
MARATONA PROFISSÃO PROGRAMADOR PYTHON



ByLearn

Seja Bem-Vindo!



Zarcky

Graduado em Ciência da Computação e Mestre em
Computação Aplicada – FFCLRP – USP.

RESUMO DA MARATONA



PYTHON

Variáveis

Uma variável é um *identificador* que se *refere a um valor*

Tipos de Dados

Cada *dado* irá possuir um *tipo*.

É isso que definirá se ele é um *número, texto, booleano...*

PYTHON

Métodos de Entrada

São formas de
enviar dados do
usuário ao
Python.

Métodos de Saída

São formas do
Python de
mostrar dados ao
usuário.

PYTHON

Dados Sequenciais

Para trabalharmos com valores sequenciais *podemos usar listas*

Elementos em Listas

Quando queremos usar um elemento da lista, *devemos utilizar o seu índice.*

Índices x Posições

Posições iniciam em 1. Já os índices, iniciam em 0.

Basicamente:
Índice = Posição – 1

PYTHON

Condições

Formas de realizar
tomadas lógicas
de decisão.

Se... Então
Senão, se... Então
Senão... Então

Operadores Lógicos

Podemos usar
operadores lógicos
para criar
condicionais
compostas.

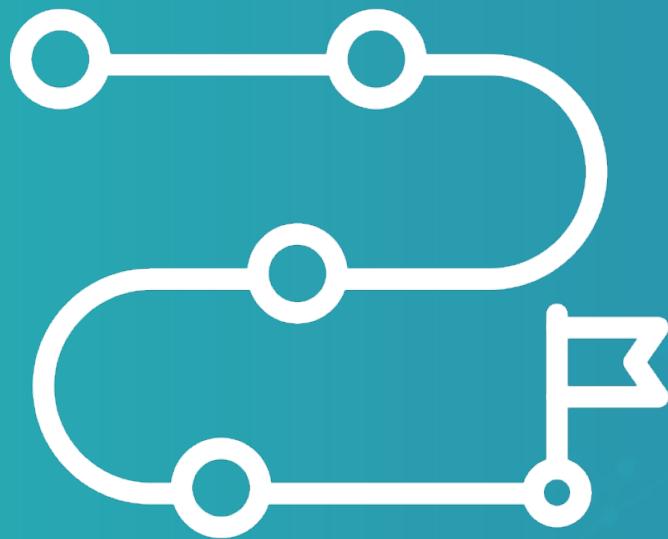
PYTHON

Laços de Repetição

Excelente forma
de *evitar*
repetição de
código.

Repetições Dinâmicas

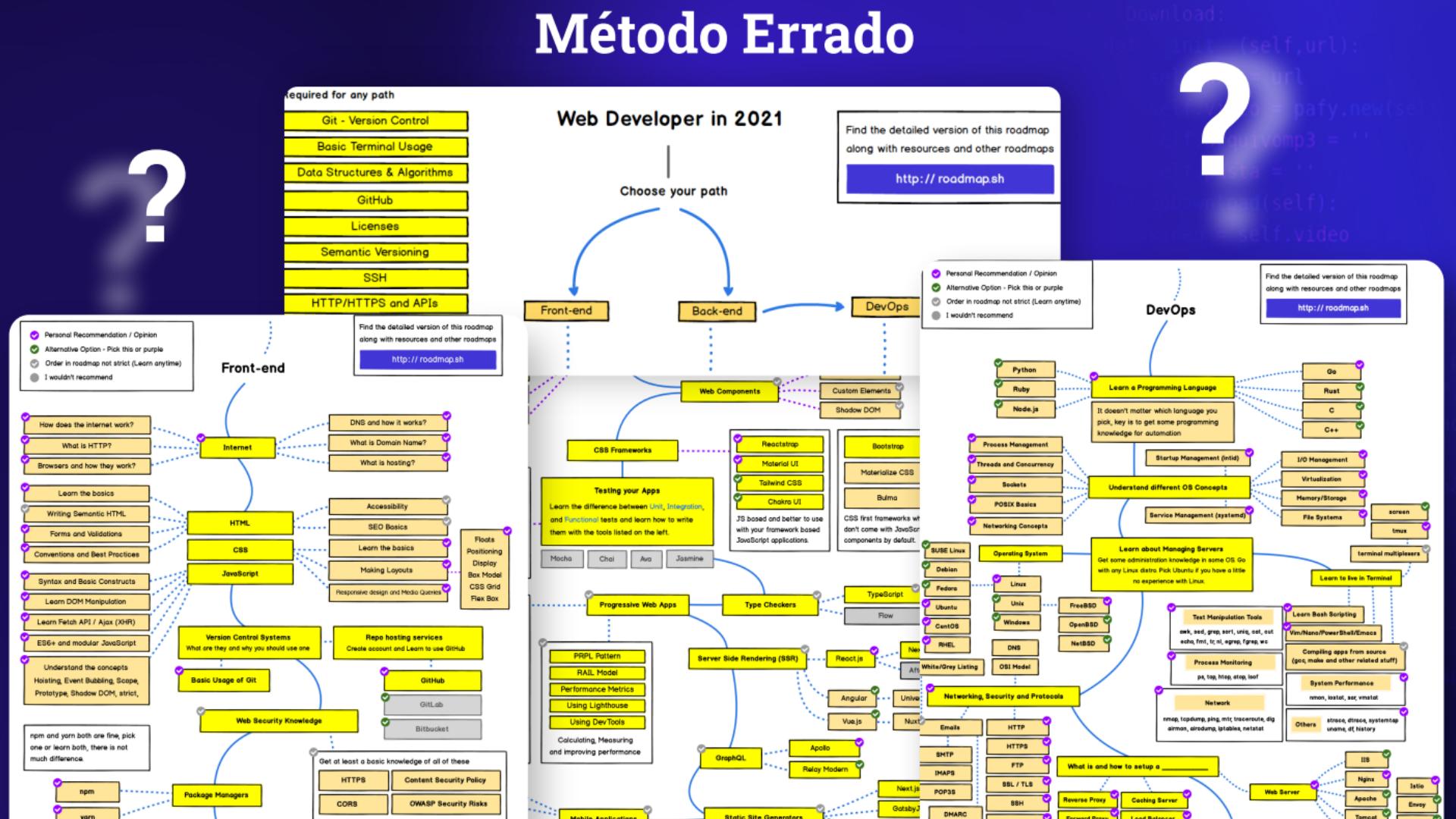
Podemos pegar o
tamanho das
sequências ou usar
a *própria variável*
nos *laços de*
repetição.



A TRILHA DO PROGRAMADOR



Método Errado



Método 3x1



mentor



linguagem de
alto nível



didática simples
de entender

Método Correto





CRONOGRAMA DA MARATONA



C R O N O G R A M A

O QUE É PROGRAMAÇÃO E QUAIS SÃO AS
OPORTUNIDADES TRABALHANDO COM O PYTHON



INICIANDO NA PROGRAMAÇÃO COM PYTHON:
CRIANDO NOSSO PRIMEIRO PROGRAMA



AVANÇANDO NA PROGRAMAÇÃO:
CRIANDO LÓGICA PARA SOFTWARES



C R O N O G R A M A

RUMO AO PROGRAMADOR FAIXA PRETA

PROJETO PRÁTICO + SEGREDO PESSOAL:
COMO GANHAR 5X MAIS EM OUTRA MOEDA

CONSTRUINDO UMA SOLUÇÃO 100%
PRÁTICA E REALISTA AO MERCADO



VAGAS ABERTAS PARA A MENTORIA



ByLearn

Funções

Outra forma de evitar repetição de código é através de funções, onde definimos uma rotina para ser executada.

Por exemplo: Definir o que fazer para verificar se um aluno foi aprovado.

Para definirmos uma função usamos o *def*.

Funções podem *retornar valores*.

Podemos *enviar valores* para funções.

Funções

Outra forma de evitar repetição de código é através de funções, onde definimos uma rotina para ser executada.

Por exemplo: Definir o que fazer para verificar se um aluno foi aprovado.

Para definirmos uma função usamos o *def*.

Funções podem *retornar valores*.

Podemos *enviar valores* para funções.

```
# Definimos a função
def mostrar_nome_felipe():
    print('Felipe')

# Função com Parâmetro
def mostrar_nome(nome):
    print(nome)

# Função com retorno
def proximo_numero(numero):
    return numero + 1

# Chamando as funções:
mostrar_nome_felipe()
mostrar_nome('ByLearner')
cinco = proximo_numero(4)
```

```
● ● ●

# Definimos a função da média
def calcular_media(nota1, nota2):
    soma = nota1 + nota2
    media = soma / 2

    return media

# Definimos a função de Aprovacão
def verificar_aprovacao(media):

    if media >= 6:
        print('O Aluno foi aprovado')
    else:
        print('O Aluno foi reprovado')

# Chamamos a função Calcular Média
# Enviamos as notas 7.75 e 4.5
# Pegamos o valor do seu retorno...
# ... e colocamos em media_aluno
media_aluno = calcular_media(7.75, 4.5)

# Chamamos a função Verificar Aprovação
# Enviamos a média calculada acima
verificar_aprovacao(media_aluno)
```

```
horario = 'manhã'
clima = 'ensolarado'
temperatura = 'quente'

if horario == 'manhã' or horario == 'tarde':
    if clima == 'ensolarado' and temperatura == 'quente':
        print("Uma piscina cairia bem")

    if (clima == 'ensolarado' or clima == 'nublado') and (temperatura == 'amena' or temperatura == 'fria'):
        print("Seria legal praticar algum esporte")

    if clima == 'chuvisco':
        print("Aproveite para treinar seu Python")
else:
    if clima == 'chuvisco':
        print("Que tal um filme, série ou jogatina?")
    else:
        print("Um jantar fora parece interessante...")
```

```
nome = input("Insira seu nome: ")

nota1 = float(input("Sua primeira nota: "))
nota2 = float(input("Sua segunda nota: "))

print("Olá", nome, "suas notas foram:")
print(nota1, "e", nota2)

soma = nota1 + nota2 media = soma / 2

print("Sua média é", media)
```

Saída:

*Olá Felipe suas notas foram:
8.25 e 7.75
Sua média é 8.0*

While

Além do laço de repetição **FOR**, nós temos o **WHILE**.

No **FOR** nós sabemos quando a repetição vai parar, mas no **WHILE** não.

Só o que temos é uma *condição de parada*.

Por exemplo, no caso ao lado vamos parar de executar o código quando o usuário for o “*Fulano*”, mas *não sabemos quando é que o Fulano vai estar usando o código*.



```
nome = ""  
  
while nome != "Fulano":  
    nome = input("Quem é você? ")  
  
print("Seja bem vindo, Fulano!")
```

```
numero_secreto = 20

print("Eu duvido você acertar meu número secreto em 10 tentativas")

tentativas = 0

numero_escolhido = 0

while numero_escolhido != 20:
    numero_escolhido = int(input("Escolha um número: "))

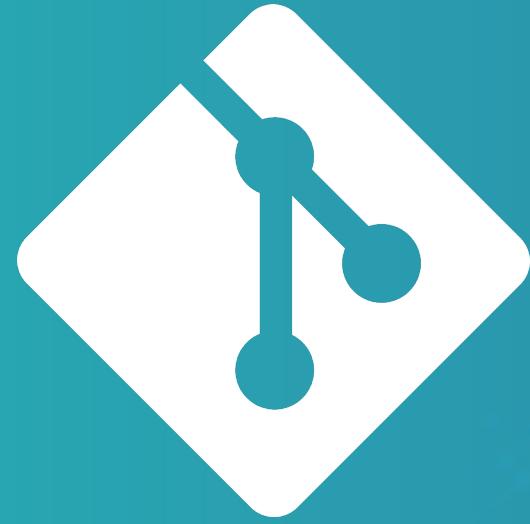
    tentativas = tentativas + 1

    if numero_escolhido != 20 and tentativas == 3:
        print("Suas tentativas acabaram")
        break

if numero_escolhido == 20:
    print("Parabéns, você acertou")
```

VAMOS PRA AULA!



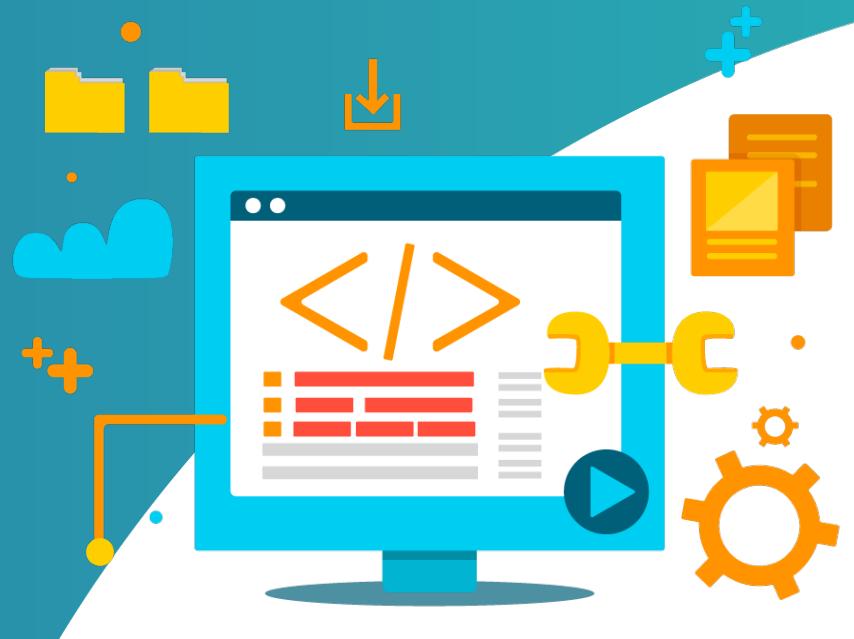


GIT



GIT

- É o *controlador de versões de código* mais utilizado no mundo!
- Basicamente, no lugar de “*projeto_v1*”, “*projeto_v2*”, “*projeto_v3*”...
- Nós usamos o *Git* para manter um projeto (*repositório*) só.
- Todas as *versões* ficam em *histórico* *estudar*.



GIT

Tecnologia que permite o controle de versões de código.

GITHUB

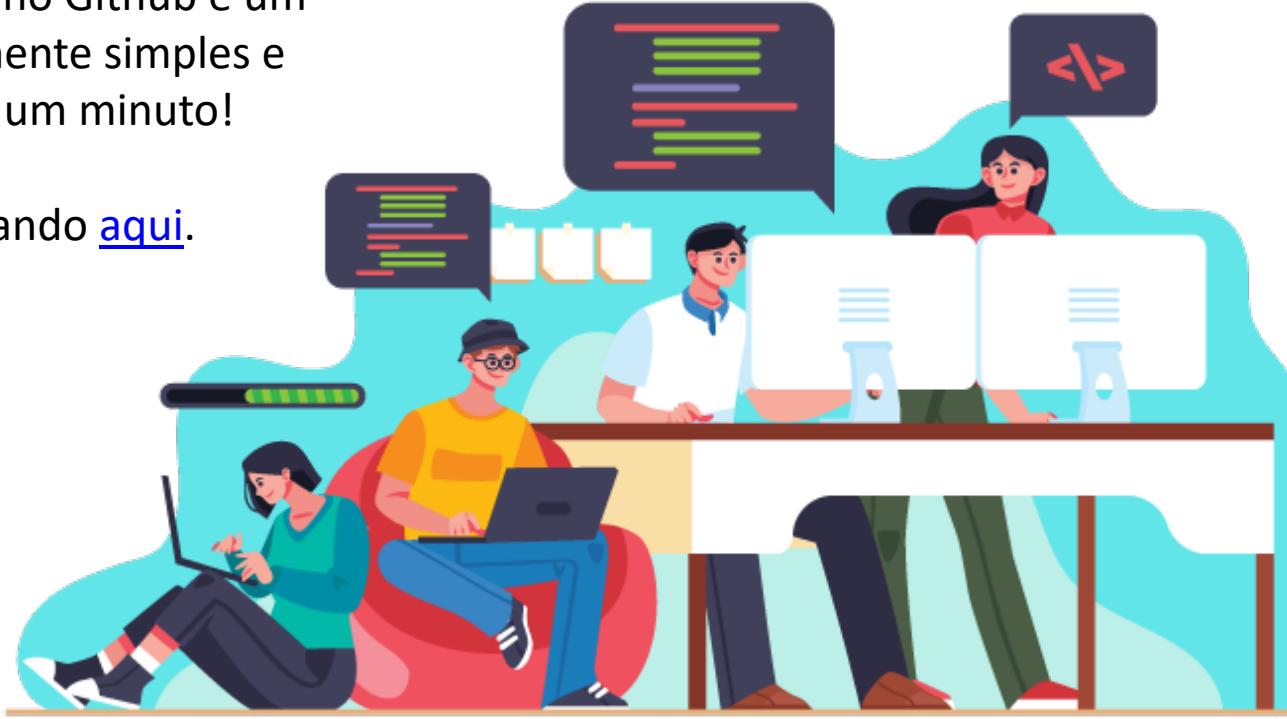
É uma plataforma (serviço) que permite o uso do GIT.



CRIANDO UM REPOSITÓRIO

Criar um repositório no Github é um processo extremamente simples e que não demora um minuto!

Saiba mais clicando [aqui](#).



INTERFACES GRÁFICAS



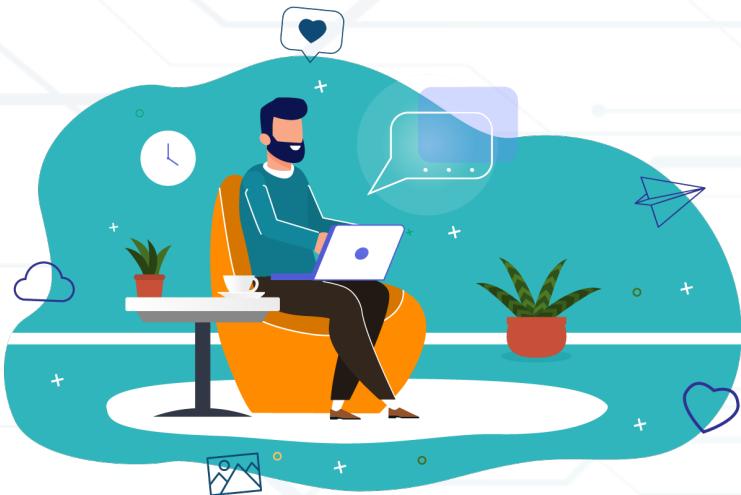
Interfaces Gráficas

Um tema muito pertinente para quem começa a programar através de linhas de comando é a questão de interfaces gráficas.

Será que é possível ter algum visual gráfico na minha aplicação?

Sempre terei que usar essas linhas de comando sem usabilidade nenhuma?

Existem aplicações modernas sem GUI, apenas CLI?

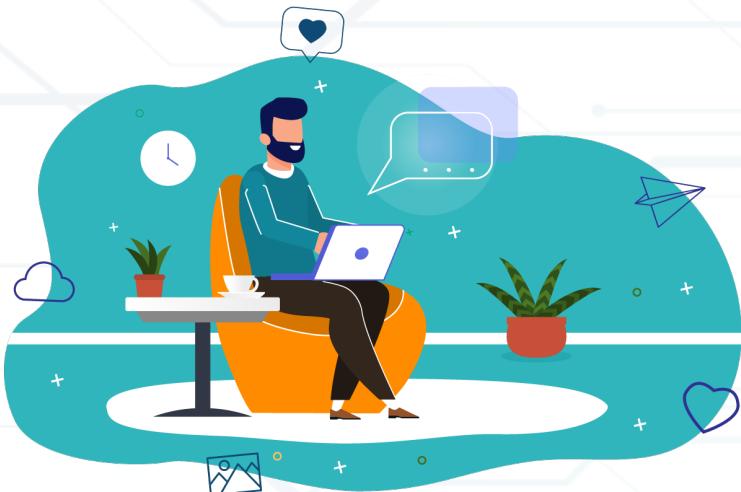


Interfaces Gráficas

Essas perguntas e ansiedade em aprender a desenvolver *GUIs* tem um bom motivo por trás!

Embora alguns scripts e automatizadores atuais não precisem de interface gráfica para realizar suas funções...

grande parte das aplicações requerem uma melhor usabilidade, muitas vezes alcançada apenas por interfaces gráficas.



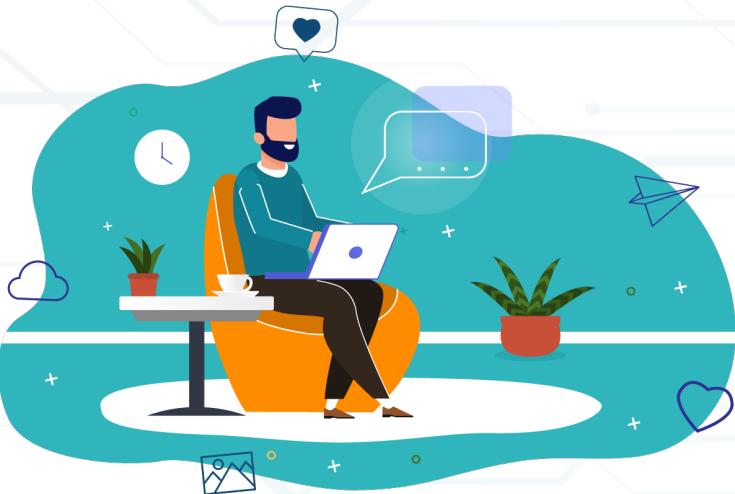
Interfaces Gráficas

Com todo o poder e flexibilidade oferecidos pelo Python, é evidente que surgiriam *inúmeras opções para a criação de interfaces gráficas.*

Temos desde opções mais básicas e simples, como o *PySimpleGUI* a até soluções mais completas e complexas, como o *PyQT5*, que possui até mesmo um “*GUI Builder*”.

Porém, algo é comum entre todas as opções:

- Todas são fáceis e poderosas!
 - *Assim como o Python, hehehe.*



01

PySide6 / PyQT6

02

PySimpleGUI

03

TKinter



04

Kivy

05

wxPython

06

PySide2 / PyQt5





Python Easy Chess GUI v1.0

Mode Game FEN Engine Help



Mode Play

White Human

Black Stockfish 10 64 POPCNT

Adviser Be3 Be7 Qd2 - Lc0 v0.21.1

Move list

```
1. d4 Nf6 { book } 2. c4 d6 { book } 3. Nc3 Bf5 { book } 4. f3 e5 { book } 5. e4 exd4 6. Qxd4 Be6
```

Comment

I like Be3. Book prefers b3. Check Adviser's recommendation. Right Adviser prefers Be3.

BOOK 1, Comp games

no book moves

BOOK 2, Human games

move	score	weight
b3	2	100.0%

Opponent Search Info

-0.13 | 21 | 5.0s | Be6 Qd2 g6 b3 Bg7



All graphic widgets in one form!

Here is some text.... and a place to enter text

This is my text

Checkbox My second checkbox!

My first Radio! My second Radio!

This is the default Text should you decide not to type anything

A second multi-line

Combobox 1 85

Menu Option 1

Listbox 1
Listbox 2
Listbox 3

25 75 10

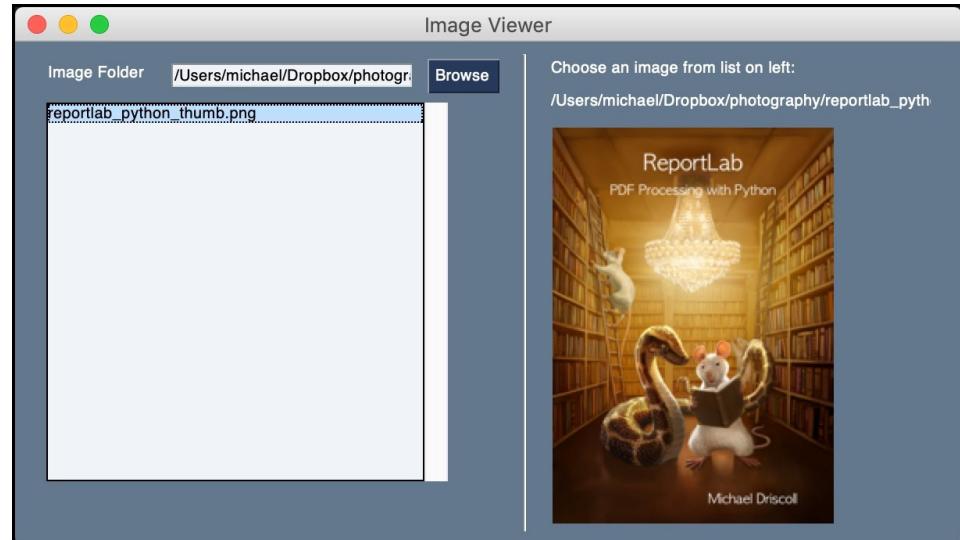
Column 1

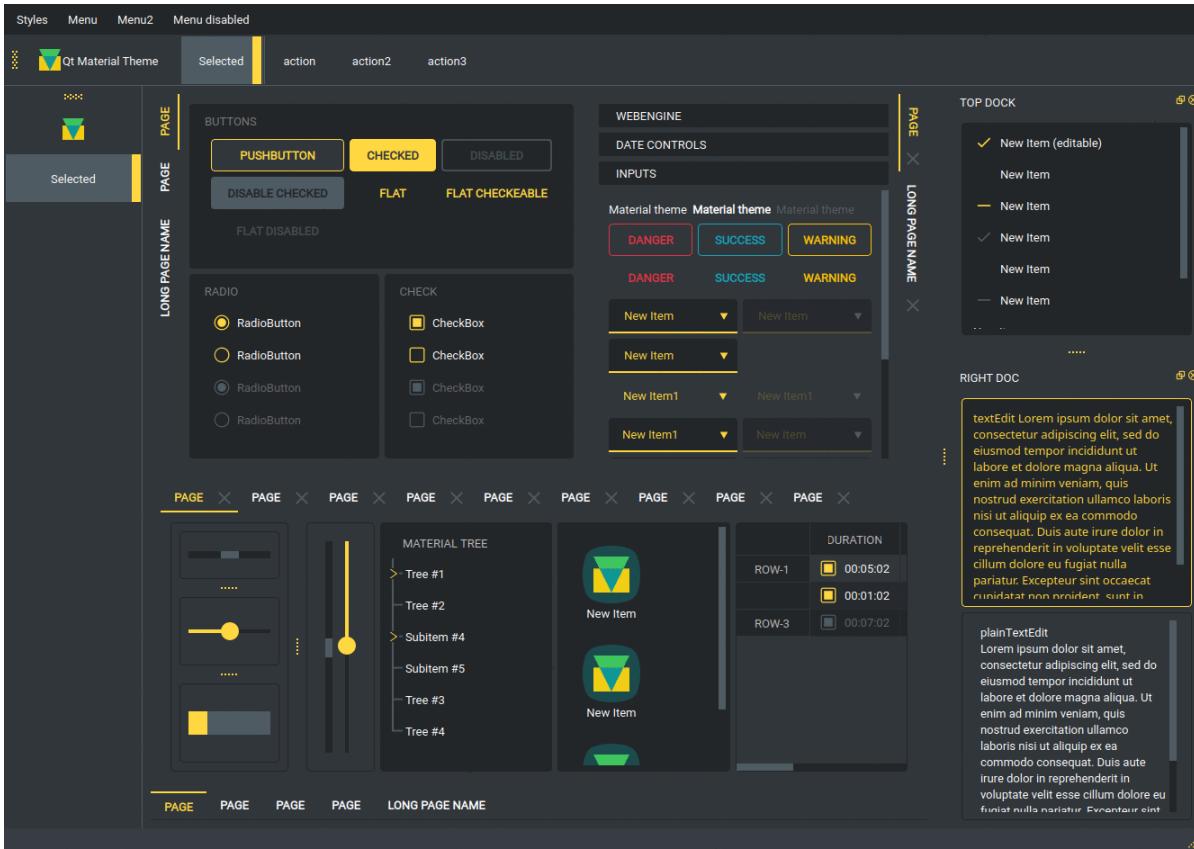
Spin Box 1
Spin Box 2
Spin Box 3

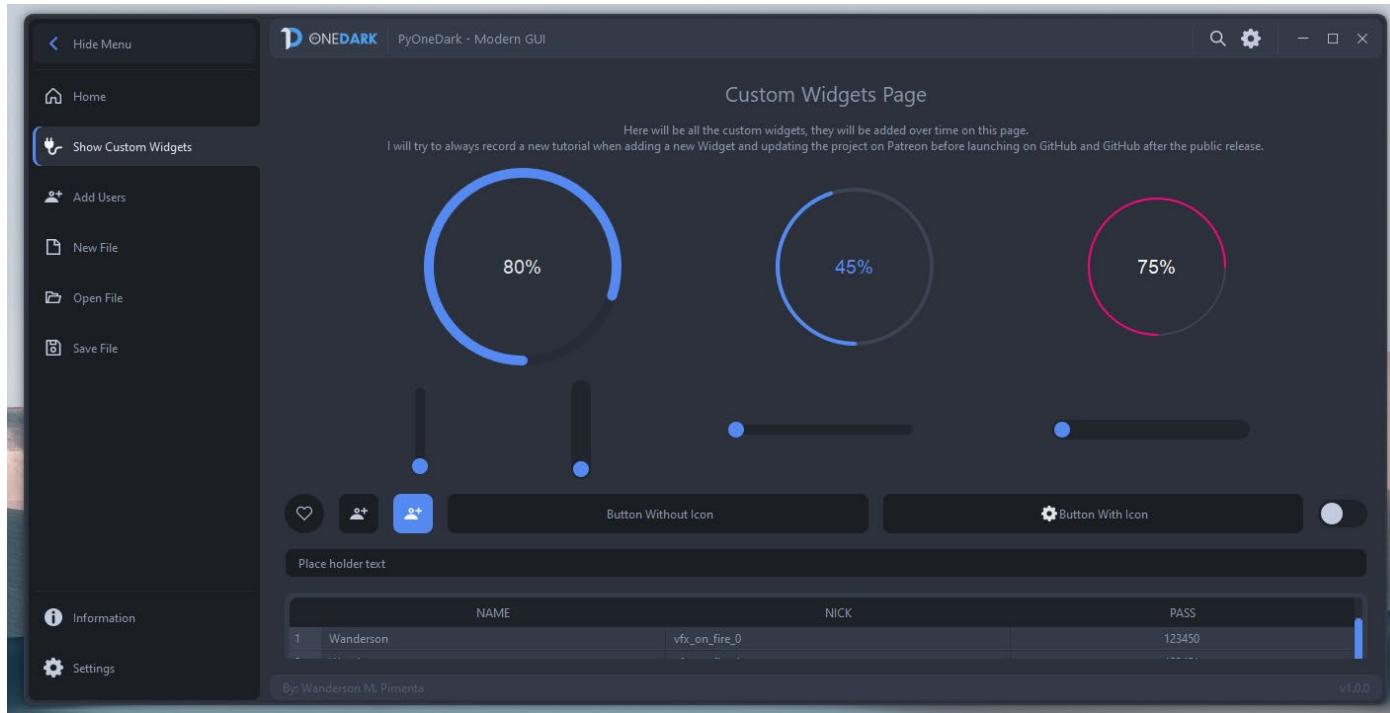
Choose A Folder

Your Folder Default Folder [Browse](#)

[Fill](#) [Cancel](#)









Banco de Dados

O que são e como usar?



Banco de Dados

Quando você precisa guardar seu dinheiro em um *lugar seguro e que consiga guardar* esse valor em dinheiro *por tempo indeterminado*, você vai até um *Banco*, certo?

Esse banco acima seria um “*Banco de Dinheiro*”.

Com dados, é a mesma coisa.

O *banco de dados* nada mais é que um lugar para *armazenar dados*.

Com dados queremos dizer:

- *Nome, Sobrenome, Idade, Login, Senha...*



Banco de Dados

Agora falando de uma forma mais técnica, os banco de dados são *coleções de dados interligados entre si*.

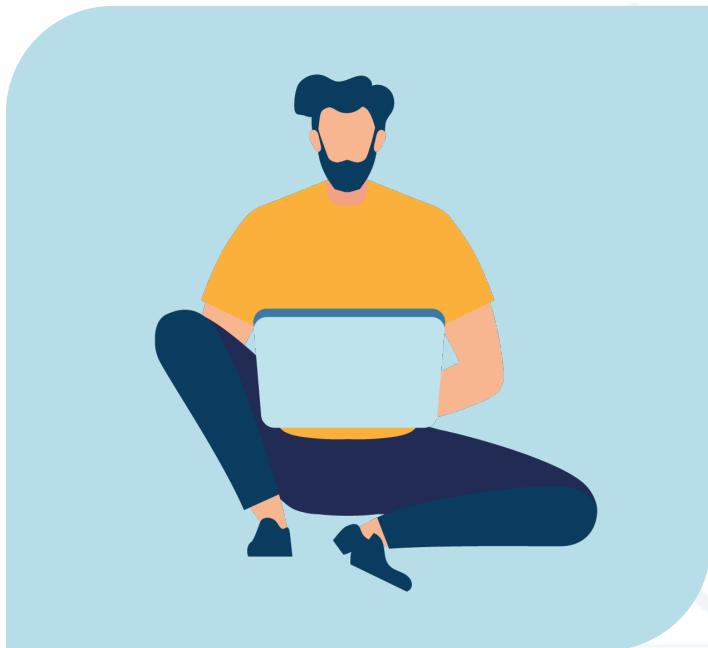
Os *principais motivos* de usar banco de dados são:

- *Salvar informações* mesmo após o término da aplicação.
- Guardar *dados de vários usuários* no seu próprio *servidor*.

Quer ler um artigo legal sobre o tema?
Então [clique aqui!](#)



OPERAÇÕES COM BANCO DE DADOS



Você pode usar banco de dados para *diversas funções*, dentre elas:

- *Adicionar* um novo valor (linha)
- *Editar* um valor existente
- *Deletar* um valor
- *Criar* uma nova tabela
- *Editar* uma tabela
- *Deletar* uma tabela
- *Recuperar* valores
- Entre outras

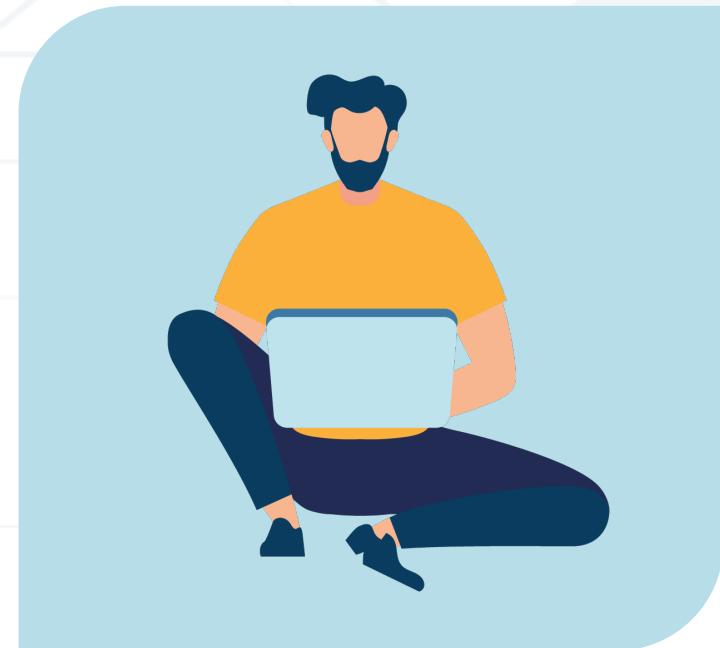
TIPOS DE BANCOS DE DADOS

Basicamente, temos dois tipos de bancos:

- *SQL* (Relacionais)
- *NoSQL* (Não Relacionais)

Os *primeiros são criados com orientação a conjuntos* e por isso suas informações são organizadas em *tabelas*, integrando as *colunas* e *linhas*.

Já os segundos, são para *dados mais complexos e que não podem ser tabulados*, como imagens, mapas e dados mistos.



Banco de Dados SQL

Para isso, nós temos as *DataBases*, que são nossas bases de dados, como:

gerenciador_aluno

Temos as tabelas, que são nossas estruturas, como:

alunos e notas

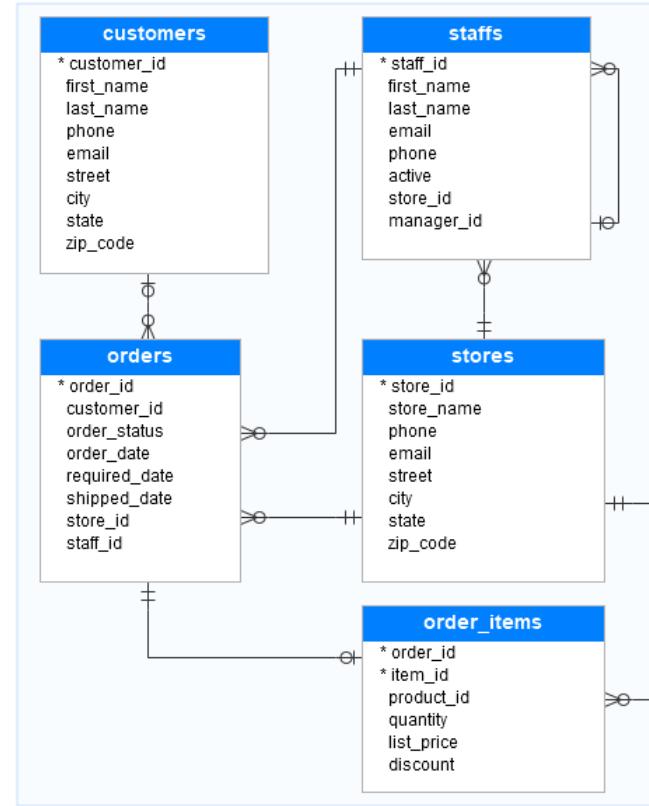
Temos as colunas, que são as propriedades das tabelas, como:

nome, sobrenome e idade

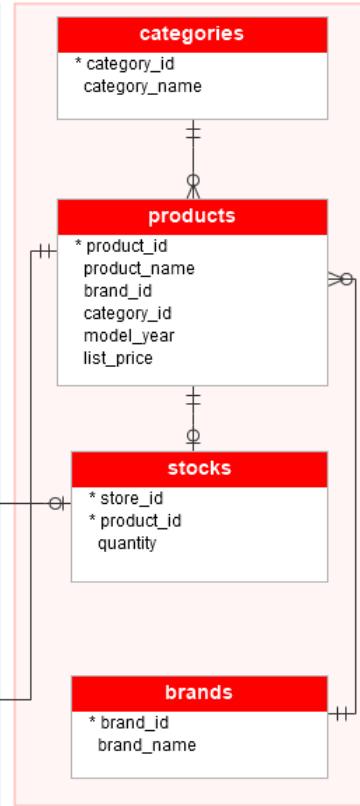
Por fim, temos *as linhas*, que são os *dados que inserimos nas tabelas*, com seus respectivos *valores para cada coluna*.



Sales



Production



Data View - Employee (C:\Program Files\Autodesk\AutoCAD 2016\Sample\Database Connec...)

— New Link Template — | — New Label T

	Emp_Id	Last_Name	First_Name	Gender	Title
▶	1000	Torbati	Yolanda	F	Programmer
	1001	Kleinn	Joel	M	Programmer
	1002	Ginsburg	Laura	F	President
	1003	Cox	Jennifer	F	Programmer
	1005	Ziada	Mauri	M	Product Designer
	1006	Keyser	Cara	F	Account Executive
	1010	Smith	Roxie	M	Programmer
	1011	Nelson	Robert	M	Programmer
	1012	Sachsen	Lars	M	Support Technician
	1013	Shannon	Don	M	Product Designer

Record 1

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	58
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotltaw	28

SQLite

A colorful illustration depicting a group of people working on computers. In the foreground, a person sits in a red beanbag chair, wearing a yellow t-shirt and blue jeans, working on a laptop. To their right, another person sits cross-legged on the floor, wearing a teal sweater and dark pants, also working on a laptop. In the background, two more people are visible: one sitting at a desk with two monitors, wearing a red shirt, and another standing behind them, wearing a white shirt. Various speech bubbles containing code snippets (like '</>' and snippets of SQL) float around the scene.

Um exemplo bem famoso de Banco de Dados SQL é o *SQLite*.

Ele é bem *simples, leve e fácil*.

Saiba mais clicando [aqui](#).

CRIANDO O BANCO

```
import sqlite3

arquivo_db = "bylearn.db"
conexao = sqlite3.connect(arquivo_db)
```

CRIANDO A TABELA

```
● ● ●  
import sqlite3  
  
sql_criar_tabela = """ CREATE TABLE IF NOT EXISTS alunos (  
                    id integer PRIMARY KEY AUTOINCREMENT,  
                    nome text NOT NULL,  
                    nota integer NOT NULL  
                ); """  
  
conexao = sqlite3.connect('bylearn.db')  
cursor = conexao.cursor()  
cursor.execute(sql_criar_tabela)
```

INSERINDO DADOS

```
import sqlite3

sql_inserir_aluno_felipe = "INSERT INTO alunos (nome, nota) VALUES ('Felipe',10)"
sql_inserir_aluno_jose = "INSERT INTO alunos (nome, nota) VALUES ('José',8)"

conexao = sqlite3.connect('bylearn.db')
cursor = conexao.cursor()

cursor.execute(sql_inserir_aluno_felipe)
cursor.execute(sql_inserir_aluno_jose)

conexao.commit()
```

BUSCANDO DADOS

```
● ● ●  
import sqlite3  
  
sql_buscar_alunos = "SELECT * FROM alunos"  
  
conexao = sqlite3.connect('bylearn.db')  
cursor = conexao.cursor()  
  
cursor.execute(sql_buscar_alunos)  
alunos = cursor.fetchall()  
  
for aluno in alunos:  
    print("O aluno", aluno[1], "tirou nota", aluno[2])
```

O BÔNUS DOS BÔNUS



TESTES **UNITÁRIOS**



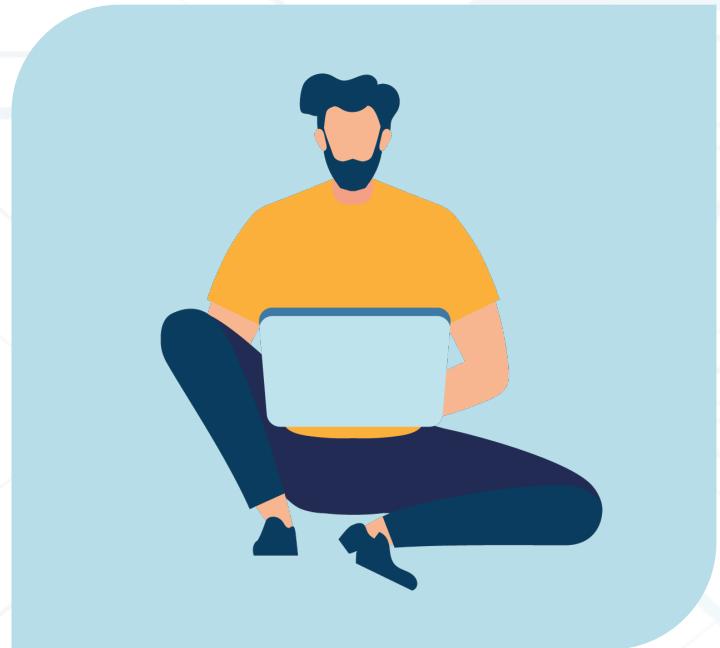
ByLearn

Importância dos Testes

Já imaginou como seria se *não testássemos* nossos códigos *antes de publicá-los*?

Bom... Eles se tornariam *jogos da Ubisoft*

Brincadeiras a parte, *catástrofes são evitadas graças ao teste de código* e eu posso te provar isso!



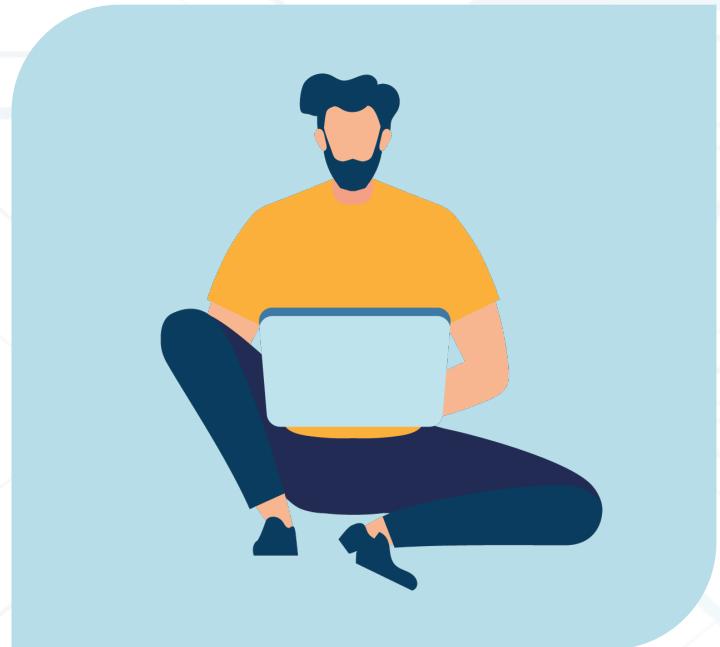
Importância dos Testes

Imagine se o Banco Central não testasse bem o Pix:

Colocar um *valor negativo* poderia te deixar com saldo extra e *negativar quem recebeu*.

$$\text{saldo} = \text{saldo} - (-10)$$

$$\text{saldo} = \text{saldo} + 10$$

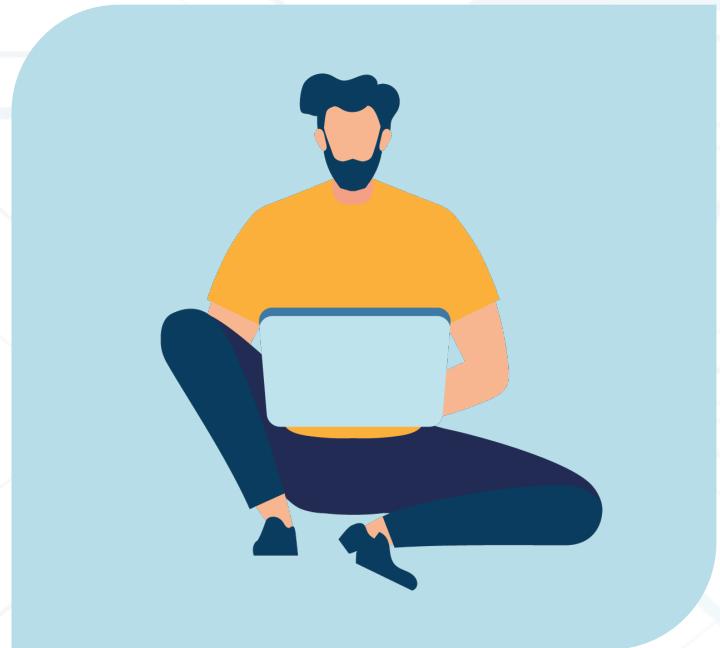


Importância dos Testes

Imagine se o Whatsapp não testasse MUITO BEM a criptografia ponta-a-ponta das mensagens:

Qualquer hacker poderia ter acesso a suas mensagens facilmente...

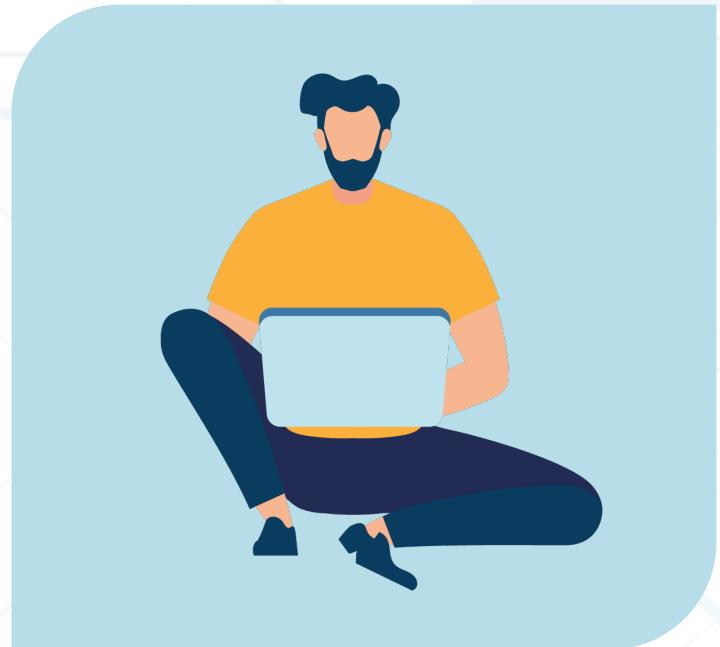
- Casamentos se desfariam;
- Fotos íntimas vazariam;
- Sociedades acabariam;
- Demissões então...



Importância dos Testes

Até mesmo em casos mais cotidianos e simples, com um *bug no cadastro* de um usuário ou *um falha na comunicação* com o Paypal podem ser catastróficos...

Imagine usuários não podendo mais se cadastrar no seu sistema, *como clientes não podendo criar contas em lojas*, ou *pagamentos não serem efetivados ou validados sem querer*.



Cobertura de Testes

Para validar que tudo estará correto e funcionando como deveria, nós precisamos *garantir uma alta cobertura dos testes.*

Isto é, garantir que *boa parte* (ou tudo) *do que existe lá foi testado* corretamente e seu funcionamento foi validado.

Tal tarefa pode ser *um processo árduo e difícil*... Ou pode *ser bem simples também!*

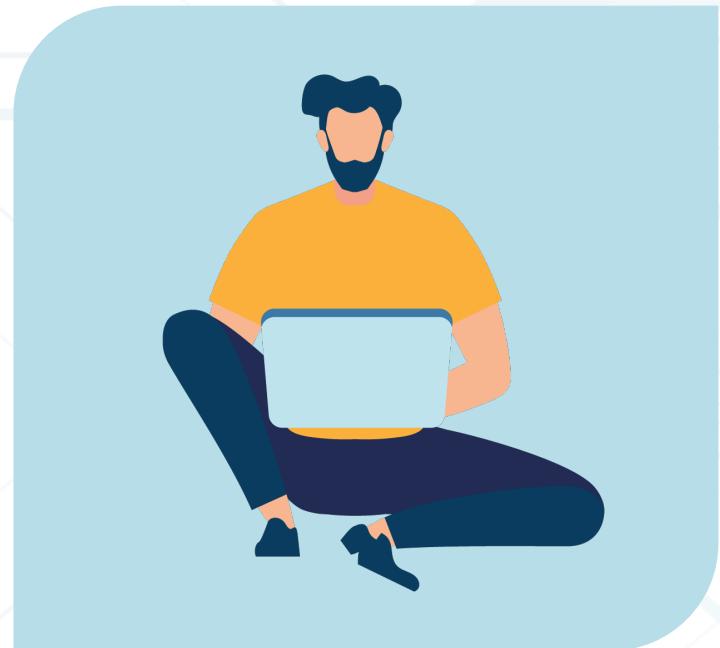


Testes Unitários

Uma metodologia criada para facilitar os testes e garantir total funcionamento foi a de *Testes Unitários*, também conhecidos por *Testes de Unidade*.

O nome acaba sendo meio sugestivo, mas só para deixar claro, essa metodologia de teste consiste em testar “*a menor parte testável de um sistema*”

Ainda tá difícil? A gente explica!



Testes Unitários

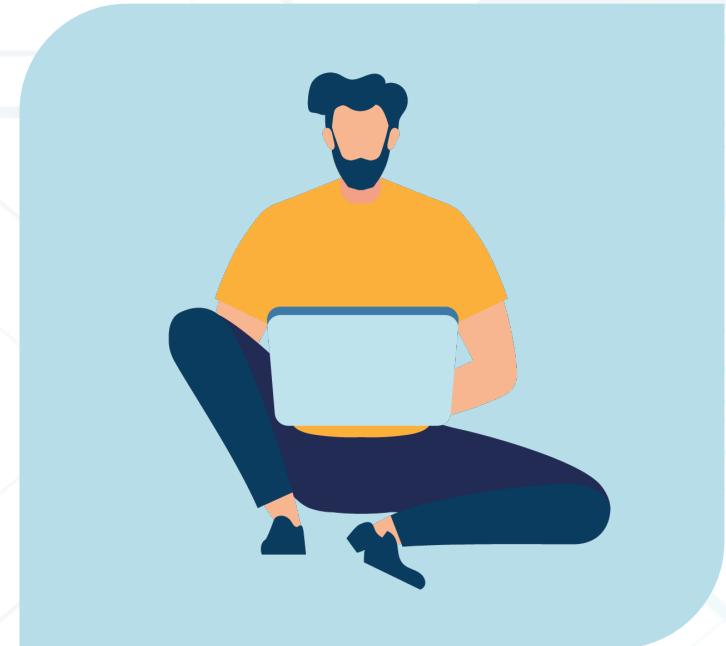
Imagine uma aplicação com 4 distintas funções, onde cada uma com uma só tarefa, como por exemplo:

“*def somar (valor1, valor2):*”

“*def subtrair(valor1, valor2):*”

“*def multiplicar (valor1, valor2):*”

“*def dividir(valor1, valor2):*”



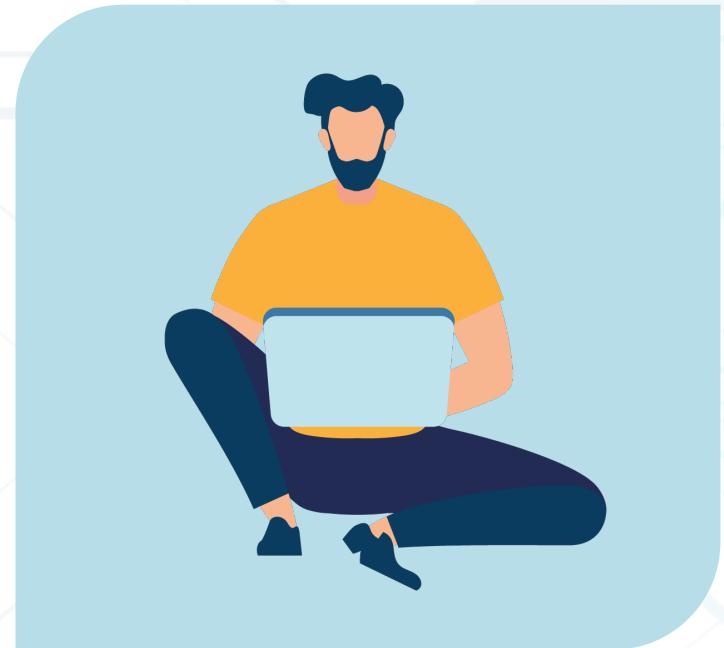
Testes Unitários

Os testes unitários dizem que devemos “testar cada fluxo/caminho possível”, ou seja...

Faremos *testes diferentes para cada uma das funções.*

Teremos testes para *soma, subtração, divisão e multiplicação.*

Todos separados e verificando uma só função



Testes Unitários

Com isso, garantimos testar a “*Menor Fração Possível*” do nosso código e validar se alguma deles tem erro.

Caso tenha erro, *saberemos exatamente onde isso ocorreu e porquê*.

Além disso, com o teste pronto e automatizado, *se mudarmos o código e isso bugar a aplicação, seremos notificados*.

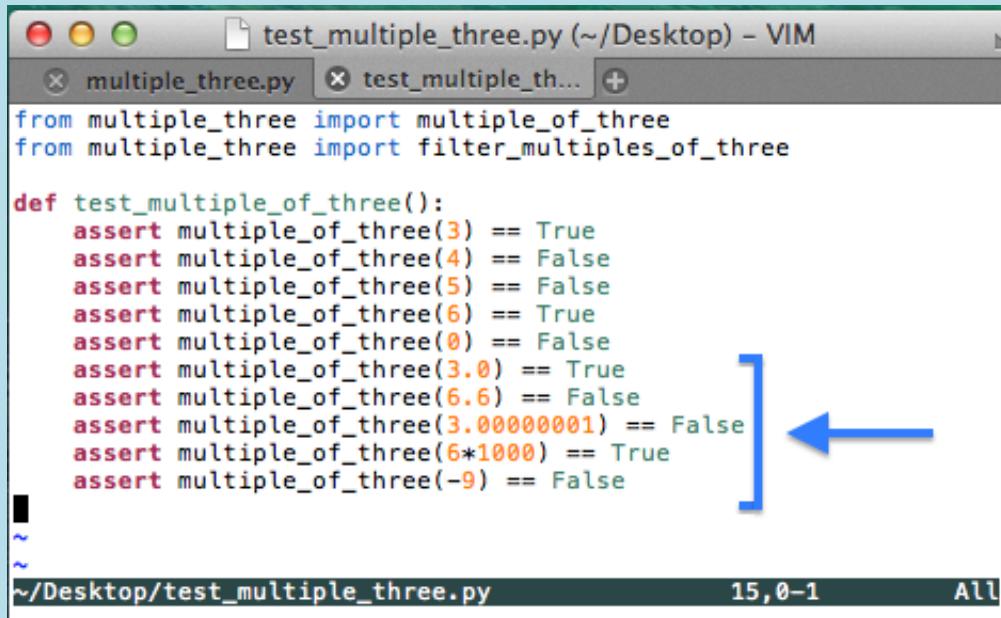


Testes Unitários

```
test_unittest.py ✘

1  import unittest
2  import inc_dec
3
4  class Test_TestIncrementDecrement(unittest.TestCase):
5      def test_increment(self):
6          self.assertEqual(inc_dec.increment(3), 4)
7
8      def test_decrement(self):
9          self.assertEqual(inc_dec.decrement(3), 4)
10
11 if __name__ == '__main__':
12     unittest.main()
```

Testes Unitários



A screenshot of a VIM editor window titled "test_multiple_three.py (~/Desktop) - VIM". The window shows Python code for testing the "multiple_of_three" function. A blue bracket and arrow highlight the test case for non-integer inputs.

```
from multiple_three import multiple_of_three
from multiple_three import filter_multiples_of_three

def test_multiple_of_three():
    assert multiple_of_three(3) == True
    assert multiple_of_three(4) == False
    assert multiple_of_three(5) == False
    assert multiple_of_three(6) == True
    assert multiple_of_three(0) == False
    assert multiple_of_three(3.0) == True
    assert multiple_of_three(6.6) == False
    assert multiple_of_three(3.00000001) == False
    assert multiple_of_three(6*1000) == True
    assert multiple_of_three(-9) == False
```

Testes Unitários

```
import unittest

class TestStringMethods(unittest.TestCase):

    def setUp(self):
        pass

    # Returns True if the string contains 4 a.
    def test_strings_a(self):
        self.assertEqual( 'a'*4, 'aaaa')

    # Returns True if the string is in upper case.
    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    # Returns TRUE if the string is in uppercase
    # else returns False.
    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())
```

Testes Unitários

Test Results		2 ms
✓	test_car	2 ms
>	✓ TestAccelerate	0 ms
✓	✓ TestBrake	0 ms
✓	✓ test_brake_once	0 ms
✓	✓ test_multiple_brakes	0 ms
✓	✓ test_multiple_brakes_a	0 ms
✓	✓ test_should_not_allow_	0 ms
>	✓ TestInit	2 ms

Testes Unitários

The screenshot shows a development environment with two main panes. On the left, a 'TEST: PYTHON' pane displays a file tree:

- Root: . (red X)
- test_pytest.py (green checkmark)
- test_unittest.py (red X)
 - Test_TestIncrementDecrement (red X)
 - test_decrement (red X)
 - test_increment (green checkmark)

A red rounded rectangle highlights the 'test_unittest.py' section and its child items.

On the right, the code editor pane shows the content of `test_unittest.py`:

```
1 import unittest
2 import inc_dec
3
4 class Test_TestIncrementDecrement(unittest.TestCase):
5     ✓ Run Test | ✓ Debug Test
6         def test_increment(self):
7             self.assertEqual(inc_dec.increment(3), 4)
8
9         ✓ Run Test | ✓ Debug Test
10        def test_decrement(self):
11            self.assertEqual(inc_dec.decrement(3), 4)
12
13    if __name__ == '__main__':
14        unittest.main()
```

Callouts with rounded rectangles highlight specific parts of the code:

- 'Run Test' and 'Debug Test' buttons for the class definition at line 4.
- 'Run Test' and 'Debug Test' buttons for the `test_increment` method at line 6.
- 'Run Test' and 'Debug Test' buttons for the `test_decrement` method at line 8.

INTEGRAÇÃO



ByLearn

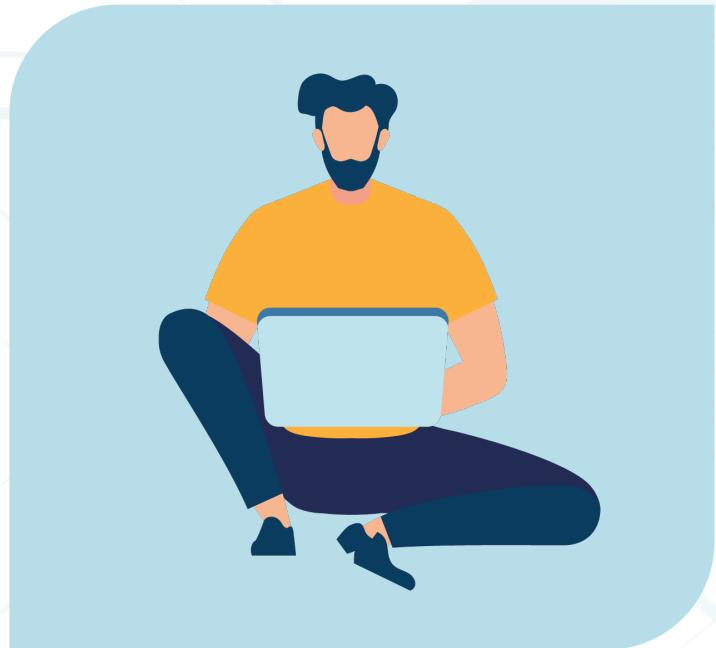
Integração

Certamente você já precisou fazer um trabalho maçante e demorado em algum programa (como criar planilhas no Excel) e pensou:

“Nossa, como eu queria automatizar isso aqui”

Tenho uma boa notícia se isso já aconteceu com você: Isso é sim possível!

Só precisar fazer uma comunicação entre tal programa e uma linguagem de programação.

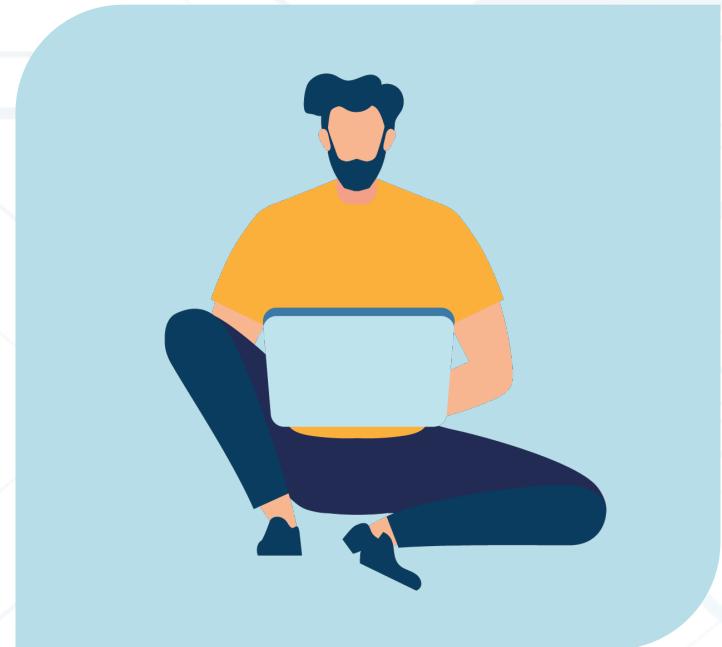


Integração

Essa comunicação, que funciona como uma “Ponte” entre uma linguagem de programação e o programa alvo *é chamada de Integração*.

Ou seja...

- Uma Integração do Python com o Excel *permite automatizar o Excel usando o Python*
- Uma *Integração do Python com* o programa “XYZ Mail Sender” pode permitir *o envio automático de e-mails!*



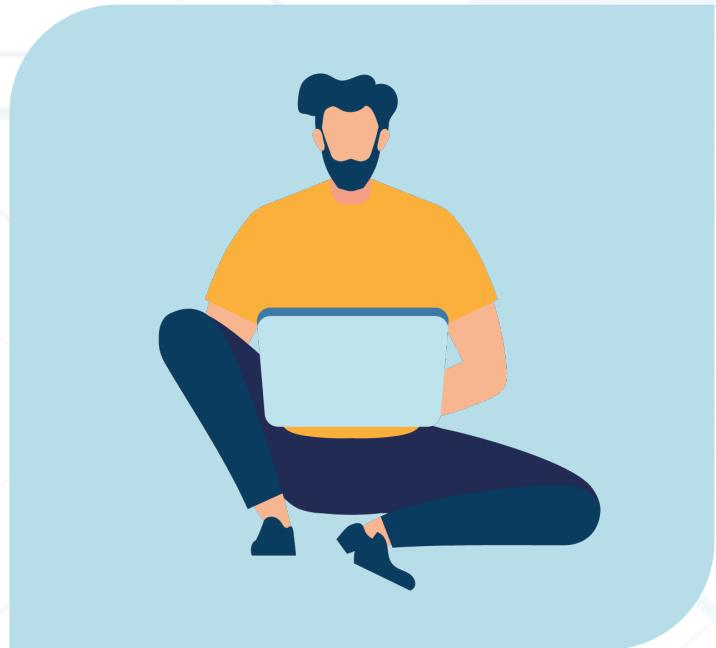
Integração

É sempre bom pesquisarmos e aprendermos sobre integrações, principalmente se nosso trabalho/função principal não é a programação.

Por exemplo, para um designer talvez compense pesquisar *integrações para automatizar o Photoshop*.

Para um contador, *integrações com Excel e PDF* vão calhar muito bem!

Há integração até com *Softwares médicos e/ou biológicos* para encontrar *curas para doenças*!



O EXTRA DO BÔNUS DO BÔNUS



EXPRESSÕES REGULARES



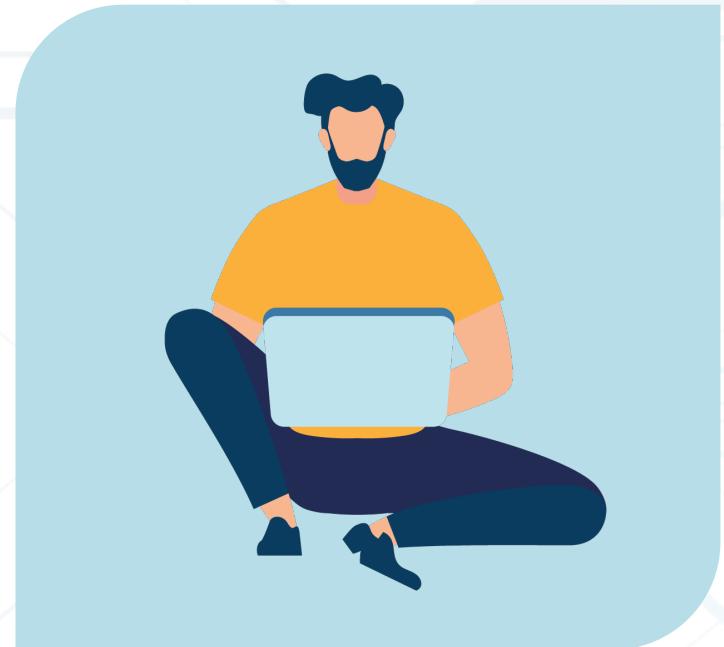
ByLearn

Expressões Regulares

“Expressões Regulares são padrões utilizados para selecionar combinações de caracteres em uma string.”

De uma forma mais simples, podemos falar que as Expressões Regulares *são strings especiais para buscar padrões em textos.*

Algo como um *CTRL+F bem poderoso!*



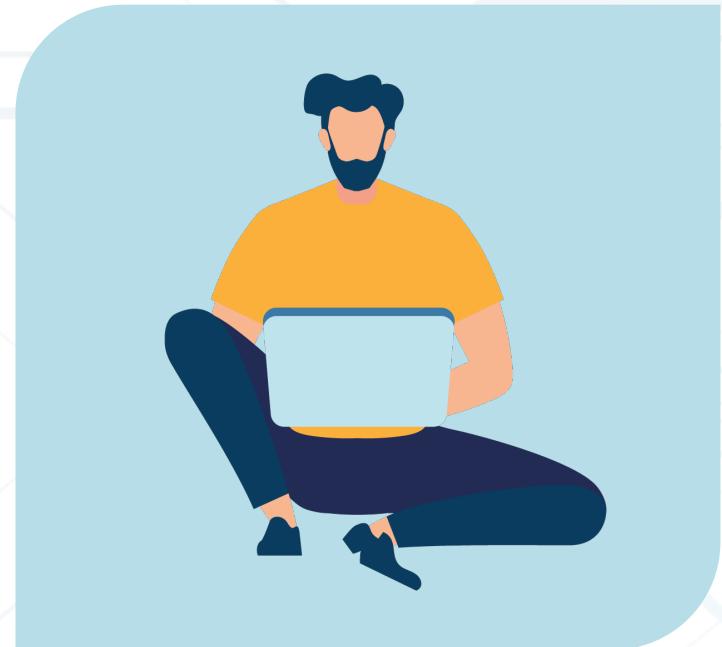
Expressões Regulares

Seu uso é bem *amplo e importante*.

Na *grande maioria dos softwares* que trabalham com validação de dados e busca de padrões, nós *temos o RegEx sendo usado!*

Exemplos de uso:

- Web Scrapping
 - A coleta/mineração de dados
- Validar E-mails
- Encontrar Palavras Chaves
- Substituir Termos





```
import re

texto = "Venha se tornar um ByLearner!"
padrao = r"\bByLearn\w*\b"

resultado = re.search(padrao, texto)

if resultado:
    print("Encontramos o padrão no texto, entre os índices: "
          +f"{resultado.start()} e {resultado.end()}")
else:
    print("Não encontramos o padrão no texto")
```

Saída:

*Encontramos o padrão no texto,
entre os índices: 19 e 28*



```
import re

texto = "O rato roeu a roupa do rei de Roma"
padrao = r"rato|roupa|rei"

ocorrencias = re.finditer(padrao, texto)

for ocorrencia in ocorrencias:
    print(f"Encontrei: {ocorrencia.group()} entre os índices {ocorrencia.span()}")
```

Saída:

Encontrei: rato entre os índices (2, 6)
Encontrei: roupa entre os índices (14, 19)
Encontrei: rei entre os índices (23, 26)

```
● ● ●  
import re  
  
texto = "Um dois quatro quatro cinco"  
padrao = r"quatro"  
substituicao = r"tres"  
  
novo_texto = re.sub(padrao, substituicao, texto, 1)  
print(novo_texto)
```

Saída:

Um dois três quatro cinco



```
import re

texto = "O rato roeu a roupa do rei de Roma"
padrao = r"rato|roupa|rei"

ocorrencias = re.finditer(padrao, texto)

for ocorrencia in ocorrencias:
    print(f"Encontrei: {ocorrencia.group()} entre os índices {ocorrencia.span()}")
```

Saída:

Encontrei: rato entre os índices (2, 6)
Encontrei: roupa entre os índices (14, 19)
Encontrei: rei entre os índices (23, 26)



```
import re

texto = "O rato roeu a roupa do rei de Roma"
padrao = r"[rR](\w*)"
substituicao = r"g\1"

novo_texto = re.sub(padrao, substituicao, texto)
print(novo_texto)
```

Saída:

O gato goeu a goupa do gei de goma

```
● ● ●  
import re  
  
texto = "Felipe Cabrera"  
padrao = r"\w+\s\w+"  
  
regex = re.fullmatch(padrao, texto)  
  
if regex:  
    print("Valido")  
else:  
    print("Inválido")
```

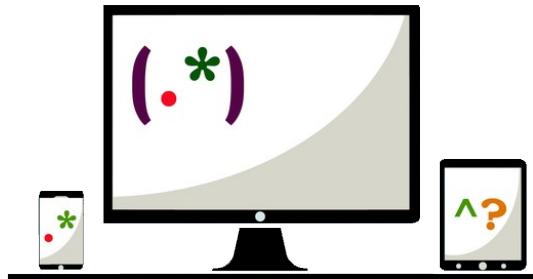
Saída:
Válido

EXPRESSÕES REGULARES

Agora que você já sabe a importância, que tal aprender de verdade as regras do RegEx?

Temos dois materiais fantásticos te esperando:

- 1) Aprenda sobre Expressões Regulares
- 2) Aprenda usar o RegEx com o Python





Boas Práticas

Nem precisa comentar, né?



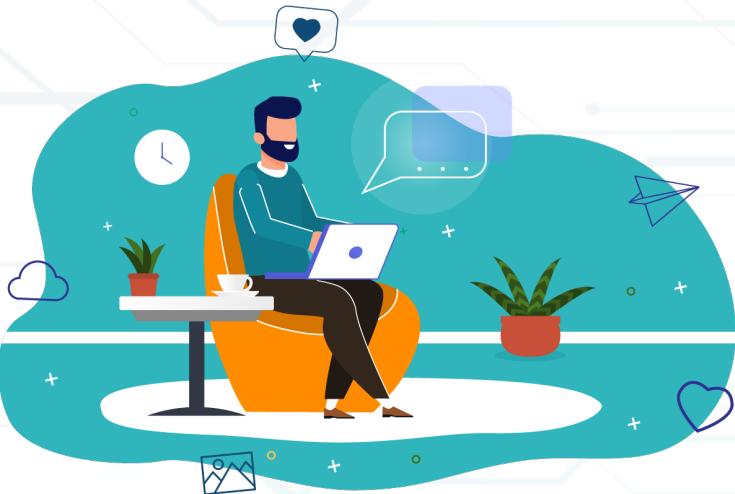
Boas Práticas

Essa nem precisava estar aqui nos bônus né?

Você comeria uma comida feita por um cozinheiro que cutuca o nariz com o dedo, não lava a mão e faz todas as comidas sem luva (e gripado)?

Você se sentiria seguro em fazer uma cirurgia com um médico que já esqueceu 2 bisturis e 1 celular dentro de pacientes anteriores?

Pois é... Programadores também não querem utilizar um código feio e mal estruturado!



Boas Práticas

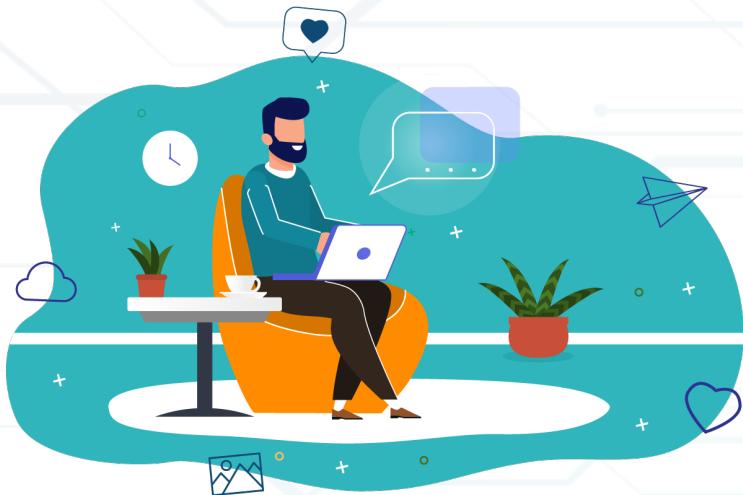
Um código de verdade vai muito além do que apenas funcionar.

O que eu mais ouço de [programadores que não seguem boas práticas](#) é:

“Nossa, nem eu sei o que esse meu código faz”

E se nem o criador sabe identificar o que aquele código faz, imagina seus colegas de trabalho...

Já vi demissões acontecerem por causa disso!



Boas Práticas

Tem um ditado muito legal que eu nunca vou conseguir esquecer, de tão importante:

“Softwares são feitos para máquinas.
Códigos são feitos para programadores”

Bom... Nós sabemos que o software final (*código alvo*) será lido e executado por uma máquina, né?

Porém, o *código fonte* sempre será lido e por programadores (pessoas).

Sendo assim, deve estar humanamente entendível!

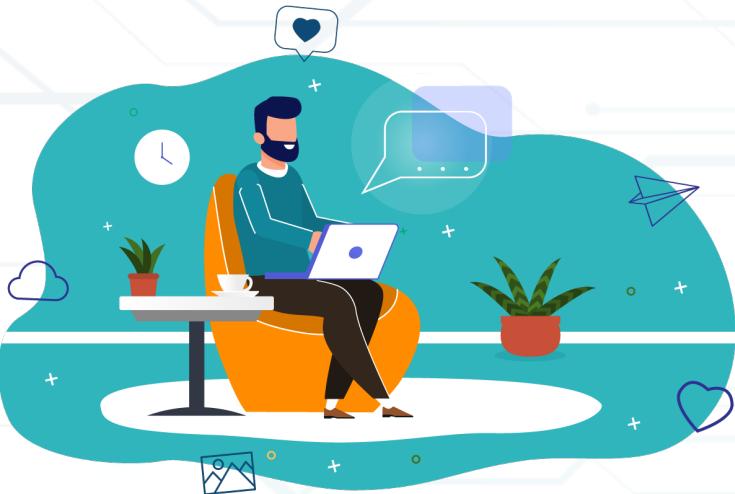


Boas Práticas

E para garantir essa *boa legibilidade* (*facilidade de leitura e entendimento*) e *organização estrutural* da sua aplicação/código as linguagens de programação possuem as “*Guidelines*”.

São elas quem ditarão as boas práticas para tornar seu código realmente bem feito e que dê um orgulho interior!

Cada linguagem tem suas especificidades, mas muitas dicas são genéricas e eficazes para todas.

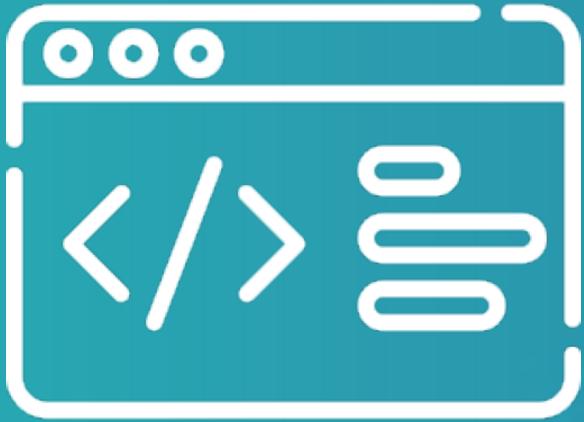


Boas Práticas

Exemplos:

- Não usar acentos e caracteres especiais em nomes de variáveis
- Dar nomes lógicos para as variáveis
- Separar blocos lógicos
- Não complique o simples
- Separar bem as funções/classes pela suas responsabilidades corretas





CONHECENDO AS IDES

Qual será sua favorita?



VS Code

Na verdade, o VS Code é considerado um *Editor de Códigos*.

Embora seja muito *poderoso* e *completo*, com inúmeras extensões, ele é muito *rápido* e *leve*.

A escolha perfeita para scripts leves e simples



Spyder

Bem similar ao *R-Studio*, uma excelente alternativa para programadores R acostumados com ele.

Possui um visualizador de variáveis bem fácil e poderoso, ótimo para ter uma *Preview* dos dados.

Se interessou? Veja nosso vídeo sobre ele [aqui](#).



PyCharm

Uma *IDE dedicada apenas ao Python*, feita pelo Jetbrains.

Bem *poderosa*, porém, bem *pesada*, com *inúmeras funcionalidades*.

Geralmente usada em projetos grandes que não envolvam visualização de dados, como, por exemplo, *Desenvolvimento Web*.



Jupyter

Muito parecido com o Colab, afinal, o *Jupyter foi a inspiração para o Google Colab.*

Trabalha com células de código e é muito utilizado em *Data Science e Machine Learning.*

Muito *poderoso, simples e organizado.*



PRODUTIVIDADE



ByLearn

Produtividade

Lembra quando falamos no início da Jornada que as IDEs ajudam e auxiliam no bom desenvolvimento de códigos?

Pois bem... *Elas estão completamente ligadas a produtividade na programação.*

Principalmente o VS Code, que tem se tornado o mais popular de todos justamente *por conta do ganho em produtividade que ele garante!*



Produtividade

Eu poderia te contar mais *CEM DICAS* que eu reuni durante os anos, mais *CEM EXTENSÕES* que eu já usei e tudo mais...

E poderia te falar que isso me fez (em menos de um ano) melhorar minha produtividade em

MAIS DE 3 VEZES

(agora você entende o porquê eu amo o VS CODE?)



Produtividade

Porém...

Como TUDO ISSO VAI ESTAR NO CURSO, eu tive
uma ideia melhor:

Amanhã eu te mostrarei na prática o uso do VS
Code para o desenvolvimento de uma aplicação
completa usando o Python!



EXEMPLOS PRÁTICOS



```
● ● ●

nota1 = 7.5
nota2 = 4.8

# Definimos o que é "Verificar uma Aprovação"
def verificar_aprovacao():
    media = calcular_media([nota1, nota2])

    if media >= 6:
        print("O Aluno Foi Aprovado!")
    else:
        print("O Aluno Foi Reprovado")

# Definimos o que é "Calcular a Média"
def calcular_media(notas):
    quantidade = len(notas)

    soma = 0
    for nota in notas:
        soma = soma + nota

    media = soma / quantidade

    return media

# Chamamos (executamos) a função de Verificar Aprovação
verificar_aprovacao()
```



```
frutas = ['Maça', 'Banana', 'Pera', 'Uva']
guloseimas = ['Bolacha', 'Batata', 'Fini', 'Chocolate']
comidas = ['Arroz', 'Feijão', 'Carne']
bebidas = ['Refrigerante', 'Suco de Laranja', 'Água']

categorias = ['Frutas', 'Guloseimas', 'Comidas', 'Bebidas']
compras = [frutas, guloseimas, comidas, bebidas]

for indice, categoria in enumerate(categorias):
    print('Você precisa comprar', len(compras[indice]), categoria+':')
    for compra in compras[indice]:
        print('-', compra)
```

```
def validar_idade(idade):
    if idade < 18:
        print('\nDesculpe, você não tem idade para prosseguir,', nome)
        return False
    else:
        print('\nÓtimo! Podemos prosseguir,', nome)
        return True

def escolher_carta():
    print("Digite uma das opções baixo:")
    print("1 - Carro\n2 - Moto\n3 - Carro e Moto")

    return int(input())

def calcular_preco(escolha):
    valor_carro = 1500
    valor_moto = 1000

    if escolha == 1:
        return valor_carro
    elif escolha ==2:
        return valor_moto
    else:
        return valor_carro + valor_moto

def desconto(valor):
    return valor - (valor * 0.10)

nome = input('Digite o seu nome: ')
idade = int(input('Digita sua idade' ))

if validar_idade(idade):
    escolha = escolher_carta()

    print('\nPerfeito! Vou calcular o valor')
    valor = calcular_preco(escolha)

    print('\n'+nome, 'o valor total é de', valor, 'reais')
    print('Mas vou ver com meu gerente se posso dar um desconto...')
    valor = desconto(valor)

    print('\nCom desconto eu consigo fazer por', valor, 'reais.')

    print('Te interessa?\n1 - Sim\n2 - Não')
    interesse = int(input())
    if interesse == 1:
        print('\nPerfeito! Começaremos amanhã!')
    else:
        print('\nTudo bem :( \nMe avise se mudar de ideia.')
```

```
animais = []

animal = input("Digite o nome do seu animal de estimação ou digite 0 se não tiver nenhum: ")

while animal != '0':
    especie = input("Digite a Espécie desse animal: ")
    animais.append([animal, especie])
    animal = input('Se tiver mais animais, digite o nome dele. Ou digite 0 se não tiver: ')

if len(animais) == 0:
    print('\n\nVocê não tem animais')
else:
    print("\n\nVocê tem os seguintes animais:")
    for animal in animais:
        print("- Nome:", animal[0], "| Espécie:", animal[1])
```

SE LIGA NESSA SUPER OFERTA

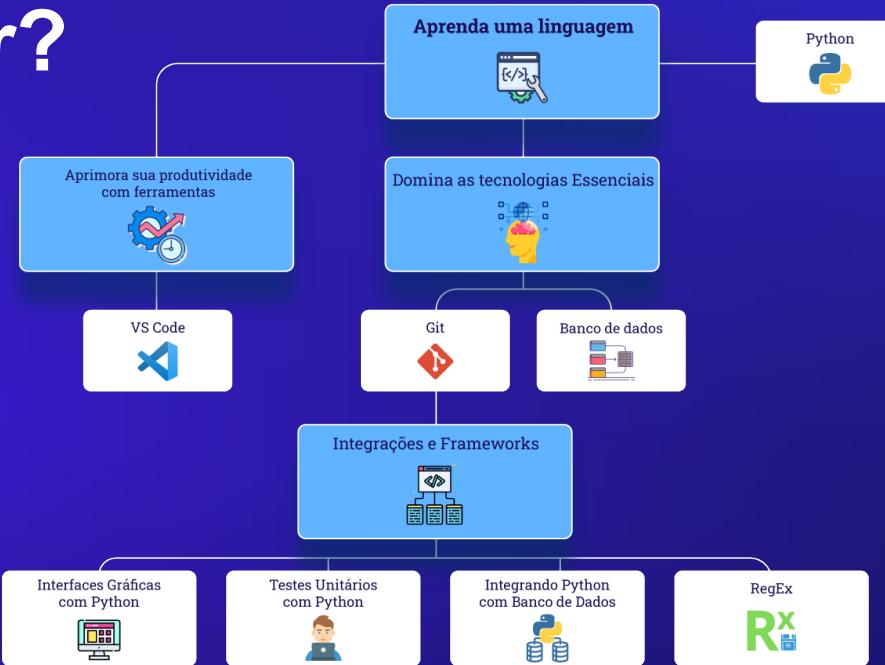


O que é a mentoria?

- ✓ É um acompanhamento com direção, desafios e tira-dúvidas a respeito de como ser um Programador (a), com minha experiência de mais de 10 anos.
- ✓ Ele contém o passo a passo detalhado com tudo o que você precisa saber para ser um programador de elite e conseguir uma carreira no brasil ou exterior, empreender na programação ou mesmo melhorar a automação e processamento de dados utilizando códigos, independente da sua área de atuação.

O que você vai receber?

- ✓ **Mentoria de 03 meses com acompanhamento e desafios semanais**
- ✓ **A trilha gravada do método triplo01**
 - Curso completo de Python do Zero a Pro
 - Git com Github
 - Regex com Python
 - Vscode produtividade infinita
 - Banco de dados
 - Testes Unitários
 - Interfaces gráficas



Todo conteúdo conta com Apostila

Bônus

✓ 05 Masterclass comigo e outros Profissionais

1. Mercado de Trabalho
2. Personal Branding
3. Marketing para Programadores
4. **Dev. Gringo (Receba 5x mais - Ganhe em dólar)**
5. **Desenvolvimento de Software (Renda Passiva)**



Bônus

- ✓ **Módulo Avançado: Expert em Python**

- Integração com Word

- Integração com Excel

- Integração com PDF

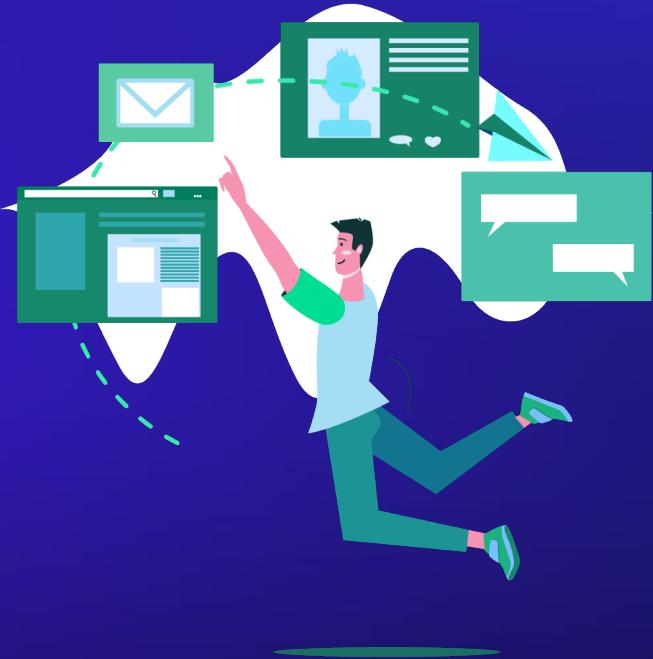
- Integração com Sistemas Operacionais

- Boas Práticas com Python

- ✓ **Grupo de Networking**

- ✓ **Certificado**

- ✓ **Garantia de 15 dias**



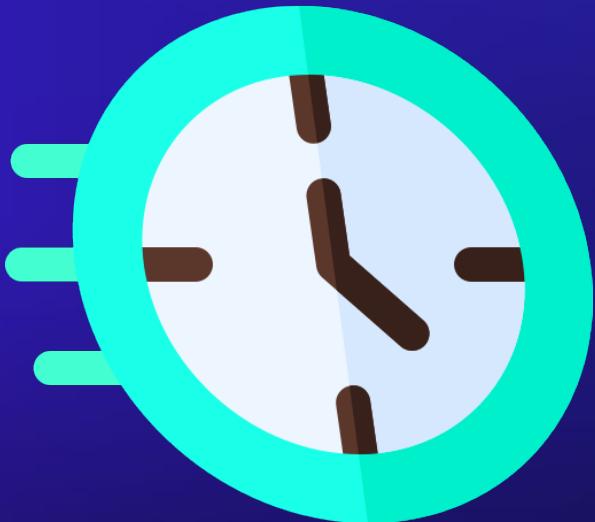
~~De 1497~~
por apenas
12x 116,62
1197

Acesso a todo material durante 02 anos. Podendo acessar por celular,
tablet e computador e assistir mesmo sem internet no celular

BÔNUS 20 PRIMEIROS

ATENÇÃO

- ✓ Os 20 primeiros Mentorandos terão um **SUPER BÔNUS**
- ✓ Acesso particular e individual a uma reunião exclusiva comigo com tema totalmente livre *por 30 minutos!*



DUPLA GARANTIA

Se dentro de 03 meses você fizer tudo o que eu mostro e não aprender nada eu devolvo o seu dinheiro e te pago ainda mais R\$ 500,00 como pedido de desculpas.

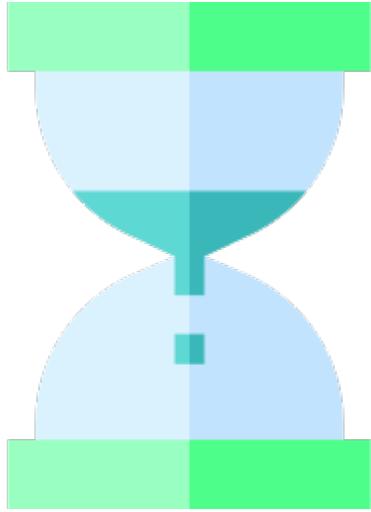


DUPLA GARANTIA

Se dentro de 03 meses você fizer tudo o que eu mostro e não aprender nada eu devolvo o seu dinheiro e te pago ainda mais R\$ 500,00 como pedido de desculpas.



Mas você terá que me provar que assistiu todas as aulas e participou de todas as mentorias ao vivo e não aprendeu nada de programação.



ENTÃO CORRA!

POIS SERÁ POR TEMPO LIMITADO E A
AMPULHETA JÁ ESTÁ CONTANDO!

VOCÊ
NÃO
VAI QUERER
FICAR FORA
DESSA TURMA!



ByLearn