This is ducksay!

-------------

>(' )
*/
((/ '----/
\ ~-= /
^~^~^~^~^~^~^~

(But which Version?)

-------------

>0_ (.--)__ _

(v2.6)

-------------

___/

\/( ()

| w----||
||---||

(by Jonathan P. Spratte)

-------------

/.

./_/\'__

<<::[0]8::>>

\---\---\---

[ | -=-==== | ]
[ | ====-=| ]
\[ ||O:====| ]
\[ ||O:==| ]
\[ ||____|| ]
\[ \__/\_||
\( \__/\_||

(Today is 2022-09-29)

-------------

\...\//\//\//\//\//
\| /\|/\|/\|/\|
\||/\||/\||/\||/\|
\o__,|//|//|//|//|
1 Acknowledgements

I was created by Plergux!*
2 Documentation

This is ducksay! A cowsay for \LaTeX. ducksay is part of \TeX\Live and MiK\TeX\ since September 2017. If it is not part of your installation it means that your \LaTeX\ installation is \textit{really} out of date, you have two options: Update your installation or try to install ducksay yourself. Chances are that if you opt for the latter, the version of expl3 in your \LaTeX installation is too old, too, and the l3regex module is not yet part of expl3. In that case you’ll get a few undefined control sequence errors. \texttt{\usepackage{l3regex}} prior to loading ducksay might fix these issues. Additionally you’ll need grabbox for version 2 of ducksay that won’t be part of your \LaTeX installation, too. Please note that I don’t actively support out of date \LaTeX\ installations, so if loading l3regex doesn’t fix the issues and you’re on an old installation, I won’t provide further support.

2.1 Downward Compatibility Issues

In the following list I use the term “version” to refer to package versions, the same is true if I use an abbreviation like “v2.0” (or anything that matches the regular expression v\d+(.\d+)?). For the code variant which can be set using the \texttt{version} option I’ll use the term “variant” or specify directly that I’m referring to that option (the used font may be a hint, too).

- \textbf{v2.0} Versions prior to v2.0 did use a regular expression for the option \texttt{ligatures}, see subsubsection 2.2.2 for more on this issue.
  - In a document created with package versions prior to v2.0 you’ll have to specify the option \texttt{version=1} with newer package versions to make those old documents behave like they used to.

- \textbf{v2.3} Since v2.3 \texttt{\AddAnimal} and \texttt{\AddColoredAnimal} behave differently. You no longer have to make sure that in the first three lines every backslash which is only preceded by spaces is the bubble’s tail. Instead you can specify which symbol should be the tail and how many of such symbols there are. See subsubsection 2.2.1 for more about the current behaviour.

- \textbf{v2.4} The add-think key was deprecated in v2.3 and was removed in v2.4 since the output symbols of the bubble tail are handled differently and more efficient now.

2.2 Shared between versions

2.2.1 Macros

A careful reader might notice that in the below list of macros there is no \texttt{\ducksay} and no \texttt{\duckthink} contained. This is due to differences between the two usable code variants (see the \texttt{version} key in subsubsection 2.2.2 for the code variants, subsubsection 2.3.2 and subsubsection 2.4.2 for descriptions of the two macros).

\texttt{\DefaultAnimal\{⟨animal⟩\}}

use the \texttt{⟨animal⟩} if none is given in the optional argument to \texttt{\ducksay} or \texttt{\duckthink}. Package default is duck.
\DucksayOptions set the defaults to the keys described in subsubsection 2.2.2, subsubsection 2.3.3 and subsubsection 2.4.3. Don’t use an (animal) here, it has no effect.

\AddAnimal adds (animal) to the known animals. (ascii-art) is multi-line verbatim and therefore should be delimited either by matching braces or by anything that works for \verb. If the star is given (animal) is the new default. One space is added to the begin of (animal) (compensating the opening symbol). The symbols signalizing the speech bubble’s tail (in the hedgehog example below the two s) can be set using the tail-symbol option and only the first tail-count occurrences will be substituted (see paragraph 2.2.2.1 for more about these options). For example, hedgehog is added with:

\AddAnimal[tail-symbol=s]{hedgehog}

It is not checked whether the animal already exists, you could therefore redefine existing animals with this macro.

\AddColoredAnimal It does the same as \AddAnimal but allows three different colouring syntaxes. You can use \textcolor in the (ascii-art) with the syntax \textcolor{(color)}{(text)}. Note that you can’t use braces in the arguments of \textcolor. You can also use a delimited \color of the form \bgroup\color{(color)}{(text)} \egroup, a space after that \egroup will be considered a space in the output, so you don’t have to care for correct termination of the \egroup (so \bgroup\color{red}RedText \egroupOtherText is valid syntax). You can’t nest delimited \colors. Also you can use an undelimited \color. It affects anything until the end of the current line (or, if used inside of the (text) of a delimited \color, anything until the end of that delimited \color’s (text)). The syntax would be \color{(color)}. The package doesn’t load anything providing those colouring commands for you and it doesn’t provide any coloured animals. The parsing is done using regular expressions provided by \LaTeX. It is therefore slower than the normal \AddAnimal.

\AnimalOptions With this macro you can set (animal) specific (options). If the star is given any currently set options for this (animal) are dropped and only the ones specified in (options) will be applied, else (options) will be added to the set options for this (animal). The set (options) can set the tail-1 and tail-2 options and therefore overwrite the effects of \duckthink, as \duckthink really is just \ducksay with the think option.

2.2.2 Options

The following options are available independent on the used code variant (the value of the version key). They might be used as package options – unless otherwise specified – or used in the macros \DucksayOptions, \ducksay and \duckthink – again unless otherwise specified. Some options might be accessible in both code variants but do
slightly different things. If that’s the case they will be explained in subsubsection 2.3.3
and subsubsection 2.4.3 for version 1 and 2, respectively.

\texttt{version=⟨number⟩}

With this you can choose the code variant to be used. Currently 1 and 2 are available. This can be set only during package load time. For a dedicated description of each version look into subsection 2.3 and subsection 2.4. The package author would choose version=2, the other version is mostly for legacy reasons. The default is 2.

\texttt{animal=⟨animal⟩}

One of the animals listed in subsection 2.6 or any of the ones added with \texttt{\AddAnimal}. Not useable as package option. Also don’t use it in \texttt{\DucksayOptions}, it’ll break the default animal selection.

\texttt{animal=⟨animal⟩}

Locally sets the default animal. Note that \texttt{\ducksay} and \texttt{\duckthink} do digest their options inside of a group, so it just results in a longer alternative to the use of \texttt{(animal)} if used in their options.

\texttt{ligatures=⟨token list⟩}

each token you don’t want to form ligatures during \texttt{\AddAnimal} should be contained in this list. All of them get enclosed by grouping \{ and \} so that they can’t form ligatures. Giving no argument (or an empty one) might enhance compilation speed by disabling this replacement. The formation of ligatures was only observed in combination with \texttt{\usepackage[T1]{fontenc}} by the author of this package. Therefore giving the option \texttt{ligatures} without an argument might enhance the compilation speed for you without any drawbacks. Initially this is set to ‘<>,’-.

\textbf{Note:} In earlier releases this option’s expected argument was a regular expression. This means that this option is not fully downward compatible with older versions. The speed gain however seems worth it (and I hope the affected documents are few).

\texttt{no-tail}

Sets \texttt{tail-1} and \texttt{tail-2} to be a space.

\texttt{random=⟨bool⟩}

If \texttt{true} a random animal will be used instead of the default one on each usage of \texttt{\ducksay} or \texttt{\duckthink}. The initial value is false and it defaults to true.

\texttt{say}

Sets \texttt{tail-1} and \texttt{tail-2} as backslashes.

\texttt{schroedinger}

Sets randomly either \texttt{animal=schroedinger-alive} or \texttt{animal=schroedinger-dead} at the time of use. Both have the same size, so this doesn’t affect the needed space.

\texttt{tail-1=⟨token list⟩}

Sets the first tail symbol in the output to be \texttt{⟨token list⟩}. If set outside of \texttt{\ducksay} and \texttt{\duckthink} it will be overwritten inside of \texttt{\duckthink} to be 0.

\texttt{tail-2=⟨token list⟩}

Sets every other tail symbol except the first one in the output to be \texttt{⟨token list⟩}. If set outside of \texttt{\ducksay} and \texttt{\duckthink} it will be overwritten inside of \texttt{\duckthink} to be 0.

\texttt{think}

Sets \texttt{tail-1=0} and \texttt{tail-2=0}.
2.2.2.1 Options for \AddAnimal

The options described here are only available in \AddAnimal and \AddColoredAnimal.

tail-count=(int)
sets the number of tail symbols to be replaced in \AddAnimal and \AddColoredAnimal.
Initial value is 2. If the value is negative every occurrence of tail-symbol will be replaced.

tail-symbol=(str)
the symbol used in \AddAnimal and \AddColoredAnimal to mark the bubble’s tail. The argument gets \detokenized. Initially a single backslash.
2.3 Version 1

2.3.1 Introduction

This version is included for legacy support (old documents should behave the same without any change to them – except the usage of \texttt{version=1} as an option, for a more or less complete list of downward compatibility related problems see subsection 2.1). For the bleeding edge version of \texttt{ducksay} skip this subsection and read subsection 2.4.

2.3.2 Macros

The following is the description of macros which differ in behaviour from those of version 2.

\begin{verbatim}
\ducksay \ducksay[(options)]{(message)}
\end{verbatim}

options might include any of the options described in subsubsection 2.2.2 and subsubsection 2.3.3 if not otherwise specified. Prints an \texttt{animal} saying \texttt{(message)}. \texttt{(message)} is not read in verbatim. Multi-line \texttt{(message)s} are possible using \texttt{\textbackslash \textbackslash}. \texttt{\textbackslash \textbackslash} should not be contained in a macro definition but at toplevel. Else use the option \texttt{ht}.

\begin{verbatim}
\duckthink \duckthink[(options)]{(message)}
\end{verbatim}

options might include any of the options described in subsubsection 2.2.2 and subsubsection 2.3.3 if not otherwise specified. Prints an \texttt{animal} thinking \texttt{(message)}. \texttt{(message)} is not read in verbatim. Multi-line \texttt{(message)s} are possible using \texttt{\textbackslash \textbackslash}. \texttt{\textbackslash \textbackslash} should not be contained in a macro definition but at toplevel. Else use the option \texttt{ht}.

2.3.3 Options

The following options are available to \texttt{\ducksay}, \texttt{\duckthink}, and \texttt{\DucksayOptions} and if not otherwise specified also as package options:

\begin{verbatim}
\texttt{bubble=(code)} use \texttt{(code)} in a group right before the bubble (for font switches). Might be used as a package option but not all control sequences work out of the box there.
\texttt{body=(code)} use \texttt{(code)} in a group right before the body (meaning the \texttt{animal}). Might be used as a package option but not all control sequences work out of the box there. E.g. to right-align the \texttt{animal} to the bubble, use \texttt{body=\texttt{\hfill}}.
\texttt{align=(valign)} use \texttt{(valign)} as the vertical alignment specifier given to the \texttt{tabular} which is around the contents of \texttt{\ducksay} and \texttt{\duckthink}.
\texttt{msg-align=(halign)} use \texttt{(halign)} for alignment of the rows of multi-line \texttt{(message)s}. It should match a \texttt{tabular} column specifier. Default is \texttt{l}. It only affects the contents of the speech bubble not the bubble.
\texttt{rel-align=(column)} use \texttt{(column)} for alignment of the bubble and the body. It should match a \texttt{tabular} column specifier. Default is \texttt{l}.
\end{verbatim}
in order to detect the width the \textit{message} is expanded. This might not work out for some commands (e.g. \texttt{url} from hyperref). If you specify the width using \texttt{wd} the \textit{message} is not expanded and therefore the command \textit{might} work out. \texttt{count} should be the character count.

you might explicitly set the height (the row count) of the \textit{message}. This only has an effect if you also specify \texttt{wd}.

\subsection{Defects}

- no automatic line wrapping
- message width detection based on token count with \texttt{edef} expansion, might fail badly
2.4 Version 2

2.4.1 Introduction

Version 2 is the current version of ducksay. It features automatic line wrapping (if you specify a fixed width) and in general more options (with some nasty argument parsing).

If you’re already used to version 1 you should note one important thing: You should only specify the version and the ligatures during package load time as arguments to \usepackage. The other keys might not work or do unintended things and only don’t throw errors or warnings because of the legacy support of version 1. After the package is loaded, keys only used for version 1 will throw an error.

2.4.2 Macros

The following is the description of macros which differ in behaviour from those of version 1.

\ducksay \ducksay[\{options\}]{\{message\}}

options might include any of the options described in subsubsection 2.2.2 and subsection 2.4.3 if not otherwise specified. Prints an \{animal\} saying \{message\}. The \{message\} can be read in in four different ways. For an explanation of the \{message\} reading see the description of the \texttt{arg} key in subsubsection 2.4.3.

The height and width of the message is determined by measuring its dimensions and the bubble will be set accordingly. The box surrounding the message will be placed both horizontally and vertically centred inside of the bubble. The output utilizes \LaTeX's coffin mechanism described in \texttt{interface3.pdf} and the documentation of \texttt{xcoffins}.

\duckthink \duckthink[\{options\}]{\{message\}}

The only difference to \ducksay is that in \duckthink the \{animal\}'s think the \{message\} and don't say it.

2.4.3 Options

In version 2 the following options are available. Keep in mind that you shouldn’t use them during package load time but in the arguments of \ducksay, \duckthink or \DucksayOptions.

\texttt{arg=\{choice\}} specifies how the \{message\} argument of \ducksay and \duckthink should be read in. Available options are \texttt{box}, \texttt{tab} and \texttt{tab*}:

\texttt{box} the argument is read in either as a \texttt{\hbox} or a \texttt{\vbox} (the latter if a fixed width is specified with either \texttt{wd} or \texttt{wd*}). Note that in this mode any arguments relying on category code changes like e.g. \texttt{\verb} will work (provided that you don’t use \ducksay or \duckthink inside of an argument of another macro of course).

\texttt{tab} the argument is read in as the contents of a \texttt{tabular}. Note that in this mode any arguments relying on category code changes like e.g. \texttt{\verb} will not work. This mode comes closest to the behaviour of version 1 of \ducksay.
the argument is read in as the contents of a \texttt{tabular}. However it is read in verbatim and uses \texttt{\scantokens} to rescan the argument. Note that in this mode any arguments relying on category code changes like e.g. \texttt{\verb} will work. You can’t use \texttt{\ducksay} or \texttt{\duckthink} as an argument to another macro in this mode however.

\texttt{b} shortcut for \texttt{out-v=b}.

\texttt{body=(font)} add \texttt{(font)} to the font definitions in use to typeset the \texttt{⟨animal⟩}'s body.

\texttt{body=(font)}

\texttt{clear any definitions previously made (including the package default) and set the font definitions in use to typeset the \texttt{⟨animal⟩}'s body to \texttt{(font)}. The package default is \texttt{\verbatim@font}. In addition \texttt{\frenchspacing} will always be used prior to the defined \texttt{(font)}.}

\texttt{body-align=(choice)}

\texttt{sets the relative alignment of the \texttt{⟨animal⟩} to the \texttt{⟨message⟩}. Possible choices are \texttt{l}, \texttt{c} and \texttt{r}. For \texttt{l} the \texttt{⟨animal⟩} is flushed to the left of the \texttt{⟨message⟩}, for \texttt{c} it is centred and for \texttt{r} it is flushed right. More fine grained control over the alignment can be obtained with the keys \texttt{msg-to-body, body-to-msg, body-x} and \texttt{body-y}. Package default is \texttt{l}.

\texttt{body-bigger=(count)}

\texttt{vertically enlarge the body by \texttt{(count)} empty lines added to the bottom. This way top-aligning two different body types is easier (by actually bottom aligning the two):}

\begin{verbatim}
\ducksay[ghost, body-x=7mm, b, body-mirrored]{Buuuh!}
\ducksay[crusader, body-bigger=4, b, out-h=r, no-bubble]{
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
\_
bubble*=(font)
clear any definitions previously made (including the package default) and set the font
definitions in use to typeset the bubble to ⟨font⟩. This does not affect the ⟨message⟩
only the bubble put around it. The package default is \verbatim@font.

bubble-bot-kern=(dimen)
specifies a vertical offset of the placement of the lower border of the bubble from the
bottom of the left and right borders.

bubble-delim-left-1=(token list)
the left delimiter used if only one line of delimiters is needed. Package default is (.).

bubble-delim-left-2=(token list)
the upper most left delimiter used if more than one line of delimiters is needed. Package
default is /.

bubble-delim-left-3=(token list)
the left delimiters used to fill the gap if more than two lines of delimiters are needed.
Package default is |.

bubble-delim-left-4=(token list)
the lower most left delimiter used if more than one line of delimiters is needed. Package
default is \.

bubble-delim-right-1=(token list)
the right delimiter used if only one line of delimiters is needed. Package default is ).

bubble-delim-right-2=(token list)
the upper most right delimiter used if more than one line of delimiters is needed. Package
default is \.

bubble-delim-right-3=(token list)
the right delimiters used to fill the gap if more than two lines of delimiters are needed.
Package default is |.

bubble-delim-right-4=(token list)
the lower most right delimiter used if more than one line of delimiters is needed. Package
default is /.

bubble-delim-top=(token list)
the delimiter used to create the top and bottom border of the bubble. The package
default is {=} (the braces are important to suppress ligatures here).

bubble-side-kern=(dimen)
specifies the kerning used to move the sideways delimiters added to fill the gap for more
than two lines of bubble height. (the left one is moved to the left, the right one to the
right)

bubble-top-kern=(dimen)
specifies a vertical offset of the placement of the upper border of the bubble from the top
of the left and right borders.

c shortcut for out-v=vc.
col=⟨column⟩

specifies the used column specifier used for the 〈message〉 enclosing tabular for arg=tab
and arg=tab*. Has precedence over msg-align. You can also use more than one column this way: \ducksay[arg=tab,col=cc]{ You & can \ do & it } would be valid syntax.

hpad=⟨count⟩

Add 〈count〉 times more bubble-delim-top instances than necessary to the upper and lower border of the bubble. Package default is 2.

ht=⟨count⟩

specifies a minimum height (in lines) of the 〈message〉. The lines’ count is that of the needed lines of the horizontal bubble delimiters. If the count of the actually needed lines is smaller than the specified 〈count〉, 〈count〉 lines will be used. Else the required lines will be used.

ignore-body=⟨bool⟩

If set true the 〈animal〉’s body will be added to the output but it will not contribute to the bounding box (so will not take up any space).

msg=⟨font⟩

dd add 〈font〉 to the font definitions in use to typeset the 〈message〉.

msg*=⟨font⟩

clear any definitions previously made (including the package default) and set the font definitions in use to typeset the 〈message〉 to 〈font〉. The package default is \verbatim@font.

MSG=⟨font⟩

same as msg=⟨font⟩, bubble=⟨font⟩.

MSG*=⟨font⟩

same as msg*=⟨font⟩, bubble*=⟨font⟩.

msg-align=⟨choice⟩

specifies the alignment of the 〈message〉. Possible values are l for flushed left, c for centred, r for flushed right and j for justified. If arg=tab or arg=tab* the j choice is only available for fixed width contents. Package default is l.

msg-align-c=⟨token list⟩

set the 〈token list〉 which is responsible to typeset the message centred if the option msg-align=c is used. It is used independent of the arg key. For arg=tab and arg=tab* it is only used if a fixed width is specified and the macro \arraybackslash provided by array is used afterwards. The package default is \centering. It might be useful if you want to use ragged2e’s \Centering for example.

msg-align-j=⟨token list⟩

set the 〈token list〉 which is responsible to typeset the message justified if the option msg-align=j is used. It is used independent of the arg key. For arg=tab and arg=tab* it is only used if a fixed width is specified and the macro \arraybackslash provided by array is used afterwards. The package default is empty as justification is the default behaviour of contents of a p column and of a \vbox. It might be useful if you want to use ragged2e’s \justifying for example.

msg-align-l=⟨token list⟩

set the 〈token list〉 which is responsible to typeset the message flushed left if the option msg-align=l is used. It is used independent of the arg key. For arg=tab and arg=tab* it is only used if a fixed width is specified and the macro \arraybackslash provided by array is used afterwards. The package default is \raggedright. It might be useful if you want to use ragged2e’s \RaggedRight for example.
msg-align-r=(token list)
set the (token list) which is responsible to typeset the message flushed right if the option msg-align=r is used. It is used independent of the arg key. For arg=tab and arg=tab* it is only used if a fixed width is specified and the macro \arraybackslash provided by array is used afterwards. The package default is \raggedleft. It might be useful if you want to use ragged2e's \RaggedLeft for example.

msg-to-body=(pole)
defines the horizontal coffin (pole) to be used as the reference point for the placement of the (animal) beneath the (message). See interface3.pdf and the documentation of xcoffins for information about coffin poles.

no-bubble=(bool)
If true the (message) will not be surrounded by a bubble. Package default is of course false.

none=(bool) One could say this is a special animal. If true no animal body will be used (resulting in just the speech bubble). Package default is of course false.

out-h=(pole)
defines the horizontal coffin (pole) to be used as the anchor point for the print out of the complete result of \ducksay and \duckthink. See interface3.pdf and the documentation of xcoffins for information about coffin poles.

out-v=(pole)
defines the vertical coffin (pole) to be used as the anchor point for the print out of the complete result of \ducksay and \duckthink. See interface3.pdf and the documentation of xcoffins for information about coffin poles.

out-x=(dimen)
specifies an additional horizontal offset of the print out of the complete result of \ducksay and \duckthink.

out-y=(dimen)
specifies an additional vertical offset of the print out of the complete result of \ducksay and \duckthink

strip-spaces=(bool)
if set true leading and trailing spaces are stripped from the (message) if arg=box is used. Initially this is set to false.

t shortcut for out-v=t.

vpad=(count)
add (count) to the lines used for the bubble, resulting in (count) more lines than necessary to enclose the (message) inside of the bubble.

wd=(count)
specifies the width of the (message) to be fixed to (count) times the width of an upper case M in the (message)'s font declaration. A value smaller than 0 is considered deactivated, else the width is considered as fixed. For a fixed width the argument of \ducksay and \duckthink is read in as a \vbox for arg=box and the column definition uses a p-type column for arg=tab and arg=tab*. If both width is not smaller than 0 and wd* is not smaller than 0pt, wd* will take precedence.
\texttt{wd*=\langle\textit{dimen}\rangle} specifies the width of the \langle\textit{message}\rangle to be fixed to \langle\textit{dimen}\rangle. A value smaller than 0pt is considered deactivated, else the width is considered as fixed. For a fixed width the argument of \texttt{\textbackslash ducksay} and \texttt{\textbackslash duckthink} is read in as a \texttt{\textbackslash vbox} for \texttt{arg=box} and the column definition uses a \texttt{p}-type column for \texttt{arg=tab} and \texttt{arg=tab*}. If both \texttt{wd} is not smaller than 0 and \texttt{wd*} is not smaller than 0pt, \texttt{wd*} will take precedence.

\texttt{wd-eq-body=\langle\textit{bool}\rangle}
if this is \texttt{true}, \texttt{wd} is smaller than 0, and \texttt{wd*} is smaller than 0pt the \langle\textit{message}\rangle will be as wide as the \langle\textit{animal}\rangle’s body. Note that because the \langle\textit{animal}\rangle bodies contain white space on their left end and due to the additional horizontal bubble delimiters the bubble will be wider than the \langle\textit{animal}\rangle’s body. If the \texttt{none} option was also used this option has no effect.
2.5 Dependencies
The package depends on the two packages xparse and l3keys2e and all of their dependencies. Version 2 additionally depends on array and grabbox.

2.6 Available Animals
The following animals are provided by this package. I did not create them (but altered some), they belong to their original creators.
2.7 License and Bug Reports

This work may be distributed and/or modified under the conditions of the \LaTeX\ Project Public License (LPPL), either version 1.3c of this license or (at your option) any later version. The latest version of this license is in the file: http://www.latex-project.org/lppl.txt

The package is hosted on https://github.com/Skillmon/ltx_ducksay, you might report bugs there.
3 Implementation

\(*\text{pkg}\*)

3.1 Shared between versions

3.1.1 Variables

3.1.1.1 Integers

\begin{verbatim}
\int_new:N \l_ducksay_msg_width_int
\int_new:N \l_ducksay_msg_height_int
\int_new:N \l_ducksay_tail_symbol_count_int
\end{verbatim}

3.1.1.2 Sequences

\begin{verbatim}
\seq_new:N \l_ducksay_msg_lines_seq
\seq_new:N \l_ducksay_defined_animals_seq
\end{verbatim}

3.1.1.3 Token lists

\begin{verbatim}
\tl_new:N \l_ducksay_align_tl
\tl_new:N \l_ducksay_msg_align_tl
\tl_new:N \l_ducksay_animal_tl
\tl_new:N \l_ducksay_body_tl
\tl_new:N \l_ducksay_bubble_tl
\tl_new:N \l_ducksay_tmpa_tl
\tl_new:N \l_ducksay_tail_symbol_out_one_tl
\tl_new:N \l_ducksay_tail_symbol_out_two_tl
\tl_new:N \l_ducksay_tail_symbol_in_tl
\end{verbatim}

3.1.1.4 Boolean

\begin{verbatim}
\bool_new:N \l_ducksay_version_one_bool
\bool_new:N \l_ducksay_version_two_bool
\bool_new:N \l_ducksay_random_animal_bool
\end{verbatim}

3.1.1.5 Boxes

\begin{verbatim}
\box_new:N \l_ducksay_tmpa_box
\end{verbatim}

3.1.2 Regular Expressions

Regular expressions for AddColoredAnimal

\begin{verbatim}
\regex_const:Nn \c_ducksay_textcolor_regex
{ \cO(?:\textcolor{\{.*\}\}{.*}) }
\regex_const:Nn \c_ducksay_color_delim_regex
{ \cO(?:\bgroup\color{\{.*\}\}(.*))\egroup} 
\regex_const:Nn \c_ducksay_color_regex
{ \cO(?:\color{\{.*\}\})}
\end{verbatim}

3.1.3 Messages

\begin{verbatim}
\msg_new:nnn { ducksay } { load-time-only }
\msg_new:nnn { ducksay } { load-time-only }
\\{ The-"#1"-key-is-to-be-used-only-during-package-load-time. \}
\end{verbatim}

3.1.4 Key-value setup

\begin{verbatim}
\keys_define:nn { ducksay }
{ 
, bubble .tl_set:N = \l_ducksay_bubble_tl
, body .tl_set:N = \l_ducksay_body_tl
}
\exp_args:Nx \DucksayOptions
\keys_define:nn { ducksay } { default_animal .meta:n = { #1 } }
\keys_define:nn { ducksay } { animal .meta:n = { duck } }
\keys_define:nn { ducksay } { msg-align .meta:n = { \l_ducksay_msg_align_tl } }
\keys_define:nn { ducksay } { rel-align .meta:n = { \l_ducksay_rel_align_tl } }
\keys_define:nn { ducksay } { ligatures .meta:n = { \l_ducksay_ligatures_tl } }
\keys_define:nn { ducksay } { tail-1 .meta:n = { \c_backslash_str }, tail-2 = { \c_backslash_str } }
\keys_define:nn { ducksay } { think .meta:n = { tail-1 = { - }, tail-2 = { - } } }
\keys_define:nn { ducksay } { random .meta:n = { tail-1 = { 0 }, tail-2 = { o } } }
\keys_define:nn { ducksay } { no-tail .meta:n = { tail-1 = { ~ }, tail-2 = { ~ } } }
\keys_define:nn { ducksay } { schroedinger .meta:n = { tail-1 = { O }, tail-2 = { o } } }
\keys_define:nn { ducksay } { version .meta:n = { version / 1 = { \l_ducksay_version_one_bool }, version / 2 = { \l_ducksay_version_two_bool } } }
\keys_define:nn { ducksay } { version .meta:n = { version .choice: version / 1 .code:n = \bool_set_false:N \l_ducksay_version_two_bool \bool_set_true:N \l_ducksay_version_one_bool version / 2 .code:n = \bool_set_false:N \l_ducksay_version_one_bool \bool_set_true:N \l_ducksay_version_two_bool version .initial:n = 2 } }
\ProcessKeysOptions { ducksay }

Undefine the load-time-only keys
3.1.4.1 Keys for \AddAnimal

Define keys meant for \AddAnimal and \AddColoredAnimal only in their own regime:

\keys_define:nn { ducksay / add-animal }
{
  ,tail-symbol .code:n = \tl_set:Nx \l_ducksay_tail_symbol_in_tl { \tl_to_str:n { #1 } }
  ,tail-symbol .initial:o = \c_backslash_str
  ,tail-count .int_set:N = \l_ducksay_tail_symbol_count_int
  ,tail-count .initial:n = 2
}

3.1.5 Functions

3.1.5.1 Generating Variants of External Functions

\cs_generate_variant:Nn \tl_replace_once:Nnn { NVn }
\cs_generate_variant:Nn \tl_replace_all:Nnn { NVn }
\cs_generate_variant:Nn \keys_set:nn { nx }

3.1.5.2 Internal

\__ducksay_everyeof:w
\cs_set_eq:NN \__ducksay_everyeof:w \tex_everyeof:D
(End definition for \__ducksay_everyeof:w.)

\__ducksay_scantokens:w
\cs_set_eq:NN \__ducksay_scantokens:w \tex_scantokens:D
(End definition for \__ducksay_scantokens:w.)

\ducksay_replace_verb_newline:Nn
\cs_new_protected:Npx \ducksay_replace_verb_newline:Nn #1 #2
{ \tl_replace_all:Nnn #1 \c_backslash_str \c_backslash_str { #2 } }
(End definition for \ducksay_replace_verb_newline:Nn.)

\ducksay_replace_verb_newline_newline:Nn
\cs_new_protected:Npx \ducksay_replace_verb_newline_newline:Nn #1 #2
{ \tl_replace_all:Nnn { \c_backslash_str \c_backslash_str } { #2 } }
(End definition for \ducksay_replace_verb_newline_newline:Nn.)
\ducksay_process_verb_newline:nnn
\cs_new_protected:Npn \ducksay_process_verb_newline:nnn #1 #2 #3
{\tl_set:Nn \ProcessedArgument { #3 } \ducksay_replace_verb_newline_newline:Nn \ProcessedArgument { #2 } \ducksay_replace_verb_newline:Nn \ProcessedArgument { #1 } }

(End definition for \ducksay_process_verb_newline:nnn.)

\ducksay_add_animal_inner:nnnn
\cs_new_protected:Npn \ducksay_add_animal_inner:nnnn #1 #2 #3 #4
{\group_begin: \keys_set:nn { ducksay / add-animal } { #1 } \tl_set:Nn \l_ducksay_tmpa_tl { \ #3 } \int_compare:nNnTF { \l_ducksay_tail_symbol_count_int } < { \c_zero_int } { \tl_replace_once:NVn \l_ducksay_tmpa_tl \l_ducksay_tail_symbol_in_tl \l_ducksay_tail_symbol_out_one_tl \tl_replace_all:NVn \l_ducksay_tmpa_tl \l_ducksay_tail_symbol_in_tl \l_ducksay_tail_symbol_out_two_tl } { \int_compare:nNnT { \l_ducksay_tail_symbol_count_int } > { \c_zero_int } { \tl_replace_once:NVn \l_ducksay_tmpa_tl \l_ducksay_tail_symbol_in_tl \l_ducksay_tail_symbol_out_one_tl \int_step_inline:nnn { 2 } { \l_ducksay_tail_symbol_count_int } { \tl_replace_once:NVn \l_ducksay_tmpa_tl \l_ducksay_tail_symbol_in_tl \l_ducksay_tail_symbol_out_two_tl } } } \tl_map_inline:Nn \l_ducksay_ligatures_tl { \tl_replace_all:Nnn \l_ducksay_tmpa_tl { ##1 } { { ##1 } } } \ducksay_replace_verb_newline:Nn \l_ducksay_tmpa_tl \l_ducksay_ligatures_tl \exp_args:NNnV \group_end: \tl_set:cn { l_ducksay_animal_#2_tl } \l_ducksay_tmpa_tl \exp_args:Nnx \keys_define:nn { ducksay } { #2 .code:n =

```latex
\begin{tabular}{|c|}
\hline
\hline
\hline
\hline
\hline
\hline
\end{tabular}
```

\begin{quote}
\begin{verbatim}
\ldots
\end{verbatim}
\end{quote}
\{ \exp_not:n \{ \tl_set_eq:n \l_ducksay_animal_tl \} \\
\exp_not:c \{ \l_ducksay_animal_#2_tl \} \\
\exp_not:n \{ \exp_args:N \DucksayOptions \} \\
\exp_not:c \{ \l_ducksay_animal_#2_options_tl \} \}
\tl_if_exist:cF \{ \l_ducksay_animal_#2_options_tl \} \\
\{ \tl_new:c \{ \l_ducksay_animal_#2_options_tl \} \}
\IfBooleanT \{ #4 \} \\
\{ \keys_define:nn \{ ducksay \} \{ default_animal \}.meta:n = \{ #2 \} \} \\
\seq_if_in:NnF \l_ducksay_defined_animals_seq \{ #2 \} \\
\{ \seq_push:Nn \l_ducksay_defined_animals_seq \{ #2 \} \}
\cs_generate_variant:Nn \ducksay_add_animal_inner:nnnn \{ nnVn \}
(End definition for \ducksay_add_animal_inner:nnnn.)
\ducksay_default_or_random_animal:
\cs_new_protected:Npn \ducksay_default_or_random_animal: \\
\{ \tl_if_empty:NT \l_ducksay_animal_tl \\
\boolexpr:NTF \l_ducksay_random_animal_bool \\
\{ \keys_set:nx \{ ducksay \} \{ \seq_rand_item:N \l_ducksay_defined_animals_seq \} \} \\
\{ \keys_set:nn \{ ducksay \} \{ default_animal \} \} \}
(End definition for \ducksay_default_or_random_animal:)

3.1.5.3 Document level
\DefaultAnimal
\NewDocumentCommand \DefaultAnimal \{ m \} \\
\keys_define:nn \{ ducksay \} \{ default_animal \}.meta:n = \{ #1 \} \}
(End definition for \DefaultAnimal. This function is documented on page 2.)
\DucksayOptions
\NewDocumentCommand \DucksayOptions \{ m \} \\
\keys_set:nn \{ ducksay \} \{ #1 \} 
(End definition for \DucksayOptions. This function is documented on page 3.)
\AddAnimal
\NewDocumentCommand \AddAnimal { s O{} m +v }
{\ducksay_add_animal_inner:nnnn { #2 } { #3 } { #4 } { #1 }}

(End definition for \AddAnimal. This function is documented on page 3.)

\AddColoredAnimal
\NewDocumentCommand \AddColoredAnimal { s O{} m +v }
{\tl_set:Nn \l_ducksay_tmpa_tl { #4 }\regex_replace_all:NnN \c_ducksay_color_delim_regex
{ \c{bgroup}\c{color}\cB\{1\cE\}\2\c{egroup} }\l_ducksay_tmpa_tl\regex_replace_all:NnN \c_ducksay_color_regex
{ \c{color}\cB\{1\cE\} } \l_ducksay_tmpa_tl\regex_replace_all:NnN \c_ducksay_textcolor_regex
{ \c{textcolor}\cB\{1\cE\}\2\cE\} }\l_ducksay_tmpa_tl\ducksay_add_animal_inner:nnVn { #2 } { #3 } \l_ducksay_tmpa_tl { #1 }}

(End definition for \AddColoredAnimal. This function is documented on page 3.)

\AnimalOptions
\NewDocumentCommand \AnimalOptions { s m m }
{\tl_if_exist:cTF { l_ducksay_animal_#2_options_tl }
{ \IfBooleanTF { #1 }
{ \tl_set:cn }
{ \tl_put_right:cn }
} \tl_set:cn { l_ducksay_animal_#2_options_tl } { #3, }
}\tl_set:cn { l_ducksay_animal_#2_options_tl } { #3, }

(End definition for \AnimalOptions. This function is documented on page 3.)

3.1.6 Load the Correct Version and the Animals
\bool_if:NT \l_ducksay_version_one_bool
{ \file_input:n { ducksay.code.v1.tex } }
\bool_if:NT \l_ducksay_version_two_bool
{ \file_input:n { ducksay.code.v2.tex } }
\ExplSyntaxOff
\input{ducksay.animals.tex}

(/pkg)
3.2 Version 1

\begin{verbatim}
\providefile{ducksay.code.v1.tex}
[ducksay@date\space v\ducksay@version\space ducksay code version 1]
\end{verbatim}

3.2.1 Functions

3.2.1.1 Internal

\ducksay_longest_line:n
Calculate the length of the longest line
\begin{verbatim}
\cs_new:Npn \ducksay_longest_line:n #1
{
  \int_incr:N \l_ducksay_msg_height_int
  \exp_args:NNx \tl_set:Nn \l_ducksay_tmpa_tl { #1 }
  \regex_replace_all:nnN { \s } { \c { space } } \l_ducksay_tmpa_tl
  \int_set:Nn \l_ducksay_msg_width_int
  { \int_max:nn { \l_ducksay_msg_width_int } { \tl_count:N \l_ducksay_tmpa_tl }
  }
}
\end{verbatim}

(End definition for \ducksay_longest_line:n.)

\ducksay_open_bubble:
Draw the opening bracket of the bubble
\begin{verbatim}
\cs_new:Npn \ducksay_open_bubble:
{
  \begin{tabular}{@{}l@{}}
    \null\\null
    \int_compare:nNnTF { \l_ducksay_msg_height_int } = { 1 } { ( }
    { /\int_step_inline:nnn { 3 } { \l_ducksay_msg_height_int } { \\kern-0.2em| }
      \detokenize{ }\null
    \end{tabular}
}
\end{verbatim}

(End definition for \ducksay_open_bubble.)

\ducksay_close_bubble:
Draw the closing bracket of the bubble
\begin{verbatim}
\cs_new:Npn \ducksay_close_bubble:
{
  \begin{tabular}{@{}l@{}}
    _\int_step_inline:nnn { 2 } { \l_ducksay_msg_height_int } { \ } \
    \mbox { - }
  \end{tabular}
}\end{verbatim}

(End definition for \ducksay_close_bubble.)
Implementation of Version 1

\begin{tabular}{@{}r@{}}
\null\
\int_compare:nNnTF { \l_ducksay_msg_height_int } = { 1 }\
{ ) }\
{ }
\detokenize { \ }
\int_step_inline:nnn\
{ 3 } { \l_ducksay_msg_height_int } { \lll \kern-0.2em }
\lll/\
\lll[-1ex]\null\
\end{tabular}

(End definition for \ducksay_close_bubble:.)

\ducksay_print_msg:nn
Print out the message

\cs_new:Npn \ducksay_print_msg:nn #1 #2
{\begin{tabular}{@{} #2 @{}}
\int_step_inline:nn { \l_ducksay_msg_width_int } { _ } \\
#1\lll[-1ex] \\
\int_step_inline:nn { \l_ducksay_msg_width_int } { { - } } \\
\end{tabular}}

\cs_generate_variant:Nn \ducksay_print_msg:nn { nV }
(End definition for \ducksay_print_msg:nn.)

\ducksay_print:nn
Print out the whole thing

\cs_new:Npn \ducksay_print:nn #1 #2
{\int_compare:nNnTF { \l_ducksay_msg_width_int } < { 0 }\
{ \int_zero:N \l_ducksay_msg_height_int \\
\seq_set_split:Nnn \l_ducksay_msg_lines_seq { \ } { #1 } \\
\seq_map_function:NN \l_ducksay_msg_lines_seq \ducksay_longest_line:n \\
}\
{ \int_compare:nNnT { \l_ducksay_msg_height_int } < { 0 }\
{ \regex_count:nnN { \c { \ } } { #1 } \l_ducksay_msg_height_int \\
\int_incr:N \l_ducksay_msg_height_int \\
}\
}\
}\group_begin:
\frenchspacing
\verbatim@font
\@noligs
\begin{tabular}{\l_ducksay_align_tl}{@{}#2@{}}
\l_ducksay_bubble_tl \\
\ducksay_open_bubble: \\
\ducksay_print_msg:nnV { #1 } \l_ducksay_msg_align_tl \\
\ducksay_close_bubble: \\
\null}
\ducksay 
\NewDocumentCommand \ducksay { O{} m } 
{ 
    \ducksay_say_and_think:nn { #1 } { #2 } 
}

(End definition for \ducksay. This function is documented on page 8.)

\duckthink 
\NewDocumentCommand \duckthink { O{} m } 
{ 
    \ducksay_say_and_think:nn { think, #1 } { #2 } 
}

(End definition for \duckthink. This function is documented on page 8.)

\duckthink

---

(End definition for \ducksay_print:nn.)

\ducksay_say_and_think:nn Reset some variables
\cs_new:Npn \ducksay_say_and_think:nn #1 #2
{ 
    \group_begin:
    \int_set:Nn \l_ducksay_msg_width_int { \c_max_int }
    \int_set:Nn \l_ducksay_msg_height_int { \c_max_int }
    \keys_set:nn { ducksay } { #1 }
    \ducksay_default_or_random_animal:
    \ducksay_print:nV { #2 } \l_ducksay_rel_align_tl
    \group_end:
}

(End definition for \ducksay_say_and_think:nn.)

3.2.1.2 Document level

\ducksay
\NewDocumentCommand \ducksay { O{} m } 
{ 
    \ducksay_say_and_think:nn { #1 } { #2 } 
}

(End definition for \ducksay_print:nn.)
3.3 Version 2

\ProvidesFile{ducksay.code.v2.tex}
\ducksay@date v\ducksay@version ducksay code version 2

Load the additional dependencies of version 2.
\RequirePackage{array,grabbox}

3.3.1 Messages
\msg_new:nnn { ducksay } { justify~unavailable }
\msg_new:nnn { ducksay } { unknown~message~alignment }
\msg_new:nnn { ducksay } { v1-key-only }
\msg_new:nnn { ducksay } { zero-baselineskip }

3.3.2 Variables
3.3.2.1 Token Lists
\tl_new:N \l_ducksay_msg_align_vbox_tl

3.3.2.2 Boxes
\box_new:N \l_ducksay_msg_box

3.3.2.3 Bools
\bool_new:N \l_ducksay_eat_arg_box_bool
\bool_new:N \l_ducksay_eat_arg_tab_verb_bool
\bool_new:N \l_ducksay_mirrored_body_bool
\bool_new:N \l_ducksay_msg_eq_body_width_bool

3.3.2.4 Coffins
\coffin_new:N \l_ducksay_body_coffin
\coffin_new:N \l_ducksay_bubble_close_coffin
\coffin_new:N \l_ducksay_bubble_open_coffin
\coffin_new:N \l_ducksay_bubble_top_coffin
\coffin_new:N \l_ducksay_msg_coffin

3.3.2.5 Dimensions
\dim_new:N \l_ducksay_hpad_dim
\dim_new:N \l_ducksay_bubble_bottom_kern_dim
\dim_new:N \l_ducksay_bubble_top_kern_dim
\dim_new:N \l_ducksay_msg_width_dim

3.3.3 Options
\keys_define:nn { ducksay }
\{ ,arg .choice:
Implementation of Version 2

\begin{verbatim}
\verbatim{\begin{quote}
\verbatim{\begin{verbatim}
,\verb+/\ arg / box .\verb+/\ code:n = \bool_set_true:N \l_ducksay_eat_arg_box_bool
,\verb+/\ arg / tab .\verb+/\ code:n = 
{ \bool_set_false:N \l_ducksay_eat_arg_box_bool
 \bool_set_false:N \l_ducksay_eat_arg_tab_verb_bool
 }
,\verb+/\ arg / tab* .\verb+/\ code:n = 
{ \bool_set_false:N \l_ducksay_eat_arg_box_bool
 \bool_set_true:N \l_ducksay_eat_arg_tab_verb_bool
 }
,\verb+/\ arg .\verb+/\ initial:n = tab
,\verb+/\ wd* .\verb+/\ dim_set:N = \l_ducksay_msg_width_dim
,\verb+/\ wd*.\verb+/\ initial:n = -\c_max_dim
,\verb+/\ wd*.\verb+/\ value_required:n = true
,\verb+/\ wd-eq-body .\verb+/\ bool_set:N = \l_ducksay_msg_eq_body_width_bool
,\verb+/\ none .\verb+/\ bool_set:N = \l_ducksay_no_body_bool
,\verb+/\ no-bubble .\verb+/\ bool_set:N = \l_ducksay_no_bubble_bool
,\verb+/\ body-mirrored .\verb+/\ bool_set:N = \l_ducksay_mirrored_body_bool
,\verb+/\ ignore-body .\verb+/\ bool_set:N = \l_ducksay_ignored_body_bool
,\verb+/\ body-x .\verb+/\ dim_set:N = \l_ducksay_body_x_offset_dim
,\verb+/\ body-x .\verb+/\ value_required:n = true
,\verb+/\ body-y .\verb+/\ dim_set:N = \l_ducksay_body_y_offset_dim
,\verb+/\ body-y .\verb+/\ value_required:n = true
,\verb+/\ body-to-msg .\verb+/\ tl_set:N = \l_ducksay_body_to_msg_align_body_tl
,\verb+/\ msg-to-body .\verb+/\ tl_set:N = \l_ducksay_body_to_msg_align_msg_tl
,\verb+/\ body-align .\verb+/\ choice:
,\verb+/\ body-align / l .\verb+/\ meta:n = { body-to-msg = l , msg-to-body = l }
,\verb+/\ body-align / c .\verb+/\ meta:n = { body-to-msg = hc , msg-to-body = hc }
,\verb+/\ body-align / r .\verb+/\ meta:n = { body-to-msg = r , msg-to-body = r }
,\verb+/\ body-align .\verb+/\ initial:n = l
,\verb+/\ body-bigger .\verb+/\ int_set:N = \l_ducksay_body_bigger_int
,\verb+/\ body-bigger .\verb+/\ initial:n = \c_zero_int
,\verb+/\ msg-align .\verb+/\ choice:
,\verb+/\ msg-align / l .\verb+/\ code:n = { \tl_set:Nn \l_ducksay_msg_align_tl { l } }
,\verb+/\ msg-align / c .\verb+/\ code:n = { \tl_set:Nn \l_ducksay_msg_align_tl { c } }
,\verb+/\ msg-align / r .\verb+/\ code:n = { \tl_set:Nn \l_ducksay_msg_align_tl { r } }
,\verb+/\ msg-align / j .\verb+/\ code:n = { \tl_set:Nn \l_ducksay_msg_align_tl { j } }
,\verb+/\ msg-align-l .\verb+/\ tl_set:N = \l_ducksay_msg_align_l_tl
,\verb+/\ msg-align-l .\verb+/\ initial:n = \raggedright
,\verb+/\ msg-align-c .\verb+/\ tl_set:N = \l_ducksay_msg_align_c_tl
,\verb+/\ msg-align-c .\verb+/\ initial:n = \centering
,\verb+/\ msg-align-r .\verb+/\ tl_set:N = \l_ducksay_msg_align_r_tl
,\verb+/\ msg-align-r .\verb+/\ initial:n = \raggedleft
,\verb+/\ msg-align-j .\verb+/\ tl_set:N = \l_ducksay_msg_align_j_tl
,\verb+/\ msg-align-j .\verb+/\ initial:n = { }
,\verb+/\ out-h .\verb+/\ tl_set:N = \l_ducksay_output_h_pole_tl
,\verb+/\ out-h .\verb+/\ initial:n = l
,\verb+/\ out-v .\verb+/\ tl_set:N = \l_ducksay_output_v_pole_tl
,\verb+/\ out-v .\verb+/\ initial:n = vc
,\verb+/\ out-x .\verb+/\ dim_set:N = \l_ducksay_output_x_offset_dim
,\verb+/\ out-x .\verb+/\ value_required:n = true
,\verb+/\ out-y .\verb+/\ dim_set:N = \l_ducksay_output_y_offset_dim
,\verb+/\ out-y .\verb+/\ value_required:n = true
\verbatim}
\end{quote}
\end{verbatim}}
\end{verbatim}
\end{quote}
Implementation of Version 2

\begin{verbatim}
\clist_map_inline:nn { align, rel-align }
{\keys_define:nn { ducksay }
   \tl_set:Nn = \l_ducksay_body_fount_tl
   \tl_set:Nn = \l_ducksay_msg_fount_tl
   \tl_set:Nn = \l_ducksay_bubble_fount_tl
   \initial:n = \verbatim@font
   \msg* .tl_set:N = \hammer
   \MSG .meta:n = { msg = #1 , bubble = #1 }
   \MSG* .meta:n = { msg* = #1 , bubble* = #1 }
   \int_set:N = \l_ducksay_hpad_int
   \initial:n = 2
   \value_required:n = true
   \int_set:N = \l_ducksay_vpad_int
   \value_required:n = true
   \tl_set:N = \l_ducksay_msg_tabular_column_tl
   \tl_set:N = \l_ducksay_bubble_top_kern_tl
   \initial:n = \c_backslash_str
   \int_set:N = \l_ducksay_bubble_delim_top_tl
   \int_set:N = \l_ducksay_bubble_delim_left_tl
   \int_set:N = \l_ducksay_bubble_delim_right_tl
   \int_set:N = \l_ducksay_msg_strip_spaces_bool
}
\keys_define:nn { ducksay }
\end{verbatim}

Redefine keys only intended for version 1 to throw an error:

\begin{verbatim}
\clist_map_inline:nn { align, rel-align }
{\keys_define:nn { ducksay }
   \initial:n = { { - } }
   \int_set:N = \l_ducksay_bubble_delim_left_tl
   \int_set:N = \l_ducksay_bubble_delim_right_tl
   \int_set:N = \l_ducksay_msg_strip_spaces_bool
}
\keys_define:nn { ducksay }
\end{verbatim}
Implementation of Version 2

3.3.4 Functions

3.3.4.1 Internal

\ducksay_evaluate_message_alignment_fixed_width_common:
\begin{verbatim}
cs_new:Npn \ducksay_evaluate_message_alignment_fixed_width_common:
\{
\str_case:Vn \l_ducksay_msg_align_tl
\{
  { l } { \exp_not:N \l_ducksay_msg_align_l_tl }
  { c } { \exp_not:N \l_ducksay_msg_align_c_tl }
  { r } { \exp_not:N \l_ducksay_msg_align_r_tl }
  { j } { \exp_not:N \l_ducksay_msg_align_j_tl }
\}
\}
\end{verbatim}

(End definition for \ducksay_evaluate_message_alignment_fixed_width_common:)

\ducksay_evaluate_message_alignment_fixed_width_tabular:
\begin{verbatim}
cs_new:Npn \ducksay_evaluate_message_alignment_fixed_width_tabular:
\{
\tl_if_empty:NT \l_ducksay_msg_tabular_column_tl
\{
  \tl_set:Nx \l_ducksay_msg_tabular_column_tl
  { >
    \ducksay_evaluate_message_alignment_fixed_width_common:
    \exp_not:N \arraybackslash
  }
  p { \exp_not:N \l_ducksay_msg_width_dim }
\}
\}
\end{verbatim}

(End definition for \ducksay_evaluate_message_alignment_fixed_width_tabular:)

\ducksay_evaluate_message_alignment_fixed_width_vbox:
\begin{verbatim}
cs_new:Npn \ducksay_evaluate_message_alignment_fixed_width_vbox:
\{
\tl_set:Nx \l_ducksay_msg_align_vbox_tl
  { \ducksay_evaluate_message_alignment_fixed_width_common: }
\}
\end{verbatim}

(End definition for \ducksay_evaluate_message_alignment_fixed_width_vbox:)

\ducksay_calculate_msg_width_from_int:
\begin{verbatim}
cs_new:Npn \ducksay_calculate_msg_width_from_int:
\{
\hbox_set:Nn \l_ducksay_tmpa_box { { \l_ducksay_msg_fount_tl M } }
\dim_set:Nn \l_ducksay_msg_width_dim
\{
  \l_ducksay_msg_width_int \box_wd:N \l_ducksay_tmpa_box
\}
\}
\end{verbatim}
Implementation of Version 2

(End definition for \ducksay_calculate_msg_width_from_int:)

\ducksay_msg_tabular_begin:
\cs_new:Npn \ducksay_msg_tabular_begin:
{ \ducksay_msg_tabular_begin_inner:V \l_ducksay_msg_tabular_column_tl }
\cs_new:Npn \ducksay_msg_tabular_begin_inner:n #1
{ \begin { tabular } { @{} #1 @{} } }
\cs_generate_variant:Nn \ducksay_msg_tabular_begin_inner:n { V }
\ducksay_msg_tabular_end:
\cs_new:Npn \ducksay_msg_tabular_end:
{ \end { tabular } }
\ducksay_width_case_none_int_dim:nnn
\cs_new:Npn \ducksay_width_case_none_int_dim:nnn #1 #2 #3
{ \dim_compare:nNnTF { \l_ducksay_msg_width_dim } < { \c_zero_dim } { \int_compare:nNnTF { \l_ducksay_msg_width_int } < { \c_zero_int } { #1 } { #2 } } { #3 } }
\ducksay_digest_options:n
\cs_new:Npn \ducksay_digest_options:n #1
{ \group_begin: \keys_set:nn { ducksay } { #1 } \ducksay_default_or_random_animal: \bool_if:NF \l_ducksay_no_body_bool \{ \hcoffin_set:Nn \l_ducksay_body_coffin \{ \frenchspacing \l_ducksay_body_fount_tl \begin{tabular} { @{} l @{} } \l_ducksay_animal_tl \ducksay_make_body_bigger: \relax \end{tabular} \} }

(End definition for \ducksay_width_case_none_int_dim:nnn.)
\bool_if:NT \l_ducksay_msg_eq_body_width_bool
  {
    \bool_lazy_and:nnT
    { \int_compare_p:nNn \l_ducksay_msg_width_int < \c_zero_int }
    { \dim_compare_p:nNn \l_ducksay_msg_width_dim < \c_zero_dim }
    {
      \dim_set:Nn \l_ducksay_msg_width_dim
      { \coffin_wd:N \l_ducksay_body_coffin }
    }
  }

\bool_if:NTF \l_ducksay_eat_arg_box_bool
  {
    \ducksay_width_case_none_int_dim:nnn
    { \ducksay_eat_argument_hbox:w }
    { \ducksay_calculate_msg_width_from_int:
      \ducksay_eat_argument_vbox:w }
    { \ducksay_eat_argument_vbox:w }
  }

{ \ducksay_width_case_none_int_dim:nnn
  { \tl_if_empty:NT \l_ducksay_msg_tabular_column_tl
    {\str_case:Vn \l_ducksay_msg_align_tl
     { l } \{ \tl_set:Nn \l_ducksay_msg_tabular_column_tl { l } \}
     { c } \{ \tl_set:Nn \l_ducksay_msg_tabular_column_tl { c } \}
     { r } \{ \tl_set:Nn \l_ducksay_msg_tabular_column_tl { r } \}
     { j }
     { \tl_error:nn { ducksay } { justify-unavailable } }
     \tl_set:Nn \l_ducksay_msg_tabular_column_tl { l }
    }
    { \ducksay_calculate_msg_width_from_int:
      \ducksay_evaluate_message_alignment_fixed_width_tabular:
    }
  }
  \ducksay_eat_argument_tabular:w
}

(End definition for \ducksay_digest_options:n.)

\ducksay_set_bubble_top_kern:
\cs_new:Npn \ducksay_set_bubble_top_kern:
  {
    \group_begin:
      ( 31 )
      \_\_\_\_\_\_\_
      \_\_\_\_\_\_\_\_
      \_\_\_\_\_\_\_\_\_\_\_\_\_
      \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
    \group_end:
Implementation of Version 2

```latex
\l_ducksay_bubble_fount_tl
\exp_args:NNNx
\group_end:
\dim_set:Nn \l_ducksay_bubble_top_kern_dim
  \{ \dim_eval:n \{ \l_ducksay_bubble_top_kern_tl \} \}
\}
(End definition for \ducksay_set_bubble_top_kern:.)
\ducksay_set_bubble_bottom_kern:
\cs_new:Npn \ducksay_set_bubble_bottom_kern:
  \group_begin:
  \l_ducksay_bubble_fount_tl
  \exp_args:NNNx
  \group_end:
  \dim_set:Nn \l_ducksay_bubble_bottom_kern_dim
  \{ \dim_eval:n \{ \l_ducksay_bubble_bottom_kern_tl \} \}
  
(End definition for \ducksay_set_bubble_bottom_kern:.)
\ducksay_make_body_bigger:
\cs_new:Npn \ducksay_make_body_bigger:
  \{ \prg_replicate:nn \l_ducksay_body_bigger_int \}
(End definition for \ducksay_make_body_bigger:.)
\ducksay_baselineskip:
This is an overly cautious way to get the current baselineskip. Inside of \texttt{tabular} the baselineskip is \texttt{0pt}, so we fall back to \texttt{\normalbaselineskip}, or issue an error and fall back to some arbitrary value not producing an error if that one is also \texttt{0pt}.
\cs_new_protected_nopar:Npn \ducksay_baselineskip:
  \the\dimexpr\ifdim\baselineskip=\c_zero_dim
    \ifdim\normalbaselineskip=\c_zero_dim
      \msg_expandable_error:nn { ducksay } { zero-baselineskip } { 12pt }
      12pt
    \else
      \normalbaselineskip
    \fi
  \else
    \baselineskip
  \fi
  \relax
(End definition for \ducksay_baselineskip:.)
\ducksay_measure_msg:
\cs_new_protected_nopar:Npn \ducksay_measure_msg:
  \{ \hbox_set:Nn \l_ducksay_tempa_box
    \l_ducksay_bubble_fount_tl \l_ducksay_bubble_delim_top_tl
  \}
(End definition for \ducksay_measure_msg:.)
```

---

\text{Implementation of Version 2}

\l_ducksay_bubble_fount_tl
\exp_args:NNNx
\group_end:
\dim_set:Nn \l_ducksay_bubble_top_kern_dim
  \{ \dim_eval:n \{ \l_ducksay_bubble_top_kern_tl \} \}
\}
(End definition for \ducksay_set_bubble_top_kern:.)
\ducksay_set_bubble_bottom_kern:
\cs_new:Npn \ducksay_set_bubble_bottom_kern:
  \group_begin:
  \l_ducksay_bubble_fount_tl
  \exp_args:NNNx
  \group_end:
  \dim_set:Nn \l_ducksay_bubble_bottom_kern_dim
  \{ \dim_eval:n \{ \l_ducksay_bubble_bottom_kern_tl \} \}
  
(End definition for \ducksay_set_bubble_bottom_kern:.)
\ducksay_make_body_bigger:
\cs_new:Npn \ducksay_make_body_bigger:
  \{ \prg_replicate:nn \l_ducksay_body_bigger_int \}
(End definition for \ducksay_make_body_bigger:.)
\ducksay_baselineskip:
This is an overly cautious way to get the current baselineskip. Inside of \texttt{tabular} the baselineskip is \texttt{0pt}, so we fall back to \texttt{\normalbaselineskip}, or issue an error and fall back to some arbitrary value not producing an error if that one is also \texttt{0pt}.
\cs_new_protected_nopar:Npn \ducksay_baselineskip:
  \the\dimexpr\ifdim\baselineskip=\c_zero_dim
    \ifdim\normalbaselineskip=\c_zero_dim
      \msg_expandable_error:nn { ducksay } { zero-baselineskip } { 12pt }
      12pt
    \else
      \normalbaselineskip
    \fi
  \else
    \baselineskip
  \fi
  \relax
(End definition for \ducksay_baselineskip:.)
\ducksay_measure_msg:
\cs_new_protected_nopar:Npn \ducksay_measure_msg:
  \{ \hbox_set:Nn \l_ducksay_tempa_box
    \l_ducksay_bubble_fount_tl \l_ducksay_bubble_delim_top_tl
  \}
(End definition for \ducksay_measure_msg:.)
implementation of Version 2

\{ \fp_eval:n
  \{
    ceil
    \( \\text{box wd:N} \ l\_ducksay\_msg\_box / \text{box wd:N} \ l\_ducksay\_tmpa\_box \) 
  \}
\}

\group_begin:
\l\_ducksay\_bubble\_fount\_tl
\exp_args:NNNx
\group_end:
\int_set:Nn \l\_ducksay\_msg\_height\_int
\{
  \int_max:nn
  \{
    \fp_eval:n
    \{
      ceil
      \( \\text{box ht:N} \ l\_ducksay\_msg\_box
        + \\text{box dp:N} \ l\_ducksay\_msg\_box
      \) 
      \( / ( \text{arraystretch} \ast \text{ducksay\_baselineskip:} ) \)
    \}
    + \\l\_ducksay\_vpad\_int
  \}
  \{ \l\_ducksay\_msg\_height\_int \}
\}
\}

(End definition for \ducksay\_measure\_msg:.)

\ducksay\_set\_bubble\_coffins:

\cs_new_protected_nopar:Npn \ducksay\_set\_bubble\_coffins:
\{
  \hcoffin_set:Nn \l\_ducksay\_bubble\_open\_coffin
  \{ \l\_ducksay\_bubble\_fount\_tl
  \begin{tabular}{@{}l@{}}
    \int_compare:nNnTF \{ \l\_ducksay\_msg\_height\_int \} = \{ \c\_one\_int \}
    \{ \l\_ducksay\_bubble\_delim\_left\_a\_tl
  \}
  \}
  \{ \l\_ducksay\_bubble\_delim\_left\_b\_tl\}
  \int_step_inline:nnn
  \{ 3 \} \{ \l\_ducksay\_msg\_height\_int \}
  \{
    \kern\l\_ducksay\_bubble\_side\_kern\_tl
    \l\_ducksay\_bubble\_delim\_left\_c\_tl
  \}
\}

( 33 )
\-----------------
\ \______
\_/\________
\_.\_\_____\
\ |\ |_______
\ \ \________
Implementation of Version 2

\l_ducksay_bubble_delim_left_d_tl
\end{tabular}
\hcoffin_set:Nn \l_ducksay_bubble_close_coffin
{ \l_ducksay_bubble_fount_tl
\begin{tabular}{@{}r@{}}
\int_compare:nNnTF \l_ducksay_msg_height_int \= \c_one_int \{
\l_ducksay_bubble_delim_right_a_tl
\}
\l_ducksay_bubble_delim_right_b_tl \\
\int_step_inline:n\{3\}\\{\l_ducksay_msg_height_int \}
\{ \l_ducksay_bubble_delim_right_c_tl
\kern-\l_ducksay_bubble_side_kern_tl
\& \}
\l_ducksay_bubble_delim_right_d_tl
\end{tabular}
\hcoffin_set:Nn \l_ducksay_bubble_top_coffin
{ \l_ducksay_bubble_fount_tl
\int_step_inline:nnn{\l_ducksay_msg_width_int + \l_ducksay_hpad_int }{3}
{ \l_ducksay_bubble_delim_top_tl }
}

(End definition for \ducksay_set_bubble_coffins:.)

\ducksay_join_bubble_to_msg_coffin:
\cs_new_protected_nopar:Npn \ducksay_join_bubble_to_msg_coffin:
{ \dim_set:Nn \l_ducksay_hpad_dim
{ ( \coffin_wd:N \l_ducksay_bubble_top_coffin - \coffin_wd:N \l_ducksay_msg_coffin ) / 2 }
\coffin_join:NnnNn\{1\}{vc}\{ \l_ducksay_msg_coffin \}
\coffin_join:NnnNn\{r\}{vc}\{ - \l_ducksay_hpad_dim \}
\coffin_join:NnnNn\{r\}{vc}\{ \l_ducksay_msg_coffin \}
\coffin_join:NnnNn\{1\}{vc}\{ \l_ducksay_hpad_dim \}
( 34 )
\------
Implementation of Version 2

\coffin_join:NnnNnnnn
\l_ducksay_msg_coffin \{ hc \} \{ t \}
\l_ducksay_bubble_top_coffin \{ hc \} \{ b \}
{ \c_zero_dim } \{ \l_ducksay_bubble_top_kern_dim \}
\coffin_join:NnnNnnnn
\l_ducksay_msg_coffin \{ hc \} \{ b \}
\l_ducksay_bubble_top_coffin \{ hc \} \{ t \}
{ \c_zero_dim } \{ \l_ducksay_bubble_bottom_kern_dim \}
}

(End definition for \ducksay_join_bubble_to_msg_coffin:)

\ducksay_shipout:
\cs_new_protected:Npn \ducksay_shipout:
{\hcoffin_set:Nn \l_ducksay_msg_coffin \box_use:N \l_ducksay_msg_box }
\bool_if:NF \l_ducksay_no_bubble_bool
{ \ducksay_measure_msg: \ducksay_set_bubble_coffins: \ducksay_set_bubble_top_kern: \ducksay_set_bubble_bottom_kern: \ducksay_join_bubble_to_msg_coffin: }
\bool_if:NF \l_ducksay_no_body_bool
{ \bool_if:NT \l_ducksay_mirrored_body_bool
{ \coffin_scale:Nnn \l_ducksay_body_coffin \{ -\c_one_int \} \{ \c_one_int \}
\str_case:Vn \l_ducksay_body_to_msg_align_body_tl
{ \l_ducksay_body_to_msg_align_body_tl \{ r \} }
{ \l_ducksay_body_to_msg_align_body_tl \{ l \} }
}
\bool_if:NTF \l_ducksay_ignored_body_bool
{ \coffin_attach:NVnNVnnn }
{ \coffin_join:NVnNVnnn }
{ \l_ducksay_msg_coffin \l_ducksay_body_to_msg_align_msg_tl \l_ducksay_body_to_msg_align_body_tl \l_ducksay_body_x_offset_dim \l_ducksay_body_y_offset_dim }
\coffin_typeset:NVVnn \l_ducksay_msg_coffin \l_ducksay_output_h_pole_tl \l_ducksay_output_v_pole_tl \l_ducksay_output_x_offset_dim \l_ducksay_output_y_offset_dim }
\group_end:

(End definition for \ducksayShipout:)

3.3.4.1.1 Message Reading Functions  Version 2 has different ways of reading the message argument of \ducksay and \duckthink. They all should allow almost arbitrary content and the height and width are set based on the dimensions.
Implementation of Version 2

(End definition for \ducksay_eat_argument_vbox:w.)

\ducksay_eat_argument_vbox:w
\cs_new_protected_nopar:Npn \ducksay_eat_argument_vbox:w
  { \ducksay_evaluate_message_alignment_fixed_width_vbox:
    \bool_if:NTF \l_ducksay_msg_strip_spaces_bool
      { \@grabbox* }
      { \@grabbox }
        { \hsize \l_ducksay_msg_width_dim
          \linewidth \hsize
          \l_ducksay_msg_align_vbox_tl
          \@afterindentfalse
          \@afterheading
          }
    \l_ducksay_msg_box \l_ducksay_msg_fount_tl \vbox {} \ducksay_shipout:
  }

(End definition for \ducksay_eat_argument_vbox:w.)

3.3.4.1.2 Generating Variants of External Functions
\cs_generate_variant:Nn \coffin_join:NnnNnnn { NVnNVnnn }
\cs_generate_variant:Nn \coffin_attach:NnnNnnn { NVnNVnnn }
\cs_generate_variant:Nn \coffin_typeset:Nnnnn { NVVnn }
\cs_generate_variant:Nn \str_case:nn { Vn }

3.3.4.2 Document level
\ducksay
\NewDocumentCommand \ducksay { O{} }
  { \ducksay_digest_options:n { #1 } }

(End definition for \ducksay. This function is documented on page 8.)

\duckthink
\NewDocumentCommand \duckthink { O{} }
  { \ducksay_digest_options:n { think, #1 } }

(End definition for \duckthink. This function is documented on page 8.)
3.4 Definition of the Animals

\ProvidesFile{ducksay.animals.tex}
[%\ducksay@date\space v\ducksay@version\space ducksay animals]
\%A some of the below are from http://ascii.co.uk/art/

\AddAnimal{duck}\
\AddAnimal{small-duck}\
\AddAnimal{duck-family}\
\AddAnimal{cow}\
\AddAnimal{head-in}\
\AddAnimal{sodomized}\
\AddAnimal{tux}
```plaintext
\AddAnimal{pig} %>>=
\AddAnimal{frog} %>>=
\AddAnimal{snowman} %>>=
\AddAnimal[hedgehog] %>>=
\AddAnimal[kangaroo] %>>=
\AddAnimal[rabbit] %>>=
```


```plaintext
\AddAnimal[tail-symbol=s,tail-count=3] %>>=
```

(39)
```plaintext
```

\AddAnimal{snowman} %>>=
Who’s gonna use it anyway?