

idwtet (I Don't Wanna Type Everything Twice)

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

```
#import "../idwtet.typ"
```

Then we init the document and have to now define the variables to use in the package. Note, that import statements are not allowed in eval blocks, such that you have to use a workaround as given here.

```
#show: idwtet.init.with(eval-scope: (
  ouset: (
    value: {import "@preview/ouset:0.1.1": ouset; ouset},
    code: "#import \"@preview/ouset:0.1.1\": ouset"
  ),
  funarray: (
    value: {import "@preview/funarray:0.2.0"; funarray},
    code: "import \"@preview/funarray:0.2.0\""
  ),
))
```

ouset package (v0.1.1)

Here we use the `typst-ex` codeblock. It evaluates the content as content, but still in local scope. To display the import, we use the `%ouset%` as a placeholder, which will automatically be replaced as defined before. The variable `ouset` itself is defined, though.

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

$$\begin{array}{ccc} \text{Expression 1} & \overset{\text{Theorem 1}}{\iff} & \text{Expression 2} \\ & \underset{\text{Theorem 7}}{\implies} & \text{Expression 3} \end{array}$$

funarray package (v0.2.0)

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

```
import "@preview/funarray:0.2.0"

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

let (primes, non-primes) = funarray.partition-map(
  funarray.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
```

`(2, 3)` return type: array

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
```