, que é de:
 - Desenvolver uma aplicação para celulares usando Java (ou outras linguagens), que chama o webservice que faz a conversão
 - Desenvolver uma aplicação para computadores usando Python (ou outras linguagens), que chama o webservice que faz a conversão
 - Desenvolver uma aplicação para browsers usando Javascript (ou outras linguagens), que chama o webservice que faz a conversão



- Note que, todos chamam o mesmo método (*converterParaDolares*). Se amanhã mudar a cotação do dólar, eu precisaria apenas mudar na linha 2 (onde está 5) e, automaticamente todos os 3 aplicativos refletiriam a mudança.

Detalhes Técnicos

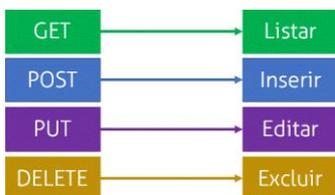
- Como vimos, os 3 aplicativos podem ser feitos em qualquer linguagem e podem chamar o nosso webservice *converterParaDolares*, que foi implementado em Python.
 - Como isso é possível? Vamos utilizar o protocolo HTTP, que possivelmente vocês não conhecem a fundo, mas é o que permite você acessar a Web.
- No HTTP, temos verbos que são a maneira com que o cliente (neste caso, um dos 3 aplicativos) chama o webservice.

💡

- A partir de agora, quando eu usar a palavra *cliente* (ou *client*, em inglês), estou me referindo “ao cara” que chama o webservice.
- Em nosso exemplo, temos 3 cliente para o webservice *converterParaDolares*: Um aplicativo para celular, um aplicativo para web (browser) e um aplicativo para desktop.

- O cliente chama o webservice de uma das maneiras a seguir:

Verbo	Significado
GET	Cliente para WebService: “Ei, me dá a lista de todos os usuários que tem no sistema”
POST	Cliente para WebService: “Ei, toma os dados de um novo usuário, podes inserir?”
PUT	Cliente para WebService: “Ei, tem como alterar os dados do usuário 123? Toma!”
DELETE	Cliente para WebService: “Ei, tem como excluir todos os dados do usuário 123?”



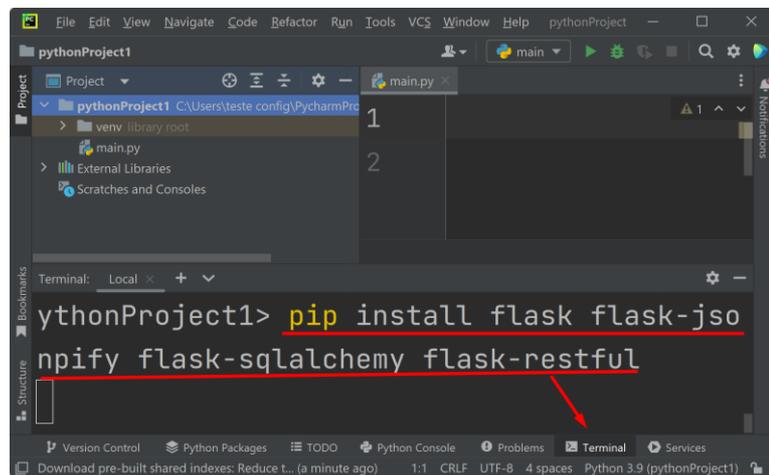
- E, quando o WebService recebe algum dos pedidos anterior (verbos HTTP), ele pode responder da seguinte forma:

Código	Significado
200	WebService para Cliente: “OK!” ou WebService para Cliente: “OK, toma aí o que você me pediu (lista de usuários)”
201	WebService para Cliente: “Created” pois, inseri um novo usuário no sistema
204	WebService para Cliente: “No content” pois, não achei o que você me pediu
400	Bad Request
404	Not Found
500	WebService para Cliente: “Internal Server Error” pois deve ter dado um pau da porra no webservice

- Para conhecer todos os códigos, eu sugiro conhecer o site <https://httpstatusdogs.com/>

02 Implementação

- Vamos tomar o sonho de Arthur uma realidade?
 - Vamos implementar um *WebService* que realiza a conversão de reais para dólar.
- Passo 1:** Cria um projeto no Pycharm
- Após criado o projeto, na parte inferior do *Pycharm*, conforme indicado na seta da imagem a seguir, clique em Terminal. Posteriormente, insira o seguinte comando: `pip install flask flask-jsonpify flask-sqlalchemy flask-restful`



- Por meio deste comando, você instalará as seguintes dependências
 - Flask:** Flask é um micro framework que utiliza a linguagem Python para criar aplicativos Web.
 - Flask-Restful:** Extensão para Flask que adiciona suporte para a construção rápida de APIs REST. É uma abstração leve que funciona com seu ORM/bibliotecas existentes. Flask-RESTful incentiva as melhores práticas com configuração mínima. Se você estiver familiarizado com o Flask, o Flask-RESTful deve ser fácil de entender.

Dict → JSON

```
{'name': 'Apple',  
 'color': 'Red',  
 'quantity': 10 }
```

```
{'name': "Apple", "color":  
 "Red", "quantity": 10 }
```

Fonte: <https://favtutor.com/blogs/dict-to-json-python>

- **Passo 2:** Importar as dependências necessárias para criar um webservice e relacionar as variáveis necessárias na aplicação

```
1 from flask import Flask, request, jsonify  
2 from flask_restful import Resource, Api  
3  
4 app = Flask(__name__)  
5 api = Api(app)
```

- **Passo 3:** Incluir o nosso método sem nenhuma alteração

```
8 def converterParaDolares(reais):  
9     dolares = reais / 5  
10    return dolares
```

- **Passo 4:** Criamos o nosso webservice que chama o método do passo 3

```
13 class Cotacao(Resource):  
14     def post(self):  
15         valorEmReal = float(request.json['valor'])  
16         result = converterParaDolares(valorEmReal)  
17         return jsonify(result)
```

- Como vimos anteriormente, temos diversos métodos HTTP que mostram como o cliente chama o webservice. Ou seja, sabemos que o cliente vai nos chamar. Neste sentido, precisamos definir qual (ou quais) dos verbos HTTP que vamos atender.
- Especificamente para a conversão, creio que devemos implementar um webservice que atenda o verbo POST, uma vez que ele vai mandar um dado (valor em reais).
- Para isto, o Flask exige a criação de uma classe, conforme podemos ver a seguir a classe `Cotacao`. E, criamos um método cujo nome é o verbo HTTP, neste caso `post`. E, como todo método dentro de classe, tem que ter como primeiro parâmetro o `self`.
- No corpo do método, na linha 15, o trecho `request.json['valor']` resgata o valor informado no campo valor pelo cliente (uma das 3 aplicações). Ainda na linha 15, convertemos este valor para float e na linha seguinte (16), chamamos o nosso método `converterParaDolares`, que faz propriamente a conversão e devolve ao cliente.
- Mas, pera aí, o que tem na linha 17? `jsonify(result)`? Basicamente converte o resultado para um JSON. Mas, pera aí, o que é JSON?

DON'T FORGET

Como vimos anteriormente, os clientes vão chamar o nosso Webservice. E, os clientes não precisam ser implementados em Python, confere? Podem ser em qualquer linguagem. Desta maneira, precisamos que: Sempre que enviar e receber dados do (e para) os webservices, temos que ter uma linguagem que todo mundo entenda, que é a JSON (O inglês da programação).

JSON significa *Javascript Object Notation* (Notação de Objetos em Javascript) e é amplamente utilizado. JSON é bem parecido com dicionários, pois guarda uma série de chaves e valor.

```
20 api.add_resource(Cotacao, '/cotacao')  
21  
22 if __name__ == '__main__':  
23     app.run()
```

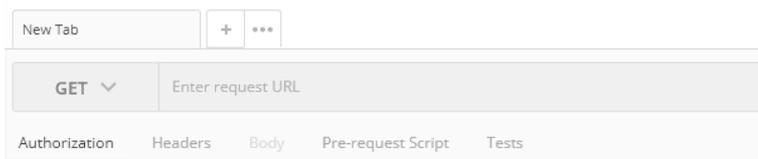
- A linha 20, nos diz que, se você chamar o webservice no endereço `http://127.0.0.1:5000/cotacao`, ele deve chamar a classe `Cotacao`. Agora é só executar, pessoal! Ao executar, espera-se a seguinte saída no console:

```
Running on http://127.0.0.1:5000 (Press CTRL+C to quit)  
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: off
```

- As linhas 22 e 23, pode inserir lá, antes de rodar né 😊

03 Testes

- Agora que implementamos um *webservice*, precisamos testar. E, quando me refiro a testar, me refiro a simular um cliente chamando o Webservice para verificar se a resposta é adequada.
- Para isto, vamos utilizar o **POSTman**, que é uma ferramenta que simula clientes.
- **Passo 1:** Instalar o POSTman
- Acesse o endereço <https://www.postman.com/downloads/>
- **Passo 2:** Criar uma *request* (requisição) que simule um cliente
- Clicamos no botão **NEW**
- Depois clique na opção **Request** para criar a nossa primeira request.
- Na janela que abre, clique em **X**, pois estes passos são desnecessários para o momento. A janela a seguir é apresentada.

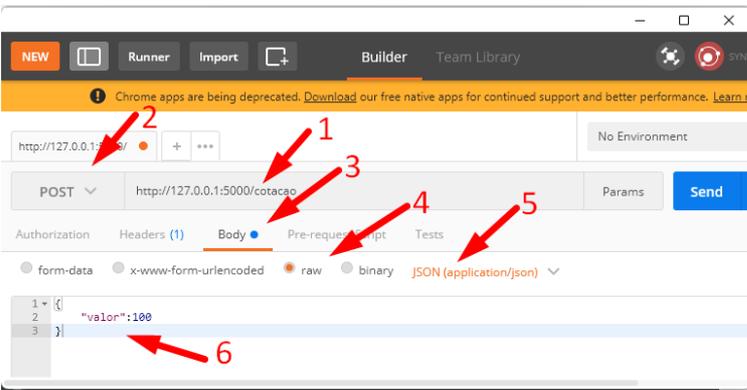


- Primeiro, você deve lembrar qual a URL do seu webservice, cuja formação é apresentada a seguir.

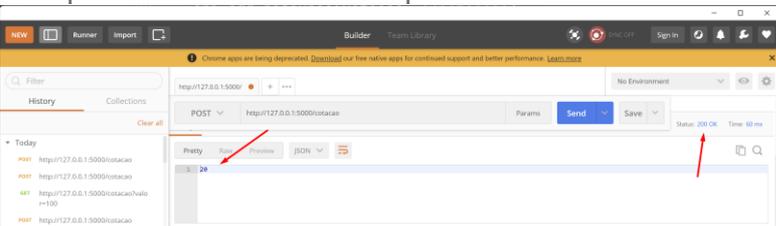
`http://127.0.0.1:5000/cotacao`

```
20 api.add_resource(Cotacao, '/cotacao')  
21  
22 if __name__ == '__main__':  
23     app.run()
```

- **Passo 3:** Realizar a primeira requisição
- Conforme imagem a seguir, devemos preencher a requisição da seguinte maneira:
- **Seta 1:** Colocamos o endereço do Webservice (vimos como fazer no passo 2)
- **Seta 2:** Colocamos o verbo HTTP, lembra que usamos POST? Linha 14 do código
- **Seta 3:** Clicamos em Body (Corpo) para preencher o corpo requisição com alguma informação
- **Seta 4:** Clicamos em raw (crú) para informar que o dados que vamos informar é cru
- **Seta 5:** E o dado raw (crú) é em formato JSON
- **Seta 6:** Escrevemos em formato JSON os dados que o cliente quer enviar para o webservice
 - Na linha 15 do código, recuperamos o valor enviado pelo cliente na Seta 6
- Por fim, clique em **Send**



- Ao clicar no botão **Send**, a tela a seguir é apresentada na qual uma seta indica que a resposta da chamada ao webservice foi o código 200, que significa OK. E, na outra seta, verificamos o conteúdo da resposta. Ou seja, sempre vem o código e, opcionalmente o conteúdo da resposta.



DON'T FORGET! Baseado no código e no conteúdo de resposta (opcional), o cliente (celular, browser ou desktop) apresenta uma tela com o resultado.

04 Implementando um Cliente em Python

- Cliente, como informamos anteriormente, é quem chama o webservice, neste caso, pode ser um aplicativo para celular, desktop ou web.
- Agora, vamos criar um exemplo de um código em Python de uma aplicativo que chama o nosso webservice.

- **Passo 1:** Crie um projeto novo no Pycharm
- **Passo 2:** Instale as dependências necessárias por meio do comando a seguir: `python -m pip install requests`
- **Passo 3:** Chame o webservice da maneira apresentada a seguir.

```

1 import requests
2
3 enderecoWebservice = "http://127.0.0.1:5000/cotacao"
4 conteudo = {"valor": 100}
5 resposta = requests.post(enderecoWebservice, json=conteudo)
6 resposta.json()

```

- Linha 1: Importamos a dependência necessária
- Linha 3: Uma string com o endereço do webservice
- Linha 4: Um JSON com um campo valor com valor 100
- Linha 5: Chamamos um POST para o endereço da linha 3, passando o valor em JSON da linha 4
- Linha 6: Obteremos o resultado da chamada

05 Hospedar o Webservice

- Até o momento, rodamos o Webservice em nossa máquina, por isso o endereço continha 127.0.0.1, que é o endereço da máquina local. Que tal implantar em um servidor? Você consegue!
 - <https://www.geeksforgeeks.org/deploy-python-flask-app-on-heroku/>
 - <https://medium.com/data-hackers/deploy-flask-api-using-heroku-d5c7128481b7>

06º que tem mais?

- Apresentamos a você, em pouco tempo, como criar um webservice. Entretanto, é muito interessante que você tenha a base teórica necessária para aplicar o conhecimento aqui adquirido em ambientes mais complexos.
- Neste sentido, sugere-se:
 - O aprofundamento sobre o protocolo HTTP
 - Entendimento completo do padrão REST
 - Aplicar os outros verbos HTTP no Flask
 - Implantar o webservice em um servidor como Heroku
- Por fim, na próxima *cheat sheet*, vou apresentar um exemplo de webservice com as principais operações e, inclusive, salvando no banco de dados.
 - Bons estudos!



Daniel Abella

Profissional PMP, PMI-ACP e CSM, Gerente de Projetos no Virtus & UniFacisa e Professor do Curso de Sistemas de Informação da UniFacisa. Autor dos livros Gestão A3 (Ágil) e Scrum Arretado.

Conteúdo:

- O que é CRUD?
- Operações CRUD no BD
- Criando WebServices com CRUD
- E agora?

Python

CRUDs e WebServices com Python

Usando Flask

01 O que é CRUD?

- Um dos termos bastante utilizados na área de Desenvolvimento de *Software* é CRUD. Mas o que é CRUD?
- Antes de “traduzir” o que significa, observe a tela abaixo.
 - Basicamente, temos uma listagem de usuários (são 5 linhas, uma com Pablo e outras 4 linhas)
 - Em cada uma das linhas, temos um botão **Edit** que altera os dados de um usuário e um botão **Delete** que exclui o usuário.

Add Record Search Username OK

ID ▲	Username	Fullname	Date		
1	Pablo	Pablo Picasso	1881-1973	Edit	Delete
2	Johannes	Johannes Vermeer	1632-1675	Edit	Delete
3	Claude	Claude Monet	1840-1926	Edit	Delete
4	Rembrandt	Rembrandt van Rijn	1606-1669	Edit	Delete
10	Leonardo	Leonardo da Vinci	1452-1519	Edit	Delete

Previous 1 2 Next

- E, caso seja necessário adicionar um novo usuário, clicamos no botão **Add Record** e a tela a seguir é apresentada. Nesta segunda tela, preenchemos os dados de um novo usuário e ao clicar no botão **Save**, que um novo usuário é criado e voltamos a tela anterior que, ao invés de 5 usuários, agora terá 6.

Save Cancel

ID	1
Username	Pablo
Fullname	Pablo Picasso
Date	1881-1973
Active	<input checked="" type="checkbox"/>
Country	Spain
Role	admin

- Esta funcionalidade inteira (listar, inserir, excluir e editar), chamamos de CRUD. Ou seja, o que acabamos de explicar foi a funcionalidade de CRUD de usuários.
- Mas, afinal, o que é CRUD? CRUD é um acrônimo para C (Create, criar, inserir), R (read, ler), U (Update, atualizar) e D (Delete, Excluir).



02 Operações CRUD no BD

- O pré-requisito é termos o Python e o PIP instalados, após isto, execute o seguinte comando no terminal do PyCharm:
 - `python3 -m pip install mysql-connector-python`
- Para isso, no PyCharm, clique em **Terminal** e informe o comando acima, como apresentado a seguir.

```

PS C:\Users\teste\config\PycharmProjects\pythonProject1
> python3 -m pip install mysql-connector-python
Collecting mysql-connector-python
  Downloading mysql_connector_python-8.0.29-cp310-cp310-win_amd64.whl (7.7 MB)
Collecting protobuf>=3.0.0
  Downloading protobuf-3.20.1-cp310-cp310-win_amd64.whl

```

- O próximo passo (e bem sugerido) é criar um módulo para guardar as operações no banco de dados. Criaremos o arquivo `operacoesbd.py` com os seguintes métodos:
 - Método para abrir uma conexão no BD.

```

1 import mysql.connector
2
3 def abrirBancoDados(host,user,password,database):
4     return mysql.connector.connect(host,user,
5                                     password,database)

```

- Do mesmo jeito que abrimos uma conexão, precisamos encerrá-la após o uso com o método a seguir.

```
7 def encerrarBancoDados(connection):
8     connection.close()
```

- Note que, no método que abre a conexão, este retorna uma conexão. Enquanto no método que encerra o banco de dados, recebe a conexão como parâmetro e a encerra.
- Na sequência, abaixo temos o método que realiza inserção de dados no banco de dados. Note que, precisamos especificar a conexão, a consulta (SQL) e os dados a serem inseridos.

```
10 def insertNoBancoDados(connection, sql, dados):
11     cursor = connection.cursor()
12     cursor.execute(sql, dados)
13     connection.commit()
14     id = cursor.lastrowid
15     cursor.close()
16     return id
```

- Para que o exemplo funcione, vamos criar uma tabela chamada produtos

```
1 create table produtos (
2     id int not null auto_increment primary key,
3     nome varchar(100) not null,
4     descricao varchar(100) not null,
5     preco float not null
6 );
```

- Uma vez criada a tabela e o método acima, podemos usá-los dentro do nosso main.py, conforme exemplo a seguir.
 - Linha 1: Import do módulo criado antes
 - Linha 3: Criamos a conexão a ser usada
 - Linha 5: Informamos a consulta de INSERT
 - Linha 6: Informamos os dados
 - Linha 7: Chamamos o SQL e os dados usando o método criado
 - Linha 9: Encerramos a conexão criada na linha 3

```
1 import operacoesbd
2
3 conn = operacoesbd.abrirBancoDados('localhost',
4                                     'root','12345','testes')
5 sql = "INSERT INTO produtos(nome,descricao,preco) VALUES (%s, %s, %s)"
6 dados = ('Banana','Bana Pacovan',8)
7 operacoesbd.insertNoBancoDados(conn,sql,dados)
8
9 operacoesbd.encerrarBancoDados(conn)
```

- No método a seguir, apresentamos todos os registros no banco de dados.
 - Devemos informar a conexão e o SQL da consulta

```
18 def listarBancoDados(connection, sql):
19     cursor = connection.cursor()
20     cursor.execute(sql)
21     results = cursor.fetchall()
22     cursor.close()
23     return results
```

- Como faço para chamar o método acima? O exemplo está a seguir.
 - Linha 5: Consulta SQL
 - Linha 6: Chamamos o método criado informando a conexão e o SQL (linha 5)
 - Pegamos o resultado e atribuímos a variável resultado
 - Linha 7: Imprimimos o resultado
 - Linha 11: Encerramos a conexão da linha 3

```
1 import operacoesbd
2
3 conn = operacoesbd.abrirBancoDados('localhost',
4                                     'root','12345','testes')
5 sql = "SELECT * FROM PRODUTOS"
6 resultado = operacoesbd.listarBancoDados(conn,sql)
7
8 for elemento in resultado:
9     print(elemento)
10
11 operacoesbd.encerrarBancoDados(conn)
```

- Seguindo, agora vamos fazer a atualização do BD com o método a seguir.
- Note que, este método retorna a quantidade de linhas que foram atualizadas por meio da consulta.

```
25 def atualizarBancoDados(connection, sql, dados):
26     cursor = connection.cursor()
27     cursor.execute(sql, dados)
28     connection.commit()
29     linhasAfetadas = cursor.rowcount
30     cursor.close()
31     return linhasAfetadas
```

- Complementarmente, o código a seguir realiza o uso do método acima.
 - Linha 7: Chamamos o método criado informando a consulta (linha 5) e os dados (linha 6)

```
1 import operacoesbd
2
3 conn = operacoesbd.abrirBancoDados('localhost',
4                                     'root','12345','testes')
5 sql = "UPDATE produtos SET nome = %s WHERE id = %s"
6 dados = ('Banana de Cozinhar',1)
7 linhasAfetadas = operacoesbd.atualizarBancoDados(conn,sql,dados)
8 print('Linhas afetadas: ',linhasAfetadas)
9
10 operacoesbd.encerrarBancoDados(conn)
```

- Por fim, vamos excluir dados do BD, né? O método a seguir faz isto. Note que, é bem parecido com o método de atualizar.

```

33 def excluirBancoDados(connection,sql,dados):
34     cursor = connection.cursor()
35     cursor.execute(sql, dados)
36     connection.commit()
37     linhasAfetadas = cursor.rowcount
38     cursor.close()
39     return linhasAfetadas

```

- Complementarmente, o código a seguir mostra como chamar o método acima.

```

1 import operacoesbd
2
3 conn = operacoesbd.abrirBancoDados('localhost',
4                                     'root','12345','testes')
5 sql = "DELETE FROM produtos WHERE id = %s"
6 data = (2,)
7 linhasAfetadas = operacoesbd.excluirBancoDados(conn,sql,dados)
8 print('Linhas afetadas: ',linhasAfetadas)
9
10 operacoesbd.encerrarBancoDados(conn)

```

03 Criando WeServices que Façam as Operações CRUD

- Na seção anterior, lembram que criamos um módulo com as operações de um banco de dados? Enfim, agora nesta seção, vamos criar o nosso webservice que usa este módulo e possui as seguintes operações:
 - **Linha 10:** Método que recebe requisições GET
 - Como todo método dentro de classe, o primeiro parâmetro é um self
 - Note que, na linha 13 colocamos a consulta, que é enviada na linha 14. Na linha 15 encerramos a conexão.
 - Por fim, na linha 16, convertemos o resultado da linha 14 em formato JSON
 - **Linha 18:** Método que recebe requisições POST
 - Linhas 21,22 e 23: Pegamos os parâmetros enviados no body (corpo) da requisição
 - Na linha 25 temos a consulta, que é enviada na linha 27.
 - **Os dois métodos GET e POST (linhas 10 e 17) estão dentro de uma classe chamada Produtos**
 - Usamos esta classe para encapsular os 2 métodos
 - Posteriormente, na linha 62, informamos: “pegue os métodos da classe Produtos e faça eles atenderem a requisições em /produtos”
 - De maneira análoga, na linha 43 criamos uma classe para abrigar métodos que recebem parâmetros por URL (tipo, GET ou DELETE passando o ID)

```

1 from flask import Flask, request, jsonify
2 from flask_restful import Resource, Api
3 import operacoesbd
4
5 app = Flask(__name__)
6 api = Api(app)
7
8 class Produtos(Resource):
9
10     def get(self):
11         conn = operacoesbd.abrirBancoDados('localhost',
12                                             'root', '12345', 'testes')
13         sql = 'SELECT * FROM PRODUTOS'
14         resultado = operacoesbd.listarBancoDados(conn,sql)
15         operacoesbd.encerrarBancoDados(conn)
16         return jsonify(resultado)
17
18     def post(self):
19         conn = operacoesbd.abrirBancoDados('localhost',
20                                             'root', '12345', 'testes')
21         nome = request.json['nome']
22         descricao = request.json['descricao']
23         preco = request.json['preco']
24
25         sql = 'INSERT INTO produtos(nome,descricao,preco) values (%s,%s,%s)'
26         dados = (nome,descricao,preco)
27         operacoesbd.insertNoBancoDados(conn,sql,dados)
28         operacoesbd.encerrarBancoDados(conn)
29         return {"status": "success"}
30
31 class ProdutosPorId(Resource):
32
33     def delete(self, id):
34         conn = operacoesbd.abrirBancoDados('localhost',
35                                             'root', '12345', 'testes')
36         sql = 'DELETE PRODUTOS where id = %s'
37         dados = (id)
38         operacoesbd.excluirBancoDados(conn,sql,dados)
39         operacoesbd.encerrarBancoDados(conn)
40         return {"status": "success"}
41
42     def get(self, id):
43         conn = operacoesbd.abrirBancoDados('localhost',
44                                             'root', '12345', 'testes')
45         sql = 'SELECT * FROM PRODUTOS where id = ' + id
46         resultado = operacoesbd.listarBancoDados(conn,sql)[0]
47         operacoesbd.encerrarBancoDados(conn)
48         return jsonify(resultado)
49
50 api.add_resource(Produtos, '/produtos')
51 api.add_resource(ProdutosPorId, '/produtos/<id>')
52
53 if __name__ == '__main__':
54     app.run()

```

- Por fim, os métodos get (por ID) e delete (por ID) estão nas linhas 45 e 54. Na linha 63, adicionamos esta nova classe, que vai atender a requisições em '/produtos/<id>'.

04 E agora? Acabou?

- Pior que sim 😞 Entretanto você pode seguir estudando, concordas?
- Uma recomendação final é que leiam o [PEP 8 - Style Guide for Python Code](#), pois foi um assunto negligenciado durante esta série, pois o objetivo era “startar” sua carreira de desenvolvedor usando a linguagem Python.
- Espero que esta série de conteúdos tenham sido de bastante utilidade para você!
- Abraços cordiais, Daniel Abella.



- Conceitos iniciais
- Instalação do Git
- Repositórios
- Repositório Local
- Repositório Remoto
- Enviando Códigos
- Descobrimo o que falta enviar
- Recebendo Códigos
- Lista de Comandos
- Como criar um bom README.md



Controle de Versão

Configuração, Utilização e Principais Comandos

01 Conceitos

- Vamos desenvolver um *software* de *e-Commerce* com uma equipe de 6 desenvolvedores; Como todos estarão trabalhando simultaneamente no desenvolvimento do projeto, como trabalhar em conjunto?
- Precisamos de um Sistema de Controle de Versão!
 - Atualmente, a melhor ferramenta, adotada pela ampla maioria das empresas e desenvolvedores é o **Git**.

02 Instalação do Git

- O primeiro passo é instalar o Git na sua estação de trabalho. Para isto, acesse o endereço <https://git-scm.com/downloads>
 - Existe versões para Windows, Linux e MacOS
- Uma vez instalado, acesse o *prompt* de comando e digite o comando `git --version`:

```
Command Prompt
(c) Microsoft Corporation. All rights reserved.
C:\Users\teste config>git --version
git version 2.35.1.windows.2
C:\Users\teste config>
```

- Funcionou? Agora precisa configurar o seu nome e e-mail utilizando os dois comandos a seguir.
 - Por que isto é necessário? Todas as vezes que você enviar um código para o repositório, seu nome e e-mails vão ser registrados.
 - Essa configuração é realizada uma única vez.

```
git config --global user.name "Daniel Abella"
git config --global user.email daniel@daniel-abella.com
```

- Caso você necessite saber quais as configurações de nome e e-mail estão vigentes no computador, basta executar o comando apresentado a seguir.

```
git config --list
```

03 Repositórios e Tipos

- Um repositório contém todos os arquivos do seu projeto, bem como o histórico de revisão de cada arquivo.
- Existem dois tipos de repositórios, repositório local e repositório remoto (*remote*).
 - Na seção 04 descreveremos como utilizar um repositório *remoto*, utilizando o Github, que é um dos serviços gratuitos para repositórios com Git.
 - Na seção 05 descreveremos como utilizar um repositório local, utilizando a sua própria máquina como repositório.

04 Repositório Local

- Um repositório local é basicamente uma pasta onde todos os seus arquivos estarão guardados e versionados pelo Git
 - A palavra *versionados*, significa que, toda a evolução (isto é, o histórico) do artefato.
 - Para criar um novo repositório, o comando é o `git init`
 - O comando `git init` cria um novo repositório do Git. Ele pode ser usado para converter um projeto existente e não versionado em um repositório do Git ou inicializar um novo repositório vazio. máquina como repositório.

```
Command Prompt
C:\Users\teste config\Downloads\curso-git>git init
Initialized empty Git repository in C:/Users/teste config/Downloads/curso-git/.git/
C:\Users\teste config\Downloads\curso-git>
```

- No exemplo acima, criamos um repositório dentro da pasta chamada `curso-git`. Ou seja, antes de criar o repositório, certifique-se em qual pasta você está.

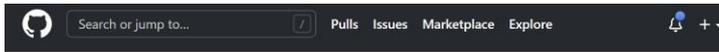
05 Repositório Remoto (GitHub)

- O GitHub é um dos repositórios Git remotos mais utilizados no mundo.
 - Quando uso o termo *remoto*, refiro-me ao fato que, o código fonte do seu projeto estará protegido e armazenado em um servidor remoto.
 - Concorrentes ao GitHub, temos o GitLab e BitBucket.

05 Repositório Remoto (GitHub)

(Continuação)

- Para utilizarmos o GitHub, acesse o endereço www.github.com e, o primeiro passo é criar uma conta por meio do botão **Sign up**. Caso já tenha conta, clique em **Sign in**
- O segundo passo é criar um repositório remoto. Para isto, clique no botão **New**, que a tela a seguir será apresentada.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

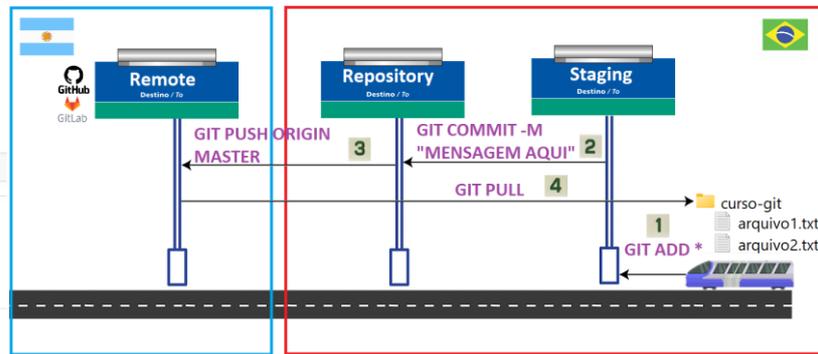
Owner * / Repository name *

Great repository names are short and memorable. Need inspiration? How about didactic-robot?

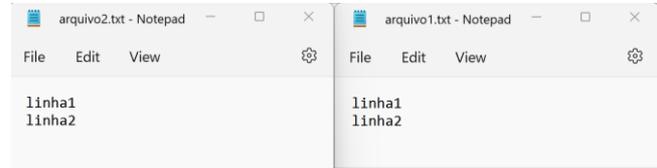
Description (optional)

Public
Anyone on the internet can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.



- Agora, para aprender como enviar seus códigos, vamos fazer uma simulação. Crie, dentro da pasta `curso-git` dois arquivos (`arquivo1.txt` e `arquivo2.txt`). Para cada um dos arquivos, vamos criar duas linhas de texto (linha1 e linha 2), conforme exemplificado na imagem a seguir.



- Como enviar nossos arquivos para o repositório do Github?
- Os passos necessários estão representados na imagem a seguir.

- Na tela acima, temos que preencher os seguintes campos:
 - **Repository Name:** nome do repositório
 - **Description:** Descrição do repositório (é opcional)
 - **Tipo de Repositório:** *Public* (Público) ou *Private* (Privado)
 - Um repositório *Public*, todo mundo pode visualizar o seu código fonte, porém apenas as pessoas autorizadas por você que podem contribuir
 - Um repositório *Private*, apenas pessoas autorizadas podem visualizar o código fonte, bem como contribuir
- Ao fim, clique no botão **Create repository** e seu repositório estará criado na nuvem! Agora, para passar a trabalhar neste repositório, acesse o prompt de comando e digite os 2 comandos a seguir:

Comandos

```
git clone https://github.com/daniel-abella/curso-git.git
cd curso-git
```

- `daniel-abella` se refere ao meu usuário no Github
- `curso-git` se refere ao meu repositório criado no Github
- Agora, na pasta `curso-git` você terá uma cópia do repositório que está no Github. E, oportunamente, você e seus colegas podem enviar seus códigos para o seu repositório do Github.

06 Enviando Códigos

- Se você chegou aqui, você criou seu repositório de uma das duas formas: repositório local (etapa 4) ou repositório no Github (etapa 5). Agora, apresentaremos como enviar seus códigos, bem como receber código dos seus colegas.

- A analogia acima se baseia em uma viagem de metrô, na qual temos 3 paradas, descritas a seguir.
- Lembram que criamos 2 arquivos anteriormente na pasta local `curso-git`? O primeiro passo é dizer quem você quer levar no metrô, ou seja, informar qual dos 2 arquivos eu quero que sejam armazenados no repositório.
 - Para isso, pode-se fazer algum dos comandos a seguir. O primeiro deles (com *), sinaliza que **todos** os arquivos novos ou modificados serão enviados ao repositório. Por outro lado, o segundo comando (com `arquivo1.txt`) sinaliza que, apenas o `arquivo1.txt` será enviado ao repositório.

Comandos para Sinalizar Códigos que Quero Enviar

```
git add *
git add arquivo1.txt
```

- O segundo passo é armazenar localmente o(s) arquivo(s) no servidor local. Para isto, executa-se o seguinte comando. A mensagem que está entre aspas detalha a sua contribuição e é muito importante que esteja bem escrita.

Comandos para Enviar para o Servidor LOCAL

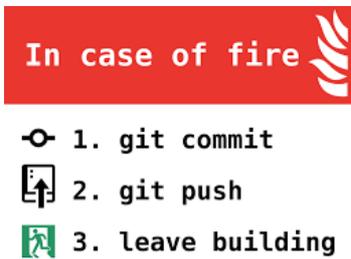
```
git commit -m "Enviando o arquivo1.txt"
```

- Agora que o arquivo está localmente, se necessário, você pode remeter ao servidor remoto (como Github), para isto, execute o comando a seguir.

Comandos para Enviar para o Servidor REMOTO

```
git push
```

- Como discutimos anteriormente, o comando `git commit -m "Mensagem"` envia os códigos marcados no `git add` para o servidor LOCAL. Enquanto que, o `git push` envia ao servidor REMOTO.
 - Posso enviar direto para o servidor REMOTO? Não, precisa enviar localmente (`git commit`) e depois remotamente (`git push`)
- Imagine-se em um incêndio e você precisa enviar o seu código para um servidor externo, porque em alguns minutos sua máquina pegará fogo.
- Quais os comandos necessários? Na imagem a seguir, temos um excelente exemplo de como proceder.



- Com base na imagem acima, inicialmente você faz um `git commit` para enviar o seu código para um repositório local (presume-se que você tenha feito o `git add` antes)
- Agora que enviado localmente, você pode enviar remotamente por meio do comando `git push`.
- Como o código já está nas nuvens (foi enviado para o servidor remoto), agora podemos deixar o prédio (passo 3).

07 Descobrindo o que falta enviar

- Como vimos, no comando `git add *` ou `git add arquivo1.txt` sinalizamos ao git o(s) arquivo(s) que queremos remeter ao servidor local e possivelmente ao servidor remoto.
- Entretanto, na maioria das vezes não saberemos quais o(s) arquivo(s) alteramos e estão passíveis de serem enviados. Para conhecer o(s) arquivo(s) nesta situação, execute o comando a seguir.

Comandos p/ Descobrir o que Pode Enviar ao Servidor

`git status`

08 Recebendo Códigos

- Agora vamos fazer o inverso da seção 6. Imagine que, outros colegas de trabalho tenham enviado código ao repositório remoto e você queira receber estas alterações. O comando é o apresentado a seguir.

Comandos p/ Receber Novidades do Servidor Remoto

`git pull`

09 Lista dos Comandos

Comando

Descrição

`git clone https://github.com/daniel-abella/curso-git.git`

Clona tudo que está no servidor remoto para minha máquina, onde: **daniel-abella** se refere ao meu usuário no Github e **curso-git** se refere ao meu repositório criado no Github

`git status`

O que falta enviar para o repositório local

`git add *`

Sinaliza que todos os arquivos novos ou modificados devem ser enviados ao repositório local

`git add arquivo1.txt`

Sinaliza que apenas o arquivo1.txt deve ser enviado ao repositório local, demais arquivos são ignorados

`git commit -m "Msg"`

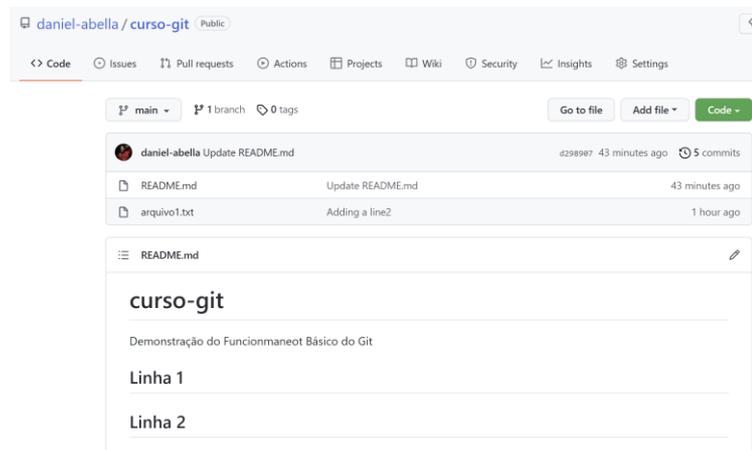
Envia para o repositório local os arquivos sinalizados e enviado a "Msg" como mensagem descrevendo

`git push`

Envia para o repositório remoto o que você tinha enviado localmente

10 Como criar um bom README.md

- Ao criar o repositório, sugere-se criar automaticamente o arquivo README.MD, que é um arquivo que é aberto ao acessarem o repositório. Este arquivo é como se fosse a porta de entrada do repositório e deve ser apresentável e explicar bem o projeto que está armazenado no repositório.



- O arquivo README.MD é escrito em uma linguagem chamada Markdown (por isso a extensão do arquivo é MD). Caso você queira usar um editor para criar um README bonito, acesse o endereço <https://dillinger.io/>