**Recipe Name:** cyclic\_min

**Inputs:**

*func*, the function whose local minimum will be returned

*start*, a sequence where each element of the sequence represents the value of the corresponding variable, with the whole sequence representing the starting point for cyclic minimization

*num\_vars*, an integer representing the number of variables in *func*

*step\_size*, a float representing how far to go along each variable for each iteration of cyclic minimization

**Outputs:**

*local\_mins*, a sequence where each element of the sequence represents the value of its corresponding variable, with each variable represented in the same order as *start*, with the whole sequence representing the found local minimum

**Steps:**

1. Initialize *local\_mins* to a copy of *start*
2. Initialize *temp\_plus* to an empty sequence
3. Initialize *temp\_min* to an empty sequence
4. *final\_local* 🡨 0
5. While *final\_local* is not equal to *num\_vars*,
   1. For each number, *var*, from 0 to *num\_vars* – 1,
      1. Initialize *found* to False
      2. While *found* is not True, do the following
         1. *temp\_plus* 🡨 *local\_mins*
         2. *temp\_min* 🡨 *local\_mins*
         3. *temp\_plusvar* 🡨 *local\_minsvar* + *step\_size*
         4. *temp\_plusvar* 🡨 *local\_minsvar*– *step\_size*
         5. If *func*(*temp\_min*)< *func*(*temp\_plus*) and *func*(*temp\_min*)< *func*(*local\_mins*), then
            1. *local\_mins* 🡨 *temp\_min*
            2. *final\_local* 🡨 0
         6. Otherwise, if *func*(*temp\_plus*)< *func*(*temp\_min*) and *func*(*temp\_plus*)< *func*(*local\_mins*), then
            1. *local\_mins* 🡨 *temp\_plus*
            2. *final\_local* 🡨 0
         7. Otherwise,
            1. *found* 🡨 True
            2. *final\_local* 🡨 *final\_local* + 1
6. Return *local\_mins*