

Solutions, Week 2, Day 2

1. The fibonacci number for the values 0 and 1 is set as 1. Subsequent values are calculated as $fn(x) = fn(x-1) + fn(x-2)$. For example $fib(10) = fib(9) + fib(8)$. Write a recursive function to calculate the fibonacci number for any positive integer.

```
def fibo (n):
    if (n in [0, 1]):
        # base case
        return 1
    else:
        # recursion to simpler / earlier form
        return fibo (n-1) + fibo (n-2)
```

2. The previous function is inefficient as each call of the function leads to calling the function again a lot of times. How many?

(I said it was $1 + 2^{(n-1)}$. This is wrong.) Writing down the series of calls, you can see that for the term 5, `fib` is called 15 times, 9 times for 4, 5 times for 3, 3 times for 2. Actually the number of calls is a fibonacci sequence of it's own, which grows explosively. This means that something like `fib(100)` may effectively be incalculable.

```
f(5) - f4 - f3 - f2 - f1
      - f0
      - f1
      - f2 - f1
      - f0
      - f3 - f2 - f1
      - f0
      - f1
```

3. Work out a way of caching results (using a dictionary) so that the fibonacci function is a lot more efficient.

We store results in a dictionary, so that they are calculated once and once only, even on subsequent calls to the function, and recursion is terminated early:

```
fibCache = {}

def fibo (n):
    if (fibCache.has_key(n)):
        return fibCache[n]
    else:
        if (n in [0, 1]):
            theRes = 1
        else:
            theRes = fibo (n-1) + fibo (n-2)
        fibCache[n] = theRes
    return theRes
```

4. Write a function that prints out whatever arguments you pass to it. That is, `myFunc (1, 'a', True)` will print out `1, a, True`.

```
def printargs (*args):
    for a in args:
        print a,
    print ""
```

5. Modify this so that it prints out the named arguments as well. That is, `myFunc (1, file='a', limit=True)` will print out `1, file=a, limit=True`.

```
def printargs (*args, **kwargs):
    for a in args:
        print a,
    print ""
    for k,v in kwargs:
        print "%s=%s" % (k, v),
    print ""
```

6. Write a function that accepts a string, a function and following arguments and returns the result of the function applied to the string with those arguments. For example, `callStrFunc (str, find, 1, 10)` will return the result of `find (str, 1, 10)`.

Passing `*args` will break the arg list down to its components so if `args` is `[1, 2, 3]` a call to `fn (*args)` will actually be `fn (1, 2, 3)`:

```
def callfunc (str, fn, *args):
    return fn (str, *args)
```