# fibonacci.py

# Our notion of the Fibonacci sequence here is 0, 1, 2, 3, 5, 8, 13, ...
# fib 0 = 0
# fib 1 = 1
# fib n = fib(n-2) + fib(n-1)

import time

def timeFib(n):
    start = time.time()
    fib(n)
    end = time.time()
    return end - start

def fib(n):
    return n if n < 2 else fib(n-2) + fib(n-1)

print [ timeFib(n) for n in range(25,30) ]
print
print "timeFib(36) =", timeFib(36)
print

# Notice that this version of fib() is highly recursive.
# fib(0) and fib(1) don't make recursive calls to fib(). But,
# fib(2) makes two recursive calls,
# fib(3) makes three recursive calls.
# fib(4) makes four recursive calls.
# fib(5) makes 14 recursive calls.
# fib(6) makes 24 recursive calls.
# fib(7) makes 40 recursive calls.

# The cost of fib(n) explodes as n gets greater; the cost is said to
grow exponentially with n.

# If we want, we can use a Python dictionary for caching values for
values of fib(n), so that we can get those values from the dictionary
rather than repeatedly recomputing them.

# First, dictionaries:

d = {} # d is an empty dictionary.

d["cat"] = 4

# >>> d["cat"]
# 4

# >>> d["monkey"]
# KeyError: 'monkey'

# >>> "monkey" in d
# False
# >>> "cat" in d
# True
# >>> d["monkey"] = 2
# >>> "monkey" in d
```python
# True
# >>> d['monkey']
# 2
# >>>

fibs = {}  # a cache of Fibonacci numbers already computed

def fib(n):
    if n in fibs:
        return fibs[n]
    fibs[n] = n if n < 2 else fib(n-2) + fib(n-1)
    return fibs[n]

print [ timeFib(n) for n in range(25,30) ]
print
print "timeFib(36) =", timeFib(36)
print

# >>>
# [0.06609106063842773, 0.10143089294433594, 0.17368602752685547,
# 0.25845789909362793, 0.421720027923584]
#
# timeFib(36) = 12.0775170326
#
# [3.0994415283203125e-05, 9.5367431640625e-07, 2.1457672119140625e-06,
# 9.5367431640625e-07, 9.5367431640625e-07]
#
# timeFib(36) = 1.38282775879e-05
# >>>
```