Ink 1.2.0/Inky 0.15 Cheat Sheet

by menaechmi (menaechmi) via cheatography.com/211056/cs/45645/

Comments

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

Choices

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

Choices (cont)

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

Choices (cont)

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

Glue

<>	"Glue", skips automatic line-break
This is a line break	This is a line break
This is <> glue	This is 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

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 Libary" functions

CHOICE_COUNT()	Number of options currently being presented
TURNS()	Total number of player choices of the game
TURNS_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

=== label *===	Creates "knot" named label
=== label	Shorthand to create knot
-> label	Divert, divert arrow. Redirects flow to label
=== knot = **stitch_label** Stitch Content	"Stitch", a subsection of a knot.
=== label**(p1, p2)**	Optional parameter for knots or stitches.
-> knot(a,b)	Divert to knot with parameters
=== knot(-> a) -> a	Use a divert as a parameter

Diverts are explained in more detail in Control Flow

Math/Logic

~	Indicates the line is not text
+ - / *	Basic math operators, addition, subtration, multiplication, division
and, **or**, **not** **&&**, **||**, **!**	Logical operators. Symbol versions will not work in all contexts.

Math/Logic (cont)

~ variable++ ~ variable--	Increment variable +1 Decrement variable -1
%	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`
-> DONE	Flow ends intentionally

Diverts are case sensitive: `-> DONE` and `-> Done` and `-> DoNe` are all separate

Conditionals

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

Conditionals (cont)

{ x: - 0: zero - 1: one - 2: two - **else**: lots **}**	Case statement

All labels are read counts of the content.

{label:...}
{!label:...}
	Has knot been visited?
	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(1)}
	List values print as names. 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

Mutlivalue Lists

Multivalue Lists are lists with references to multiple list items. Items in the list are "active"

mvlist = (a), (b)
mvlist = (a, b)
mvlist = (a = 1), (b) =2
	Set active items

mvlist **+=** a mvlist **+=** (a, b)	Add items to the list to activate them

mvlist -= a
mvlist -= (a,b)
	Remove items from the list to deactivate them

Referencing a multivalue 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

Ink 1.2.0/Inky 0.15 Cheat Sheet

by menaechmi (menaechmi) via cheatography.com/211056/cs/45645/

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**}** **{ stopping:**	1,2,3,-3,3,3...	Stopping Sequence - repeats last option when out
{&1,2,3**}** **{ cycle:**	1,2,3,-1,2,3...	Cycle - repeats options when out
{!1**}**	1	Once-only
{!1,2,3**}** **{ once:**	1,2,3	Once-only sequence
{~heads, tails**}** **{ shuffle:**	heads, heads, tails...	Shuffle - chooses from options each time
{ shuffle stopping: - a - b - c**}**	b,a,c,-c,c...	Shuffles all but the last entry, plays through it, and then repeats the last entry
{ shuffle once: - a - b - c**}**	b,c,a	Shuffles the list and plays through it one time
{,,3}	,,3	Empty options don't display
{&a, {!2},c}	a,2,c,a,c	Alternatives can be nested
{1,2, ->a}	1, 2, a	Alternatives can use diverts

Content Alternatives (cont)

+ a {!b,c}	1. a b 1. a c	Choices can use alternatives
+\ {&a,b}	1. a 1. b	Escape whitespace with "\" to start choices with alternative

Variables

VAR name = value	Global variable. Accessible from the runtime and the story.
CONST NAME = value	Defines a variable that cannot be changed
~	Used for lines that are game logic, not text
~ temp name = value	Temporary variable. Stitch-level context
~ name = value	Change the value of a 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.stitch	Story Address/Divert
"a", "a b", "{~a|b|c}"	Content

Variable types (cont)

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

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

Control Flow

Start ===knot1	Ink tries to start a story from anything not in a knot
-> start ===start	To start from within a knot, divert to the knot
=== knot a =stitch	Knot control defaults to any content not under a stitch
=== knot =stitch a = stitch2	If there is no header content, the first stitch will play instead.

If a section ends without diverting, flow will end.

-> knot.stitch	Divert to a stitch using full address
-> stitch_b	Divert to stitch from within the same knot

Diverts can go to any labeled element

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

Control Flow (cont)

- - (label)	Gathers can be nested. They will 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 -> tunnel -> more content	Calling a tunnel. After the tunnel, continues at more content
-> tunnel1 -> tunnel2 ->	You can chain calling tunnels
-> tunnel -> knot	Or divert elsewhere
=== tunnel content ->->	Tunnels end with a double divert
===tunnel_a -> tunnel_b -> ->->	Tunnels can also divert, as 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.

<- knot_name	Start a thread
<- knot1 <- knot2 **-> DONE**	-> DONE Tells the compiler the story continues at the threads, not here.
<- Choice_a <- Choice_b <- Choice_c -> DONE	1. a 2. b 3. c Story continues at the choice made
<- Choice_a <- Choice_b <- Choice_c more content	1. a 2. b 3. c Story continues at 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)
```