DYALOG

Experimenting with Large Language Models for APL

Rich Park

Neural Networks and Deep Learning in 2024

April 2023

Will Smith Eating Spaghetti

by Reddit user chaindrop using StableDiffusion

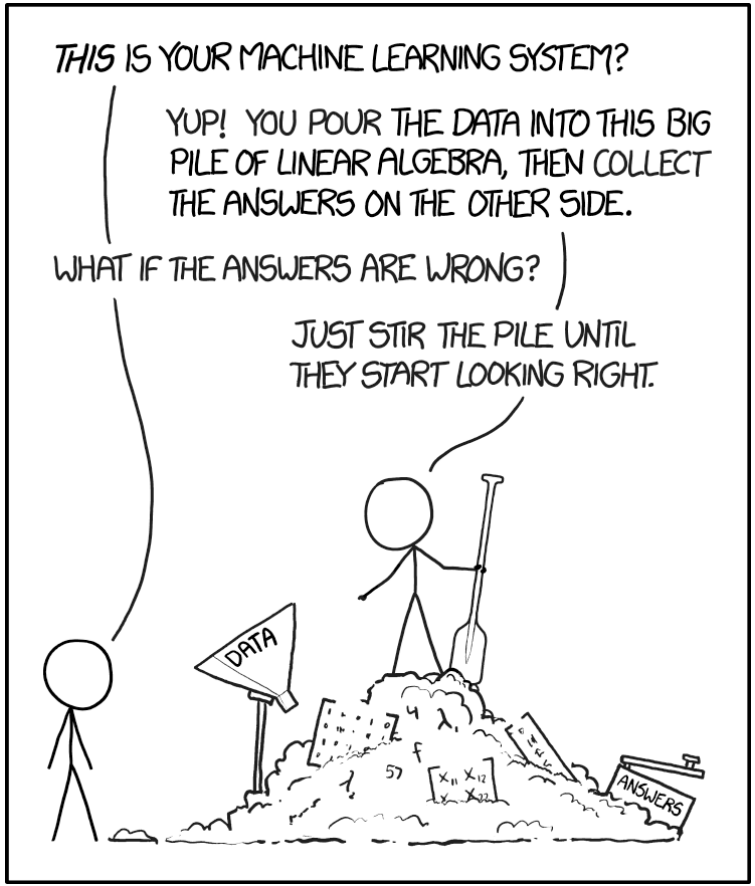
https://www.youtube.com/watch?v=XQr4Xklqzw8

June 2024

Will Smith Eating Spaghetti

by Reddit user jwhkccc using Kling

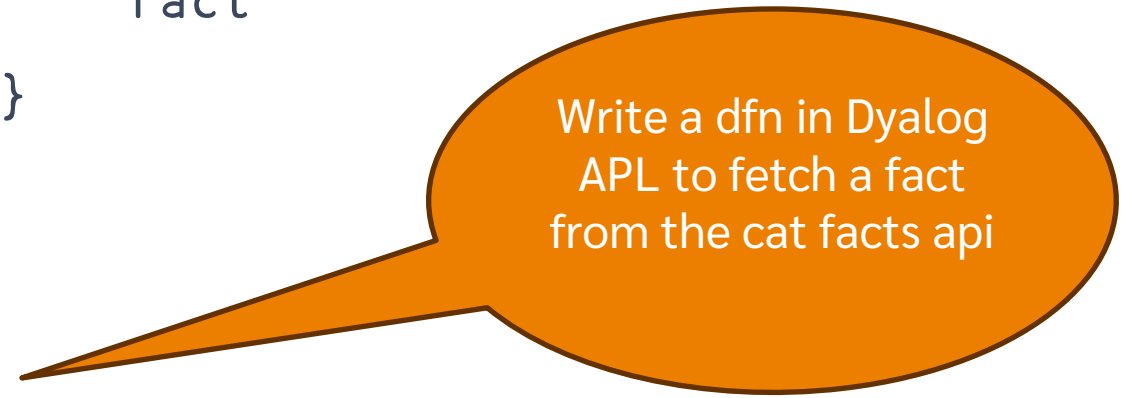
https://www.reddit.com/r/ChatGPT/comments/1dqbysz/will_smith_eating_spaghetti_kling_june_2024/



XKCD 1838

OpenAI GPT 3.5

```
GetCatFact ← {
    response ← ⎕CURL 'https://catfact.ninja/fact'
    fact ← response ⊣ 'fact' ⍳ response
    fact
}
```



OpenAI GPT 3.5

```
GetCatFact ← {
    response ← ⎕CURL 'https://catfact.ninja/fact'
    fact ← response ⊣ 'fact' ⍳ response
    fact
}
```

Explanation:

1. **`⎕CURL 'https://catfact.ninja/fact'`**: This sends a GET request to the Cat Facts API endpoint. `⎕CURL` fetches the JSON response from the API.

2. **`response ⊣ 'fact' ⍳ response`**: This extracts the value associated with the key "fact" from the JSON response by using the **Key Operator** (`⍳`).

3. **`fact`**: The result is a string containing a cat fact.

Claude 3.5 Sonnet

```
CatFact←{
      url←'https:/
      req←⎕NEW
      result←r
      fact←⎕JSO
      fact
}
```

```
LENGTH ERROR
CatFact[2] req←⎕NEW
HttpCommand'GET'(url)
                 ^
```

Claude 3.5 Sonnet

```
CatFact←{
    url←'https://catfact.ninja/fact'
    req←⎕NEW HttpCommand(('GET' url))
    result←req.Run
    fact←⎕JSON result.Data
    fact.fact
}
```

Claude 3.5 Sonnet

```
      CatFact''

British cat owners spend roughly 550
million pounds yearly on cat food.
```

Welcome to the APL Challenge!

The deadline for the current round (2024.4) is Friday 31 January 2025 at 09:00 UTC (in 67 days).

APL is a programming language that will change the way you think about solving problems. You don't need to know any APL or programming at all to participate in this challenge; the question texts will teach you everything you need to know to progress. To do this, we have kept what we teach very simple, even though APL is slightly more complicated than what we describe.

There are four rounds of the APL Challenge each year. You don't need to participate in an earlier round to participate in the current one. Each round has ten problems and runs for three months, after which Dyalog Ltd awards three USD 100 prizes. You don't have to answer every question, but your chance of winning increases as you answer more questions. A list of winners of previous rounds is available on the Dyalog website.

If you're interested but don't want to participate in this round, please register anyway (using the button at the top) so that we can notify you of updates and tell you about future rounds. You can ask us to stop sending you emails at any time.

Let's get started...

Use TryAPL to experiment and find your answer. You can write APL symbols by clicking on them in the language bar above the area where you write APL code.

Although you can begin without registering – most browsers will store your answers until you register – you must be registered and logged in to submit (send in) your answers. When you're happy with your answer, hit Submit. The system will only allow you to submit correct answers.

Several problems include some data you need. You should be able to copy text that is written with `code letters like this` by clicking on it, but note that not all browsers allow this.

Expand all

Collapse all

Prizes

Detailed rules

Frequently Asked Questions (FAQ)

Let me help solve this first APL problem.

The instruction introduces the iota operator `⍳` which generates a sequence of integers from 1 to a given number.

Since we need the numbers from 1 to 42, we simply need to write:

apl Copy

```apl
⍳ 42
```

This will generate exactly what we need: a sequence of integers from 1 to 42, with each number separated by a space.

The output will be:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42

Would you like to move on to the next problem?

ⓘ Claude does not have the ability to run the code it generates yet.

RP

3: Selection

In APL, text must appear between single quotes (`'`). For example, `'ABC'` will give a result that looks like `ABC`.

You can select specific letters from a text by putting square brackets (`[]`) to the right of the text. Inside the square brackets, you put numbers that say which letters you want to get from the text. For example, `'ABCDE'[1]` gives you the letter `A` because that's the 1st letter of `'ABCDE'`. Similarly, `'ABCDE'[5 2 2]` gives you `EBB`, because:

RP

That's not quite right. Your expression returns `OLAOAAL` when it should return `GO POLLY`. Try again - your indices should refer to the positions of the characters you want to return.

apl Copy

```apl
'DYALOG APL'[7 5 4 8 5 3 3 10]
```

ⓘ Claude does not have the ability to run the code it generates yet.

Let me count the positions for each character I need:

G is at 4

O is at 5

space is at 7

P is at 8

O is at 5

L is at 6

L is at 6

Y is at 2

apl Copy

```apl
'DYALOG APL'[4 5 7 8 5 6 6 2]
```

ⓘ Claude does not have the ability to run the code it generates yet.

RP G is not at position 4 of `'DYALOG APL'`. G is at position 6.

Ah, let me count again:

D Y A L O G (space) A P L

1 2 3 4 5 6 7 8 9 10

For "GO POLLY":

G is at 6

O is at 5

space is at 7

P is at 9

O is at 5

L is at 4

L is at 4

Y is at 2

apl Copy

```apl
'DYALOG APL'[6 5 7 9 5 4 4 2]
```

ⓘ Claude does not have the ability to run the code it generates yet.

RP Wow, you got that one straight away! Okay, here's our last problem. Good luck!

10: X Marks the Spot

Using everything that you've learnt so far, write a function that takes a size as right argument and draws a large X of that size. Use backslash symbols (`\`) and slash symbols (`/`) for the diagonals:

Copy

Let me solve this step by step:

```
      {answer} 5
\   /
 \ /
  X
 / \
/   \
```

RP Well, if you have a character vector `' \/X'`, what numbers would you need to get each of those characters from an indexing expression?

Ah! Let me think:

- Space ' ' would be 1
- Backslash '\' would be 2
- Forward slash '/' would be 3
- 'X' would be 4

So if I could create a matrix of these numbers in the right pattern, I could just index into `' \/X'`! Let me try:

apl Copy

```
{(' \/X')[1+(⍳⍵)∘.=⍳⍵+((⍳⍵)∘.=⌽⍳⍵)]}
```

ⓘ Claude does not have the ability to run the code it generates yet.

RP That's really really close! The expression to generate indices is missing parentheses, so it generates an error.

Ah yes! We need to properly group the operations to create our index matrix. Here's the corrected version:

apl Copy

```
{(' \/X')[1+((⍳⍵)∘.=⍳⍵)+((⍳⍵)∘.=⌽⍳⍵)]}
```

ⓘ Claude does not have the ability to run the code it generates yet.

RP Okay, lastly, we need to make sure that trailing diagonals have the number 3 while leading diagonals stay as they are in your solution with the number 2. How might we increase the value of trailing diagonals?

Ah - if we add 1 to the trailing diagonal matrix instead of just adding it directly, that would turn those 1s into 2s, making them index the forward slash instead of the backslash:

apl Copy

```
{(' \/X')[1+((⍳⍵)∘.=⍳⍵)+2×((⍳⍵)∘.=⌽⍳⍵)]}
```

```
LyndonWords←{
    ⍝ Implements Duval's algorithm for generating Lyndon words
    ⍝ α: alphabet (default Binary)
    ⍝ ω: max length of Lyndon words to generate
    ⍝ ←: all Lyndon words up to length ω
    ⎕IO←1
    α←'01'
    alphabet←α
    z←≠alphabet    ⍝ Mnemonic: z is the last letter of the alphabet
    length←ω
    result←''

    Generate←{
        0∊⍴ω:result                          ⍝ No more words to find: return result
        next←1+⍸(≠ω)⊢ω                       ⍝ Increment last non-z symbol
        result,←⊂alphabet[next]              ⍝ Append word to result
        ∇ word↓⍨-+/∧\⌽z=word←lengthpnext     ⍝ Repeat word to max length and drop trailing zs
    }

    ⍝ This seed means the first word generated will be ,1
    ⍝ (the word containing only the first symbol of the alphabet)
    Generate,0
}
```

Claude 3.5 Sonnet

```
      LyndonWords 4
┌─┬────┬───┬────┬──┬───┬────┬─┐
│0│0001│001│0011│01│011│0111│1│
└─┴────┴───┴────┴──┴───┴────┴─┘
      'APL' LyndonWords 3
┌─┬───┬───┬──┬───┬───┬──┬───┬───┬─┬───┬──┬───┬─┐
│A│AAP│AAL│AP│APP│APL│AL│ALP│ALL│P│PPL│PL│PLL│L│
└─┴───┴───┴──┴───┴───┴──┴───┴───┴─┴───┴──┴───┴─┘
```

Claude 3.5 Sonnet

```
      'APL' LyndonWords 3
┌─┬───┬───┬──┬───┬───┬──┬───┬───┬─┬───┬──┬───┬─┐
│A│AAP│AAL│AP│APP│APL│AL│ALP│ALL│P│PPL│PL│PLL│L│
└─┴───┴───┴──┴───┴───┴──┴───┴───┴─┴───┴──┴───┴─┘

      ⊂⍤⍋⍛⌷(⍳3)⌽¨⊂'AAP'
┌───┬───┬───┐
│AAP│APA│PAA│
└───┴───┴───┘
```

Behind operator ⍛
Coming in version 20.0

```c
// Example usage
int main() {
    const char* alphabet = "DYA";  // Binary alphabet
    int max_len = 5;               // Maximum word length

    clock_t begin = clock();

    WordList* words = lyndon_words(alphabet, max_len);

    clock_t end = clock();

    printf("Lyndon words up to length %d:\n", max_len);
    for (int i = 0; i < words->count; i++) {
        printf("%s\n", words->words[i]);
    }

    double time_spent = (double)(end - begin);
    printf("computed in %f milliseconds\n", time_spent);

    free_word_list(words);
    return 0;
}
```

```
      ]runtime "'DYALOG'LyndonWords 8"

* Benchmarking "'DYALOG'LyndonWords 8"
      |      | (ms)
 CPU (avg):  1501
 Elapsed:    1501
```

```
  YAYAA
  YAA
  YAAA
  YAAAA
  A
  computed in 53.000000 milliseconds
```

```
:Namespace SQLite
    ⍝ Initialize core SQLite function bindings
    dir←⊃⎕SH'pwd'
    ⎕NA 'I4 ',dir,'/libsqlite3.so|sqlite3_open <C[] >P'
    ⎕NA 'I4 ',dir,'/libsqlite3.so|sqlite3_close P'
    ⎕NA 'I4 ',dir,'/libsqlite3.so|sqlite3_exec P <C[] P P >P'
    ⎕NA 'U4  ',dir,'/libsqlite3.so|sqlite3_errmsg P'

    ⍝ Statement handling
    ⎕NA 'I4 ',dir,'/libsqlite3.so|sqlite3_prepare_v2 P <0T I4 >P >P'
    ⎕NA 'I4 ',dir,'/libsqlite3.so|sqlite3_step P'
    ⎕NA 'I4 ',dir,'/libsqlite3.so|sqlite3_finalize P'
    ⎕NA 'I4 ',dir,'/libsqlite3.so|sqlite3_reset P'

    ⍝ Column access
    ⎕NA 'P  ',dir,'/libsqlite3.so|sqlite3_column_text P I4'
    ⎕NA 'I4 ',dir,'/libsqlite3.so|sqlite3_column_int P I4'
```

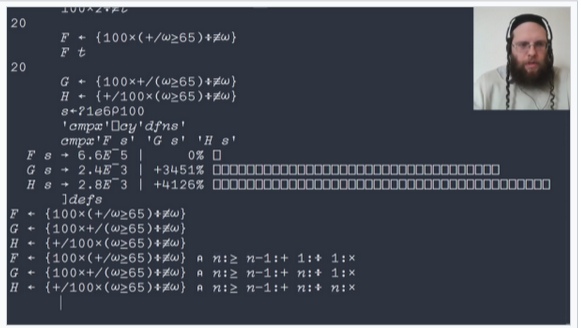
```
∇ (rc db)←OpenDatabase filename;handle
  ⍝ Opens a SQLite database
  handle←0
  rc←⎕NA'sqlite3_open filename handle
  db←handle
∇

∇ rc←CloseDatabase db
  ⍝ Closes a SQLite database
  rc←⎕NA'sqlite3_close db
∇

∇ (rc msg)←ExecuteSQL(db sql);errmsg
  ⍝ Executes a SQL statement
  errmsg←0
  rc←⎕NA'sqlite3_exec db sql 0 0 errmsg
  :If rc≠0
      msg←'ERROR: ',GetError db
      →0
  :EndIf
  msg←''
∇
```

APL Quest

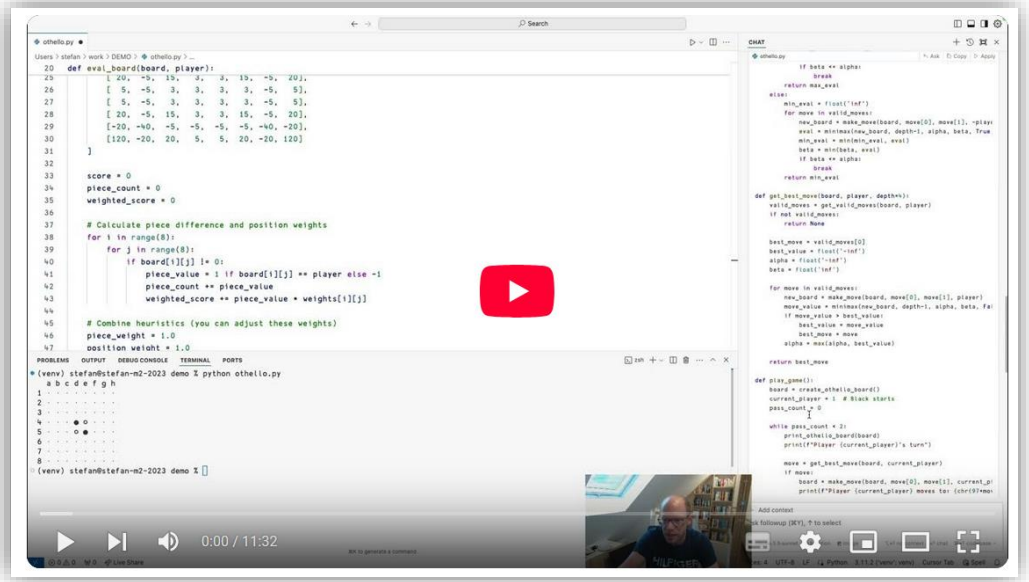
APL Quest is the title applied to the collection of phase 1 problems from the 2013–2023 APL Problem Solving Competition rounds, and to the associated live chat sessions, sample solutions[1] and video follow-ups.[2] The problem texts were written by Brian Becker[3] and the chat events were held by Adám Brudzewsky (who also recorded the videos) with guest appearances by Richard Park, from 4 February 2022 until 8 March 2024. The problems are available online as a learning resource, complete with automatic validation of solutions.[4]



Still frame from a follow-up video.

Contents [hide]

AI Assistant in Cursor



Dyalog Librarian (DAAL):
I'm DAAL, the Dyalog APL AI Librarian. I have access to Dyalog's documentation that I can search through to provide detailed answers and references. Note: I occasionally get things wrong, so please always consult the source references.

How can I serialise and compress an arbitrary APL array?

Dyalog Librarian (DAAL):
To serialize and compress an arbitrary APL array, you can use the following approach which involves two main steps: serialization followed by compression.

Step 1: Serialize the APL Array

You can serialize the array using the I-Beam operator `220⌶`. The process is initiated by calling `1(220⌶)` with the array you want to serialize. This operation converts the array into a *sint_vector* (a special type of integer vector) that contains the serialized form of the array.

Example:

```
a ← 'Hello, World!'
serializedArray ← 1(220⌶) a
```

Step 2: Compress the Serialized Array

After obtaining the serialized vector, you can compress it using another I-Beam operator, `219⌶`. The first argument specifies the compression method; for example, you could use `1` for LZ4 compression.

Example:

```
compressedArray ← 1(219⌶) serializedArray
```

Complete Example:

Here's how you would write it all together in APL:

```
a ← 'Hello, World!'                ⍝ Original array
serializedArray ← 1(220⌶) a         ⍝ Step 1: Serialize
compressedArray ← 1(219⌶) serializedArray  ⍝ Step 2: Compress
```

Deserialization and Decompression

To reverse the process (i.e., to deserialize and decompress), you would perform the following steps:

1. Decompress the compressed data:

```
decompressedArray ← 0(219⌶) compressedArray
```

How do I generate random numbers between 30 and 50 inclusive?

Generating Random Numbers in Dyalog APL

To generate random numbers between 30 and 50 inclusive in Dyalog APL, you can use the **Roll** (`?`) function along with some arithmetic operations. The **Roll** function generates random integers within a specified range.

Here's how you can do it:

- **Determine the range:** You want numbers between 30 and 50 inclusive, a total of 21 numbers (50 - 30 + 1 = 21).
- **Generate random integers from 1 to 21:** `?21` will generate a single random integer between 1 and 21. You can replace '21' with any positive integer to generate a different number of random integers. 1 2
- **Shift the range:** Add 29 to the result to shift the range from 1-21 to 30-50: `29+?21` 3

Example:

```
      29+?21
34
```

This will generate a single random number between 30 and 50. You can generate multiple random numbers by changing the left argument of the roll function. For example:

```
      29+?10⍴21
39 46 35 38 47 34 43 36 49 45
```

This will generate a vector of 10 random numbers between 30 and 50. 4

Key Points:

- The **Roll** function generates integers. If you need random decimal numbers, you can use the approach described in source 3 .
- The random number generator in Dyalog APL uses a seed value stored in the system variable `⎕RL`. You can control the seed to generate repeatable sequences of random numbers. 5 6
- You can find more information on the `?` function, `⎕RL`, and generating random numbers in sources 1 2 3 4 ... .



Save to note

How do I tell a running Dyalog session to accept RIDE connections?

Enabling RIDE Connections in a Running Dyalog Session

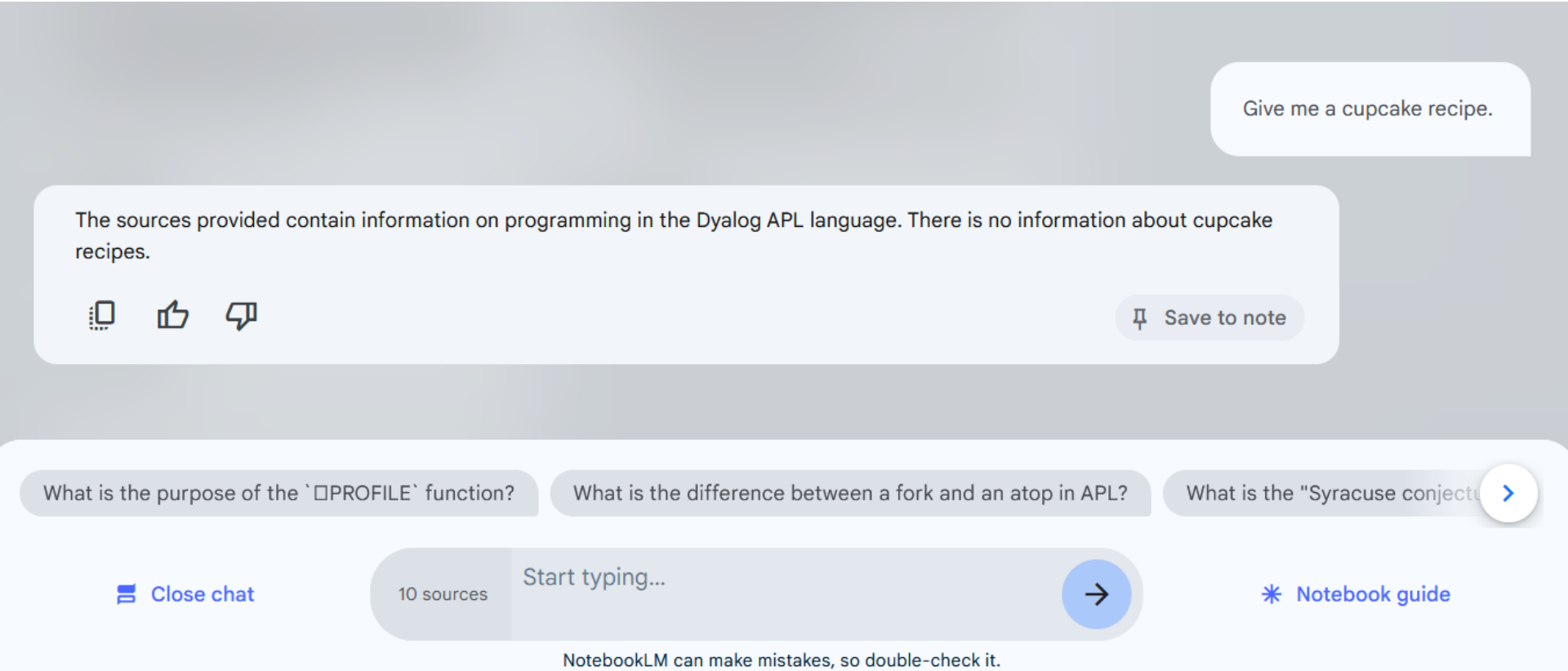
The sources indicate that you can control RIDE connections to a running Dyalog interpreter using the system function **3502⌶**. [1]

- Calling **3502⌶0** will disconnect any currently active RIDE connections and prevent any future connections from being made. [2]
- **3502⌶1** will attempt to enable RIDE. The interpreter uses the value of the initialisation string to determine the connection details. [2]
  - If the initialisation string is not defined correctly, the result (R) will be 64. [2]
  - If the necessary Conga DLLs/shared libraries are not available, R will be 32. [2]
- You can also supply a simple character vector as the right argument (Y) to **3502⌶**. [1] [2] This character vector represents the initialisation string and allows you to update the connection details. For example, to update the initialisation string to 'SERVE::4511' you would call: **3502⌶'SERVE::4511'**. [3] You could then enable RIDE with the updated initialisation string using **3502⌶1**. [3]

Important Considerations:

- **Runtime vs. Development Interpreter:**
  - **Runtime interpreters** have RIDE disabled by default, regardless of whether the RIDE_INIT environment variable is set. The only way to enable RIDE in a runtime interpreter is by calling **3502⌶1**. [4]
  - **Development interpreters** will enable RIDE if the RIDE_INIT environment variable is set on startup, assuming the RIDE DLL/shared library is available and the RIDE_INIT variable is correctly formatted. If these conditions aren't met, the interpreter will exit with a non-zero exit code. If RIDE_INIT is not set, the development interpreter will start with RIDE disabled. [4]
- **Security:** Enabling RIDE access to applications using the runtime interpreter exposes the application's APL code. **The APL code must explicitly grant permission for a RIDE client to connect to the runtime interpreter**, ensuring that connection rights are always controlled within the code itself. [3]
- **RIDE User Guide:** For more detailed information about RIDE and its usage, refer to the **RIDE User Guide**, as referenced in the sources. [1] [5] [6] [7]

Let me know if you have any specific scenarios or questions about managing RIDE connections, and I'll do my best to help!


Give me a cupcake recipe.
The sources provided contain information on programming in the Dyalog APL language. There is no information about cupcake recipes.
Save to note
What is the purpose of the `⎕PROFILE` function?
What is the difference between a fork and an atop in APL?
What is the "Syracuse conjectu
Close chat
10 sources
Start typing...
Notebook guide
NotebookLM can make mistakes, so double-check it.

RP Hi there Claude. Today, I'd like you to help me implement a k-means clustering algorithm in APL. In particular, you're going to be our mathematician and I will be your expert APLer. I have some test data, but I'd like you to help me by describing the steps to implement k-means.

We'll start with 2-dimensions. The data is the Lot Area and Sale Price of homes in NYC.

I'll help you break down the k-means clustering algorithm for 2D data focused on Lot Area and Sale Price. K-means is an iterative algorithm that aims to partition n observations into k clusters. Let me outline the key steps:

1. Initialization:
   - Choose k initial cluster centroids randomly from your data points
   - Each centroid will be a 2D point with (Lot Area, Sale Price) coordinates
2. Assignment Step:
   - For each data point, calculate its distance to all k centroids
   - Assign each point to the nearest centroid

Clustering Algorithms

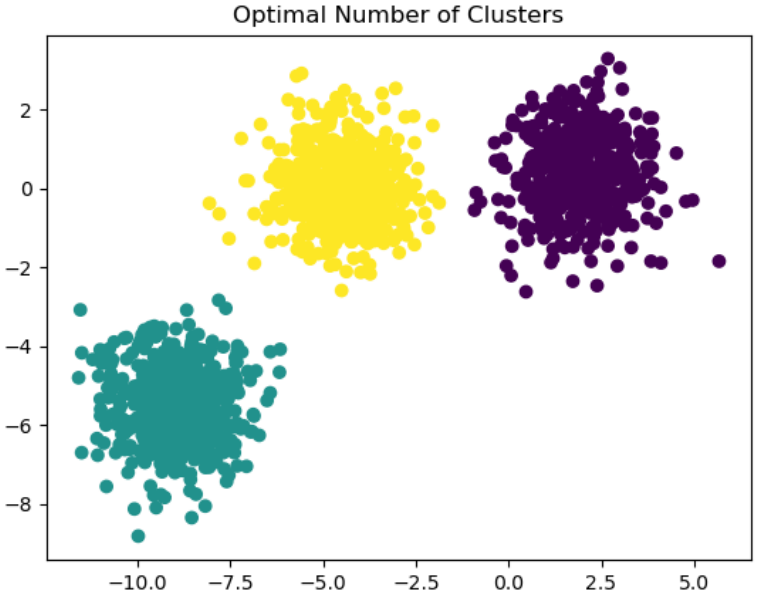


Image from Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

```
 KMeans←{
  (n max)←⍺
⍝ max: maximum iterations :: scalar integer
⍝   n: number of clusters :: scalar integer
⍝   ⍵: data set           :: numeric matrix 1 column per field
  End←{
    ⍺≡⍵:1          ⍝ converged
    i=max⊣i+←1     ⍝ maximum iterations reached
  }
  ComputeCentroids←{
    d←0.5*⍨+/×⍨⍺(-⍤1⍤1 2)⍵   ⍝ distance from points to centroids
    g←d⍳⍤1 0⌊/d              ⍝ cluster (group) for each data point
    g{(+⌿÷1⌈≠)⍵}⌸⍺           ⍝ new clusters are means of points in each group
  }
  i←0
  I←{(⊂⍺)⌷⍵}
  c←n(?⍤0≠I⊢)⍵               ⍝ guess random centroids
  ⍵ ComputeCentroids⍣End⊢c   ⍝ compute centroids
}
```

RP Those are great suggestions - but it seems like a lot of configuration for a single function. I have some ideas, but I'd like to hear how you'd go about incorporating these.

For handling multiple configuration options in APL, I can think of a few approaches:

1. Namespace-based Configuration:

apl Copy

```
∇ ns←DefaultConfig
  ns←⎕NS ''
  ns.k←3                  ⍝ number of clusters
  ns.maxiter←100          ⍝ max iterations
  ns.epsilon←1E¯6         ⍝ convergence threshold
  ns.distance←'euclidean' ⍝ or fn for custom distance
  ns.init←'random'        ⍝ or 'kmeans++' or matrix of centroids
  ns.scale←'none'         ⍝ or 'minmax' or 'zscore'
∇
```

Then users could:

apl Copy

apl Copy

```
∇ ns←DefaultConfig
  ns←⎕NS ''
  ns.k←3                    ⍝ number of clusters
  ns.maxiter←100            ⍝ max iterations
  ns.epsilon←1E¯6           ⍝ convergence threshold
  ns.distance←'euclidean'   ⍝ or fn for custom distance
  ns.init←'random'          ⍝ or 'kmeans++' or matrix of centroids
  ns.scale←'none'           ⍝ or 'minmax' or 'zscore'
∇
```

Then users could:

apl Copy

```
cfg←DefaultConfig
cfg.k←5
cfg.scale←'minmax'
data KMeans cfg
```

2. Modifier-based Approach: Split into smaller functions with modifiers:

apl Copy

```
data (k KMeans WithDistance 'manhattan') WithScale 'zscore'
```

3. Dictionary/Nested Array Configuration:

apl Copy

```
opts←⊂('k' 3)('maxiter' 100)('epsilon' 1E¯6)...
data KMeans opts
```

I'm curious which approach you were thinking of? Each has different trade-offs in terms of APL idioms and user-friendliness.

```
 KMeans←{
  (n max)←⍺
⍝ max: maximum iterations :: scalar integer
⍝   n: number of clusters :: scalar integer
⍝   ⍵: data set           :: numeric matrix 1 column per field
  End←{
    ⍺≡⍵:1          ⍝ converged
    i=max⊣i+←1     ⍝ maximum iterations reached
  }
  ComputeCentroids←{
    d←0.5*⍨+/×⍨⍺(-⍤1⍤1 2)⍵   ⍝ distance from points to centroids
    g←d⍳⍤1 0⌊/d              ⍝ cluster (group) for each data point
    g{(+⌿÷1⌈≠)⍵}⌸⍺           ⍝ new clusters are means of points in each group
  }
  i←0
  I←{(⊂⍺)⌷⍵}
  c←n(?⍤0≠I⊢)⍵               ⍝ guess random centroids
  ⍵ ComputeCentroids⍣End⊢c   ⍝ compute centroids
}
```

Convergence Threshold (ε)

```
End←{
  α≡ω:1          ⍝ converged
  i=max⊣i+←1     ⍝ maximum iterations reached
}
```

Convergence Threshold (ε)

```
End←{
    d2←+/×⍨⍺-⍵    ⍝ squared distance from previous centroids
    ∧/d2≤⎕e*2:1   ⍝ converged
    i=max⊣i+←1    ⍝ maximum iterations reached
}
```

```
 KMeans←{
  (n max)←⍺
⍝ max: maximum iterations :: scalar integer
⍝   n: number of clusters :: scalar integer
⍝   ⍵: data set           :: numeric matrix 1 column per field
 End←{
    d2←+/×⍨⍺-⍵    ⍝ squared distance from previous centroids
    ∧/d2≤e*2:1    ⍝ converged
    i=max⊣i+←1    ⍝ maximum iterations reached
 }
  ComputeCentroids←{
    d←0.5*⍨+/×⍨⍺(-⍤1⍤1 2)⍵    ⍝ distance from points to centroids
    g←⊃⍳⍤1 0⌊/d               ⍝ cluster (group) for each data point
    g{(+⌿÷1⌈≠)⍵}⌸⍺           ⍝ new clusters are means of points in each group
  }
  i←0
  I←{(⊂⍺)⌷⍵}
  c←n(?⍤0≠I⊢)⍵                ⍝ guess random centroids
  ⍵ ComputeCentroids⍣End⊢c    ⍝ compute centroids
}
```

Distance Metric

```
d←0.5*⍨+/×⍨⍺(-⍤1⍤1 2)⍵    ⍝ distance from points to centroids
```

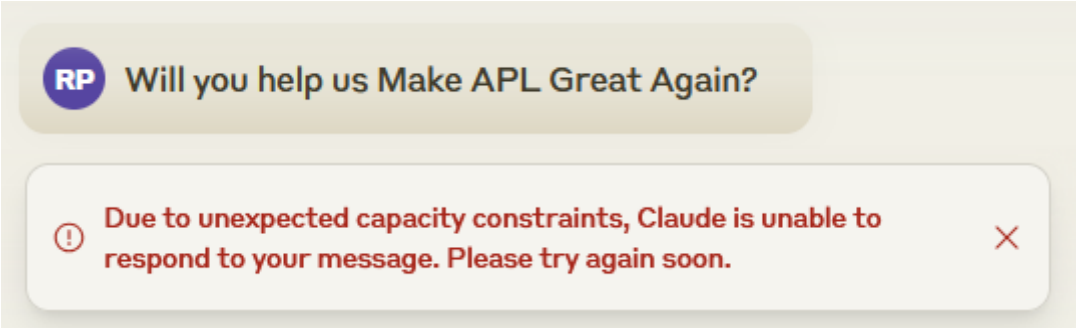
Distance Metric

```
d←0.5*⍨+/×⍨⍺(-⍤1⍤1 2)⍵          ⍝ Euclidean distance
d←+/|⍺(-⍤1⍤1 2)⍵                ⍝ Manhattan distance
d←(⍺+.×⌽⍵)÷⍺∘.×⍤{0.5*⍨+/×⍨⍵}⍵   ⍝ Cosine similarity
```

RP Where do you think is the place for tools of thought like APL in a world in which LLMs are able to answer questions about data?

APL and similar tools of thought remain vital even with LLMs because they provide precise, reproducible ways to manipulate data that build mathematical intuition. While LLMs can explain patterns, APL lets users directly express and experiment with computational ideas in a notation that mirrors mathematical thinking. This hands-on manipulation and expression of ideas complements rather than competes with LLMs' explanatory abilities.

The ideal future might combine both - using APL's notation and array-oriented thinking to clearly specify computations, while using LLMs to explain, teach, and help debug these expressions. What's your view on how these tools could work together?


RP
Will you help us Make APL Great Again?
Due to unexpected capacity constraints, Claude is unable to respond to your message. Please try again soon.