Callback Funktion und **kwargs

1. Implementieren Sie den unten angegebenen Code. Tätigen Sie die Aufrufe `algorithm(1)` und `algorithm(1, callback=callbacl` Was passiert? Fügen Sie Docstrings hinzu, die kurz das Verhalten der Funktionen `algorithm` und `callback` erklären.

2. Nun soll die Funktion `callback()` nur dann etwas ausgeben, wenn `xstart` größer als ein gewisser `threshold` ist. Fügen Sie diese Eigenschaft und das Schlüsselwortargument `threshold` der Funktion `callback()` hinzu. Verändern Sie dabei die äußere Funktion `algorithm()` nicht! Welche Parameter übergeben Sie an `algorithm()`, wenn Sie nur Iterierte größer als `threshold=1.6` gedruckt bekommen möchten?

   Hinweis: Verwenden Sie einen optionalen Parameter: `callback(xstart, threshold=0.0)`.

3. Erläutern Sie, welchen praktischen Vorteil **kwargs in verschachtelten Funktionsauswertungen haben kann.

```python
from random import random

def algorithm(xstart, callback=None, **kwargs):
   for _ in range(10):
      if callback:
         callback(xstart, **kwargs)
      xstart += random() - 0.5  # der Aufruf random() generiert eine Zufallszahl in [0,1]
   return xstart

def callback(xstart):
   print(xstart)
```

Solution:

```python
#!/usr/bin/env python
# coding: utf-8
# <h1>Table of Contents<span class="tocSkip"></span></h1>
# <div class="toc"><ul class="toc-item"><li><span><a href="#Nested-Functions-with-**kwargs"
    data-toc-modified-id="Nested-Functions-with-**kwargs-1"><span class="toc-item-num">1 &
    nbsp;</span>Nested Functions with **kwargs</a></span></li></ul></div>
# ### Nested Functions with **kwargs
from random import random
def algorithm(xstart, callback = None, **kwargs):
    """
    dummy algorithm.

    :param xstart: float
    :param callback: callback function with signature callback(xstart, **kwargs)
    :param **kwargs: additional parameters for callback
    """
    for _ in range(10):
        if callback:
            callback(xstart, **kwargs)
        xstart += (random()-.5)
    return xstart
def callback(x):
    """
    callback function. Prints the iterate x whenever it is called in the outer algorithm.

    :param x: Current iterate
    """
    print(x)
xfinal = algorithm(1, callback=callback)
```

```
print("xfinal: ", xfinal)
# **Vorteil der Nutzung von \*\*kwargs**
#
# Wir definieren nun `callback()` neu, ohne `algorithm()` zu ver ndern. Das bedeutet,
# der Nutzer von `algorithm()` kann seine callback-Funktion nach belieben definieren und einsetzten und muss dazu (den eventuell schrecklich komplizierten) Algorithmus nicht anfassen!
def callback(x, threshold=0.0):
    """
    callback function. Prints the iterate x if its called and x > threshold.

    :param x: Current iterate
    :param threshold: threshold=0.0 (optional)
    """
    if x > threshold:
        print(x)
xfinal = algorithm(1, callback=callback, threshold=1.6)
print("xfinal: ", xfinal)
```