Making Change

In this problem, you will solve a different problem using the powerful use-it-or-lose-it strategy.

Imagine that you just got a job working for Cash Register Advanced Products. (The public relations division has suggested that the company avoid using an acronym for the company name.) The company builds electronic cash registers and the software that controls them. The cash registers are used all around the world in countries with different coin systems.

Next, imagine that we are given a list of the coin types in a given country. For example, in the U.S. the coin types are:

```
[1, 5, 10, 25, 50]
```

But, in the Kingdom of Shmorbodia, the coin types are:

```
[1, 7, 24, 42]
```

In general, the coin system could be anything, except that there is **always a 1 unit coin (penny)**. In these examples, the denominations were given smallest to largest, but that's not necessary. There's nothing about use-it-or-lose-it that depends on the order of the coins.

Here's the problem. Given an amount of money and a list of coin types, we would like to find the **least number of coins** that makes up that amount of money. For example, in the U.S. system, if we want to make 48 cents, we give out 1 quarter, 2 dimes, and 3 pennies. That solution uses 6 coins and that's the best we can do in this case. Making 48 cents in the Shmorbodian system, however, is different. Giving out a 42 cent coin - albeit tempting - will force us to give the remaining balance with 6 pennies, using a total of 7 coins. We could do better by simply giving two 24 cent coins.

Your first task is to write a function called `change(amount, coins)` where `amount` is a non-negative integer indicating the amount of money to be made and `coins` is a list of coin values with 1 always being in the list when we first call the function. (This ensures that it is always possible to make change for any positive amount.) The function should return a non-negative integer indicating the minimum number of coins required to make up the given `amount`.

Here is an example of this function in action:

```
>>> change(48, [1, 5, 10, 25, 50])

6

>>> change(48, [1, 7, 24, 42])

2
```

```
>>> change(35, [1, 3, 16, 30, 50])

3
```

A few notes and tips...

Not surprisingly, the secret-to-all-happiness is to use the *use-it-or-lose-it* recursion strategy.

Second, you may want to use the built-in function `min(x, y)` which returns the smaller of its two inputs.

Third, in the event that `change` is confronted with a problem for which there is no solution, returning the number infinity is appropriate to indicate that there is no number of coins that would work. This happens, for example, when we ask to make change for some positive amount of money but there are no coins in the list. What is the number infinity in Python? Using `float("inf")` will do the trick. This behaves like infinity should. Here are some examples:

```
>>> float("inf") > 42

True

>>> 42 + float("inf")

inf                      <-- It's still infinity!

>>> min(42, float("inf"))

42
```