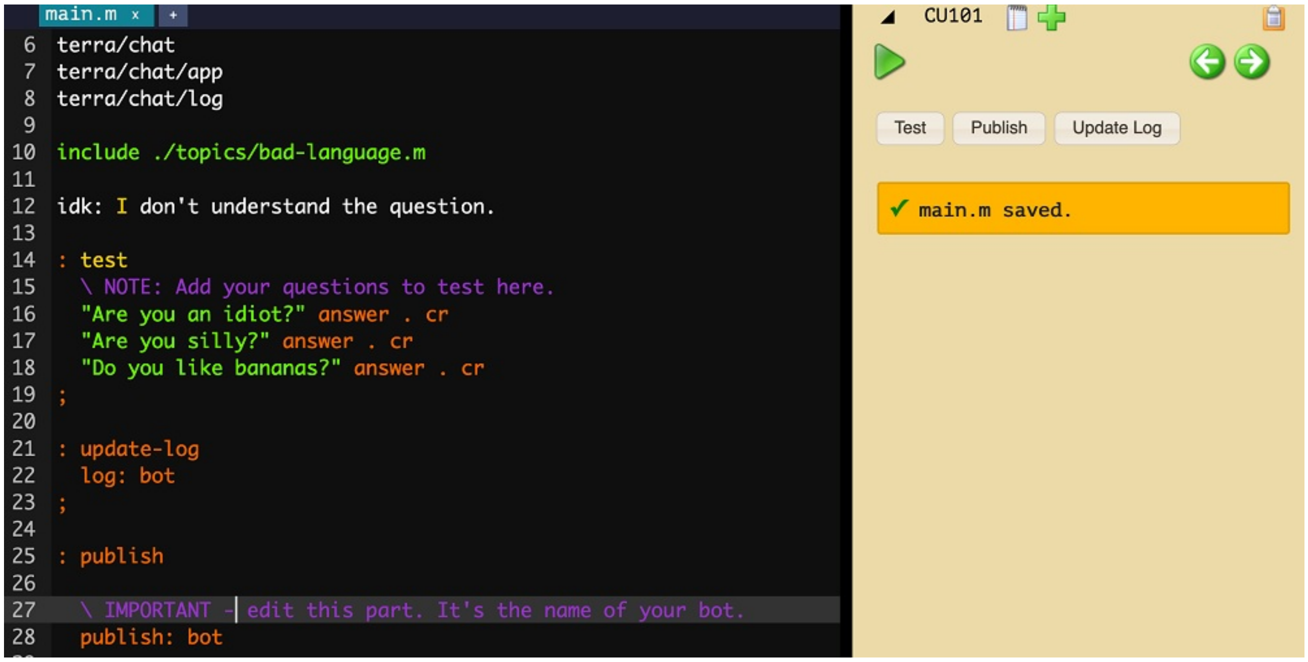
SESSION 1 : Introduction

Step 1: Click on `main.m` to open it. You should see three buttons, `Test` , `Publish` and `Update Log` appear on the right Output panel.

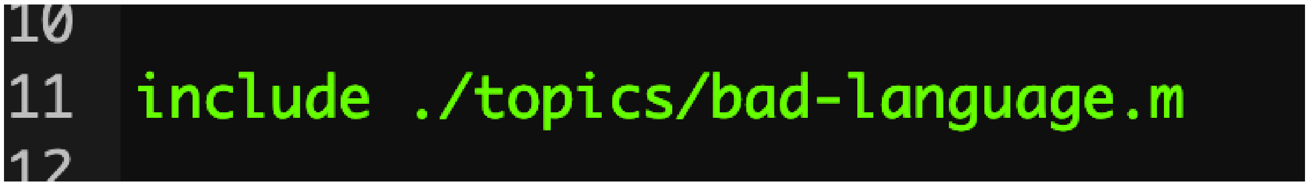


```
terra/chat
terra/chat/app
terra/chat/log

include ./topics/bad-language.m

idk: I don't understand the question.

: test
  \ NOTE: Add your questions to test here.
  "Are you an idiot?" answer . cr
  "Are you silly?" answer . cr
  "Do you like bananas?" answer . cr
;

: update-log
  log: bot
;

: publish

  \ IMPORTANT - edit this part. It's the name of your bot.
  publish: bot
```

```
✓ main.m saved.
```

Fig 1: What you should see after opening main.m

Step 2: Click on `Test` . This produces responses to each of the questions in the `test` word on the left Code panel. This chatbot uses just one topic:



```
include ./topics/bad-language.m
```

Fig 2: Using the bad-language.m topic

`bad-language.m` responds to users who put in bad language. The `include` word activates that topic. (Quiz: what happens if the user doesn't enter a bad word?)

Step 3: Click on `Publish` . This produces launches your simple chatbot to the Smojo Cloud. You can play with this bot using the URL:

https://app.smojo.org/ `USERNAME` /bot

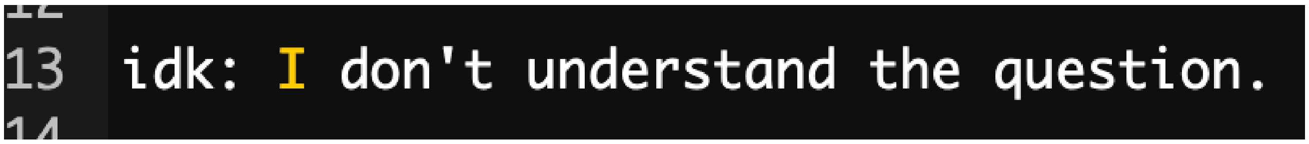
You have to replace `USERNAME` with your own username.

Further Exercises for Session 1

1.0 Add in more test questions. Does your bot respond as expected?

1.1 Edit the `bad-language.m` topic to add more bad words. The file is located in the `topic` folder in `CU101` . Re-test your bot to make sure it works.

1.2 What happens if the user doesn't enter a bad word? Edit the `idk:` (**I** **D**on't **K**now) response to make it more natural. In later sessions, you will learn better ways of handling idk situations.



```
idk: I don't understand the question.
```

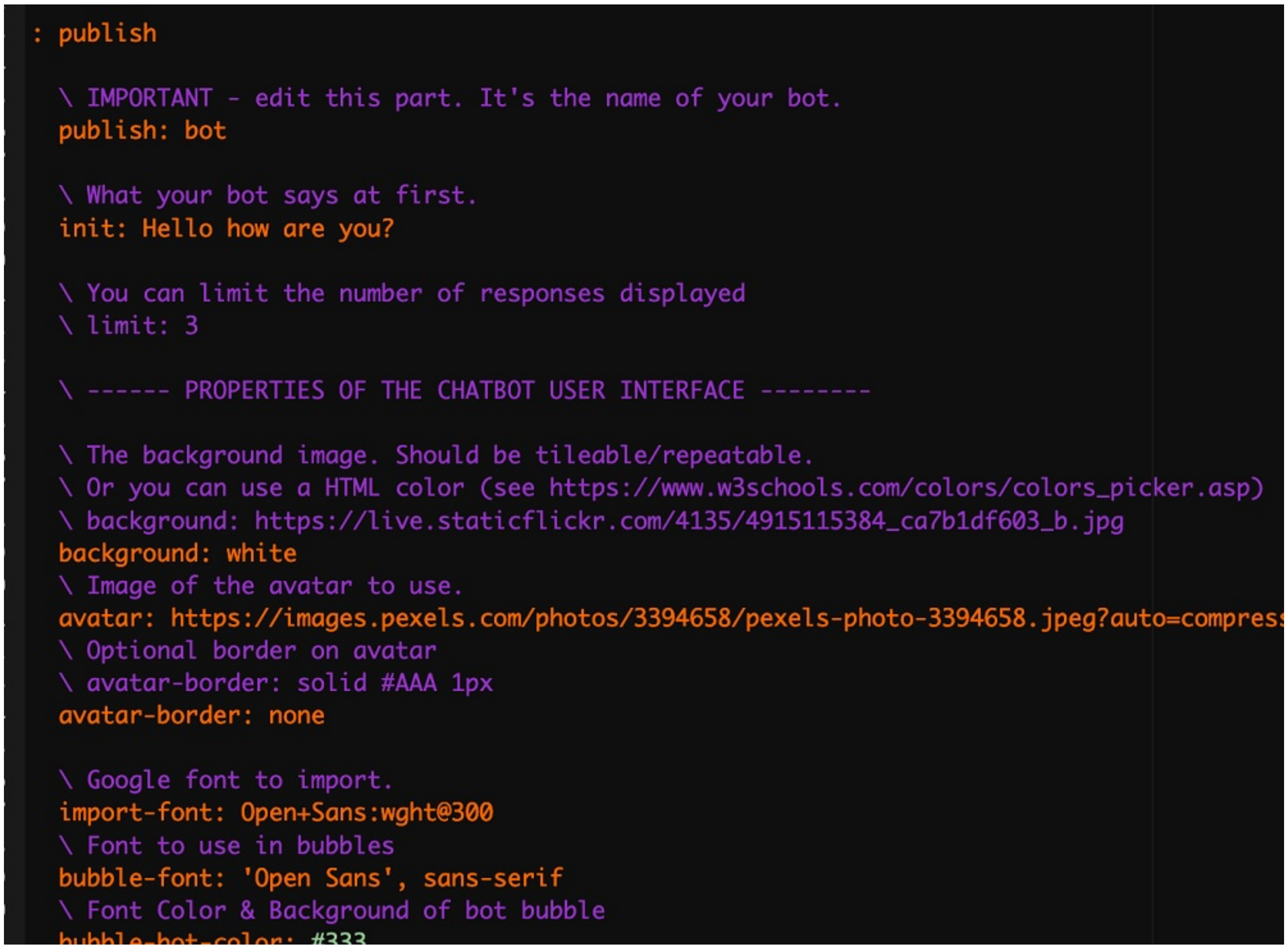
Fig 3: The default IDK (I Don't Know) response.

1.3 Add or change the responses in `bad-language.m` , so that your chatbot sounds more natural. Re-test your bot and if you're happy it works well, publish it.

1.4 You can customize everything about your chatbot's appearance. Look at the `publish` word in `main.m` See Fig 4 below.

Note #1: The `title` , `url` , `description` and `thumbnail` are what you see when you share your chatbot over social media or social messaging (eg WhatsApp, Telegram etc).

Note #2: Anything in purple is a comment for humans to read only. They are ignored by Autocaffe.



```
: publish

  \ IMPORTANT - edit this part. It's the name of your bot.
  publish: bot

  \ What your bot says at first.
  init: Hello how are you?

  \ You can limit the number of responses displayed
  \ limit: 3

  \ ------ PROPERTIES OF THE CHATBOT USER INTERFACE --------

  \ The background image. Should be tileable/repeatable.
  \ Or you can use a HTML color (see https://www.w3schools.com/colors/colors_picker.asp)
  \ background: https://live.staticflickr.com/4135/4915115384_ca7b1df603_b.jpg
  background: white
  \ Image of the avatar to use.
  avatar: https://images.pexels.com/photos/3394658/pexels-photo-3394658.jpeg?auto=compress
  \ Optional border on avatar
  \ avatar-border: solid #AAA 1px
  avatar-border: none

  \ Google font to import.
  import-font: Open+Sans:wght@300
  \ Font to use in bubbles
  bubble-font: 'Open Sans', sans-serif
  \ Font Color & Background of bot bubble
  bubble-bot-color: #333
```

Fig 4: The default publish word

1.5 Extend your bot by adding the `greeting.m` topic. This topic handles simple questions when the user questions. Hint: You need to `include` this topic. Should you include it before or after `bad-language.m` ? Test your bot by adding new Test questions. When you are happy with it, Publish your bot.

SESSION 2 : Types, Things and Templates

Step 1: Create a new topic called `likes.m` . Using the lecture video as a guide, replicate the likes chatbot in `likes.m` . Use the final version in the video that has the types `love` and `likes` defined. Test this bot thoroughly.

Step 2: Extend this bot to add in new verbs for `love` , and also add new things you bot likes. Test your changes thoroughly before moving on.

Step 3: Create a new type called `dislike` for things your bot hates. Add in all your bot's `dislike:` things.

Step 4: Create a new type called `hate` that captures the meaning of the verb hate.

Step 5: Using `$dislike` and `$hate` , create new templates so your bot responds correctly when the user asks about things your bot hates. Test your bot thoroughly as usual.

Step 6: You now want your bot to respond with **images** instead of words. Select 3 images on the internet to represent the ideas "Love", "Hate" and "I don't know". Note: you have to right-click and select "Copy Image Location", since you need the image's URL. Ensure that these URLs are image URLs, ie they end in `.jpg` , `.jpeg` , `.gif` or `.png` .

Step 7: Each time your bot responds to a `love` , just paste in the URL for the love image in to the response. Example:

```
Q: $love $likes|$likes+s
A: https://upload.wikimedia.org/wikipedia/commons/a/a3/1328101861_Thumbs_Up.png
--
```

Step 8: Do the same for the verb `hate` .

Step 9: You can create a "catch all" template that captures all user responses. Here's how to do it:

```
Q: $_
A: I don't know
--
```

Where should you place this template within `likes.m` ? Hint: specific templates should to the top and more general templates to the bottom. Use this example to display the "I don't know" image if your user says something which is not a `love` or a `hate` .

SESSION 3 : References and Guards

Exercises for References

Step 1: Include the topic `fav.m` into `main.m` and test it out. Play with the bot and add in more drinks your bot likes. Test your changes thoroughly before moving on.

Step 2: Add in a new type called `food` .

Step 3: Fill in `food:` with actual dishes your bot likes.

Step 4: Amend the existing templates so that your bot responds correctly to questions on its likes for `food` items. You should **not** have to add templates. Just amend the ones already in `fav.m` . Test your changes thoroughly before moving on.

Step 5: If you've done Step 4 correctly, you'll notice lots of repetitions in your `Q:` rules. This isn't good because it makes your bot more prone to bugs. We want to fix this now. Add in a new type called `loves` . This will be a **super-type**.

Step 6: Instead of adding in things into `loves` , add in the things `:drinks` and `:food` into it like so:

```
loves:  :drinks :food
```

This means `loves` contains all things from both `drinks` and `food` . Use `@loves` to do the matching instead of using `@drink` and `@food` separately. Are your `Q:` rules simple again?

Exercises for Guards

(continue from the previous exercise for References)

Step 7: Amend your templates so that if the user says "like coffee" or "love coffee" the bot responds with "Crazy about coffee. But can't drink it cos I am a bot". You need at least one guard for this. Be sure to put spaces in your guards!

Step 8: Change your templates again so that if your user asks "Do you like X" and if X is not "coffee", your bot responds with "Don't you like coffee?" Hint: `same? not` means "not the same as".

SESSION 4 : Logging and Debugging

Step 1: Include the topic `likes.m` from Session 2's exercise into `main.m` We want to log the IDK responses from the catch all.

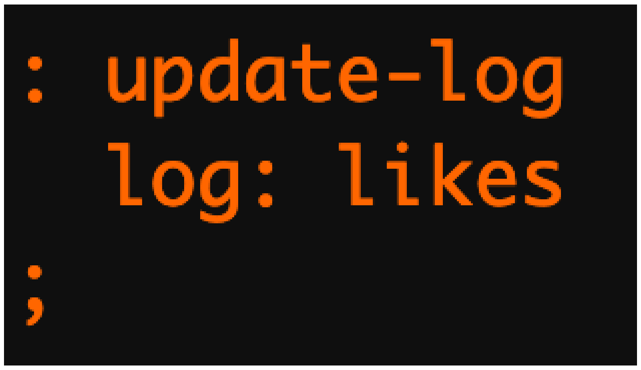
Step 2: Call your bot `likes` so that the IDKs will go into the file `likes.log`

Step 3: Add in a `L:` rule into the catchall to log the user's responses:

```
Q: $_
A: I don't understand the question
L: $_
--
```

Step 4: Publish your bot again (because you've now added logging). Play with the bot to get it to respond with the IDK message.

Step 5: Edit the bot name in `update-log` word so that it corresponds to your chatbot's name (in this case, `likes`):



```
: update-log
  log: likes
;
```

Fig 5: The update-log word. Change the name to match your chatbot's name. In this example, the chatbot's name is "likes"

Step 6: Click on `Update Log`. This updates the chatbot's log file using the latest data from the chatbot server.

Step 7: Click on the File Explorer button to show the list of files. You should see the `likes.log` file. Open it up and examine its contents.

NOTE: **The logs take 30 seconds to update**. Don't worry if you don't see your logged IDK messages immediately. Just click `Update Log` again and close/open the `likes.log` file.

Step 8: Get a friend to provoke another IDK from this bot. Check that these appear in the logs. You have to close then re-open `likes.log` to view the new responses.

Step 9: Your bot should log just a single word since `$_` matches just the first word in the user's response. To get it to log the entire user sentence, you have to use a **smart context** like so, with the word `last-question`.

```
Q: $_
A: I don't understand the question
L: ${ last-question }
--
```

Note the spaces!

Make this change to your logging, **publish your bot** and provoke an IDK. After clicking `Update Log` , check the `likes.log` file to see if the user messages have come through. Remember, you may have to wait for 30 seconds to see the updated log file.

SESSION 5 : Associations, Searches and Memory

Exercises for Associations

Step 1: Include the topic `animals.m` into `main.m` and test it out. Play with the bot and add in more animals.

Step 2: Create a second template so that the bot responds to plurals: horse**s**, pon**ies**, hippopotamus**es** etc. Are there other plural endings? Try to make the `A:` response grammatical (have vs has).

We want to amend the microtopic to correctly answer the question "How many feet have 3 horses?" or "two ducks" Think about this a bit.

Step 3: Include the `numbers.m` microtopic in `main.m` . Be sure this is the first one included. This microtopic has **three special types:** `@int` , `@real` and `@number` These respectively match integers, real numbers or both. `numbers.m` also includes conversion words `int` and `real` that convert the text into actual integers or real numbers so you can do calculations.

Step 4: Create a new template that matches "three ducks" or "2 horses". Hint: `$x.@int` matches an integer, including small textual numbers like "two".

Step 5: In your `A:` response use a smart context to retrieve the feet for the animal and multiply by the number of animals. Hint: `${ $x int $animal feet * }` means: first get the reference `$x,` covert to an integer. Get the animal's feet. Then multiply `*` these two integers.

Step 6: Create new test questions and test your bot thoroughly before publishing.

Exercises for Search

Step 1: Include the topic `wise.m` into `main.m` and test it out. Play with the bot and add in more wise sayings.

Step 2: Add in a new type called `food` that has various foods (eg, `nasi_lemak` , `char_kuay_teow` , etc)

Step 3: Add a new `search:` database called `food-sayings` that lists all kinds of food to tell the user about. Eg "Nasi Lemak is delicious at noon"

Step 4: Add in a new Template to respond with a food saying if the user talks about a `food` item.

Bonus: What happens if the user's sentence has both a `food` and a `keyword` ? Can you create a template to handle this separately? Where should you place this template? (Hint: specific templates go first, general ones last.)

Exercises for Memory

Step 1: Include the topic `book.m` into `main.m` and test it out. (Remember to remove other includes or they may clash with your book bot).

Step 2: Add new keywords, eg "suspense" , "murder" , "crime" etc.

Step 3: Add 5 or more book titles and their authors. Be sure to include these from the new keywords.

Test your book bot **thoroughly** before moving on!

You want to include a picture of the book along with the title and author.

Step 4: Create a new association called `image` . Make this an association of book ids to a HTML link for the book image. Eg:

```
assoc: image
{{
   1 "https://i.gr-assets.com/images/S/compressed.photo.goodreads.com/books/1359299332l/83346.jpg
   2 "https://i.gr-assets.com/images/S/compressed.photo.goodreads.com/books/1388212809l/93101.jpg
}} +image
```

(FYI: the images are from goodreads.com.) Obtain images for all the other books.

Step 5: Put in a smart context to display the book's image in the bot's answer.

SESSION 6 : Rooms and Keys

Step 1: Select any **three** microtopics from the preceding Sessions (eg, `animal.m` , `book.m` and `wise.m`)

Step 2: You want to combine these three microtopics into a single chatbot. Our goal is to get your new chatbot to correctly answer questions from these 3 different microtopics. This is easy if the topics are very different and have very different templates - all you need to do is to `include` them! But let's say we want to bot to stick to a single topic before moving on to another. Include the three microtopics you selected in `main.m`

Step 3: Put these microtopics into rooms. Rather than doing this by adding `room:` ... `end-room` within each microtopics, you may also do it in the `include` in `main.m` like so:

```
room: animal
include ./topics/animals.m
end-room

room: book
include ./topics/book.m
end-room

room: wise
include ./topics/wise.m
end-room
```

Step 4: Create a new microtopic called `bot.m` . Using the online video as a guide, add 3 templates in `bot.m` to shuttle your users between these rooms. Publish your bot and test it thoroughly. Remember to include `bot.m` last of all in `main.m` .

This section concludes the labs.

Chatbot Recipes

Here are a few "recipes" to spice up your chatbot. Have fun!!!

1. Running Commands in Templates

You can run any Smojo word in your templates, using the `C:` directive. For example, if you want to set a mem called `count` after your template matches, you can use `C:` like so:

```
Q: @help
A: You asked for help
C: count 1+ count!
--
```

In this example, `count` is incremented by 1 every time this template is matched. You can use guards with your `C:` 's

```
Q: @help
A: You asked for help
C: count 10 < % count 1+ count!
C: count 10 > % 0 count!
--
```

Note that **all** `C:` directives with matching guards are executed. In the example above, count is reset to zero if it is 10. There are much better ways to do a reset, eg: `count 1+ 10 mod count!`) , the example is only for illustration.

Typically, you want to use one `C:` per action, but you can do multiple things in one `C:` directive. For example, if we had words `update-mems` and `clean-editor` , then you can call them both in one line:

```
Q: @help
A: You asked for help
C: update-mems clean-editor
--
```

or you could also use multiple `C:` directives. `C:` directives are processed in order.

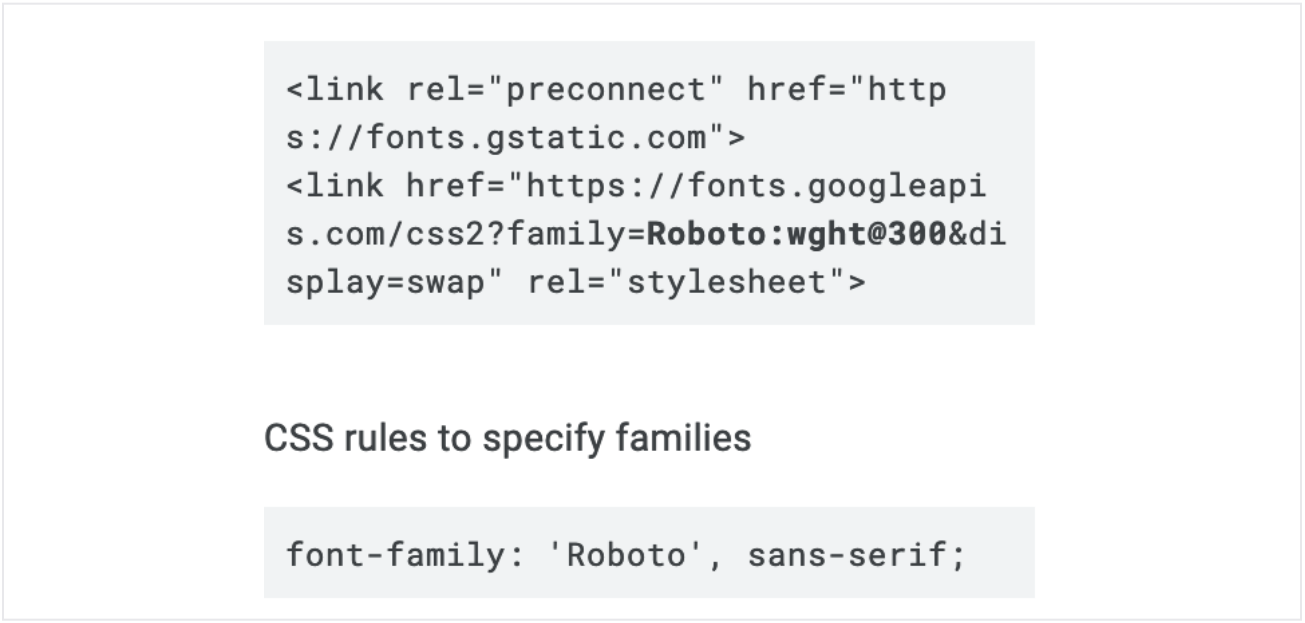
2. Custom Text Bubbles

There are 4 words you can use to customize the color of the chat bubbles and text, for the bot and user. Examples:

```
bubble-bot-background: purple
bubble-user-background: #AABBCC
bubble-bot-color: black
bubble-user-color: green
```

Use the HTML colors (eg `#AABBCC`) for fine control of the color you want. You should place these anywhere inside the `publish` word.

In addition to these, you can also amend the bubble font and size. To amend the font, you can import Google fonts. Search and select the font you want to use. For example,if you choose the "Roboto":



```
<link rel="preconnect" href="https://fonts.gstatic.com">
<link href="https://fonts.googleapis.com/css2?family=Roboto:wght@300&display=swap" rel="stylesheet">
```

CSS rules to specify families

```
font-family: 'Roboto', sans-serif;
```

Figure A1: - the select Family dialog in Google Fonts.

You should then include the following in your `main.m` 's `publish` word:

```
import-font: Roboto:wght@300
bubble-font: 'Roboto', sans-serif
```

Finally, you can change the radius used on the bubbles. By default, this is 16 pixels, which gives very rounded corners. You can also set the border to use around all bubbles. By default, there are no borders on bubbles.

```
bubble-radius: 6
bubble-border: solid #ABC 1px
```

The colors are the usual HTML colors. You can find more colors here.

3. Adding Buttons

The `button` widget specifies a button. You can use this in a smart context. For example:

```
A: ${ "How are you?" button }
```

You can also specify the CSS for these buttons using the word `(button)` . Here's an example, which uses a Smojo word called `myButton` . You can make your own words too.

```
\ Creates a custom button
: myButton ( "s" -- "s" )
  q{
        margin-left:10px;
        margin-top:10px;
        padding:10px;
        font-size:15px;
        color:#DDD;
        background:#AAA;
  }q  (button)
;

Q: $_
A: Hello ${ "How are you?" myButton }
--
```

4. Adding Images

To add an image, you need its URL. To display the image, use the widget `image` in a smart context. Example:

```
Q: $_
A: ${ "http://www.jsbach.net/bass/elements/bach-hausmann.jpg" image }
--
```

If you want to set the CSS style of your image, use `(image)` instead:

```
Q: $_
A: ${ "http://www.jsbach.net/bass/elements/bach-hausmann.jpg" "width:10px;height:auto;" (image) }
--
```

5. Youtube Videos

To add a Youtube video, you need its ID. To display the video , use the widget `utube` in a smart context. Example:

```
Q: $_
A: ${ "nVoFLM_BDgs" utube }
--
```

If you want to specify the width and height, use `(utube)` instead:

```
Q: $_
A: ${ "nVoFLM_BDgs" 100 200 (utube) }
--
```

The last 2 arguments into `(utube)` are width (=100) and height (=200).

6. Special links: Telephone, SMS, etc.

You can add ordinary HTML links using the word link:

```
Q: $_
A: I love ${ "AI4IMPACT" "https://ai4impact.org" link }
--
```

Telephone (mobile only)

```
Q: hotline
A: ${ "Call our hotline now" "+65 555 12345" tel }
--
```

SMS (mobile only)

```
Q: help
A: ${ "Click to SMS us for help!" "+65 555 12345" "help me please..." sms }
--
```

Email

```
Q: $_
A: ${ "Click here to send us an email..." "abc@example.com" "I need help!" email }
--
```

Whatsapp (best on mobile)

This link may not work well on desktops, but should work on phones.

```
Q: $_
A: ${ "Click to DM us on whatsapp" "+65 555 12345" "Hello, I need help" whatsapp }
--
```

7. Emojis

Once you know the ID of the emoji you want to use (you can search here or elsewhere), in decimal or "hex" format, you can use it in your responses. For example, the emoji 🦁 is `129409` (decimal) or `x1F981` (hex). Note that hex emojis should be preceded by an `x` . Once the ID is known, you can embed it in your responses using a smart context and the `emoji` widget:

```
Q: $_
A: Hello ${ "x1F981" emoji } is here.
--
```

8. Wrapping Text in a Button

Use the CSS `white-space:normal;` along with a fixed `width` . Here's an example:

```
: myButton ( "s" -- "s" )
  q{
     width:100px;
     white-space:normal;
     color:red;
     font-size:20px;
  }q (button)
;



Q: $_
A: ${ "Not a valid email. This is bad. this is getting very very long also. Oh no so long!!!! Too
--
```

9. Multiple Choice Questions

The MCQ widget helps you create multiple-choice entries.

Note: To use this widget, I've assumed you know some basic Smojo.

The word to do this is:

`MCQ ( seq "prefix" "button-message" -- "s" )`

Notes:

1. You should only use this in a smart context.
2. `seq` is a sequence of messages for the `mcq` . Eg: `"Java" "Ruby" "Matlab" "Python" 4 list`
3. `prefix` is a string that will be put into the first part of the response. Use this for literal matching into a rule that processes your response.
4. `button-message` is what to display on the button.

The actual response of the MCQ is encoded as a string of numbers, 1 = selection #1, 2 = selection #2, etc. The word `MCQ>#` takes this string and converts it into a hash for easy processing:

```
MCQ># ( "s" -- # )
```

Here is an example code, which stores the numerical hash representing the user's selection into the Memory item called `lang` :

```
mem: lang

Q: languages: $x
A: Your response was $x ${ "Display Languages" button }
C: $x MCQ># lang!
--

Q: Display Languages
A: lang= ${ lang }
--

Q: $_
A: ${ "Java" "Python" "Matlab" 3 list "languages:" "OK" MCQ }
--
```

You can use the word `#contains?` to retrieve specific selections, or retrieve the selection IDs using `#values` . Both these words are described in the tutorial on using hashes.

10. Checking Email Address Format

You can use the programmatic type `@email` , which is contained in the topic `email.m` to check for valid email addresses. Include this topic at the top of your `main.m` and you're ready to go. Here's an example:

```
Q: $x.@email
A: Congratulations! Your response $x is a valid email address.
--

Q: $_
A: Not a valid email. This is bad.
--
```

11. Logging the Entire User's Response in IDKs

Often, when you hit an IDK, you want to log the entire user's response. To do this, use the word last-question, which will get the entirety of the user's response. Here's an example logging the IDK:

```
Q: $_
A: Sorry, I don't understand
L: ${ last-question }
--
```

12. Customize the Send button

We provide a default send button, but you can customize this. Here's a sample from using an icon from https://flaticon.com. You need to add these lines to your `main.m` 's `publish` word:

```
send-button-image-url: https://www.flaticon.com/svg/static/icons/svg/3652/3652532.svg
send-button-image-style: padding-left:10px;
```

Note that you will often need to adjust the CSS. If you want a circular button, then use `border-radius:20px;`

13. Responding Silently

Sometimes you **don't** want to display:

1. The user's response (eg, if it was through a button or MCQ) , or
2. You might want to skip a bot response and process the user's request silently.

You can do this by adding `#silent#` at the start of your response. For example:

```
Q: $_
A: #silent# I don't know how to process this response!
L: ${ last-question }
--
```

In this example, the `#silent#` suppresses the subsequent response "I don't know how to process this response!"

You can silence buttons using the `silent` word:

```
Q: $_
A: ${ "Yes" silent button } ${ "No" silent button }
--
```

Here the word `silent` wraps `#silent#` into the responses "Yes" and "No". When the user clicks Yes or No buttons, the response is processed as usual by your bot, but the "Yes" and "No" responses are not displayed.

Here's an example using MCQs:

```
Q: $_
A: ${ "What do you mean" "Why are u saying this?" 2 list "language" silent "OK" mcq }
--
```

Note that `silent` must be put in after the prefix setting of `MCQ` .

The user's response is processed as usual but the result of the selection is not displayed in the conversation list.

14. Removing the Avatar

You may sometimes want to remove the avatar icon. To do this, set `avatar: none` in main.m's publish word.

15. Increasing the Bot Bubble width

The `bot-bubble-width:` setting in `publish` in `main.m` allows you to to this. For example, to set the maximum bubble width to 380 pixels:

```
bot-bubble-width: 380
```

16. Safely handling IDKs in Assocs and Searches

If your Association or Search does not have a valid value for a given input, then this will result in a 502 error (when the bot is published) or "null exception" (when the bot is tested). For example, with the assoc `feet` :

```
assoc: feet
{{
   :dog 4
   :cat 4
}} +feet
```

You will get the 502/null errors if you request for `"ELEPHANT" feet` , since the output is an IDK. To prevent this from happening in your templates, you can use the `safe` word below:

```
: safe ( "s"|null "alt" -- "s" ) { s alt }
        s null? -> alt exit |. s
;
```

Insert `safe` anywhere before your templates are used. `safe` uses an "alternative" text to be displayed when the assoc or search gives and IDK.

You should call `safe` in your templates like so:

```
Q: $x.@animal
A: ${ $x capitalize } has ${ $x feet "an unknown number of" safe } feet
--
```

THE END