Mike Hahn – hahnbytes@gmail.com

Jysta Hub

Jysta.org is the hub website of 6 projects: Jovelyst, Qpicary, Sites, Lystplayer, Jovelearn, and Psyvaspace. Jovelyst is an open source web programming language. Qpicary.com is a tool used to organize and share your image folders. Lystplayer.com is a website where you can play 2-player non-animated games. Jovelearn.com is a website which links tutors with students. Psyvaspace.org is an online community of consumer/survivors. Sites enables users to create websites written using Jovelyst and Lystagger (a simplified version of HTML). Jysta members pay one low subscription fee of $10 per year. Jysta users who are non-members (called no-name users) do not pay any fees, but have access to less functionality than members. Pysvaspace members (who are clients of the partner organizations) pay no fees, and receive a free Jysta membership. Jovelyst, as well as core Sites functionality, is written in Java, whereas the other 4 projects are written in Jovelyst and Lystagger.

Mandate

The mandate of Jysta.org is to raise money for organizations which serve consumer/survivors, and to fund the Psyvaspace online community of consumer/survivors. The source of the money raised comes from the subscription fees, plus fees charged to Sites web designers, and 10 percent of the fees Jovelearn tutors charge their students. That money is distributed to the Psyvaspace partner organizations in proportion to the number of Psyvaspace member logins per month for each partner organization.

Qpicary

Qpicary is a tool which lets you organize and share your image folders. It makes use of 2 open source third-party tools: an embedded web server called Jetty, and a text search engine called Lucene. Users must first launch the Qpicary web launcher and then point their web browsers to http://localhost:6886/. All image files are stored on the user's local hard drive. Qpicary is bundled with and implemented using an open source web programming language called Jovelyst, along with a markup language (simplified HTML, also open source) called Lystagger. For efficiency, the text search engine capabilities of Qpicary (interfacing with Lucene) are written in Java. Qpicary rhymes with military: CUE-puh-carry.

Qpic Folders

A qpic is a queue of image files contained in a folder, and qpic folders can contain other qpic folders recursively. Newly added image files go to the head of the queue in each qpic. Every qpic contains a special queue which is a subset of the main queue of images. Both the main and the special queues support image reordering commands: head, tail, move left, move right. The head of the queue is the leftmost image in the queue. Every qpic has 2 yes/no flags: the H-flag and the X-flag. H stands for hidden (not shared publicly) and the X-flag warns users that its images are such that if the user is at work or sitting at a public computer, then proceed with caution. By default, qpics having an X-flag value of yes are hidden from the user.

Business Model

Qpicary no-name users pay no fees. Qpicary members pay a subscription fee of $10 per year. All users can share qpics with other users by emailing links. All users can perform text-based searches, based on image captions, slide panels, and qpic names/descriptions. All users can browse non-hidden qpics of members without restrictions. All images contained in qpics of no-name users can be browsed, but any sub-qpics of a given no-name user's qpic are not displayed. All search results returned belonging to no-name users are stripped of all text: image captions, qpic names/descriptions, and user names. All of that same text information, including user names, is not displayed when browsing images in any qpic of a no-name user. All users can create static and dynamic web content using Jovelyst and Lystagger, and that code (unless created by a no-name user) can be downloaded by all users.

Commands

- **Enter** - down a level
- **Up Arrow** - up a level
- **Left/Right Arrow** - previous/next
  - qpic/screen/image/slide
- **Down Arrow** - toggle main/special
- **Ctrl+Down Arrow** - toggle slide mode
- **1** - first
- **0** - last
- **Shift+Up Arrow** - move to top
- **Shift+Left Arrow** - move left
- **Shift+Right Arrow** - move right
- **Shift+Down Arrow** - move to bottom
- **L** - like image: make it special
- **U** - undo L command
- **I** - insert image/qpic
- **D** - delete (slide mode)
- **Ctrl+D** - delete image/qpic
- **Ctrl+X** - cut qpic
- **Ctrl+C** - copy qpic
- **Ctrl+V** - paste qpic
- **Ctrl+N** - new qpic
- **Ctrl+R** - rename qpic
- **J** - justify (left, center, right)
- **B** - bookmark qpic/access bookmarks
- **H** - hidden qpic on/off
- **X** - X-flag qpic on/off
- **Y** - sync qpic
- **S** - search
- **Q** - quit
- **F1** - cycle: menu/help/normal
- **F11** - fill screen
- **[x]** - close menu bar

Folders Mode

Displays parent folder name followed by an indented list of folder names. Current folder is highlighted (enclosed in square brackets). Folder properties: name, description, img-flag, H-flag, X-flag. If img-flag is false, folder contains no images, only other folders. By convention, images stored in non-image folders reside in a sub-folder called "$".

Tiles Mode

Whenever the user is in Folders Mode and presses Enter when an image folder is highlighted, the current mode becomes Tiles Mode. The display is divided into 3 rows of equal height (or n rows where n > 1). Each row contains images. All portrait-mode images are of equal height but of varying widths. Every landscape-mode image is the same width as the height of the row which contains that image. All images are separated by a white, one-pixel gap (user may increase pixel count of gap, globally). Clicking on an image will display it in Image Mode. Pressing Up Arrow makes the current mode become Folders Mode.

Image Mode

Single image is displayed, expanded by the maximum amount available on the user's display. Pressing L or U modifies special flag if needed and the current mode becomes Tiles Mode. Pressing Up Arrow makes the current mode become Tiles Mode. Pressing Enter enables editing of one-line caption, and/or toggling display of image captions, and/or toggling the filtering out of images which lack captions. If this image is accompanied by a link to a video, click on play to follow that link under a new browser tab. The image-view count of the user who is the image owner is incremented, if different from the user viewing the image.

Slide Mode

Press Ctrl+Down Arrow to toggle between Tiles Mode and Slide Mode. Slide Mode displays between 1 and 3 images per slide. Each slide tries to fill the entire display. Click on an image in slide mode to enter image mode, then press D to delete the image from the slide. In slide mode, press I to insert an image. The next time the I command is used in image mode, the image is inserted into the slide. Various image arrangements (side-by-side, stacked, or some combination) are used automatically, depending on the aspect ratios of the images on the slide. New slides can only be inserted at the head of the slide list. One of the panels on a given slide (which contains 1 to 3 panels) can contain Lystagger code instead of an image.

Image Folder Sharing

Google Drive (or possibly Dropbox) will be used as the image sharing platform. Users can use Qpicary to keep track of their favorite image-sharing users and the names of their favorite image folder names shared by those users.

Bookmark/Insert Commands

The Insert command displays the select-user web page. After selecting a user, the Local mode switch is off. Further Insert commands insert images/folders into the current local folder, and allow the user to change the current local folder. The Quit command sets the Local mode switch back to on. The Bookmark command bookmarks the current folder when Local mode is off (enabling bookmark parent list selection), and accesses the bookmark tree when Local mode is on.

Searching

Popularity is used in searches: how many times an image/folder has been viewed, downloaded, or bookmarked. Searching is used to search for users and the images held by those users. Text searches involve qpic names/descriptions, image captions, and slide panels.

Image File Names

Newly added image files can have arbitrary file names. After the sync command is used, newly added image files (except for the least recently added image, which is manually appended with .1 when saved by the user) are renamed to $QNNNN.ext, where ext is a graphics file type such as PNG or JPEG and NNNN is a 4-digit number. Clock arithmetic is used, where 9999 corresponds to -1, 9998 corresponds to -2, and so on. Usually, the head of the queue is positive, and the tail of the queue is 0 or negative. No qpic can have more than 10,000 image files.

The sync command programmatically removes the .1 extension of the least recently added image. When the user goes to add the next batch of newly added images, and tries to save the same image file F that originally had the .1 extension, it will already exist in the current qpic. This will let the user know when to stop adding images. The next time the sync command is performed, image file F is renamed to have the same $QNNNN format as all of the older image files in the current qpic.

Lystagger

Lystagger is a simplified markup language used to replace HTML. Arbitrary Lystagger code can be embedded in the Jovelyst echo statement. Lystagger syntax, where asterisk (*) means repetition, is defined as follows:

- Tags:
  - [tag]
  - [tag: body]
  - [tag (fld val)*: body]
- Body:
  - text
  - [: text]*
  - [(fld val)*: text]*
- Call Jovelyst code:
  - [expr: <expr>]
  - [exec: <stmt>...]
  - [lyst: <path>]

Monospace Mode

In monospace mode, all body text rendered to the screens of Qpicary end-users is in a mono-spaced, typewriter-style font. Every character takes up 2 square cells: an upper cell and a lower cell. Superscripts and subscripts are handled by employing a vertical offset of one square cell. Header text is also mono-spaced, and each character takes up 2 oversized square cells.

Additional Formatting

The grid of characters can be subdivided into panels, which can themselves be subdivided into more panels, and so on. Any panel can contain zero or more text boxes, which may overlap each other. Vertical grid lines each take up one square cell per row of square cells. Horizontal grid lines are displayed in the same pixel row as underscore characters. Any row of square cells containing a horizontal grid line which is 2 pixels wide is taller by exactly one pixel. The following bracket characters: ( ) [ ] { } can be oriented vertically or horizontally, taking up a single column or row of at least 2 square cells, respectively. Widgets

such as check boxes, radio buttons, and combo box arrows take up 4 square cells (2 by 2). Images, animations, and diagrams are contained in canvas objects, which can appear anywhere panels can appear.

Rich-Text Mode

In rich-text mode, a given header or paragraph of body text can consist of a single variable-width font. Paragraphs have before/after spacing, left/right indent, and line spacing (single, double, 1.5, etc.). Panels have margins on all 4 sides. In both rich-text and monospace modes, text is rendered to the HTML5 canvas object. Some features like form fields and submit buttons use hidden HTML.

Jovelyst

Jovelyst 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). Jovelyst code is often accompanied by Lystagger screen definition files. Lystagger is similar to HTML, except open tags begin with an open square bracket and a keyword, and the closing tag is simply a close square bracket. Any text enclosed in a tag is preceded by a colon. File extensions include .LYST (Jovelyst) and .LSTG (Lystagger).

Special Characters

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

Version 0.1

- No inheritance
- No interfaces
- No IDE
- No rich text

Differences from Python

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

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. The close delim switch automatically inserts a closing parenthesis/double quote when the open delimiter is inserted.

Jysta and Psyvaspace

Lystplayer

Lystplayer is a website where you can play 2-player non-animated games: think board games and card games. You can also create your own Lystplayer games using Jovelyst and Lystagger. Lystplayer is implemented using Jovelyst and Lystagger.

Lystplayer Members

Lystplayer members can create home pages/bios written in Jovelyst and Lystagger, post in forums, view games in progress, participate in tournaments, and hold player rating values for each Lystplayer game they are involved with. An example of a player rating value is a chess rating, where a rating of 1700 or more would be held by a very skilled chess player. The "Outer Forum" is a special forum used only by no-name users, who have read/write access to that forum.

Jovelearn

Jovelearn is a website which links tutors with students. Some tutors charge their students an hourly rate, and Jovelearn receives 10 percent of that revenue stream. All volunteer tutors teach members of Psyvaspace, an online community of consumer/survivors. The tutors teach math, literacy, and coding. A web-based interactive whiteboard enables the tutor to interact with a single student. The tutor and student take turns interacting with the whiteboard. At the beginning of each turn, every move (mouse clicks and text entered during the previous turn) is replayed, and then further interaction takes place. Prefabricated lessons are prepared by volunteer curriculum-writers and displayed on the interactive whiteboard. The whiteboard is written in Jovelyst.

Psyvaspace

Psyvaspace.org is an online community of consumer/survivors (those with mental health issues). All Psyvaspace members who qualify for a free membership must be clients of a partner organization such as Progress Place, which is a clubhouse of consumer/survivors. Other examples of possible partner organizations include CAMH (a psychiatric hospital) and Sound Times (a drop-in center), and the primary mandate of all partner organizations is to serve consumer/survivors. Progress Place may or may not agree to partner with Psyvaspace, whereby members of the Clerical Unit at Progress Place perform data entry for Psyvaspace.

All clients of the partner organizations become Jysta members for free. Psyvaspace members can have home pages/bios, post in organization-specific or more general consumer/survivor forums, join chat rooms, write blogs, play 2-player games, and interact with tutors. Most or all of the above web-based functionality is written in Jovelyst. The forums and chat rooms are moderated by volunteers recruited by Jysta. Psyvaspace users can access a database of mental health resources which is maintained by Progress Place.

Sites

Jysta members can create websites using Jovelyst and Lystagger. A given web designer pays a monthly fee to Jysta if the websites she creates use more than double (a factor being greater than 2) the median amount of resources consumed. Those amounts equal resources consumed for each web designer. The amount of resources consumed is calculated by multiplying megabytes of image files downloaded by kilo-nodes processed. A kilo-node equals 1000 nodes, and a node is usually 12 bytes long. It seems likely that most Jovelyst programs process many kilo-nodes per user session. One node is considered to be processed every time a new node is created in RAM. Every Jovelyst web page is accompanied by a static HTML web page with a link at the top to the associated Jovelyst web page.

factor	monthly fee	factor	monthly fee
2	$10	256	$125
4	15	512	150
8	25	1024	175
16	40	2048	200
32	60	4096	225
64	80	8192	250
128	100	16,384	275
		>32,768	300

About Me

I am Mike Hahn, the founder of Jysta.org. 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. I'm now a volunteer math/computer tutor at West Neighbourhood House. My hobbies are reading quora.com questions/answers and the news at cbc.ca. About twice a year I get together with my sister Cathy who lives in Victoria. She comes here or I go out there usually in the summer. A few months prior to starting my Qpicary project I used to lie on the couch a lot, not being very active. Now I'm busy most of the time. I visit my brother Dave once a month or so and I also visit my friends Main and Steph once or twice a month.

Contact Info

Mike Hahn
Founder, Jysta.org
515-2495 Dundas St. West
Toronto, ON M6P 1X4

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

Implementation Steps

1. Read Murach's Java Servlets and JSP book
2. Implement Jabbler: web-based HTML5 Scrabble game, user vs. robot
    - Jabbler is currently console-based Java Scrabble game
3. Implement Jovelyst 0.1, console-based
    - Token parsing and building program tree has already been implemented
4. Finish Jovelyst, console-based
5. Write Lystagger design specs
6. Implement monospace mode
7. Implement rich-text mode
8. Implement Qpicary
9. Design website
10. Launch website
11. Beta test Qpicary/Jovelyst period 6 mos.
    - Free 2-year membership for beta testers
12. Implement Sites
    - Accept credit card payments
13. Implement Jovelyst IDE (open source):
    1. Code editor
    2. WYSIWYG board/piece editor
    3. Codeless prototyping system
14. Implement Jabbler: web-based, 2-player
15. Implement monospace mode, dual user
16. Implement Lystplayer
    - Support rich-text mode, dual user
17. Implement Jovelearn
18. Approach Progress Place
19. Implement Psyvaspace
20. Hire staff
    1. Board members
    2. Executive Director
    3. Forums Coordinator
    4. Tutor Coordinator
    5. Head of Curriculum Development
    6. Mike is the Head of Software Development
    7. Recruit moderators, tutors, and curriculum-writers
21. After Step 20.1
    - Psyvaspace.org is incorporated as non-profit organization
22. Acquire office space

Jovelyst Grammar

White space occurs between tokens (parentheses and semicolons need no adjacent white space, also any semicolon before a close parenthesis may be omitted):

<source file>:
- [<use>] [* <vars>] [<vars>] [do <block>] [<def>]… [<class>]… [do <block>]

<use>:
    use (<import-semi>...)

<import-semi>:
    <import-stmt> **;**

<import stmt>:
    import <module>
    import (<module>…)
    from <rel module> import <mod list>
    from <rel module> import all

<module>:
    <name>
    (<name> as <name>)
    (**:** <name>… [as <name>])

<mod list>:
    <id as>
    (<id as>…)

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

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

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

<class>:
- ([*] <cls typ><name> [<base class>] [<does>] [* <vars>] [<vars>] <def>…)
- ([*] scool <name> [<does>] [<const list>] [<def hdr>]...)
- ([*] enum <name><elist>)

<cls typ>:
    class
    abclass

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

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

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

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

<def hdr>:
    (<defun><name> ([<parms>]) [<dec>])

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

<defun>:
    def
    abdef

<vars>:
    var (<id>...)

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

<parm>:
    <id>
    (tuple <id>...)
    (<set op><id><expr>)
    (<set op> (tuple <id>...) <expr>)

<dec>:
    decor <call expr>...

<block>:
    ([<stmt-semi>]…)

<stmt-semi>:
    <stmt> **;**

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

<pjump stmt>:
    return <expr>
    ** <raise stmt>

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

<stmt>:
    <open stmt>
    <closed stmt>

<open stmt>:
    <if stmt>
    <while stmt>
    <for stmt>
    ** <try stmt>
    <pjump stmt>
    <pcall stmt>
    <asst stmt>
    <del stmt>

<closed stmt>:
    <jump stmt>
    <call stmt>
    <print stmt>
    <lstg tag>

<call expr>:
- (<name> [<arg list>])
- (**:** <obj expr> [<colon expr>]…
  (<method name> [<arg list>]))
- (call <expr> [<arg list>])

<call stmt>:
    (<name> [<arg list>])

<pcall stmt>:
- **:** <obj expr> [<colon expr>]…
  (<method name> [<arg list>])
- call <expr> [<arg list>]

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

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

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

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

<set op>:
    set | =

<target expr>:
    <name>
    (**:** <name> [<colon expr>]… <name>)
    (slice <arr><expr> [<expr>])
    (slice <arr><expr> all)

<arr>: *// string or array/lyst*
    <name>
    <expr>

<obj expr>:
    <name>
    <call stmt>

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

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

<for stmt>:
    for <name> in <expr> do <block>

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

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

<return stmt>:
    return

<break stmt>:
    break

<continue stmt>:
    continue

<del stmt>:
    del <expr>

<paren stmt>:
    (<open stmt>)
    <closed stmt>

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

<expr>:
    <keyword const>
    <literal>
    <name>
    (<unary op><expr>)
    (<bin op><expr><expr>)
    (<multi op><expr><expr>…)
    (<quest><expr><expr><expr>)
    <lambda>
    (quote <expr>...)
    <renum expr>
    <tuple expr>
    <lyst expr>
    <dict expr>
    <bitarray expr>
    <string expr>
    <bytezero expr>
    <bytes expr>
    <target expr>
    <obj expr>
    <cast>

<quest>:
    quest | ?

<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>

<tuple expr>:
    (tuple <expr>…)

<lyst expr>:
    (lyst [<expr>]…)

<dict expr>:
    (dict [<pair>]…)

<bitarray expr>:
    (bitarray <enum name> [<elist>])
    (bitarray <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>
    <idt>
    (<id><id>)

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

<renum expr>
    (renumize <expr><ren id>...)
    (renumize <expr><ren int>...)
    (renumize <expr><ren ch>...)

<ren id>:
    (0 <id>)
    (1 <id>)
    (1 <id><id>)

<ren int>:
<ren ch>:
    // expr is <dec int> | <char lit>
    (0 <expr>)
    (0 <expr><expr>)
    (1 <expr>)
    (1 <expr><expr>)

<cast>:
    (cast <type><expr>)

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

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

No white space allowed between tokens, for rest of Jovelyst 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>:
    none
    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

<string lit>:
    [$ <str prefix>] <short long>

<str prefix>:
    r | u | R | U

<short long>:
    " [<short item>]... "
    " " " [<long item>]... " " "

<short item>:
    <short char>
    <escaped str char>

<long item>:
    <long char>
    <escaped str char>

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

<long char>:
    any source char. except "\"

<bytes lit>:
    $ <byte prefix><shortb longb>

<byte prefix>: *// any case/order*
    b | br

<shortb longb>:
    " [<shortb item>]... "
    " " " [<longb item>]... " " "

<shortb item>:
    <shortb char>
    <escaped char>

<longb item>:
    <longb char>
    <escaped char>

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

<longb char>:
    any ASCII char. except "\"

<escaped char>:
    \newline
        ignore "\", newline chars.
    \\ *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*