Mike Hahn - hahnbytes@gmail.com - HELP

Parthonet

Parthonet.com is a client-server application. The client uploads code written in a text markup language called Parthotags, and the server downloads Parthotags code along with code written in a programming language called Parthonyte to the client. When in peer-to-peer mode: the host consists of a user which can upload Parthotags and Parthonyte code to the server, which is downloaded to the client (the host and the client communicate using a split screen).

Apps in client-server mode can be written in Parthonyte and/or Java. The server hosts Java code translated from Parthonyte, along with a stripped down version of the Parthonyte runtime, enabling it to run Parthonyte code.

Server Pricing

App authors hosted with Kamatera, Hostinger, or other Java hosting providers pay a Parthonet hosting fee of $10/year plus roughly $5/month to the Java hosting provider. They can charge arbitrary fees, including subscription fees and in-app purchases, to converted users. They get to keep 75 percent of ad revenue. App authors hosted with Parthonet pay a base rate of $20/year plus a surcharge for high bandwidth apps. They can only charge a one-time fee to converted users. They get to keep 25 percent of ad revenue.

Client Pricing

Converted users pay a subscription fee of $10/year. Other users (called advertoozers) pay no fees, but are exposed to ads. Only converted users are allowed to pay fees to app authors. Clients in peer-to-peer mode who are advertoozers are limited to one 24-hour ad-free period per week for each host they interact with. Hosts in peer-to-peer mode pay $20/year.

Phase 1 and 2

Phase 1 supports Windows, Mac, and Linux. Phase 2 supports Android and iOS. The flagship app in Phase 2 includes social network functionality similar to Facebook. Members of Progress Place and other clubhouses use the social network app without ads. Progress Place is a clubhouse for consumer/survivors located in Toronto.

Mandate

Parthonet users are encouraged to pay the subscription fee since Parthonet profits help to support online communities of Progress Place members and other consumer/survivors. Those profits also help organizations such as West Neighbourhood House connect free math and literacy tutors with learners who can't afford paid tutors. Please click on Math/Revenue to learn more about tutor/learner support and sources of revenue, respectively.

Parthoscreen

The Parthoscreen is used for teaching math, and is written in Java and Parthonyte. The hosts are the math tutors and the clients are the math students. A simplified version of the Parthoscreen is used for teaching coding and other subjects. The Parthoscreen can also be used for zoom-style meetings, utilizing speech-to-text instead of video.

Monospaced Mode

Only monospaced mode is supported by the Parthoscreen. All characters in a given panel are the same size, and adjacent cells in a given panel may be merged to form a subpanel. Panels and subpanels can contain a widget, graphic, or a block of text. Different panels containing text need not share the same font size. Subscripts and superscripts are offset vertically by half the height of a character cell. Lines of text can optionally be separated by a gap equal to half of a character cell. The canvas property of a panel supports drawing graphics such as lines, circles, and rectangles. Panels with canvases do not contain a character grid.

Split Screen

The top half of the Parthoscreen is used by the tutor/chair, and the bottom half of the screen is used by the student/participant. Both users can highlight/copy text on the other user's halfscreen and paste it on their own screen. Both halfscreens can scroll vertically, so they can contain dozens of lines if needed. Chat window: always on top, yellow background, variable width font. Audio is provided by cellphones on speaker phone. Only monospaced mode is supported: all characters within a given panel are the same size. Speech-to-text is utilized in zoom-style meetings.

Conversation Mode

Clicking on a message in the top half of the Parthoscreen enters conversation mode, similar to a normal chat window with messages from all participants in chronological order. Clicking on a message in conversation mode enters split screen mode. The bottom half of the screen is the current user if the message clicked on belongs to the current user or the tutor/chair, otherwise it is a different user. When in split screen mode, if the bottom half of the screen belongs to a user other than the current user, then clicking on a message in the top half of the Parthoscreen does not enter conversation mode. Instead the bottom half of the screen reverts to the current user.

Parthoteach

Parthoteach.org is a website linking free math, coding, and literacy tutors with learners who can't afford paid tutors. The tutors and the learners use the Parthoscreen to communicate with one another. Parthoteach will approach West Neighbourhood House and ask them if their tutors and learners are willing to become Parthoteach users, or at least beta testers.

Math Commands

- Use the arrow keys to move the cursor.
- Type underscore(s) to underline the numerator of a fraction.
- Use the special character command (Ctrl+K) to insert special characters such as pi, square root, sum, and integral.
- Use Ctrl+Tab/Shift+Tab to display/undo the next step in the math problem being solved.
- Type question mark (?) to explain the current step or to break the current step down into lower-level steps.
- Click on Help after typing question mark to access the help system.

Miscellaneous Commands

- Use asterisk and slash for multiply and divide.
- Fractions or matrices enclosed in brackets use tall brackets.
- Mixed numbers (example: three and a half) employ a vertical offset of half a character cell for the whole number.
- Smart down/up arrow: press it after inserting a character moves the cursor beneath/above that character.
- Functions such as lines and parabolas can be plotted interactively on a graph.
- The default-to-upper-case setting assumes that all letters entered are upper case (use the shift key to enter a lower case letter), so Caps Lock is redundant.

Expression Language

Mathematical expressions are encoded (internally) using a Parthonyte-style expression. Each step in the math problem being solved manipulates this expression. Even if the user enters steps in a different order than the default ordering, the simplification logic can handle that. The user can type Ctrl+Tab/Shift+Tab to redo/undo her previous step, as well as to redo/undo the computer's previous step.

Superscripts

Superscripts and subscripts are handled by employing a vertical offset of half a line per level of superscripting or subscripting. The caret symbol (^) is used as a superscript prefix, double-caret (^^) is used as a subscript prefix, and backslash (\) is used as an escape character (terminate super/subscript with a semicolon). Carets and double-carets cannot be mixed (exception: one level of superscript can be combined with one level of subscript).

Advanced Parthoscreen Commands

These next 2 paragraphs may be ignored (they are written in computerese). Use Shift+Arrow Key to highlight a rectangular block. Press Insert to insert a row or column of spaces before a highlighted block (insert blank line if no highlight). Press Shift+Insert/Delete to insert/delete an entire row/column when a block is highlighted. The next paragraph discusses commands handling multiple indent levels.

Press Enter at end of a line of text: insert blank line, back up on that line to line up with beginning of text on previous line. Press Enter on blank line to back up to line up with beginning of text on a previous line, or insert blank line if already at beginning of line. Press Tab to move forward to line up with beginning of first or next word on a previous line. Press Home to move to beginning of text on current line, press it again to toggle between beginning of line and beginning of text. The user doesn't have to memorize these commands: type question mark at any time to access the help system.

Keyboard Aid

The close delim switch of the Parthonyte code editor enables the automatic insertion of a closing parenthesis, brace, or double quote whenever the open delimiter is inserted. The optional keyboard-aid 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.

Steps

1. Develop foundation of Parthonyte code execution - ***done!***
2. Develop rest of Parthonyte code execution: WCNMIL
    1. Wrap up core foundation features
    2. Classes and objects
    3. Non-scalar data types
    4. Modules
    5. Inheritance + Interfaces (hedrons)
    6. Library
3. Release Parthonyte as console-based compiler on GitHub
4. Begin recruiting contributors
5. Write Parthotags design specs
6. Develop Parthotags
7. Integrate Parthonyte with Parthotags
8. PYRE: Parthonyte Runtime Environment (open source)
9. Develop Parthonyte code editor
10. Expand code editor to Parthonyte SDK
11. Phase 1: Windows, Mac, and Linux
12. Client-server:
    1. SPYRE: Server-side ParthonYte Runtime Environment
    2. SPYRE is closed source, stripped down, may be multi-user
13. Develop Parthonyte-to-Java converter
14. Develop Parthoscreen
15. Develop monetizing functionality
16. Perform beta testing using West Neighbourhood House
17. Launch website
18. Purchase Google AdWords advertising
19. Lower priority features:
    1. Implement Keyboard Aid (bells and whistles of editor)
    2. Develop WYSIWYG Parthotags screen editor
    3. Variable-width fonts
20. Phase 2: Android and iOS
21. Hire Watcoder: Waterloo Co-op Student
22. Watcoder to port PYRE to Android
23. Meanwhile, pitch project idea to DMZ tech incubator
24. Search for angel investor
25. If angel investor is found, then Watcoder becomes co-founder
26. Else:
    1. Watcoder is laid off
    2. Swift programmer wages are paid using my own money
27. Hire Swift programmer
28. Convert PYRE to Swift
29. Port PYRE to iOS
30. Add speech-to-text to Parthoscreen
31. Develop Psyberhood: social network app
32. Members of Progress Place use system for free
33. Partner with West Neighbourhood House
34. Make SPYRE open source in case Parthonyte goes out of business

Java Converter

Converts Parthonyte code to server-side Java code. Parthonyte variable declarations are handled by using variable names beginning with an uppercase letter preceded by a type character: b for boolean, i for int, j for long, c for char, f for double, and s for string. Object variables followed by comment beginning with the class name.

Revenue

Making 3 assumptions: 1) assume that 1500 host users and 15,000 converted client users exist, and host users are split evenly 3 ways: peer-to-peer, client-server hosted by Parthonet, client-server hosted elsewhere; 2) assume that average host user in client-server mode, hosted by Parthonet, pays $50/year; 3) assume that ad revenue from advertoozers equals half of revenue from converted users. Then annual revenue equals 15000(10)(1.5) + 500(20) + 500(50) + 500(10) = 225,000 + 10,000 + 25,000 + 5000 = $265,000.

To calculate the amount of annual revenue per 100,000 converted users, including host users, multiply 265,000 by 100,000 divided by (15,000 + 1500) = $1.6 million.

Surcharge for Heavy Users

Host users in client-server mode, when hosted by Parthonet, pay a minumum hosting fee (tier zero) of $20/year. Those with high-bandwidth requirements pay $5/month in the lowest data usage tier, and $20/month in the highest tier. So annual charges are always either $20 (tier zero) or between $60 and $240. The annual fee of $20 is paid up front. After 30 days, the appropriate data usage tier is determined, which is recalculated every 3 months.

Data Usage

Data usage is determined by amount of data downloaded/uploaded, and server memory usage. Server memory usage is in proportion to the number of 10-byte nodes allocated by server-side Parthonyte code. Usually the Java version of the server-side code is executed, but randomly and up to 5 percent of the time, the Parthonyte version of the server-side code is executed.

Psyberhood

Psyberhood.org is an online community of consumer/survivors. In case Progress Place becomes a partner organization, most users will be clubhouse members. Parthoteach.org is a website linking free tutors with learners who can't afford paid tutors.

Partners

Psyberhood hopes to partner with 2 organizations: Progress Place and West Neighbourhood House. These are arms-length partnerships. Once every 3 months a link to Psyberhood: psyberhood.org is posted in the Pipeline newsletter of Progress Place. Alternatively, a desktop shortcut to that URL will be included on all the public computers. All West Neighbourhood House tutors and learners will be given a different link: parthoteach.org. When users follow those links, they are given instructions on how they can participate in 2 online communities, for Progress Place and West Neighbourhood House. Partner organizations and their clients use system for free. The launching of Psyberhood.org and Parthoteach.org will take place even if no partner organizations are lined up at that time.

Progress Place

Members of the online Progress Place community can submit posts and comments similar to a Facebook group. Paid moderators are employed by Psyberhood. Eventually online communities will be formed for clubhouses in other cities. Different mental health diagnoses, such as schizophrenia, bipolar, and depression, will have their own online communities. In case Progress Place is not interested in becoming a partner organization, Mike will seek permission to write an article in their newsletter, recruiting members who wish to become involved in founding Psyberhood.org.

West Neighbourhood House

Math and literacy tutors are connected with learners on the Parthonet server. All Parthonet users must first install the Parthonet client on their computer or smartphone. The Parthoscreen is used to facilitate math lessons given by the tutor to the learner. A simplified version of the Parthoscreen is used to facilitate literacy lessons.

Zoom Meetings

Members of the Progress Place and West Neighbourhood House online communities can get together using zoom-style meetings enabled by the Parthoscreen. Speech-to-text is utilized instead of video.

Logo

Parentheses + Mountain + Pi = parenthesized version of the Python programming language, which is named after Monty Python = Mont(y) + Py(thon).

Glossary

Parthonet	Project name
Parthonyte	New programming language, similar to Python
Parthotags	Text markup language, similar to HTML
Parthoscreen	Whiteboard used for teaching math
Parthoteach	Dot-org website linking free tutors with learners
Psyberhood	Online community of consumer/survivors
PYRE	Parthonyte Runtime Environment
SPYRE	Server-side ParthonYte Runtime Environment

.PTHY	Parthonyte source file
.POTG	Parthotags file
.POJS	Parthotags/JSON file
.PTHX	Parthonyte compiled unit

About Us

I am Mike Hahn, the founder of Parthonet.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 science/tech articles (under Google) on my phone. I visit my brother about once a month.

Contact Info

Mike Hahn
Founder
Parthonet.com
2495 Dundas St. West
Ste. 515
Toronto, ON M6P 1X4

Phone: 416-533-4417
Email: hahnbytes (AT) gmail (DOT) com
Web: treenimation.net/hahnbytes/
Country: Canada

Parthonyte

Parthonyte (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). Parthonyte source files have a .PTHY extension. Parthotags is a text markup language, with a .POTG extension. Parthonyte boasts an ultra-simple Lisp-like syntax unlike all other languages.

Special Characters

Core:

- `()` grouping
- `-` word separator
- `;` end of stmt.
- `:` dot operator
- `"` string delimiter
- `\` escape char.

Operators:

- + - * / %
- = < >
- & | ^ ~ ! ?

Other:

- `# comment`
- `{}` block comment
- `_` used in identifiers
- `$` string prefix char.

Differences from Python

- Parentheses, not whitespace
- Operators come before their operands
- Integration with Parthotags
- 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

Parthotags

Parthotags is a simplified markup language used to replace HTML. Mock JSON files using Parthotags syntax have a .POJS extension, and include no commas. Instead of myid: val, use [myid: val]. Instead of [1, 2, 3], use [arr: [: 1][: 2][: 3]]. Arbitrary Parthotags code can be embedded in the Parthonyte echo statement. Parthotags 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]*

Parthonyte call:

- [expr: <expr>]
- [exec: <stmt>...]
- [pthy: <path>]

Note: for fld = style, corresponding val = (fld val)*

Benefits of Parthonyte

Parthonyte is simpler than any other object-oriented programming language, and integrates nicely with Parthotags, which is simpler than HTML. Operators come before their operands, also statements are semicolon-separated, so they never start with a parenthesis. Parthonyte is written in Java, so apps can be written in one or both of the Java and Parthonyte programming languages. Parthotags is simpler than HTML, based on nested rows and columns. A row cell is divided into multiple variable-width column cells, and a column cell is divided into multiple variable-height row cells.

MORE: Please click on More to access miscellaneous Parthonyte documentation. Only the first 2 paragraphs of the More web page contain up-to-date info, the rest of that web page is obsolete.

Parthonyte Grammar

At least one occurrence of a white space character (or a comment block), open parenthesis, close parenthesis, or semicolon occurs between adjacent tokens. A comment block consists of a pair of brace brackets enclosing zero or more characters.

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*

```
<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>]...

<swix expr>:
    (swix <expr><swix body> [else <expr>])

<swix body>:
    [(case <id><expr>)]...
    [(case <dec int><expr>)]...
    [(case <str lit><expr>)]...
    [(case <tuple expr><expr>)]...

<return stmt>:
    return

<break stmt>:
    break

<continue stmt>:
    continue

<paren stmt>:
    (<stmt>)

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

<quest>:
    quest | ?

<cquest>:
    cquest | ??

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

<expr>:
    <keyword const>
    <literal>
    <name>
    (<unary op><expr>)
    (<bin op><expr><expr>)
    (<multi op><expr><expr>…)
    (<quest><expr><expr><expr>)
    (<cquest> [(case <expr><expr>)]...)
    <swix 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>:
      ( lyst [<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 Parthonyte 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*
\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
```