Numerical Methods (NUM101) — Optional Coursework ShtPile

This coursework consists of one part. It is worth 17 marks which will replace your worst coursework mark of the three regular courseworks (overall 17% of the credits for this unit).

Deadline	hand-in or upload?
18 May (Wed)	22:30 upload to Victory Assignment `shtpile` **no** hardcopy to CAM
18 May (Wed) or before	demonstration of working demo script and function in lab to lecturer

Instructions and rules

- Material to be uploaded to Victory: function files `mymove.m`, and `mystart.m` , and, if you need them, function files of other functions that you call inside your main functions.
- Marks will be awarded for your work only if you demonstrate the usage and the inner workings of your solution to the lecturer in the lab.
- Credit:

  100% code performs computation correctly and efficiently, is well structured and commented;

  ≥80% code performs computation correctly and efficiently

  ≥60% code performs computation correctly but has problems[1];

  ≥40% code does not perform computations correctly but could be made to work with minor corrections;

  ≥20% the intentions behind the code are discernible with some effort.

- This is **individual** coursework. **No declared collaboration is permitted**.
- For questions, clarifications and further help contact:

Jan Sieber (`jan.sieber@port.ac.uk`, office LG.146).

[1] for example, the function works correctly most of the time but fails for some valid arguments. Other examples of problematic constructions (look also for warnings in the Matlab editor):

- hard-coded 'magic' numbers spread throughout the code,
- functions that should be general but only work for this example,
- one part of the code is a repetition of another part,
- stray brackets, misleading variable names or variable usages (say, using `x(:)` if `x` is scalar),
- arrays grow inside a loop,
- a variable is defined but not used.

Question 1: A kid's game: beat Heinz

A simple two-player game (not sure about its English name, in German it's called S***häufchen) goes like this: one erects a random number of piles with a random number of stones in each pile. The players take turns to remove stones. In each turn the moving player selects one pile and removes as many stones from this pile as he/she likes (but at least one). The player who is forced to take the last stone is the loser. Initially, between humans one player can choose how the stones are arranged into piles and the other player is allowed to choose who makes the first move.

Download `ShtPile.m` and `Heinz.p` into your current directory. The function `ShtPile()` implements the game (either by taking inputs from the command line if you play interactively, or by letting two computer players play against each other and printing the run of play). The initial pile is chosen randomly and you can choose who starts. The file `Heinz.p` contains a function for the computer player. You can play the game interactively against Heinz by simply calling `Shtpile();` (see Hints for an example output from the interface).

You have to program your own computer player, which has to beat a Heinz every time. The computer player is implemented as a function

```
newpile=mymove(oldpile)
```

For example, if oldpile is `[2,0,3]` and you want to remove a single stone from pile 3 then the function should return `[2,0,2]` as `newpile`. This should, of course, be a fully automatic function requiring no user input.

The other part of your computer play is a function that chooses if you want to start:

```
s=mystart(pile)
```

which should return `s=true` (true means your computer player `mymove` will make the first move) or `s=false`. You can make your choice depending on the initial (random) pile.

When you have finished your computer player you can call `ShtPile` like this:

```
ShtPile('makemove',@mymove,'start',@mystart);
```

if your functions are in `mymove.m` and `mystart.m`.

Total for Question 1: 17 marks

Hints and further instructions

- **Marking scheme** Marks get awarded only if both functions work following the rules of the game and beat Heinz. Deductions from the maximal mark are then taken for flaws in the code according to the front sheet.
- The game has a simple safe winning strategy (whoever can choose who starts is the winner). You can find the strategy on the web if you don't know it already. Your function `mymove` has to implement this winning strategy.
- Your function `mymove` must not call `Heinz` or input from the user.
- Look at the function `ShtPile` for other useful options during experimentation (for example, you can set the initial pile).

Question 1 continues overleaf

- Example of output from interactive play:

```
>> ShtPile();
Current piles (1st row: number of the pile, 2nd row: number of stones)
     1     2     3
     6     8     7
Do you want to start? (0=no, 1=yes) 1
You start
Current piles (1st row: number of the pile, 2nd row: number of stones)
     1     2     3
     6     8     7
Which pile? 1
How many stones to be removed? 6
Your  move: 6 stone(s) from pile 1.
Current piles (1st row: number of the pile, 2nd row: number of stones)
     1     2     3
     0     8     7
You will lose!
Heinz' move: 1 stone(s) from pile 2.
Current piles (1st row: number of the pile, 2nd row: number of stones)
     1     2     3
     0     7     7
Which pile? 2
How many stones to be removed? 7
Your  move: 7 stone(s) from pile 2.
Current piles (1st row: number of the pile, 2nd row: number of stones)
     1     2     3
     0     0     7
You will lose!
Heinz' move: 6 stone(s) from pile 3.
Current piles (1st row: number of the pile, 2nd row: number of stones)
     1     2     3
     0     0     1
Which pile? 3
How many stones to be removed? 1
Your  move: 1 stone(s) from pile 3.
You lost!
>>
```