wordometer

This manual contains 1299 words.

A small Typst package for quick and easy in-document word counts.

github.com/Jollywatt/typst-wordometer

Version 0.1.3

Basic usage

```
#import "@preview/wordometer:0.1.3": word-count, total-words

#show: word-count

In this document, there are #total-words words all up.

#word-count(total => [
  The number of words in this block is #total.words
  and there are #total.characters letters.
])
```

Excluding elements

You can exclude elements by function (e.g., `heading`, `table`, `figure.caption`), by where-selector (e.g., `raw.where(block: true)`), or by label (e.g., `<no-wc>`).

```
#show: word-count.with(exclude: (heading.where(level: 1), strike))

= This Heading Doesn't Count
== But I do!

In this document #strike[(excluding me)], there are #total-words words all up.

#word-count(total => [
  You can exclude elements by label, too.
  #[That was #total.words, excluding this sentence!] <no-wc>
], exclude: <no-wc>)
```

Details

The basic `word-count-of()` function accepts content and returns a dictionary of statistics. The main `word-count()` function wraps this in a more convenient interface, for use in a document show rule:

```
#show: word-count
```

...or for scoped word counts:

```
#word-count(total => [There are #total.words total words.])
```

The actual word counting works by using `extract-text()` to convert content to a plain string which is split up into words, sentences, and so on. You can specify exactly what is counted via the `counter` argument of most functions.

The `extract-text()` function uses `map-tree()` to traverse a content tree and accumulate the text from each leaf node. The `map-tree()` function (and most of the other functions) has an `exclude` parameter, which is used to exclude certain elements from the word count.

The following elements have no text content and are always excluded: `bibliography`, `cite`, `display`, `equation`, `h`, `hide`, `image`, `line`, `linebreak`, `locate`, `metadata`, `pagebreak`, `parbreak`, `path`, `polygon`, `ref`, `repeat`, `smartquote`, `space`, `style`, `update`, and `v`.

Where this doesn't work

Word counting with `extract-text()` occurs **before show rules are applied**, which means content modified by show rules may get counted differently to how it looks finally.

Ways to exclude elements

- By element:

`#word-count-of(exclude: strong)[One *not* two.].words`	2
`#word-count(exclude: (heading, highlight), total => [` `= Not me` `One two #highlight[me neither] three. \` `#total.words words including this line.` `])`	**Not me** One two me neither three. 8 words including this line.

- By label:

`#word-count-of(exclude: <not-me>, [` `One #[(not I)] <not-me> two three.` `])`	`(characters: 12, words: 3, sentences: 1)`
`#word-count(exclude: <not-me>, total => [` `One, two, three, four.` `#[That was #total.words words.] <not-me>` `])`	One, two, three, four. That was 4 words.

- By `where` selector:

`#word-count-of(exclude: raw.where(lang: "python"))[` ` ```python` `print("Hello, World!")` ` ``` ` `Only I count.` `]`	`(characters: 11, words: 3, sentences: 1)`

Functions

`word-count()`

Perform a word count on content.

Master function which accepts content (calling `word-count-global()`) or a callback function (calling `word-count-callback()`).

Parameters

```
word-count(
  arg: content fn,
  ..options:
) -> dictionary
```

arg `content` or `fn`

Can be:

- `content`: A word count is performed for the content and the results are accessible through `#total-words` and `#total-characters`. This uses a global state, so should only be used once in a document (e.g., via a document show rule: `#show: word-count`).
- `function`: A callback function accepting a dictionary of word count results and returning content to be word counted. For example:

  ```
  #word-count(total => [This sentence contains #total.characters letters.])
  ```

..options

Additional named arguments:

- `exclude`: Content to exclude from word count (see `map-tree()`). Can be an array of element functions, element function names, or labels.
- `counter`: A function that accepts a string and returns a dictionary of counts.
- `method`: Content traversal method to use (see `word-count-of()`).

`word-count-callback()`

Simultaneously take a word count of some content and insert it into that content.

It works by first passing in some dummy results to `fn`, performing a word count on the content returned, and finally returning the result of passing the word count retults to `fn`. This happens once — it doesn't keep looping until convergence or anything!

For example:

```
#word-count-callback(stats => [There are #stats.words words])
```

Parameters

```
word-count-callback(
  fn: function,
  ..options:
) -> content
```

fn `function`

A function accepting a dictionary and returning content to perform the word count on.

..options

Additional named arguments:
- `exclude`: Content to exclude from word count (see `map-tree()`). Can be an array of element functions, element function names, or labels.
- `counter`: A function that accepts a string and returns a dictionary of counts.
- `method`: Content traversal method to use (see `word-count-of()`).

`word-count-global()`

Get word count statistics of the given content and store the results in global state. Should only be used once in the document.

The results are accessible anywhere in the document with `#total-words` and `#total-characters`, which are shortcuts for the final values of states of the same name (e.g., `#locate(loc => state("total-words").final(loc))`)

Parameters

```
word-count-global(
  content: content,
  ..options:
) -> content
```

content `content`

Content to word count.

..options

Additional named arguments:
- `exclude`: Content to exclude from word count (see `map-tree()`). Can be an array of element functions, element function names, or labels.
- `counter`: A function that accepts a string and returns a dictionary of counts.
- `method`: Content traversal method to use (see `word-count-of()`).

`word-count-of()`

Get word count statistics of a content element.

Returns a results dictionary, not the content passed to it. (See `string-word-count()`).

Parameters

```
word-count-of(
  content: content,
  exclude: array,
  counter: fn,
  method: string
) -> dictionary
```

exclude `array`

Content to exclude from word count (see `map-tree()`). Can be an array of element functions, element function names, or labels.

Default: `()`

counter `fn`

A function that accepts a string and returns a dictionary of counts.

For example, to count vowels, you might do:

```
#word-count-of([ABCDEFG], counter: s => (
    vowels: lower(s).matches(regex("[aeiou]")).len(),
))
```

Default: `string-word-count`

method `string`

The algorithm to use. Can be:
- `"stringify"`: Convert the content into one big string, then perform the word count.
- `"bubble"`: Traverse the content tree performing word counts at each textual leaf node, then "bubble" the results back up (i.e., sum them).

Performance and results may vary by method!

Default: `"stringify"`

`concat-adjacent-text()`

Simplify an array of content by concatenating adjacent text elements.

Doesn't preserve content exactly; `smartquote`s are replaced with `'` or `"`. This is used on `sequence` elements because it improves word counts for cases like "Digby's", which should count as one word.

For example, the content

Qu'est-ce **que** c'est !?

is structured as:

```
(
  [Qu],
  smartquote(double: false),
  [est-ce],
  [ ],
  strong(body: [que]),
  [ ],
  [c],
  smartquote(double: false),
  [est],
  [ ],
  [!?],
)
```

This function simplifies this to:

```
([Qu'est-ce ], strong(body: [que]), [ c'est !?])
```

Parameters

```
concat-adjacent-text(children: array)
```

children `array`

Array of content to simplify.

`extract-text()`

Extract plain text from content

This is a quick-and-dirty conversion which does not preserve styling or layout and which may introduces superfluous spaces.

Parameters

```
extract-text(
  content: content,
  ..options:
)
```

content `content`

Content to extract plain text from.

..options

Additional named arguments:

- `exclude`: Content to exclude (see `map-tree()`). Can be an array of element functions, element function names, or labels.

`map-tree()`

Traverse a content tree and apply a function to textual leaf nodes.

Descends into elements until reaching a textual element (`text` or `raw`) and calls `f` on the contained text, returning a (nested) array of all the return values.

Parameters

```
map-tree(
  f: function,
  content: content,
  exclude: array
)
```

f `function`

Unary function to pass text to.

content `content`

Content element to traverse.

exclude `array`

Content to skip while traversing the tree, specified by:
- name, e.g., `"heading"`
- function, e.g., `heading`
- selector, e.g., `heading.where(level: 1)` (only basic `where` selectors are supported)
- label, e.g., `<no-wc>`

Default value includes equations and elements without child content or text: `"bibliography"`, `"cite"`, `"display"`, `"equation"`, `"h"`, `"hide"`, `"image"`, `"line"`, `"linebreak"`, `"locate"`, `"metadata"`, `"pagebreak"`, `"parbreak"`, `"path"`, `"polygon"`, `"ref"`, `"repeat"`, `"smartquote"`, `"space"`, `"style"`, `"update"`, and `"v"`.

To exclude figures, but include figure captions, pass the name `"figure-body"` (which is not a real element). To include figure bodies, but exclude their captions, pass the name `"caption"`.

Default: `IGNORED_ELEMENTS`

`string-word-count()`

Get a basic word count from a string.

Returns a dictionary with keys:
- `characters`: Number of non-whitespace characters.
- `words`: Number of words, defined by `regex("\b[\w'’]+\b")`.
- `sentences`: Number of sentences, defined by `regex("\w+\s*[.?!]")`.

Parameters

```
string-word-count(string: string) -> dictionary
```