Python para economistas

Ambientes controlados

Introdução

O ambiente controlado é onde a boa parte das coisas acontecem na programação. Sem ele, um programa é simplesmente uma lista de instruções que são executadas sequencialmente. Com o fluxo de controle, você pode executar blocos de código condicionalmente ou repetidamente. Iremos os principais blocos de construção básicos neste seção: if..else, while e loop. O uso destas operações pode otimizar boa parte das tarefas e funções no Python.

Criando funções

Até agora a gente utilizou funções prontas de pacotes, ou até mesmo funções nativas do Python. Agora, iremos ver como podemos criar funções personalizadas no Python. Para isso, basta usar o a expressão "def" seguida do nome da função que estamos criando e um parenteses, em que podemos colocar parâmetros da função. Por exemplo, na função abaixo nós estamos criando uma função chamada "f" que utiliza o parâmetro "x". Após isso, nas linhas abaixo, com identação ("parágrafo") iremos escrever o que a função faz. No nosso caso temos apenas uma linha que faz a operação "3*x - 4". Tudo que estiver após o **return** será retornado pela função.

Para obtermos o resultado da função, basta utilizarmos ela como qualquer outra. Como ela só exige um parâmetro, basta colocar qualquer número dentro do parênteses.

In [9]:
```
def f(x):
    return 3*x - 4
```

In [10]:
```
f(10)
```

Out[10]: 26

A mesma função, agora com duas linhas e criando uma variável intermediária:

In [11]:
```
def f_alterada(x):
    y = 3*x
    return y - 4

f_alterada(10)
```

Out[11]: 26

If.. else

O objetivo da estrutura de "if" e "else" é bastante trivial. Resumidamente, buscamos um resultado a depender de uma condição que impusermos. No caso abaixo fizemos uma função

simples para mostrar essa funcionalidade. A função depende de um parâmetro "x" e retorna mensagens dependendo do valor de X. Se ele for menor que 23, ela diz para tentarmos um número maior. Se for maior, para tentarmos um número menor. Caso x seja igual a 23, ela retorna "esse número é 23".

In [12]:
```
def iguala23(x):
    n = 23
    if x > n:
        print("Tente um número menor")
    elif x < n:
        print("Tente um número maior")
    else:
        print("Esse número é", x)

iguala23(2)
iguala23(25)
iguala23(23)
```

```
Tente um número maior
Tente um número menor
Esse número é 23
```

While loop

No Python, "While" é usado para executar um bloco de instruções repetidamente até que uma determinada condição seja satisfeita. E quando a condição se torna falsa, a linha imediatamente após o loop no programa é executada.

A sequência de Fibonacci é uma das mais conhecidas sequências numéricas da matemática. É uma sequência de números inteiros em que cada termo corresponde à soma dos dois anteriores. Iremos criar uma função que retorna "n" primeiros números dessa sequência. Na nossa função, enquanto i for menor do que n, repetiremos o loop.

In [50]:
```
def fibonacci(n):
    i = 0
    primeiro_valor = 0
    segundo_valor = 1

    while(i < n):
        if(i <= 1):
                prox = i
        else:
                prox = primeiro_valor + segundo_valor
                primeiro_valor = segundo_valor
                segundo_valor = prox
        print(prox)
        i = i + 1


fibonacci(10)
```

```
0
1
1
2
3
5
8
13
```

```
21
34
```

For Loop

O for loop é de certa forma parecido com o "while". Entretanto, ao invés de impormos uma condição para o algoritmo parar de realizar iterações, nós mostramos a ele sobre quais valores precisamos realizar determinadas funções.

Para mostrar isso, criamos uma função que retorna o o número de caracteres para cada elemento da lista. Basicamente o que o for loop faz é executar a função para todos os elementos da lista. Veja que agora o parâmetro da função é uma lista, não um número.

In [13]:
```
def caracteres(x):

    for a in x:
        print(a, len(a))


caracteres(['gato', 'cachorro', 'passaro', "jaguatirica"])
```

```
gato 4
cachorro 8
passaro 7
jaguatirica 11
```

Range

A função "range" cria uma sequência de números inteiros de 0 até n-1, sendo n o valor parametrizável. Tecnicamente, ela retorna uma sequência de n números, começando pelo 0. Pode ser muito útil para utilizar junto ao "for loop".

In [60]:
```
range(33)
```

Out[60]:
```
range(0, 33)
```

In [67]:
```
for i in range(12):
    p = 10
    print(i, "elevado a", p, "é igual a", i**p)
```

```
0 elevado por 10 é igual a 0
1 elevado por 10 é igual a 1
2 elevado por 10 é igual a 1024
3 elevado por 10 é igual a 59049
4 elevado por 10 é igual a 1048576
5 elevado por 10 é igual a 9765625
6 elevado por 10 é igual a 60466176
7 elevado por 10 é igual a 282475249
8 elevado por 10 é igual a 1073741824
9 elevado por 10 é igual a 3486784401
10 elevado por 10 é igual a 10000000000
11 elevado por 10 é igual a 25937424601
```

Break e continue

As expressões "break" e "continue" são utilizadas dentro do loop (tanto, while quanto for) para realizar controles internos na operação.

break - Essa expressão encerra o loop no qual é usada. Se a instrução break for usada dentro de loops aninhados, o loop atual será encerrado e o fluxo continuará com o código seguindo o que vem após o loop.

continue -Essa expressão ignora o código que vem depois a ela e o controle é passado de volta ao início para a próxima iteração.

As instruções de loop podem ter também a expressão "else"; ele é executado quando o loop termina por esgotamento do iterável (com for) ou quando a condição se torna falsa (com while), mas não quando o loop é encerrado por uma instrução break.

Por exemplo, uma função para encontrar números primos. A operacionalização desta função ocorre da seguinte maneira:

1. Definimos um X, que será o valor máximo que nós iremos obter os números primos
2. Iremos iterar todos os números entre 2 e X duas vezes, criando "n" e "y".
3. Assim, iremos verificar se "n" é divisível para cada um dos "y".
4. No primeiro caso em que "n" for divisível por "y", o loop irá parar ("break"). Ou seja, iremos para o próximo número.
5. Se não parar, a função irá retornar "n", que é um número primo.

In [19]:
```
def primos(x):
    for n in range(2, x):
        for y in range(2, n):
             if n % y == 0:
                break
        else:

            print(n)

primos(30)
```

```
2
3
5
7
11
13
17
19
23
29
```

Já a função "continue", ao invés de parar a execução do loop, apenas manda para a próxima iteração.

In [22]:
```
for item in range(6):
    if item == 4:
        continue
    print(item)
```

```
0
1
2
3
5
```