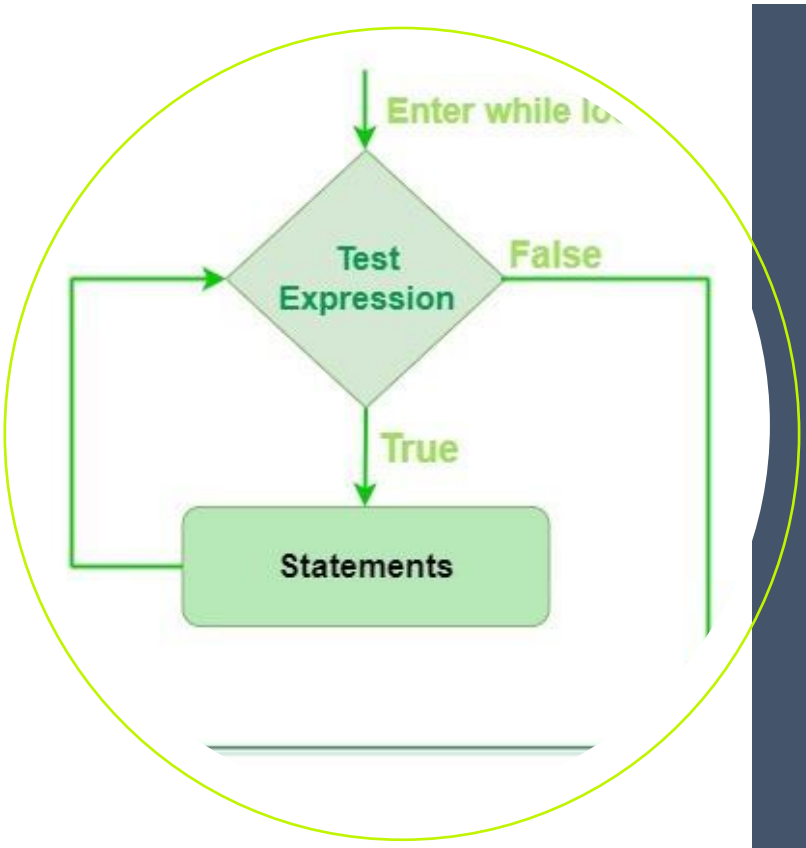
FUNDAMENTOS DE PROGRAMAÇÃO

SUMÁRIO



ENQUANTO

Você quase pode ler o comando while como se fosse inglês. Ele significa, “Enquanto (while) n for maior do que 0, siga exibindo o valor de n e diminuindo 1 do valor de n. Quando chegar a 0, exiba a palavra “Fogo!”. Este tipo de fluxo é chamado de um loop (ou laço) porque o terceiro passo cria um “loop” ou um laço de volta ao topo. Note que se a condição for falsa na primeira vez que entrarmos no loop, os comandos dentro do loop jamais serão executados.

O corpo do loop poderia alterar o valor de uma ou mais variáveis de modo que eventualmente a condição se torne falsa e o loop termine. Se não for assim, o loop se repetirá para sempre, o que é chamado de um loop infinito. Uma fonte de diversão sem fim para os cientistas da computação é a observação de que as instruções da embalagem de xampu, “Lave, enxágue, repita” é um loop infinito.

Uma utilização adequada de while, são para menus em Prompt, onde não sabemos quando o usuário vai encerrar o programa ou acessar uma opção.

WHILE

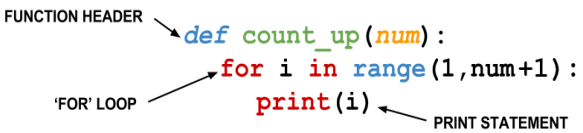
Os computadores são muito utilizados para automatizar tarefas repetitivas. Repetir tarefas idênticas ou similares sem cometer erros é uma coisa que os computadores fazem bem e que as pessoas fazem poorly (mal).

Analise o seguinte algoritmo utilizando o while:

```
n = 5

while n > 0:

    print (n)

    n = n-1

print ("Fogo!")
```

FUNÇÕES

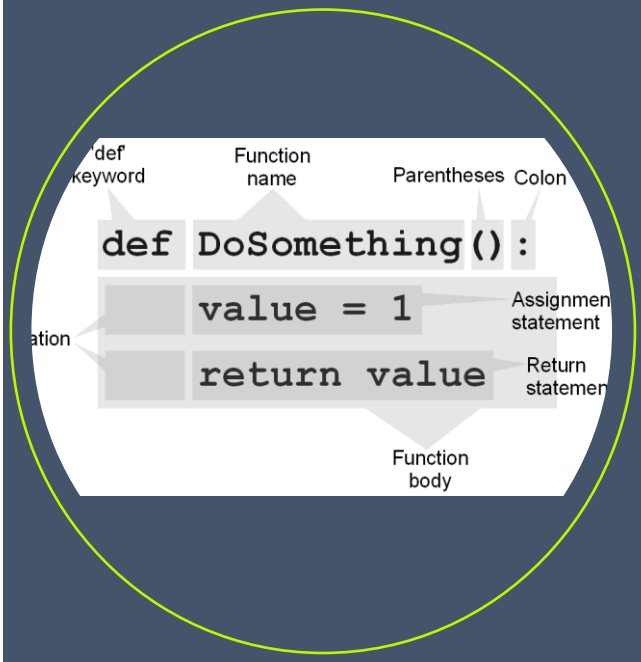
Até aqui, utilizamos somente as funções que vêm com Python, chamadas nativas, mas também é possível adicionar novas funções. Criar funções para resolver seus próprios problemas é uma das coisas mais úteis de uma linguagem de programação de propósito geral. No contexto de programação, função é uma sequência nomeada de instruções ou comandos, que realizam uma operação desejada. Esta operação é especificada numa definição de função. Até agora, as funções que usamos no curso são pré-definidas e suas definições não foram apresentadas. Isso demonstra que podemos usar funções sem ter que nos preocupar com os detalhes de suas definições.



A sintaxe para uma definição de função é:

```
def NOME_DA_FUNCAO(LISTA DE PARAMETROS ) :
    COMANDOS
```

Até agora, as funções que usamos no curso são pré-definidas e suas definições não foram apresentadas.



NOMENCLATURA DE FUNÇÕES

Você pode usar o nome que quiser para as funções que criar, exceto as palavras reservadas do Python. A lista de parâmetros especifica que informação, se houver alguma, você tem que informar para poder usar a nova função.

Uma função pode ter quantos comandos forem necessários, mas eles precisam ser endentados a partir da margem esquerda (quadro espaços). As primeiras funções que vamos mostrar não terão parâmetros, então, a sintaxe terá esta aparência:

```
def novaLinha():
    print ()
```

Esta função é chamada de novaLinha. Os parênteses vazios indicam que ela não tem parâmetros. Contém apenas um único comando, que gera como saída um caractere de nova linha (isso é o que acontece quando você chama a função print sem qualquer argumento).

A sintaxe para a chamada desta nova função é a mesma sintaxe para as funções nativas:

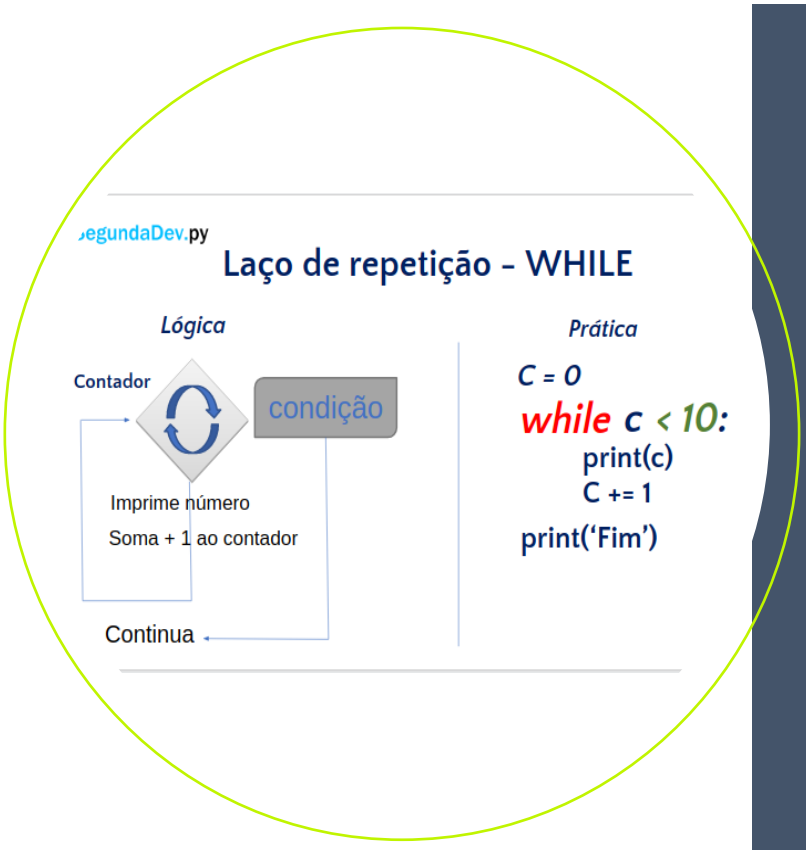
```
print ('Primeira Linha.')
novaLinha()
print ('Segunda Linha.')
```

A saída deste programa é:

```
Primeira Linha.
Segunda Linha.
```

Observe o espaço extra entre as duas linhas. E se quiséssemos mais espaço entre as linhas? Poderíamos chamar a mesma função repetidamente:

```
print ('Primeira Linha.')
novaLinha()
novaLinha()
novaLinha()
print ('Segunda Linha.')
```



ALGUNS COMPORTAMENTOS

Algumas coisas que devem ser observadas sobre este programa:

- Você pode chamar o mesmo procedimento repetidamente. Isso é muito comum, além de útil.
- Você pode ter uma função chamando outra função; neste caso tresLinhas chama novaLinha.

Pode não estar claro, até agora, do que vale o esforço para criar funções - existem várias razões, mas este exemplo demonstra duas delas:

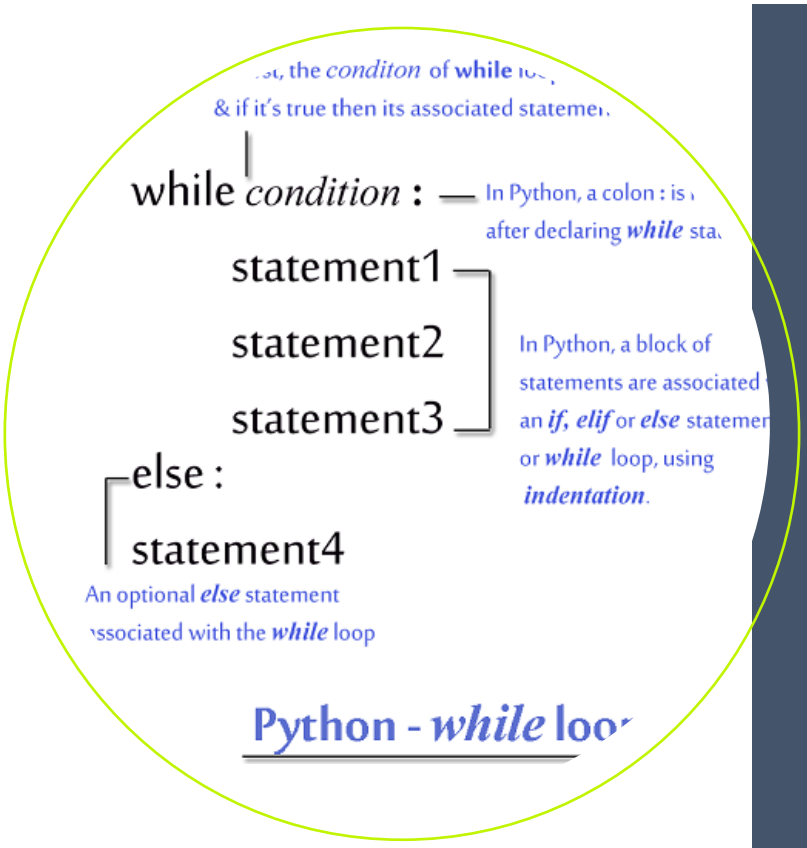
- Criar uma função permite que você coloque nome em um grupo de comandos. As funções podem simplificar um programa ao ocultar a execução de uma tarefa complexa por trás de um simples comando cujo nome pode ser uma palavra em português, em vez de algum código misterioso.
- Criar uma função pode tornar o programa menor, por eliminar código repetido. Por exemplo, um atalho para 'imprimir' nove novas linhas consecutivas é chamar tresLinhas três vezes.

FUNÇÃO CHAMANDO FUNÇÃO

Poderíamos escrever uma nova função chamada tresLinhas, que produzisse três novas linhas.

```
def tresLinhas() :
    novaLinha()
    novaLinha()
    novaLinha()
print ('Primeira Linha.')
tresLinhas()
print ('Segunda Linha.')
```

Esta função contém três comandos, todos com recuo de quatro espaços a partir da margem esquerda. Já que o próximo comando não está endentado, Python reconhece que ele não faz parte da função.



PARÂMETROS E ARGUMENTOS

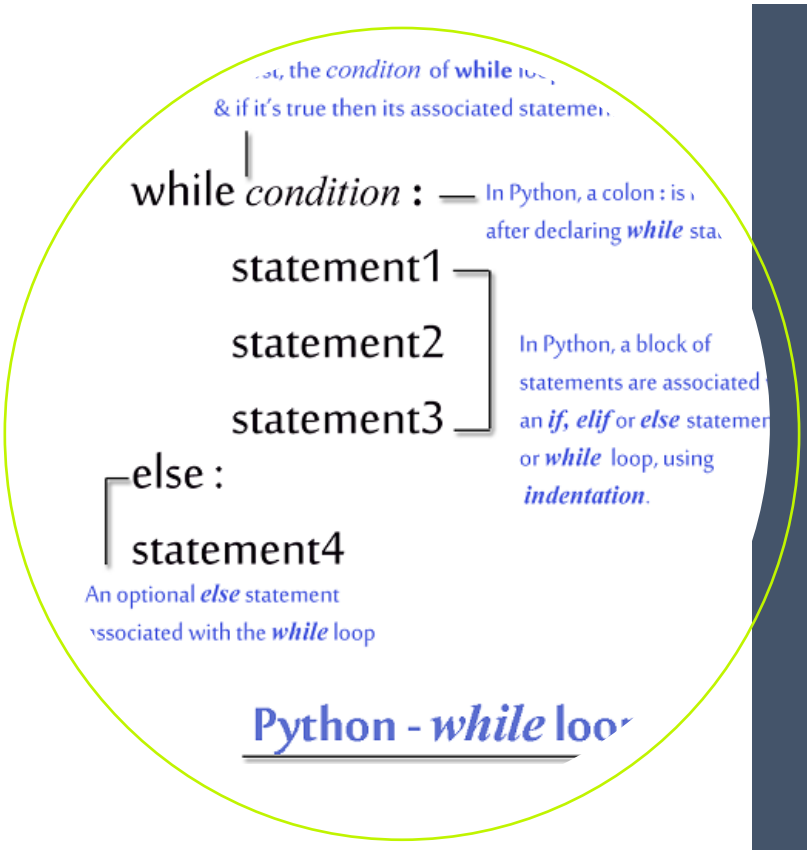
Algumas das funções nativas que você já usou requerem argumentos, aqueles valores que controlam como a função faz seu trabalho. Por exemplo, se você quer achar o seno de um número, você tem que indicar qual número é. Deste modo, sin recebe um valor numérico como um argumento. Algumas funções recebem mais de um argumento. Por exemplo, print, pode receber diversos argumentos. Dentro da função, os valores que lhe são passados são atribuídos a variáveis chamadas parâmetros.

Veja um exemplo de uma função definida pelo usuário, que recebe um parâmetro:

```
def imprimeDobrado(bruno):
    print (bruno, bruno)
```

Esta função recebe um único argumento e o atribui a um parâmetro chamado bruno. O valor do parâmetro (a essa altura, não sabemos qual será) é impresso duas vezes, seguido de uma nova linha. Estamos usando bruno para mostrar que o nome do parâmetro é decisão sua, mas claro que é melhor escolher um nome que seja mais ilustrativo.

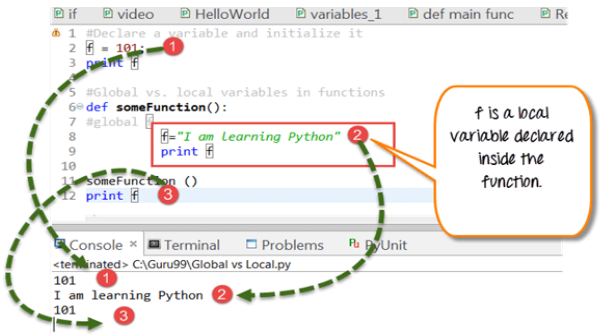
Algumas funções recebem mais de um argumento. Por exemplo, print, pode receber diversos argumentos.



VARIÁVEIS E PARÂMETROS SÃO LOCAIS

Quando você cria uma variável local dentro de uma função, ela só existe dentro da função e você não pode usá-la fora de lá. Por exemplo:

```
def concatDupla(parte1, parte2)
    concat = parte1 + parte2
    imprimeDobrado(concat)
```



A função imprimeDobrado funciona para qualquer tipo que possa ser impresso:

```
>>> imprimeDoobrado('Spam')
Spam Spam
>>> imprimeDobrado(5)
5 5
```

Na primeira chamada da função, o argumento é uma string. Na segunda, é um inteiro. Na terceira é um float. As mesmas regras de composição que se aplicam a funções nativas também se aplicam às funções definidas pelo usuário, assim, podemos usar qualquer tipo de expressão como um argumento para imprimeDobrado:

```
>>> imprimeDobrado('Spam'*4)
SpamSpamSpamSpam SpamSpamSpamSpam
```

Como acontece normalmente, a expressão é avaliada antes da execução da função, assim imprimeDobrado imprime SpamSpamSpamSpam SpamSpamSpamSpam em vez de 'Spam'*4 'Spam'*4.

Esta função recebe dois argumentos, concatena-os, e então imprime o resultado duas vezes. Podemos chamar a função com duas strings:

```
>>> canto1 = 'Pie Jesu domine, '
>>> canto2 = 'dona eis requiem. '
>>> concatDupla(canto1, canto2)
Pie Jesu domine, Dona eis requiem. Pie Jesu domine, Dona eis requiem.
```

Quando a função concatDupla termina, a variável concat é destruída. Se tentarmos imprimi-la, teremos um erro:

```
>>> print (concat)
NameError: concat
```

Parâmetros são sempre locais. Por exemplo, fora da função imprimeDobrado, não existe nada que se chama bruno. Se você tentar utilizá-la, o Python vai reclamar.

Abaixo a estrutura base de uma função simples: