idwtet (I Don't Wanna Type Everything Twice)

To load this package in this repo, we have to:

```
#import "../idwtet.typ"
#show: idwtet.init
```

ouset package (v0.1.1)

Here we use the `typst-ex` codeblock. It evaluates the content as content, but still in local scope.

```
#import "@preview/ouset:0.1.1": ouset
$
"Expression 1" ouset(&, <==>, "Theorem 1") "Expression 2"\
               ouset(&, ==>,, "Theorem 7") "Expression 3"
$
```

$$\text{Expression 1} \overset{\text{Theorem 1}}{\iff} \text{Expression 2}$$

$$\overset{\text{Theorem 7}}{\implies} \text{Expression 3}$$

funarray package (v0.2.0)

Here we use the `typst-ex-code` codeblock. It evaluates the content as code, wraps the code appropiately and shows the return type.

```
import "@preview/funarray:0.2.0": chunks, partition-map

let numbers = (1, "not prime", 2, "prime", 3, "prime", 4, "not prime")

let (primes, non-primes) = partition-map(
    chunks(numbers, 2),      // transforms (a,b,c,d,e) to ((a,b), (c,d), (e,))
    x => x.at(1) == "prime", // partition criterion
    x => x.at(0)             // map of each group
)
primes
```

return type: `array`

```
(2, 3)
```

Other examples

`typst-ex-code` codeblocks, evaluate in code mode and display the return type:

```
let hello = 2
```

return type: `none`

```
[Wow]
```

return type: `content`

Wow

Also there are `typst-code` and `typst` (re-)defiend, here shown in order. Both only show typst code without executing. You can still use `typc` and `typ` for standard typst behaviour.

```
let hello = 2
```

```
let hello = 2
```