

LABORATÓRIOS DE PROGRAMAÇÃO

Funções e Bibliotecas

Aula 4

Prof^a. Kamilla Dória

TÓPICOS

- Escopo de Variáveis
- Otimização de Código
- Biblioteca de Funções



ESCOPO DE VARIÁVEIS

ESCOPO DE VARIÁVEIS

- Indica sua visibilidade, ou seja, a partir de onde, no código, a variável é acessível;
- Há dois escopos de variáveis:
 - Escopo Local
 - Escopo Global

- Uma variável local (criada dentro de uma função) existe apenas dentro da função onde foi declarada;
- As variáveis locais são inicializadas a cada nova chamada à função.
- Desta forma, não é possível acessar seu valor fora da função onde ela foi declarada.
- Para que possamos interagir com variáveis locais, passamos parâmetros e retornamos valores nas funções.

ESCOPO LOCAL

- Exemplo:

Escopo de n

```
def fatorial(n):  
    fat = 1  
    for i in range(2, n + 1):  
        fat *= i  
    return fat
```

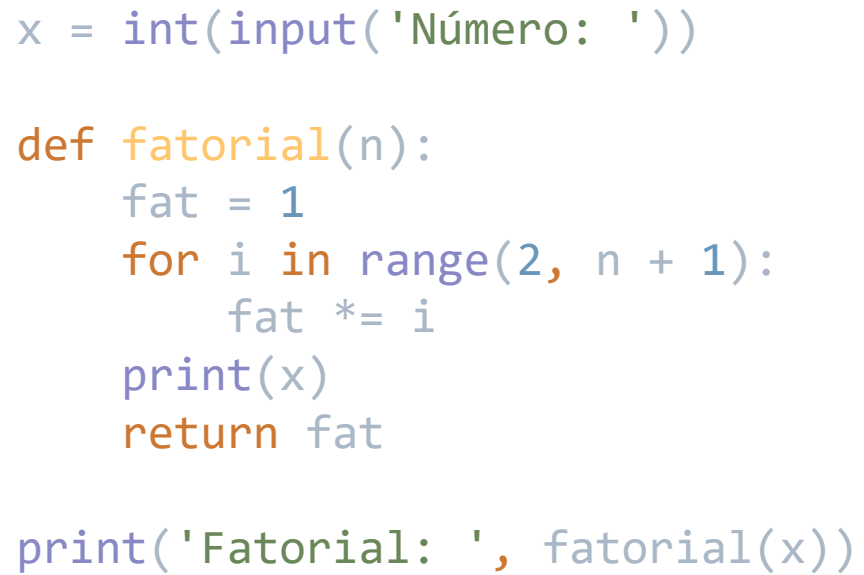
Escopo de x

```
x = int(input('Número: '))  
print('Fatorial: ', fatorial(x))
```

- É declarada (criada) fora das funções e pode ser acessada por todas as funções presentes no módulo onde é definida;
- Também pode ser acessada por outros módulos, caso eles importem o módulo onde a variável foi definida;
- Utilizada também para o armazenamento de valores constantes no programa, acessíveis a todas as funções.

- Exemplo:

Escopo de x



```
x = int(input('Número: '))

def fatorial(n):
    fat = 1
    for i in range(2, n + 1):
        fat *= i
    print(x)
    return fat

print('Fatorial: ', fatorial(x))
```


- Sempre devemos tomar cuidado ao alterar o valor de uma global dentro de uma função. Se for atribuído valor a ela, será na verdade criada uma nova variável, local, com o mesmo nome da global.

Escopo de n

Escopo de n

```
def fatorial(n):  
    fat = 1  
    for i in range(2, n + 1):  
        fat *= i  
    return fat  
  
n = int(input('Número: '))  
print('Fatorial: ', fatorial(n))
```

ALTERAÇÃO DE ARGUMENTOS

- Para realizar alteração de valor de um argumento deve-se utilizar funções que retornem o valor que deseja que seja alterado.

```
def inverte_sinal(numero):  
    return -numero  
  
x = 10  
x = inverte_sinal(x)  
print(x)
```

```
"AlteraçãoArgumentos.py"  
-10
```

```
Process finished with exit code 0
```

EXERCÍCIO 1

Faça um programa que converta da notação de 24 horas para a notação de 12 horas. Por exemplo, o programa deve converter 14:25 em 2:25 P.M. A entrada é dada em dois inteiros.

Deve haver pelo menos duas funções: uma para fazer a conversão e uma para a saída. Registre a informação A.M./P.M. como um valor 'A' para A.M. e 'P' para P.M. Assim, a função para efetuar as conversões terá um parâmetro formal para registrar se é A.M. ou P.M.

Inclua um loop que permita que o usuário repita esse cálculo para novos valores de entrada todas as vezes que desejar.



OTIMIZAÇÃO DE CÓDIGO

OTIMIZAÇÃO DE CÓDIGO

- Consiste em fazer programas de forma que a quantidade de instruções executadas para resolver o problema seja a mínima possível;
 - Em nome da otimização não se deve abrir mão da clareza;
- O uso de funções eficientes é uma das formas de otimização.

OTIMIZAÇÃO DE CÓDIGO

- Exemplo: função para verificar se o número é primo:

```
#verifica se o número é primo validando se os números menores que ele e maior que 1 sao seus divisores (resto div = 0)
def é_primo(numero):
    primo = True
    for divisor in range(2, numero):
        if numero % divisor == 0:
            primo = False
    return primo
```

OTIMIZAÇÃO DE CÓDIGO

- Código otimizado:

```
#verifica se o número é primo validando se os números menores que ele e maior que 1 são seus divisores (resto div = 0)
def é_primo(numero):
    for divisor in range(2, numero//2+1): #não é necessário percorrer todos os números, apenas até a metade
                                           #2+1 para evitar que o número não seja inteiro
        if numero % divisor == 0:
            return False #não é necessário validar todos, ao encontrar um caso Falso, já pode retornar
    return True
```

EXERCÍCIO 2

Analise o código do Exercício 1 e verifique o que pode ser otimizado.



BIBLIOTECA DE FUNÇÕES

BIBLIOTECA DE FUNÇÕES

- Conjunto de funções que podem ser chamadas por outros programas;
- Organizadas em arquivos dentro do projeto;
- Separação entre o programa principal e funções (melhor organização do código).

- Exemplo:
 - Arquivo python para o programa: programa.py
 - Arquivo python para as funções: funções.py
- Sugestão: utilizar nomes curtos e que remetam ao objetivo do arquivo.

BIBLIOTECA DE FUNÇÕES

funcoes.py

```
def é_primo(numero):  
    for divisor in range(2, numero//2+1):  
        if numero % divisor == 0:  
            return False  
    return True
```

programa.py

```
import funcoes  
  
print(funcoes.é_primo(int(input('Digite um número: '))))
```

EXERCÍCIO 3

Implemente um arquivo chamado `funções.py` com duas funções: uma para ver se o número é par ou ímpar e outra para calcular a média entre dois números.

Na sequência crie um arquivo `programa.py` que implemente o uso dessas duas funções.



OBRIGADA!