

Comments

//Comment	Single Line Comment
/*	
Comment	Multi-line comment
*/	
TODO:	Printed during compilation:
Reminder!	Reminder!
# Tag	Creates a tag, accessible from the runtime

Choices

Prompt	
*/+ (label) {condition} [option]	
Response	
+ (label)	Options can be given a label
Option	
*	Option only shows up once
+	"sticky" option - can be chosen multiple times
Prompt	Prompt
* Option 1	1: Option 1
* Option 2	2: Option 2
*/+ Option	Responses are written with text on a new line
Response	
Prompt	Prompt
* A	1: A
1	2: B
+ B	>2 B
2	2

Choices (cont)

*/+	Options in bracket are displayed in choice list, not in output
[Option]	
1: Option	
>1	
*/+ A [B]	Text before bracket is displayed in both choice list and output
1: A B	
>1 A	
*/+ A [B]	Text after brackets is only shown in output
C	
1: A B	
>1 A C	
*/+ Hello[.] ,	1: Hello.
how are you?	1> Hello,
	how are
	you?
*/+	If condition = true, display option
{condition}	
[option]	
* {a} [a]	If a = true:
* b	1. a
	2. b
	If a = false:
	1. b

Choices (cont)

*/+ -> knot	Fallback option. Never displayed to player, automatically used
Fallback	*A 1. A
without	Nothing >1 A
diverting to	* -> Nothing
a knot	You Died You Died.
	-> END
- (label)	"Gather" all choices back to this point. See Content Flow for more information.
Content	

Glue

<>	"Glue", skips automatic line-break
This is a line break	This is a line break
This is	This is glue
<>	
glue	
This is midsentence<> -> glue	
=== glue	
divert with glue	
	This is midsentence divert with glue
This is midsentence-> noglue	
=== divert	
divert without glue	
	This is midsentence divert without glue



By menaechmi (menaechmi)

cheatography.com/menaechmi/

Not published yet.

Last updated 14th February, 2025.

Page 1 of 5.

Sponsored by [Readable.com](https://readable.com)

Measure your website readability!

<https://readable.com>

Functions

Functions add a call stack and optional return values to knots

```
=== function name(parameters) ===
function body
~ return return_value
```

name() Functions are always called with parentheses

a{letter(b)}c Functions are called with <> glue by default

(ref parameter) Pass parameter by reference, default behavior is by value

Functions cannot contain stitches, choices, or diverts

Safe to use recursively. See Variables for details.

"Standard Library" functions

CHOICE_COUNT() Number of options currently being presented

URNS() Total number of player choices of the game

URNS_SINCE(-> knot) Number of player choices since a knot was seen, -1 has never been seen, 0 is current.

SEED_RANDOM(seed) Fixes the random number generator to produce the same outcomes

READ_COUNT(-> knot) Number of times knot has been seen. Equivalent to {knot}

List Functions

LIST_VALUE(list.item) Prints item's position in list 1-indexed

LIST_ALL(list) List all values in list

Multivalue list functions assume an active element. Use LIST_ALL(list) for other lists

LIST_COUNT(mvlist) Count active item

LIST_MIN(mvlist) Get active item with the lowest index

LIST_MAX(mvlist) Get active item with the highest index

LIST_RANDOM(mvlist) Get a random active item in list

LIST_RANGE(mvlist, min, max) Gets the inclusive values between min and max. Min/max can be integers or list items.

LIST_INVERT(mvlist) Flips active and inactive. Empty list returns null

Math Functions

INT(x), FLOAT(x) Cast x to type

FLOOR(x) Round x down to nearest integer. (-1.5 rounds to -2)

POW(x,y) Raises x to the y power

RANDOM(min,max) Generates a number between min and max, inclusive

Knot/Stitch

=== label(parameter) ===
Content

=== lab- el *=== Creates "knot" named label

=== label Shorthand to create knot

-> label Divert, divert arrow. Redirects flow to label

=== knot = stitch-_label "Stitch", a subsection of a knot.

Stitch
Content

=== label(p1, p2) Optional parameter for knots or stitches.

-> knot(a,b) Divert to knot with parameters

=== knot(-> a) Use a divert as a parameter
-> a

Diverts are explained in more detail in Control Flow

Math/Logic

~ Indicates the line is not text

+ - / * Basic math operators, addition, subtraction, multiplication, division

and, or, not Logical operators. Symbol versions will not work in all contexts.

&&, ||, !



By menaechmi (menaechmi)

Not published yet.

Last updated 14th February, 2025.

Page 2 of 5.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>

Math/Logic (cont)

~	Increment variable +1
variable++	Decrement variable -1
~	
variable--	
%	Mod operator, returns the remainder after division.
~ x = 2/3	Math types are implicit, so x is 0

Special Diverts

-> END	End the story. CSS class .end
-	Flow ends intentionally
> DONE	
Diverts are case sensitive: -> DONE and -> Done and -> DoNe are all separate	

Conditionals

{	Conditionals take place inside of curly brackets, and can control story content
}	
>, <, >=, <=, ==, !=	Standard operators
"a"=="a", "a"!="b", "ab"? "a"	String queries. Equal, unequal, contains
{condition: true	If-else statement. The else is optional
- else: false }	
{ - condition1: statement1	If/else if/ else statement. Evaluates in order
- condition2: statement2	
- else: statement }	

Conditionals (cont)

{ x:	Case statement
- 0:	zero
- 1:	one
- 2:	two
- else:	
lots }	
All labels are read counts of the content.	
{label:...}	Has knot been visited?
{!label:...}	Is knot unvisited?
{label > x:...}	Has the knot been seen more than x times?

Lists

LIST list = a, b	Create a list and a variable
LIST list = a, (default)	Parenthesis selects state at assignment
var = a	Assign value a from list
var = list.a	Specify which list with selectors
{list.a}	List values print as names.
{list(1)}	Both a in this case.
LIST numbers = one =1, two, five = 5	Set custom list numbering. Skipped numbers increment by one (1).
var++	Point to next item in list
var--	Point to previous item in list

Multivalued Lists

Multivalued Lists are lists with references to multiple list items. Items in the list are "active"	
mvlist = (a), (b)	Set active items
mvlist = (a, b)	
mvlist = (a = 1), (b) = 2	
mvlist += a	Add items to the list to activate them
mvlist += (a, b)	
mvlist -= a	Remove items from the list to deactivate them
mvlist -= (a, b)	
Referencing a multivalued list is assumed to refer to active states only.	
{ mvlist: has active no active }	Conditionals are true if any state is active
{ mvlist == (1,2): exactly 1,2 not exactly 1,2 }	Equality checks if an exact set is matched. 1,2,3 and 1 will both fail here.
{ mvlist has a: { mvlist ? (a,b) }	Has all (?) (a AND b)
{mvlist hasnt (a,b): {mvlist !? a	Hasnt all (!?) (!a AND !b)
list_a ^ list_b	Intersection (^) Has some
statelist () ?	Returns false



By menaechmi (menaechmi)

Not published yet.

Last updated 14th February, 2025.

Page 3 of 5.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>

Content Alternatives

Most alternatives have two syntaxes, symbolic and multi-line block (indicated as **{ alternate: }**) which requires each option to start with "-"

{1,2,3} 1,2,3,- Stopping Sequence
{ 3,3,3... - repeats last option
stopping: when out

{&1,2,3} 1,2,3,- Cycle - repeats
{ cycle: 1,2,3... options when out

{!1} 1 Once-only

{!1,2,3} 1,2,3 Once-only
{ once: sequence

{~heads, heads, Shuffle - chooses
tails} heads, from options each
{ shuffle: tails... time

{ shuffle b,a,c,- Shuffles all but the
stopping: c,c... last entry, plays
 - a through it, and then
 - b repeats the last
 - c} entry

{ shuffle b,c,a Shuffles the list and
once: plays through it one
 - a time
 - b
 - c}

{.,3} .,3 Empty options don't
 display

{&a, a,2,c,a,c Alternatives can be
{!2},c} nested

{1,2, ->a} 1, 2, a Alternatives can use
 diverts

Content Alternatives (cont)

+ a 1. Choices can use alternatives
{!b,c} a
 b
 1.
 a
 c

+ \ 1. Escape whitespace with "\" to
{&a,b} a start choices with alternative
 1.
 b

Variables

VAR Global variable. Accessible
name = from the runtime and the
value story.

CONST Defines a variable that cannot
NAME = be changed
value

~ Used for lines that are game
 logic, not text

~ temp Temporary variable. Stitch-
name = level context
value

~ name = Change the value of a
value variable

{name} Curly brackets print variables
 in text

Temporary variables are safe to use in
 recursion. Globals are not. See Functions
 for details.

Variable types

1,2,3 Integer

0.5,0.9,0,6 Floating point

true, false Boolean (lowercase
 only)

-> knot, -> knot.s- Story Address/Divert
 titch

"a", "a b", "{~a|b|- Content
 c}"

Variable types (cont)

Type Content can contain ink, but are
 evaluted to a string based on seed. **VAR va**
r = " {a| b}" is therefore not allowed

Variables are also used to reference lists.
 See Lists and Multivalue Lists for details.

Control Flow

Start Ink tries to start a story from
===knot1 anything not in a knot

-> start To start from within a knot,
===start divert to the knot

=== knot Knot control defaults to any
a content not under a stitch
=stitch

=== knot If there is no header content,
=stitch the first stitch will play instead.
a
= stitch2

If a section ends without diverting, flow will
 end.

-> knot.s- Divert to a stitch using full
titch address

-> Divert to stitch from within the
stitch_b same knot

Diverts can go to any labeled element

***a** Choices (and content) can be
++b nested, so that different
****b** choices have different
***a** outcomes.
++c



By menaechmi (menaechmi)

Not published yet.

Last updated 14th February, 2025.

Page 4 of 5.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>

Control Flow (cont)

-- Gathers can be nested. They will (label) collect choices of the same or deeper lever.

See Tunnels and Threads for even more control flow options.

Tunnels

Tunnels return the story to where they were called, letting you reuse the same segment in different parts of the story or run sub-stories.

content Calling a tunnel. After the
-> tunnel -> tunnel, continues at more
more content
content

-> tunnel1 - You can chain calling
> tunnel2 -> tunnels

-> tunnel -> Or divert elsewhere
knot

=== tunnel Tunnels end with a double
content divert
->->

===tunnel_a Tunnels can also divert, as
-> tunnel_b long as it ends with a double
-> divert
->->

->-> knot Go to knot instead of
returning the tunnel

Safe to use recursively

Threads

Threads follow knots, collecting choices to present to the player at a single point. They can be used to split content up or fork a story.

<- Start a thread
knot_name

<- knot1 -> DONE Tells the compiler
<- knot2 the story continues at the
-> DONE threads, not here.

<- 1. a
Choice_a 2. b
<- 3. c
Choice_b Story continues at the choice
<- made
Choice_c
-> DONE

<- 1. a
Choice_a 2. b
<- 3. c
Choice_b Story continues at more
<- content
Choice_c
more
content

If using to present a choice in many places, it might be helpful to include a return location divert as a paramater.

= location
<- common_choice(-> location)
* more_choices...
= common_choice(-> return)



By menaechmi (menaechmi)

cheatography.com/menaechmi/

Not published yet.

Last updated 14th February, 2025.

Page 5 of 5.

Sponsored by [Readable.com](https://readable.com)

Measure your website readability!

<https://readable.com>