Draft 2

```
def preprocess(source):

    final = ""
    string_type = ""
    escaped = False
    SSL_needed = False

    for char in source:
        if char in STRING_CHARS:
            if string_type == char:
                if escaped:
                    escaped = False
            else:
                string_type = NONE
        elif string_type == "":
            if escaped:
                escaped = False
        else:
            string_type = char

    else:

    elif char == "\":
        if escaped: Escaped = False
        else: escaped = True
```

The String uncompressor

Standard/spaceless - spaceful

`bottle¥s` ⇒ "bottle" + value of ¥s

`...` | e.g. `hello` `s1;s2;` `\n;`

↓ "hello" ↓

word @ s1
word @ s2

newline + ";"

Space after var or non-valid name

"`" {char | (scc ";")} "`"

char

scc = [char][char]

→ "`" ((\d | (. ";"))+) "`"

replace w/ ¶ for spaceful

"`" {<char> | (<scc> ";") | ("¥" <name> " ")} "`"

"`" {<char> | (<scc> ";") | ("¥" <name> <non-name-char>)} "`"

2nd look at Special compressed strings

Delimiter: «

~~Spaceless by~~

Spaceful by default

« s1 s2 |, « → s1, s2

« s1 s2 !, « → s1 s2,

| means that args are ~~processed~~ applied left to right

! means that args are applied right to left

Form:

« ...1 Δ: ...2 « | ...1 = sccs ...2 = seperators

: = | or !

1 2 3 4 5 6 7 8

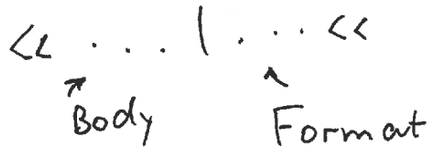
« s1 s2 |, ! → 8 byte "Helb World"

...1 The scc part

Design:

¥x s1 s2

« ¥x s1 s2 s3 s4 | ~~¥0~~



Body

↳ SCCs only. No vars or other chars

Format

↳ List of expressions

↳ keywords and $ are treated as standard chars

↳ only standard strings remain a keyword

↳ Escaping allowed

« s1 $2 | 'hi' » → s1 hi s2

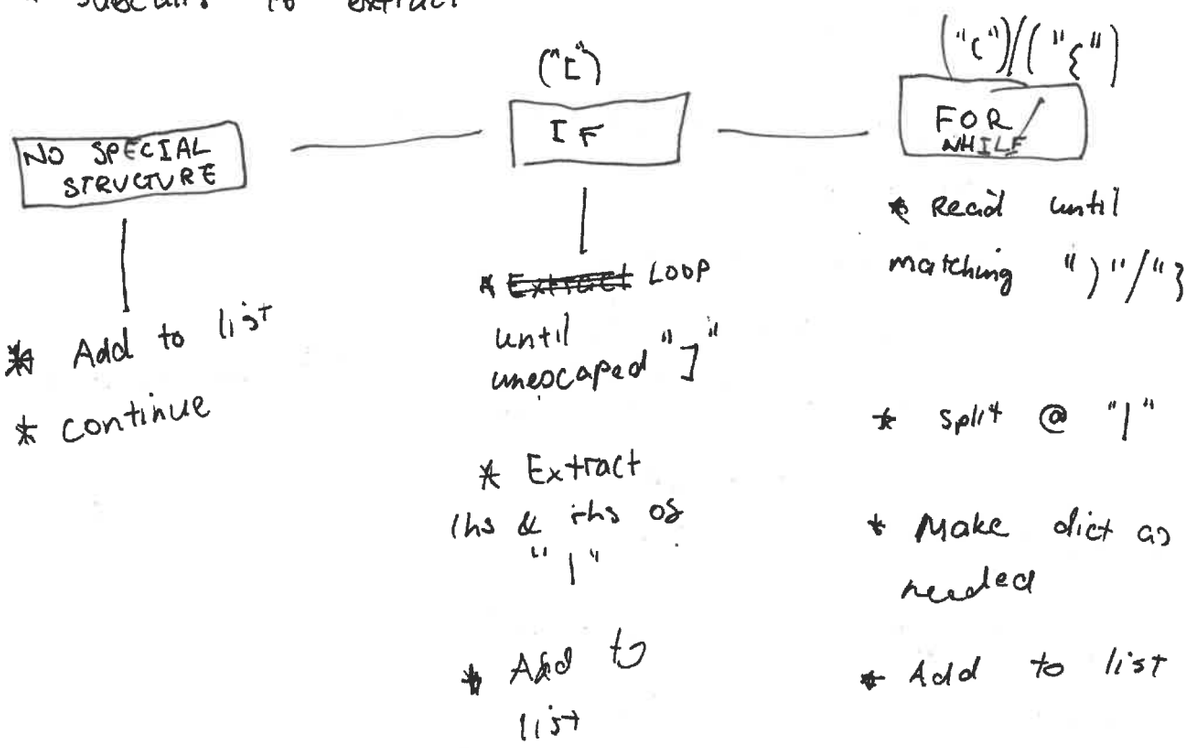
{

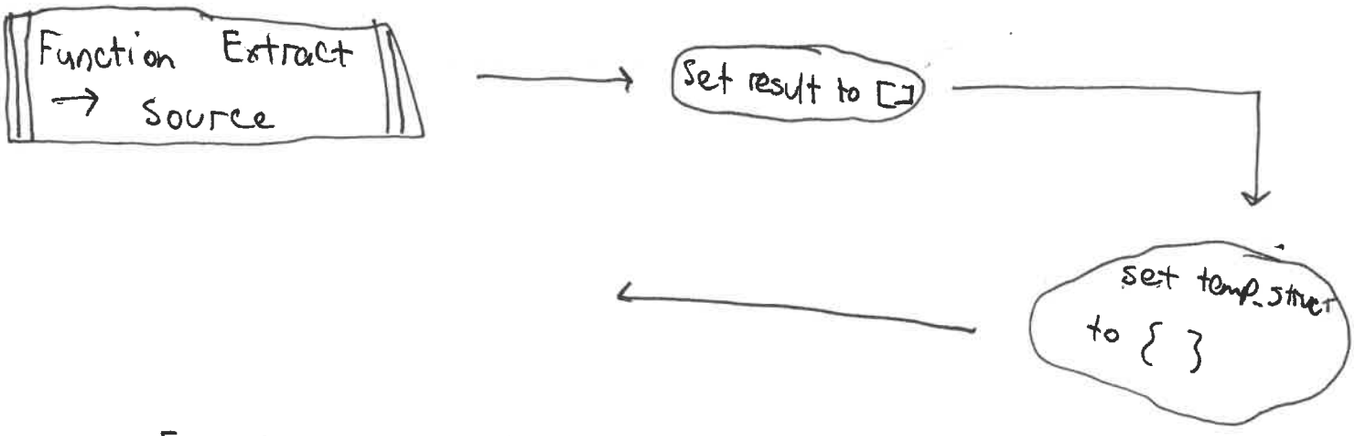
name: . . .

data: . . .

}

* list of structures
* Store current section of data
* temporary list of dictionaries
* data dictionary
* subcalls to extract






```
Function Extract(source)
    Result = [] ; escaped = False
    Temp_struct = { } ; temp = ""
    structures = []
    For char in source:
        If structures is empty:
            If char in "[{(@" and not escaped:
                structures.append(char)
            Else:
                Results.append({name: "cmd", data: char})
        Else, if char matches opening bracket at pos -1 in struct.
            Structures.pop( )
            If char == "]":
                lhs ; rhs =
```

S1 = "Hello world"

S1 = Hello S2 = world Goal: "Hello, world!"

Uncompressed:

`Hello, world!` → 15 (14 a)

Compressed, spaceless:

`S1;, S2;!` → 11 (10 a)

Compressed, spaceful:

Impossible (no punctuation)

Compressed, special, structure:

«S1S2|,! « 9 (8 a)

Compressed, special, constants

~~Z«S1S2~~

~~«S1;S2;«~~ «« S1;, S2;! «« 10 (9 a)

Spaces & SCC:

`abc;def;` → abc def

or

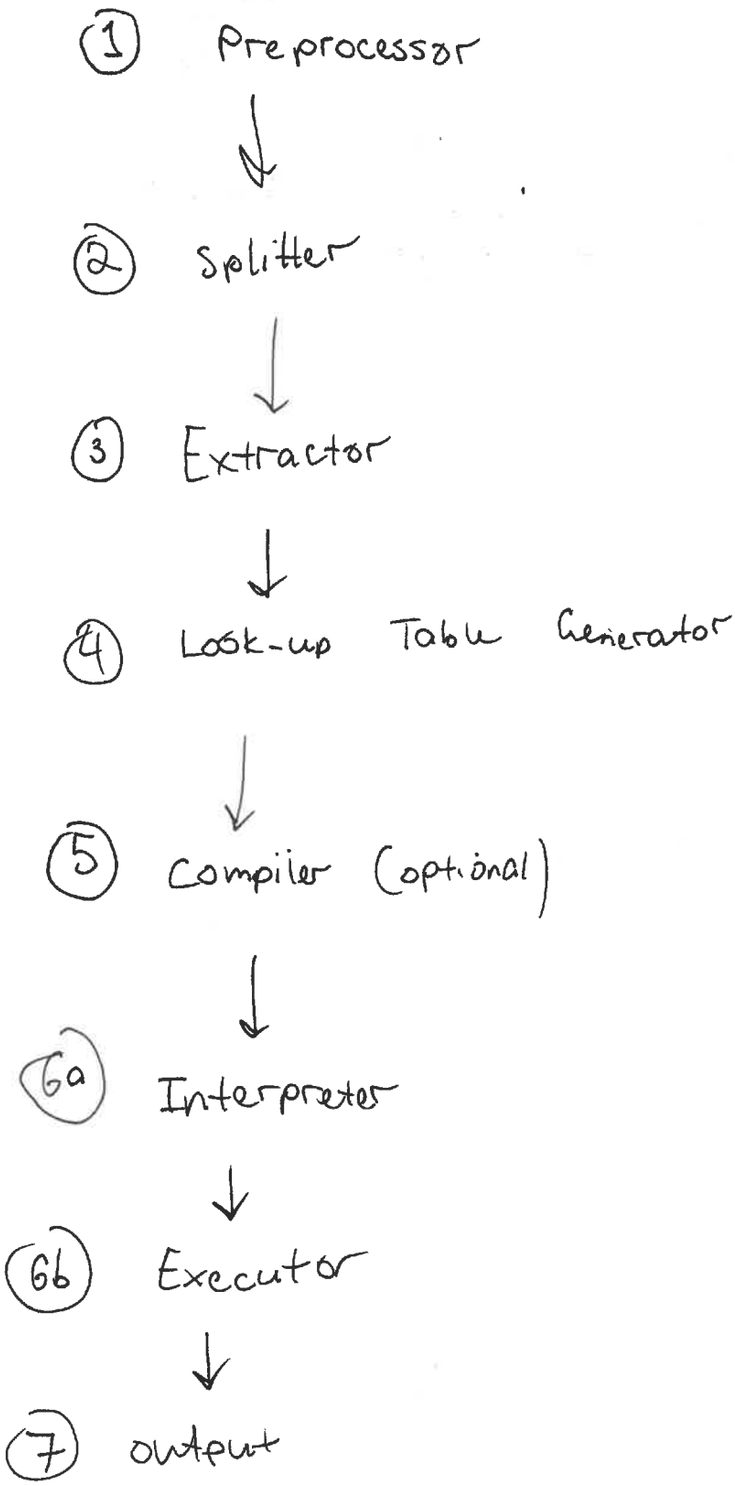
→ abc def

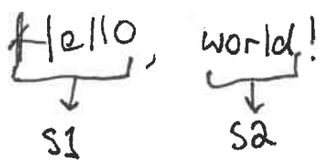
`abc;d` → abc d

or

→ abcd

7 Steps


1 Preprocessor
2 Splitter
3 Extractor
4 Look-up Table Generator
5 Compiler (optional)
6a Interpreter
6b Executor
7 output



`s1;, s2;!` → 11

9 s1;s2;9 → 8 (but no punctuation)

~~>>s1s2<<~~

~~>>s1s2|,!>>~~ → ~~10~~

<< s1 s2 |, ! → 8 chars!

Spaceful strings

¶ contents ¶

like normal spaceless strings, but spaces are placed between sccs:

Hello␣world!

`s1;␣s2;!`

¶s1;s2;!¶ ← hmm

→ 1 byte shorter

Spaceful Special Compressed Strings

Delimited by 

Usage:

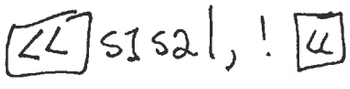
sccs | seperators

sccs: any series of sccs (no ;)

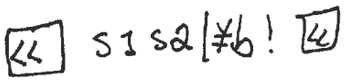
seperators: each seperator is either a single char, or a variable and is placed after the corresponding scc.

Examples

s1 = Hello s2 = World s3 = Bottles s4 = Bottle
¥a = s ¥b = , ¥c = " "

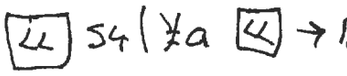


s1 s2 | , ! → Hello, World!



s1 s2 | ¥b ! → " " " "

s3 → Bottles



s4 | ¥a → Bottles

String types

`...` → standard / as-is / spaceless

`S1; S2;!` → S1, S2!

9...9 → standard / spaceful

`S1; S2;!` S1, S2 !

(...) → SCC only / spaceless

(S1 S2) → S1S2

(( )) → SCC only / spaceful

((S1 S2)) → S1 S2