Itertools Package

This package provides some convinient functional functions for typst to use on arrays. Let us define

```
a = (
  1,
  "not prime",
  2,
  "prime",
  3,
  "prime",
  4,
  "not prime",
  5,
  "prime",
)
```

chunks

The chunks function translates the array to an array of array. It groups the elements to chunks of a given size and collects them in an bigger array.

```
chunks(a, 2) = (
  (1, "not prime"),
  (2, "prime"),
  (3, "prime"),
  (4, "not prime"),
  (5, "prime"),
)
```

unzip

The unzip function is the inverse of the zip method, it transforms an array of pairs to a pair of vectors.

```
unzip(b) = (
  (1, 2, 3, 4, 5),
  (
    "not prime",
    "prime",
    "prime",
    "not prime",
    "prime",
  ),
)
```

cycle

The cycle function concatenates the array to itself until it has a given size.

```
let c = cycle(range(5), 8)
c = (0, 1, 2, 3, 4, 0, 1, 2)
```

Note that there is also the functionality to concatenate with + and * in typst.

windows and circular_windows

This function provides a running window

```
windows(c, 5) = (
  (0, 1, 2, 3, 4),
  (1, 2, 3, 4, 0),
  (2, 3, 4, 0, 1),
  (3, 4, 0, 1, 2),
)
```

whereas the circular version wraps over.

```
circular_windows(c, 5) = (
  (0, 1, 2, 3, 4),
  (1, 2, 3, 4, 0),
  (2, 3, 4, 0, 1),
  (3, 4, 0, 1, 2),
  (4, 0, 1, 2, 4),
  (0, 1, 2, 4, 0),
  (1, 2, 4, 0, 1),
  (2, 4, 0, 1, 2),
)
```

partition and partition_map

The partition function seperates the array in two according to a predicate function. The result is an array with all elements, where the predicate returned true followed by an array with all elements, where the predicate returned false.

```
let (primesp, nonprimesp) = partition(b, x => x.at(1) == "prime")
primesp = ((2, "prime"), (3, "prime"), (5, "prime"))

nonprimesp = ((1, "not prime"), (4, "not prime"))
```

There is also a partition_map function, which after partition also applies a second function on both collections.

```
let (primes, nonprimes) = partition_map(b, x => x.at(1) == "prime", x => x.at(0))
primes = (2, 3, 5)

nonprimes = (1, 4)
```

group_by

This functions groups according to a predicate into maximally sized chunks, where all elements have the same predicate value.

```
let f = (0,0,1,1,1,0,0,1)
let g = group_by(f, x => x == 0)
g = ((0, 0), (1, 1, 1), (0, 0), (1,))
```

flatten

Typst has a `flatten` method for arrays, however that method acts recursively. For instance

```
(((1,2,3), (2,3)), ((1,2,3), (1,2))).flatten() = (1, 2, 3, 2, 3, 1, 2, 3, 1, 2)
```

Normally, one would only have flattened one level. To do this, we can use the typst array concatenation method `+`, or by folding, the `sum` method for arrays:

```
(((1,2,3), (2,3)), ((1,2,3), (1,2))).sum() = ((1, 2, 3), (2, 3), (1, 2, 3), (1, 2))
```

intersperse

This function inserts item inbetween all elements of the array.

```
intersperse(f, 2) = (0, 2, 0, 2, 1, 2, 1, 2, 1, 2, 0, 2, 0, 2, 1)
```

Combined with flatten we can do this:

```
let h = intersperse(g, (0.25, 0.5, 0.75)).flatten()
h = (
  0,
  0,
  0.25,
  0.5,
  0.75,
  1,
  1,
  1,
  0.25,
  0.5,
  0.75,
  0,
  0,
  0.25,
  0.5,
  0.75,
  1,
)
```

take_while and skip_while

These functions do exactly as they say.

```
take_while(h, x => x < 1) = (0, 0, 0.25, 0.5, 0.75)
```

```
skip_while(h, x => x < 1) = (1, 1, 1, 0.25, 0.5, 0.75, 0, 0, 0.25, 0.5, 0.75, 1)
```

Unsafe functions

The core functions are defined in `funarray_unsafe.typ`. However, assertions are not there. Still, if being cautious, one can use the imported `funarray_unsafe` module in `funarray(.typ)`. All function names are the same.