1. Here are the stack data contents.

   (i) a = 2, b = 20
   (ii) a = 2, b = 20, r = 3, q = 3
   (iii) a = 2, b = 20, r = 3, q = 3, s = 3, t = 10, p = 6
   (iv) a = 2, b = 20, r = 3, q = 7
   (v) a = 2, b = 8

2. (a) Under call-by-value, twiddle has no effect on p0 or p1, so the first two outputs are “0 1”; swizzle actually exchanges the contents of the a fields, so the second two outputs are “1 0”.

   (b) Under call-by-reference, twiddle actually exchanges the values of p0 and p1, so the first two outputs are “1 0”; swizzle exchanges things as before, so the second two outputs are “0 1”.

   (c) Assuming un-boxed semantics, the parameters to twiddle and swizzle are copied when they are passed, creating new objects, and similarly for the assignment to z in twiddle. This has no effect on the visible behavior of twiddle, so the first two outputs are again “0 1”. But swizzle now operates on local copies of its arguments, so it has no effect on the variables in main, and the second two outputs are also “0 1”.

3. Function s and t are not tail-recursive, as they each perform an addition after the return of the recursive call. (The fact that the recursive call comes last on the line in s makes no difference; it’s the order of operations that counts.)

Function even is tail-recursive, and can be rewritten thus:

```scala
def even(x:Int) : Boolean = {
  var y = x
  while (y > 1) { y = y - 2 }
  return (y == 0)
}
```

(We have to assign x to a new var before we can change it, because Scala parameters are always immutable vals. Notice the convenience of returning the value of a boolean expression directly. The return keyword is not strictly necessary.)

fac is not tail-recursive, since it performs a multiplication after the return of the recursive call.

facn is tail-recursive, and can be rewritten thus:

```scala
def facn(x:Int,y:Int) = {
  var x1 = x
  while (y > 1) { x1 = x1 * y; y = y - 1 }
  return x1
}
```
var y1 = y
while (x1 >= 2) {
    y1 = x1 * y1  // need to do this first to avoid overwriting x1 too soon
    x1 = x1 - 1
}
y1