Q19

```
fact = lambda n: 1 if n == 0 else n * fact(n-1)
```

Q10

```
def concat_strings(lst):
   return reduce(lambda x, y: x + ' ' + y, lst)
print(concat_strings(['Hello', 'world', 'this', 'is', 'a', 'test']))
```

Q11

```
def cumulative_sum_of_squares(lst):
   return (lambda l: [(lambda sublist: (lambda evens: (lambda squares: (lambda cumsum: cumsum)([sum(squares[:i+1]) for i in range(len(squares))]))([num**2 for num in evens]))([num for num in sublist if num % 2 == 0]))(sublist) for sublist in l])(lst)
print(cumulative_sum_of_squares( [[1, 2, 3], [4, 5, 6], [7, 8, 9]]))
```

Q12

```
def cumulative_sum_of_squares(lst):
   return reduce(lambda x, y: x + y, map(lambda x: x**2, filter(lambda x: x % 2 == 0, lst)))
```

Q13

```
def count_palindromes(lst):
   return list(map(lambda sublist: len(list(filter(lambda x: x == x[::-1], sublist))), lst))
```

Q14

In the lazy evaluation section of the program, the values are generated lazily using a list comprehension.

The list comprehension iterates over the generator object returned by generate_values() and squares each value as it is generated.

This means that the values are generated and squared one by one, only when they are needed, resulting in a more efficient use of resources.