Guia Definitivo: VBA para Programadores Python

Este guia traduz conceitos de Python para VBA, permitindo que você aproveite sua lógica de programação enquanto aprende a sintaxe da Microsoft.

1. Variáveis e Tipagem

Python: Tipagem dinâmica. Você cria a variável na hora. **VBA:** Tipagem estática (recomendada). Você deve declarar a variável e seu tipo antes de usar.

Declaração Básica

Python:

```
nome = "Pedro"      # str
idade = 25          # int
preco = 10.50       # float
ativo = True        # bool
```

VBA:

```
Dim nome As String
Dim idade As Integer      ' Ou Long (Long é melhor para números grandes)
Dim preco As Double       ' Double é o float do VBA
Dim ativo As Boolean

nome = "Pedro"
idade = 25
preco = 10.50
ativo = True
```

> **Dica Pro:** Sempre use `Option Explicit` no topo do seu arquivo VBA. Isso te obriga a declarar variáveis, evitando erros de digitação (ex: digitar `usaurio` em vez de `usuario` e o VBA criar uma variável nova vazia).

2. Estruturas de Controle (If / Loops)

Condicionais (If / Else)

A lógica é idêntica, muda apenas a sintaxe de fechamento (`End If`).

Python:

```
if idade >= 18:
    print("Maior")
```

```
elif idade > 12:
    print("Adolescente")
else:
    print("Criança")
```

VBA:

```
If idade >= 18 Then
    Debug.Print "Maior"
ElseIf idade > 12 Then
    Debug.Print "Adolescente"
Else
    Debug.Print "Criança"
End If
```

Loops (For / While)

Python (Iterar intervalo):

```
for i in range(1, 6): # 1 a 5
    print(i)
```

VBA:

```
Dim i As Integer
For i = 1 To 5  ' Inclui o 5!
    Debug.Print i
Next i
```

Python (Iterar lista/coleção):

```
frutas = ["Maça", "Banana"]
for fruta in frutas:
    print(fruta)
```

VBA (For Each):

```
Dim fruta As Variant
' Supondo que 'frutas' seja um Array ou Collection
For Each fruta In frutas
    Debug.Print fruta
Next fruta
```

3. Estruturas de Dados Nativas (Sem bibliotecas externas)

Aqui é onde o Python brilha e o VBA sofre um pouco. Sem importar bibliotecas extras (como `Scripting.Dictionary`), ficamos limitados.

Listas vs Arrays/Collections

Python (Listas):

```
lista = []
lista.append("A")
print(lista[0])
```

VBA (Collection): A `Collection` é o mais próximo de uma lista dinâmica nativa.

```
Dim lista As New Collection
lista.Add "A"
lista.Add "B"

' Acesso (VBA começa em 1, não 0, para Collections!)
Debug.Print lista(1) ' Imprime "A"
```

VBA (Arrays): São rápidos, mas chatos de redimensionar.

```
Dim arr(2) As String ' Cria espaços 0, 1 e 2
arr(0) = "A"
arr(1) = "B"

' Redimensionar mantendo dados (custoso)
ReDim Preserve arr(5)
```

4. Funções (Function) vs Procedimentos (Sub)

No Python, tudo é `def`. No VBA, separamos em dois tipos:

1. **Sub (Subroutine):** Executa uma ação, mas **não retorna valor** (Void).
2. **Function:** Executa uma ação e **retorna um valor**.

A. Sub (Ação)

Python:

```
def logar_msg(msg):
    print(f"Log: {msg}")
```

```
logar_msg("Erro")
```

VBA:

```
Sub LogarMsg(msg As String)
    Debug.Print "Log: " & msg
End Sub

' Como chamar uma Sub:
' Opção 1 (Moderna/Simples - Sem parenteses):
LogarMsg "Erro"

' Opção 2 (Usando Call - Parenteses obrigatórios):
Call LogarMsg("Erro")
```

> Por que usar `Call`? É questão de estilo. O `Call` deixa claro que você está executando outra rotina. Sem o `Call`, parece uma instrução nativa. Muitos programadores VBA preferem `Call` para organização.

B. Function (Retorno)

Python:

```
def somar(a, b):
    return a + b

res = somar(10, 20)
```

VBA:

```
Function Somar(a As Integer, b As Integer) As Integer
    ' Em VBA, não existe "return".
    ' Você atribui o valor ao NOME da função.
    Somar = a + b
End Function

Dim res As Integer
res = Somar(10, 20)
```

5. Tratamento de Erros (Try/Except vs On Error)

VBA não tem blocos `try/except` estruturados. Usamos `GoTo` (sim, o temido GoTo).

Python:

```
try:
    x = 1 / 0
except Exception as e:
    print(f"Erro: {e}")
finally:
    print("Fim")
```

VBA:

```
Sub TesteErro()
    On Error GoTo TratarErro ' Inicia o "try"

    Dim x As Double
    x = 1 / 0

    ' Se der certo, pula o tratamento e vai pro final
    GoTo Fim

TratarErro: ' Bloco "except"
    Debug.Print "Erro: " & Err.Description
    Resume Fim ' Opcional: Vai para o bloco Fim

Fim: ' Bloco "finally"
    Debug.Print "Fim"
End Sub
```

6. Classes (POO)

VBA tem classes, mas são mais verbosas.

- Crie um arquivo novo do tipo "Módulo de Classe".
- Nomeie o arquivo como `Pessoa` (No painel de propriedades F4).

Python:

```
class Pessoa:
    def __init__(self, nome):
        self.nome = nome

    def falar(self):
        print(f"{self.nome} diz oi")

p = Pessoa("Ana")
p.falar()
```

VBA (Arquivo Classe: Pessoa.cls):

```
' Variável privada (encapsulamento é forte no VBA)
Private pNome As String

' Propriedade (Get/Set) - Equivalente ao @property
Public Property Let Nome(valor As String)
    pNome = valor
End Property

Public Property Get Nome() As String
    Nome = pNome
End Property

' Método
Sub Falar()
    ' Me é o "self" do VBA
    Debug.Print Me.Nome & " diz oi"
End Sub
```

VBA (No Módulo Principal):

```
Sub TestarClasse()
    Dim p As New Pessoa ' Instancia
    p.Nome = "Ana"
    p.Falar
End Sub
```

7. Passagem de Parâmetros: ByVal vs ByRef (IMPORTANTE!)

Isso pega muita gente do Python.

- **ByVal (Por Valor):** Envia uma cópia da variável. Se a função mudar o valor, a original **não muda**. (Padrão em muitos casos modernos).
- **ByRef (Por Referência):** Envia o endereço da memória. Se a função mudar o valor, a original **MUDA. Esse é o padrão do VBA se você não especificar!**

```
Sub Alterar(ByRef numero As Integer)
    numero = 50
End Sub

Sub Teste()
    Dim n As Integer
    n = 10
    Call Alterar(n)
    Debug.Print n ' Vai imprimir 50!
End Sub
```

> **Regra de ouro:** Sempre escreva `ByVal` nos argumentos de funções, a menos que você realmente queira alterar a variável original.

8. Organização de Código: Sub Principal e Call

Em automações grandes, não colocamos tudo em um bloco só. Criamos uma "Orquestração".

Estrutura Recomendada:

```
' Módulo: MainModule

Sub Main()
    ' Esta é a função que você aperta o "Play"
    On Error GoTo ErroGeral

    Application.ScreenUpdating = False ' Desativa tela piscando (otimização)

    ' Variáveis globais/de configuração
    Dim dataHoje As String
    dataHoje = Format(Date, "yy_mm_dd")

    ' 1. Passo: Baixar Arquivos (Chamando nosso código anterior)
    ' Note o Call: Deixa claro que é uma etapa separada
    Call ExecutarDownloadAutenticado

    ' 2. Passo: Processar Dados
    Call ProcessarDados(dataHoje)

    ' 3. Passo: Gerar Relatório
    Call GerarRelatorio

    MsgBox "Processo concluído!", vbInformation

Sair:
    Application.ScreenUpdating = True
    Exit Sub

ErroGeral:
    MsgBox "Erro fatal: " & Err.Description, vbCritical
    Resume Sair
End Sub

Sub ProcessarDados(data As String)
    Debug.Print "Processando dados do dia " & data
    ' Lógica aqui...
End Sub

Sub GerarRelatorio()
    Debug.Print "Gerando PDF..."
    ' Lógica aqui...
End Sub
```