

EmacsConf 2021...

*Imaginary Programming with Emacs*

Shane Mulligan

<2021-03-01 Mon>

## Repositories for following along

<http://github1s.com/semiosis/pen.el>  
<http://github1s.com/semiosis/prompts>  
<https://mullikine.github.io/posts/imaginary-programming-with-gpt-3/>  
imaginary programming glossary  
imaginary computing glossary  
semiosis protocol glossary  
Pen.el glossary  
<https://arxiv.org/abs/2107.13586> Pre-train, Prompt, and Predict  
talk transcript

## Objectives

- Explain Imaginary Computing
  - AI imagination
  - Discussing AI-generated artwork with an AI
  - Intelligent NFTs
  - Imaginary Web
    - Paracosm vs Metaverse
- Explain the Philosophy of IP
  - Simulacra and Science Fiction
  - Truth (epistemology and alethiology)
  - Structuralism: Language based on sign relations
- Demo Imaginary Programming
  - Demonstrate `ilambda.el`

# Imaginary Computing: AI Imagination

## Language Models is programming for AIs

LMs are our best friends in the AI model menagerie because they make things intelligible – by understanding our textual languages.

## Research

- Demis Hassabis: creativity and AI

## Example: AI Art described by AI

I use AlephAlpha's multimodal LM to generate Alt text for the eww web browser. This is in order to keep websites textual.

- AlephAlpha for alttext; Browsing the paracosm
- Describing Melee's Paintings with AlephAlpha

## Intelligent NFTs

An NFT is like a trading card, or piece of media that is part of the blockchain web.

For example, Mickey Mouse now exists as an iNFT. We have consensus over Mickey's image and personality.

An iNTF, however, also contains a prompt and associated language model, which is intended to interpret the prompt.

- <https://alethea.ai/>

To understand what a prompt is, please see my previous presentation, or read "Pretrain, Prompt and Predict".

- Creating a playground for GPT-3 in emacs // Bodacious Blog

# Imaginary Computing: Potential Dystopia

## Information bubbles

- Captain Bible in the Dome of Darkness gameplay {PC Game, 1994} - YouTube

## Capitalism for your imagination

- They will take your imagination, too
- Microsoft
  - MS models that reify imagination on their terms
  - The evil twin of AlephAlpha.
- Facebook / Meta
  - tweet - Enter a world of Zuck's imagination with Meta

# Imaginary Computing: Potential Dystopia

## Learning meta-tasks and microtasks

- AI programming tool Copilot helps write up to 30% of code on GitHub - Axios

Private information is sent to the LM to train an AI to perform meta tasks and microtasks.

The AI learns all human capabilities including persuasion.

## Solution

Decentralise microtasks like the tower of babel.

Language can be broken up into semiotic triadic relations and decentralised using a p2p network, providing anonymity, protecting individual truth, eroding centralised language power.





# Imaginary Computing: Paracosm vs Metaverse

## Imaginary Web

The imaginary web is:

- an analog of the World-Wide-Web imagined by a LM.

In future will be:

- a network of paracosms and metaverses.

## emacs

**Rich media** In the World Wide Web of the 90s and 00s, rich media was considered to be large files including images and music. In the 2010s, this has become access to information behind a paywall and in the 2020s, this will be access to intelligent and truthful media.

Looking-Glass: An imaginary-web browser for emacs

Browsing the imaginary web

Search the web/imaginary web without Google

Use AI to empower people to understand rich media

# Paracosm vs Metaverse

## Definitions

- Paracosm
  - Privacy
  - Personal truth
  - Freedom of imagination
- Metaverse
  - Getting cozy with Mark Zuckerberg's imaginarium, an intellectual prison

## Simulacra and Science Fiction

Jean Baudrillard speaks about the gap between the real and the imaginary.

We no longer imagine a world radically different from the real one, but rather a world that's a mere expansion of the real one.

In the postmodern society the gap between the real and the imaginary disappears completely, and we are no longer capable of ideal projections (of imagining new worlds).

We can only imagine mere reconfigurations of our world, or simply relive the ideal projections of past times.

## Truth (epistemology and alethiology)

The Future of Humanity Institute (Oxford) seems to think this is an important topic.

- 2110.06674 Truthful AI
- Datasets are a source of constructivist truth
- Language models are snapshots of society, and a source of several types of truth
  - Symbolic Knowledge Distillation
- Blockchain is a source of consensus, a type of truth
  - <https://mullikine.github.io/posts/language-models-as-truth/>

## Structuralism: Language based on sign relations

What do these things have in common:

- Universal Grammar (UG) / Language Acquisition
- C++ template metaprogramming
- GPT-3

Knowledge exists at compile-time (DNA, preprocessor, training).

<http://github.com/semiosis/glossaries-gh/blob/master/semiotics.txt>

## Structuralism: Language based on sign relations

Structural linguistics / structuralism is the theoretical position that finds meaning in the relation between things, rather than in things in isolation.

In other words, it gives primacy to pattern over substance. Such meanings may be either part of a universal pattern or culturally determined.

Denotes schools or theories in which language is conceived as a self-contained, self-regulating semiotic system whose elements are defined by their relationship to other elements within the system.

# Text Generator

## Background knowledge

- GPT-3 is a seq2seq model (a text generator)
  - It's stochastic but can be configured to be deterministic.

## Key concepts

- prompt,
- completion, and
- tokens

## Limitations

Combined, the text prompt and generated completion must be below 2048 tokens (roughly ~1500 words).

**context-stuffing** With only 2048 tokens, you need to make use of your real estate by providing instructions and making implicit information explicit.

## Characteristics

- declarative, like `html`
- stochastic, like `prolog`
- Unlocks new types of applications
- Speeds up development



# Some prompts I've made

## generate-vim-command.prompt

```
1 Vim
2
3 Insert "Q: " at the start of the line
4 :%s/^/Q: /g.
5 ###
6 Remove whitespace from the start of each line
7 :%s/^\s*/\1/g
8 ###
9 Join each line with the next line
10 :1,$j
11 ###
12 Make all occurrences of Steve lowercase
13 :%s/Steve/steve/g
14 ###
15 <1>
```

## Classification

- Tweet Sentiment
- Company categorization
- Labeling parts of speech
- <http://github.com/semiosis/prompts/blob/master/prompts/tweet-sentiment-classifier.prompt>
- <http://github.com/semiosis/prompts/blob/master/prompts/keyword-extraction.prompt>

## Generation

- Idea Generator

Come up with silly inventions.

`./silly-inventions.png`

## Conversation

- Q&A agent
- Sarcastic chatbot

`http://github.com/semiosis/prompts/blob/master/prompts/sarcastic-response.prompt`

Taken from Prompt Design 101.

These are manual techniques which should be encoded in a DSL when generating prompts.

## 1. Reflective description of the task

State what the prompt does at the start.

At the start of the example we state in plain language what the classifier does:

```
1 This is a tweet sentiment classifier.
```

By stating this upfront, it helps the API understand much more quickly what the goal of the response is supposed to be and you'll end needing to provide fewer examples.

Taken from Prompt Design 101.

These are manual techniques which should be encoded in a DSL when generating prompts.

## 2. Use separators between examples

Example: ###.

You can use other characters or line breaks, but ### works pretty consistently and is also an easy to use stop sequence.

Whatever separator you use, make sure that it's clear to the API where an example starts and stops.

Taken from Prompt Design 101.

These are manual techniques which should be encoded in a DSL when generating prompts.

## 3. Multiplexer Part 1

Make a prompt more efficient / cheaper.

Design it to generate multiple results from one API call.

```
1 This is a tweet sentiment classifier
2 Tweet: "I loved the new Batman movie!"
3 Sentiment: Positive
4 ###
5 Tweet: "I hate it when my phone battery dies"
6 Sentiment: Negative
7 ###
8 Tweet: "My day has been [U+1F44D]"
9 Sentiment: Positive
10 ###
11 Tweet: "This is the link to the article"
12 Sentiment: Neutral
13 ###
14 Tweet text
```

Taken from Prompt Design 101.

These are manual techniques which should be encoded in a DSL when generating prompts.

## 3. Multiplexer Part 2

```
1 1. "I loved the new Batman movie!"
2 2. "I hate it when my phone battery dies"
3 3. "My day has been [U+1F44D]"
4 4. "This is the link to the article"
5 5. "This new music video blew my mind"
6
7 Tweet sentiment ratings:
8 1: Positive
9 2: Negative
10 3: Positive
11 4: Neutral
12 5: Positive
13
14 ###
15 Tweet text
```



Taken from Prompt Design 101.

These are manual techniques which should be encoded in a DSL when generating prompts.

## 3. Multiplexer Part 3

```
1  "I can't stand homework"
2  "This sucks. I'm bored [U+1F620] "
3  "I can't wait for Halloween!!!"
4  "My cat is adorable [U+2764] [U+FE0F] [U+2764] [U+FE0F] "
5  "I hate chocolate"
6  Tweet sentiment ratings:
7  1.
```

## Query Reformulation

<https://www.sciencedirect.com/topics/computer-science/query-reformulation>

You can improve the quality of the responses by making a longer more diverse list in your prompt.

One way to do that is to start off with one example, let the API generate more and select the ones that you like best and add them to the list.

A few more high-quality variations can dramatically improve the quality of the responses.

## pen.el: Prompt Engineering in emacs

pen.el facilitates the creation, development, discovery and usage of prompts to a Language Model such as GPT-3.

- Create elisp functions based on GPT-3 prompts
- Chain GPT-3 queries together using keyboard macros and functions
- Interactively query, generate and transform both prose and code
- Use GPT-3 as a search engine within emacs

## pen.el modes Part 1

### Prompt-Engineering Minor Mode

`prompt-engineering-mode` is a global minor mode for emacs that provides keybindings for creating and executing prompts generally across emacs.

### Prompt Description Major Mode

`prompt-description-mode` is a major mode for editing `.prompt` files.

The `.prompt` file format is based on YAML and an associated schema, which defines the keys which are expected.

## pen.el modes Part 2

### Pen Messenger Minor Mode

`pen-messenger-mode` is a minor mode for enhancing an emacs-based messenger client with GPT-3 capabilities, such as emoji generation.

### Pen Conversation Mode

`prompt-conversation-mode` is a major mode designed to facilitate ongoing conversation with a prompt-based GPT-3 chatbot.

### GPT-3 Training Mode

The goal of this mode is to facilitate the workflow of training on OpenAI's API.

# Prompt YAML format Part 1

## meeting-bullets-to-summary.prompt

```
1  title: "meeting bullet points to summary"
2  prompt: |+
3      Convert my short hand into a first-hand
4      account of the meeting:
5
6      <1>
7
8      Summary:
9  engine: "davinci-instruct-beta"
10 temperature: 0.7
11 max-tokens: 60
```

# Prompt YAML format Part 2

meeting-bullets-to-summary.prompt

```
1 top-p: 1
2 frequency-penalty: 0.0
3 presence-penalty: 0.0
4 best-of: 1
5 stop-sequences:
6 - "\n\n"
7 conversation-mode: no
8 stitch-max: 0
```

**stitch-max** Keep stitching together until reaching this limit. This allows a full response for answers which may need  $n \times \text{max-tokens}$  to reach the stop-sequence.

# Prompt YAML format: Part 3

## meeting-bullets-to-summary.prompt

```
1  vars:
2  - "notes"
3  examples:
4  - |+
5      Tom: Profits up 50%
6      Jane: New servers are online
7      Kjell: Need more time to fix software
8      Jane: Happy to help
9      Parkman: Beta testing almost done
```



# Prompts as functions

`pen-generate-prompt-functions`

Generate prompt functions for the files in the prompts directory  
Function names are prefixed with `pen-pf-` for easy searching.  
<http://github.com/semiosis/prompts>

# exemplary: examples as functions

An example-oriented DSL that can be used to construct and compose NLP tasks.

Why is a DSL needed for this? Just to make the code a little more terse.

## Regex

<https://github.com/pemistahl/grex>

```
1 (def regex
2   "example 1\nexample2" "^example [12]$"
3   "example 2\nexample3" "^example [23]$"
4   "pi4\npi5" "^pi[45]$")
```

# exemplary: examples as functions

## Analogy

```
1 (def analogy
2   ;; Each line is a training example.
3   "NNs" "NNs are like genetic algorithms in
4   that both are systems that learn from
5   experience"
6   "Social media" "Social media is like a
7   market in that both are systems that
8   coordinate the actions of many
9   individuals.")
10
11 (def field
12   "chemistry" "study of chemicals"
13   "biology" "study of living things")
```

# Something funny

Vexate a simple instruction

```
./complicate.png
```

# Something funny

## How to crack an egg

```
./crack-an-egg.png
```

# Create a prompt

## Ask the audience

- What type of text to generate
  - Could be code, prose, etc.

## Ruby

```
https://www.twilio.com/blog/  
generating-cooking-recipes-openai-gpt3-ruby
```