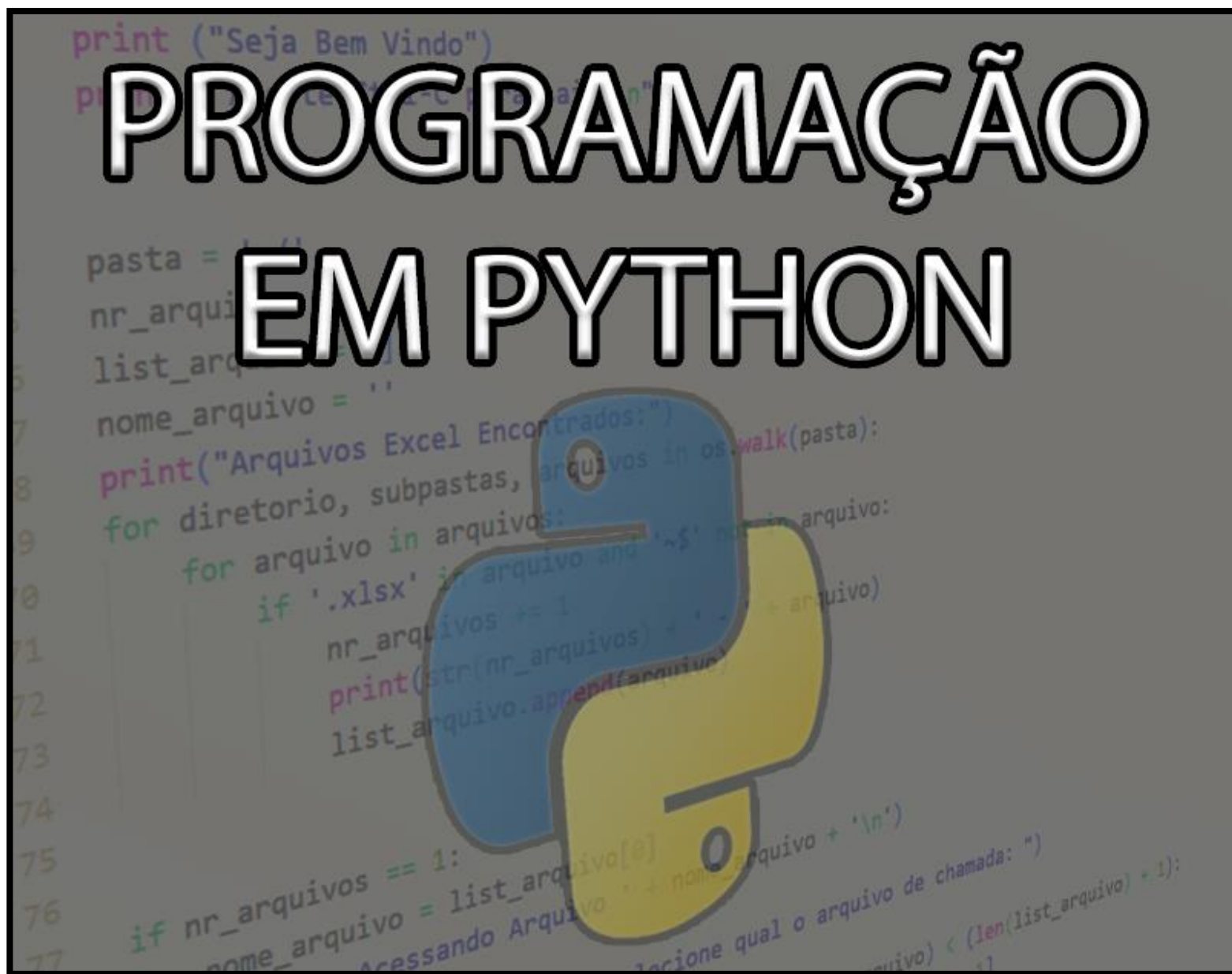


PROGRAMAÇÃO

EM PYTHON



Prof. Me Jean Justino

Função e Parâmetros

Até o momento, vimos algumas formas simples de utilizar uma função, porém existem várias maneiras de utilizar a mesma função, isto é, passando argumentos opcionais da função:

Exemplo: a função **print**:

```
print(*objects, sep=' ', end='\n', file=None, flush=False)
```

É obrigatório passar o argumento **objects****, sendo este argumento qualquer tipo de **objeto(s)** em Python:

```
print('teste')
```

```
print(1, 'teste', 'abc', 12.5, True)
```

** Vale lembrar que não passar nenhum objeto ao print é o mesmo que passar um objeto **None**, por isso não dá erro

Função e Parâmetros

```
print(*objects, sep=' ', end='\n', file=None, flush=False)
```

O argumento **sep** é um argumento opcional do tipo **nomeado**, ou seja, para passar este argumento é necessário escrever o nome dele.

Este argumento serve para modificar o separador entre objetos:

```
print('teste', '123', sep='----->')
```

Vai imprimir: **teste----->123**

O argumento **end** é um argumento opcional do tipo **nomeado** que tem como valor padrão o **ENTER (\n)**. Então sempre que você utiliza a função print, o sistema coloca um ENTER no final se você não alterar este parâmetro.

Mas podemos alterar o parâmetro para ele se comportar de outra forma:

```
print('teste', '123', sep='___+++___', end='__FINAL__')
```

```
print('outra função')
```

Este código irá exibir: **teste___+++___123__FINAL__outra função**

Função e Parâmetros



Para consultar todos os parâmetros de uma função, basta acessar a documentação oficial do Python (ou exemplos de utilização na internet).

Documentação Oficial: <https://docs.python.org/pt-br/3/library/functions.html>

Documentação Função Print: <https://docs.python.org/pt-br/3/library/functions.html?highlight=print#print>

Métodos



Um método é uma função que “pertence” a um objeto.

O objeto **str**, por exemplo, têm vários métodos para manipular este tipo de dado:

```
>>> nome = 'daniel'
>>> print(nome.upper())
DANIEL
>>> print(nome.lower())
daniel
>>> print(nome.title())
Daniel
```

Documentação: <https://docs.python.org/pt-br/3/library/stdtypes.html#string-methods>

Métodos



Alguns métodos muito utilizados em **str** são:

- `.upper()` → Retorna uma **str** toda em letra Maiúscula
- `.lower()` → Retorna uma **str** toda em letra Minúscula
- `.title()` → Retorna uma **str** com a primeira letra de cada palavra em Maiúscula
- `.format()` → **Permite formatar uma string de acordo com um padrão**
- `.isalnum()` → Se é Alfanumérico (texto e número) retorna **True**, se não retorna **False**
- `.isalpha()` → Se é Alfabético (texto) retorna **True**, se não retorna **False**
- `.isascii()` → Se existe na tabela ASCII retorna **True**, se não retorna **False**
- `.isdecimal()` → Se é número retorna **True**, se não retorna **False**
- `.replace()` → Retorna uma **str** com as letras/palavras substituídas
- `.split()` → Fatia o texto determinando um caractere de separação e retorna uma lista com cada item
- `.splitlines()` → Fatia um texto por linha e separa cada linha em um item da lista

Formatação de string Old Style: printf-style

Com ele conseguimos alterar a forma de exibir ou salvar uma **str**. Alguns exemplos de como pode ser utilizado:

- Colocando uma reserva de variável (%). Neste método as variáveis são colocadas na ordem que iriam ser inseridas no texto:

```
nome = 'Daniel'
idade = 29
info = 'Seu nome é %s e você tem %d anos' %(nome,idade)
print(info)
```

- Colocando uma reserva nomeada de variável. (%). Agora pode inverter a ordem, mas precisa descrever de qual reserva esta se referindo:

```
nome = 'Daniel'
idade = 29
info = 'Seu nome é %(valor_nome)s e \
você tem %(valor_idade)d anos' %{'valor_nome' : nome, 'valor_idade' : idade}
print(info)
```

Funções

Podemos criar funções em python com ou sem parâmetros e com ou sem retorno.

Para criar uma função simples em python, basta usar a palavra-chave **def** como no exemplo

```
def minha_funcao():  
    print("Comandos da função")  
    print("Pode ter varias linhas de código")  
    print("Dentro de uma mesma função")
```

```
minha_funcao()
```

Para chamar a função para executa-la, basta executar o nome dela com **()** no final.

Funções

Para criar uma função com parâmetros basta passar as informações desejadas dentro do parênteses:

```
def idade_em_2050(idade):  
    print('Sua idade em 2050 será ' + str(2050-2021+idade) + " anos")  
  
idade_em_2050(30)  
idade_em_2050(15)  
idade_em_2050(50)
```

```
Sua idade em 2050 será 59 anos  
Sua idade em 2050 será 44 anos  
Sua idade em 2050 será 79 anos
```

Neste exemplo, quando chamamos a função **idade_em_2050** passamos um argumento numérico (ex: **30**) e na criação da função esse valor numérico é atribuído ao parâmetro **idade**.

Funções

Podemos passar mais de um parâmetro dentro de uma função:

```
def idade_em_2050(nome, idade):  
    print(nome + ', a sua idade em 2050 será ' + str(2050-2021+idade) + " anos")  
  
idade_em_2050('João',30)  
idade_em_2050('Maria',15)  
idade_em_2050('José',50)
```

```
João, a sua idade em 2050 será 59 anos  
Maria, a sua idade em 2050 será 44 anos  
José, a sua idade em 2050 será 79 anos
```

Funções

Também podemos colocar palavras-chave nos parâmetros, assim não precisamos nos preocupar com a ordem dos argumentos passados:

```
def idade_em_2050(idade, nome):  
    print(nome + ', a sua idade em 2050 será ' + str(2050-2021+idade) + " anos")  
  
idade_em_2050(nome = 'João', idade = 30)  
idade_em_2050(idade = 15, nome = 'Maria')  
idade_em_2050(nome = 'José', idade = 50)
```

```
João, a sua idade em 2050 será 59 anos  
Maria, a sua idade em 2050 será 44 anos  
José, a sua idade em 2050 será 79 anos
```

Funções

É possível definir parâmetros com valores padrões (**default**), pra quando não for passado aquele argumento ter um valor pré-definido:

OBS: Nestes casos, na hora de criar a função, coloque os parâmetros que terão valores **default** por último

```
def idade_em_2050(idade, nome = 'Zé Ninguém'):  
    print(nome + ', a sua idade em 2050 será ' + str(2050-2021+idade) + " anos")
```

```
idade_em_2050(nome = 'João', idade = 30)  
idade_em_2050(idade = 15, nome = 'Maria')  
idade_em_2050(nome = 'José', idade = 50)  
idade_em_2050(idade = 20)
```

```
João, a sua idade em 2050 será 59 anos  
Maria, a sua idade em 2050 será 44 anos  
José, a sua idade em 2050 será 79 anos  
Zé Ninguém, a sua idade em 2050 será 49 anos
```

Funções

Podemos definir retorno para funções e salvar o valor retornado ou utiliza-lo diretamente em outra função:

```
def idade_em_2050(idade):  
    return 2050-2021 + idade
```

```
print(idade_em_2050(30))
```

```
idade_jose = idade_em_2050(25)  
print(idade_jose)
```

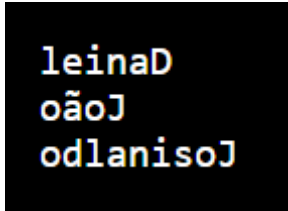


59

54

```
def nome_ao_contrario(nome):  
    nome_invertido = ''  
    for indice, letra in enumerate(nome):  
        nome_invertido += nome[-indice-1]  
    return nome_invertido
```

```
print(nome_ao_contrario('Daniel'))  
print(nome_ao_contrario('João'))  
print(nome_ao_contrario('Josinaldo'))
```



leinaD
oãoJ
odlanisoJ

OBS: A função **enumerate** utilizada no exemplo serve para conseguir retornar o índice em um loop For, neste caso, como o loop For é de uma string, o índice será a posição de cada letra da palavra.

Funções

Podemos chamar uma função dentro da própria função:

```
def teste(letra, quant):  
    if(quant > 0):  
        print(letra*quant)  
        teste(letra, quant - 1)  
  
teste('a', 22)
```

```
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaa  
aaaaaaa  
aaaaaaa  
aaaaaaa  
aaaaaaa  
aaaaa  
aaaaa  
aaaa  
aaaa  
aaa  
aaa  
aa  
aa  
a
```

Formatação de string New Style: .format()

Outra forma de alterar uma **str**, assim como o **printf-style**:

- Pode utilizar somente reservando espaços:

```
nome = 'Daniel'
idade = 29
info = 'Seu nome é {} e você tem {} anos'.format(nome, idade)
print(info)
```

- É possível reservar numericamente os espaços para inserção das variáveis:

```
nome = 'Daniel'
idade = 29
info = 'Seu nome é {1} e você tem {0} anos'.format(idade, nome)
print(info)
```

- É possível reservar com **palavra-chave** o espaço para inserção das variáveis:

```
nome = 'Daniel'
idade = 29
info = 'Seu nome é {valor_nome} e você \
tem {valor_idade} anos'.format(valor_nome = nome, valor_idade = idade)
print(info)
```

Formatação de string: formatando tipos de dados

Ambos os métodos de formatação de uma **str** permitem utilizar formatação de tipos de dados. Exemplo, para formatar um **float** para **2 casas** decimais:

```
print('Eu tenho R$%.2f reais' %(22))  
print('Eu tenho R${:.2f} reais'.format(22))  
  
print('Eu tenho R$%(dinheiro).2f reais' %{'dinheiro':22})  
print('Eu tenho R${dinheiro:.2f} reais'.format(dinheiro = 22))
```

Alguns tipos de formatação só existem no formato **.format()**, por exemplo, para colocar **0 casas decimais** após uma porcentagem:

```
print('Eu ganhei {:.0%} de desconto'.format(0.3))
```

Outro exemplo, para centralizar uma variável:

```
print('aaaaa{: ^20}aaaaa'.format('Python'))
```

Mais informações: https://www.w3schools.com/python/ref_string_format.asp