

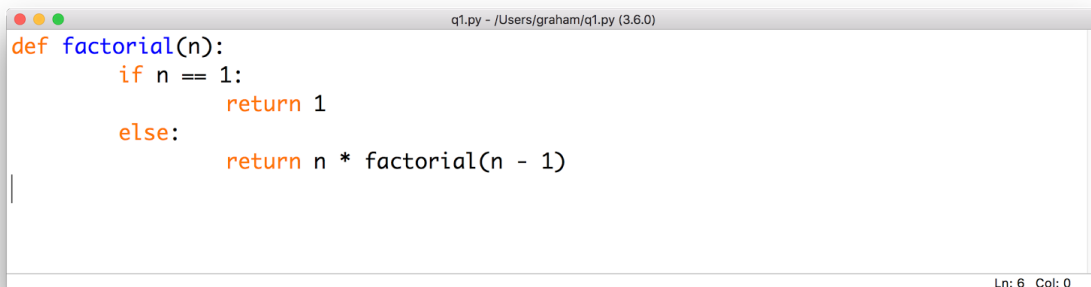
Solutions 3

Summer 2017

Task 1 - Recursive Factorials

We define a $\text{factorial}(n)$ as n multiplied by $\text{factorial}(n-1)$, with the base case $\text{factorial}(1) = 1$.

We will not be concerned with the fact that $\text{factorial}(0)$ is 1, so that could be used as a better base case, since while the code does not grow, subtlety is unnecessary.



```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n - 1)
```

Ln: 6 Col: 0

```
*Python 3.6.0 Shell*
===== RESTART: /Users/graham/q1.py =====
>>> factorial(4)
24
>>> factorial(1)
1
>>>
```

Task 2 - String Manipulation

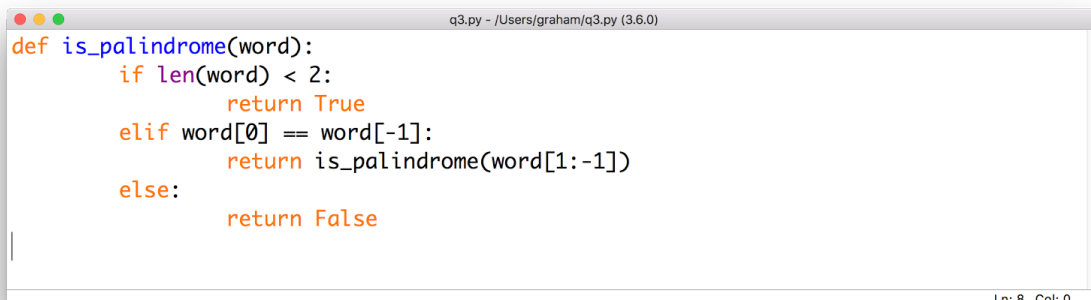
This task is very similar to the task given in the first session regarding lists, and is specifically designed as a lead into task 3.

```
Python 3.6.0 Shell
Python 3.6.0 (default, Dec 24 2016, 18:27:04)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> word = "hello"
>>> word[0]
'h'
>>> word[-1]
'o'
>>> word[1:-1]
'ell'
>>> |
```

Task 3 - Recursive Palindromes

A palindrome is a word which reads the same backward or forward, such as madam or kayak.

We define our recursive function with the definition that if the first and last letters are the same, and the middle is a palindrome, then the word is a palindrome. Our base case is if the of length less than 2. We define those to be palindromes.

A screenshot of a code editor window titled 'q3.py - /Users/graham/q3.py (3.6.0)'. The window contains a Python function definition for 'is_palindrome'. The code is as follows:

```
def is_palindrome(word):  
    if len(word) < 2:  
        return True  
    elif word[0] == word[-1]:  
        return is_palindrome(word[1:-1])  
    else:  
        return False
```

The status bar at the bottom right of the editor shows 'Ln: 8 Col: 0'.

```
Python 3.6.0 Shell
===== RESTART: /Users/graham/q3.py =====
>>> is_palindrome('test')
False
>>> is_palindrome('hannah')
True
>>>
```

Task 4 - Palindromes Again

A simpler non-recursive definition reverses the string, and then checks if this is equal to the original.

The simpler version might be undesirable for a very large string though, since Python has to reverse the entire string, rather than potentially just checking the start and end characters, and seeing they're different.

```
q3.py - /Users/graham/q3.py (3.6.0)
def is_palindrome(word):
    return word[::-1] == word
```

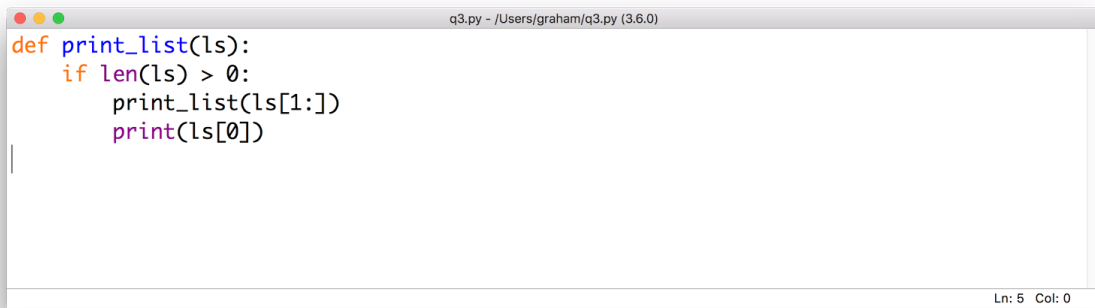
```
Python 3.6.0 Shell
===== RESTART: /Users/graham/q3.py =====
>>> is_palindrome('test')
False
>>> is_palindrome('hannah')
True
>>>
```

Task 5 - Printing a List

Our original function works by printing the first element, and then printing the rest, recursively. The base case is implicit here, since it really is just “do nothing”. See how this function does not “return”, it only has the “side effect” of writing to the screen.

In order to modify this, we simply move the call to print under the recursive call, so instead of printing the element, and then printing the rest of the list, we now print the rest of the list, and then printing the rest of the list.

This demonstrates a way that recursion can make our life a lot easier, by using Python call stack implicitly to keep track of our progress.

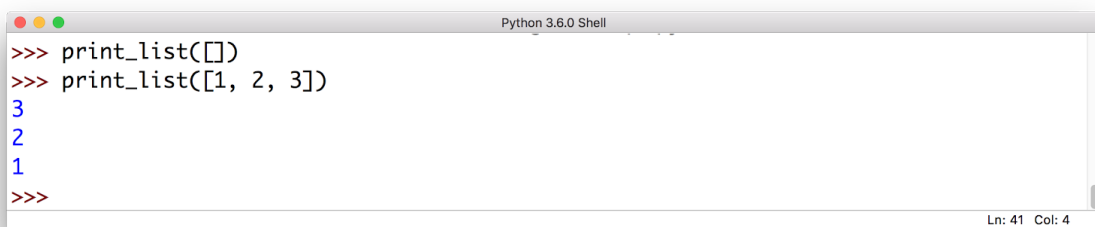
A screenshot of a Python script editor window titled 'q3.py - /Users/graham/q3.py (3.6.0)'. The window contains a recursive function definition for printing a list. The code is as follows:

```
def print_list(ls):  
    if len(ls) > 0:  
        print_list(ls[1:])  
        print(ls[0])
```

The cursor is positioned at the end of the second line. The status bar at the bottom right indicates 'Ln: 5 Col: 0'.

```
def print_list(ls):  
    if len(ls) > 0:  
        print_list(ls[1:])  
        print(ls[0])
```

Ln: 5 Col: 0

A screenshot of a Python 3.6.0 Shell window titled 'Python 3.6.0 Shell'. The window shows the execution of the recursive function defined in the previous block. The input is an empty list, followed by a list containing the elements 1, 2, and 3. The output shows the elements 3, 2, and 1 printed on separate lines, which is the reverse of the input list. The status bar at the bottom right indicates 'Ln: 41 Col: 4'.

```
>>> print_list([])  
>>> print_list([1, 2, 3])  
3  
2  
1  
>>>
```

Ln: 41 Col: 4