Leibniz-Universität Hannover
Institut für Kartographie & Geoinformatik
Prof. Dr. Martin Werner

Big Geospatial Data (Summer Term 2017)
Tutorial 1

Literatur: Blog zu Functional Programming (3 Pages)
http://blogs.mathworks.com/loren/2013/01/10/introduction-to-functional-programming-with-anonymous-functions-part-1/

Aufgabe 1: Functional Programming in MATLAB

Read the blog given above. In this blog consisting of three articles, you are introduced to functional programming using MATLAB anonymous functions.

- Read about functional programmming, for example at *https://en.wikipedia.org/wiki/Functional_programming*
- Inform yourself about anonymous functions and the varargin mechnaism for variable argument lists.
- Recall the central idea of functional programming and think about its purpose, pros and cons.
- Implement Euklids algorithm for the greatest common divisor. Therefore, complete the following MATLAB file (Tip: The loop construct can easily be extended to a state holding a and b and the function curly can be used to finally extract the second part of the state, which holds the result at the end of the algorithm)

```
1;
% Implement the GGT algorithm of Euclid in its
% recursive form  just as in the following
% imperative recursive implementation using
% functional programming in MATLAB

function ret = rggt(a,b)
   disp(sprintf("%d -- %d", a,b))
   if(b == 0)
     ret = a;
   else
     ret = rggt(b, mod(a,b));
   endif
endfunction

%
% Therefore, use the following expressions from
% http://blogs.mathworks.com/loren/2013/02/07/introduction-to-functional-
    programming-with-anonymous-functions-part-3/
%


iif     = @(varargin) varargin{2*find([varargin{1:2:end}], 1, 'first')}();
recur   = @(f, varargin) f(f, varargin{:});
curly   = @(x, varargin) x{varargin{:}};
loop = @(x0, cont, fcn) ...                                      % Header
       recur(@(f, x) iif(~cont(x{:}), x, ...                     % Continue
           ?
                          true,        @() f(f, fcn(x{:}))), ... %
                              Iterate
             x0);                                                % from x0.

% Add your code here
% The code below assumes a function
% fggt taking two arguments a and b and returning one value, just as rggt
    does for the recursive implementation.


a = int8(rand()*100);
b = int8(rand()*100);

disp(sprintf("Theg gcd of %d and %d is %d. True gcd is %d.",a,b,fggt(a,b),
    rggt(a,b)))
```

https://github.com/mwernerds/big_geospatial_data_lecture/blob/master/03_matlab_functional_-euklid/euklid_assignment.m

Solution:

The following source code is an example of how to solve this problem in Octave:

```
1;
% Implement the GGT algorithm of Euclid in its
% recursive form just as in the following
% imperative recursive implementation using
% functional programming in MATLAB
function ret = rggt(a,b)
disp(sprintf("%d -- %d", a,b))
if(b == 0)
ret = a;
else
ret = rggt(b, mod(a,b));
endif
endfunction
%
% Therefore, use the following expressions from
% http://blogs.mathworks.com/loren/2013/02/07/introduction-to-
   functionalprogramming-with-anonymous-functions-part-3/
%


iif = @(varargin) varargin{2*find([varargin{1:2:end}], 1, 'first')}();
recur = @(f, varargin) f(f, varargin{:});
curly = @(x, varargin) x{varargin{:}};
```

```
loop = @(x0, cont, fcn) ... % Header
recur(@(f, x) iif(~cont(x{:}), x, ... % Continue?
true, @() f(f, fcn(x{:}))), ... % Iterate
x0); % from x0.
% Add your code here
% Define the algorithm with state {a,b}
ggtalgo = @(a,b) loop({a,b}, ... % Initialize state, k, to 0
@(k,b) k != 0, ... % While k < n
@(k,b) {b,mod(k,b)}); % k = k + 1 (returned as cell
array)
% and the interface, which selects b for the final value
fggt = @(a,b) curly(ggtalgo(a,b),2);
a = int8(rand()*100);
b = int8(rand()*100);
disp(sprintf("Theg gcd of %d and %d is %d. True gcd is %d.",a,b,fggt(a,b),rggt
(a,b)))
```

An alternative solution uses the recur function and looks like

```
ggt = @(a,b) recur(...
@(f,k,l) iif(l==0, k,...
true, @() f(f,l, mod(k,l))), a,b);
```

For Matlab, there is also a possible solution:

```
% Implement the GGT algorithm of Euclid in its
% recursive form just as in the
% imperative recursive implementation using
% functional programming in MATLAB
%
% Therefore, use the following expressions from
% http://blogs.mathworks.com/loren/2013/02/07/introduction-to-functional-
    programming-with-anonymous-functions-part-3/
%
iif     = @(varargin) varargin{2*find([varargin{1:2:end}], 1, 'first')}();
recur   = @(f, varargin) f(f, varargin{:});
curly   = @(x, varargin) x{varargin{:}};
loop = @(x0, cont, fcn) ...                                     % Header
       recur(@(f, x) iif(~cont(x{:}), x, ...                    % Continue?
                         true,        @() f(f, fcn(x{:}))), ... %   Iterate
             x0);                                               % from x0.


% Add your code here
% The code below assumes a function
% fggt taking two arguments a and b and returning one value, just as rggt does
     for the recursive implementation.

%Solution 1:
%compute greatest common divisor (ggt) with recur
fggt1 = @(a,b) recur(...
                @(f,k,l) iif(l==0, k,... %end recursions if l==0
                             true, @() f(f,l, mod(k,l))),... %recursively call
                                  fggt1 with l, mod(k,l)
                a,b);

%Solution 2:
%compute greatest common divisor (ggt) with loop
ggtalgo =   @(a,b) loop({a,b}, ...          % Initialize state with a and b
                  @(k,b) k ~= 0, ...    % While k is not 0
                  @(k,b) {b,mod(k,b)});     %   Compute ggt for b, mod(k,b)


% and the interface, which selects b for the final value
fggt2 = @(a,b) curly(ggtalgo(a,b),2);

a = int8(rand()*100);
b = int8(rand()*100);

disp(sprintf('Theg gcd of %d and %d is %d (recur)/ %d (loop). True gcd is %d.'
    ,a,b,fggt1(a,b),fggt2(a,b),rggt(a,b)))
```

In Matlab, the imperative function rggt must be stored in a separate file rggt.m:

```
%imperative recursive implementation of Euklid's algorithm
function [ret] = rggt(a,b)
   disp(sprintf('%d -- %d', a,b))
   if(b == 0)
     ret = a;
   else
     ret = rggt(b, mod(a,b));
   end
end
```

Aufgabe 2: Introduction to Using Large Images (optional)

In this assignment, we want to have a look at Hanover's nights. In preparation, read about the NASA mission for creating night light maps at
https://www.nasa.gov/feature/goddard/2017/new-night-lights-maps-open-up-possible-real-time-applications
and download the following tiles:
https://www.nasa.gov/specials/blackmarble/2012/tiles/georeferrenced/BlackMarble_2012_C1_geo.tif
and
https://www.nasa.gov/specials/blackmarble/2016/tiles/georeferrenced/BlackMarble_2016_C1_geo.tif.

- Read and follow the C++-Example given on Github *https://github.com/mwernerds/big_geospatial_data_lecture* for reading and extracting PNG snippets from these images.
- Install R, the package rgdal and raster and visualize the Hanover area at night in both images.
- Show the difference of Hanover in 2012 and 2016 by showing, where it was significantly ligher or darker.
- Find some spot throughout the whole tile (or the world, if you want to use even larger images), where there is a significant change and try to identify, why it changed.
- If you like to help others, use your favourite programming language for the visualization and push it to our Github repository using fork and pull. The README on Github will inform you about the intended directory structure, so you know, where to put it inside the repository.

Solution:

The following solution for R shows, how powerful R and functional programming can be: the file is not actually read until we select Hanover and then, it only reads the Hanover area from the file. Thus, the program is faster than a typical sequence of reading the file, processing the file and outputting the results. It is also available via GitHub *https://github.com/mwernerds/big_geospatial_data_lecture/blob/master/01_-night_light_images/solution.R*

```
library(raster)

# Load the image and crop it to hanover
r = raster("images/BlackMarble_2012_C1_geo.tif")
r.hanover = crop(r,extent(c(9,10,52,53)))

r2 = raster("images/BlackMarble_2016_C1_geo.tif")
r2.hanover = crop(r2,extent(c(9,10,52,53)))

pdf("plots.pdf") # remove this in an interactive session

plot(r.hanover, main="Hanover at night in 2012")
points(9.71730046412,52.375993496,pch=23,bg="red",col="red",cex=2)

plot(r2.hanover, main="Hanover at night in 2016")
points(9.71730046412,52.375993496,pch=23,bg="red",col="red",cex=2)

plot(r.hanover - r2.hanover, main="Difference of Hanover at night")
points(9.71730046412,52.375993496,pch=23,bg="red",col="red",cex=2)

dev.off();
```

Aufgabe 3: OpenCV Face Detection (optional)

Install python and OpenCV in order to be able to run the OpenCV face detection example from the first lecture on your system. If you run into problems with specific platforms including Windows or Mac, you can either install a virtual machine (for example using VirtualBox) running Debian (the reference OS for this lecture) or try installing everything into a Docker container.

Solution:

You can find a project showing face detection with python on GitHub at *https://github.com/shantnu/FaceDetect/*. You need OpenCV and python on your computer. The easiest way is to use a Debian or Ubuntu-based Linux distirbution, which should pull in all dependecies using `sudo apt-get install python-opencv`. For convenience, the source of the video taken from the referenced repository is

```
# This script will detect faces via your webcam.
# Tested with OpenCV3

import cv2

cap = cv2.VideoCapture(0)

# Create the haar cascade
faceCascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

while(True):
   # Capture frame-by-frame
   ret, frame = cap.read()

   # Our operations on the frame come here
   gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

   # Detect faces in the image
   faces = faceCascade.detectMultiScale(
      gray,
      scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30)
        #flags = cv2.CV_HAAR_SCALE_IMAGE
    )

    print("Found {0} faces!".format(len(faces)))

    # Draw a rectangle around the faces
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)


    # Display the resulting frame
    cv2.imshow('frame', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```