



PARADIGMAS DE LINGUAGENS DE PROGRAMAÇÃO EM PYTHON

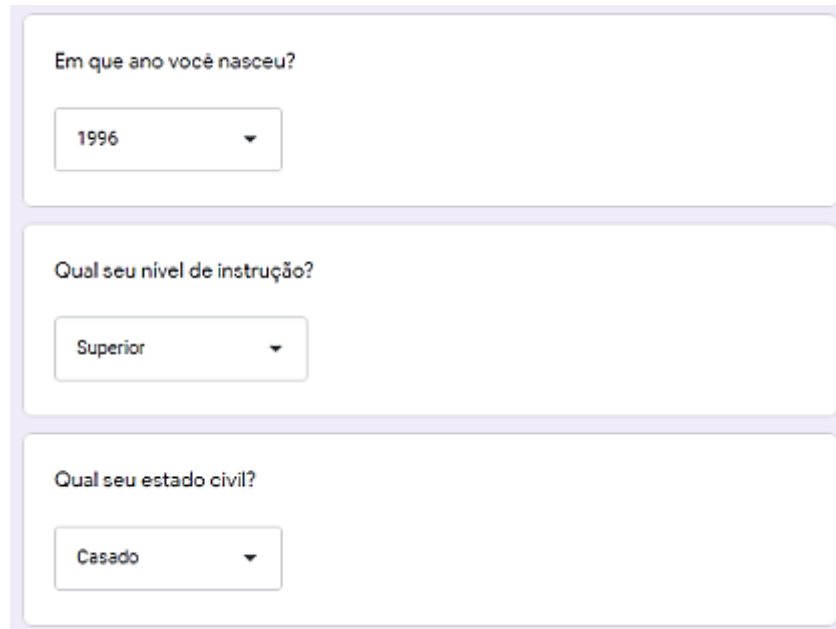
SUBPROGRAMAS, BLOCOS E ESCOPO DE VARIÁVEIS EM PYTHON

Prof. Dr. Daniel Caetano

2021 - 2

Compreendendo o problema

- **Situação:** diversos contadores de repetição
 - Exemplo: formulário web
 - “Criar” as alternativas de cada uma das listas suspensas?



Em que ano você nasceu?

1996 ▼

Qual seu nível de instrução?

Superior ▼

Qual seu estado civil?

Casado ▼

- Que nome daria para cada contador?



<https://www.menti.com/>

Compreendendo o problema

- **Situação:** diversos contadores de repetição
 - Exemplo: formulário web
 - “Criar” as alternativas de cada uma das listas suspensas?



The image shows a screenshot of a web form with three dropdown menus. The first dropdown is labeled 'Em que ano você nasceu?' and has '1996' selected. The second dropdown is labeled 'Qual seu nível de instrução?' and has 'Superior' selected. The third dropdown is labeled 'Qual seu estado civil?' and has 'Casado' selected. The form is enclosed in a light purple border.

**Não seria mais
prático usar
sempre o
mesmo nome?**

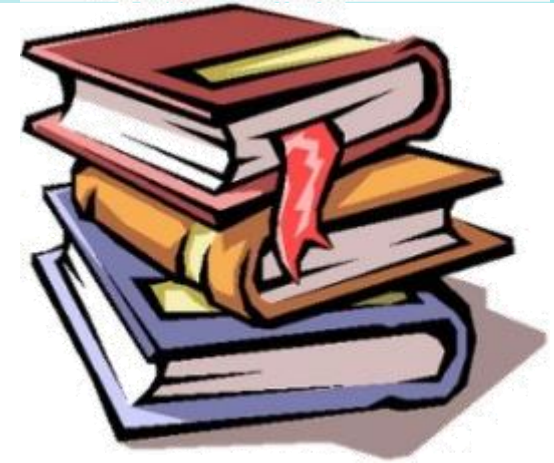
- Que nome daria para cada contador?

Objetivos

- Conceituar blocos e subprogramas
- Compreender o conceito de escopo de variáveis (estáticos e dinâmicos)
- Compreender os escopos locais e globais e sua influência no tempo de vida das variáveis
- Conhecer o conceito de constantes nomeada
- **Atividade Avaliativa C!**



Bibliografia da Aula



Material

Acesso ao Material

Apresentação

<https://www.caetano.eng.br/aulas/2021b/ara0066.php>
(Paradigmas de Programação – Aula 5)

Livro Texto

Capítulo 5, páginas 215 a 226

Aprenda Mais!

- Vídeo: Escopo de Identificadores
<https://youtu.be/Q7mL05tx1tg>
- Vídeo: Escopo Estático e Dinâmico
<https://youtu.be/Zursy21Zzls>

SUBPROGRAMAS



Mentimeter

<https://www.menti.com/>

Prof. Dr. Daniel Caetano

Subprogramas

- Programas que são usados por programas
 - Procedimentos, funções, métodos...
- Trecho de código que recebe um nome

```
def subprograma():  
    print ("Oi!")  
    print ("Sou um subprograma!")  
    print ("Agora eu vou terminar.")
```

- Vejamos o que isso faz...

Subprogramas

- Programas que são usados por programas
 - Procedimentos, funções, métodos...
- Trecho de código que recebe um nome

```
def subprograma():  
    print ("Oi!")  
    print ("Sou um subprograma!")  
    print ("Agora eu vou terminar.")
```

```
subprograma()
```

- E agora...?

Subprogramas

- Pode ser executado várias vezes!

```
def subprograma():  
    print ("Oi!")  
    print ("Sou um subprograma!")  
    print ("Agora eu vou terminar.")
```

```
subprograma()  
subprograma()
```

- Qual o resultado?

Subprogramas

- Pode haver código antes e/ou depois...

```
print ("Início do programma principal")
```

```
def subprograma():  
    print ("Oi!")  
    print ("Sou um subprograma!")  
    print ("Agora eu vou terminar.")
```

```
subprograma()
```

```
print ("Fim do programa principal")
```

- Qual o resultado?

Subprogramas

- Posso fazer isso?

```
subprograma()
```

```
def subprograma():  
    print ("Oi!")  
    print ("Sou um subprograma!")  
    print ("Agora eu vou terminar.")
```

- Qual o resultado?

Subprogramas

- Posso fazer isso?

```
subprograma()
```

```
def subprograma():  
    print ("Oi!")  
    print ("Sou um subprograma!")  
    print ("Agora eu vou terminar.")
```



<https://www.menti.com/>

Em Python, só posso usar um subprograma depois que ele foi criado... De cima pra baixo!



BLOCOS

Blocos

- São trechos de código delimitados
- Blocos, em geral, não são nomeados
 - Só são executados “na sequência”
- Exemplo: estrutura de decisão

```
x = 50
```

```
if x < 10:
```

```
    print ("Bloco do if!")  
    print ("O valor de X é menor que 10!")
```

```
else:
```

```
    print ("Bloco do else!")  
    print ("O valor de X não é menor que 10!")
```

```
print ("Fim do programa!")
```

- O que isso faz?

Blocos

- São trechos de código delimitados
- Blocos, em geral, não são nomeados
 - Só são executados “na sequência”
- Exemplo: estrutura de decisão completa

```
x = 50
```

```
if x < 10:
```

```
    print ("O valor de X é menor que 10!")
```

```
elif x < 50:
```

```
    print ("O valor de X é maior ou igual que 10 e menor que 50!")
```

```
else:
```

```
    print ("O valor de X não é menor que 10!")
```

```
print ("Fim do programa!")
```

Composição de regras com **and** e **or**

Blocos

- São trechos de código delimitados
- Blocos, em geral, não são nomeados
 - Só são executados “na sequência”
- Exemplo: estrutura de repetição

```
x = 1
```

```
while x <= 10:
```

```
    print ("Bloco do while", x, "!")
```

```
    x = x + 1
```

```
print ("Fim do programa!")
```

- O que isso faz?

O QUE É ESCOPO?



Mentimeter

<https://www.menti.com/>

Prof. Dr. Daniel Caetano

Escopo

- Escopo:
 - Área de interesse, foco.
- Em que isso se aplica em programação?



Escopo: Variáveis/Identificadores

- Qual você acha que será o resultado?

A

```
x = 1
def sub():
    print(x)
sub()
```

B

```
x = 1
def sub():
    x = 2
sub()
print(x)
```

C

```
def sub():
    x = 1
sub()
print(x)
```



Mentimeter

<https://www.menti.com/>

Escopo: Variáveis/Identificadores

- Qual você acha que será o resultado?

A

```
x = 1
def sub():
    print(x)
sub()
```

B

```
x = 1
def sub():
    x = 2
sub()
print(x)
```

C

```
def sub():
    x = 1
sub()
print(x)
```

Para entender o porquê, é preciso compreender o conceito de escopo dos identificadores!

Escopo de Variáveis

- Região do programa em que variável “vale”
 - Onde se pode acessar um valor pelo identificador
- O escopo pode ser, dependendo da linguagem:
 - Escopo Estático
 - Maioria das linguagens
 - Escopo Dinâmico
 - Perl, Lisp...



Escopo Estático

- Regra geral: de cima para baixo
 - Identificador só existe após ser declarado* (C89?!)
 - Explícita ou implicitamente

```
int main() {  
    int x;  
    x = 1;  
    cout << x << endl;  
}
```



```
x = 1  
print(x)
```



```
int main() {  
    x = 1;  
    int x;  
    cout << x << endl;  
}
```



```
print(x)  
x = 1
```



Escopo Estático – Repetição

- Proibido repetir identificador no mesmo escopo
 - Para linguagens que exige declaração explícita

```
int main() {  
    int x = 1;  
    int x = 2;  
    cout << x << endl;  
}
```



```
x = 1  
x = 2  
print(x)
```



Escopo Estático – Hierarquia

- Hierarquia inclui blocos e subprogramas
 - Quem está dentro de quem?

```
x = 1
if x < 10:
    print(x)
```

```
int main() {
    int x = 1;

    if (x < 10) {
        cout << x << endl;
    }
}
```

```
x = 1
sub():
    y = 2
    if x < 10:
        print(x+y)

sub()
```


Escopo Estático – Hierarquia

- Pela hierarquia, o que se espera nesses casos?

A

```
x = 1
def sub():
    y = 2
    if x < 10:
        print(x+y)
        z = x+y

sub()
print(z)
```

B

```
x = 1
def sub():
    y = 2
    if x < 10:
        print(x+y)

x = 2
sub()
```

Mentimeter

<https://www.menti.com/>

Escopo Dinâmico

- Hierarquia de chamada
 - Quem chamou quem?

```
$x = 1;
sub sub1 {
    print "$x";
}
sub sub2 {
    $x = 2;
    sub1();
}
sub1();
```

1

Perl

```
$x = 1;
sub sub1 {
    print "$x";
}
sub sub2 {
    $x = 2;
    sub1();
}
sub2();
```

```
x = 1
def sub1():
    print(x)
def sub2():
    x = 2
    sub1()
sub2()
```

1

Python

2

Escopo Dinâmico – Trade Offs

- Vantagem:
 - Mais “limpo” na chamada de subprogramas.
- Desvantagens:
 - Impossível checar tipos em tempo de compilação
 - Difícil de ler e depurar (conhecer a ordem!)
 - Pode ser mais lento para acessar valores.





ESCOPOS EM LINGUAGENS DE ESCOPO ESTÁTICO

Linguagens de Escopo Estático

- Em geral, falamos em dois tipos de escopo:
 - Escopo Global
 - Escopo Local



Escopo Global



- Variáveis/Identificadores:
 - Declarados fora de qualquer função ou bloco
 - Valem em qualquer parte do programa*

```
x = 1  
def sub():  
    print(x)  
    if x < 10:  
        print(x)  
  
sub()  
print(x)
```

```
#include <iostream>  
using namespace std;  
  
int x = 1;  
int main() {  
    cout << x << endl;  
    if (x < 10) {  
        cout << x << endl;  
    }  
}
```

Escopo Global



- Alterar essas variáveis tem efeito global
 - Mas podem exigir algum tipo de “artifício”

```
#include <iostream>
using namespace std;

int x = 1;
int main() {
    cout << x << endl;
    x = x + 1;
    if (x < 10) {
        cout << x << endl;
        x = x + 1;
    }
    cout << x << endl;
}
```

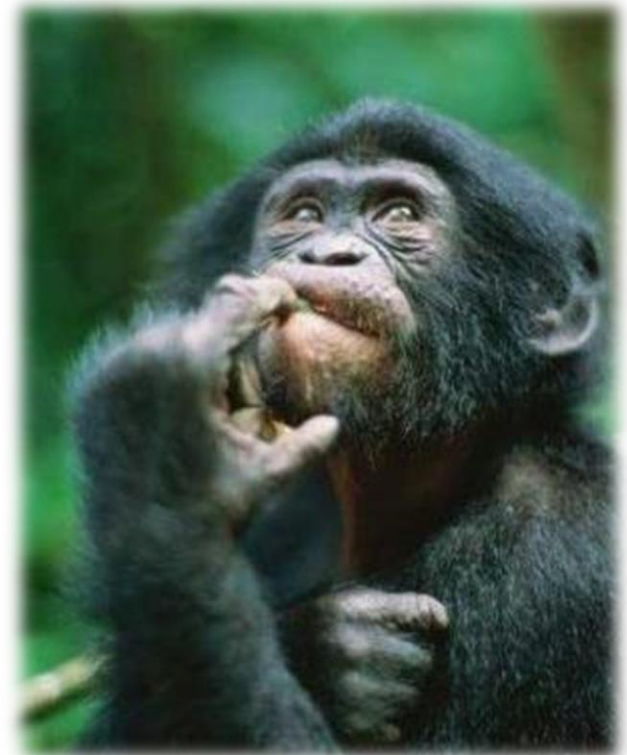
```
x = 1
def sub():
    global x
    print(x)
    x = x + 1
    if x < 10:
        print(x)
        x = x + 1

sub()
print(x)
```

Escopo Global



- Existe em praticamente qualquer linguagem...
 - ... **menos em Java.**
- Por quê?
 - Tornam difícil compreender o programa
 - Erro em uma parte do programa “vaza” para outras
 - Conflito de nomenclatura
 - ...



Escopo Local



- Variáveis/Identificadores:
 - Declarados em um subprograma ou bloco*
 - Valem hierarquicamente a partir de sua criação

```
def sub():  
    x = 1  
    print(x)  
    if x < 10:  
        x = x + 1  
        print(x)  
  
sub()  
print(x) # causa um erro!
```

Escopo Local



- Variáveis/Identificadores:
 - Declarados em um subprograma ou bloco*
 - Atenção: em Python, if/while não cria escopo novo!

```
def sub():  
    x = 1  
    print(x)  
    if x < 10:  
        y = 2  
        print(y)  
    print(y)  
  
sub()
```

```
#include <iostream>  
using namespace std;  
int main() {  
    int x = 1;  
    cout << x << endl;  
    if (x < 10) {  
        int y = 2;  
        cout << y << endl;  
    }  
    cout << y << endl; // Erro!  
}
```

Escopo Local



- No Python, com subprogramas aninhados...
 - E se eu quiser mudar a variável declarada “acima”?

```
def sub():  
    x = 1  
    print(x)  
    def sub2():  
        x = x + 1 # Erro!  
        print(x)  
    sub2()  
sub()
```

```
def sub():  
    x = 1  
    print(x)  
    def sub2():  
        nonlocal x  
        x = x + 1  
        print(x)  
    sub2()  
sub()
```

Escopo Local



- Em C++, C# e Java...
 - A declaração do laço **for** é um novo escopo!

```
#include <iostream>
using namespace std;

int main() {
    for (int x = 1; x < 10; x++) {
        cout << x << endl;
    }
    cout << x << endl; // Erro!
}
```



OCULTAMENTO DE VARIÁVEIS

Ocultamento de Variáveis

- Lá atrás, dissemos:
 - Proibido repetir identificador no mesmo escopo
 - Mas podemos repetir em escopos diferentes?

```
int main() {  
    int x = 1;  
    int x = 2;  
    cout << x << endl;  
}
```



```
#include <iostream>  
using namespace std;  
int main() {  
    int x = 1;  
    cout << x << endl;  
    if (x < 10) {  
        int x = 2;  
        cout << x << endl;  
    }  
    cout << x << endl;  
}
```



Ocultamento de Variáveis

- Lá atrás, dissemos:
 - Proibido repetir identificador no mesmo escopo
 - Mas podemos repetir em escopos diferentes?

```
int main() {  
    int x = 1;  
    int x = 2;  
    cout << x << endl;  
}
```



```
#include <iostream>  
using namespace std;  
int main() {  
    int x = 1;  
    cout << x << endl;  
    if (x < 10) {  
        int x = 2;  
        cout << x << endl;  
    }  
    cout << x << endl;  
}
```



Como acessar o valor do x global aqui?

Algumas linguagens fornecem meios, dependendo do caso!

Ocultamento de Variáveis

- Nem todas as linguagens permitem...
 - Ocultamento dentro de blocos... como C# e Java

```
public class Exemplo {  
    public static void main( String[] args) {  
        int x = 1;  
        System.out.println(x);  
        if (x < 10) {  
            int x = 2; // Erro!  
            System.out.println(x);  
        }  
        System.out.println(x);  
    }  
}
```




ESCOPO E O TEMPO DE VIDA DAS VARIÁVEIS



Mentimeter

<https://www.menti.com/>

Tempo de Vida de uma Variável

- Vida da variável?
 - Valor preservado na memória
 - Mantém-se acessível pelo identificador.
- Regra básica:
 - Existe enquanto seu escopo está em contexto
 - Escopo em contexto?
 - Está em execução
 - Um subprograma chamado está em execução.



Tempo de Vida de uma Variável

- Regra Geral aplicada:
 - Variáveis locais:
 - “Existem”: subprograma ou bloco está em contexto
 - Variáveis Globais
 - “Existem”: todo o tempo de vida da aplicação ou objeto



- O programador pode interferir!

Controlando o Tempo de Vida

- Tornando variáveis “permanentes”
 - Em subprogramas (C, C++, Java) ou objetos*: **static**
 - São criadas/inicializadas na primeira vez que usadas
 - Passam a durar o tempo de vida da aplicação
 - Valor permanece entre chamadas do subprograma



Controlando o Tempo de Vida

- Alocação de espaços “temporários”
 - Algumas linguagens permitem **alocação explícita**
 - Declara-se como “referência” ou “ponteiro”
 - Apenas identificador, sem reservar memória no processo.
 - Memória alocada por comando (ex.: malloc, new...)
 - A liberação do espaço pode variar:
 - Manual (C, C++...): comandos (ex.: free, delete...)
 - Automática (Java, Python, C#...): coleta de lixo.



VALORES IMUTÁVEIS



Mentimeter

<https://www.menti.com/>

Prof. Dr. Daniel Caetano

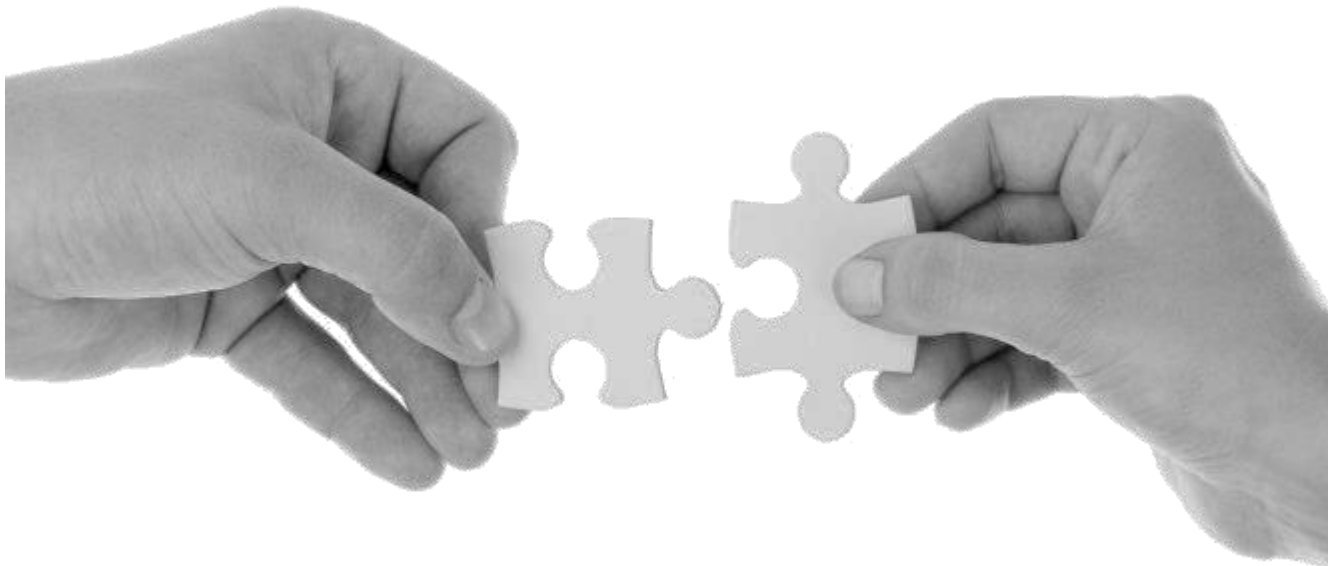
Imutabilidade



- Hein?
 - Se é “variável”, como pode ser “imutável”?
 - Simples: não é variável, é uma **constante nomeada**
- Para quê?
 - Para poder escrever PI ao invés de 3.1415926535...
- Em tudo se parece com uma variável...
 - Mas valor é definido uma vez e se torna fixo
 - Existência é a do escopo em que é declarado
 - Em geral, global

Imutabilidade – Tipos

- Em função da vinculação, há dois tipos
 - Vinculação estática
 - Valor atribuído à constante em tempo de compilação.
 - Vinculação dinâmica
 - Valor atribuído à constante em tempo de execução.



Imutabilidade – Tipos

- Cada linguagem tem seu “esquema”
 - Java: **final** (vinculação estática)
 - C/C++: **const** (vinculação estática ou dinâmica)
 - C#: **const** (v. estática) e **readonly** (v. dinâmica)

```
final double PI = 3.1415926535;
```

```
// Estático  
const double PI = 3.1415926535;
```

```
// Dinâmico  
double x = 1.0;  
const double PI = 4*atan(x);
```

```
// Estático  
const double PI = 3.1415926535;
```

```
// Dinâmico  
double x = 1.0;  
readonly double PI = 4*atan(x);
```

Imutabilidade – Exemplos

```
public class HelloWorld{  
    public static void main(String []args) {  
        final double PI = 3.1415926535;  
        System.out.println("Hello World " + PI);  
        PI++; // Erro!  
    }  
}
```

Imutabilidade – Exemplos

```
#include <iostream>
#include <math.h>
using namespace std;

int main() {
    // Estático
    const double PI = 3.1415926535;

    // Dinâmico
    double x = 1.0;
    const double PI2 = 4*atan(x);

    cout << "Hello World " << PI << " " << PI2 << endl;
    PI++;      // Erro!
    PI2++;    // Erro!
}
```



ATIVIDADE

Atividade 1

- Exemplo
- Faça um Programa que verifique se uma letra digitada é "F" ou "M". Conforme a letra escrever: F - Feminino, M - Masculino, Sexo Inválido.

Atividade 2

- Individual, em Python – 10 minutos
- Faça um programa para a leitura de duas notas parciais de um aluno. O programa deve calcular a média alcançada por aluno e apresentar:
 - A mensagem "Aprovado", se a média alcançada for maior ou igual a sete;
 - A mensagem "Reprovado", se a média for menor do que sete;
 - A mensagem "Aprovado com Distinção", se a média for igual a dez.

Atividade 3

- Individual, em Python – 10 minutos
- Faça um Programa que leia três números e mostre o maior e o menor deles.

Atividade 4

- Individual, em Python – 10 minutos
- Faça um programa que peça uma nota, entre zero e dez. Mostre uma mensagem caso o valor seja inválido e continue pedindo até que o usuário informe um valor válido.

Atividade 5

- Individual, em Python – 10 minutos
- Faça um programa que imprima na tela apenas os números ímpares entre 1 e 50.



ENCERRAMENTO

Resumo e Próximos Passos

- Subprogramas e blocos
 - Escopos Estáticos e Dinâmicos, Locais e Globais
 - Tempo de Vida das Variáveis
 - Constantes Declaradas
 - **Pós Aula:** Aprenda Mais, Pós Aula e Desafio!
 - No padlet: <https://padlet.com/djcaetano/paradigmas>
-
- Tipos nativos de dados
 - E iniciando com vetores e enumerações



PERGUNTAS?