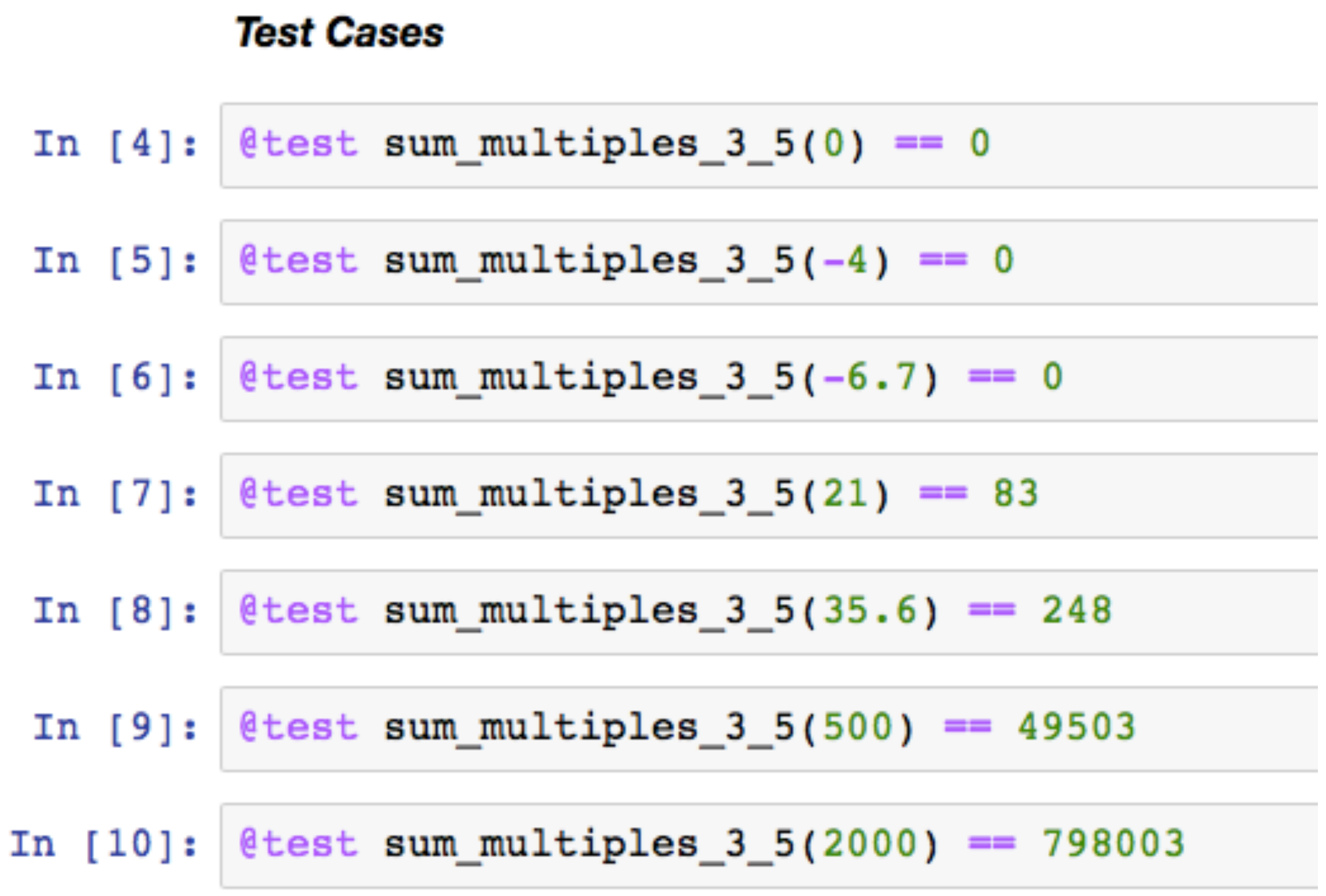
CS 696 Intro to Big Data: Tools and Methods
Fall Semester, 2016
Doc 8 Assignment 1 Comments
Sep 20, 2016

Dont use rar to encode your files



Test Cases

```
In [4]: @test sum_multiples_3_5(0) == 0

In [5]: @test sum_multiples_3_5(-4) == 0

In [6]: @test sum_multiples_3_5(-6.7) == 0

In [7]: @test sum_multiples_3_5(21) == 83

In [8]: @test sum_multiples_3_5(35.6) == 248

In [9]: @test sum_multiples_3_5(500) == 49503

In [10]: @test sum_multiples_3_5(2000) == 798003
```

Testset for Julia 0.4

```
using BaseTestNext
const Test = BaseTestNext
@testset "Sample" begin
  @test 1 == 2
  @test 1 == 1
end
```

```
foo(x) = length(x)^2

@testset "Arrays $i" for i in 1:3
     @test foo(zeros(i)) == i^2
     @test foo(ones(i)) == i^2
end
```

```
Test Summary: | Pass  Fail  Total
Sample        |    1     1      2
```

Names

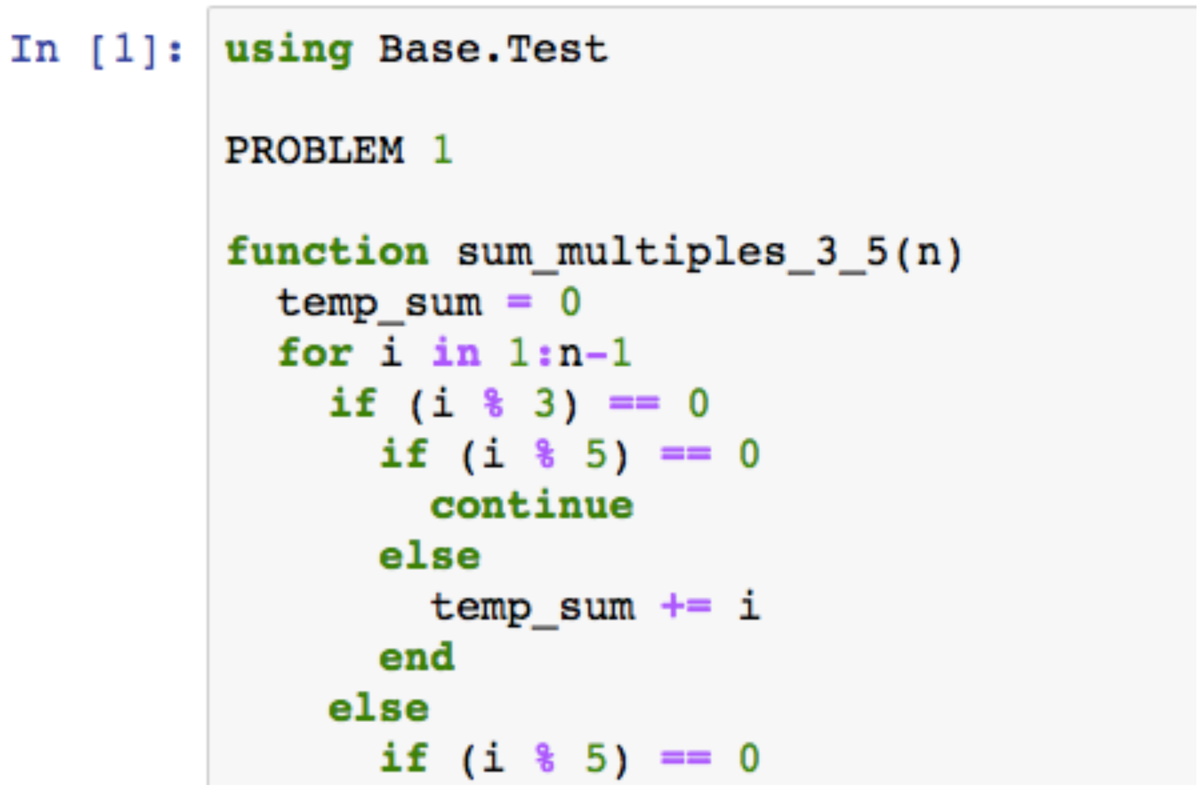
```
function getWordMatchCount(baseString::AbstractString, searchString::AbstractString,
includeOverlap::Bool = false)

function most_frequent_word(str::AbstractString, n::Integer)

function isIntegerAnOctaldrome(currentValue::Integer)
function nth_octaldrome(n::Integer)
function pattern_count(text,pattern)
    if typeof(text) == ASCIIString && typeof(pattern) == ASCIIString
        ArrText = bytestring(text)
        ArrPattern = bytestring(pattern)
        lenText=length(ArrText)
        lenPattern=length(ArrPattern)
```

```
function pattern_count(text::ASCIIString,pattern:: ASCIIString)
    ArrText = bytestring(text)
    ArrPattern = bytestring(pattern)
    lenText=length(ArrText)
    lenPattern=length(ArrPattern)
```

```
function pattern_count(text:: ASCIIString,pattern:: ASCIIString)
    array_text = bytestring(text)
    array_pattern = bytestring(pattern)
    text_len =length(array_text)
    pattern_len=length(array_pattern)
```

```
function nth_octaldrome(n::Int64)
    if typeof(n) == Int64
```

```
@test nth_octaldrome('c') == -1
@test nth_octaldrome("shdkd") == -1
```



```
In [1]: using Base.Test

        PROBLEM 1

        function sum_multiples_3_5(n)
          temp_sum = 0
          for i in 1:n-1
            if (i % 3) == 0
              if (i % 5) == 0
                continue
              else
                temp_sum += i
              end
            else
              if (i % 5) == 0
```

```
function most_frequent_word(input_string::ASCIIString, word_length::Int64)
    if(word_length<0) then
        word_lenth = 0
    end

    stuff removed so will fit on slide

    println(most_frequent_words)
  return most_frequent_word
end
```

println(most_frequent_words) / **return most_frequent_word**

```
function foo(n)
  n + 1
end

bar = foo

bar(3)
```

```
function gc_content(s::ASCIIString)
     total_length = length(s)
     count_c = 0
     count_g = 0
     gc_count = 0

    if(total_length>0) then
     for i in 1:length(s)
        if(s[i]=='G')
          count_g +=1
        elseif(s[i]=='C')
          count_c +=1
        end
    end
      end
       gc_count = count_g + count_c

       if gc_count > 0
          gc_content = gc_count/total_length
          println(gc_content)
         return gc_content
      else
        return gc_content
```

```
     if gc_count > 0
        gc_content = gc_count/total_length
        println(gc_content)
       return gc_content
    else
      return gc_content
    end
end
```

```
cat = 3
if cat > 4
  zoo = 4
else
  zoo
end
```

```
UndefVarError: zoo not defined
```

```
cat = 3
zoo2::Int64
if cat > 4
  zoo2 = 4
else
  zoo2
end
```

```
UndefVarError: zoo2 not defined
```

```
cat = 3
zoo3 = 0
if cat > 4
  zoo3 = 4
else
  zoo3
end
```

```
0
```

```
function foo(n)
  bar::Int64
  if n > 5
    bar = 1
  end
  bar
end
```

```
foo(2)
```

```
UndefVarError: bar not defined
```

```
function pattern_count(text::ASCIIString,pattern::ASCIIString)
```

```
using Base.Test
@test pattern_count("ababa","ba") == 2
@test pattern_count("abababa","BA") == 0
@test pattern_count(1232323,56) == 0
@test pattern_count("21876721",213212) == 2
@test pattern_count(21232121,"21") == 3
```

```
function digit_distribution(RA)
    inputType = typeof(RA)
    retDict = Dict{Int64,Int64}()
    #ensure the inpout array is either a Float of Int type
    if (inputType == Array{Float64,1} || inputType == Array{Int,1} )
         blah
    else
        println("I am sorry, $RA is not a valid input")
        return nothing
    end
    return retDict
end
```

```
function sum_multiples_3_5(number)
  if typeof(number) != Int64 || number <=0
    return "invalid input"
  end
  etc.

function sum_multiples_3_5(number::Int64)
  if number <=0
    return "invalid input"
  end

function sum_multiples_3_5(number::Integer)
  if number <=0
    return "invalid input"
  end

function sum_multiples_3_5(number::Integer)
  if number <=0
    error(“Invalid argument $(number), number must be positive”)
  end
```

```
function most_frequent_word(x::AbstractString, y::Integer)
  word = x
  step = y
  found=Array(AbstractString,(cld(length(word),step)))
  for w in 1:1:length(found)
    found[w]=""
  end
  i = 1
  while i <= (length(word) - step + 1)
    j = i + step -1
    searchWord=word[i:j]
    amountFound = 0
    q = 1
    while q <= (length(word) - step + 1)
      if word[q:(q+step-1)] == searchWord
        amountFound += 1
      end
      q += 1
    end
    tempString=found[amountFound]
    tempArr=split(tempString)
    nameFound=0
    if length(tempArr) != 0
      for r in 1:1:length(tempArr)
        if tempArr[r] == searchWord
          nameFound=1
        end
      end
    end
    if nameFound == 0
      found[amountFound]=string(tempString," ",searchWord)
    end
    i += 1
  end
  p=length(found)
  while p > 0
    if sizeof(found[p]) > 0
      templine=found[p]
      println(templine)
      break
    end
    p -= 1
  end
end
```

```
tempString=found[amountFound]
tempArr=split(tempString)
nameFound=0
if length(tempArr) != 0
  for r in 1:1:length(tempArr)
    if tempArr[r] == searchWord
      nameFound=1
    end
  end
end
```

```
function sum_multiples_3_5(x::Integer)
  y=0
  z=0
  while z<=x
    if (z%5 == 0) && (z%3 == 0)
    elseif (z%5 == 0)
      y+=z
    elseif (z%3 == 0)
      y+=z
    end
    z=z+1
  end
  println(y)
end
```

```
"1. Returns sum of multiples of 3 or 5, but not multiple of both"
function sum_multiples_3_5(N::Int)
  sumOfMultiples = 0
  if(N == 3)
    return sumOfMultiples + N
  end
  for i = 3:N-1
    if(i % 3 == 0 && i % 5 == 0)
      continue
    elseif(i % 3 == 0)
      sumOfMultiples += i
    elseif(i % 5 == 0)
      sumOfMultiples += i
    end
  end

  return sumOfMultiples
end
```

```
function most_frequent_word(text,n)
len = length(text)
i=1
arr = String[]
res = String[]
while i <= len-n+1
    list = text[i:i+(n-1)]
    unshift!(arr,list)
    i +=1
end
len=length(arr)
j=1
count=0
while j < len
    tempcount=0
    i=j+1
    while i <= len
        if arr[i]== arr[j]
        tempcount+=1
        end
```

```
function nth_octaldrome(n::Int64)
  i=0
  counter = n
  while counter != 0
        i+=1
    oct_number =  oct(i)
    if oct_number == reverse(oct_number)
      counter = counter-1
    end
  end
  return i
end
```

i?????

```
function sum_multiples_3_5(input::Int64)
  sum=0
  for i = 1:input-1
    if i % 3 == 0 || i % 5 ==0
      if i % 5 != 0 || i % 3 != 0
        sum = sum + i
      end
    end
  end
  return sum
end
```

```
function sum_multiples_3_5(input::Int64)
  sum=0
  for i in 1:input-1
    if (i % 3 == 0 || i % 5 ==0) && ( i % 15 !=0)
        sum = sum + i
    end
  end
  sum
end
```

```
is_multiple_3_5_not_15(k) =
     (k % 3 == 0 || k % 5 ==0) && ( k % 15 !=0)
```

```
function sum_multiples_3_5(input::Int64)
  sum=0
  for k in 1:input-1
    if is_multiple_3_5_not_15(k)
        sum = sum + k
    end
  end
  sum
end
```

```
# Julia 0.5
sum_multiples_3_5(n) =
    sum(k for k in 1:n-1
            if is_multiple_3_5_not_15(k))
```

```
function most_frequent_word(sentence::AbstractString, max_len::Int64)
 ans = []
 number_occurance = 0
 for i = 1: length(sentence)-max_len+1
  counter = 0
  y = sentence[i:i+max_len-1]
  for j = i: length(sentence)-max_len+1
   if sentence[j:j+max_len-1] == y
    counter = counter +1
   end
  end
  if counter >= number_occurance
   if number_occurance < counter
    ans = []
   end
   number_occurance = counter
   push!(ans, y)
  end
 end
 return ans
end
```

What is i? j?

```
function digit_distribution(integers)
  a = Dict()
  number_str = join(integers)
  for i in 0:9
    counter = 0
    for j= 1:length(number_str)
      if number_str[j] != '.'
        if parse(Int, number_str[j]) == i
          counter = counter + 1
        end
      end
    end
    if counter > 0
      b = Dict(i=>counter)
      a = merge(a,b)
    end
  end
  return a
end
```

```
function sum_multiples_3_5(n)
    if typeof(n) != Int64
        error("Please enter an integer value for n")
    elseif(n<0)
        error("Please make sure n is positive")
    end
    sum = 0
    for i=1:n-1
        if(i%3==0) || (i%5==0)
            if(i%3==0) && (i%5==0)
                continue
            else
                sum = sum+i
            end
        end
    end
    return sum
end
```

```
function sum_multiples_3_5(N)
    sum_multiples = 0
    if N > -1
        for i in 1:N-1
            if (i%3 == 0 || i%5 == 0) && i%15 != 0
                sum_multiples = sum_multiples + i
            end
        end
    end
    return(sum_multiples)
end
```

```
function sum_multiples_3_5(N)
    if N < 3
        return 0
    end
    sum_multiples = 0
    for i in 1:N-1
        if (i%3 == 0 || i%5 == 0) && i%15 != 0
            sum_multiples = sum_multiples + i
        end
    end
    sum_multiples
end
```

```
function sum_multiples_3_5(n)
 if typeof(n)!= Int64 || (n<0)
  error("please enter a value which is greater than zero or Int64 pattern!")
 end
 finalSum :: Int64 = 0
 for i = 1:n-1
   if (mod(i,15) == 0)
     continue
   elseif ((mod(i,3) == 0) || (mod(i,5) == 0))
     finalSum += i
    end
 end
 return finalSum
end
```

```
function digit_distribution(number_array)
  digit_count_dict = Dict{Int64,Int64}()
  array_element = join(number_array)
  for i = 0:9
    count = 0
    for j in array_element
      if j != '.'
        if i == parse(Int, j)
          count = count + 1
          digit_count_dict[i] = count
        end
      end
    end
  end
  return digit_count_dict
end
```

```
function pattern_count(text::AbstractString,pattern::AbstractString)
a = searchindex(text,pattern)
count = 0;
 while a > 0
   count = count+1
   a= a + 1
   a = searchindex(text,pattern,a)
 end
 return count
end
```

```
function sum_multiples_3_5(number)
        if typeof(number)!= Int64 || number <=0
                return "Please enter a valid number"
        end
    reverse_Multiples=0
    for i in 1:number-1
        if ((mod(i,15)!=0)&&((mod(i,3)==0)||(mod(i,5)==0)))
                        reverse_Multiples = reverse_Multiples +i
        end
    end
        return reverse_Multiples
end
```