Memoization

In this lecture we will discuss memoization and dynamic programming. For your homework assignment, read the Wikipedia article on Memoization, before continuing on with this lecture!

Memoization effectively refers to remembering ("memoization" -> "memorandum" -> to be remembered) results of method calls based on the method inputs and then returning the remembered result rather than computing the result again. You can think of it as a cache for method results. We'll use this in some of the interview problems as improved versions of a purely recursive solution.

A simple example for computing factorials using memoization in Python would be something like this:

In [1]:
```python
# Create cache for known results
factorial_memo = {}

def factorial(k):

    if k < 2:
        return 1

    if not k in factorial_memo:
        factorial_memo[k] = k * factorial(k-1)

    return factorial_memo[k]
```

In [2]:
```python
factorial(4)
```

Out[2]:
```
24
```

Note how we are now using a dictionary to store previous results of the factorial function! We are now able to increase the efficiency of this function by remembering old results!

Keep this in mind when working on the Coin Change Problem and the Fibonacci Sequence Problem.

We can also encapsulate the memoization process into a class:

In [3]:
```python
class Memoize:
    def __init__(self, f):
        self.f = f
        self.memo = {}
    def __call__(self, *args):
        if not args in self.memo:
            self.memo[args] = self.f(*args)
        return self.memo[args]
```

Then all we would have to do is:

In [4]:
```python
def factorial(k):

    if k < 2:
        return 1
    return k * factorial(k - 1)

factorial = Memoize(factorial)
```

Try comparing the run times of the memoization versions of functions versus the normal recursive solutions!