Dickinson User Guide

Vanessa McHale

Contents

Introduction

Dickinson is a text-generation language for generative literature. Each time you run your code, you get back text. The text is chosen randomly based on your code.

Program Structure

Dickinson files begin with `%-`, followed by definitions.

Example

Here is a simple Dickinson program:

```
%-

(:def main
  (:oneof
    (| "heads")
    (| "tails")))
```

Save this as `gambling.dck`. Then:

```
emd run gambling.dck
```

which will display either `heads` or `tails`. The `:oneof` construct selects one of its branches with equal probability.

In general, when you `emd run` code, `emd` will display the result of evaluating `main`.

Definitions + Names

We can define names and reference them later:

```
%-

(:def gambling
  (:oneof
    (| "heads")
    (| "tails")))

(:def main
  gambling)
```

We can `emd run` this to the same results.

Interpolation

We can reference and recombine past definitions via string interpolation:

```
(:def adjective
  (:oneof
    (| "beautiful")
    (| "auspicious")
    (| "cold")))

(:def main
  "What a ${adjective}, ${adjective} day!")
```

REPL

Libraries

Dickinson allows pulling in definitions from other files with `:include`.

Using Libraries

Example

The `color` module is bundled by default:

```
(:include color)

%-

(:def main
  "Today's mood is ${color}")
```

The `:include` must come before the `%-`; definitions come after the `%-` as above.

`color.dck` contains:

```
%-

(:def color
    (:oneof
        (| "aubergine")
        (| "cerulean")
        (| "azure")
        ...
```

Writing Libraries

Examples