# Project: book shelves

**STMO**

2020-2021

Project by missing

# Outline

In this project, we will arange my collection of books in my fancy bookshelf. My bookshelf consists of 31 shelves, each with a position (`pos`), a `width` and a `height` (everything is in meters). These shelves are given as a list of `NamedTuple`s.

```
NamedTuple{(:pos, :width, :height),Tuple{Tuple{Float64,Float64},Float64,Float64}}[(p =
```

```
shelves
```

For example:

```
first_shelve =   (
                 p =   (0.0,  0.0)
                 w = 0.7690727178631521
                 h = 0.4999432280876094
               )
```

```
first_shelve = first(shelves)
```

```
(0.0,  0.0)
```

```
first_shelve.pos  # position
```

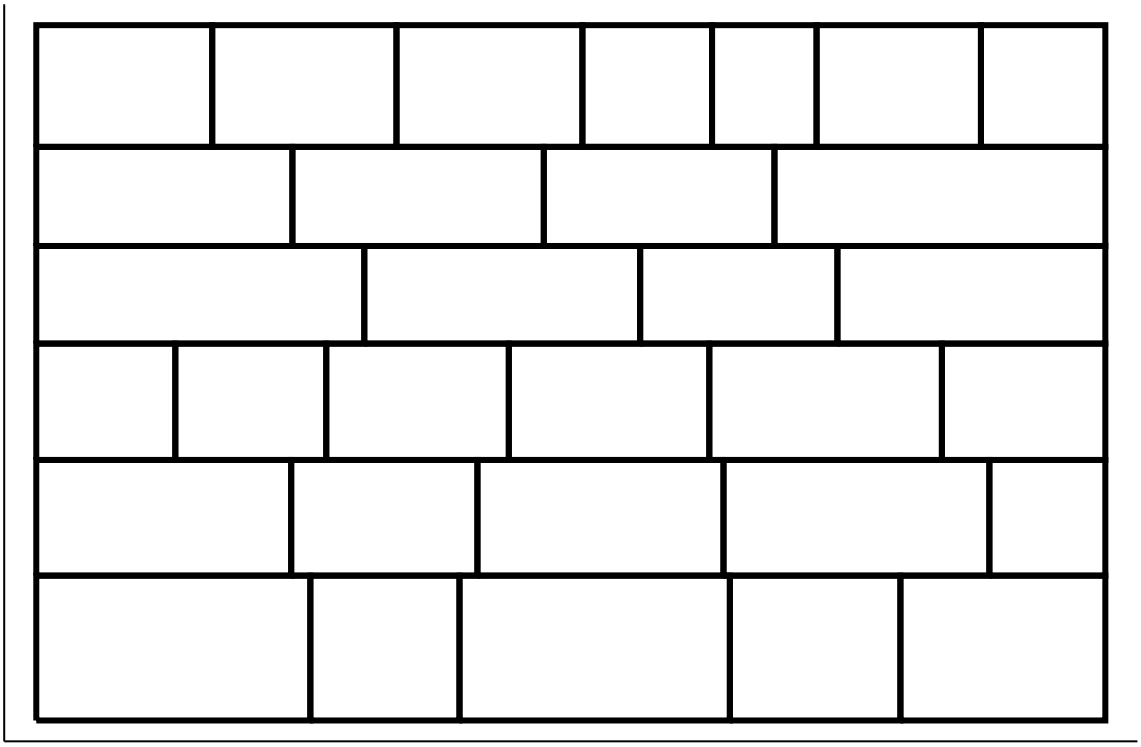
```
0.7690727178631521
```

```
first_shelve.width  # width
```

```
0.4999432280876094
```

```
first_shelve.height  # height
```

We can plot the bookshelf.



```
plot_shelf()
```

Looks a bit empty. Luckily we have 300 books to fill them with! Each book has a width, height and a color.

```
NamedTuple{(:width, :height, :color),Tuple{Float64,Float64,ColorTypes.RGB{Float64}}}[(w
```

```
books
```

Similarly, take a look at a randomly chosen book:

a_book = (w = 0.105, h = 0.26, c = )

```
a_book = rand(books)
```

```
a_book.color
```

```
0.105
```

```
a_book.width
```

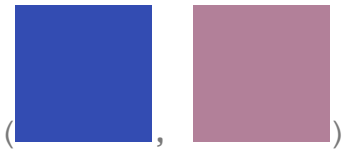
```
0.26
```

```
a_book.height
```

The goal of this project is to find an optimal placement of the books in the different shelves. These placements have natural constraints:

- the height of a book can not exceed the height of the shelf it is placed on;
- the sum of all the widths of the books on a shelf cannot exceed that shelf's width.

In addition, we want to place the books in an aesthetically pleasing way: books placed to each other have to be similar in color scheme. To this end, we can use the sums of the difference scores computed by the function `colordiff`. For example:



```
c1, c2 = RGB(0.2, 0.3, 0.7), RGB(0.7, 0.5, 0.6)
```

```
33.41617261263215
```

```
colordiff(c1, c2)
```

**some_book_colors =**

```
some_book_colors = [book[:color] for book in books[1:5]]
```

```
sum_of_colordiffs (generic function with 1 method)
```

```
function sum_of_colordiffs(colors)
    total = 0.0
    n_colors = length(colors)
    # fewer than two books, return 0.0
    n_colors < 2 && return total
    for i in 2:n_colors
        c1, c2 = colors[i-1], colors[i]
        total += colordiff(c1, c2)
    end
    return total
end
```

```
304.1518315225648
```

```
sum_of_colordiffs(some_book_colors)
```

Let's compare with a different order of the colors.

**book_colors_reordered =**

```
book_colors_reordered = some_book_colors[[5, 3, 1, 4, 2]]
```

```
138.38044690020402
```

```
sum_of_colordiffs(book_colors_reordered)
```

So your goal is find a **valid** placement of the books that minimizes the sum of all color differences of adjacent books. You also have minize the unoccupied amount of space. **Per square meter shelf area that is free, you pay a penalty of 1000**. So leave as few free space as possible.

Let us generate a naive solution, just placing the books in order that they appear in the list.

```
mysolution =
  (18 ⇒ Int64[145, 147, 150, 152, 153, 154, 155, 159, 164], 30 ⇒ Int64[229, 230,
```

```
mysolution = generate_naive_solution()
```

The structure of the solution is a dictionary (e.g., `Dict(1 => [3, 4, 5]...)`) containing the shelves' ids as keys and the a list of book ids as the values. In this solution, we used 243 of the 300 books.

We can use the function `isvalidsolution` to check if this solution is valid.

```
true
```

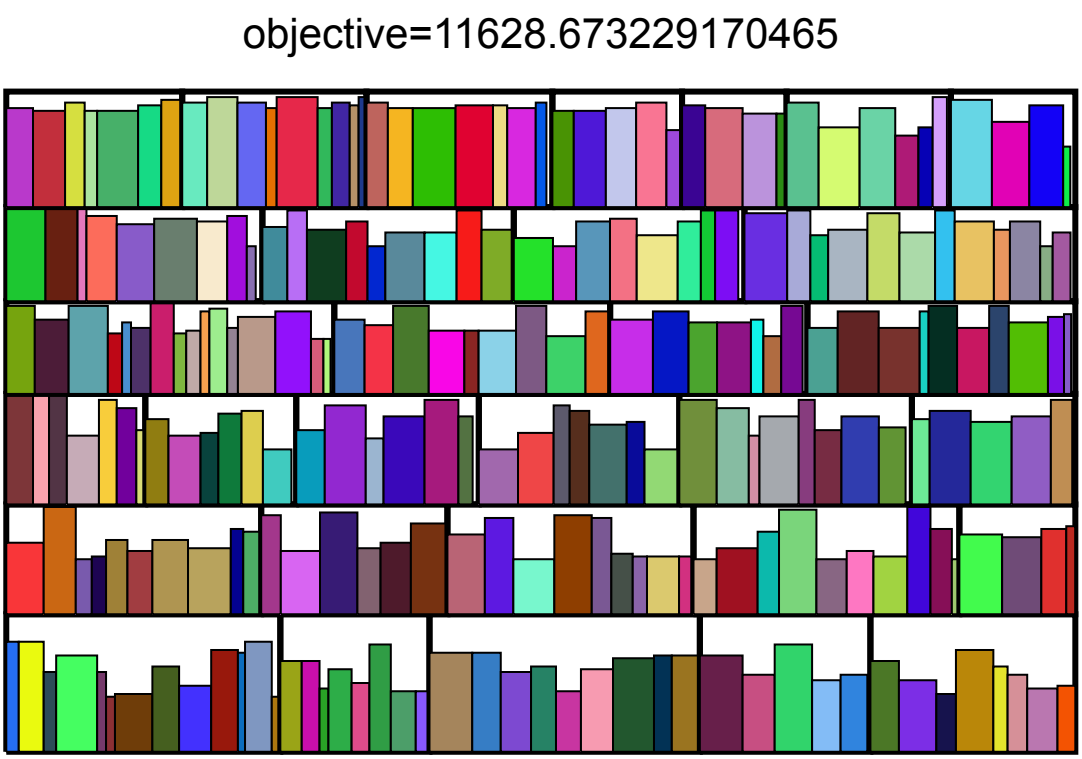
```
isvalidsolution(mysolution)
```

We can easily compute the objective:

```
11628.673229170465
```

```
compute_objective(mysolution)
```

Of course, we can visualise the solution.




```
show_solution(mysolution)
```

# Assignments

Use your knowledge of optimization to generate the best solution that you can. Send your notebook to **me**, both as Pluto notebook file (.jl) and as **PDF or HTML file**. Also send your solution as a JSON file. Use the function `save_solution` to generate this file. Use your name(s) in the file name!

You will be graded on two accounts:

- the quality of your solution relative to the other students;
- the originality and quality of your approach you used, including writing tidy code, adding documentation and comments.

You can do this project alone or in pairs. The deadline is **Friday 4 December**.

`student_names = missing`

```
student_names = missing
```

```
Enter cell code...
```

```
Enter cell code...
```

```
Enter cell code...
```

```
Enter cell code...
```

```
Enter cell code...
```

```
Enter cell code...
```

```
#savesolution("mycleversoluton.json", mysolution)
```

```
#loadsolution("mycleversoluton.json")
```

# Utilities

Functions that are useful. Nothing for you to change here.

`1000`

```
const gappenalty = 1_000
```

```
using Colors, Plots, RecipesBase, Statistics  # install these if you don't have them
```

`Main.workspace3.isvalidsolution`

```
"""Check if a solution is valid."""
function isvalidsolution(solution)
    books_used = Set(1:length(books))
    for s_id in keys(solution)
        shelf = shelves[s_id]
        books_ids = solution[s_id]
```

```
        # check if book ids are valid
        @assert issubset(books_ids, books_used) "Error in book ids"
        setdiff!(books_used, books_ids)
        # check if books are not used on other self
        books_on_shelf = books[books_ids]
        # check heights
        s_h = shelf.height
        @assert all((b.height for b in books_on_shelf) .≤ s_h) "Books exceed heigth
at shelf $s_id!"
        # check widths
        s_w = shelf.width
        @assert sum((b.width for b in books_on_shelf)) ≤ s_w "Total book with
exceeds width of self $s_id!"
    end
    return true
end
```

Main.workspace103.compute_objective

```
"""Compute the objective of your solution (errors when not valid)"""
function compute_objective(solution; gappenalty=gappenalty)
    isvalidsolution(solution)
    obj = 0.0
    for s_id in keys(solution)
        shelf = shelves[s_id]
        books_ids = solution[s_id]
        books_on_shelf = books[books_ids]
        books_surface = sum((b.width * b.height for b in books_on_shelf))
        shelf_surface = shelf.width * shelf.height
        obj += gappenalty * (shelf_surface - books_surface)
        length(books_on_shelf) > 1 || continue
        for i in 1:length(books_on_shelf)-1
            obj += colordiff(books_on_shelf[i].color, books_on_shelf[i+1].color)
        end
    end
    return obj
end
```

Main.workspace29.plot_shelf

```
"""Plot only the selve, bit empty."""
function plot_shelf()
    pl = plot()
    for (pos, w, h) in shelves
        plot!(Square(pos, w, h), lw=3)
    end
    pl
end
```

Main.workspace103.show_solution

```
"""Plots the solution. You can change stuff, e.g., adding keyword
arguments such as `title=\"my solution\"`"""
function show_solution(solution; kwargs...)
    pl = plot(title="objective=$(compute_objective(solution))", axis=([], false);
kwargs...)
    for (s_id, (pos, w, h)) in enumerate(shelves)
        x, y = pos
        plot!(Square(pos, w, h), lw=3)
        for b_id in solution[s_id]
            book = books[b_id]
            w, h = book.width, book.height
            plot!(Square((x, y), w, h), book.color)
            x += w
        end
    end
    return pl
end
```

Main.workspace3.savesolution

```
"""Stores `solution` to the `fname`."""
function savesolution(fname, solution::Dict)
    open(fname, "w") do fh
        write(fh, JSON.json(solution))
    end
end
```

Main.workspace27.loadsolution

```
"""Loads a solution from a file `fname`, stored in JSON. Raises an error if
invalid."""
function loadsolution(fname)
    solution = JSON.read(fname, String) |> JSON.parse
    solution = Dict(parse(Int, k) => [v...] for (k, v) in solution)
    @assert isvalidsolution(solution) "Solution is not valid!"
    return solution
end
```

```
struct Square
    pos
    width
    heigth
end
```

```
@recipe function f(s::Square, color = nothing)
    linecolor   --> :black
    seriestype  :=  :shape
    label := ""
    fillalpha --> (color isa Nothing ? 0.0 : 1.0)
    fillcolor := color
    xticks := []
    yticks := []
    [s.pos[1], s.pos[1]+s.width, s.pos[1]+s.width, s.pos[1]], [s.pos[2], s.pos[2],
s.pos[2]+s.heigth, s.pos[2]+s.heigth]
end
```

Main.workspace3.generate_naive_solution

```
"""A function to generate a bad solution. You can do better."""
function generate_naive_solution()
    # keep track of all books used
    books_used = Set{Int}()
    solution = Dict{Int,Vector{Int}}()
    # for all shelves, go over the books,
    # if there is room, add it to the selve
    for (sh_id, shelf) in enumerate(shelves)
        solution[sh_id] = []
        h_shelf = shelf.height
        w_shelf = shelf.width
        for (book_id, book) in enumerate(books)
            # check if book is not yet used
            book_id in books_used && continue
            w_book, h_book = book.width, book.height
            # check height of book
            h_book > h_shelf && continue
            # if still room, add book
            if w_book ≤ w_shelf
                push!(solution[sh_id], book_id)
                push!(books_used, book_id)
                w_shelf -= w_book
            end
        end
    end
    return solution
end
```

# Parsing the data

Generating the shelves and books. Danger zone, keep out!

```
using JSON
```

```
const shelves = JSON.Parser.parse(shelves_json) .|> parse_shelve;
```

```
const books = JSON.Parser.parse(books_json) .|> parse_book;
```

```
parse_book (generic function with 1 method)
```

```
parse_shelve (generic function with 1 method)
```