Scientific Figures and Presentations in Ipe
Winter Term 2026/27
scale.iti.kit.edu

Karlsruhe Institute of Technology

Exercise Sheet 2: Customization

Submision due by November 22

On this exercise sheet you learn how to customize Ipe to your likings. In some of the exercises you only modify a file called *customization.lua*. Hand in this file once you are done with all of the exercises.

1 Basic Setup

Look at your Ipe configuration [Help → Show configuration] to find out where your Ipelets are stored; see for an example Figure 1. If multiple directories are listed you may choose any one of the directories that are listed. Later on you will also need the style directory, lua code directory as well as the latex directory.

Go to your Ipelet directory[1] and create a file called *customization.lua*. First, to make sure that you are in the right directory, add the line `shortcuts.close = nil` to the file. Then, close Ipe if it is open, open it again and press [Ctrl + W]. If Ipe did not close, you succesfully removed the shortcut for closing Ipe.

When adding a new shortcut you always have the same pattern:

```
shortcuts.name_of_the_function="some_keys_or_nil"
```

In the example before, the name of the function is `close` and instead of some combination of keys we set the shortcut to `nil`, meaning we removed the shortcut. Try now to add a shortcut for editing groups. The name of the function is `edit_group`. You should also add a shortcut for ending group edit. The name for the function somewhat surprisingly is not `end_edit_group` but rather `end_group_edit`. For a list of all functions that can be assigned a shortcut, refer to the document *list_of_shortcuts.pdf*.

Warning: When you assign some shortcut that is already in use (e.g., [Ctrl + E]) Ipe will do neither of the functions assigned with this shortcut upon pressing the keys. To find out which shortcuts are already in use you can add the line `prefs.developer = true` to your *customization.lua* file. Then, you can go to [Help → Developer → List shortcuts]. This will print all shortcuts to the terminal (provided that you started Ipe via the terminal[2]). Alternatively, you can go here and install the Ipelet as explained in Section 2.

For some of the tasks you need to have an external editor. If you are using Ipe on a Windows or Mac machine, there should be an external editor set by default. If you are using some other operating system you need to explicitly add one to your *customization.lua*. In the following you see two examples how to do this:

```
prefs.external_editor = "emacsclient %s"
prefs.external_editor = "code --wait %s"
```

The first line sets Emacs as the external editor. The second line sets Visual Studio Code as the external editor. The variable `%s` contains the path to the current file.

[1] On MacOS, accessing the Ipelet folder might involve rightclicking the Application Ipe and selecting "Show Package Contents"

[2] On Windows you can start Ipe using the command `.\bin\ipe.exe | Out-Host` to see the output.

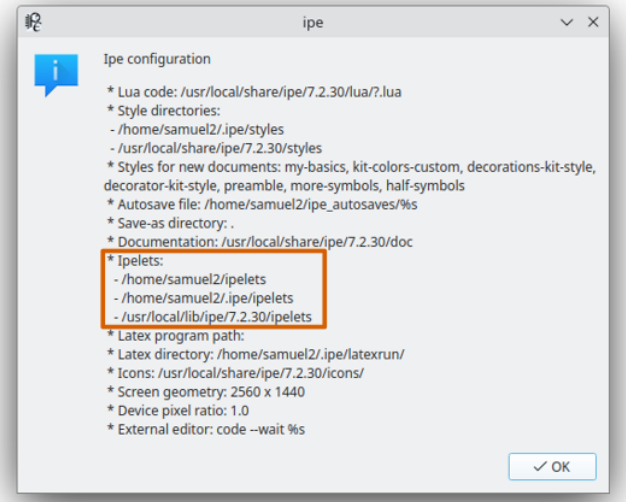

Figure 1: An example for an Ipe configuration. The directories containing the Ipelets are listed in the seventh entry.

2 Installing Ipelets

Ipelets are small programs written in lua or C++ that can enhance Ipe. In this course we only focus on Ipelets written in lua. Such an Ipelet consists (almost always) of a single lua file that you have to copy to one of your Ipelet directories. So for example in the same directory as your *customization.lua* file. As an example we are going to install the *offset* Ipelet. To do this, go to

`https://github.com/thobl/ipelets/tree/master/offset`

and download the file *offset.lua* into your Ipelet directory. To verify that you succesfully added the Ipelet, (re-)open Ipe and go to the *Ipelet* tab. You should see an entry with the name *Offset*.

Next, we want to add shortcuts for the functions provided by the Ipelet. To do this, go to your `customization.lua` file and add the line

```
shortcuts.ipelet_1_offset = "Shift+Alt+P"
```

Verify that you succesfully added the shortcut by opening Ipe, drawing a polygon, and pressing `[Shift + Alt + P]` (with the polygon still selected).

As you probably noticed the syntax for adding a shortcut for an Ipelet is almost the same as for adding any other shortcut. Only the naming scheme of the function differs a bit. To add a shortcut for a function of an Ipelet you need two things: the file name of the ipelet and the number of the function. If an Ipelet provides only a single function, this function has the number `1`. An example for this is the *Select by type/attribute* Ipelet that is included in Ipe by default. If you want to add a shortcut for this Ipelet you can do so by adding the following line to your *customization.lua*.

```
shortcuts.ipelet_1_selectby = "Shift+S"
```

Other Ipelets like the *Offset* Ipelet provide multiple functions. For these you can find out the number of a function by going to the Ipelet in the Ipelet tab. For example, if you go to the Offset Ipelet you can see that *Offset path* is the first entry in the list. So the number of this function is `1`. Similarly, the function *Offset area* has the number `2`. To complete this task, add a shortcut for the *Distribute horizontally* function provided by the *Align & Distribute* Ipelet that is included in Ipe by default.

3 Snappy's Favorite Colors

Another way to customize Ipe is by creating *stylesheets*. These are files that can add new colors, symbols, textstyles, and much more. The goal of this task is to create a stylesheet *snappys-colors.isy* providing additional colors. More precisely, your task is to add all colors used in the file *snappys-colors.pdf* to this stylesheet. To do this, create a file *snappys-colors.isy* in your style directory. Then, add the following content to it:

```
<ipestyle name="snappys-colors">
<color name="snappyblue1" value="0.227 0.463 0.941"/>
</ipestyle>
```

The first and last line begin and end the scope of the stylesheet. The `name` attribute in the first line is the name of the stylesheet. It is usually a good idea to have this be the same as the name of the file. The second line defines a new color with the name *snappyblue1*. The `value` attribute corresponds to the decimal RGB values of the color.

One way to obtain the RGB values of the colors is to use the color picker provided by Ipe. To do this, click on the button on the top-left of the properties (see Figure 2). Depending on your operating system, you should now see a button named something like *Pick Screen-Color* or an equivalent option that lets you pick the color of a pixel of your choice on your screen. Doing this should also display you RGB values between 0 and 255. To obtain the decimal RGB values, you have to divide each number by 255 and round it to three decimal places. Another way to obtain the RGB values is the following: create an object with the absolute color you just picked as stroke color, then select the object and edit its XML [`Rightclick → Edit as XML`]. The RGB values are the three numbers following the substring `stroke=`.

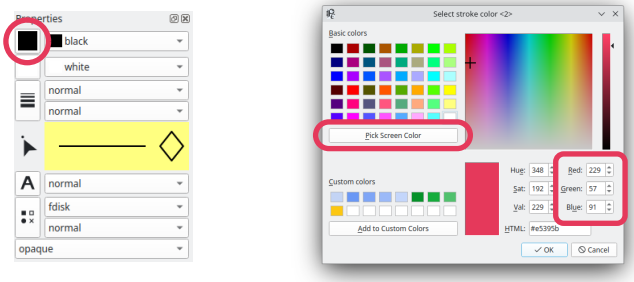

Figure 2: The button you have to press to get to the color picker and the window it opens (the window on the right could look different on your OS).

Open now the file *lost-colors.ipe*. You should get a warning that some symbolic attributes are not defined. The warning also tells you the names of these missing symbolic attributes[3]. You should see that all "color blobs" are displayed as if they had the stroke color black. Check their XML ([Rightclick → Edit as XML]) to see that each blob, however, has assigned a stroke color from the list of undefined attributes. Define the missing colors in *snappys-colors.isy*, using the color picker as described above. The leftmost blue in *snappys-colors.pdf* is the color *snappyblue1* which we already defined above. In total, your stylesheet should contain 11 colors. Finally, to ensure that you have succesfully added the colors, add the stylesheet to *lost-colors.ipe* [Ctrl + Shift + S → add]. It should look like *snappy-colors.pdf* without changing anything else.

What to hand in? Hand in the *lost-colors.ipe* file with the *snappys-colors.isy* added and nothing else changed.

4 Symbols

Symbols in Ipe are shapes that you define in a stylesheet to be able to easily re-use them. You encounter these naturally when using Ipe in a variety of shapes. For example marks are symbols defined in a stylesheet. So every time you create a mark, you create an object that is simply a reference to a definition in some stylesheet. The same is true for arrows and decorations.

Your first task is to create a Snappy symbol. To do so, open your Ipe file from the last sheet that contains Snappy, select a single horizontally aligned copy of Snappy (it should be a single group) and create a new Symbol [Ipelet → Symbols → create symbol (in new stylesheet)] with the name *snappy*. Now, try to place another copy of Snappy by using the symbols Ipelet [Ipelet → Symbols → use symbol] and observe what is happening.

As you probably noticed, the symbol is not placed at your mouse cursor. When creating a new symbol, the center of the symbol is the point $(0, 0)$. So when creating a symbol, you want it to be centered at this point. Delete the stylesheet you just created [Ctrl + Shift + S → delete], center Snappy (for any reasonable definition of center) at the point $(0, 0)$[4] and create the symbol again [Ipelet → Symbols → create symbol (in new stylesheet)].

You might have already noticed that the Snappy symbol only has a single snapping point. This can be a bit incovenient. Imagine for example building the figure from the previous sheet consisting of 12 Snappys with only a single snapping point. To change this, it is probably convenient to first save the stylesheet containg the Snappy symbol to your style directory [Ctrl + Shift + S → save] to be able to use an editor of your choice (while still being able to work in the Ipe file). Currently, the stylesheet should look something like this:

```
<ipestyle>
<symbol name="snappy">
<group>
    some messy stuff
</group>
```

[3] You can also display a reminder of that list with [Edit → Check symbolic attributes]

[4] Ipe always shows you the coordinates of your cursor in the bottom right

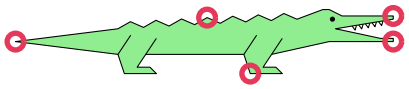

Figure 3: The points you should make snapping points: The tip of the tail, the top-most and bottom-most corners of the jaw, the tip of the middle back-spike, and the bottom-left corner of the front foot.

```
</symbol>
</ipestyle>
```

Now is a good time to add a name to the stylesheet by adding `name="snappy"` to the line `<ipestyle>` (inside of the angle brackets). To add snapping points, you can add an attribute `snap` to your symbol with a list of coordinates as value. For example, the following would create snapping points at the coordinates $(1, 2)$ and $(3, 4)$.

```
<ipestyle>
<symbol name="snappy" snap="1 2 3 4">
<group>
    some messy stuff
</group>
</symbol>
</ipestyle>
```

The points you are supposed to make snapping points are shown in Figure 3. To obtain the coordinates it might be useful to use snap to vertex [F4] and disable the other snapping modes. Note that the coordinates of your mouse cursor are shown on the bottom-right of the Ipe window. When you have added all the snapping points to your stylesheet, delete the old version of the stylesheet from your Ipe file [Ctrl + Shift + S → delete] and add the updated stylesheet [Ctrl + Shift + S → add]. To avoid having to delete and then re-add a stylesheet, Ipe lets you update all currently loaded stylesheets at once [Ctrl + Shift + U]. For each loaded stylesheet, this function searches your current directory as well as the style directory for a stylesheet of the same name. Then, if it finds such a stylesheet, it replaces the currently loaded stylesheet by the one in your style directory. For this to work properly, you want the name of the file and the name of the stylesheet to be the same, i.e., in this example you want the file containing the stylesheet to have the name *snappy.isy*. To demonstrate that you succesfully added all the snapping points, re-create the figure from the third task of the first exercise sheet.

What to hand in? Hand in a file *snappy.ipe* that contains the final result. Each snappy should be a symbol.

In the rest of this task you will create several new symbols. You should add all of them into a single stylesheet with the name *symbols* that you add to the file *symbols.ipe*. This should result in the file not causing any warning because symbols are missing and all your symbols should be displayed. You should not change anything else in the file. Any colors you use in your symbols should be display properly.

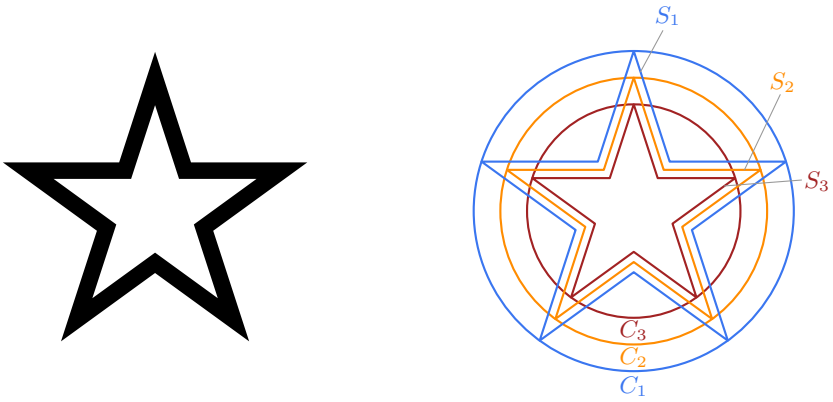

Figure 4: On the left is the mark you should create. On the right is a depiction of the three stars that form the mark.

4.1 Marks

As mentioned earlier, marks are a special kind of symbol. More precisely, a mark is a symbol with a name that starts with the prefix `mark/`. Furthermore, it ends with a sequence of letters indicating the tranformable attributes of the symbol. The possible letters are `s` (stroke), `f` (fill), `p` (pen), and `x` (size). The legal suffixes for marks are of the form `(Ax)` with A being a sub-sequence of `sfp`. In particular, the size of a mark is the only attribute that has to be transformable. For each of the other attributes that you make transformable, you get access to the symbolic value with the name `sym-stroke`, `sym-fill`, and `sym-pen`, respectively. These stand for the currently active value of the respective attribute at the time of the creation of the symbol.

Your task is to create a star-shaped mark type with the name `mark/fstar(sfx)`. Your mark should look like Figure 4 and fulfil the following:

- The mark consists of three stars S_1, S_2, and S_3. The stars S_1 and S_3 are composed to form an area that is colored with `sym-stroke`. The third star S_2 is colored with `sym-fill`. Both objects are set to filled only.
- The tips of each star are the corners of a regular pentagon.
- For each star S_i, the cycle C_i going through its tips is centered at $(0, 0)$.
- The radii of the cycles C_1, C_2, and C_3 are the same as the radii of the cycles that the `fdisk` mark is composed of. The cycle C_1 is the largest, followed by C_2.
- Each star is the result of taking a straightline drawing of the complete graph with the vertices positioned at the tips of the star, deleting the bounding cycle, and taking the union of the segments bounding this drawing.
- Each star is rotated such that one of its tips is at the top.

For this task it might be useful to use the *clone symbol* function `[Ipelet → Symbol → clone symbol]` to get a better look at the `fdisk` mark or to look at the definition of `fdisk` in the stylesheet *basic* that is loaded by default.

4.2 Arrows

Similar to marks, arrows are just another type of symbol. Their names start with the prefix `arrow/` and the only allowed suffixes are `(sfpx)` and `(spx)`. That is,

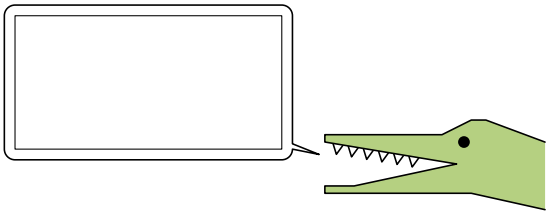

Figure 5: The decoration you should create. The rectangle is not part of the decoration.

all attributes but fill must be transformable. There are two types of arrows: the "normal" arrows and the "mid" arrows. Normal arrows have their tip at the end of a polyline. Mid arrows have their tip at the middle of a polyline. More precisely, for a normal arrow, the center of the symbol (so whatever is at the coordinate $(0, 0)$) is placed at the end of the polyline. For a mid arrow, the center of the symbol is placed at the middle of the polyline. To indicate that an arrow is a mid arrow, its name starts with the prefix `arrow/mid-` while a normal arrow does not start with this prefix. Create two new arrows with the names `arrow/cool-new-arrow(spx)` and `arrow/mid-cool-new-arrow(spx)`. The only requirement is that they are distinct from all arrows that Ipe contains by default.

4.3 Decorations

Sometimes (especially when creating presentations) you want to put some things (most often text) into some boxes. Since this is very tedious to do by hand, Ipe offers a more sophisticated solution: decorations. A decoration is a type of symbol that can be used to "decorate" groups. To see how this looks, you can add the stylesheet *decorations.isy* that is installed with Ipe by default and add a decoration to a selected group [right click → Decoration].

To create a decoration you should first create a rectangle with the corners at the points $(100, 100)$, $(300, 100)$, $(300, 200)$, and $(100, 200)$. Alternatively, you can add any object, edit its XML [Rightclick → Edit as XML], and replace everything by the following.

```
<path stroke="black">
100 100 m
300 100 l
300 200 l
100 200 l
h
</path>
```

The rectangle you just created is not part of the decoration but rather simulates the bounding box of the group you want to decorate. Now, you can create your decoration. It should consist of a single group consisting of path object only, i.e., polygons, splines, and arcs. When you are done you can add it as a new symbol [Ipelet → Symbols → create new symbol (in new stylesheet)]. To indicate that it is a decoration the name of the symbol has to start with `decoration/`.

Create a decoration with the name *decoration/snappy* that looks like Figure 5.

What to hand in? Hand in the file *symbols.ipe* that contains the final result. The mark, the two arrows, and the decoration should be displayed properly.

5 More Options

In the tasks up to this, you already saw that stylesheets can provide new colors and new symbols. But stylesheets can provide more options to many kinds of attributes. Some attributes can be assigned both an absolute value (e.g., size of a text object or symbol) and some attributes can only be assigned a symbolic value (e.g., opacity or size of an arrow). In this task, you create new symbolic values for some of these attributes.

What to hand in? Hand in a single file *options.sty* that contains everything you create in this task. Make sure that including the stylesheet in an Ipe file does not lead to any errors.

5.1 Sizes

Various different things in Ipe can be assigned a size. For each of those you can define a symbolic size as shown here:

```
<symbolsize name="small" value="2"/>
<arrowsize name="large" value="10"/>
<pen name="normal" value="0.4"/>
<anglesize name="30 deg" value="30"/>
<gridsize name="30 pts" value="30"/>
```

The definition consists of the `name` of the symbolic attribute and the `value` of the symbolic attribute. The first can be any string starting with a letter. The second can be any positive number[5] (note that `\small` is also a number).

To complete this task, create the following:

- a symbolsize with the name *huge* that is twice the size of the symbolsize *large*,
- an arrowsize with the name *huge* that is twice the size of the arrowsize *large*,
- a pen with the name *thin* that is half as thick as the pen *normal*,
- an anglesize[6] with the name *180 deg* and the value *180*, and
- a gridsize with the name *1 pts* and the value *1*.

5.2 Opacity

Similar to the different sizes from before, a symbolic attribute for opacity consists of a `name` and a `value`. The value, however, has to be a number from the interval $[0, 1]$. Create symbolic attributes for opacities 20%, 40%, 60%, 80%, and 90% similar to the following.

```
<opacity name="75%" value="0.75"/>
```

[5] Gridsize can only be a positve integer.

[6] The anglesize determines the size of the angle of the coordinate system that you can set with [F1].

5.3 Dashstyles

We already created some symbolic attributes that can be used by path objects. Namely, we created possible values for the attribute types "pen" and "arrowsize". A slightly more complicated attribute type is *dashstyle*. A symbolic attribute for the dashstyle consists once again of a `name` and a `value`. The value, however, is a bit more complicated. For example the symbolic attribute `dash dot dotted` that is defined in the `basic` stylsheet included by default looks like this:

```
<dashstyle name="dash dot dotted" value="[4 2 1 2 1 2] 0"/>
```

The `value` consists of two parts. The first part inside the squarebrackets consists of a sequence of positive numbers indicating the length of each segment. For example, the sequence `4 2 1 2 1 2` means that the line starts with a segment that is 4 points long, followed by a gap that is 2 points long, followed by a segment that is 1 point long, and so on. The second part, i.e., the `0` after the squarebrackets in this example, indicates an offset. For example, if we replace the `0` in this example by a `1`, the first segment would be only 3 points long; everything else would be unchanged. Create a new dashstyle with the name *densly dotted* that alternates between segments of length 1 and gaps of length 1.

5.4 Layout

Sometimes the default canvas of Ipe is not the right size. Maybe you want to make a presentation and want a wider canvas or you want to make some notes in Ipe and want a much larger canvas. A layout consists of three mandatory attributes:

- the attribute `paper` that consists of two number that indicate the width and the height of the paper in points,
- the attribute `origin` that consists of two numbers that indicate the x- and y-coordiante of the origin of the grid, i.e., the bottom-left corner of the grid,
- the attribute `frame` that consists of two numbers that indicate the width and the height of the grid.

Additionally, there is an optional attribute `crop` with the value `yes` or `no` (the default is `yes`). If crop is set to `no`, the resulting pdf is not cropped to the bounding box of the visible objects. Instead the whole canvas is displayed. The layout used by default is the following:

```
<layout paper="595 842" origin="0 0" frame="595 842"/>
```

Create a new layout with the same size of the paper, a grid that is centered on this paper with a distance of 100 points to each side of the paper, and cropping turned off.

5.5 Tilings

Filled path objects can be assigned a tiling pattern (with `[Rightclick on object → Tiling pattern]`). The `basic` stylesheet defines the following two patterns:

```
<tiling name="falling" angle="-60" width="1" step="4"/>
<tiling name="rising" angle="30" width="1" step="4"/>
```

The attributes `angle`, `width`, and `step` are each assigned a number indicating the angle of the tiling pattern, the width of the colored parts and the period of the tiling pattern. Create a new tiling pattern with the name *rising_wide* such that the stripes in the patterns are going from bottom left to top right with an angle of 45°. Additionally, the stripes and the gaps between them should both have width 4.

5.6 Text Sizes

Similar to marks and arrows, you can add symbolic attributes for the size of text. Once again, the definition consists of a `name` and a `value` of the symbolic attribute. The value, however, "is a piece of LaTeX source code selecting the desired font size" according to the Ipe manual. For example, the following is the definition of the textsize `small`:

```
<textsize name="small" value="\small"/>
```

However, you can also add absolute values or set the font to bold using `\bf`. Create a new textsize with the name *smallcaps* that has the same size as `small` but is set in smallcaps.

5.7 Text Styles

There are two different types of text objects in Ipe: *labels* [L] and *paragraphs* [G]. Labels display the text in a single line and have variable width. Paragraphs have a fixed width (that you can change later on [Alt + W]) and behave similar to minipages in latex.

Each text object has an attribute `textstyle`. By default, labels can have one of the two styles *normal* and *math* that are defined as follows:

```
<textstyle name="normal" type="label" begin="" end=""/>
<textstyle name="math" type="label" begin="$" end="$"/>
```

The `begin` and `end` attributes can contain any string, which are placed before and after the text of the text object before compiling latex. The `type` attribute is set to `label` to indicate that this is a textstyle for labels. If it is omitted, this results in a textstyle for paragraph objects.

The default styles for paragraphs are defined as follows:

```
<textstyle name="center" begin="\begin{center}"
end="\end{center}"/>
<textstyle name="itemize" begin="\begin{itemize}"
end="\end{itemize}"/>
<textstyle name="item" begin="\begin{itemize}\item{}"
end="\end{itemize}"/>
```

Create a new style for labels that adds a `\strut` before the text. Ensure that the text is printed without leading whitespace. Additionally, create a new style for paragraphs that prints the text as a subitem.

6 Latex

You already saw how to add things to your preamble via [Ctrl + Shift + P] on the first sheet. Sometimes there are things you want to add to almost all of your preambles. In this case, it can be handy to add them to a stylesheet. This way you can import them more easily (or include them by default). For example you can include the package qrcode like this:

```
<preamble>
  \usepackage{qrcode}
</preamble>
```

Create a new stylesheet with the name *preamble* and include the packages fontspec and amssymb analogously to the example shown above. Note that anything you define in a preamble in a stylesheet is loaded before the preamble in your document properties (the one that you open with [Ctrl + Shift + P]).

Add the stylesheet *preamble* that you just created to an Ipe document [Ctrl + Shift + S → add]. This should cause an error, telling you that you need to use luatex or xetex. In Ipe you can do this via the document properties [Ctrl + Shift + P → Latex engine]. Select the option luatex.

One option using luatex opens up is to use true type fonts. To do this, go to https://fonts.google.com/specimen/Press+Start+2P and download the font. Copy the file *PressStart2P-Regular.ttf* to your latex directory (see Section 1 for how to find this directory). To include the font, add \setmainfont{PressStart2P-Regular} to your stylesheet *preamble*. Try now to add a text object containing some text as well as the substring $\implies$. As you can see the $\implies$ symbol does not look quite right. The reason is that $\implies$ is constructed by using = and ⇒. Since the font you just loaded contains a symbol for = but not for ⇒ they do not match anymore. Even if you do not include a new font, $\implies$ looks a bit messed up sometimes. To avoid this, you can include the package unicode-math. Include this now in your stylesheet *preamble*. Note that the unicode-math package requires luatex.

What to hand in? Hand in a file *luatex.pdf* that contains some text in the *Press Start 2P* font as well as $\implies$.

7 Default Values

Ipe allows you to change the default values for most things. You can find out what you can change, by looking at the file *prefs.lua* in the lua code directory (see Section 1 for how to find this directory). Note that you should not change anything in the *prefs.lua* file. Instead you should make the changes in your *customization.lua* file. We recommend reading the *prefs.lua* file entirely, it is fairly easy to read and well documented. We only highlight a few things that we think are especially useful:

- changing the stylesheets that are loaded by default (for example to include some nicer colors and additional symbols),
- setting the autosave interval to something like 30 seconds (Ipe tends to crash from time to time),
- creating a directory for all the autosaves instead of storing them in the same directory as the current Ipe file,

- changing the snapping modes that are enabled by default,
- increasing the maximum zoom,
- increasing the size of the editor,
- increasing the threshold for joining two paths,
- automatically exporting the file as pdf when compiling it,
- setting the default grid size to something smaller,
- changing the default mark type to `mark/fdisk(sfx)`,
- changing the colors of the primary and secondary selection to make them better distinguishable (this probably only applies if you are color blind),
- opening the external editor by default when editing text objects (this is is probably only useful if you have an external editor that starts very quickly).

Find out for yourself which changes are good for you and include them in your *customization.lua*.

8 Edit as XML

Ipe allows you to edit most objects by changing the underlying XML [Rightclick → Edit as XML].[7] We look at two use cases for this. The first is to convert a polyline to a polygon (or a spline to a splinegon). To see how you can do this, compare the XML code of a polyline and a polygon with the same control points. Then, open the file *xml.ipe* and close the path object using this method. The second use case is to do mass edits. For example, if you have a large group and want to replace every occurence of the color *red* with the color *blue*, you can edit the XML of the group, open it in an external editor replace all occurences at once. Use this method to color all red marks in *xml.ipe* with the color blue instead.

What to hand in? Hand in the file *xml.pdf*.

[7] The only type of object where you should not try to do this are image objects. In earlier versions than 7.3.1 this will crash Ipe.