# Exercise Creation Guideline

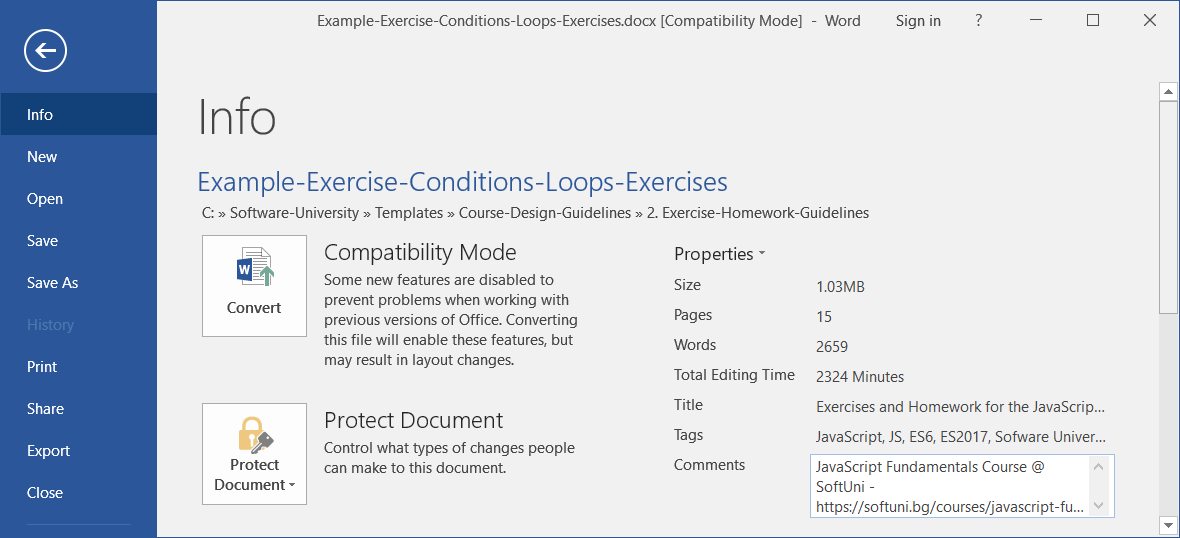
This document describes the **guidelines** for creating course **exercises**, in-class exercises (labs) and **homework** assignments for the internal training program at the Software University (SoftUni).

## Objectives

* **Short and clear descriptions**
  + **Describe** the assignments well, but in **short form**, without unneeded details or stories.
  + Use **examples** instead of long text descriptions.
  + Use **pictures** / **diagrams** / **screenshots** to illustrate the problem visually.
  + Provide well prepared **sample input and output**.
* **Learn by doing**
  + Let the students **do live exercises** in class or at home, to boost writing code every day.
  + Make sure the first 2-3 problems are **easy enough for everyone** (even the weakest students).
  + Provide a way for **automated evaluation** for the student’s code (e.g. in a judge system).
* **Increasing difficulty**
  + Start with an **easy problem** with a lot of **hints** with partial or full **solution** source code.
  + Slowly **increase the difficulty** from problem to problem.
  + Provide more detailed **hints** for the **first** few exercises and **less** for the **last** few exercises.
  + At final step, give a **1-2 complex exercises**, but as **last in the list** and explicitly denoted with “**\*\*\***”.

## Document Structure

* Document **metadata** – ensure your document holds correct metadata (title, tags, comments, author):



* + **Title**: Exercises and Homework for the ***{course name}*** Course at SoftUni
  + **Tags**: ***{course tag 1}***, ***{course tag 2}***, ***{course tag 3}***, …, Software University, SoftUni, programming, coding, software development, education, training, course
  + **Comments**: ***{course name}*** Course @ SoftUni - <https://softuni.bg/courses/>... (course URL)
  + **Categories**: programming, education, software engineering, software development
  + **Subject**: ***{course name}***
  + **Author**:Software University

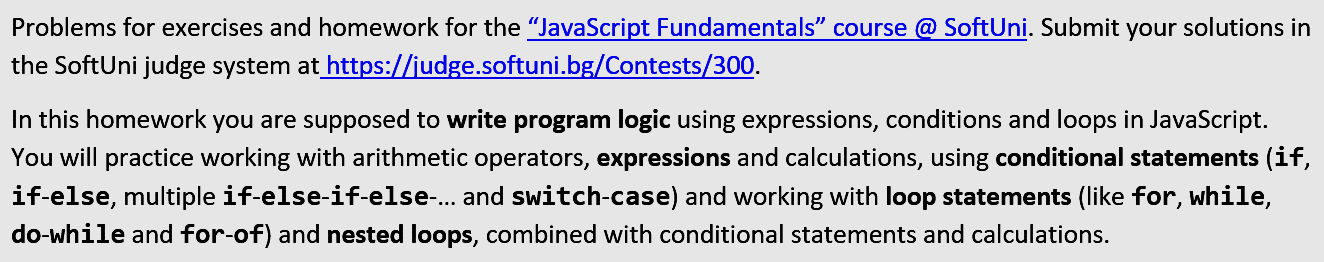
Maintain **consistent metadata** for the entire course (similar metadata for all course documents).

* Document **title** – should match the course topic, e.g.



* Document **intro section**
  + Explains the **purpose of the document**: exercises for certain course (+link) and certain topic + link to submit the solutions (in the judge).
  + Shortly explain the **learning objectives** – what learning content this course will cover.

Example:



* **Problems** (all problems follow the same structure)
  + Short **title** – explain in 1-3 words the problem (like the subject in an email)
  + Short **description** – keep it short, explain by examples and figures, not with long text
  + **Input** and **output** description – describe shortly the input and output format (if available)
  + **Constraints** – only when appropriate, as addition to the input / output description
  + **Examples** / **screenshots** – obligatory put examples: sample input, sample output, figure (when appropriate), screenshot (when appropriate)
  + **Hints**: first problems should have more hints, the last – less.
* Order the problems in **increasing complexity**: from the easiest to the hardest!
* Denote the most **complex problems** with asterisks “\*”.

## How to Describe Problems?

Explain the problems **shortly**. Avoid long descriptive texts. Use **figures**, **examples** and **screenshots** instead.

**Use English**, unless you have a good reason to make an exception. **English** is the primary language for the training resources at SoftUni.

### Problem Title

Use **short title** that answers the question “**what is this problem about?**”.

* Examples of **good names**: “Point in Rectangle”, “Age Calculator”, “Biggest of 3 Numbers”, “Voting System”.
* Examples of **bad names** (titles too long and descriptive):“Calculate the Biggest Number from Array of Integers”, “Read Numbers, Sort them and Print them in JSON Format”, “HTML Page Holding Students”.
* Examples of **bad names** (titles too short and say nothing about the problem): “Biggest”, “Number”, “Age”.

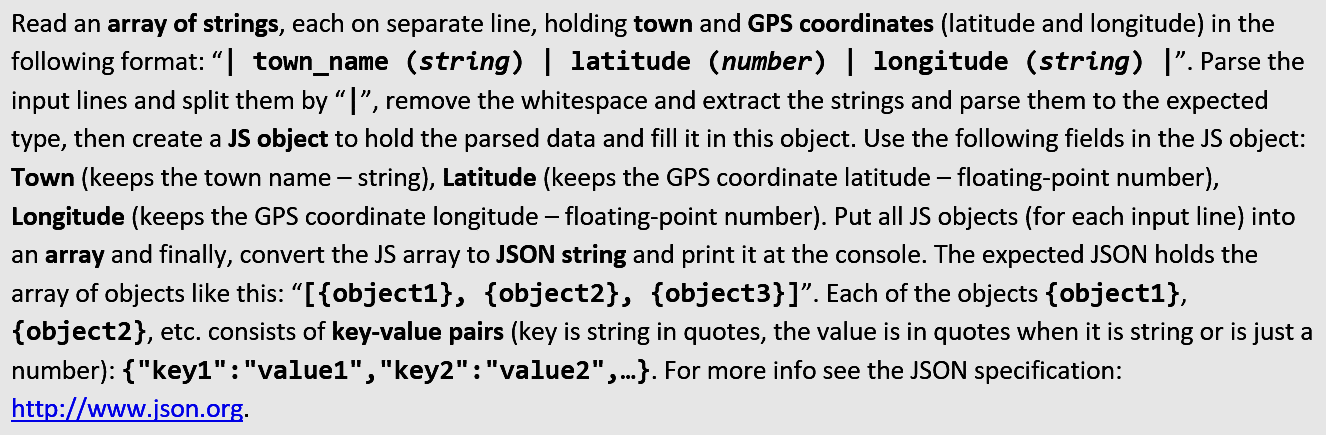
Optionally, use asterisks to denote hard problems:

* One asterisk “\*” means **a bit more complex** problem than the level in the current lesson.
* Two asterisks “\*\*” mean **a more complex** problem than the level in the current lesson and usually the student needs to **find additional information in Internet**.
* Three asterisks “\*\*\*” mean **a complex** problem that needs **a lot of time** + **finding information in Internet**.
* **More asterisks** mean even **more complex** or **time-consuming** problem.
* For most courses at SoftUni the **problems denoted with asterisks** are assumed to be **non-mandatory**, for champions only.
* **Order the problems** with asterisks by their complexity / time needed, in increasing order (hardest problem should be last).

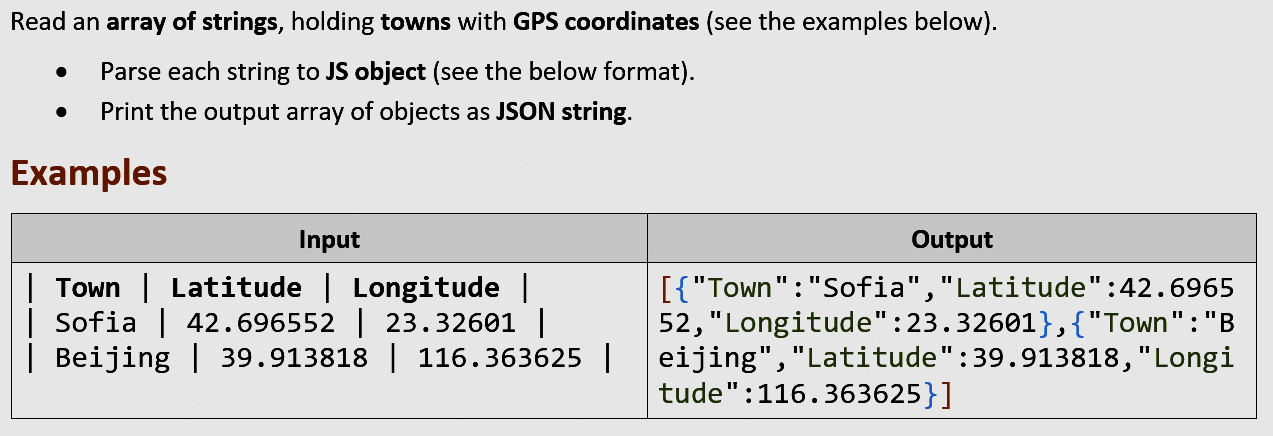
### Problem Description

Describe the problem **shortly**, without unneeded explanations. Use **examples** and **figures** instead of long text descriptions. Don’t describe the input and output format when it is visible from the examples (unless something non-obvious should be clarified).

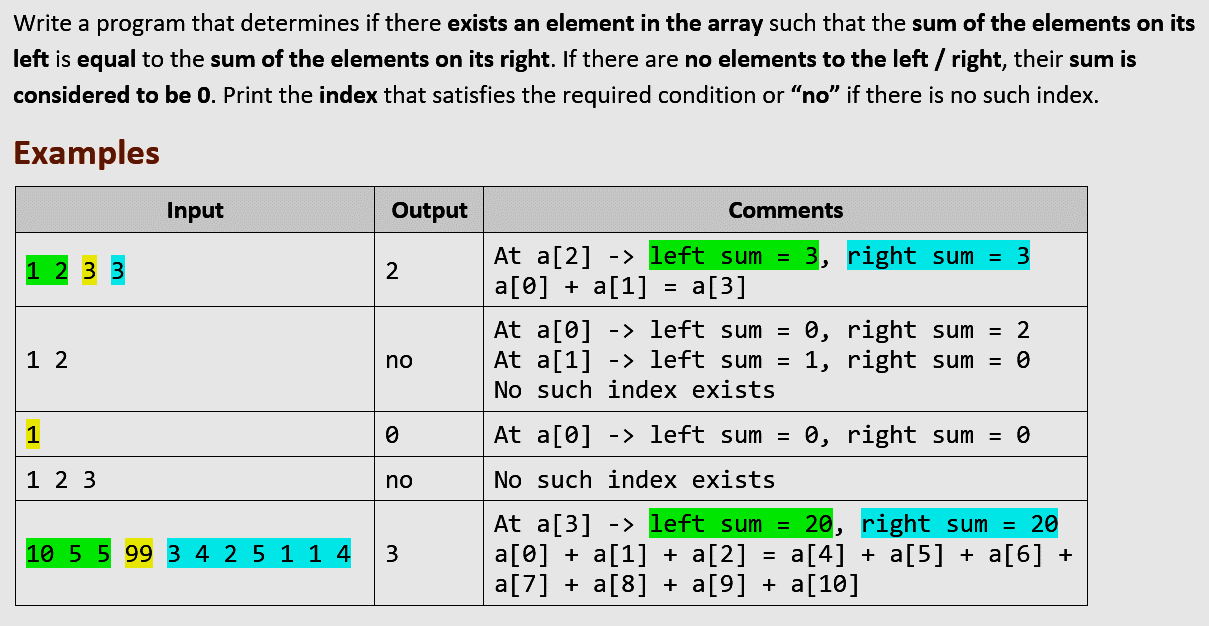
* **Bad example** of too descriptive text. This example is a **disaster** for the students, don’t do it!



* **Good example** of short and clear description of the same problem with a colored input / output:

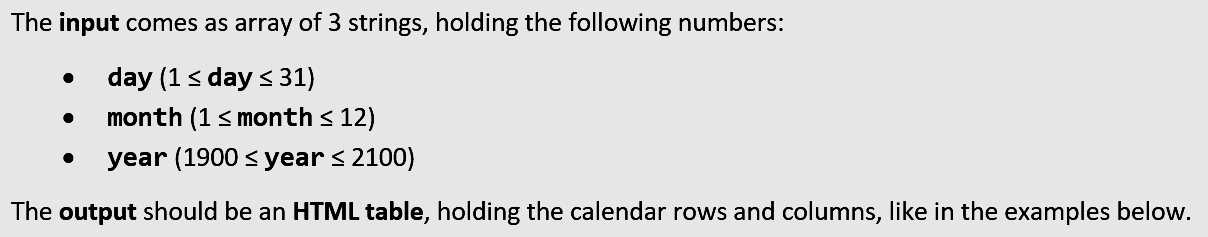


* Another **good example**:
  + Thе problem is best described through **examples**, not through text.
  + **Short description** of the problem, just a few sentences.
  + Good **examples**, covering the **normal** and **special cases**.
  + **Colors** to help understanding the examples.
  + **Comments** to help understand each of the examples.
  + The **input / output formats** are not explicitly described, because are visible from the examples.
  + **No constraints** – they are not critical for this problem. It is not clear if the input can hold non-integers or negative numbers or 0 numbers, but this is not important for the solution of this problem. If these were important, such example should be given.



### Format of Input / Output + Constraints

* In most cases, **you don’t need to explicitly describe the input + output format + constraints**.
  + So, this section is not obligatory for most problems.
* If you feel the students need an **explanation** of the **input** and **output**, add it, but **keep it short**.
* **Examples** will explain the input and output better than a text.
* Explain the **non-obvious information**.
  + For example, if you enter 4 numbers, describing rectangle location + size, it is critical to explain, that the input holds at each line **left**, **top**, **width** and **height**, not just 4 numbers.
* **Constraints** will be needed for some problems, but prefer to **give good examples** instead of descriptions.
  + For example, instead of explaining that the rectangle location and size could use **floating-point numbers**, just give examples holding non-integer values.
  + **Constraints** should explain non-obvious limitations, like “**0 ≤ hours < 12**” and “**0 ≤ minutes < 60**”.
  + If the constraints are not important or are **obvious**, skip them. For example, if the problems says “check a number and print whether it is ‘odd’, ‘even’ or ‘non-integer’”, no constraints should be described. Just **give good examples** (positive number, negative number, 0, non-integer number, non-number string, etc.).
* A **good example** of input / output / constraints section from the problem description:



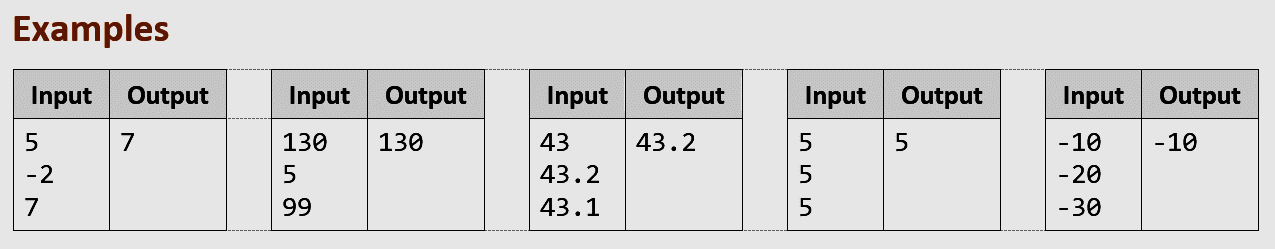
### Examples (Sample Input / Output)

**Examples** (sample input + output) are **very, very, very important**, even the most important part of each problem description!

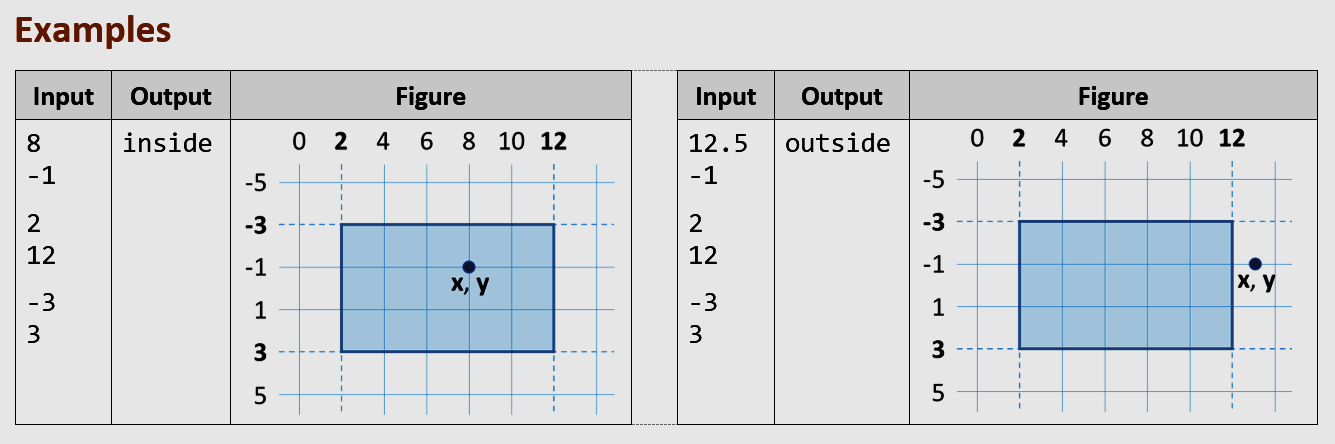
* Give **good examples**! Good examples cover the **normal case** (usual case) and all **special cases** (border cases).
  + A **big mistake** is to have a **special case, not covered in the examples**.
  + Examples should be **the main source of information** about the problem, its allowed input and its expected output.
* Give **several examples**, not just 1 or 2. Complex problems may give 5-10 examples.
* Instead of explaining what to print in a **special situation**, just **give an example** to make it obvious.
* Use **colors** or **figures** or **comments** to explain each test case (when needed).
* For **UI problems** (e.g. in the HTML course) provide **screenshots**, which make obvious the requirements.

Examples of **good sample input + output**:

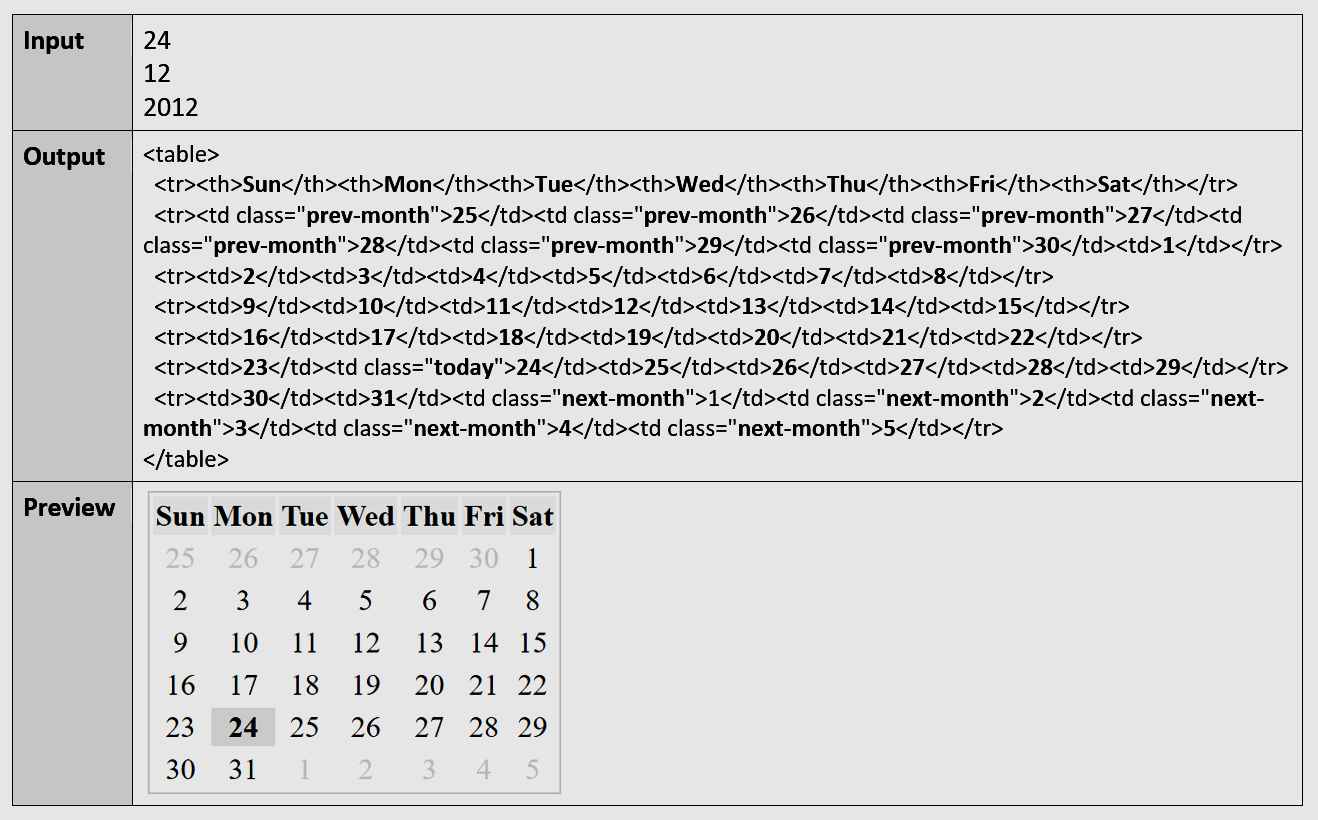
* Simple problem with **clear input and output** (biggest 3 numbers). Examples answer what to do when all numbers are **equal**, what to do with **negative numbers**, what to do with **non-integers**.
  + The below examples **implicitly say** that the expected input holds **always 3 lines** and **always numbers**.
  + If the problem allows **non-integers**, you should obligatory put such an example.
  + If the problem allows **less than 3 numbers**, you should obligatory have such an example.
  + If the problem allows **more than 3 numbers**, you should obligatory have such an example.



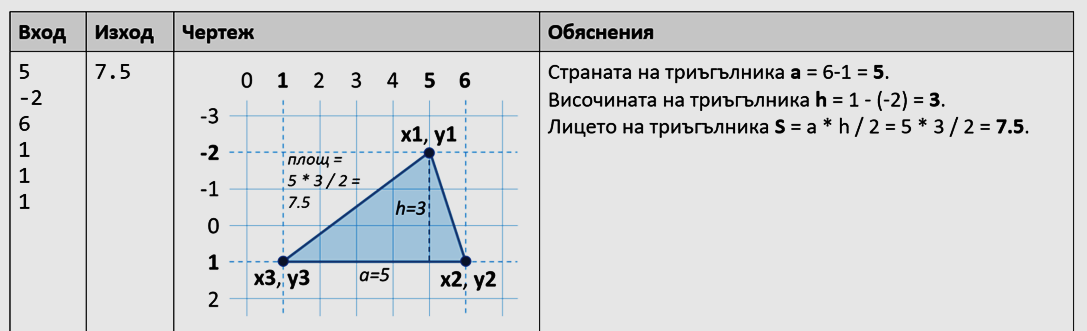
* Input which is meaningless unless **visualized** à just visualize it. Examples of such problems might be geometry (points and figures in the plane), transformations of elements that need visual explanation or UI-related problems that need a screenshot.



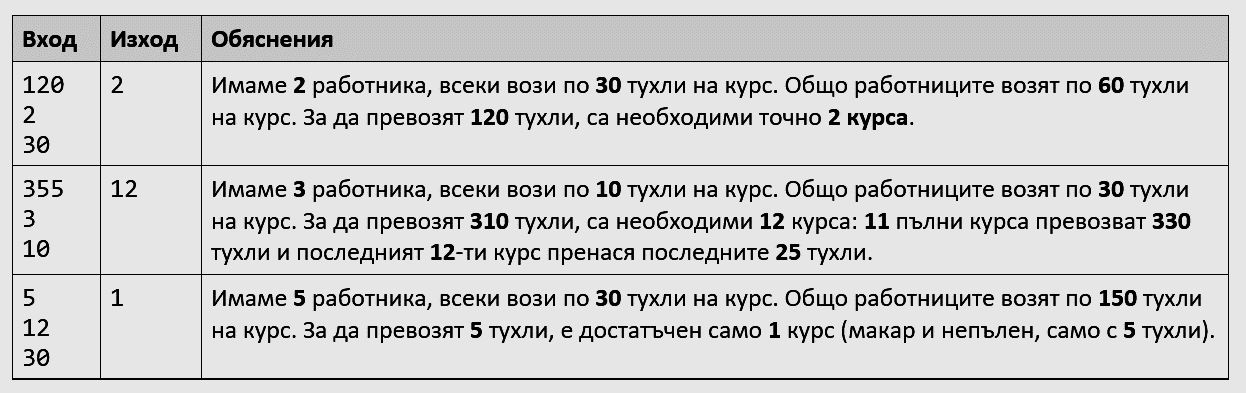
* Output, which is **hard to be perceived** unless visualized. The example below shows how important is to **visualize** certain input or output:



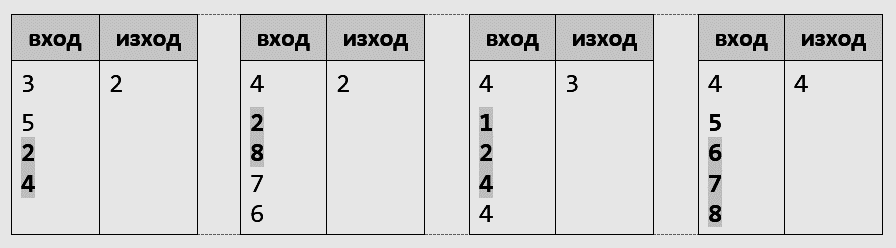
* Example of **visualization** + **explanations** for each example:



* Example of good **example explanation**:



* Example of using **colors** to **improve the examples**:



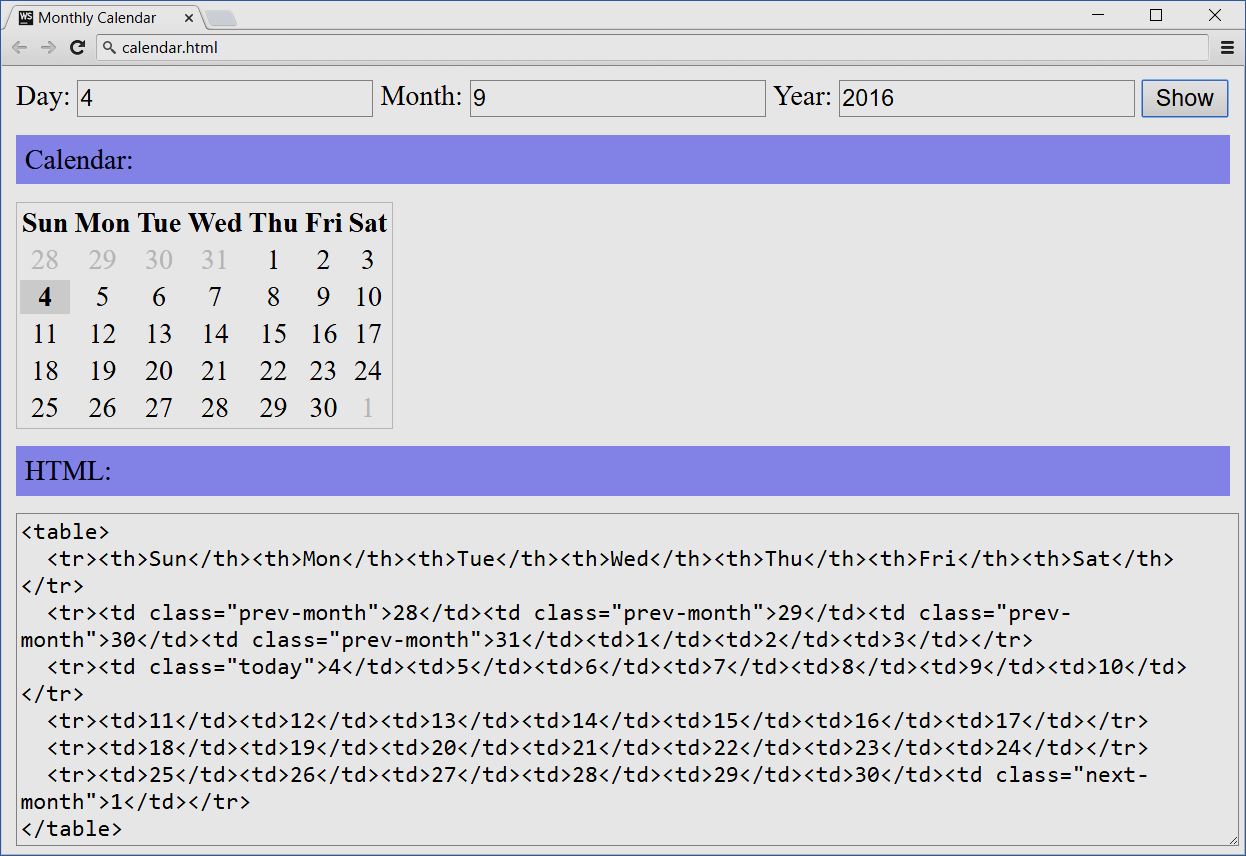
### Screenshots

For problems, where **UI** (user interface) is involved, the **screenshots** are **very, very, very important**, even the most important part of each problem description! Most UI problems need just a **few screenshots** and **very little text**.

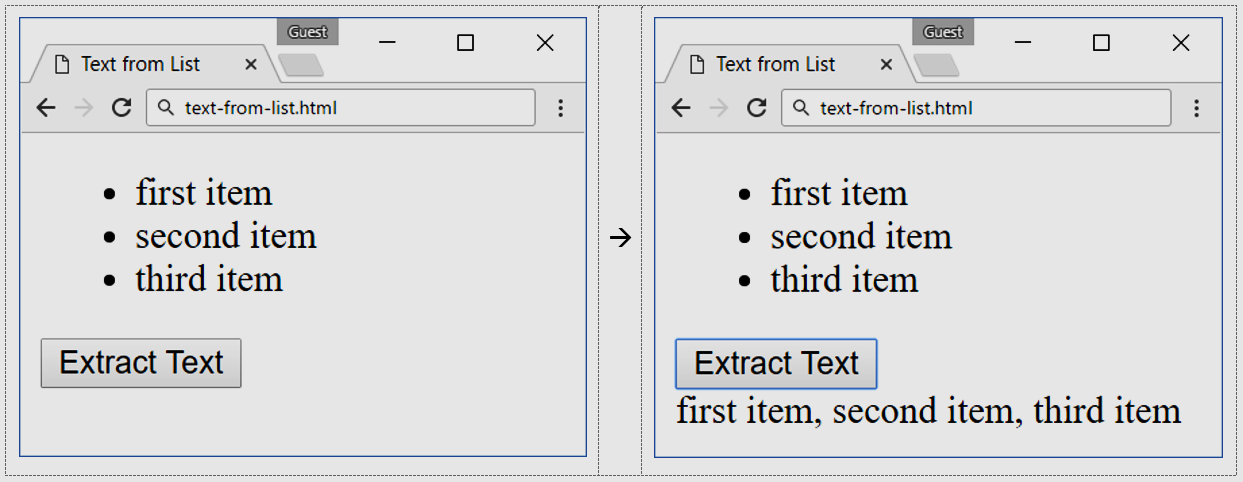
Use **good screenshots** to **explain visually the application functionality**. Good screenshots are **small images** using **large fronts**, showing the interesting part of the screen only (without unneeded areas) and showing what happens during the **user interaction** with the application (what happens when users click at each link / button / form).

Examples of **good screenshots**, explaining the problem without need of text:

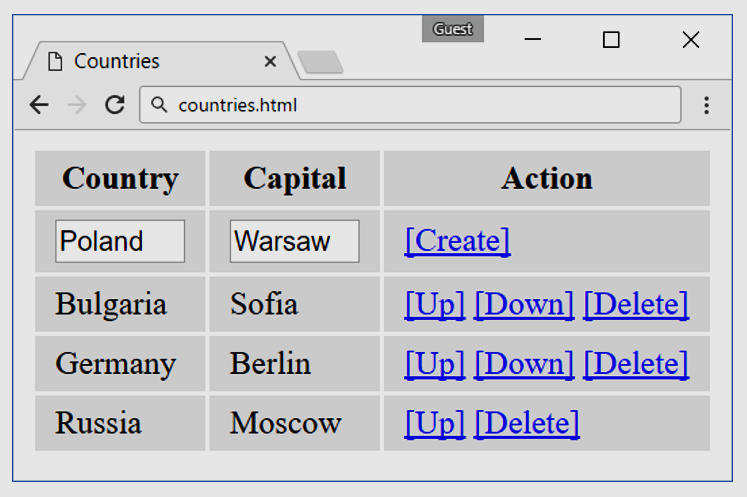
* The problem in this example is to **print an HTML calendar** by **day** + **month** + **year**. This screenshot explains everything in big details and no explanatory text is needed:



* **Screenshots** showing the **application in action** (what happens on button click in this case):



* This **screenshot** **explains the problem** at 80%. Even if you say “just create this app”, it is clear what is expected. Most UI problems need just a **few screenshots** and **very little text**.



### Hints

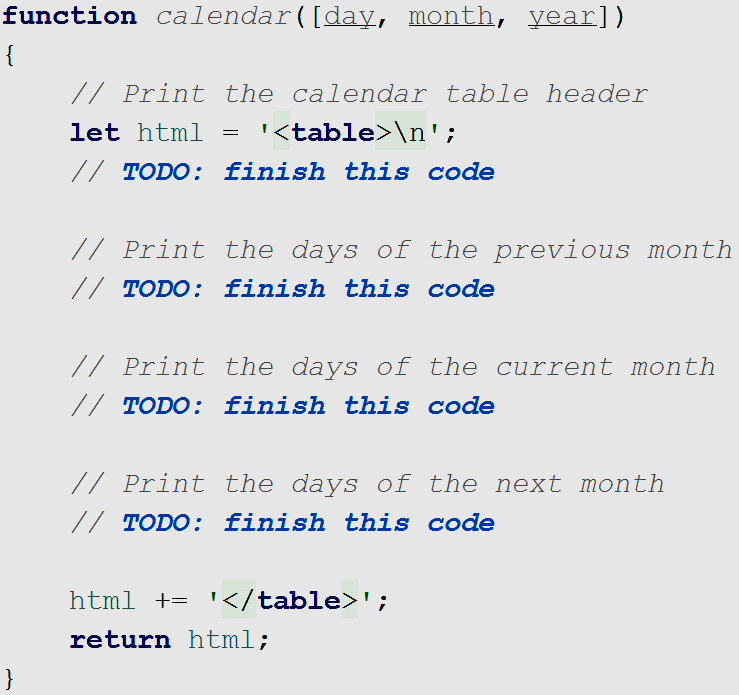
**Hints** for each problem aim to **help the students** solve it.

The **hints** should **gradually change from 90% to 10% of the solution** when going from the first to the last problem.

* Typically, **the very first problem should be solved at 90%**, so even a very weak student should gain a non-zero result with ease.
* The **next 2-3 problems** should have **detailed hints**, so that average students should be able to solve these problems with ease (using the hints).
* The **problems in the middle** should come with less hints, just with some small **directions**.
* The **last few problems** should have **little** or even **no hints**.
* Hints may hold **code snippets**, **explanations**, **links** to external resources, other **directions** (text, figures, …).

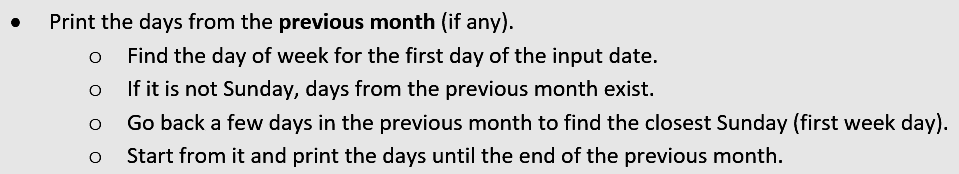
**How to create hints** for the problems?

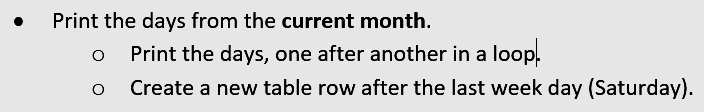
* Split the problem into **steps** and describe the goal of each step. Example:



* Now give **hints for some or all of the steps**. Examples:

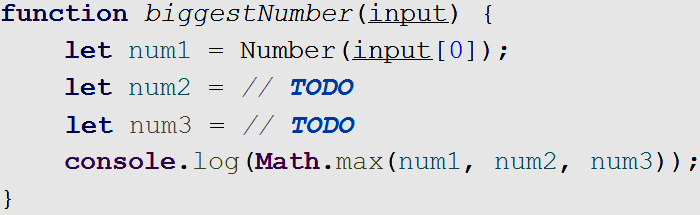








* Put the **code as image (screenshot)**, not as text to **avoid copy / paste**.
  + Take the screenshot snippet of **colored code** from your IDE.
* Give the **most important pieces** of the problem as code snippets:



* Don’t give 100% of the code. **Give 80%-90% of the code**, to force students to think and **avoid copy / paste**. Either make unreadable (using a **blur effect**) some pieces of the code (see below) or use **TODOs** (see above).

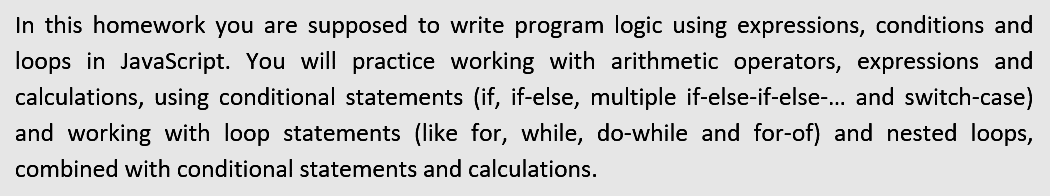
## Document Formatting

Use the **document template** from the sample exercise document. It holds the styles for all elements:

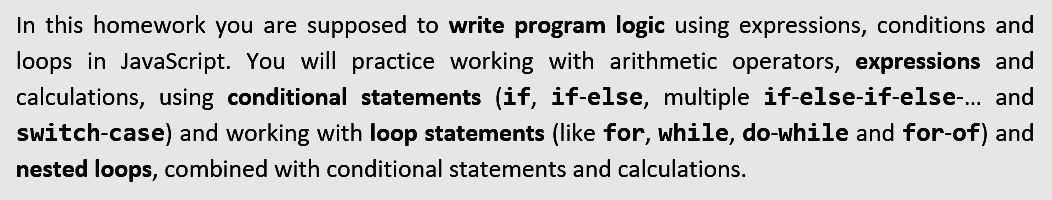
* **Heading 1** style – for the document title
* **Heading 2** style – for the problem titles
* **Heading 3** style – for the sections inside the problem (e.g. Input / Output, Examples, Hints)
* **Heading 4** style – for sub-sections (if needed)
* **Code** style – for source code elements (like variable names and file names).

### Text Formatting

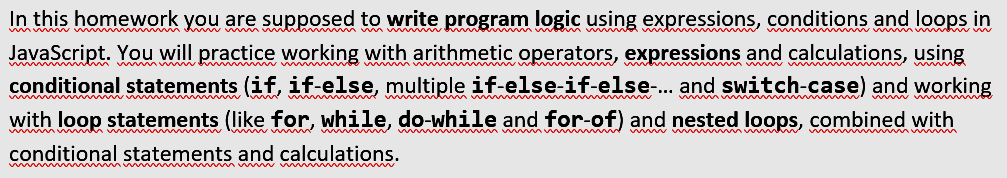
* **Bold the key words / phrases** to enhance text readability. Don’t bold too much, just the most important key words or phrases, at least one per paragraph and at most one per sentence.
* Use the style “Code” from the MS Word styles for pieces for code pieces, e.g. file names, variable names, function names, class names, language keywords, etc.
* **Bad example** (hard to read text with no bold):



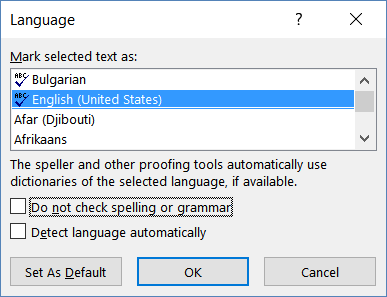
* **Good example** (better formatted text with **bold** keywords):



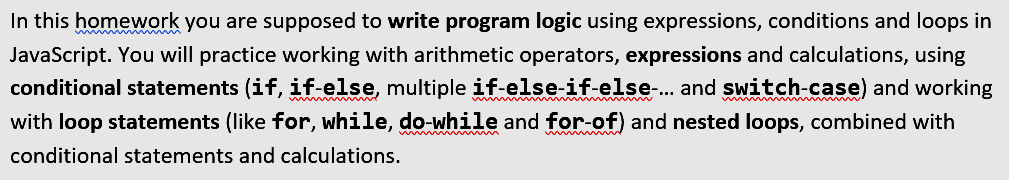
* Ensure the **text spelling and grammar correction** is used properly. **Bad example**:



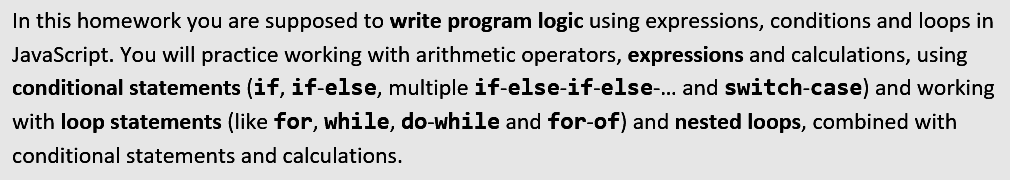
* The above can be fixed by choosing “**English (United States)**” as spelling language + enable spelling:



* After enabling the **spell checker**, it still **may not recognize** (and keep undelined) some pieces of the text, e.g. variable names, file names, etc. **Bad example**:



* You can **disable the spell checker** for these words / phrases checking the **[Do not check spelling or grammar]** in the above dialog box. **Good example**:



### Formatting Titles in the Document

Type the **titles** in the document, starting each word with **upper letter**. Exception: keep lowercase the conjunctionс, prepositions and articles (like ***a*** and ***the***). **Good example** of correctly formatted title:



**Bad examples** of incorrectly formatted titles:





### Code Formatting

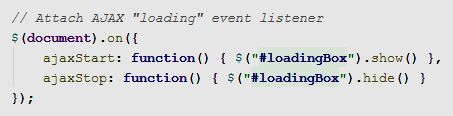
There are two types of source code in the exercise document:

* Code intended to be copy / pasted.
* Code intended to be written by the student (no copy / paste).

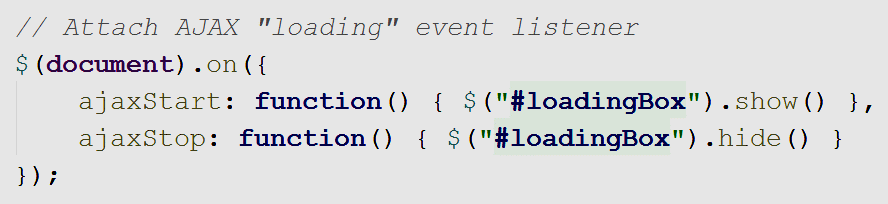
#### Formatting Not for Copy / Paste Code

Put the code snippets **not intended to be copy-paste-able** in the document as **sequence of images**.

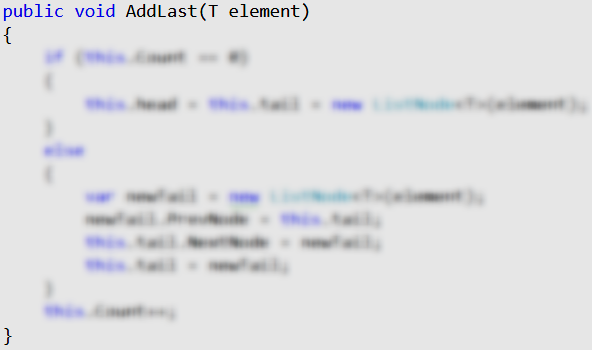
* Put the **code as image (screenshot)**, not as text to **avoid copy / paste**.
* Take the screenshot snippet of **colored code** from your IDE.
* Use **high resolution image** (large fonts in the IDE) and resize it down in the Word document. Thus, if the document is zoomed at 150%, the image will not lose its quality.
  + **Never use JPEG** images, because JPG loses the image quality, especially for screenshots!
  + Use the **PNG format** – it is lossless.
* **Bad example** of low-resolution image (small fonts, image resized to 140%):



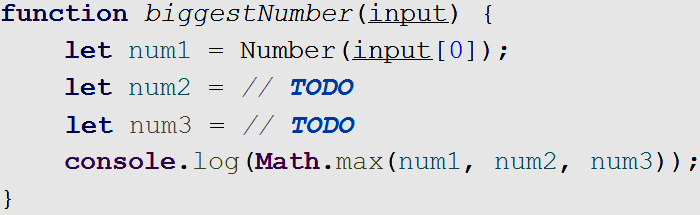
* **Good example** of high-resolution image (large fonts, image resized to 70%):



* Sometimes, you may use **blurred code** to enforce the students to think instead of mechanically copy / paste the code. This works well when you have **several very similar methods**: show the code of the first method + blurred code of the similar methods:



* Another similar technique is to use **TODOs** in the code:

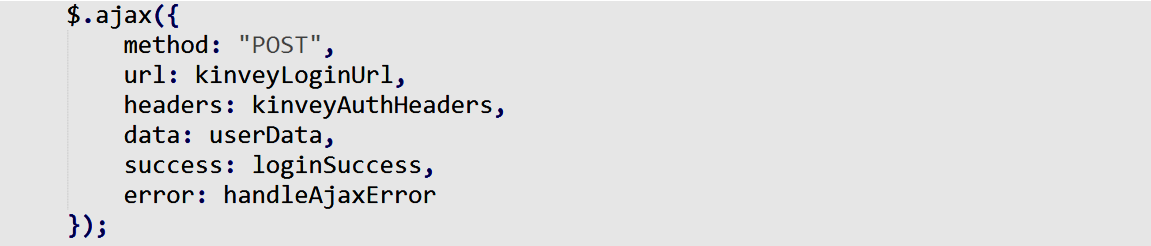


* For long code use a **sequence of images** instead of single image. This allows **page breaks** between the code sections in the document. Otherwise, some pages in the document will have too much empty space.
* **Bad example** (the image vertical size too big and may cause a large blank area on the page):



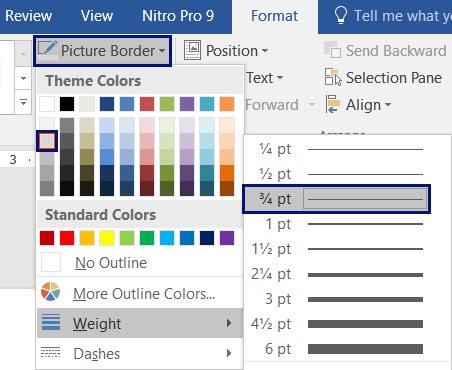
* **Good example** (the image is split into several smaller-height images):



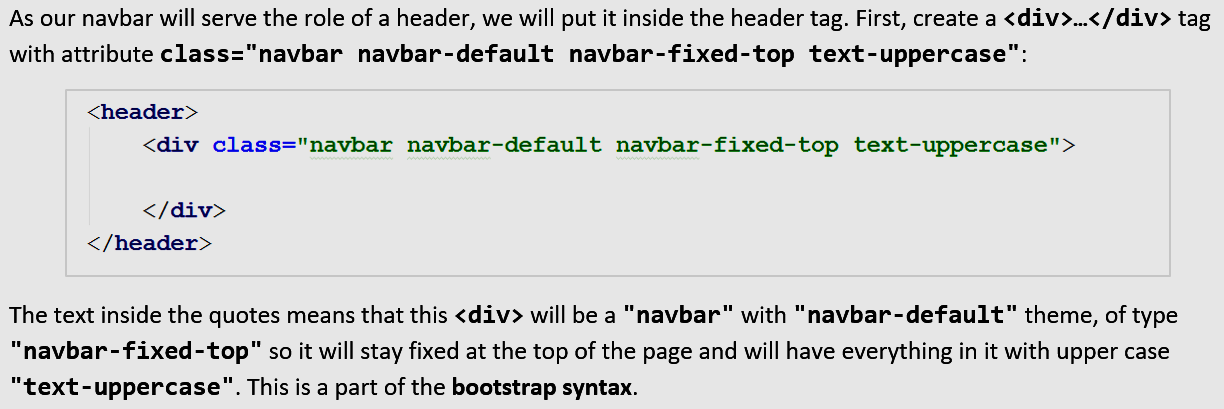




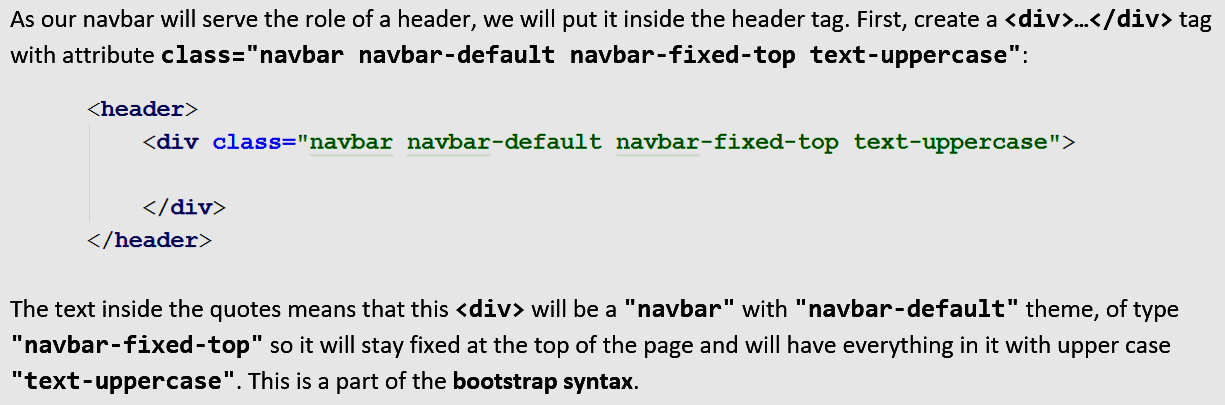
* Use **border around the code blocks**. This improves the document readability. Use the following **border**:



* **Good example** of **source code with border** around it:



* **Bad example** of **source code without border** around it:



#### Formatting Code Intended for Copy / Paste

When a piece of code is **part of the problem description**, and the student should use it directly by **copy / paste**, provide this code either as an **external file** (e.g. a ZIP archive with files and resources) or (when it is short enough), **put it directly in the text**. When you put code snippets, **intended for copy/paste**, format them in readable form:

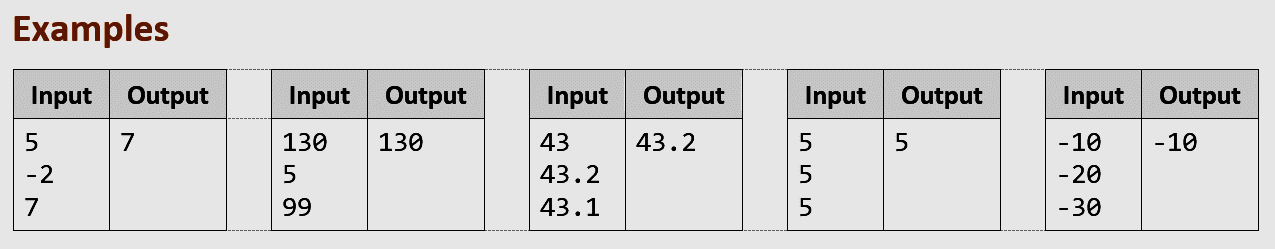
* Copy / paste the **colored code** from your **IDE**. Colors significantly improve code readability.
* Use font Consolas, 11pt. You may copy / paste the code from Visual Studio / Eclipse / Web Storm to preserve its coloring and later change the font in the MS Word document.
* Beware to preserve the correct **indentation of the code** (e.g. 2 or 4 spaces). Sometimes the **[Tab]** character is not displayed correctly in MS Word, so check the indentation after you paste the code.

Example of **well formatted source code**, intended to be copy / pasted:

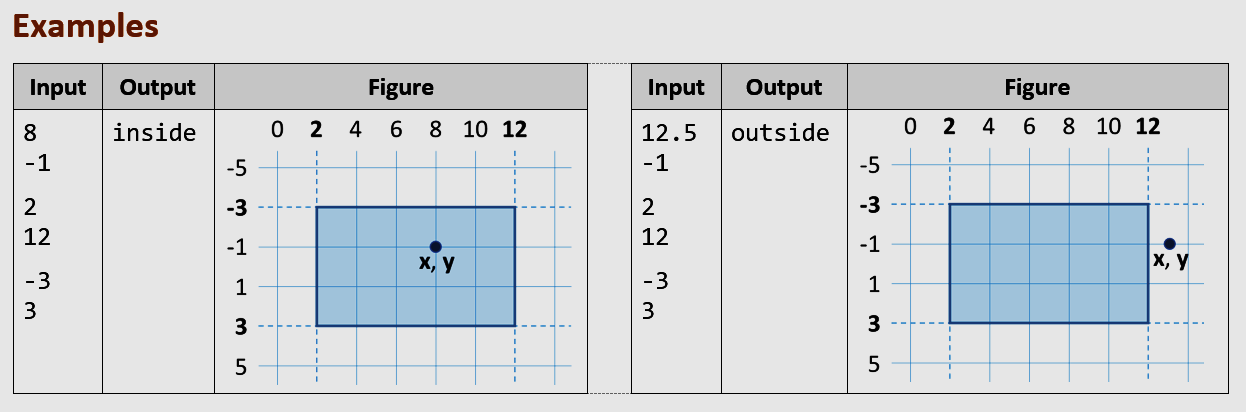
|  |
| --- |
| **calendar.html** |
| <!DOCTYPE html> <**html**> <**head**>  <**title**>Monthly Calendar</**title**>  <**style**>  .**prev-month**, .**next-month** { color: **#CCC** }  .**today** { font-weight: **bold**; background: **#DDD**; }  .**title** { background: **#AAAAFF**; margin: 10**px 0**; padding:5**px** }  **table** { border: 1**px solid #CCC**;}  **td** { text-align: **center**; }  **#calendarCode** { width: 100%; }  </**style**>  <**script**>  **function** *calendar*([day, month, year])  {  *//* ***TODO: return the HTML text holding the calendar table***  }  </**script**> </**head**>  <**body**>  Day: <**input** id=**"day"** type=**"number"** value=**"4"** />  Month: <**input** id=**"month"** type=**"number"** value=**"9"** />  Year: <**input** id=**"year"** type=**"number"** value=**"2016"** />  <**input** type=**"button"** value=**"Show"** onclick=**"let *calendarHTML*** =  *calendar*([**document**.getElementById(**'day'**).**value**,  **document**.getElementById(**'month'**).**value**,  **document**.getElementById(**'year'**).**value**]);  **document**.getElementById(**'calendar'**).**innerHTML** = ***calendarHTML***;  **document**.getElementById(**'calendarCode'**).**innerText** = ***calendarHTML*"** />  <**div** class=**"title"**>Calendar:</**div**>  <**div** id=**"calendar"**>Calendar will be shown here</**div**>  <**div** class=**"title"**>HTML:</**div**>  <**textarea** rows=**"12"** id=**"calendarCode"**></**textarea**> </**body**>  </**html**> |

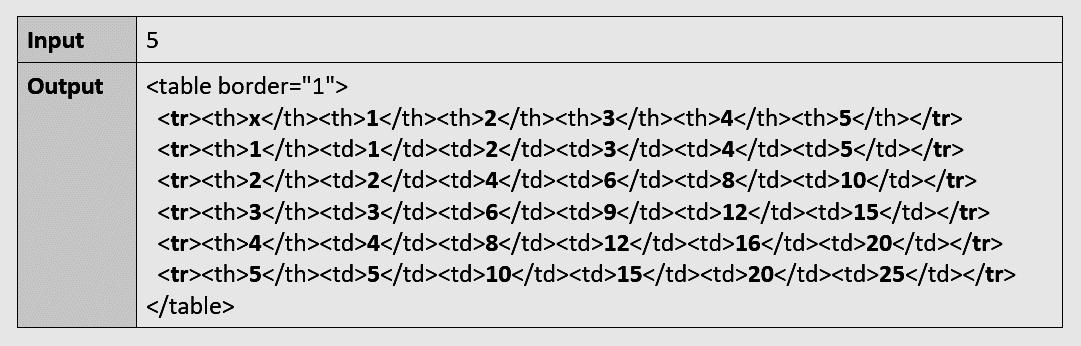
### Input / Output Examples Formatting

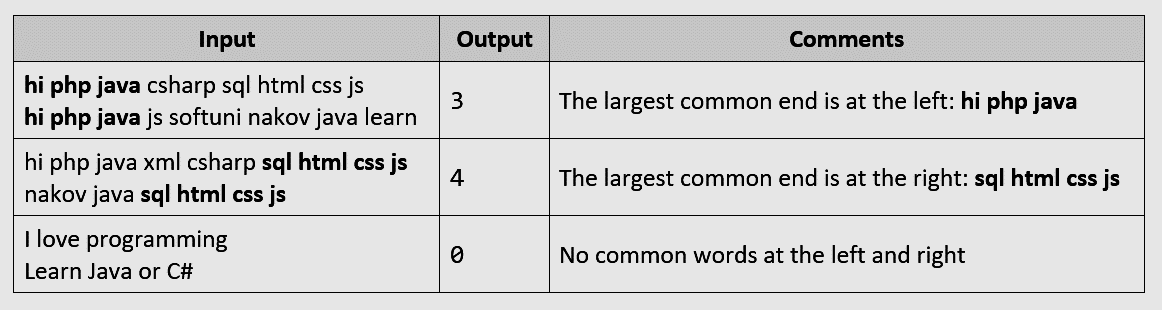
Format the **input / output examples** in **tables** like shown below. When the input / output is short, put several examples in the same table to save space in the document:



When the **input / output is longer**, use different **table layouts**:

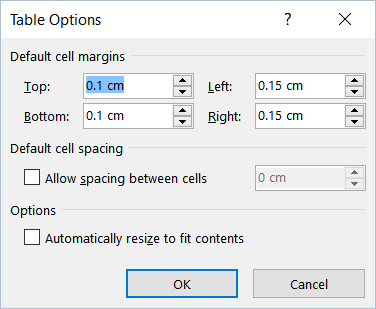
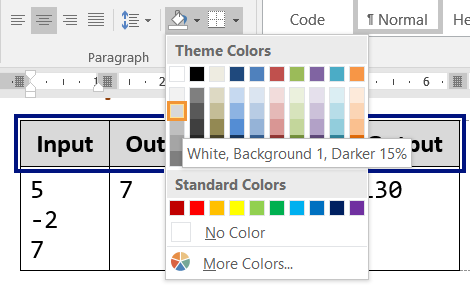


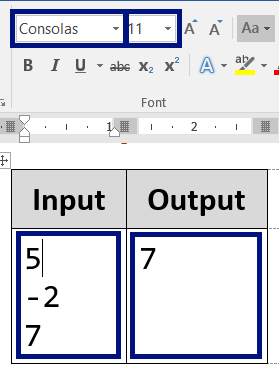
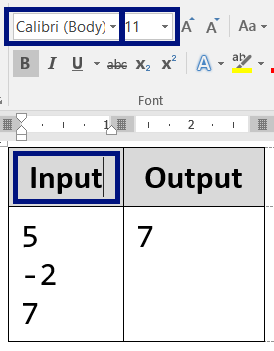
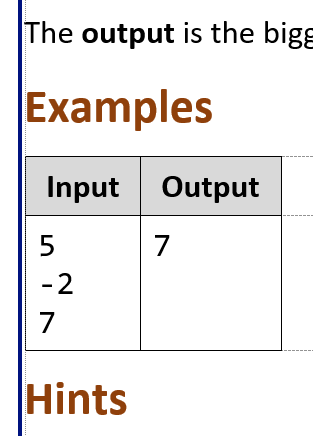




**Format the tables** holding sample input / output like in the sample document:

* **Margins**: 0.1 cm (top + bottom), 0.15 cm (left + right)
* **Header**: background: 15% grey, font: **Calibri, 11pt, bold**
* **Input / output rows**
  + Font: **Consolas, 11pt, bold**
  + You may use a smaller font size by exception to fit the code
* **Left align the table borders** at the same level with the text (see the image below).

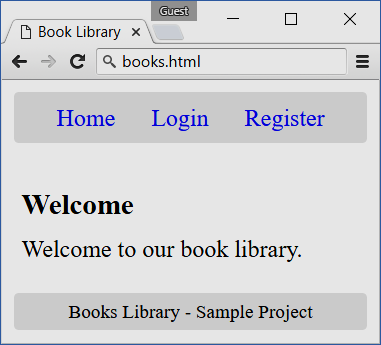
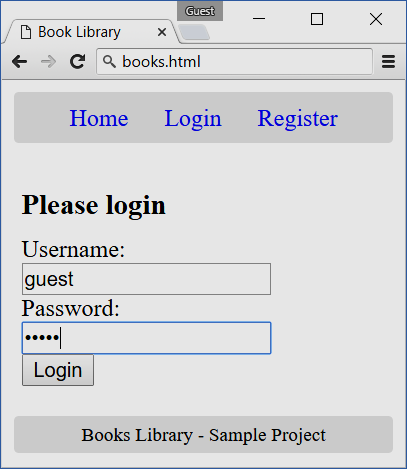
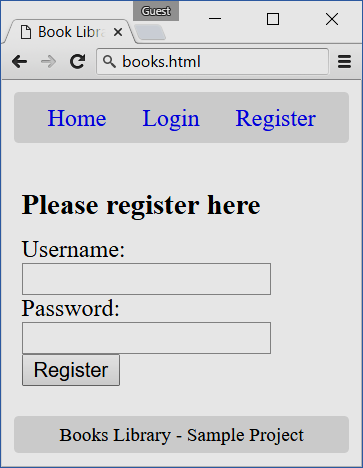
  

### Screenshot Formatting

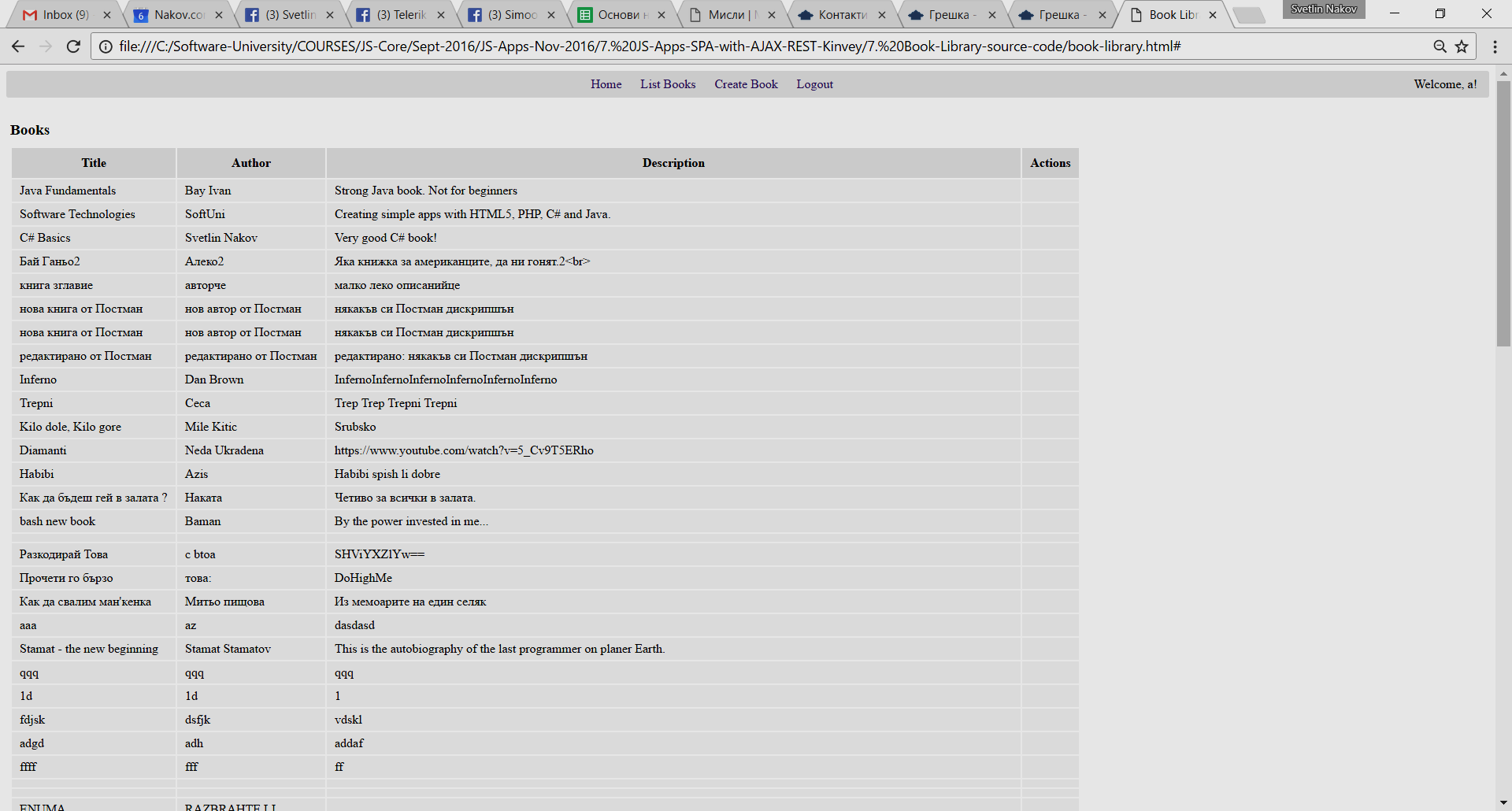
When you put **screenshots** in the document, use **small images** with **large fonts**.

* Only show the **essential part** of the screen.
* Use **large fonts**, so the text to be readable.
* **Keep images small**. Put several images on a single row when possible.

**Good screenshots** (home page + login form + user registration form – 3 small screenshots on a single row):

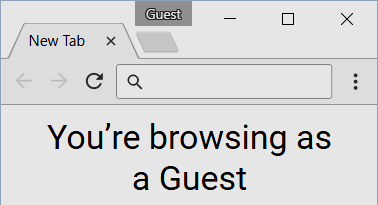
  

**Bad screenshots** example:



Hints for **better screenshots**:

* Use anonymous browser (**Guest session**), instead of logged in browser.

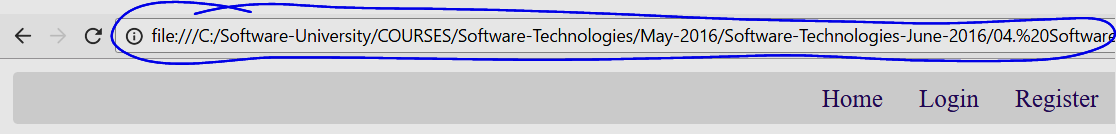


* You may take a **partial screenshot** (without the browser title) like this:

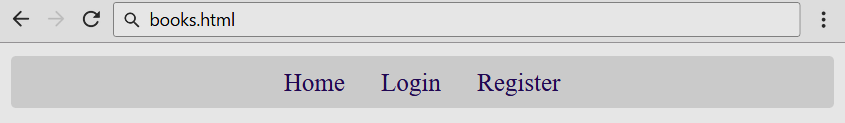


Such a screenshot illustrates very well the intended functionality and does not lose space.

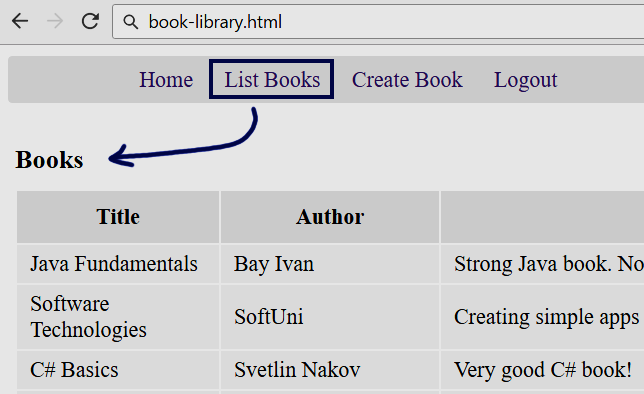
* Keep clean the **URL bar** in the browser. It should hold the file name without long paths. **Bad example**:



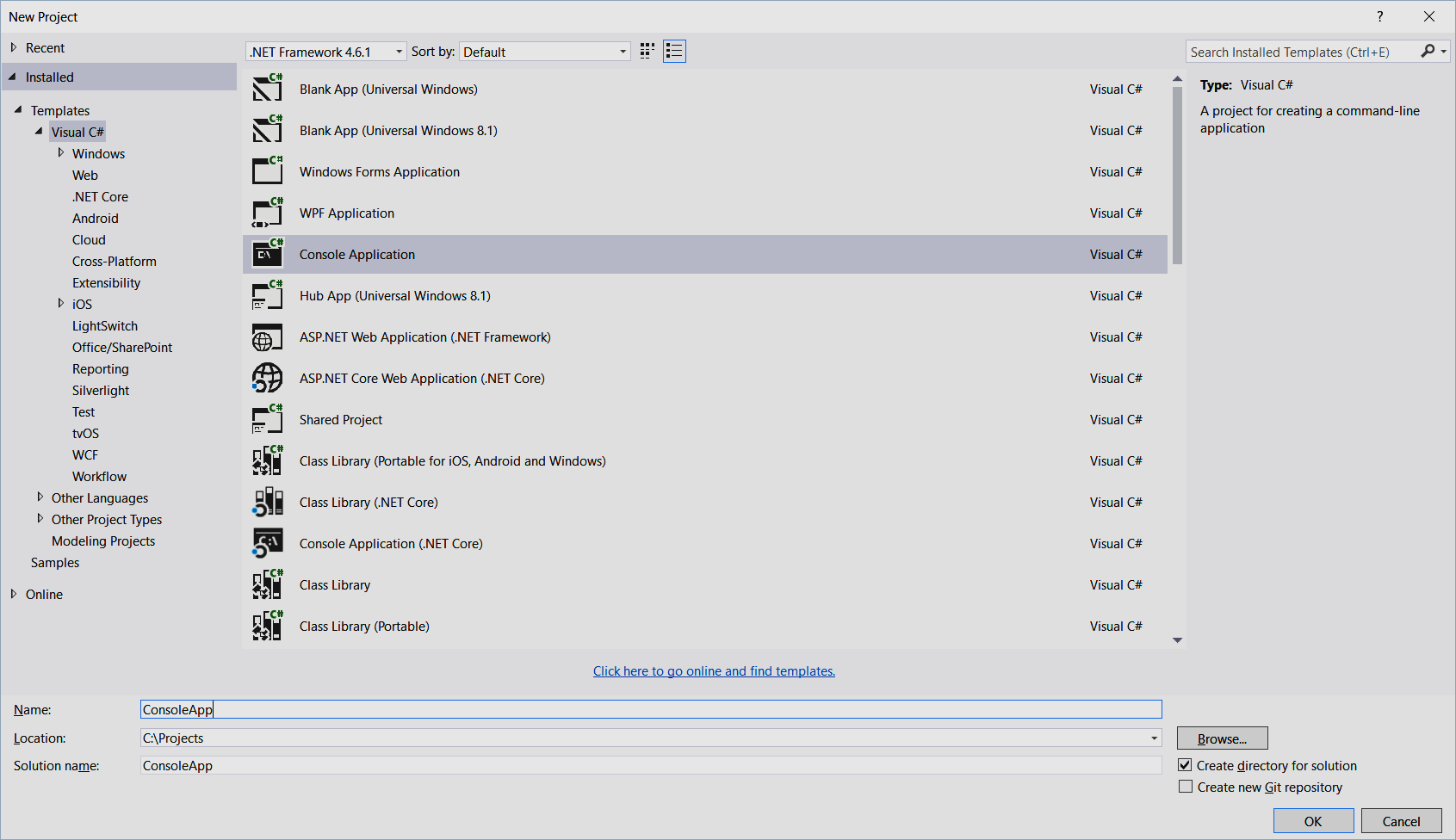
* The above may be fixed, by just **editing the URL bar** in the browser (yes, you may change it):



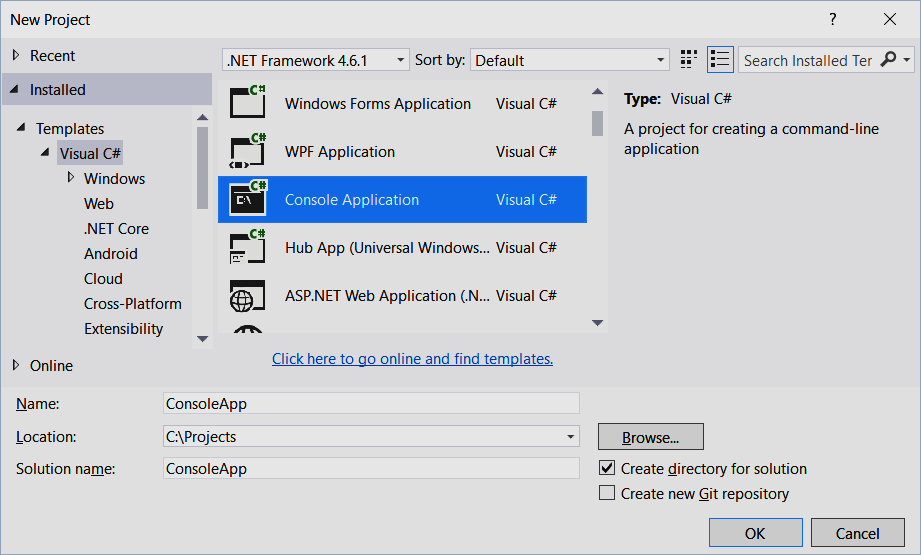
* Accent at certain screen areas using a **blue rectangle** like shown below. You may also use **blue arrows**. Use **dark** **blue color** (not red or green). You may use red for problems or errors (to show something wrong).



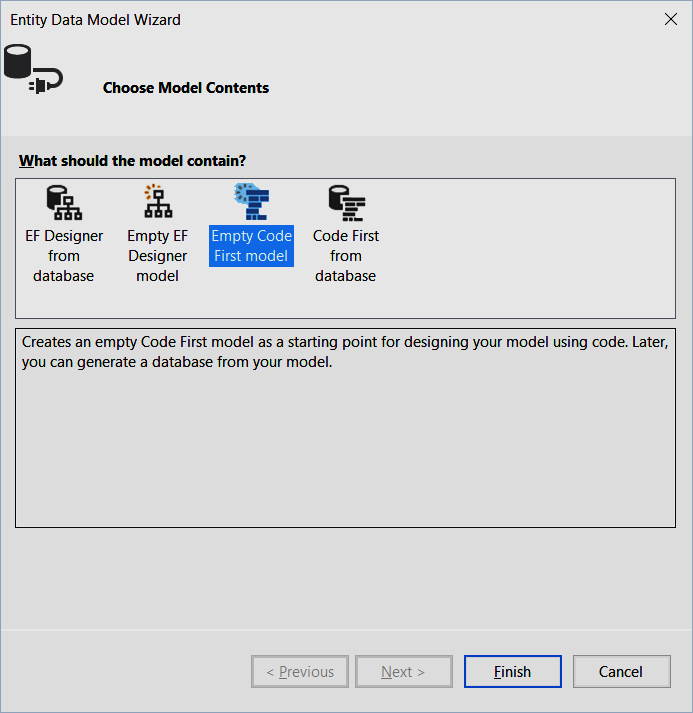
* Sometime you may **edit the screenshots** in some image editor (like [Paint.net](http://www.getpaint.net/)) to make them smaller.
* **Bad example** ofbig screenshot with **small unreadable text** (before editing), which loses a lot of space:



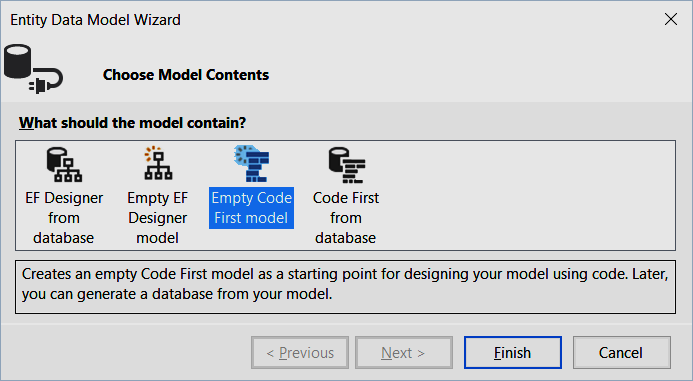
* **Good example** of smaller screenshot (after editing), showing the same screen using less space:



* Sometimes, even **the smallest possible window** taken as a screenshot is **too large**. In the following example, the window height is too much:



* The same screenshot can be **further edited in Paint.net** (or another image editor) to become smaller:



## Judge Tests

**Automate** the check-up for **as much as possible problems**, ideally for all of them.

* Use the **SoftUni Judge System** for automated testing.
* Use **input / output based tests** for the console-style problems (text input + text output).
* Use SQL tests for DB problems, JS DOM tests for JS UI problems and HTML.
* Write custom **C# unit tests** / **Mocha unit tests** with mocking to check classes and other code that does not run through the console.

How to **design test cases**?

* Add the tests from the examples as **open tests** (input + output is visible after each submit).
* Add a few **more open tests**, covering most of the special cases.
* Add a few **closed tests** (hidden output, not visible for the students).

## Typical Mistakes

Let’s review some **typical mistakes** made by the exercise authors.

### Bad Exercise Title

|  |
| --- |
|  |
| * Compare what? * Titles should be **short**, but **descriptive**. * A correct title might be the following: |
|  |
| * The above title if **over-descriptive**. It can be made shorted and still descriptive. * A correct title might be the following: |

### Bad Title Formatting

|  |
| --- |
|  |
| * Keep **uppercase each word** in the document titles. * The correct formatting is the following: |

### Too Much Text in the Problem Description

|  |
| --- |
|  |
| * The **text is too long** and boring. * Avoid giving **unneeded stories**. * Explain the problem **by examples**, **figures** and **screenshots**, not by long text. * **Structure** the text better, e.g. with bullets. |

### Bad Text Formatting

|  |
| --- |
|  |
| * Keywords / key phrases should be formatted in **bold** to improve readability. * Variable names should be in “Code” style. * The **spell checker** should be configured to ignore the code elements (like variable names). * This is how the reformatted text looks like (**good example**): |

### Bad Table Formatting

|  |
| --- |
|  |
| * Incorrectly configured **spell checker**. * Incorrectly formatted **table**: incorrect margins and coloring. * Table **left border position** incorrect (the table should be positioned slightly on the left). * This is the **correctly formatted table** (good example): |

### Inappropriate Fonts / Colors / Styles

|  |
| --- |
|  |
| * Use **consistent formatting**. At SoftUni we use the same formatting guidelines and styles in all documents in the technical trainings: homework assignment, exams, tutorials, teamwork project descriptions, etc. * The fonts, colors and styles are already defined, just follow them. |

### Broken Page Footer

|  |
| --- |
|  |
| * **Page footer** (and also the document styles) might **get broken** when the document is edited by non-Microsoft editors like OpenOffice, Google Docs, Apple iWork or another tool. * Use the **original document template** for SoftUni and the original **Microsoft Word 2016** / 2012 / 2010. * Keep the documents in **DOCX format**, not DOC or ODT. |

### Missing Sample Input / Output for Console Programs

|  |
| --- |
|  |
| * This problem description is pointless without **sample input + output**. * It says nothing about many **special cases**: less than 3 numbers, 0 numbers, negative numbers, equal numbers, non-integers, etc. * The best way to fix this: give **sample inputs + outputs** which cover the nominal case and all special cases. |

### Missing Screenshots for UI Problems

|  |
| --- |
|  |
| Insert screenshots to give a visual idea what the student should build: |

### Bad Sample Input / Output

|  |
| --- |
|  |
| * Always give **several examples**. The sample input + output is only one, not enough for this problem. * It is not clear which number is **n** and which is **k** (the first or the second). * No example explains what happens if **n** < **k**. |
| An **improved set of examples** answer the above questions and explain what happens in all **special cases**. The below examples + comments explain the problem very well without need of detailed text explanations: |

### Bad Hints

|  |
| --- |
|  |
| * The first problem is assumed to be the easiest and hints should provide **80%-90% of its solution**. The hints should explain the **steps of the solution** and should hold the most of the source code for each step. * The above hints are meaningless. They don’t give **steps**, don’t give **source code** and don’t give explanations what to search in Internet. |