R Notebook example: Hello World

What is a library?

Number of Cores

```
# Cntrl-Alt-I is your friend

library(parallel);          # install.packages("parallel", dependencies=TRUE);
parallel::detectCores();
```

```
## [1] 8
```

```
# Cntrl-Shift-C is also your friend
```

Can you find another library that counts cores? Is it a core or a thread? My laptop is a QUAD-core.

Prime Numbers

```
library(pracma);
pracma::primes(22);
```

```
## [1]  2  3  5  7 11 13 17 19
```

```
prime.pracmaPrimes = function(n, first=FALSE)
    {
    # this duplicates the primary logic of pracma::primes
    # by computing 'sqrt' one time, it speeds up things 'slightly'
    # it allows for firstN or fromN with first=FLAG
    gn = n;
    if(first) { gn = ceiling( n * log(n) + n * log(log(n)) ); }
    gn.sqrt = floor( sqrt(gn) );  # needs to round down so the "seq by k" doesn't break ...

    p = seq(1, gn, by = 2); # odd numbers
    q = length(p);
    p[1] = 2;    # replace 1 with 2 (prime)
                 # 9 is the first non-prime?
    if(gn >= 9)
        {
        for (k in seq(3, gn.sqrt, by = 2) )
            {
            k.idx = (k + 1)/2;
            if (p[k.idx] != 0)
                {
                # using a squared rule on indexes ?
                k2.idx = (k * k + 1)/2;

                # cat("\n", " k = ", k, "... (k+1/2) = ", k.idx, "... (k * k + 1)/2 = ", k2.idx, "\n");
```

```
                p[ seq(k2.idx, q, by = k) ] = 0;
                }
            }
        }

    p = p[p > 0];

    if(first)
        {
        p[1:n];
        } else {
                p[p < n];
                }
    }
```

```
prime.pracmaPrimes(22);
```

```
## [1]  2  3  5  7 11 13 17 19
```

```
prime.pracmaPrimes(22, first=TRUE);
```

```
##  [1]  2  3  5  7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79
```

Benchmarking

```
library(microbenchmark);
microbenchmark::microbenchmark(
                                pracma::primes(22) ,
                                prime.pracmaPrimes(22),
                                prime.pracmaPrimes(22, first=TRUE)
                                );
```

```
## Unit: microseconds
##                                  expr   min     lq    mean median     uq    max
##                     pracma::primes(22) 220.0 226.85 258.166 232.40 245.25  671.0
##                  prime.pracmaPrimes(22) 214.6 220.85 252.028 225.45 235.05  601.9
##  prime.pracmaPrimes(22, first = TRUE) 370.7 380.50 450.122 385.75 449.75 1039.8
##  neval cld
##    100  a
##    100  a
##    100   b
```

One line of code is inefficient in 'pracma', can you find it?

```
pracma::primes
```

```
## function (n)
## {
##     if (!is.numeric(n) || length(n) != 1)
##         stop("Argument 'n' must be a numeric scalar.")
##     n <- floor(n)
##     if (n < 2)
##         return(c())
##     p <- seq(1, n, by = 2)
##     q <- length(p)
##     p[1] <- 2
```

```
##     if (n >= 9) {
##         for (k in seq(3, sqrt(n), by = 2)) {
##             if (p[(k + 1)/2] != 0)
##                 p[seq((k * k + 1)/2, q, by = k)] <- 0
##         }
##     }
##     p[p > 0]
## }
## <bytecode: 0x0000000013d73650>
## <environment: namespace:pracma>
```

The 'sqrt(n)' is not changing but is being computed several times in the 'k' sequence.

Does 'hello world' work?

```
source("https://raw.githubusercontent.com/MonteShaffer/MasterClassDataAnalytics/main/functions/functions
hello();
```

```
## [1] "hello world"
##
##  hello there  ...  world
```

Does Python Work?

```
library(reticulate);        # install.packages("reticulate", dependencies=TRUE);
reticulate::use_python("C:/Python/Python39")
```

Pay attention to detail. This will only work if all the building blocks are in place.

```
print("hello world");
```

```
## hello world
```

```
def pownum(base, pow):
  return base ** pow

print(pownum(9,5));
```

```
## 59049
```

```
print(pownum(3,5));
```

```
## 243
```

Does C++ Work?

```
library(Rcpp);        # install.packages("Rcpp", dependencies=TRUE);
# https://stackoverflow.com/questions/64839024/
```

Bits and Such

```
Rcpp::cppFunction("long long RShift(long long a, int b) { return a >> b;}")
```

Shifty shifting

The "right shift" operator in R is based on S+ and has some limitations when it comes to signed (negative) integers.

```
y = 1732584194;
RShift(y, 16);
```

```
## [1] 26437
```

```
bitwShiftR(y, 16);
```

```
## [1] 26437
```

```
y = -1732584194;
RShift(y, 16);
```

```
## [1] -26438
```

```
bitwShiftR(y, 16);
```

```
## [1] 39098
```

Libraries 'bit' and 'bit64' may be of some benefit, but beforewarned when working with bits using R.

```
bitShiftR = function(x, bits, unsigned=FALSE)
  {
  if(!is.negative(x) | unsigned) { return( bitwShiftR(x,bits) ); }
  -bitwShiftR(-x,bits) - 1; #  - 1;                  # >>>
  }

is.negative = function(x, ..., tol = sqrt(.Machine$double.eps), part="Re")
  {
  more = unlist(list(...)); x = c(x, more);
  x = if(part == "Im") { x = Im(x); } else { x = Re(x); }
  x < ( -1 * tol );
  }

y = -1732584194;
RShift(y, 16);
```

```
## [1] -26438
```

```
bitwShiftR(y, 16);
```

```
## [1] 39098
```

```
bitShiftR(y, 16);
```

```
## [1] -26438
```

Convert decimal number to a binary string (and vice versa)

```
# https://stackoverflow.com/questions/6614283/converting-decimal-to-binary-in-r

dec2bin = decbin = function(decnum)
    {
    bvect = rep(0, 1 + floor(log(decnum, 2))); # pre-populate with zeroes
    while (decnum >= 2)
```

```
        {
        power = floor(log(decnum, 2));
        bvect[1 + power] = 1;
        decnum = decnum - 2^power;
        }
    bvect[1] = decnum %% 2;
    paste(rev(bvect), collapse = ""); # convert to a string, reversed
    }
```

decbin function

```
decbin(57);
```

decbin example

```
## [1] "111001"
```

If you write a function, you should also have its inverse.

```
# two names for the function
  # bin2dec is matlab
  # bindec is PHP
bin2dec = bindec = function(binstr)
  {
  n = strlen(binstr);
    res = 0; power = 0;
    for(i in n:1) # we reversed it in the for loop
        {
        bit = as.integer(charAt(binstr,i));
        add = 0;
        if(bit == 1) { add = 2^power; }

        res = res + add;
        power = 1 + power;
        }
    res;
  }
```

```
## bin2dec('111001');  # you may want to comment this out when you Knit-HTML as it may throw an "intent
```

bindec function

```
strlen = function(str)
  {
  # history :: # https://en.cppreference.com/w/c/string/byte/strlen
  # http://www.cplusplus.com/reference/cstring/
  # https://en.wikipedia.org/wiki/C99
  # https://www.programiz.com/c-programming/library-function/string.h/strlen
  # vectorized ... already
  nchar( as.character(str), type="chars");
  }
```

```
charAt = function(str,idx)
  {
  substr(str,idx,idx);
  }
```

helper functions

```
bin2dec('111001');
```

bindec example

```
## [1] 57
```

```
bindec('111001');
```

```
## [1] 57
```

```
bindec( decbin(57) );
```

INVERSE check

```
## [1] 57
```

```
decbin( bindec('111001') );
```

```
## [1] "111001"
```

```
typeof( 57 );
```

```
## [1] "double"
```

```
typeof( decbin(57) );
```

```
## [1] "character"
```

String PAD zeroes You could left-side 'strPadLeft' with zeroes if you wanted it to be a certain bit length

```
strPadLeft = function(str, final.str.len, padding="0", method="stringi")
  {
  if( isTRUE(requireNamespace("stringi", quietly = TRUE)) && method=="stringi" )
    {
    stringi::stri_pad_left(str, final.str.len, pad = padding);
    } else {
            n = strlen(str);
            r = final.str.len - n;
            if(r < 0) { stop("strPadLeft is too short!"); }
            paste0(paste(rep(padding,r),collapse=""),str);

            }
  }
```

```
strPadLeft( decbin(57), 8);
```

```
## [1] "00111001"
```

```
strPadLeft( decbin(57), 8, method="base");
```

```
## [1] "00111001"
```

```
strPadLeft( decbin(57), 8, method="Adljblkjadlk");
```

```
## [1] "00111001"
```

```
library(microbenchmark);
microbenchmark::microbenchmark(
                                strPadLeft( decbin(57), 8) ,
                                strPadLeft( decbin(57), 8, method="base"),
                                strPadLeft( decbin(57), 8, method="Adljblkjadlk")
                                );
```

Benchmarking speed

```
## Unit: microseconds
##                                                 expr  min    lq    mean median
##                             strPadLeft(decbin(57), 8) 88.2 91.20 106.962  93.50
##          strPadLeft(decbin(57), 8, method = "base") 94.2 96.85 106.231  98.80
##  strPadLeft(decbin(57), 8, method = "Adljblkjadlk") 94.2 97.10 106.120  98.75
##      uq   max neval cld
##   97.75 332.8   100   a
##  103.70 309.2   100   a
##  103.10 304.2   100   a
```

We are also benchmarking the 'decbin' function which likely can also be improved upon. It is a good idea to isolate what you are actually timing, but testing in context is not a bad idea.

Since the library 'stringi' is written in C++, it has some native efficiencies over the R interpreted 'base' solution. [https://cran.r-project.org/web/packages/stringi/index.html]

Matrices with External C++ file

This will source and compile the code. Maybe give it a minute.

```
Rcpp::sourceCpp("multiply.cpp");
```

```
## Registered S3 methods overwritten by 'RcppEigen':
##   method               from
##   predict.fastLm       RcppArmadillo
##   print.fastLm         RcppArmadillo
##   summary.fastLm       RcppArmadillo
##   print.summary.fastLm RcppArmadillo
```

```
A = matrix(rnorm(10000), 100, 100); # fully populated, 100 x 100, relatively small
B = matrix(rnorm(10000), 100, 100);

# base::matrix
```

```
library(microbenchmark);
microbenchmark::microbenchmark(
                                eigenMatTrans(A),
                                A%*%B,
                                armaMatMult(A, B),
                                eigenMatMult(A, B),
                                eigenMapMatMult(A, B)
                                );
```

```
## Unit: microseconds
##                   expr    min      lq     mean  median      uq     max neval
##       eigenMatTrans(A)  173.4  196.30  281.232  218.75  272.55  4043.0   100
##                A %*% B 2891.0 2937.05 3240.270 3036.00 3266.65  5459.6   100
##      armaMatMult(A, B) 2909.2 2968.25 3564.796 3148.10 3384.65 11155.9   100
##     eigenMatMult(A, B)  623.1  686.25 4644.049 1739.95 4970.05 28132.9   100
##  eigenMapMatMult(A, B)  568.2  623.05 4423.513 1284.25 5291.10 34963.1   100
##  cld
##   a
##    b
##    b
##    b
##    b
```

It appears 'eigen' performs well for me; 'arma' is about equivalent to the built-in R multiplication.

Can we use sparse matrices and pass them into C++? Can we pass an 'R' sparse matrix into a C++ function call for speed purposes?

Imagery

Two powerful C/C++ tools now accessible within R.

```
library(magick); #install.packages("magick", dependencies=TRUE);
```

```
## Linking to ImageMagick 6.9.12.3
## Enabled features: cairo, freetype, fftw, ghostscript, heic, lcms, pango, raw, rsvg, webp
## Disabled features: fontconfig, x11
```

```
# https://cran.r-project.org/web/packages/magick/vignettes/intro.html#The_grid_package
# https://www.datanovia.com/en/blog/easy-image-processing-in-r-using-the-magick-package/
# https://www.imagemagick.org/discourse-server/viewtopic.php?t=18433

#
image_content = function(x, ...)
  {
  x = magick::image_data(x, ...);
  as.integer(x);
  }

tiger         = magick::image_read_svg('http://jeroen.github.io/images/tiger.svg', width = 350);

tiger;
```

```
tiger.png     = magick::image_convert(tiger, "png");
tiger.png;
```

```
tiger.matrix  = image_content(tiger.png);
dim(tiger.matrix);
```

```
## [1] 350 350   4
```

```
# 3D array
# tiger.matrix[,,1];  # x,y, z ... z is likely RGBa = 4 layers

# tiger.matrix;
```

```
library(tesseract); # install.packages("tesseract");
# https://github.com/ropensci/magick/issues/154

img.file = "iris-ocr-intro.png";
img = magick::image_read( img.file );
#img.txt = tesseract::image_ocr(img);  # renamed during LIBRARY upgrade?
img.txt = tesseract::ocr(img);

cat(img.txt);
```

```
## + I. Inrropuction
```

```
img.file = "iris-ocr.png";

img = magick::image_read( img.file );
# img.txt = tesseract::image_ocr(img);  # renamed during LIBRARY upgrade?
```

```
img.txt = tesseract::ocr(img);

cat(img.txt);
```

```
## Geneticist to the Missouri Botanical Garden
## Professor of Botany in the Henry Shaw School of Botany of Washington University
## I. Iytropuction
##
## Asa biological phenomenon the species problem is worthy of
## serious study as an end in itself, and not as a mere corollary to
## work in some other field. It is, to be sure, a problem so funda-
## mentally important that it touches many such fields. Workers
## in any one of these are humanly prone to regard the evidence
## from that field as all important and its techniques as all suffi-
## cient (particularly if they are themselves unacquainted with
## other aspects of the problem). When, however, one takes up the
## problem, as a problem, and studies it from the diverse view-
## points of genetics, taxonomy, cytology, and biometry, he real-
## izes that he not only needs most of the existing techniques but
## that he must devise new ones as well.
```

Does Java Work?

Natural language processing requires java running under the hood.

```
library(openNLP);  # this requires rJava ... Java
library(NLP);

sentence.a = openNLP::Maxent_Sent_Token_Annotator();
word.a     = openNLP::Maxent_Word_Token_Annotator();
```

```
s = anna = "Happy families are all alike; every unhappy family is unhappy in its own way.";

sw.a       = NLP::annotate(s, list(sentence.a, word.a));

pos.a      = openNLP::Maxent_POS_Tag_Annotator(probs=TRUE);
swpos.a    = NLP::annotate(s, list(pos.a), sw.a);
```

```
swpos.a.words = subset(swpos.a, type=="word");

(swpos.a.words);
```

```
##  id type start end features
##   2 word     1   5 POS=JJ, POS_prob=0.8770581
##   3 word     7  14 POS=NNS, POS_prob=0.9943596
##   4 word    16  18 POS=VBP, POS_prob=0.993953
##   5 word    20  22 POS=RB, POS_prob=0.4868905
##   6 word    24  28 POS=RB, POS_prob=0.8186156
##   7 word    29  29 POS=:, POS_prob=0.9326554
##   8 word    31  35 POS=DT, POS_prob=0.9445861
##   9 word    37  43 POS=JJ, POS_prob=0.9951879
##  10 word    45  50 POS=NN, POS_prob=0.9890899
##  11 word    52  53 POS=VBZ, POS_prob=0.9826753
##  12 word    55  61 POS=JJ, POS_prob=0.9860051
```

```
##  13 word    63  64 POS=IN, POS_prob=0.994442
##  14 word    66  68 POS=PRP$, POS_prob=0.9906345
##  15 word    70  72 POS=JJ, POS_prob=0.9929793
##  16 word    74  76 POS=NN, POS_prob=0.9987191
##  17 word    77  77 POS=., POS_prob=0.9947943
```

```
# use substring to map back to the original value
substring(s, swpos.a.words$start, swpos.a.words$end);
```

```
##  [1] "Happy"    "families" "are"      "all"      "alike"    ";"
##  [7] "every"    "unhappy"  "family"   "is"       "unhappy"  "in"
## [13] "its"      "own"      "way"      "."
```

```
# NOT a dataframe, so will have to further manipulate
```

```
# seldom will I use an "apply" macro.
tags = sapply(swpos.a.words$features, `[[`, "POS");
sort(table(tags),decreasing = TRUE);
```

```
## tags
##   JJ   NN   RB    .    :   DT   IN  NNS PRP$  VBP  VBZ
##    4    2    2    1    1    1    1    1    1    1    1
```