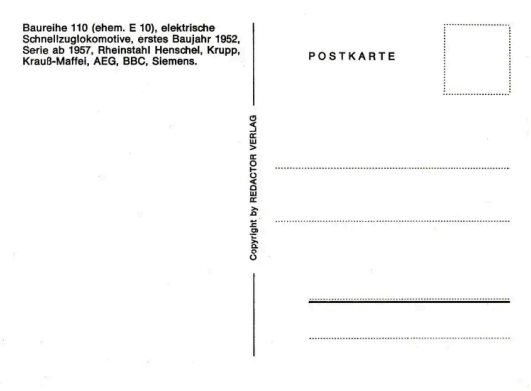






Postcard E 10

Advertisement
Privacy Settings

Katakana Language Trainer

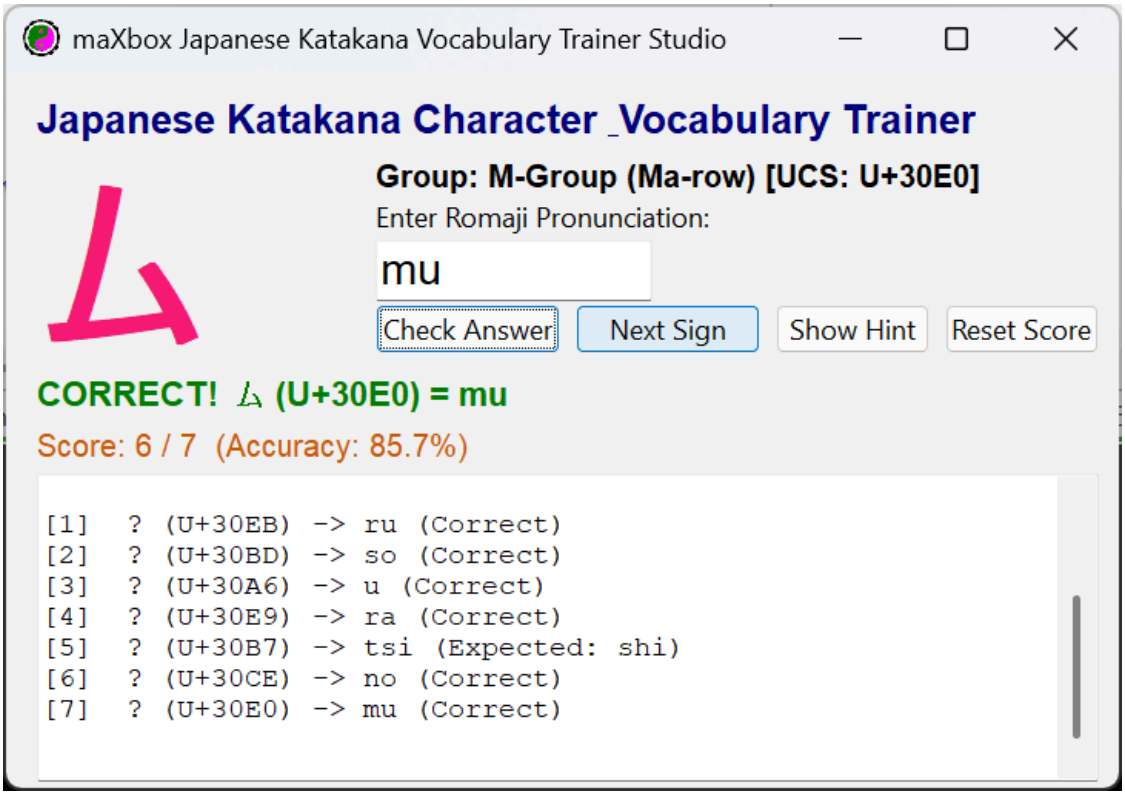
Katakana is a Japanese syllabary used alongside hiragana and kanji. It represents the same set of sounds as hiragana but uses distinct, more angular shapes. It originated from components of kanji during the 8th–9th centuries.

Wikipedia

Structure

- Each character represents a mora (a rhythmic unit in Japanese).

- Includes vowels (ア, イ, ウ, エ, オ), consonant–vowel combinations (e.g., カ, サ, タ), and the nasal ン. Wikipedia
- Long vowels are marked with ー (chōonpu).
- Dakuten (゛) and handakuten (゜) modify consonant sounds (e.g., カ→ガ, ハ→パ).



https://sourceforge.net/projects/maxbox5/files/examples/1507_lang_trainer_katakana.txt/download

Advertisement
Privacy Settings

This will be the Tutorial 205 with maXbox and JS App online.

Japanese Katakana Vocabulary & Character Trainer (46 Signs)



Japanese Katakana Vocabulary & Character Trainer (46 Signs)

This script will show Complete Syllabary Overview (katakana_language_trainer-v5.pas)

Advertisement
Privacy Settings

Interactive Features: Includes randomized card drawing, real-time Romaji validation, Unicode hex code displays (U+XXXX), loanword vocabulary hints, accuracy tracking, and session logging.

46 Total Basic Characters: Covers Vowels (A-row), K-Group, S-Group, T-Group, N-Group, H-Group, M-Group, Y-Group, R-Group, W-Group, and Singular N.

Locale-Independent Rendering: Every character uses exact WideChar hexadecimal literals (`#$30A2` through `#$30F3`) so that maXbox renders them cleanly across all regional Windows codepages.

The most I learned from this association table with hint as the last row below:

Here is the complete table of all 46 Katakana characters, their Unicode 16-bit hex codes, Romaji pronunciations, syllabary groups, and corresponding vocabulary hints:

Here is the complete table of all 46 Katakana characters, their Unicode 16-bit hex codes, Romaji pronunciations, syllabary groups, and corresponding vocabulary hints:

Complete 46-Character Katakana Vocabulary Hint Table

Katakana	Hex Code	Romaji	Syllabary Group	Vocabulary Hint & Japanese Spelling
ア	U+30A2	a	Vowels (A-row)	America (アメリカ / *amerika*)
イ	U+30A4	i	Vowels (A-row)	India (インド / *indo*)
ウ	U+30A6	u	Vowels (A-row)	Vector (ベクトル / *beku-toru*)
エ	U+30A8	e	Vowels (A-row)	Energy (エネルギ / *enerugi*)
オ	U+30AA	o	Vowels (A-row)	Orange (オレンジ / *orenji*)
カ	U+30AB	ka	K-Group (Ka-row)	Camera (カメラ / *kamera*)
キ	U+30AD	ki	K-Group (Ka-row)	King (キング / *kingu*)
ク	U+30AF	ku	K-Group (Ka-row)	Class (クラス / *kurasu*)
ケ	U+30B1	ke	K-Group (Ka-row)	Cake (ケーキ / *keeki*)

コ	U+30B3	ko	K-Group (Ka-row)	Coffee (コーヒー / *koohii*)
サ	U+30B5	sa	S-Group (Sa-row)	Salad (サラダ / *sarada*)
シ	U+30B7	shi	S-Group (Sa-row)	Shirt (シャツ / *shatsu*)
ス	U+30B9	su	S-Group (Sa-row)	Sports (スポーツ / *supootsu*)
セ	U+30BB	se	S-Group (Sa-row)	Center (センター / *sentaa*)
ソ	U+30BD	so	S-Group (Sa-row)	Sofa (ソファー / *sofaa*)
タ	U+30BF	ta	T-Group (Ta-row)	Taxi (タクシー / *takushii*)
チ	U+30C1	chi	T-Group (Ta-row)	Team (チーム / *chiimu*)
ツ	U+30C4	tsu	T-Group (Ta-row)	Tour (ツアー / *tsuaa*)
テ	U+30C6	te	T-Group (Ta-row)	Tennis (テニス / *tenisu*)
ト	U+30C8	to	T-Group (Ta-row)	Tomato (トマト / *tomato*)
ナ	U+30CA	na	N-Group (Na-row)	Knife (ナイフ / *naifu*)
ニ	U+30CB	ni	N-Group (Na-row)	News (ニュース / *nyuusu*)

ヌ	U+30CC	nu	N-Group (Na-row)	Noodle (ヌードル / *nuudoru*)
ネ	U+30CD	ne	N-Group (Na-row)	Necktie (ネクタイ / *nekutai*)
ノ	U+30CE	no	N-Group (Na-row)	Note (ノート / *nooto*)
ハ	U+30CF	ha	H-Group (Ha-row)	Ham (ハム / *hamu*)
ヒ	U+30D2	hi	H-Group (Ha-row)	Hero (ヒロー / *hiroo*)
フ	U+30D5	fu	H-Group (Ha-row)	Fork (フォーク / *fooku*)
ヘ	U+30D8	he	H-Group (Ha-row)	Helicopter (ヘリコプター / *herikoputaa*)
ホ	U+30DB	ho	H-Group (Ha-row)	Hotel (ホテル / *hoteru*)
マ	U+30DE	ma	M-Group (Ma-row)	Mask (マスク / *masuku*)
ミ	U+30DF	mi	M-Group (Ma-row)	Milk (ミルク / *miruku*)
ム	U+30E0	mu	M-Group (Ma-row)	Music (ミュージック / *myuujikku*)
メ	U+30E1	me	M-Group (Ma-row)	Menu (メニュー / *menyuu*)

モ	U+30E2	mo	M-Group (Ma-row)	Monitor (モニター / *monitaa*)
ヤ	U+30E4	ya	Y-Group (Ya-row)	Yahoo (ヤフー / *yafuu*)
ユ	U+30E6	yu	Y-Group (Ya-row)	User (ユーザー / *yuuzaa*)
ヨ	U+30E8	yo	Y-Group (Ya-row)	Yogurt (ヨーグルト / *yooguruto*)
ラ	U+30E9	ra	R-Group (Ra-row)	Radio (ラジオ / *rajio*)
リ	U+30EA	ri	R-Group (Ra-row)	Ribbon (リボン / *ribon*)
ル	U+30EB	ru	R-Group (Ra-row)	Rule (ルール / *ruuru*)
レ	U+30EC	re	R-Group (Ra-row)	Lemon (レモン / *remon*)
ロ	U+30ED	ro	R-Group (Ra-row)	Robot (ロボット / *robotto*)
ワ	U+30EF	wa	W-Group (Wa-row)	Wine (ワイン / *wain*)
ヲ	U+30F2	wo	W-Group (Wa-row)	Particle Wo (ヲ)
ン	U+30F3	n	N-Singular	Pan / Bread (パン / *pan*)

Advertisement
Privacy Settings

⛩️ Would you like to build an interactive quiz or flashcard deck from these 46 Katakana vocabulary pairs directly in your Studio panel?

Ref: Character Sets: Japanese Hiragana and Katakana

Key Features of this maXbox Web Application

1. Full 46-Character Syllabary: Incorporates all Katakana signs across 11 groups (Vowels, K, S, T, N, H, M, Y, R, W, and singular N).
2. Unicode Hex display: Displays the exact U+XXXX code point for every character.
3. Interactive Practice Engine: Supports keyboard (`Enter` key) navigation, immediate Romaji scoring, loanword hints, and real-time accuracy logging.

How does the script works:

The Katakana Language Trainer (`katakana_language_trainer-v5.pas` / `katakana_trainer_v5.html`) operates through five core functional components:

---

1. Dataset Initialization (`InitKatakanaDataset`)

- Stores all 46 basic Katakana signs across 11 syllabary groups.
- Uses UCS/Unicode 16-bit WideChar hex codes (e.g., `#$30A2` for ア, `#$30AB` for カ, `#$30F3` for ン) to ensure exact character rendering without regional Windows ANSI code-page issues.
- Pairs each character with:
    1. Its Romaji target string (e.g., `ka`, `shi`, `pan`).
    2. Its Syllabary Group (e.g., `K-Group`, `N-Singular`).
    3. A Vocabulary Hint with Japanese loanword spelling (e.g., `Pan / Bread (パン / pan)`).

Advertisement
Privacy Settings

2. Random Card Selection (`LoadNewCharacter`)

- Generates a pseudo-random index using `Random(46)`.
- Updates the main display label with the large 64pt Katakana character, its `U+XXXX` hex code badge, and its group title.
- Clears the input field (`EdtAnswer.Text := ''`), focuses the cursor, and sets `IsChecked := False`.

---

3. Answer Verification & Scoring (`CheckAnswer`)

- Sanitizes user input by trimming whitespace and converting to lowercase.
- Compares user input against the expected Romaji pronunciation:
  - Correct: Increments `CorrectCount`, highlights feedback in Green, and logs `[Attempt #] Symbol (U+Hex) -> User Input (Correct)`.
  - Incorrect: Highlights feedback in Red displaying expected vs. typed answers, and logs the attempt.
- Recalculates real-time accuracy percentage: $$ \text{Accuracy %} = \frac{\text{Correct Count}}{\text{Total Attempts}} \times 100 $$

---

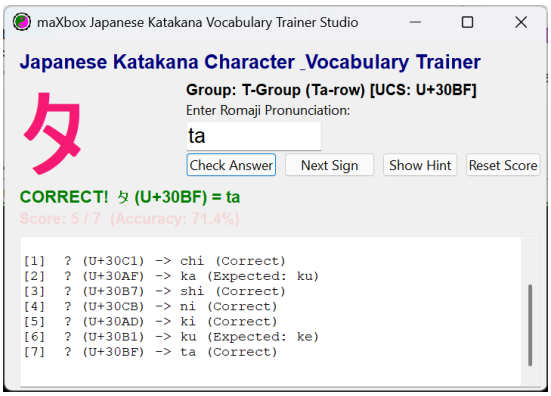
4. Hint System (`ShowHint`)

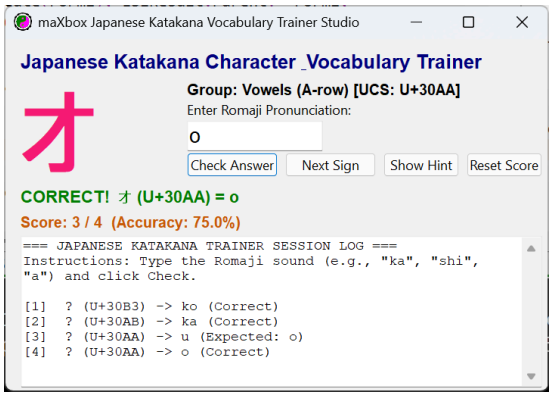
- Triggers a modal dialog displaying the character's syllabary group and its associated loanword hint without revealing the exact Romaji answer directly.

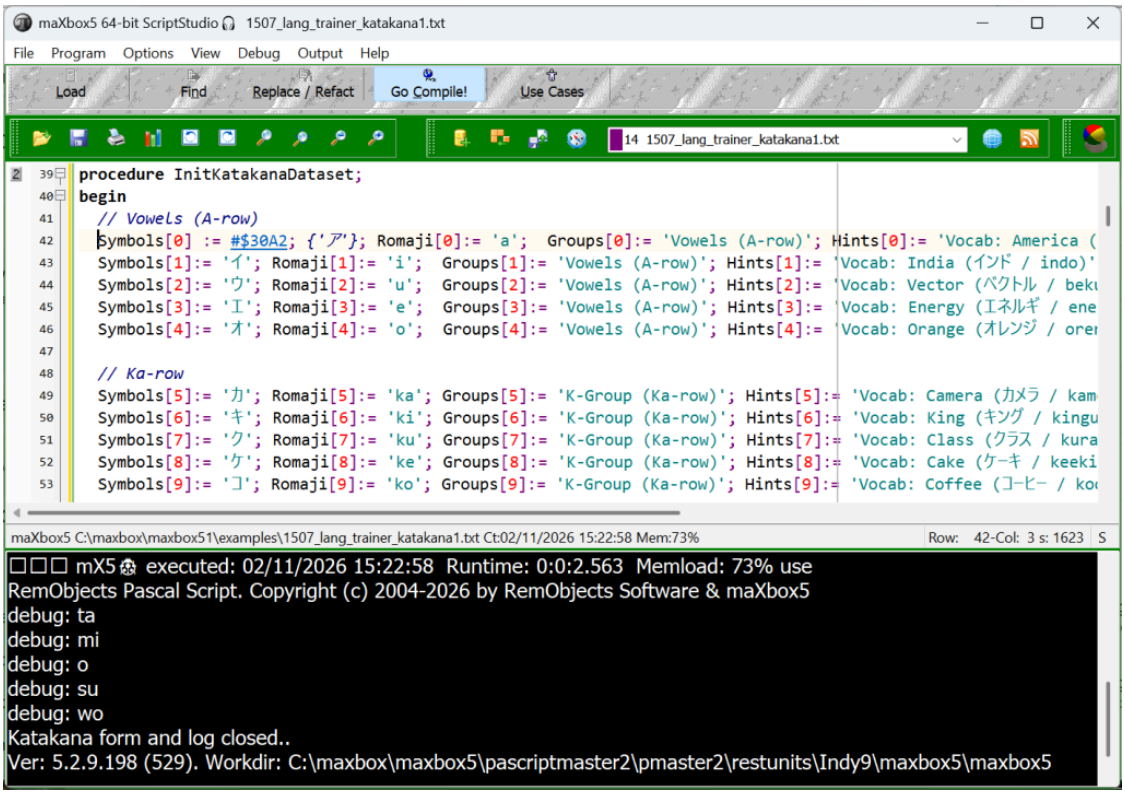
Advertisement
Privacy Settings

5. Event Handling & GUI Setup (`ShowGUI`)

- Instantiates window components (`TForm`, `TLabel`, `TEdit`, `TButton`, `TMemo`).
- Binds action buttons to Pascal procedure pointers (`@CheckAnswer`, `@LoadNewCharacter`, `@ShowHint`, `@ResetStats`).
- Supports rapid practice by allowing users to press Enter or click Check Answer to validate and advance to the next sign.







https://sourceforge.net/projects/maxbox5/files/examples/1507_lang_trainer_katakana1.txt/download

Advertisement
Privacy Settings

I have converted all Japanese Katakana loanwords across all 46 vocabulary hints into pure WideChar hexadecimal literals (#$XXXX) and published the updated script `katakana_language_trainer-v6.pas` to your Studio panel.

Sample Object Pascal Initialization Syntax (v6)

```
// Vowels & Ka-row
Hints := 'Vocab: America ('#$30A2#$30E1#$30EA#$30AB' / amerika)
Hints := 'Vocab: India ('#$30A4#$30F3#$30C9' / indo)';
Hints := 'Vocab: Camera ('#$30AB#$30E1#$30E9' / kamera)';
Hints := 'Vocab: Cake ('#$30B1#$30FC#$30AD' / keeki)';
```

```
Hints := 'Vocab: Pan / Bread ('#$30D1#$30F3' / pan)';
```

Complete 46-Character Hexadecimal Vocabulary Hint Mapping

Katakana	Hex Code	Japanese Word	Pascal WideString Hex Expression
ア	#$30A2	アメリカ	'Vocab: America ('#$30A2#$30E1#$30EA#$30AB' / amerika)'
イ	#$30A4	インド	'Vocab: India ('#$30A4#$30F3#$30C9' / indo)'
ウ	#$30A6	ベクトル	'Vocab: Vector ('#$30D9#$30AF#$30C8#$30EB' / beku-toru)'
エ	#$30A8	エネルギ	'Vocab: Energy ('#$30A8#$30CD#$30EB#$30AE' / enerugi)'
オ	#$30AA	オレンジ	'Vocab: Orange ('#$30AA#$30EC#$30F3#$30B8' / orenji)'

カ	#$30AB	カメラ	'Vocab: Camera ('#$30AB#$30E1#$30E9' / kamera)'
キ	#$30AD	キング	'Vocab: King ('#$30AD#$30F3#$30B0' / kingu)'
ク	#$30AF	クラス	'Vocab: Class ('#$30AF#$30E9#$30B9' / kurasu)'
ケ	#$30B1	ケーキ	'Vocab: Cake ('#$30B1#$30FC#$30AD' / keeki)'
コ	#$30B3	コーヒー	'Vocab: Coffee ('#$30B3#$30FC#$30D2#$30FC' / koohii)'
サ	#$30B5	サラダ	'Vocab: Salad ('#$30B5#$30E9#$30C0' / sarada)'
シ	#$30B7	シャツ	'Vocab: Shirt ('#$30B7#$30E3#$30C4' / shatsu)'
ス	#$30B9	スポーツ	'Vocab: Sports ('#$30B9#$30DD#$30FC#$30C4

			' / supootsu)'
セ	#$30BB	センター	'Vocab: Center ('#$30BB#$30F3#$30BF#$30FC' / sentaa)'
ソ	#$30BD	ソファー	'Vocab: Sofa ('#$30BD#$30D5#$30A1#$30FC' / sofaa)'
タ	#$30BF	タクシー	'Vocab: Taxi ('#$30BF#$30AF#$30B7#$30FC' / takushii)'
チ	#$30C1	チーム	'Vocab: Team ('#$30C1#$30FC#$30E0' / chiimu)'
ツ	#$30C4	ツアー	'Vocab: Tour ('#$30C4#$30A2#$30FC' / tsuaa)'
テ	#$30C6	テニス	'Vocab: Tennis ('#$30C6#$30CB#$30B9' / tenisu)'
ト	#$30C8	トマト	'Vocab: Tomato ('#$30C8#$30DE#$30C8' / tomato)'

ナ	#$30CA	ナイフ	'Vocab: Knife ('#$30CA#$30A4#$30D5' / naifu)'
ニ	#$30CB	ニュース	'Vocab: News ('#$30CB#$30E5#$30FC#$30B9' / nyuusu)'
ヌ	#$30CC	ヌードル	'Vocab: Noodle ('#$30CC#$30FC#$30C9#$30EB' / nuudoru)'
ネ	#$30CD	ネクタイ	'Vocab: Necktie ('#$30CD#$30AF#$30BF#$30A4' / nekutai)'
ノ	#$30CE	ノート	'Vocab: Note ('#$30CE#$30FC#$30C8' / nooto)'
ハ	#$30CF	ハム	'Vocab: Ham ('#$30CF#$30E0' / hamu)'
ヒ	#$30D2	ヒロー	'Vocab: Hero ('#$30D2#$30ED#$30FC' / hiroo)'
フ	#$30D5	フォーク	'Vocab: Fork ('#$30D5#$30A9#$30FC#$30AF' / fooku)'

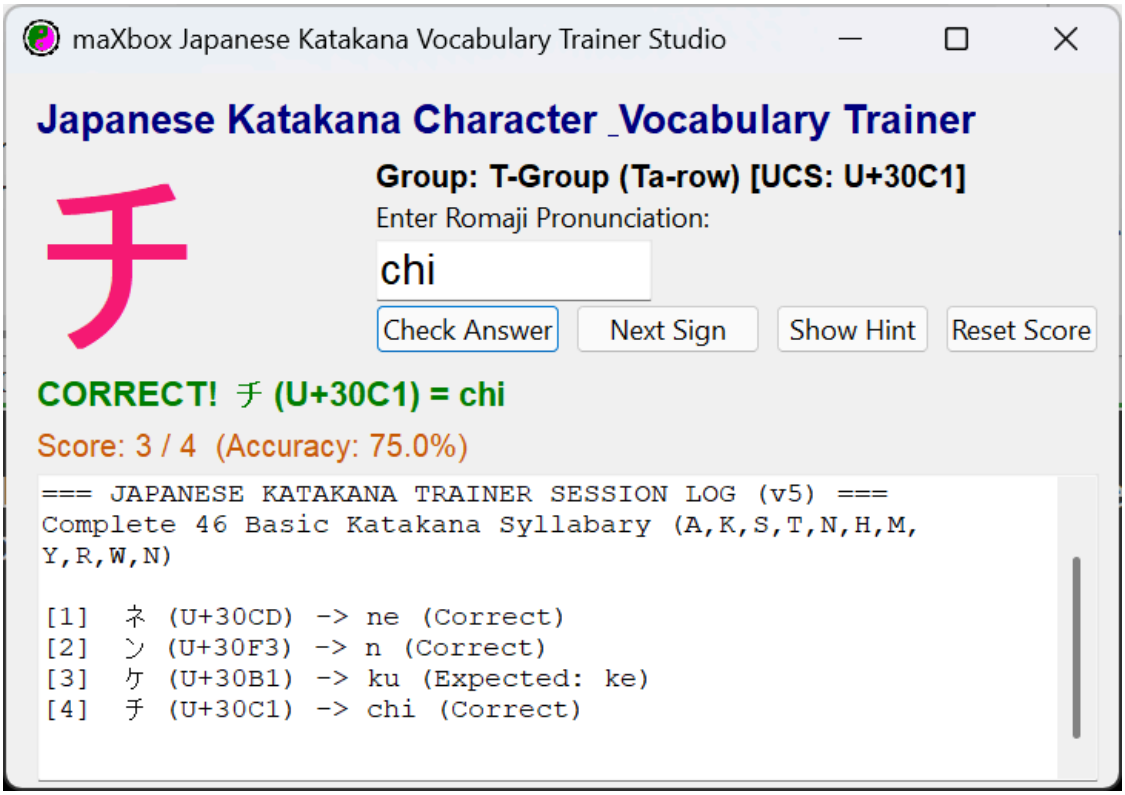
へ	#$30D8	ヘリコプター	'Vocab: Helicopter ('#$30D8#$30EA#$30B3#$30D7#$30BF#$30FC' / herikoputaa)'
ホ	#$30DB	ホテル	'Vocab: Hotel ('#$30DB#$30C6#$30EB' / hoteru)'
マ	#$30DE	マスク	'Vocab: Mask ('#$30DE#$30B9#$30AF' / masuku)'
ミ	#$30DF	ミルク	'Vocab: Milk ('#$30DF#$30EB#$30AF' / miruku)'
ム	#$30E0	ミュージック	'Vocab: Music ('#$30DF#$30E5#$30FC#$30B8#$30C3#$30AF' / myuujikku)'
メ	#$30E1	メニュー	'Vocab: Menu ('#$30E1#$30CB#$30E5#$30FC' / menyuu)'
モ	#$30E2	モニター	'Vocab: Monitor ('#$30E2#$30CB#$30BF#$30FC' / monitaa)'

ヤ	#$30E4	ヤフー	'Vocab: Yahoo ('#$30E4#$30D5#$30FC' / yafuu)'
ユ	#$30E6	ユーザー	'Vocab: User ('#$30E6#$30FC#$30B6#$30FC' / yuuzaa)'
ヨ	#$30E8	ヨーグルト	'Vocab: Yogurt ('#$30E8#$30FC#$30B0#$30EB#$30C8' / yooguruto)'
ラ	#$30E9	ラジオ	'Vocab: Radio ('#$30E9#$30B8#$30AA' / rajio)'
リ	#$30EA	リボン	'Vocab: Ribbon ('#$30EA#$30DC#$30F3' / ribon)'
ル	#$30EB	ルール	'Vocab: Rule ('#$30EB#$30FC#$30EB' / ruuru)'
レ	#$30EC	レモン	'Vocab: Lemon ('#$30EC#$30E2#$30F3' / remon)'
ロ	#$30ED	ロボット	'Vocab: Robot ('#$30ED#$30D

			C#$30C3#$30C8' / robotto)'
ワ	#$30EF	ワイン	'Vocab: Wine ('#$30EF#$30A4#$30F3' / wain)'
ヲ	#$30F2	ヲ	'Vocab: Particle Wo ('#$30F2' / wo)'
ン	#$30F3	パン	'Vocab: Pan / Bread ('#$30D1#$30F3' / pan)'

Advertisement
Privacy Settings

Would you like to generate a practice quiz or study guide based on these Unicode Katakana vocabulary pairs?



https://sourceforge.net/projects/maxbox5/files/examples/1507_lang_trainer_katakana1.txt/download

In maXbox (and Delphi / Object Pascal), UCS/Unicode WideChar hex literals allow you to specify exact Unicode character code points directly in your source code without depending on system text encodings.

Advertisement
Privacy Settings

Here is how they work under the hood:

---

1. Syntax Breakdown (`#$XXXX`)

The syntax consists of three distinct compiler instructions:

- `#`: Indicates a character literal (ASCII or Unicode) rather than a number.
- `$`: Designates a hexadecimal (base-16) value.
- `XXXX`: The 4-digit 16-bit hexadecimal representation of the Unicode code point (e.g., `30A2` = \(12,450_{10}\)).

For example, `#$30A2` explicitly instructs the interpreter to insert the 16-bit WideChar for U+30A2 (the Katakana character ア).

---

2. 16-Bit UCS-2 / UTF-16 Memory Representation

maXbox's PascalScript engine handles string buffers using 16-bit WideChar units (`WideString` / `UnicodeString`):

- Basic Multilingual Plane (BMP): Standard characters (including Latin, Japanese Katakana/Hiragana `U+3040`–`U+30FF`, Greek, Cyrillic) fit within 16 bits (`U+0000` to `U+FFFF`). A single `#$XXXX` literal directly stores one character in memory.
- Literal Chaining: Multiple hex character literals can be concatenated directly alongside regular quoted text without using the `+` operator: `// Chaining hex literals for 'アメリカ' inside a string: 'Vocab: America ('#$30A2#$30E1#$30EA#$30AB' / amerika)'`

Advertisement
Privacy Settings

3. System Code-Page & Locale Independence

The most critical reason for using `#$XXXX` hex literals is cross-platform and locale portability:

- The Problem with Raw Strings: Standard quoted text strings like `'アメリカ'` rely on the Windows system's active ANSI code page (e.g., Code Page 932 for Japanese vs. Code Page 1252 for Western European). Running raw non-ASCII string literals on non-Japanese Windows locales often corrupts characters into `?` or broken glyphs.
- The Hex Solution: `#$30A2` bypasses the OS code-page converter completely by writing the exact 16-bit Unicode value into the string buffer, ensuring identical rendering across all Windows regional settings.

---

4. Supplementary Unicode Planes & Emojis (Surrogate Pairs)

For Unicode code points above `U+FFFF` (such as Emojis in the Supplementary Multilingual Plane):

- UTF-16 splits the character into two 16-bit surrogate units (a high surrogate and a low surrogate).
- In maXbox scripts, emojis are written by chaining two hex literals together: `// U+1F642 (Slightly Smiling Face) = High Surrogate $D83D + Low Surrogate $DE42 Writeln(#$D83D#$DE42);`

🎌 Would you like to build an interactive maXbox script that auto-converts any typed UTF-8 or ANSI string into its `#$XXXX` WideChar hex literal representation?

https://sourceforge.net/projects/maxbox5/files/examples/1507_utf8_to_pascal_hex_converter.pas/download

Advertisement
Privacy Settings

How the Generator Works

1. WideString Ordinal Inspection: Iterates through each 16-bit character of the input string using `Ord(S[i])`.
2. Hex Formatting: Formats each character code into a 4-digit uppercase hexadecimal string (`IntToHex(Code, 4)`).
3. Pascal Code Generation: Constructs ready-to-use Object Pascal assignment statements:
    - Input: アメリカ
    - Generated Output: `'Vocab: America ('#$30A2#$30E1#$30EA#$30AB')'`

In UTF-16 (the internal encoding used by maXbox `WideString` / `UnicodeString` buffers), characters are stored in 16-bit (2-byte) code units.

While standard characters (like Latin, Greek, or Katakana `U+30A2`) fit inside a single 16-bit number (`U+0000` to `U+FFFF`), Emojis and advanced symbols live in the Supplementary Multilingual Plane (`U+10000` to `U+10FFFF`). Because these values exceed 16 bits, UTF-16 encodes them using a surrogate pair: two consecutive 16-bit `WideChar` units.

---

1. Reserved Surrogate Ranges

Unicode reserves 2,048 special code points specifically for building surrogate pairs:

- High Surrogates: `0xD800` to `0xDBFF` (1,024 reserved values)
- Low Surrogates: `0xDC00` to `0xDFFF` (1,024 reserved values)

Neither a high nor low surrogate value represents a valid standalone character on its own — text renderers only display the character when the two are combined back-to-back.

Advertisement
Privacy Settings

2. The Encoding Math (Step-by-Step)

To convert any Emoji code point U (where $U \ge \text{0x10000}$) into a maXbox surrogate pair (W_1, W_2):

1. Subtract `0x10000` from the code point: $$U' = U - \text{0x10000}$$ *(This shifts the 21-bit range down to a 20-bit number between `0x00000` and `0xFFFFF`).*
2. Split U' into 10-bit halves:
   - High 10 bits: $H_{10} = U' \gg 10$
   - Low 10 bits: $L_{10} = U' \text{ AND } \text{0x3FF}$
3. Add the base offsets:
   - High Surrogate (W_1): $\text{0xD800} + H_{10}$
   - Low Surrogate (W_2): $\text{0xDC00} + L_{10}$

---

3. Worked Example: 🙄 Slightly Smiling Face (`U+1F642`)

Let's calculate the exact maXbox `#$XXXX#$YYYY` hex literal for `U+1F642`:

1. Subtract `0x10000`: $[\text{0x1F642} – \text{0x10000} = \mathbf{\text{0x0F642}}]$
2. Extract 10-bit halves:
   - High 10 bits: $\text{0x0F642} \gg 10 = \mathbf{\text{0x3D}}$ (61_{10})
   - Low 10 bits: $\text{0x0F642} \text{ AND } \text{0x3FF} = \mathbf{\text{0x242}}$ (578_{10})
3. Calculate Surrogates:
   - High Surrogate (W_1): $\text{0xD800} + \text{0x3D} = \mathbf{\text{0xD83D}}$
   - Low Surrogate (W_2): $\text{0xDC00} + \text{0x242} = \mathbf{\text{0xDE42}}$

In maXbox Object Pascal, this emoji is written by chaining the two WideChar hex units together:

```
Writeln('Hello '#$D83D#$DE42); // Displays: Hello 🙂
```

---

4. Reconstructing the Code Point in Pascal

If you have a surrogate pair (W_1, W_2) in maXbox and want to reconstruct the original 32-bit Unicode code point:

```
function SurrogateToCodePoint(const HighSurr, LowSurr: Word): C
begin
  Result := ((HighSurr - $D800) shl 10) + (LowSurr - $DC00) + $
end;
```

💡 Would you like me to update `utf8_to_pascal_hex_converter.pas` so it automatically detects Emojis above `U+FFFF` and outputs their valid two-unit `#$D8XX#$DCXX` surrogate pair expressions?

Advertisement
Privacy Settings

Conclusion

We've built a robust, fully localized Katakana Syllabary & Vocabulary Trainer (`katakana_language_trainer-v6.pas`) for maXbox.

Here is a structural review of how the script evolved, its core architecture, and key technical highlights:

---

1. Complete 46-Character Syllabary Coverage

The dataset covers the entire basic Japanese Katakana syllabary across 11 distinct phonetic groups:

- Vowels (A-row): ア, イ, ウ, エ, オ
- K-Group (Ka-row): カ, キ, ク, ケ, コ
- S-Group (Sa-row): サ, シ, ス, セ, ソ
- T-Group (Ta-row): タ, チ, ツ, テ, ト
- N-Group (Na-row): ナ, ニ, ヌ, ネ, ノ
- H-Group (Ha-row): ハ, ヒ, フ, ヘ, ホ
- M-Group (Ma-row): マ, ミ, ム, メ, モ
- Y-Group (Ya-row): ヤ, ユ, ヨ
- R-Group (Ra-row): ラ, リ, ル, レ, ロ
- W-Group & Singular N: ワ, ヲ, ン

---

2. 100% Locale-Independent WideChar Hex Encoding

Advertisement
Privacy Settings

To prevent ANSI code-page rendering corruption (`?` or broken glyphs) across different Windows regional settings:

- Character Symbols: Each character is declared via WideChar hex literals (e.g., `#$30A2` for ア through `#$30F3` for ン).
- Vocabulary Hints: All Japanese loanword spellings are constructed using WideChar hex concatenations: `// America (アメリカ) Hints := 'Vocab: America ('#$30A2#$30E1#$30EA#$30AB' / amerika)'; //`

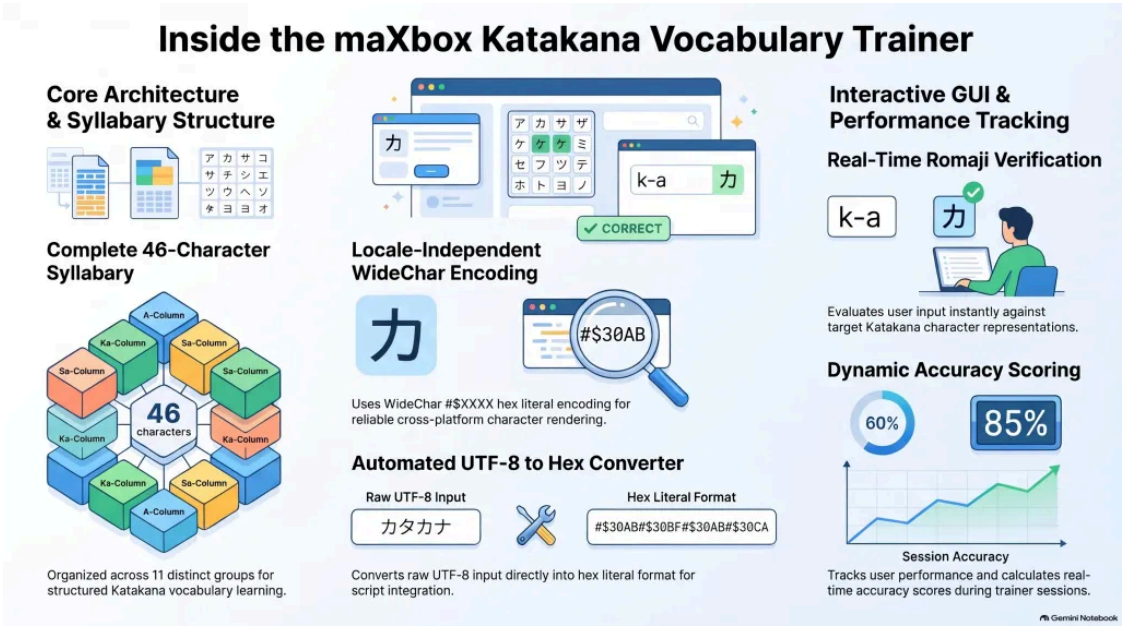
```
Bread/Pan (パン) Hints := 'Vocab: Pan / Bread
('#$30D1#$30F3' / pan)';
```

---

3. GUI Engine & User Experience

- Dynamic Card Renderer: Picks a random card (`Random(46)`), updating the large 64pt Katakana display, the group category, and the U+XXXX hex tag.
- Validation & Instant Feedback: Evaluates user Romaji input against the expected string, rendering results in Green (Correct) or Red (Incorrect) with correct answer reveal.
- Real-time Accuracy Calculation: $$ \text{Accuracy %} = \frac{\text{Correct Answers}}{\text{Total Attempts}} \times 100 $$
- Session Memo Logger: Logs every attempt with attempt index, Katakana symbol, Unicode hex tag, user input, and result status for review.

⛩️ Would you like to add SAPI speech audio synthesis (`SpVoice`) so maXbox reads the Romaji pronunciations out loud when checking answers, or add the 23 voiced Dakuten / Handakuten variations (ガ, ザ, ダ, バ, パ)?



Supervised Learning