



SEJA UM PROGRAMADOR COM 50

PÁGINAS

**EBOOK QUE VAI TE AJUDAR A
SAIR DO 0 NA PROGRAMAÇÃO**

DANIEL GOMES



MAZE



Por:
DANIEL GOMES

Desenvolvedor de software e fundador
da Maze Software

SUMÁRIO

1 - RESUMO SOBRE PYTHON

2 - INSTALANDO O PYTHON

3 - PRIMEIROS PASSOS

4 - Tipos de dados

5 - Listas, Tuplas e dicionários

6 - Condições

7 - Laços

8 - Criando GUI's

9 - Aprofundando no Tkinter

10 - Cap. Bônus

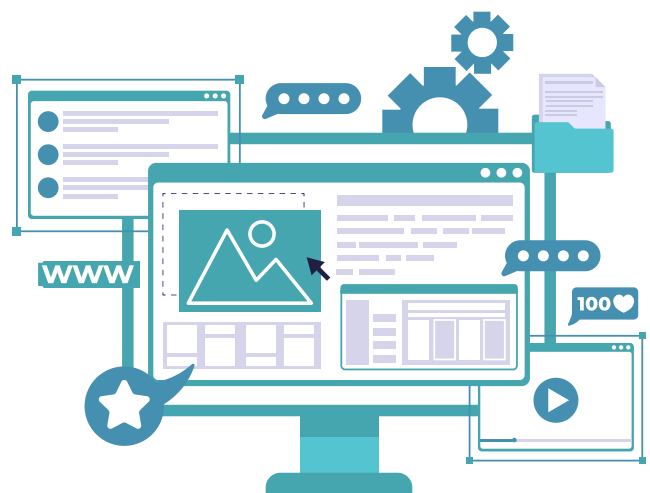
Obs: Os caps estão divididos em 5 páginas cada , para um bom aproveitamento do conteúdo é necessário seguir TODOS OS PASSOS .

Ass Daniel Gomes.

PYTHON

O Python é uma linguagem interpretada e fracamente tipada (não precisamos declarar o tipo de uma variável, por exemplo). Além disso, é uma linguagem de propósito geral. Portanto, pode ser utilizada para solucionar qualquer tipo de problema, o qual pode ser atendido com um sistema desktop, para a web ou mobile.

Python foi criado no final dos anos oitenta(1989) por Guido van Rossum no Centro de Matemática e Tecnologia da Informação (CWI, Centrum Wiskunde e Informatica), na Holanda, como sucessor da linguagem de programação ABC, capaz de lidar com exceções e interagir com o sistema operacional Amoeba.



PYTHON

O nome da língua vem do gosto de seu criador pelos humoristas britânicos Monty Python. Van Rossum é o principal autor de Python, e seu papel central contínuo na decisão da direção de Python é reconhecido, referindo-se a ele como Ditador de Vida Benevolente (em inglês: Benevolent Dictator for Life, BDFL).

Python é uma linguagem de programação interpretada cuja filosofia enfatiza uma sintaxe favorecendo um código mais legível, além de ser “free”.

Usa tipagem dinâmica e forte, isso, significa que o próprio interpretador do Python infere o tipo dos dados que uma variável recebe, sem a necessidade que o usuário da linguagem diga de que tipo determinada variável é.

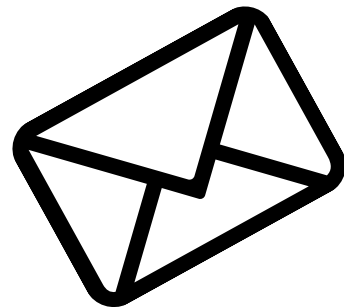
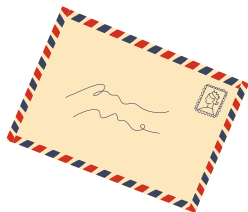


PYTHON

Definindo com termos mais técnicos, Python é interpretada, orientada a objetos, funcional, tipada, imperativa e de script.

Vamos entender um pouco melhor alguns princípios da linguagem e onde ela pode ser utilizada.

É comum ouvirmos a expressão de que “programar em Python é como escrever uma carta em inglês para o computador”, pois a linguagem tenta utilizar comandos intuitivos, como “print” para imprimir um texto na tela, “open” para abrir um arquivo, ou “find” para encontrar a posição de uma palavra.



PYTHON

A linguagem de programação Python foi desenvolvida sob 19 princípios, são eles:

Bonito é melhor do que feio.

Explícito é melhor do que implícito.

Simple é melhor do que complexo.

Complexo é melhor do que complicado.

Horizontal é melhor do que aninhado.

Esparsos é melhor que denso.

A legibilidade conta.

Casos especiais não são especiais o suficiente para quebrar as regras.

Porém, a praticidade supera a pureza.

Os erros nunca devem passar silenciosamente.

A menos que sejam explicitamente silenciados.

Diante da ambiguidade, recuse a tentação de adivinhar.

Deve haver uma, e de preferência apenas uma, forma óbvia de se fazer algo.

Embora essa forma possa não ser óbvia no início, a menos que você seja holandês.

Agora é melhor do que nunca.

Mas “nunca” é melhor do que “imediatamente agora”.

Se a implementação é difícil de explicar, é uma má ideia.

Se a implementação for fácil de explicar, pode ser uma boa ideia.

Namespaces são uma ótima ideia – vamos fazer mais disso!

PYTHON

Agora que já vimos os benefícios da linguagem, chegou a hora de ter uma visão mais abrangente sobre o que é possível fazer com Python.

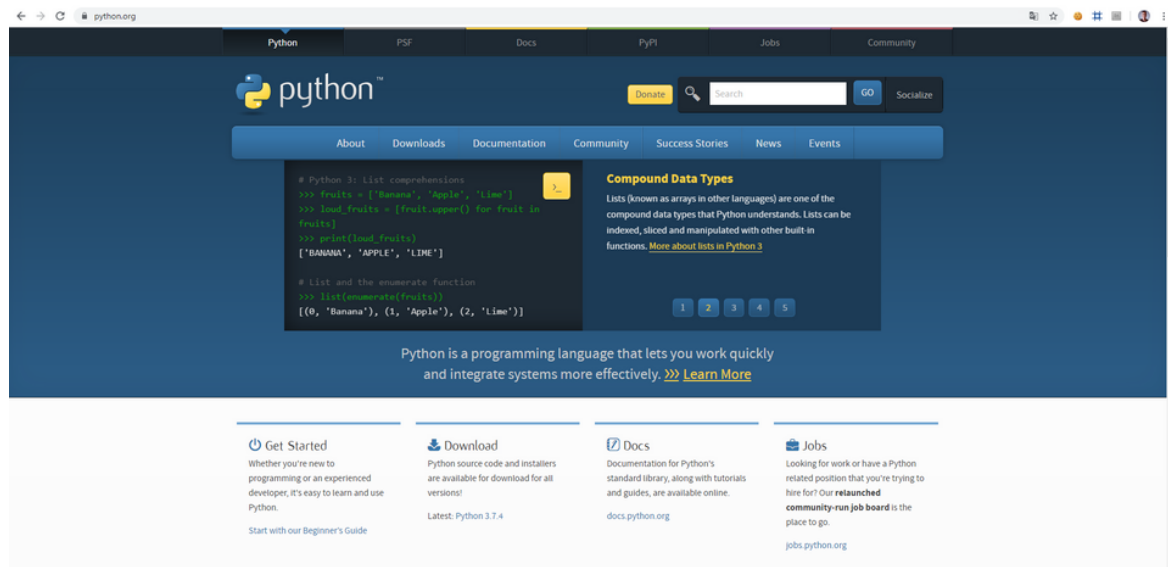
Python serve para:

- Automatizar tarefas repetitivas, criando códigos que interagem com seu sistema operacional;
- Varrer a internet (web scraping) navegando por sites, coletando, organizando e salvando informações;
- Monitorar e minerar redes sociais, conectando-se diretamente via APIs que facilitam a extração de dados;
- Construir um site ou uma aplicação para a web;
- Construir um aplicativo mobile;
- Criar aplicações em blockchain (diversos projetos descentralizados já possuem suporte para Python);
- Criar jogos;



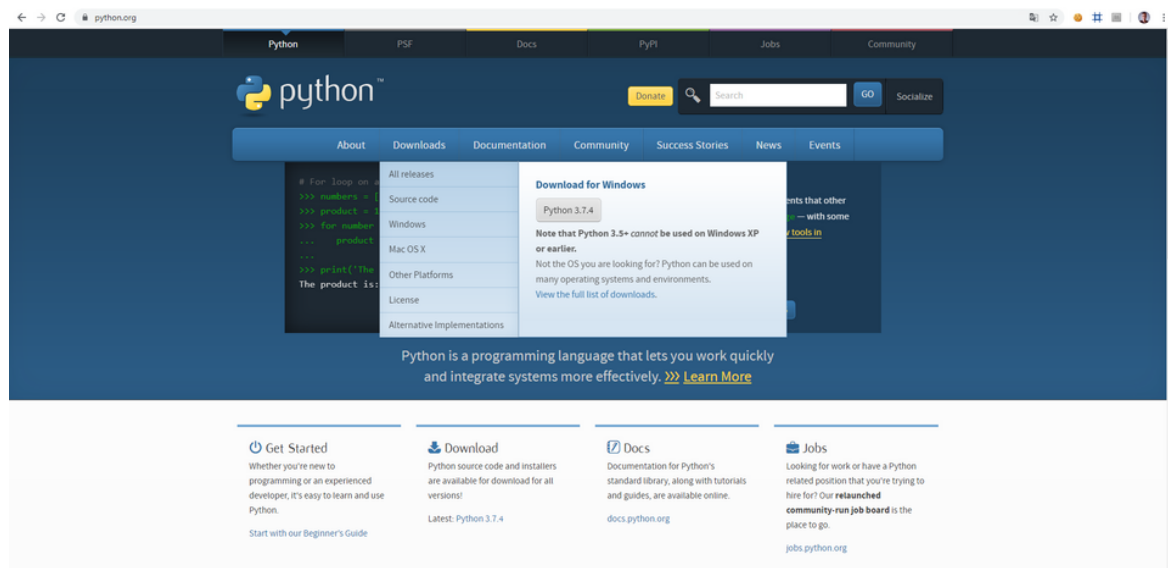
INSTALANDO O PYTHON

Para programar em Python é necessário fazer o download do arquivo de instalação que está no site : python.org (**Figura 1**).



INSTALANDO O PYTHON

Para baixar o arquivo, vá no item “Downloads” e clique no link que tem o nome da versão mais recente estável. Na data de publicação desse conteúdo, a versão é a 3.7.4, como vemos na **Figura 2**



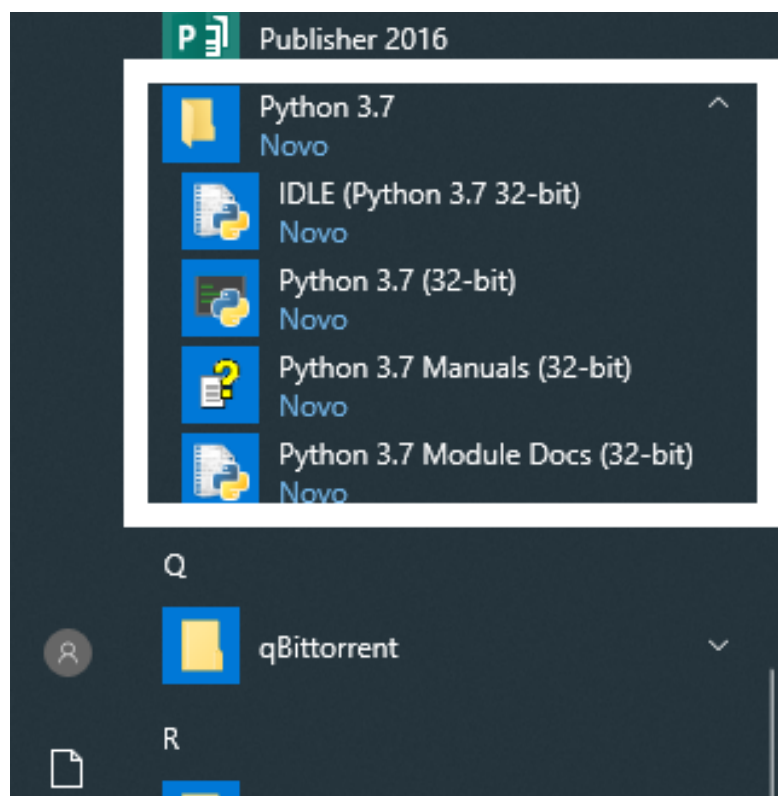
INSTALANDO O PYTHON

Após o término do download, abra o executável para começar a instalação do Python. A tela inicial da instalação abrirá, como na **Figura 3**. Marque a opção “Add Python 3.7 to Path”, para já deixar o Python configurado nas variáveis de ambiente do Windows. Clique em “Install Now” e aguarde até que a instalação seja concluída.



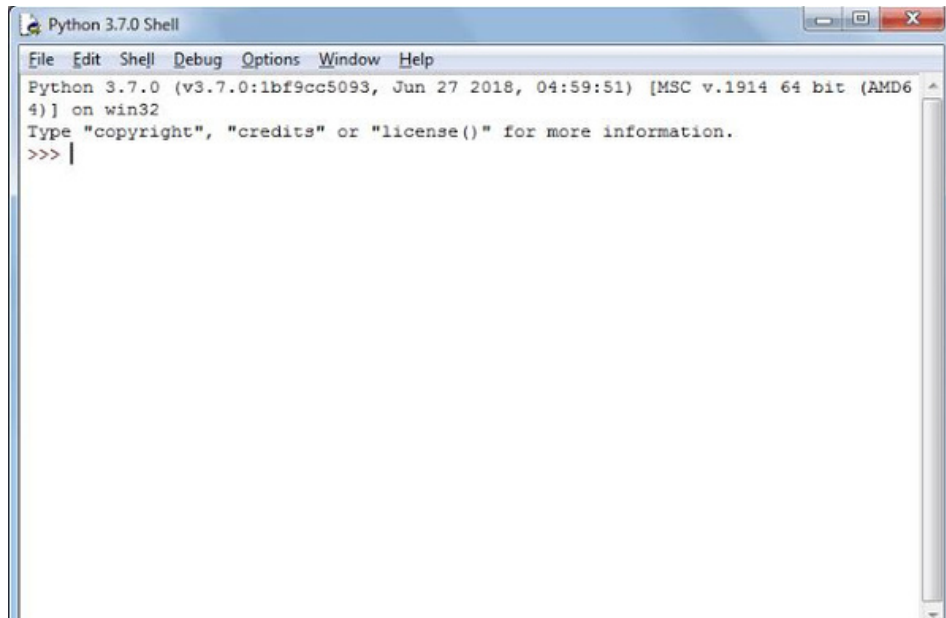
INSTALANDO O PYTHON

Para verificar se o Python foi instalado corretamente, abra o menu Iniciar do Windows e procure na lista de programas pelos arquivos que foram instalados, conforme é exibido na **Figura 4.**



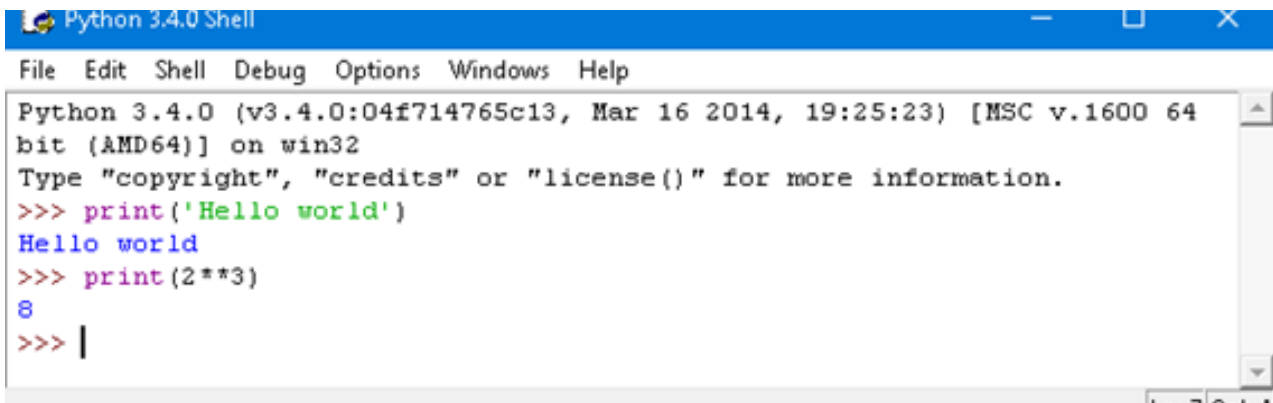
INSTALANDO O PYTHON

E agora é só abrir o IDLE E CODAR



PRIMEIROS PASSOS

No shell do Python, podemos executar alguns comandos para confirmar sua correta instalação. O código abaixo apresenta alguns exemplos. Nas linhas 1 e 2 é possível ver a execução de um “Hello world”, e nas linhas 3 e 4, a execução de uma expressão matemática.

A screenshot of the Python 3.4.0 Shell window. The title bar is blue and says "Python 3.4.0 Shell". Below it is a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following: "Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:25:23) [MSC v.1600 64 bit (AMD64)] on win32", "Type 'copyright', 'credits' or 'license()' for more information.", ">>> print('Hello world')", "Hello world", ">>> print(2**3)", "8", and ">>> |". The text is color-coded: "print" is purple, "Hello world" is green, and "8" is blue. The window has a scrollbar on the right side.

```
Python 3.4.0 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:25:23) [MSC v.1600 64
bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print('Hello world')
Hello world
>>> print(2**3)
8
>>> |
```

PRIMEIROS PASSOS

Comandos no Python

Um comando é uma instrução que o interpretador Python pode executar.

Quando um comando é digitado no console, o Python o executa e mostra seu resultado, caso haja algum valor a ser exibido. O resultado do comando `print`, por exemplo, é a impressão na tela do conteúdo que lhe é informado.

Vemos alguns exemplos de uso.

```
mensagem = 'Exemplo de mensagem'  
n = 25  
print (mensagem)  
print (n)  
print(n**2)
```

PRIMEIROS PASSOS

Operadores

Operadores são símbolos que representam operações matemáticas. Os principais são: +, -, *, / e **, que representam, respectivamente, adição, subtração, multiplicação, divisão e exponenciação. O código a seguir demonstra um exemplo de multiplicação entre horas e segundos:

```
>>> hora=60  
>>> segundos=60  
>>> print ('1 hora contém:  
' ,hora*segundos,'segundos')  
1 hora contém: 3600 segundos
```


PRIMEIROS PASSOS

Quando a expressão utiliza mais de um operador, a linguagem levará em consideração a regra de precedência, semelhante ao que ocorre na matemática convencional. Sendo assim, parênteses têm a maior precedência e podem ser utilizados na expressão para produzir um resultado desejado, fazendo com que uma determinada operação seja executada antes das outras. Em seguida, a exponenciação é a próxima na ordem, ou seja, $3^{*1}+1$ é igual a 4 e não a 5, da mesma forma que $2^{*1^{*3}}$ é igual a 2 e não a 8.

PRIMEIROS PASSOS

Já a multiplicação e divisão possuem a mesma precedência, sendo maior que a da adição e subtração, que também possuem a mesma precedência. Logo, $2*3-1$ é 5, não 4, e $6+4/2$ é 8, e não 5. Quando dois operadores têm a mesma precedência, eles são avaliados da esquerda para a direita. Ou seja, $5-3-1$ vale 1, e não 3. Isso ocorre porque inicialmente é feito o cálculo de $5-3$ e em seguida é subtraído o valor de 1. Em caso de dúvida, deve-se utilizar os parênteses para ter certeza que as expressões saíram conforme desejado.

TIPOS DE DADOS

Números inteiros

x = 1

y = 2

resultado = x + y

print(resultado)

print(type(resultado))

3

<class 'int'>

A função `type()` retorna o tipo do valor passado como parâmetro. No nosso exemplo, como a soma `x + y` possui um valor inteiro, a função `type()` retorna `<class 'int'>` indicando que o valor é de fato um número inteiro.

TIPOS DE DADOS

Se o nome ponto flutuante te assusta, não se preocupe. Números de ponto flutuante nada mais são que números com vírgula, ou como costumamos dizer "números quebrados", contrastando com os números inteiros com os quais estamos mais acostumados. Vejamos alguns exemplos de expressões com números de ponto flutuante.

```
# Números de ponto flutuante    3.30000000000000003  
    x = 1.1                        <class 'float'>  
    y = 2.2  
    resultado = x + y  
    print(resultado)  
    print(type(resultado))
```

TIPOS DE DADOS

Existem conversões explícitas de tipos em Python. Por exemplo, se quisermos que o resultado a expressão acima seja um número inteiro, podemos "forçar" que isso aconteça usando uma conversão explícita de dados:

```
x = 10  
y = 2.5  
resultado = int(x * y)  
print(resultado)  
print(type(resultado))
```

```
25  
<class 'int'>
```

TIPOS DE DADOS

Strings nada mais são do que sequências (cadeias) de caracteres. Em outras palavras, uma string é simplesmente uma sequência de zero ou mais letras juntas. Vejamos alguns exemplos.

```
string_vazia = ""  
uma_letra = 'a'  
varias_letras = 'abacate'  
print(type(string_vazia))  
print(type(uma_letra))  
print(type(varias_letras))
```

```
<class 'str'>  
<class 'str'>  
<class 'str'>
```

TIPOS DE DADOS

Outro nome que representa algo bem simples (mas que costuma assustar as pessoas) é o nome variáveis booleanas.

Essas variáveis nada mais são do que uma forma de se armazenar um valor lógico, ou seja, verdadeiro ou falso, que, em Python, são representados como True e False, respectivamente.

No exemplo abaixo, criamos uma variável booleana e imprimimos o tipo dessa variável.

Variáveis booleanas (verdadeiro/falso)

a = True

print(type(a))

<class 'bool'>

LISTA, TUPLA E DICT

Em algum momento durante a programação será necessário lidar com Listas, Tuplas e Dicionários, e na linguagem de programação Python também é possível e simples.

Vamos começar pela lista, uma lista é representada por sequência de objetos separados por vírgula e entre colchetes, além disso, ao ser declarada cada objeto recebe uma posição, pela qual será possível consultar o valor nessa posição através do index criado.

Esse index inicia com 0 (zero) e segue de acordo com a quantidade de informação da lista, um outro fator importante de utilizar a lista na programação, é que é mutável, isso significa que pode ser alterado a qualquer momento.

```
SegundaDev.py
Lista
Palestras = [ 'Django', 'Java', 'React', 'Docker', 'PHP' ]
              0      1      2      3      4
              -5     -4     -3     -2     -1

print(Palestras[3])    ==    Docker
print(Palestras[-2])

A lista PODE ser alterada a qualquer momento
após ser declarada, é mutável.
```


LISTA, TUPLA E DICT

Toda lista na linguagem Python é definida por colchetes e, dentro destes colchetes estão os objetos separados por aspas e vírgulas.

Neste exemplo, temos a variável palestras recebendo uma lista, ao ser declarada o Python define também o seu lugar dentro da memória, onde cada objeto recebe um número sendo que, da esquerda para direita inicia no espaço com número 0 (zero) e segue até completar todos os objetos.

Neste exemplo temos 5 objetos alocados em 0 a 4, mas da direita para esquerda os objetos recebem um outro número começando por -1 (menos um), e segue até completar todos os objetos.

Neste exemplo temos 5 objetos alocados em -1 a -5 e, podemos consultar cada objeto individualmente utilizando essa regra de alocação de objetos na memória do computador quando executado o programa.

Sendo assim, no exemplo mostrado no slide, imprimindo as posições 3 e -2 o resultado será correspondente ao objeto Docker.

Outra característica muito importante de trabalhar com listas, é que pode alterar o valor a qualquer momento, pois são mutáveis.

LISTA, TUPLA E DICT

Tupla é muito parecido com o comportamento da lista, porém deixo destacado que quando a Tupla é criada entre parênteses após ser declarada não pode ser mais alterada, ou seja, a Tupla é imutável.

A Tupla tem as mesmas opções de fatiamento que a Lista, ou seja, é possível consultar um objetivo de acordo com a sua posição, da esquerda para a direita iniciando do 0 (zero) e finalizando de acordo com a quantidade de objetos.

Da mesma forma para consultar objetos da direita para esquerda, iniciando do -1 e finalizando de acordo com a quantidade de objetos, ou seja, também temos index.

SegundaDev.py

Tupla

```
Palestras = ( 'Django', 'Java', 'React', 'Docker', 'PHP' )
```

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

```
print(Palestras[3])  
print(Palestras[-2])
```

= Docker

A tupla **NÃO PODE** ser alterada a qualquer momento após ser declarada, **é imutável**.

LISTA, TUPLA E DICT

Chegamos ao último item deste artigo com o Dicionários na linguagem Python, é criado por chaves e os objetos são compostos por uma chave, um valor para esta chave e os dois juntos a chave mais o seu valor, forma um item, porém para definir um item na linguagem de programação Python, entre a chave e o valor é necessário um símbolo, que é o : (dois pontos.) e o item por virgula.

E pode ser consultado somente a chave do objeto, somente o valor do objeto e somente o item de um dicionário.

SegundaDev.py

Dicionário

```
Palestras = { '1':'Django', '2':'Java', '3':'React', '4':'Docker', '5':'PHP' }  
print(list(Palestras.Keys()))  
print(list(Palestras.values()))  
print(list(Palestras.items()))
```

O dicionário PODE ser alterado a qualquer momento após ser declarado, é mutável.

CONDIÇÕES

Estruturas de condição são artifícios das linguagens de programação para determinar qual bloco de código será executado a partir de uma determinada condição. No Python, assim como em outras linguagens, podemos trabalhar com as estruturas de condição utilizando o if/else como veremos abaixo.

CONDIÇÕES

O if e o else são comandos que verificam determinada condição na programação.

O uso do if em um programa em Python visa verificar se determinada ação é verdadeira e executar o bloco de código contido em seu escopo. Basicamente é feita da seguinte forma:

```
media = 7  
if media > 6.9:  
    print ("Você foi aprovado")
```

CONDIÇÕES

No exemplo do código acima, utilizamos apenas o if para verificar se a variável media é maior que 6.9.

Como esta condição é verdadeira, imprimimos a mensagem na tela “Você foi aprovado”. Caso esta condição fosse falsa, o código seguiria normalmente ignorando, desta forma, a linha 3, o nosso print.

Já o uso o if/else fará com que uma das ações sejam executadas, já que se a condição dentro do if não for verdadeira, será executado o código contido no else. O if/else irá testar caso a condição seja verdadeira e executar uma determinada ação ou caso a mesma não seja executar outra.

CONDIÇÕES

```
media = 7
if media < 6.9:
    print ("Você foi reprovado")
else:
    print ("Você foi aprovado")
```

O código acima contém dois códigos a serem executados. Caso a media informada seja menor que 6.9, a pessoa será reprovada na disciplina, porém, caso a condição contida no if falhar, o código contido no else será executado. Desta forma, será exibido em tela que o aluno foi aprovado, já que a condição do if é falsa.

CONDIÇÕES

if...elif...else

O uso de if/elif/else serve para quando mais de uma condição precisar ser verificada. Imagine que possuímos duas condições: A primeira, se o aluno possuir uma média menor que 5 e a segunda, se a média for menor que 7 e maior que 5. Vimos anteriormente que utilizamos o if para testar se uma condição é verdadeira, porém, quando precisamos verificar uma segunda condição utilizamos o elif e adicionamos a condição a ser testada.

```
media = 7  
if media < 5:  
    print ("Você foi reprovado")  
elif media > 5 and media < 7:  
    print ("Você fará a recuperação")  
else:  
    print ("Você foi aprovado")
```


LAÇOS

Em Python, os loops são codificados por meio dos comandos for e while. O primeiro nos permite percorrer os itens de uma coleção e, para cada um deles, executar um bloco de código. Já o while, executa um conjunto de instruções várias vezes enquanto uma condição é atendida.

LAÇOS

Na Listagem 1 temos um exemplo de uso do comando for.

```
nomes = ['Pedro', 'João', 'Leticia']  
for n in nomes:  
    print(n)
```

A variável definida na linha 1 é uma lista inicializada com uma sequência de valores do tipo string. A instrução for percorre todos esses elementos, um por vez e, em cada caso, atribui o valor do item à variável n, que é impressa em seguida. O resultado, então, é a impressão de todos os nomes contidos na lista.

LAÇOS

O comando while, por sua vez, faz com que um conjunto de instruções seja executado enquanto uma condição for atendida. Quando o resultado passa a ser falso, a execução é interrompida, saindo do loop, e passa para o próximo bloco.

LAÇOS

Na Listagem 2 a seguir, vemos um exemplo de uso do laço while, onde definimos a variável contador, iniciando com 0, e enquanto seu valor for menor que 5, executamos as instruções das linhas 3 e 4.

```
contador = 0  
while contador < 5:  
    print(contador)  
    contador = contador + 1
```

Observe que na linha 4 incrementamos a variável contador, de forma que em algum momento seu valor igual a 5. Quando isso for verificado na linha 2, o laço será interrompido. Caso a condição de parada nunca seja atingida, o loop será executado infinitamente, gerando problemas no programa.

LAÇOS

Estruturas de repetição estão presentes na maioria das linguagens de programação e representam uma parte fundamental de cada uma delas. Sendo assim, é muito importante conhecer a sintaxe e o funcionamento dessas estruturas.

CRIANDO GUI'S

Tkinter é uma biblioteca da linguagem Python que acompanha a instalação padrão e permite desenvolver interfaces gráficas. Isso significa que qualquer computador que tenha o interpretador Python instalado é capaz de criar interfaces gráficas usando o Tkinter, com exceção de algumas distribuições Linux, exigindo que seja feita o download do módulo separadamente

CRIANDO GUI'S

Um dos motivos de estarmos usando o Tkinter como exemplo é a sua facilidade de uso e recursos disponíveis. Outra vantagem é que é nativo da linguagem Python, tudo o que precisamos fazer é importá-lo no momento do uso, ou seja, estará sempre disponível.

CRIANDO GUI'S

O Tkinter já vem por padrão na maioria das instalações Python, então tudo que temos que fazer é importar a biblioteca. Para importar todo o conteúdo do módulo usamos o seguinte comando:

From Tkinter import *

CRIANDO GUI'S

Vamos começar a montar a interface. Começaremos a escrever nosso primeiro código usando a Listagem 1.

```
class Application:  
def __init__(self, master=None):  
    pass  
root = Tk()Application(root)  
root.mainloop()
```

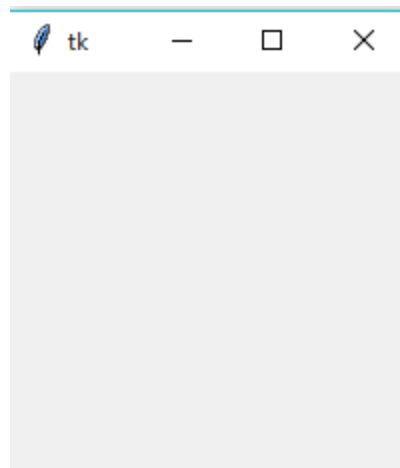
CRIANDO GUI'S

Vamos entender o código: primeiro importamos todos os componentes do módulo Tkinter. Logo após, criamos uma classe chamada `Application`, que no momento ainda não possui nenhuma informação. É nela que criaremos os controles que serão exibidos na tela.

Depois instanciamos a classe `TK()` através da variável `root`, que foi criada no final do código. Essa classe permite que os widgets possam ser utilizados na aplicação.

Em `Application(root)` passamos a variável `root` como parâmetro do método construtor da classe `Application`. E para finalizar, chamamos o método `root.mainloop()` para exibirmos a tela. Sem o event loop, a interface não será exibida.

O código foi salvo em um arquivo `.py` (extensão Python) e depois executado. O resultado obtido é o mesmo da Figura 2.



APROFUNDANDO TKINTER

O módulo Tkinter oferece três formas de trabalharmos com geometria e posicionamento:

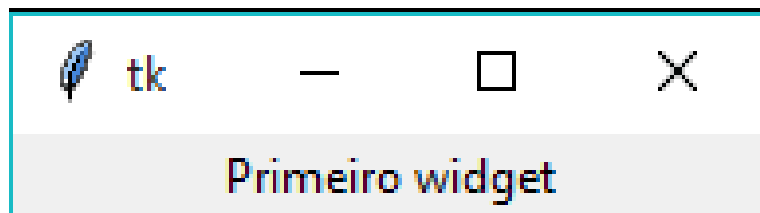
- 1. Pack;**
- 2. Grid;**
- 3. Place.**

Neste artigo abordaremos o gerenciador de posicionamento Pack. Caso um widget não seja aplicado a nenhum gerenciador geométrico, ele continua existindo, mas invisível ao usuário.

Vamos criar nosso primeiro widget, como uma caixa de texto dentro do container principal, como mostra a Listagem 2.

APROFUNDANDO TKINTER

```
from tkinter import *  
class Application:  
def __init__(self, master=None):  
    self.widget1 = Frame(master)  
    self.widget1.pack()  
self.msg = Label(self.widget1, text="Primeiro  
    widget")  
    self.msg.pack ()  
root = Tk()Application(root)  
root.mainloop()
```



APROFUNDANDO TKINTER

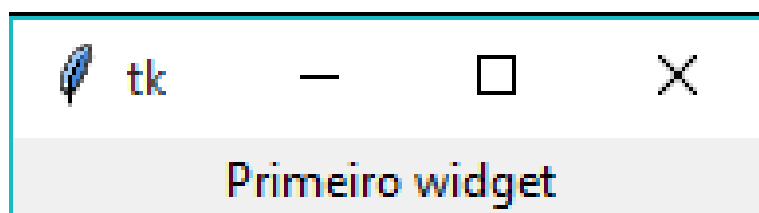
Comparando com a primeira figura, quando não tínhamos definido nenhum widget, a tela diminuiu de tamanho. Isso aconteceu porque como não tinha nenhum conteúdo definido e a tela assumiu o tamanho padrão, mas a partir do momento que definimos um widget dentro da janela top-level (principal), a mesma assumiu o tamanho deste widget.

Veja na linha 4 do código que criamos o primeiro container chamado widget1 e passamos como referência o container pai.

O frame master é o nosso top level, que é o elemento máximo da hierarquia, ou seja, é a nossa janela de aplicação, que contém o título, e botões de maximizar, minimizar e fechar.

Na linha 5 informamos o gerenciador de geometria pack e usamos um widget label para imprimir na tela as palavras "Primeiro widget" e informamos que o container pai é o widget1, que foi passado como parâmetro, conforme a linha 6.

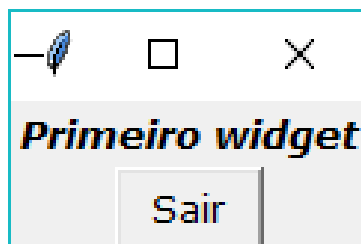
Por fim, exibimos na tela mais uma vez, usamos o gerenciador pack (linha 7).



APROFUNDANDO TKINTER

Agora, vamos adicionar outros widgets à nossa interface, conforme mostra a Listagem 3.

```
class Application:  
def __init__(self, master=None):  
    self.widget1 = Frame(master)  
    self.widget1.pack()  
self.msg = Label(self.widget1, text="Primeiro  
    widget")  
self.msg["font"] = ("Verdana", "10", "italic",  
    "bold")  
self.msg.pack ()  
self.sair = Button(self.widget1)  
self.sair["text"] = "Sair"  
self.sair["font"] = ("Calibri", "10")  
self.sair["width"] = 5  
self.sair["command"] = self.widget1.quit  
self.sair.pack ()  
  
root = Tk()Application(root)  
root.mainloop()
```



APROFUNDANDO TKINTER

Veja que adicionamos um widget do tipo Button e depois atribuímos os valores e estilização, como altura, largura e tipo da fonte a ser exibida.

CAP. BONUS

**Você provavelmente já viu o esse comando dentro do Python
não é mesmo?**

**Mas você sabe o que é o pip no Python ou para que ele serve?
Hoje eu vou te mostrar o que ele faz e como podemos utilizá-lo
dentro do Python.**

CAP. BONUS

Por padrão o pip já vem instalado no seu Python e ele é um gerenciador de pacotes no Python, então ele vai gerenciar as bibliotecas que você vai instalar, desinstalar e atualizar.

Lembrando que uma biblioteca é um pacote de códigos que alguém já criou para que você já possa utilizar.

Então você não vai precisar criar os códigos do zero, então uma biblioteca é um pacote de códigos já pronto para utilizar para criar seus programas.

O pip install vai instalar uma biblioteca específica que você queira, lembrando que as bibliotecas só precisam ser instaladas uma única vez, depois você só vai precisar importar no seu código se for utilizá-la.

Um erro que muitas pessoas que estão iniciando no Python fazem é tentar utilizar o pip install dentro de onde escrevemos o código em Python.

E esse não é o local adequado para a utilização do pip, ele é feito dentro de um terminal ou dentro do prompt de comando do computador.

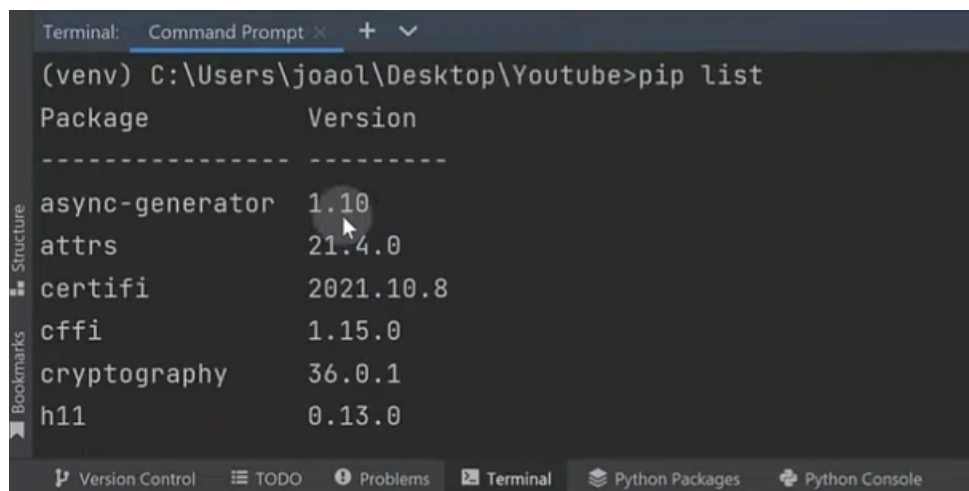
CAP. BONUS

Sabendo disso agora nós vamos entrar em outra parte da aula que é para te mostrar os outros comandos do pip.

O mais comum que você mais deve ter visto é o pip install, mas temos alguns outros comandos também.

Temos o pip uninstall (para desinstalar uma biblioteca), temos o pip install -upgrade (que é para atualizar uma biblioteca).

O próximo comando é o pip list, onde ele vai listar todas as bibliotecas instaladas nesse Python do seu computador.



```
Terminal: Command Prompt × + v
(venv) C:\Users\joaol\Desktop\Youtube>pip list
Package            Version
-----
async-generator    1.10
attrs              21.4.0
certifi            2021.10.8
cffi               1.15.0
cryptography       36.0.1
h11                0.13.0
```



UM LIVRO...

**Com uma abordagem diferente ,
linguagem clara e que permite o
fácil entendimento de um dos
assuntos mais em alta quando o
assunto é programação.**

**Desenvolva apps , sites e mais
com nossa metodologia de ensino**

.

