Mike Hahn - hahnbytes@gmail.com

CoopleNet

CoopleNet is a tool used to make mobile social networks, built using a new programming language called JCoople. Each network (called a coople group) has a sponsor organization, which pays CoopleNet an annual fee for each member user. The user opens up the CoopleNet smartphone app and selects a coople group. The global coople group is everyone: all CoopleNet users. Coople groups display a scrolling feed containing posts, with a non-scrolling region at the top. All posts and the top panel are encoded in Joopletags, which is similar to HTML. Each sponsor organization can infinitely customize their coople group by writing JCoople and Joopletags code. Developers can release mobile apps written in JCoople (called j-capps), and premium users use those apps for free, without ads.

Business Model

Sponsor organizations which are registered charities pay no fees (other nonprofits pay no fees: must have website, domain and email containing domain name, not gmail). Other sponsor organizations pay $1/year/member. Lower fees apply to the 51st member and all subsequent members: $0.30/year/member. Sponsor organizations which incur a lot of data downloads/uploads or use a lot of disk space pay data hosting fees starting at $5/month.

The system keeps track of user tap counts and node-creation counts. Every time the user taps the screen, the node-creation count is added to a total T and reset to zero, and U is incremented. A node is a piece of smartphone memory which is 10 bytes long, and a typical user tap may result in hundreds of node-creation events. The node-creation count is converted into a percentile, ranked with all the other node-creation counts for that user in the current session. To calculate R, resources used per session, R = U(mean node-creation count in bottom quartile).

Premium users pick an amount to donate on an annual basis. Let D = sum of all premium user donations, p = no. of premium users, F = sum of all membership fees. When the premium user is deciding which amount to donate, a ratio Q is displayed:

- Q = (D / F)(R of for-profit sponsors) / (R of nonprofits)

Q equals 1 when the total amount of donations balances the ratio of resources used (for-profits to nonprofits), and dividing that by the gross revenue (membership fees). Q is less than 1 if there is a shortfall, and greater than 1 if there is a surplus.

A currency value d is also displayed:

- a = average donation = D / p
- d = a / Q = (F / p)(R of nonprofits) / (R of for-profit sponsors)
- Q' = 1 = (D' / F)(R of for-profit sponsors) / (R of nonprofits)
- D' = value plugged into Q formula needed to make Q = 1
- D' = D / Q

The value d represents what the average donation would be based on the same values in the Q formula using D' instead of D, in which the new value of Q = Q' = 1.

Premium users make donations of $5, $10, $20, or $50 per year. They can choose to donate up to 50 percent of their annual fee to a nonprofit coople group, or visit one or more coople groups and donate any desired amount. They can use any j-capp for free, without ads.

Joopletags Database

JCoople code can access master MySQL databases having one or more of the following fields: id, parentid, firstname, lastname, startingdate, endingdate. The parentid field is a foreign key. The detail database has the following fields: id, tableid, recordid, lineno, text. The text field is 100 chars. long and consists of Joopletags code, often organized in a format which emulates a different language called JSON. Each record in the master MySQL databases is associated with zero or more records in the detail database. Some built-in tables do not follow the above format.

App Notifications

When the user opens the CoopleNet app, a screen of network icons is displayed, each icon corresponding to a different coople group. To the left of each icon is a column of notification mini-shapes, in different colors. Single notifications are denoted with a circle, multiple notifications are denoted with a single square. The colors are specific to each coople group. Users have control over which notifications from which coople groups have high priority and which have low priority.

Backup Project

The backup project of CoopleNet is CoopleTeach, a tool used for teaching STEM subjects such as math and computers. In case CoopleNet never gets off the ground, will revert to JCoopiter (see CoopleTeach).

Implementation Steps

1. Develop foundation of JCoople code execution - ***almost done!***
2. Approach Progress Place
3. Develop rest of JCoople code execution
4. Release JCoople as console-based compiler on GitHub
5. Implement GUI: monospaced mode
6. Release JCoople/GUI on GitHub
7. Write Joopletags design specs
8. Develop Joopletags
9. Integrate JCoople with Joopletags
10. JCoople/Joopletags: JCoople RUn-time Environment (JCRUE)
11. JCRUE for Linux is open source
12. Integrate JCRUE with Android: JCoople Library
13. Develop basic CoopleNet coople group
14. Develop JCoople code editor
15. Expand code editor to JCoople SDK
16. Develop monetizing functionality
17. Perform beta testing using PP
18. Launch website
19. Purchase Google AdWords advertising
20. Convert JCRUE from Java to Swift
21. Port Android system to iOS
22. Implement Keyboard Aid (bells and whistles of editor)
23. Develop WYSIWYG Joopletags screen editor
24. Backup: develop CooopleTeach

Revenue and Expenses

- Let U = universe size
  - Twitter has 330 million monthly active users
  - U = population of US, UK, and Canada
  - Then U = 435 million
- Let M = member count of all coople groups
  - Then M = U x 0.46 percent
  - M = 2 million (to pick a round figure)
  - Assume that 80 percent of them belong in the first 50 members of a given coople group
  - Assume that 40 percent of the members are in for-profit coople groups
- Let F = sum of all membership fees
  - Then F = 2,000,000 x (0.8 + (0.2)(0.30)) x 0.4
  - F = 800,000 x (0.8 + 0.06)
  - F = 0.86 x 800,000
  - F = $688,000 gross
  - Net value of F = gross(0.85)
  - F' = Net value of F after App Store commission
  - F' = $688,000 x 0.85
  - F' = $585,000
- Let D = sum of all premium user donations
  - Assume that 1 percent of all members make an average yearly donation of $10
  - Then D = 2 million x 0.01 x 10
  - D = $200,000
- Let R = total revenue
  - Then R = 585,000 + 200,000
  - R = $785,000
- Let P = payroll expenses
  - Assume expenses of Google AdWords and web hosting eat up $85,000, and remainder of revenue goes to payroll expenses
  - P = $700,000
  - P' = 700,000 x 1.27
  - P' = C$889,000
- Let S = Average programmer salary
  - S = C$65,000
- Let N = no. of employees if they're all programmers
  - Then N = 889 / 65
  - N = 14

- Results:
  - U = universe size = 435 million
  - M = member count = 2 million
  - F = membership fees = $585,000
  - D = premium user donations = $200,000
  - R = revenue = $785,000
  - E = expenses non-payroll = $85,000
  - P = payroll = $700,000
  - S = average salary = C$65,000
  - N = no. of employees = 14

Pessimistic Scenario

- Let N = no. of employees
    - N = 5
    - Founder is 6th employee
    - Founder draws a salary of C$5000/year
- Let S = average salary
    - S = C$65,000
- Let P = payroll
    - Then P = (65,000 x 5 / 1.27) + (5000 / 1.27)
    - P = (65,000 x 5 + 5000) / 1.27
    - P = (325,000 + 5000) / 1.27 = 335,000 / 1.27
    - P = $264,000
- Let E = expenses non-payroll
    - E = $36,000
- Let R = revenue
    - R = P + E
    - R = 264,000 + 36,000
    - R = $300,000
- Let D = sum of all premium user donations
- Let F = Net value of total of membership fees, after paying App Store commission
    - Then R = D + F
    - R = $300,000
- Let D' and F' equal corresponding values in non-pessimistic scenario
    - Let D / F = D' / F' = 200,000 / 585,000
    - D / F = 1 / 2.925
    - Let F' = D' x 2.925
    - Also F = D x 2.925
    - R = D + D x 2.925
    - R = 300,000
    - D(1 + 2.925) = 300,000
    - D = 300,000 / 3.925
    - D = $76,433
    - F = 76,433 x 2.925
    - F = $223,567
- Let F' = gross value of F
    - F' = net value / 0.85
    - F' = 223,567 / 0.85
    - F' = $263,020
    - F' = sum of all membership fees
    - Also F' = M x (0.8 + (0.2)(0.30)) x 0.4
    - F' = M x 0.86 x 0.4
    - M = F' / (0.86 x 0.4)
    - M = 263,020 / 0.344
    - M = 764,593
- Let r = U / M
    - r = 435,000,000 / 764,593
    - r = 569
    - Therefore 1 out of every 569 people who live in the 3 main English speaking countries need to be members, assuming no other countries participate
- Let d = percentage of members who donate average of $10
    - Then D = 10Md
    - d = D / 10M
    - d = 76,433 / (10 x 764,593)
    - d = 7643.3 / 764,593
    - d = 1 percent

- Results:
    - d = percentage of members who donate = 1 percent
    - U = universe size = 435 million
    - r = fraction of universe population who are members = 1 / 569
    - M = member count = 764,593
    - F = membership fees = $223,567
    - D = premium user donations = $76,433
    - R = revenue = $300,000
    - E = expenses non-payroll = $36,000
    - P = payroll = $264,000
    - S = average salary = C$65,000
    - N = no. of employees = 5

Excessive Optimism

In case the previous 2 scenarios prove to be excessively optimistic, simply double or triple the membership fees, currently $1/year/member for first 50 members, and $0.30/year for subsequent members.

CoopleNet is Cool

Startup costs are $130,000 to hire 2 Java/Android programmers for one year. Most of the Android development work can be accomplished in that timeframe. I will hire those programmers after Step 13: Develop basic CoopleNet coople group. After launching the website, I can use revenue to fund iOS development. My savings are over $500,000 not including my condo, which is mortgage-free. I will get money from the government 3 years from now when I turn 65. If you google "toronto small business grants", you can view a couple of short lists of agencies which fund startups, so maybe I won't have to risk the entire $130,000.

CoopleNet is a for-profit company, but is partially dependent on the donations of the premium users. Donations aren't mandatory, but you get access to a lot of free apps which would otherwise cost you money. Many premium users will elect to donate directly to the nonprofits, and the coople groups are useful to the nonprofit sector, as well as to many for-profit companies. CoopleNet is not ad-supported, increasing quality of user experience.

Because CoopleNet is partially dependent on donations, big companies like Google and Facebook are unlikely to feel the need to acquire or duplicate CoopleNet. If CoopleNet is successful, my parents (dedicated to left wing causes) would be proud that their youngest child, late in life, will have made a difference in this world.

About Us

I am Mike Hahn, the founder of CoopleNet.com. I was previously employed at Brooklyn Computer Systems as a Delphi Programmer and a Technical Writer (I worked there between 1996 and 2013). At the end of 2014 I quit my job as a volunteer tutor at Fred Victor on Tuesday afternoons, where for 5 years I taught math, computers, and literacy, and became a volunteer math/computer tutor at West Neighbourhood House. I quit that job in mid-2019. I have a part-time job working for a perfume store. My hobbies are reading and I often go for walks. I don't read books very often, but on March 19, 2021 I started reading a biography of Steve Jobs which my brother gave me. I read the CBC news website, news/tech articles on my Flipboard app, and miscellaneous articles on my phone (same screen as my Google web page). I visit my brother once a month or more. For almost 30 years I was depressed on and off (I'm a rapid cycler), but it was greatly minimized after I ramped up development of my previous Aljegrid project in early March 2021.

JCoople

JCoople (implemented in Java) is an open source Python dialect in which all operators precede their operands, and parentheses are used for all grouping (except string literals, which are delimited with double quotes, also statements are separated by semicolons). JCoople source files have a .JOOP extension. Joopletags files (the sister language of JCoople, a text markup language) have a .JPTG extension. JCOOPLE (Java-Centric Object-Oriented Programming Language/Environment) boasts an ultra-simple Lisp-like syntax unlike all other languages.

Special Characters

( )	grouping
-	word separator
;	end of stmt.
:	dot operator
"	string delimiter
\	escape char.
#	comment
_	used in identifiers
$	string prefix char.
{ }	block comment

Op Characters

+ - * / %

= < >

& | ^ ~ ! ?

Differences from Python

- Parentheses, not whitespace
- Operators come before their operands
- Integration with Joopletags
- Information hiding (public/private)
- Single, not multiple inheritance
- Adds interfaces ("hedron" defs.)
- Drops iterators and generators
- Adds lambdas
- Adds quote and list-compile functions, treating code as data
- Adds cons, car and cdr functionality

Grammar Notation

- Non-terminal symbol: <symbol>
- Optional text in brackets: [*text*]
- Repeats zero or more times: [*text*]…
- Repeats one or more times: <symbol>…
- Pipe separates alternatives: *opt1* **|** *opt2*
- Comments in *italics*

Keyboard Aid

This optional feature enables hyphens, open parentheses, and close parentheses to be entered by typing semicolons, commas, and periods, respectively. When enabled, keyboard aid can be temporarily suppressed by using the Ctrl key in conjunction with typing semicolons, commas, and periods (no character substitution takes place). By convention, hyphens are used to separate words in multi-word identifiers, but semicolons are easier to type than hyphens. Similarly, commas and periods are easier to type than parentheses. Typing semicolon converts previous hyphen to a semicolon, and previous semicolon to a hyphen (use the Ctrl key to override this behaviour). Typing semicolon after close parenthesis simply inserts semicolon. Typing space after hyphen at end of identifier converts hyphen to underscore. The close delim switch automatically inserts a closing parenthesis/double quote when the open delimiter is inserted.

Joopletags

Joopletags is a simplified markup language used to replace HTML. Mock JSON files using Joopletags syntax have a .JPJS extension, and include no commas. Instead of myid: val, use [myid: val]. Instead of [1, 2, 3], use [arr: [: 1][: 2][: 3]]. Arbitrary Joopletags code can be embedded in the JCoople echo statement. Joopletags syntax, where asterisk (*) means occurs zero or more times, is defined as follows:

Tags:
- [tag]
- [tag (fld val)*: body]
- [tag (fld val)*| body |tag]

Body:
- text
- [(fld val)*: text]*

Call: (JCoople code)
- [expr: <expr>]
- [exec: <stmt>...]
- [joop: <path>]

JCoople Grammar

White space occurs between tokens (parentheses and semicolons need no adjacent white space):

<source file>:
- do ([<imp>]... [<def glb>] [<def>]... [<class>]...)

<imp>:
    <import stmt> **;**

<import stmt>:
    import <module>...
    from <rel module> import <mod list>
    from <rel module> import all

<module>:
    <name>
    (**:** <name><name>...)
    (as <name><name>)
    (as (**:** <name><name>...) <name>)

<mod list>:
    <id as>...

<id as>:
    <mod id>
    (as <mod id><name>)

<mod id>:
    <mod name>
    <class name>
    <func name>
    <var name>

<rel module>:
    (**:** [<num>] [<name>]...)
    <name>   *// ?*

<cls typ>:
    class
    iclass

<hedron>:
    hedron
    ihedron

<class>:
- <cls typ><name> [<base class>] [<does>] [<vars>] [<ivars>] do (<def>…) **;**
- abclass <name> [<base class>] [<does>] [<vars>] [<ivars>] do (<anydef>…) **;**
- <hedron><name> [<does>] [<const list>] do ([<abdef>]... [<defimp>]...) **;**
- enum <name><elist> **;**
- ienum <name><elist> **;**

<does>:
    (does <hedron name>...)

<hedron name>:
<base class>:
    <name>
    (**:** <name><name>…)

<const list>:
    (const <const pair>...)

<const pair>:
    (<name><const expr>)

<def glb>:
    gdefun [<vars>] [<ivars>] do <block> **;**

<def>:
- <defun> (<name> [<parms>]) [<vars>] [<gvars>] [<dec>] do <block> **;**

<defimp>:
- defimp (<name> [<parms>]) [<vars>] [<gvars>] [<dec>] do <block> **;**

<abdef>:
    abdefun (<name> [<parms>]) [<dec>] **;**

<defun>:
    defun
    idefun

<anydef>:
    <def>
    <abdef>

```
<vars>:
    ( var [<id>]... )

<ivars>:
    ( ivar [<id>]... )

<gvars>:
    ( gvar [<id>]... )

<parms>:
    [<id>]... [<parm>]... [ ( * <id> ) ] [ ( ** <id> ) ]

<parm>:
    ( <set op><id><const expr> )

<dec>:
    ( decor <dec expr>... )

<block>:
    ( [<stmt-semi>]… )

<stmt-semi>:
    <stmt> ;

<jump stmt>:
    <continue stmt>
    <break stmt>
    <return stmt>
    return <expr>
    <raise stmt>

<raise stmt>:
    raise [<expr> [ from <expr>] ]

<stmt>:
    <if stmt>
    <while stmt>
    <for stmt>
    <switch stmt>
    <try stmt>
    <asst stmt>
    <del stmt>
    <jump stmt>
    <call stmt>
    <print stmt>
    <bool stmt>

<call expr>:
•   ( <name> [<arg list>] )
•   ( : <colon expr>… <name> )
•   ( : <colon expr>… ( <method name>
    [<arg list>] ))
•   ( :: <colon expr>… <name> else <expr> )
•   ( :: <colon expr>… ( <method name>
    [<arg list>] ) else <expr> )
•   ( call <expr> [<arg list>] )

<call stmt>:
•   <name> [<arg list>]
•   : <colon expr>… ( <method name>
    [<arg list>] )
•   call <expr> [<arg list>]

<colon expr>:
    <name>
    ( <name> [<arg list>] )

<arg list>:
    [<expr>]... [ ( <set op><id><expr> ) ]...

<dec expr>:
    <name>
    ( <name><id>... )
    ( : <name><id>... )
    ( : <name>... ( <id>... ))

<dot op>:
    dot | :

<dotnull op>:
    dotnull | ::

<del stmt>:
    del <expr>

<set op>:
    set | =

<asst stmt>:
    <asst op><target expr><expr>
    <set op> ( tuple <target expr>... ) <expr>
    <inc op><name>

<asst op>:
    set | addset | minusset | mpyset | divset |
    idivset | modset |
    shlset | shrset | shruset |
    andbset | xorbset | orbset |
    andset | xorset | orset |
    = | += | -= | *= | /= |
    //= | %= |
    <<= | >>= | >>>= |
    &= | ^= | '|=' |
    &&= | ^^= | '||='

<target expr>:
    <name>
    ( : <colon expr>… <name> )
    ( slice <arr><expr> [<expr>] )
    ( slice <arr><expr> all )
    ( <crop><cons expr> )
<arr>:          // string or array/list
    <name>
    <expr>
```

<if stmt>:
- if <expr> do <block> [elif <expr> do <block>]… [else do <block>]

<while stmt>:
    while <expr> do <block>
    while do <block> until <expr>

<for stmt>:
- for <name> [<idx var>] in <expr> do <block>
- for (<bool stmt>**;** <bool stmt>**;** < bool stmt>) do <block>

<try stmt>:
- try do <block> <except clause>… [else do <block>] [eotry do <block>]
- try do <block> eotry do <block>

<except clause>:
    except <name> [as <name>] do <block>

<bool stmt>:
    quest [<expr>]
    ? [<expr>]
    <asst stmt>

<switch stmt>:
    switch <expr><case body> [else do <block>]

<case body>:
    [case <id> do <block>]...
    [case <dec int> do <block>]...
    [case <str lit> do <block>]...
    [case <tuple expr> do <block>]...

<return stmt>:
    return

<break stmt>:
    break

<continue stmt>:
    continue

<paren stmt>:
    (<stmt>)

<qblock>:
    (quote [<paren stmt>]...)

<quest>:
    quest | ?
<inc op>:
    incint | decint | ++ | --

<expr>:
    <keyword const>
    <literal>
    <name>
    (<unary op><expr>)
    (<bin op><expr><expr>)
    (<multi op><expr><expr>…)
    (<quest><expr><expr><expr>)
    <lambda>
    (quote <expr>...)
    <cons expr>
    <tuple expr>
    <list expr>
    <dict expr>
    <venum expr>
    <string expr>
    <bytes expr>
    <target expr>
    <call expr>
    <cast>

<unary op>:
    minus | notbitz | not |
    - | ~ | !

<bin op>:
    <arith op>
    <comparison op>
    <shift op>
    <bitwise op>
    <boolean op>

<arith op>:
    div | idiv | mod | mpy | add | minus |
    / | // | % | * | + | -

<comparison op>:
    ge | le | gt | lt | eq | ne | is | in |
    >= | <= | > | < | == | !=

<shift op>:
    shl | shr | shru |
    << | >> | >>>

Note: some operators delimited with single quotes for clarity (quotes omitted in source code)

<bitwise op>:
    andbitz | xorbitz | orbitz |
    & | ^ | '|'

<boolean op>:
    and | xor | or |
    && | ^^ | '||'

<multi op>:
    mpy | add | strdo | strcat |
    and | xor | andbitz | xorbitz |
    or | orbitz |
    * | + | % | + |

```
    && | ^^ | & | ^ |
    '||' | '|'

<const expr>:
    <literal>
    <keyword const>

<literal>:
    <num lit>
    <str lit>
    <bytes lit>

<cons expr>:
    ( cons <expr><expr> )
    ( <crop><expr> )
<tuple expr>:
    ( tuple [<expr>]… )
    ( <literal> [<expr>]… )
    ( )

<list expr>:
    ( jist [<expr>]… )

<dict expr>:
    ( dict [<pair>]… )

<pair>:
    // expr1 is a string
    ( : <expr1><expr2> )
    ( : <str lit><expr> )

<venum expr>:
    ( venum <enum name> [<elist>] )
    ( venum <enum name><idpair>... )

<elist>:
    <id>...
    <intpair>...
    <chpair>...

<intpair>
    // integer constant
    <int const>
    ( : <int const><int const> )
```

```
<chpair>
    // one-char. string
    <char lit>
    ( : <char lit><char lit> )

<idpair>
    <id>
    ( : <id><id> )
<cast>:
    ( cast <literal><expr> )
    ( cast <class name><expr> )

<print stmt>:     // built-in func
    print <expr>…
    println [<expr>]…
    echo <expr>…

<lambda>:
    ( lambda ( [<id>]... ) <expr> )
    ( lambda ( [<id>]... ) do <block> )
    ( lambdaq ( [<id>]... ) do <qblock> )
    // must pass qblock thru compile func
```

No white space allowed between tokens, for rest of JCoople Grammar

```
<white space>:
    <white token>...

<white token>:
    <white char>
    <line-comment>
    <blk-comment>

<line-comment>:
    # [<char>]... <new-line>

<blk-comment>:
    { [<char>]... }

<white char>:
    <space> | <tab> | <new-line>

<name>:
•   [<underscore>]… <letter> [<alnum>]…
    [<hyphen-alnum>]… [<underscore>]…

<hyphen-alnum>:
    <hyphen><alnum>…

<alnum>:
    <letter>
    <digit>
```

In plain English, names begin and end with zero or more underscores. In between is a letter followed by zero or more alphanumeric characters. Names may also contain hyphens, where each hyphen is preceded and succeeded by an alphanumeric character.

```
<num lit>:
    <dec int>
    <long int>
    <oct int>
    <hex int>
    <bin int>
    <float>

<dec int>:
    [<hyphen>] 0
    [<hyphen>] <any digit except 0> [<digit>]…

<long int>:
    <dec int> L

<float>:
    <dec int><fraction> [<exponent>]
    <dec int><exponent>

<fraction>:
    <dot> [<digit>]…

<exponent>:
    <e> [<sign>] <digit>…

<e>:
    e | E

<sign>:
    + | -

<keyword const>:
    null
    true
    false

<oct int>:
    0o <octal digit>…

<hex int>:
    0x <hex digit>…
    0X <hex digit>…

<bin int>:
    0b <zero or one>…
    0B <zero or one>…

<octal digit>:
    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
<hex digit>:
    <digit>
    A | B | C | D | E | F
    a | b | c | d | e | f

<str lit>:
    " [<str item>]... "

<str item>:
    <str char>
    <escaped str char>
    <str newline>

<str char>:
    any source char. except "\", newline, or
    end quote

<str newline>:
    \ <newline> [<white space>] "

<escaped char>:
    \\    backslash
    \"    double quote
    \}    close brace
    \a    bell
    \b    backspace
    \f    formfeed
    \n    new line
    \r    carriage return
    \t    tab
    \v    vertical tab
    \ooo      octal value = ooo
    \xhh      hex value = hh

<escaped str char>:
    <escaped char>
    \N{name}    Unicode char. = name
    \uxxxx      hex value (16-bit) = xxxx

<crop>:
    c <crmid>... r

<crmid>:
    a | d
```

Not implemented: string prefix and bytes data type (rest of grammar)

```
<str lit>:
    [ $ <str prefix>] <quoted str>

<str prefix>:
    r | R

<quoted str>:
    " [<str item>]... "
<bytes lit>:
    $ <byte prefix><quoted bytes>

<byte prefix>:   // any case/order
```

```
    b | br

<quoted bytes>:
    " [<bytes item>]... "

<bytes item>:
    <bytes char>
    <escaped char>
    <str newline>

<bytes char>:
    any ASCII char. except "\", newline, or
    end quote
```