Over/under explores the connection between weaving and programming and the almost mystical connection between women and software writing, embedded deep in women's tradition of weaving not just threads, but networks.

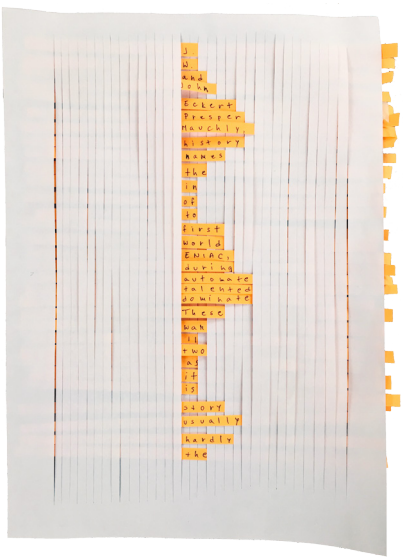
Following this thread [pun intended] through the history of computing, I discovered the Jacquard loom, invented in 1804 by Joseph Jacquard. His mechanical loom worked with a deck of punch cards which controlled the pattern which the machine executed. This was the first instance of a programmable machine. Later on, Charles Babbage's Analytical Machine was designed to be able to perform all four arithmetic operations, with the use of a similar punch card system. This technique was further developed for the US census in 1890 by Herman Hollerith, whose company later became IBM.

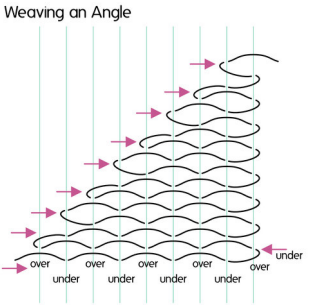
over/under

The connection between weaving and programming can be represented through an equation:

input + rules = output

In the case of weaving, the input is thread, the rules are the pattern, and the output is fabric. In the case of software, the input is data, the rules are the sequence, syntax, etc, and the output is data modified according to the rules. In order for this connection to be stronger, it is necessary that anyone who analyzes the output and knows the available rules can retrace all the steps that were taken in the process of weaving/programming.





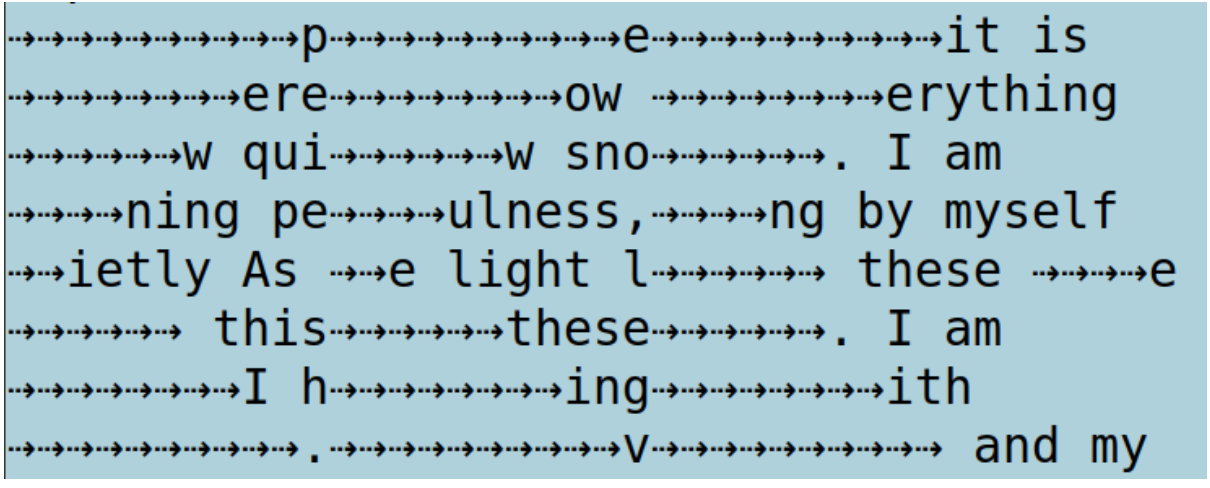
I wanted to be able to reveal the rules applied to the input through the output, by tracing back the instructions line by line. To do this in software form, the instructions had to be written in a language that was simple and straightforward, one that emulated weaving instructions as closely as possible.

From all this came the writing of an interpreted programming language in Python that could receive as rules basic weaving instructions ‘over’ and ‘under’ and apply a pattern onto text, simulating a plain weave. For the command ‘over’, the pattern would show the respective characters as they are. For the command ‘under’, the respective characters would be replaced with an unicode character. The code is adapted after Peter Norvig’s code on writing a Lisp interpreter in Python.

When running the program, an interpreter is opened, with the use of the repl function. After that, the program is ready to take commands. There are currently 8 commands for the over/under language.

- load — loads the input text
- show — shows the current line
- over + int(x) — keeps x characters intact
- under + int(x) — replaces x characters with a symbol
- pattern — generates the pattern
- save — saves the pattern in a .txt file
- enter/return — pressed twice, moves you to the next line
- quit — quits the program
- e.g. — over 10, under 9, over 8, under 7

Instructions


p e it is
ere ow erything
w qui w sno . I am
ning pe ulness, ng by myself
ietly As e light l these e
this these . I am
I h ing ith
. v and my

```
import linecache
import textwrap
import sys
from sys import exit

class LeavingProgram(Exception):
    pass

def parse(program):
    cmds = program.split(',')
    splitted_cmds = []
    for cmd in cmds:
        splitted = cmd.split()
        splitted_cmds.append(splitted)
    return splitted_cmds

def tokenize(s):
    return s.split()

def repl():
    while True:
        try:
            val = eval(parse(input('> ')))
            if val is not None:
                print(val)
        except LeavingProgram:
            break

text = None
line_number = 0
last_index = 0

def eval(cmds):
    global text
    global line_number
    global last_index
    global pattern

    for cmd in cmds:
        if cmd == []:
            line_number += 1
            last_index = 0

```

```
            elif cmd[0] == 'load':
                contents = open('text/yourfile.txt').read()
                text = textwrap.wrap(contents, 40, break_long_words=True)
                print('\n'.join(text))
                line_number = 0
                last_index = 0

            elif cmd[0] == 'show':
                print(text[line_number])

            elif cmd[0] == 'under':
                current_line = text[line_number]
                char_number = int(cmd[1]) - 1
                char_list = list(current_line)

                x=range(last_index, char_number + last_index + 1)
                for time in x:
                    if time < len(char_list):
                        char_list[time] = u'\u21e2'

                last_index += char_number + 1

                joined = ''.join(char_list)
                text[line_number] = joined

            elif cmd[0] == 'over':
                last_index += int(cmd[1])

            elif cmd[0] == 'pattern':

                pattern = text[0:line_number + 1]
                print('\n'.join(pattern))

            elif cmd[0] == 'save':
                pattern = text[0:line_number + 1]
                pattern_file = open('output/patternfile.txt', 'w')
                pattern_file.write('\n'.join(pattern))
                pattern_file.close()
                print('Your pattern has been saved in the output folder.')

            elif cmd[0] == 'quit':
                print('Come back soon!')
                raise LeavingProgram()
            else:
                joined = ' '.join(cmd)
                print('Did not understand command {}'.format(joined))

    if __name__ == '__main__':
        repl()
```

```
$ python3 overunder.py
> load
> over 1,  under 5, over 1
>
> over 2, under 4, over 2
>
> over 3, under 3, over 3
>
> over 4, under 2, over 4
>
> over 5, under 1, over 5
> save
> quit
```