E008 - Largest product in a series

Given a string 1000-digit number:

1. Find the greatest product of four consecutive digits in the 1000-digit number. (warm up)
2. Find the greatest product of five consecutive digits in the 1000-digit number.
3. Find the thirteen adjacent digits in the 1000-digit number that have the greatest product. What is the value of this product?

```
const SERIE1000 =
"7316717653133062491922511967442657474235534919493496983520312774506326239578318016984801869478851843858615607891129
```

```
const SERIE1000 = """
73167176531330624919225119674426574742355349194934
96983520312774506326239578318016984801869478851843
85861560789112949495459501737958331952853208805511
12540698747158523863050715693290963295227443043557
66896648950445244523161731856403098711121722383113
62229893423380308135336276614282806444486645238749
30358907296290491560440772390713810515859307960866
70172427121883998797908792274921901699720888093776
65727333001053367881220235421809751254540594752243
52584907711670556013604839586446706324415722155397
53697817977846174064955149290862569321978468622482
83972241375657056057490261407972968652414535100474
82166370484403199890008895243450658541227588666881
16427171479924442928230863465674813919123162824586
17866458359124566529476545682848912883142607690042
24219022671055626321111109370544217506941658960408
07198403850962455444362981230987879927244284909188
84580156166097919133875499200524063689912560717606
05886116467109405077541002256983155200055935729725
71636269561882670428252483600823257530420752963450
""" |> s -> split(s, "\n") |> a -> join(a)
```

```
const SERIE18 = "134506207689915732"
```

```
const SERIE18 = "134506207689915732"
```

1. Brute Force approach

largest_bf (generic function with 1 method)

```
function largest_bf(serie::String, len::Int; verbose=false)
    largest = 0
    for ix ∈ 1:length(serie) - len
        prod = 1
        for jx ∈ 0:len - 1
            d = parse(Int64, serie[ix + jx])
            d == 0 && break
            prod *= d
            # verbose && print("$(d) x ")
        end
        # verbose && println(" => $(prod)")
        prod > largest && (largest = prod)
    end
    largest
end
```

3888

```
@time @show largest_bf(SERIE18, 4)
```

```
largest_bf(SERIE18, 4) = 3888
  0.000176 seconds (44 allocations: 201.297 KiB)
```

27216

```
@time @show largest_bf(SERIE18, 5; verbose=true)
```

```
largest_bf(SERIE18, 5; verbose = true) = 27216
  0.000142 seconds (46 allocations: 201.516 KiB)
```

40824

```
@time @show largest_bf(SERIE1000, 5)
```

```
largest_bf(SERIE1000, 5) = 40824
  0.000209 seconds (106 allocations: 204.203 KiB)
```

285768

```
@time @show largest_bf(SERIE1000, 6)
```

```
largest_bf(SERIE1000, 6) = 285768
  0.000216 seconds (108 allocations: 204.688 KiB)
```


23514624000

```
@time @show largest_bf(SERIE1000, 13)
```

```
largest_bf(SERIE1000, 13) = 23514624000
  0.000228 seconds (107 allocations: 204.391 KiB)
```

2. Less of a Brute Force approach

Skipping ahead id 0 if found

largest_v2 (generic function with 1 method)

```
function largest_v2(serie::String, len::Int; verbose=false)
    largest = 0
    ix, limit = 1, length(serie) - len
    offset = 1
    while ix ≤ limit
        prod = 1
        for jx ∈ 0:len - 1
            d = parse(Int64, serie[ix + jx])
            if d == 0
                # skipping the 0 in a product
                prod = 0
                offset = jx + 1
                break
            end
            prod *= d
            # verbose && print("$(d) x ")
        end

        if prod > 0
            # verbose && println(" => $(prod)")
            prod > largest && (largest = prod)
            ix += 1
        else
            ix += offset
        end
    end

    largest
end
```

@time @show largest4b = 3888

```
@time @show largest4b = largest_v2(SERIE18, 4; verbose=true)
```

```
largest4b = largest_v2(SERIE18, 4; verbose = true) = 3888
  0.004449 seconds (114 allocations: 205.250 KiB, 95.96% compilation time)
```

27216

```
@time @show largest_v2(SERIE18, 5; verbose=true)
```

```
largest_v2(SERIE18, 5; verbose = true) = 27216
  0.000106 seconds (46 allocations: 201.516 KiB)
```

40824

```
@time @show largest_v2(SERIE1000, 5)
```

```
largest_v2(SERIE1000, 5) = 40824
  0.000140 seconds (44 allocations: 201.312 KiB)
```

285768

```
@time @show largest_v2(SERIE1000, 6)
```

```
largest_v2(SERIE1000, 6) = 285768
  0.000152 seconds (46 allocations: 201.797 KiB)
```

23514624000

```
@time @show largest_v2(SERIE1000, 13)
```

```
largest_v2(SERIE1000, 13) = 23514624000
  0.000149 seconds (45 allocations: 201.500 KiB)
```

3. DP approach

[DP] Dynamic Programming

largest_dp (generic function with 1 method)

```
function largest_dp(serie::String, len::Int)
    largest, zeros = 0, 0
    prod = 1

    for ix ∈ 1:length(serie)
        if ix > len
            # need to update product by "cancelling" effect of leftmost digit in t
            # need to avoid division by zero!
            old_digit = serie[ix - len]
            if old_digit == '0'
                zeros -= 1
            else
                prod ÷= old_digit - '0' # using ascii offset, rather than parsing
            end
        end

        # serie[ix] is our next digit - either it is a 0 or not
        if serie[ix] == '0'
            zeros += 1
        else
            prod *= serie[ix] - '0'
        end

        if ix > len && zeros == 0 && prod > largest
            largest = prod
        end
    end

    largest
end
```

27216

```
@time @show largest_dp(SERIE18, 5)
```

```
largest_dp(SERIE18, 5) = 27216
  0.000189 seconds (106 allocations: 204.188 KiB)
```

27216

```
@time @show largest_dp(SERIE18, 5)
```

```
largest_dp(SERIE18, 5) = 27216
  0.000126 seconds (44 allocations: 201.297 KiB)
```

40824

```
@time @show largest_dp(SERIE1000, 5)
```

```
largest_dp(SERIE1000, 5) = 40824
  0.000158 seconds (44 allocations: 201.312 KiB)
```

285768

```
@time @show largest_dp(SERIE1000, 6)
```

```
largest_dp(SERIE1000, 6) = 285768
  0.000154 seconds (107 allocations: 204.391 KiB)
```

23514624000

```
@time @show largest_dp(SERIE1000, 13)
```

```
largest_dp(SERIE1000, 13) = 23514624000
  0.000098 seconds (45 allocations: 201.500 KiB)
```

📖 Table of Contents