Lambdas and Continuations

1. Consider the code sample below. Modify `make_greater_than` to use a lambda instead of a functor. Explain why the two are equivalent.

```
class GreaterThan {
public:
    GreaterThan(int threshold_in) : threshold(threshold_in) {}

    bool operator()(int value) const { return value > threshold; }

private:
    const int threshold;
};

auto make_greater_than(int threshold) {
    return GreaterThan(threshold);
}

int main() {
    auto gt3 = make_greater_than(3);
    cout << gt3(2) << endl;
    cout << gt3(20) << endl;
    cout << make_greater_than(30)(20) << endl;
}
```

2. Demonstrate two different ways to call the Scheme **`map`** procedure to square all of the numbers in the list `'(1 2 3 4)`. Use a user-defined procedure in the first solution and use a lambda expression in the second.

```
(define (map func lst)
  (if (null? lst) lst
      (cons (func (car lst))
            (map func (cdr lst))
      )
   )
)
```

3. Write a Python generator that generates the sequence of squared integers. Use the `naturals()` generator from lecture in your solution.

```
def squares():
```

If you were to add a print statement before and after the **`yield`** statement in your generator code above, what will print in the following code sample? Assume your two print statements are `print("before yield")` and `print("after yield")`

```
s = squares()
for i in range(0,3):
    print("Square of {} = {}\n".format(i, next(s))
```

Bonus: write the squares generator instead as a generator expression

4. What is the output of the following code and why?

```
def squares2(start, end):
    for i in range(start, end):
        yield i * i

r = squares2(0,5)
for i in r:
    print(i)
for i in r:
    print(i)
```