Nick Largey
Exploration #2
Scientific Computing

2.1.8:
Smaller would be preferred for the forward error. Although a case for similar could be made, because of the 3rd criteria of a "small perturbation in the problem data results in a commensurately small perturbation in the solution". Because of the ambiguity of the word "similar" it could be interpreted to fit the criteria, although smaller output from the forward error, would allow for more control over the output for the backwards error.

2.1.9:

```
f(n) = return exp(n)
y = f(2)
y_hat = f(y)
println("y: ", y) # y: 7.38905609893065
println("y_hat: ", y_hat) # y_hat: 1618.1779919126539

g(n) = return exp(-n)
x = g(2)
x_hat = g(x)
println("x: ", x) # x: 0.1353352832366127
println("x_hat: ", x_hat, '\n') # x_hat: 0.8734230184931167

k_f(c) = return abs((c * exp(c)) / exp(c))
println("K relative for f(): ", k_f(2)) # K relative for f(): 2.0

k_g(d) = return abs((d * -exp(d)) / exp(d))
println("K relative for g(): ", k_g(2), '\n') # K relative for g(): 2.0

forward(n_hat, n) = return abs(n_hat - n) / abs(n)

f_bool(z) = return forward(y_hat, y) >= (k_f(2) * z)
g_bool(z) = return forward(x_hat, x) >= (k_g(2) * z)

println(f_bool(10^-2)) # true
println(g_bool(10^-2), '\n') # true

# 2.1.14:
f_sin(x) = return sin(x)
f_prime(h) = return (f_sin(1 + h) - f_sin(1)) / h
f_prime2(h) = return (f_sin(1 + h) - f_sin(1 - h)) / (2 * h)

println(f_prime(10^-1)) # 0.4973637525353891
println(f_prime2(10^-1), '\n') # 0.53940225216976
```

```
println(f_prime(10^-2)) # 0.5360859810118689
println(f_prime2(10^-2), '\n') # 0.5402933008747334

println(f_prime(10^-3)) # 0.5398814803603269
println(f_prime2(10^-3), '\n') # 0.5403022158176896

println(f_prime(10^-14)) # 0.5440092820663267
println(f_prime2(10^-14), '\n') # 0.5440092820663267

println(f_prime(10^-15)) # 0.5551115123125783
println(f_prime2(10^-15), '\n') # 0.5551115123125783

println(f_prime(10^-16)) # 0.0
println(f_prime2(10^-16), '\n') # 0.5551115123125783

println(f_prime(10^-17)) # 0.0
println(f_prime2(10^-17), '\n') # 0.0

println(f_prime(10^-18)) # 0.0
println(f_prime2(10^-18), '\n') # 0.0
```

Explanation:
Once we hit 10^-14 for our h value, we start to hit the limitations of floating-point representation for our computers. When we exceed the level of precision needed, our CPU begins to panic and will just convert the number to "0.0" to continue operating.