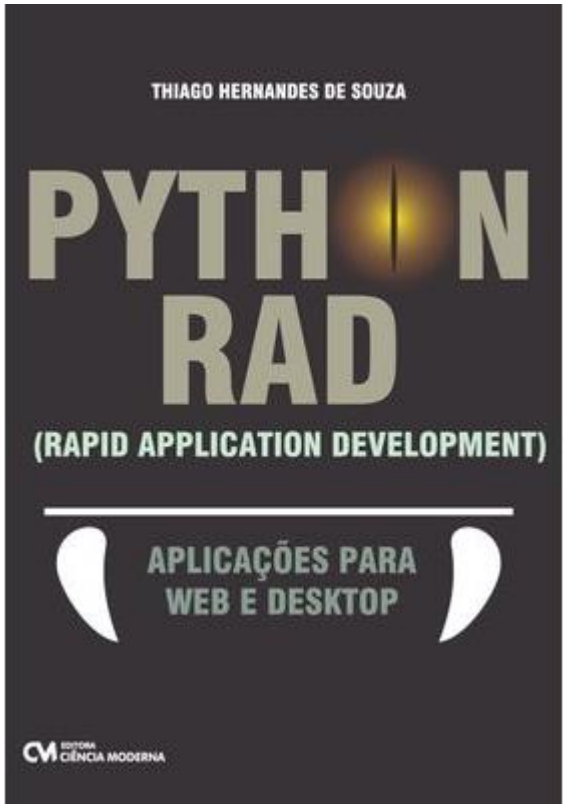
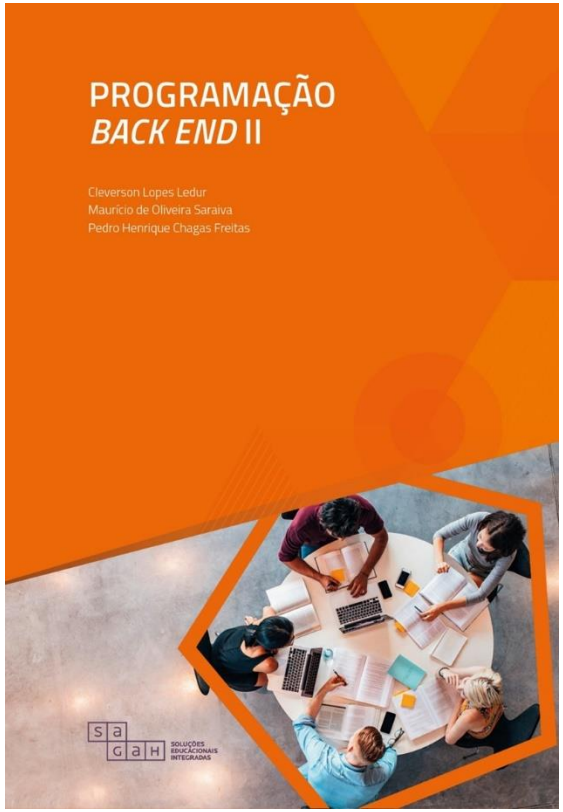
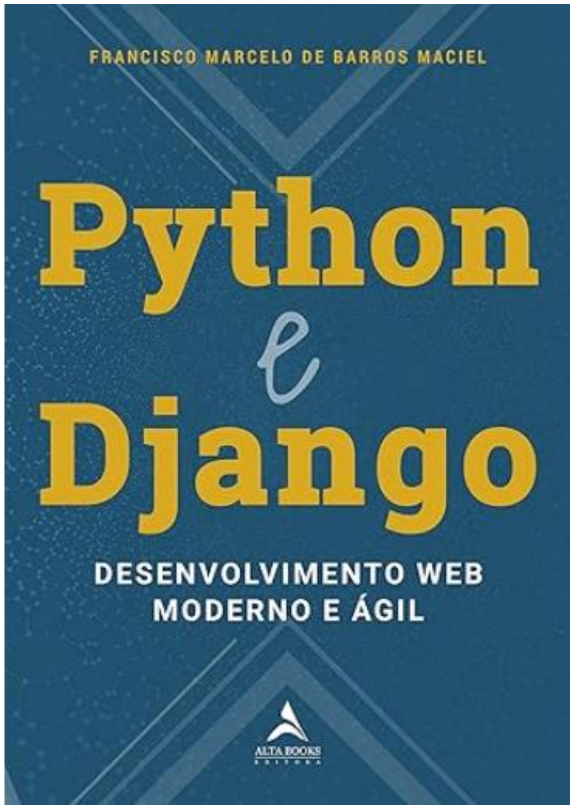
PROGRAMAÇÃO BACK-END







Python (Prática)

Comando If, Elif e Else

Condições Python e instruções If

Python suporta as condições lógicas usuais da matemática:

Igual a: a == b
Não é igual: a! = b
Menor que: a < b
Menor ou igual a: a <= b
Maior que: a > b
Maior ou igual a: a >= b

PROGRAMAÇÃO BACK-END

A "instrução if" é escrita usando a palavra-chave if .

teste01.py > ...

```
1    x = 77
2    y = 100
3    if y > x:
4        print("y é maior que x")
```

A palavra-chave elif é utilizada para verificar uma nova condição. Caso ela seja verdadeira, será executado comandos. O elif é executado se a condição do if for falsa.

teste02.py > ...

```
1    x = 77
2    y = 77
3    if y > x:
4        print("y é maior que x")
5    elif x == y:
6        print("x e y são iguais")
```

PROGRAMAÇÃO BACK-END

A palavra-chave else server para executar comandos, caso nenhuma condição anterior seja verificada como verdadeira.

teste03.py > ...

```
1   x = 100
2   y = 77
3   if y > x:
4     print("y é maior que x")
5   elif x == y:
6       print("x e y são iguais")
7   else:
8     print("x é maior que y")
```

Teste se x for maior que y E se z for maior que x

teste04.py > ...

```
1   x = 100
2   y = 77
3   z = 190
4   if x > y and z > x:
5       print("Ambas conditions são verdadeiras")
```

Teste se x for maior que y, OU se x for maior que z

teste05.py > ...

```
1   x = 100
2   y = 77
3   z = 190
4   if x > y or x > z:
5       print("Pelo menos uma das condições é verdadeira")
```

Teste se x NÃO é maior que y

```python
x = 77
y = 100
if not x > y:
    print("x não é maior que y")
```

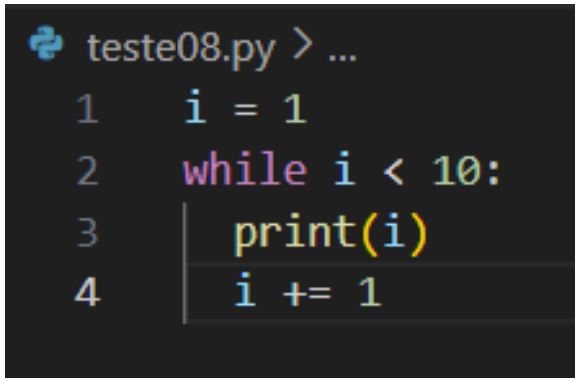
Pode-se ter instruções if dentro de instruções if, isso é chamado de instruções aninhadas

```python
x = 50
if x > 15:
    print("Acima de 10")
    if x > 30:
        print("e também acima de 30")
    else:
        print("mas não acima de 20.")
```

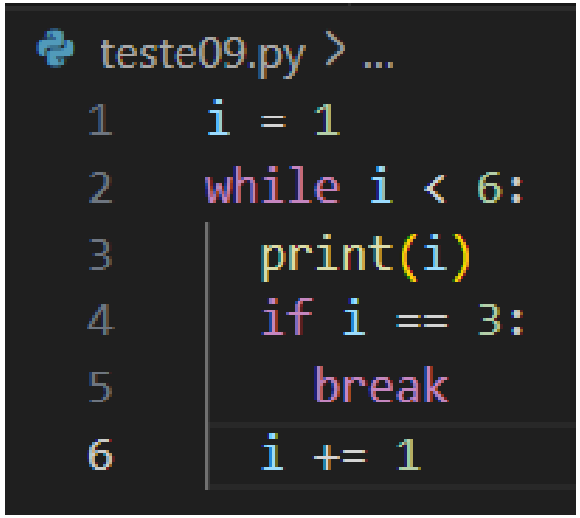
Comando While

PROGRAMAÇÃO BACK-END

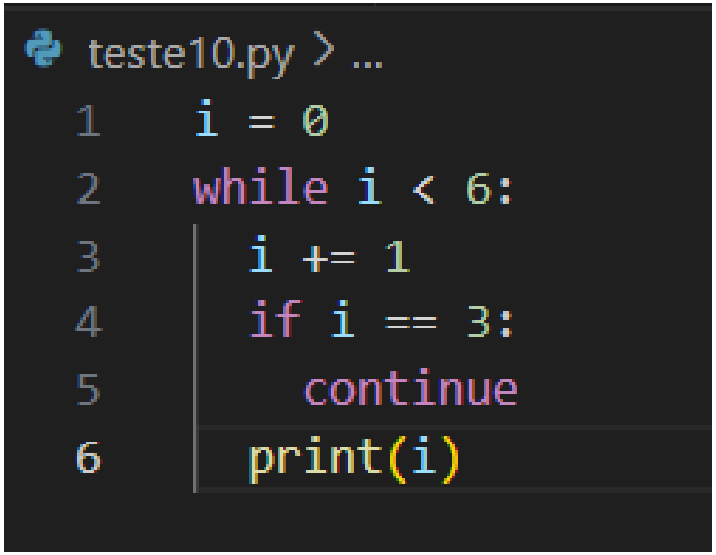
Com o loop while podemos executar um conjunto de instruções, desde que uma condição seja verdadeira.



Com a instrução break pode-se parar o loop mesmo se a condição while for verdadeira:



Com a instrução continue pode-se parar a iteração atual e continuar com a próxima



Com a instrução else pode-se executar um bloco de código uma vez quando a condição não for mais verdadeira

teste11.py > ...

```
1   i = 1
2   while i < 7:
3       print(i)
4       i += 1
5   else:
6       print("i não é inferior a 7")
```

Comando For

Imprima cada cor em uma lista de cores

teste12.py > ...

```
1    cores = ["vermelho", "azul", "verde", "amarelo"]
2    for x in cores:
3        print(x)
```

Percorra as letras da palavra “amarelo":

teste13.py > ...

```
1    for x in "amarelo":
2        print(x)
```

Saia do loop quando x for "amarelo"

teste14.py > ...

```python
cores = ["vermelho", "azul", "verde", "amarelo", "laranja", "rocho"]
for x in cores:
  print(x)
  if x == "amarelo":
    break
```

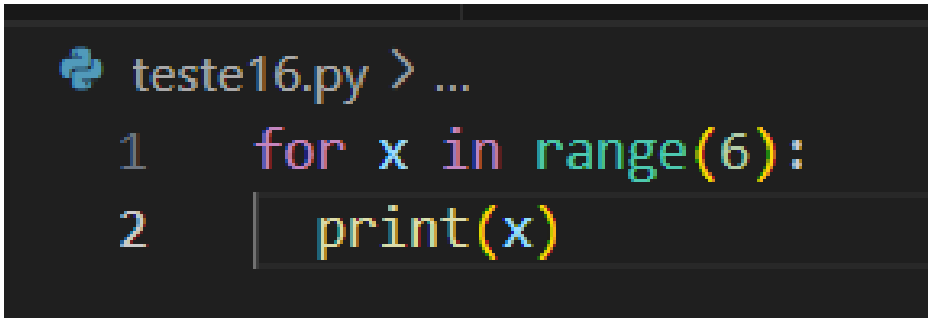
Sai do loop quando x for "amarelo", mas dessa vez o break vem antes do print

teste15.py > ...

```python
cores = ["vermelho", "azul", "verde", "amarelo", "laranja", "rocho"]
for x in cores:
  if x == "amarelo":
    break
  print(x)
```

PROGRAMAÇÃO BACK-END

A função range() retorna uma sequência de números, começando em 0 por padrão, e incrementando em 1 (por padrão), e termina em um número especificado

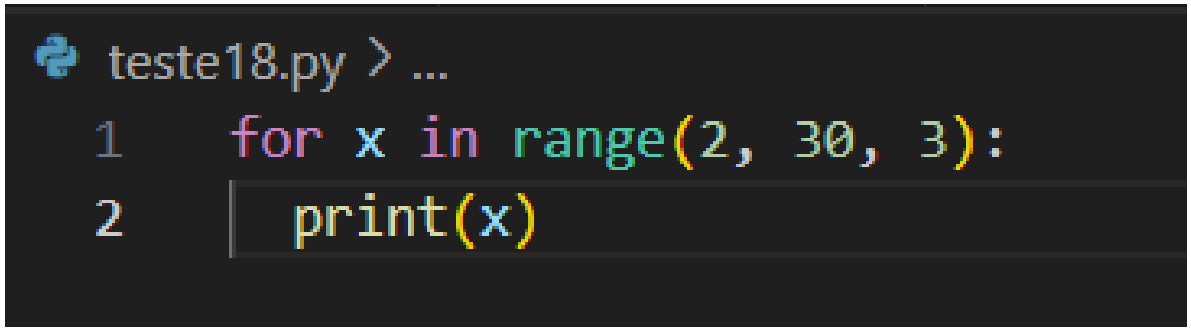


A função range() tem como padrão 0 como valor inicial, porém é possível especificar o valor inicial adicionando um parâmetro: range(2, 6), que significa valores de 2 a 6 (mas não incluindo 6):

teste17.py > ...

```
1    for x in range(2, 6):
2        print(x)
```

A função range() tem como padrão incrementar a sequência em 1, porém é possível especificar o valor do incremento adicionando um terceiro parâmetro: range(2, 30, 3)



Finalize o loop quando x for 3 e veja o que acontece com o bloco else

teste19.py > ...

```
1   for x in range(6):
2     if x == 3: break
3     print(x)
4   else:
5     print("Finalizado")
```

PROGRAMAÇÃO BACK-END

Um loop aninhado é um loop dentro de um loop.

O "loop interno" será executado uma vez para cada iteração do "loop externo":

teste20.py > ...

```
1   adj = ["madura", "grande", "saborosa"]
2   frutas = ["maça", "banana", "laranja"]
3
4   for x in adj:
5     for y in frutas:
6       print(y, x)
```