Starting

Starting typing in Typst is easy. You don't need packages or other weird things for most of things.

Blank line will move text to a new paragraph.

Btw, you can use any language and unicode symbols without any problems as long as the font supports it: ßçœ̃ɛ̃ø∀αβёыა😆 …

Markup

This was a heading. Number of = in font of name corresponds to heading level.

Second level heading

Okay, let's move to *emphasis* and **bold** text.

Markup syntax is generally similar to `AsciiDoc` (this was `raw` for monospace text!)

New lines & Escaping

You can break
line anywhere you
want using "\" symbol.

Also you can use that symbol to escape _all the symbols you want_, if you don't want it to be interpreted as markup or others special symbols.

Comments & codeblocks

You can write comments with `//` and `/* comment */`:

```
Just in case you didn't read the source, this is how it is written:

// Like this
/* Or even like
this */

By the way, I'm writing it all in a _fenced code block_ with *syntax highlighting*!
```

Smart quotes

What else?

There are no much things in basic "markup" syntax, but we will see much more interesting things very soon! I hope you noticed auto-matched "smart quotes" there.

Lists

- Writing list in a ssimple way is great.
- Nothing complex, start your points with `-` and this will become a list.
  - Indented lists are created via indentation.

1. Numbered lists star with `+` instead of `-`.
2. There is no alternative markup syntax for lists
3. So just remember `-` and `+`, all other symbols wouldn't work in an unintended way.
   1. That is a general property of Typst's markup.
   2. Unlike Markdown, there is only one way to write something with it.

Notice

Typst numbered lists differ from markdown-like syntax for lists. If you write them by hand, numbering is preserved:

1. Apple
1. Orange
1. Peach

Math

I will hist mention math ($a + \frac{b}{c} = \sum_1 x^i$) is possible and quite pretty there:

$$7.32 \beta + \sum_{i=0}^{\nabla} \frac{Q_i (a_i - \varepsilon)}{2}$$

To learn more about math, see corresponding chapter.

Functions

Okay, let's now move to more complex things.

First of all, there are **lots of magic** in Typst. And it major part of it is called "scripting".

To go to scripting mode, type **#** and **some funcion name** after that. We will start with *something dull*:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

*That **function** just generated 50 "Lorem Ipsum" words!*

More functions

functions can do everything!

Like Really Everything

Figure 1: This is an imaginary screenshot from one of first theses written in Typst.
All these things are written with custom functions too.

How to call functions

First, stat with #. Then write the name. Finally, write some parentheses and maybe something inside.

You can navigate lots of built-in functions in Official Reference.

> That's right, links, quotes and lots of other document elements are created with functions.
>
> — Typst Examples Book

Function arguments

There are *two types* of function arguments:

1. **Positional.** Like `50` in `lorem(50)`. Just write them in parentheses and it will be okay. If you have many, use commas.
2. **Named.** Like in `#quote(attibution: "Whoever")`. Write the value after a name and a colon.

If argument is named, it has some *default value*. To find out what it is, see Official Typst Reference.

Extra: Types/modes in Typst

- Math
- Content
- Script

Content

The most “universal” type un Typst language is **content**. Everything you write in the document becomes content.

But you can explicitly create it with *scripting mode* and **square brackets**.

There, in square brackets, you can use any markup functions or whatever you want.

Markup and code modes

When you use `#`, you are “switching” to code mode. When you use `[]`, you turn back:

hello **world**

Passing content into functions

So what are these suare brackets after funcionts?

If you **write content rifht after function, it will be passed as positional arguments there**.

> So *that* allows me to write *literally anything in things I pass to functions!*

Passing content, part II

So, just to make it clear, when I write

```
- #text(red)[red text]
- #text([red text], red)
- #text("red text", red)
//      ^        ^
// Quotes there mean a plain string, not a content!
// This is just text
```

It all will result in a red text

Basic styling

Set rule

That was great, but using functions everywhere, especially with many arguments every time is awfully cumbersome.

That's why Typst has *rules*. No, not for you, for the document

And the first rule we will consider there is `set` rule. As you see, I've just used it on `par` (which is short from paragraph) and now all paragraphs became *justified*.

It will apply to all paragraphs after the rule, but will work only in it's *scope* (we will discuss them later).

Of course you can override a `set` rule. This rule just sets the *default value* of an argument of an element.

By the way, at first line of this snuppet I've reduced page size to make justifyng more visible, also increasing margins to add blank space on left and right.

A bit about length units

Before we continue with rules, we should talk about length. There are several absolute length units int Typst:

Points	
Milimeters	
Centimeters	
Inches	
Relative to font size	

`1 em` = current font size.
It is a very convenient unit, so we are going to use it a lot

Setting something else

Of course, you can use `set` rule with all built-in functions and all their named arguments to make some argument "default".

For example, let's make all quoptes in this snippet authored by the book:

> Typst is great!
>
> — Typst Examples Book

> The problem with quotes on the internet is that it is hard to verify their authenticity.
>
> — Typst Examples Book

Opinionated defaults

That allows you to set Typst default styling as you want it:

- List item
- List item

1. Enum item
2. Enum item

Don't complain about bad defaults! `Set` your own.

Numbering

Some of elements have a property called "numbering". They accept so-called "numbering patterns" and are very useful width set rules. Let's see what I mean.

I: This is first level

II: Another first

II.1: Second

II.2: Another Second

II.2.1: Now third

II.3: And second again

III: Now returning to first

IV: These are actual romanian numerals

Of course, there are lots of other cool propierties that can be *set*, so feel free to dive into Official Reference and explore them!

And now we are moving into something much more interesting…