```
# This is the original HOTPO function from HW1, converted
# into a function.

# Notice the function definition comes first. When Python
# reads the definition, it doesn’t execute anything.

# The input to the function is a single value, which we map
# onto the variable n; this variable is local to the function.
def hotpoLength(n):
  count = 0
  while n>1:
    if n%2:
      n = 3*n + 1
    else:
      n = n/2
    count = count + 1
  # Note we don’t print anything out, but instead return
  # a value. This is the value that the invocation of the
  # function will yield when evaluated.
  return count

# Here’s the invocation of the function. Since we evaluate from
# "inside out," the integer equivalent of the user input is "passed"
# to the hoptoLength(n) function as the value of the local function
# variable n. The hoptoLength() function returns a value, which is
# then printed.
print hotpoLength(int(raw_input("Enter a number: ")))
```

```
# This is the same function defined previously, with a slight twist to
# reduce the number of iterations. Can you spot it?
def hotpoLength(n):
  count = 0
  while n>1:
    if n%2:
      n = (3*n + 1)/2
      count = count + 2
    else:
      n = n/2
      count = count + 1
  return count

# Here is a new function definition. hotpoLengthMax() returns the
# longest hotpoLength() of any number between 1 and n-1. It does this
# by repeated invocation of the hotpoLength() function above. Note
# that the variable n in the hotpoLengthMax() function signature is
# not the same variable n as the one in the hotpoLength() signature.
def hotpoLengthMax(n):
  i = 1
  # We’ll use the maxlen variable to keep track of the longest
  # hotpoLength() encountered so far. Note that maxlen only exisits
  # within the hotpoLengthMax() function; it is not defined outside of
  # the function.
  maxlen = 0
  while i < n:
    # Here’s the invocation of hotpoLength(). On invocation, i is
    # evaluated and its value is bound to the n variable in the
    # hotpoLength() function definition signature. When hotpoLength()
    # completes, the value it returns is compared against maxlen.
    if hotpoLength(i)>maxlen:
       maxlen = i
    i = i + 1
  # The value returned by the function is the longest hotpoLength()
  # encountered.
  return maxlen

# Here’s the invocation of hotpoLengthMax(). The integer equivalent of
# the user’s input is mapped to the variable n in the hotpoLengthMax()
# function signature. The value returned by hotpoLengthMax() is
# printed.
print hotpoLengthMax(int(raw_input("Enter a number: ")))
```

```
# This is the same function defined previously.
def hotpoLength(n):
  count = 0
  while n>1:
    if n%2:
      n = (3*n + 1)/2
      count = count + 2
    else:
      n = n/2
      count = count + 1
  return count

# A slight twist on the hotpoLengthMax() function of the previous
# example. Here, instead of returning the longest hotpoLength()
# encountered between 1 and n-1, we return the index of the longest
# hotpoLength() encountered between lo and hi-1, provided it exceeds
# maxsofar. We are basically breaking the hotpoLengthMax() range into
# chunks.
def hotpoLengthMaxInRange(lo, hi, maxsofar):
    i = lo
    while i < hi:
        if hotpoLength(i)>maxsofar:
            # A return here is like a super break; it exits not only
            # the while loop but the entire function!
            return i
        i = i + 1
    # No values > maxsofar.
    return hi

# Note how the two raw_input() statements are evaluated in order left
# to right when you execute.
print hotpoLengthMaxInRange(int(raw_input("Enter lo: ")), int(raw_input("Enter hi: ")), 0)
```

```
# Further development of the previous version. Our goal is to produce
# the numbers listed on http://oeis.org/A006877 -- corresponding to
# the set of integers with the longer hoptoLength than all of their
# smaller integers.

# Unchanged from previous examples.
def hotpoLength(n):
    count = 0
    while n>1:
        if n%2:
            n = (3*n + 1)/2
            count = count + 2
        else:
            n = n/2
            count = count + 1
    return count

# This is pretty much the same function defined previously, except
# that now it is returning two values: the new max hoptoLenght() as
# well as the integer index that produces it. Notice how every return
# statement returns two values, and notice how, when the function is
# invoked below, there are two variable set to "receive" the returned
# values.
def hotpoLengthMaxInRange(lo, hi, maxsofar):
    i = lo
    while i < hi:
        h=hotpoLength(i)
        if h > maxsofar:
            return (i, h)
        i = i + 1
    return (hi, maxsofar)

n = int(raw_input("Enter an upper limit: "))
i = 1
j = n
# Initial max hotpoLength() artificially set to -1 so that we "notice"
# hotpoLength(1) is 0, a new max.
max = -1
while i < n:
    # Repeatedly invoke the new range.
    (j , max) = hotpoLengthMaxInRange(i, j, max)
    # If j==n, we’ve exhausted the originally specified range from 1
    # to n-1, and no new winner was found in the invocation of
    # hotpoLengthMaxInRange().
    if j == n:
        break
    # OK, must have found a new "winner;" print it out.
    print j
    # Update the lower end of the range to start the next invocation
    # of hotpoLengthMaxInRange() just beyond the last winner.
    i = j + 1
    # Reset the upper end of the range to the original limit.
    j = n
```