Paper 184-2025

Fun With SAS® and Emoji: What Might a Rebus-Influenced Programming Language Look Like?

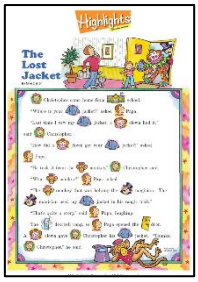
Ted Conway, Chicago, IL

ABSTRACT

Remember those fun *Highlights for Children* stories in which words were replaced with pictures to help and engage young readers?

Ever wonder what that might look like in a programming language?

In this paper, we'll not only take a whimsical look at some examples of rebus-flavored SAS® and SQL code snippets, but also look at some rudimentary SAS and Python preprocessor code snippets to translate programs with emojis back into executable code using both SAS's Unicode string 'K' functions and the Python regex package.



HIGHLGHTS REBUS STORY

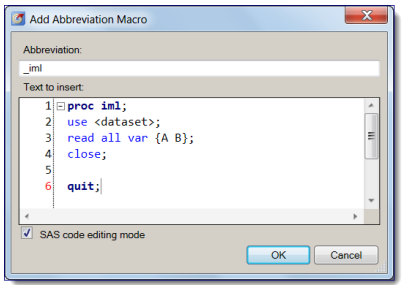
INTRODUCTION

One of my most-viewed SAS Communities posts, oddly enough, was a half-joking, half-serious 2022 post that proposed the creation and use of a single-character, emoji-like "running semicolon" character to replace the SAS "RUN;" statement. While the post received some 3,000+ views, it received only one "like", and elicited several respectful-but-sternly-disapproving responses (no ☹️'s). So, why in 2025 would I decide to follow-up with a paper proposing that emojis or images be incorporated into program languages? Well, let me try to explain in the next few pages! 😄

RUNNING SEMICOLON

WHY EMOJIS?

The idea of using built-in and user-defined custom keyboard shortcuts and abbreviations to increase one's efficiency and accuracy, especially in programming editors, is hardly controversial. And it's certainly nothing new – keyboard shortcuts could be found on both the mainframe and PCs more than four decades ago! But what if instead of placing what can be a lengthy amount of text associated with a shortcut key in our code, how about if we instead just placed a meaningful single-character emoji or image into the code that represents the associated text? By doing so, we not only save ourselves some typing, we also now get the benefit of maintaining our source code at a somewhat higher level – a single emoji or image, rather than all of the text that's assoicated with it.



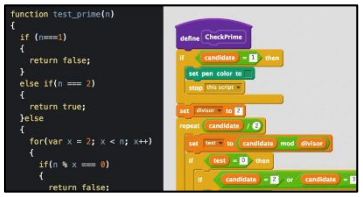
SAS ABBREVIATION

Another staple of program editors is syntax highlighting, a feature to display source code in different colors and fonts to help make understanding code snippets easier for readers. The use of emojis offers an additional way to distinguish coding elements. Like syntax highlighting, the use of emojis is intended only for human readers and does not affect the meaning of the text it represents, which is what will still be passed to the interpreter or compiler for processing.

```
1 PROC SQL;
2 CREATE TABLE MYCARS AS
3 SELECT MODEL, AVG(MPG_CITY)
4 FROM SASHELP.CARS
5 GROUP BY 1 ORDER BY 1;
```

SYNTAX HIGLIGHTING

Finally, as cellphone owners can attest to, emojis provide a sense of fun and whimsy that has certainly contributed to their wild popularity and usage in communication. For complete newcomers to programming and those making a transition to text from block-based languages such as *Scratch*, a hybrid text and emoji programming language may help make a new, otherwise wall-of-text programming language more approachable.

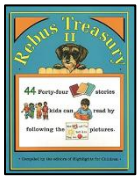


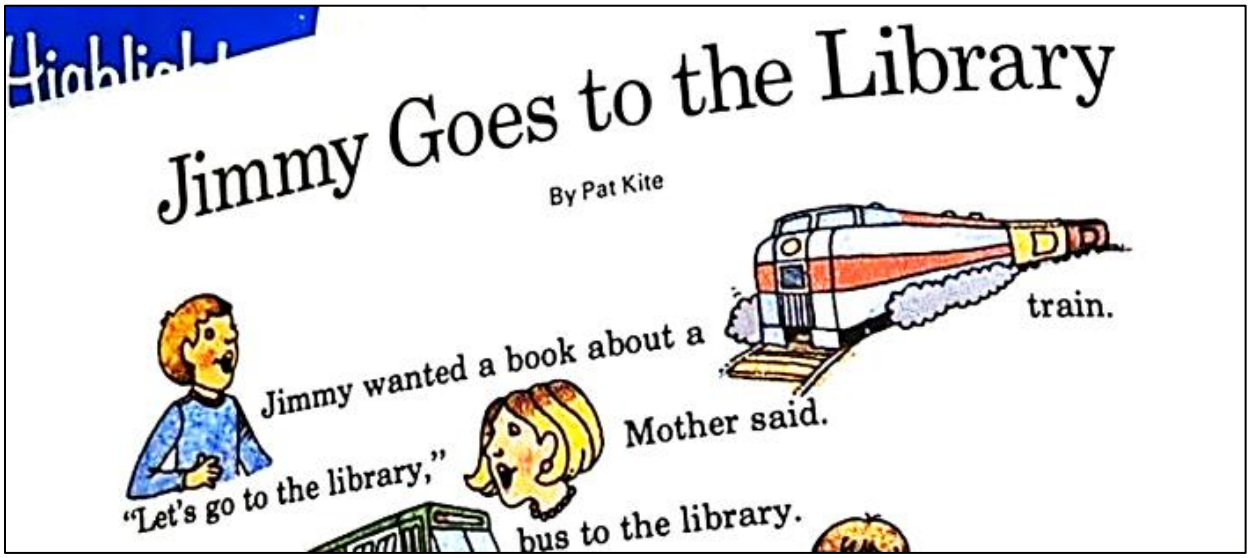
TEXT VS. BLOCK-BASED CODING

So, what might an emoji-influenced programming language look like?

AN EMOJI-INFLUENCED PROGRAMMING LANGUAGE?

For inspiration for what an emoji-influenced programming language might look like, one needn't look any further than the so-called rebus stories for children featured in *Highlights Magazine for Kids*, where images replace or complement words in sentences. In a rebus-inspired coding language, elements including keywords, variable names, values, and macro names would be replaced by or complemented with pictograms.





SAS CODE (TEXT)	SAS CODE (EMOJI)

data test;
set;
if a=b then c=d; else e=f;
if g=h then i=j; else k=substr(L, 1, 3);
dt=date(); tm=time();
proc print; run;
	💾 test;
	set;
	❓ a=b ✅ c=d; ❎ e=f;
	🤔 g=h 👍 i=j; 👎 k=✂️(L, 1, 3);
	dt=🗓️(); tm=🕒();
	proc 🖨️; 🏃

HIGHLIGHTS MAGAZINE-INSPIRED SAS AND EMOJI DATA STEP

A QUICK & DIRTY WORKING PROTOTYPE

With an idea of what an emoji-inspired coding language might look like, we turn our attention to what's needed to make a rudimentary working prototype for a demo.

```
)ATTR
  % TYPE(TEXT) INTENS(HIGH)
  + TYPE(TEXT) INTENS(LOW)
  _ TYPE(INPUT) INTENS(HIGH)
```

IBM DMS MAP

First, we need a way for users to **map emoji to the text it represents**. IBM's ISPF Dialog Manager has (*since 1980!*) allowed mainframe programmers to use ANSI characters like '+' and '_' in screen designs as a shortcut for attributes for data entry and other display fields. These mappings, which provide simple 1-character macro like functionality, are specified ahead of use in what IBM calls an **ATTR**ibute section.

Next, we'll need to **parse the emoji-laden code, replace emojis with the text they represent** (using our emoji → text mapping), and finally **emit the translated emoji-free text to the programming language interpreter** for execution. As part of this process, we'll of course need to be able to identify emojis in a stream of text which, as we'll see, is surprisingly complicated.

Now let's try to construct a rudimentary emoji pre-processor to demo how things might work!

💾	DATA
❓	IF
✅	THEN
❎	ELSE
✂️	SUBSTR

EMOJI MAP

PLAN A: PICTOGRAM MACROS

To a SAS user, the above requirements might at first glance look like they could be satisfied with a combination of single-emoji macro variables and statement-style macros. But – for now at least – SAS macro variable names must begin with a letter or an underscore and macro names are subject to standard SAS naming conventions (SAS does support emoji name literals like '😊'N). Oh, well!

```
%macro 🎀;
substr
%mend;
```

🎀 MACRO

PLAN B: A QUICK-AND-DIRTY SAS EMOJI PREPROCESSOR

With that approach ruled out, our Plan B is to cobble together a simple SAS emoji preprocessor that will:

1. Read in the user's SAS code containing emojis and text
2. Use a SAS format to replace emojis with the text they represent
3. Write the text-only code to a temporary file that will in turn be run by SAS (using %INCLUDE)

Complicating our preprocessor is that character representation in the Unicode and UTF-8 world of emojis is quite a leap from the ASCII world, where each character requires only one byte. With Unicode, one character may now occupy 1-4 bytes. So, while the letter 'A' may still be represented as a single-byte x'41', the 'Thinking Face' emoji (🤔) is represented as 'F09FA494' (4 bytes) and the 'Check Mark' emoji (✅) is represented as x'E29C85' (3 bytes).

To process this kind of multi-byte encoding, SAS provides a set of specialized string functions called the SAS K functions (K is short for Kanji!). The K functions – KSUBSTR, KLENGTH, KINDEX, etc. – operate on characters rather than bytes. As such, we utilize these K functions in our emoji preprocessor that's shown below.

```
proc format;                * Define emojis and associated text;
value $emojif '🤔'='if' '✅'='then' '❎'='else' '🏃'='run;';

data _null_;                * Read in code, replace emojis w/associated text;
length chr $ 255.;
infile "/home/ted.conway/test_pgm_in.sas" lrecl=32767 recfm=f truncover length=l;
file "/home/ted.conway/test_pgm_out.sas" recfm=f;
input pgm $varying32767. l;
do c=1 to klength(pgm);
  chr=put(ksubstr(pgm,c,1), $emojif.);
  lo=length(chr); put chr $varying. lo@;
end;
run;

options source2;            * %INCLUDE and run preprocessed code;
%include "/home/ted.conway/test_pgm_out.sas";
run;
```

INPUT FILE (test_pgm_in.sas)	OUTPUT FILE (test_pgm_out.sas)
data _null_; 🤔 a=1 ✅ b=1; ❎ b=0; 🏃	data _null_; if a=1 then b=1; else b=0; run;

SAS CODE (INPUT & OUTPUT) + SAS PREPROCESSOR CODE

So, we seem to be getting close to what we need for a simple prototype, but read on for why we'll need a Plan C!

PLAN C: A QUICK-AND-DIRTY PYTHON PREPROCESSOR

In the last example, did you notice the "Running Person" emoji (🏃) representing "**RUN;"** faced left? So, what would it take to make it face right like the SAS Run icon? Surprisingly, quite a bit – the Unicode Standard is very complex. The left-facing Running Person emoji is rather simple, requiring only one Unicode code point, which the SAS K functions support. But there are more complex multiple code point graphemes, such as the Man Running Facing Right emoji and additional emoji modifiers, such as skin tone, are also available.

	Component	Unicode Code Point	UTF-8 Bytes	Description
Man Running Facing Right 	🏃	U+1F3C3	F0 9F 8F 83	Person Running
	ZWJ	U+200D	E2 80 8D	Zero Width Joiner
	♂	U+2642	E2 99 82	Male Sign
	VS-16	U+FE0F	EF B8 8F	Variation Selector-16 (emoji style)
	ZWJ	U+200D	E2 80 8D	Zero Width Joiner
	➡	U+27A1	E2 9E A1	Right Arrow
	VS-16	U+FE0F	EF B8 8F	Variation Selector-16

EMOJI ENCODING – MAN RUNNING FACING RIGHT

So, what can we do if we want our prototype to also be able to use symbols from the extended set of emoji characters? Well, for our Plan C, let's tap into the Python world, specifically the regex package, which can be used to extract emojis from text. The Python emoji preprocessor is patterned after the SAS one, with Python dictionary and regex functionality being used (instead of SAS formats) to replace emojis with their associated text.

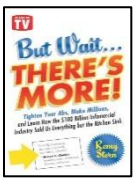
```
sascode='''
data _null_;                          * Test SAS code;
🤔 a=b ✅ c=d; ❎ e=f; 🏃
'''
dict = {'🤔': 'if', '✅': 'then', '❎': 'else', '🏃':'run;'}

import regex, json, pandas as pd   # Extract emojis from text;
def grapheme_positions(text):
    matches = regex.finditer(r'\X', text)
    return [{'grapheme': m.group()} for m in matches]
                                   # Replace emojis with text
df = pd.DataFrame.from_dict(grapheme_positions(sascode), orient='columns')
df['replstr'] = df['grapheme'].map(dict)
df['replstr'].fillna(df['grapheme'], inplace=True)
sascodeout="".join(df["replstr"].tolist());
print("INPUT SAS CODE (TEXT+SYMBOLS)\n",sascode,
      "\nOUTPUT SAS CODE (TEXT)\n",sascodeout)
sas_session.submitLST(sascodeout)   # Run SAS Code using SASPy
```

INPUT SAS CODE (sascode variable)	OUTPUT SAS CODE (sascodeout variable)
data _null_; 🤔 a=1 ✅ b=1; ❎ b=0; 🏃	data _null_; if a=1 then b=1; else b=0; run;

SAS CODE (INPUT & OUTPUT) + PYTHON PREPROCESSOR CODE

BUT WAIT, THERE'S MORE!

Examples of SAS and SQL code employing a wider variety of emojis than the above simple introductory examples can be found in the Appendix, including the use of emojis with SAS macros.



THERE'S MORE!

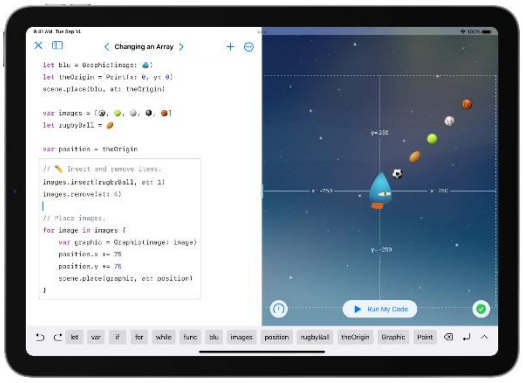
SASPy was used to run these examples to allow Python & SAS to be run on two different platforms. The Python preprocessor was run from Microsoft VS Code on a Windows laptop, and it submitted the stripped-of-emojis SAS code to run in the Cloud on SAS OnDemand for Analytics (*free for learning!*) . You'll have additional options – PROC FCMP, PROC PYTHON, X and SYSTEM commands – to integrate SAS and Python if both languages are available on the same server, container, or laptop.

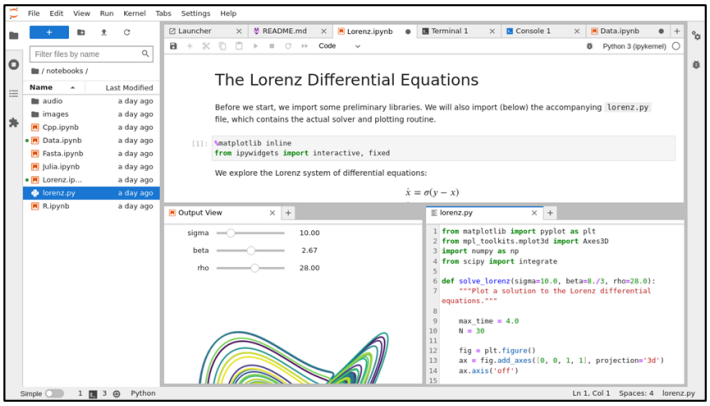
CONCLUSION

So, might people actually find the incorporation of pictograms – be they emojis, icons, other images, or icon-like fonts – into their programming language helpful? This is probably one of those "we'll never know until we try" things. After all, people didn't know they wanted icons and emojis until Steve Jobs, Susan Kare, and Apple brought them to the masses, who overwhelmingly decided they did!


SUSAN KARE ICONS

While even the straightforward insert-symbol problem has vexed people since the dawn of APL some 50+ years ago, one wonders how much of that may be for lack of trying. Apple's "built for touch" Swift Playgrounds, originally designed for children learning to code on iPads, offers a taste of what a text + graphics language might look like, as did Microsoft's Touch Develop.


APPLE SWIFT PLAYGROUNDS APP



JUPYTER NOTEBOOK

Bad graphics solutions can indeed be worse and create more problems than no graphics solution at all. Still, I have hopes that someone will have a *Eureka!* moment on this and figure out good solutions for graphics and text to co-exist in code. Perhaps programming languages are just waiting for their Susan Kare to come along and design a delightful set of icons that people simply can't resist. Will it be for everyone? Of course not, but neither are GUI's, emojis/icons, notebooks, and graphical presentations of programming workflows/pipelines, which many aren't fans of, but they still enjoy widespread adoption!

Eventually, I believe user-defined symbols/Unicode – and HTML/markdown, images, hyperlinks, sound, video, etc. – will be able to be interspersed directly in program code and comments (not just in process flows or markdown cells in notebooks!) in some widely-used programming languages and will prove to be extremely helpful to many. And layouts to present programs will be much more customizable, going way beyond even the most flexible of today's notebooks and IDEs. Think of a magazine or newspaper-style layout designed for a large monitor, with multiple windows into the code, pictograms representing certain code elements, sidebars for functions and subroutines, and other separate or embedded windows with documentation and videos to explain and demonstrate how things work. But that's a topic for a different paper! 😊

NEW YORK TIMES

RECOMMENDED READING

- ***How to work with emojis in SAS.*** Chris Hemedinger.
  https://blogs.sas.com/content/sasdummy/2024/07/11/emojis-sas/
- ***Have a Comprehensive understanding of SAS® K functions.*** Leo (Jiangtao) Liu, Elizabeth Bales.
  https://support.sas.com/resources/papers/proceedings18/1902-2018.pdf
- ***Bring Your SAS® and Python Worlds Together With SASPy!*** Ted Conway.
  https://www.lexjansen.com/mwsug/2024/BB/MWSUG-2024-BB-042.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Ted Conway
ted.j.conway@gmail.com
@vivasasvegas (Twitter/X)

APPENDIX – EMOJI USE WITH SAS & SQL (INPUT SAS CODE & PYTHON PREPROCESSOR)

PYTHON DICTIONARY

```
dict = {'❓': 'if', '✅': 'then', '❎': 'else', '😳':'if', '👍': 'then', '🤌': 'else',
        '💾':'data', '✂️':'substr', '🗓️':'date', '🕒':'time', '🏃':'run;', '🖨️':'print',
        '🧵':'create table', '🗑️':'drop table', '🛒':'select', '⬅️':'from',
        '🔗':'join', '↕️':'order by', '⬇️':'desc', '∑':'group by'}
```

SAS & SQL

```
sascode='''
💾 dt_tm(keep=dt tm);
❓ a=b ✅ c=d; ❎ e=f;
😳 g=h 👍 i=j; 🤌 k=✂️(L, 1, 3);
dt=🗓️(); tm=🕒();
format dt 🗓️10. tm 🕒8.; 🏃

proc 🖨️; 🏃

proc sql noprint;
🧵 car_summary as
🛒 make, model, max(dt) as max 🗓️ format=🗓️10., min(tm) as min 🕒 format=🕒8.
⬅️ sashelp.cars t1 🔗 dt_tm t2 on 1=1
∑ make, model ↕️ make, model ⬇️;

proc 🖨️ 💾=car_summary(obs=5); 🏃

proc sql; 🗑️ car_summary;

%macro 🗓️🕒; * Macro to print current date/time;
🗓️🕒=put(🗓️(),date10.)||" "||compress(put(🕒(),🕒8.));
file print; put 🗓️🕒;
%mend;

💾 _null_; * Get current time twice, sleep 5s between;
%🗓️🕒; call sleep(5,1); %🗓️🕒; 🏃
'''
```

PYTHON PREPROCESSOR

```
import regex, json, pandas as pd
def grapheme_positions(text):
   matches = regex.finditer(r'\X', text)
   return [{'grapheme': m.group()} for m in matches]

df = pd.DataFrame.from_dict(grapheme_positions(sascode), orient='columns')
df['replstr'] = df['grapheme'].map(dict)
df['replstr'].fillna(df['grapheme'], inplace=True)
sascodeout="".join(df["replstr"].tolist());
print("INPUT SAS CODE (TEXT+SYMBOLS)\n",sascode,
      "\nOUTPUT SAS CODE (TEXT)\n",sascodeout)
sas_session.submitLST(sascodeout)
```

SAS CODE (BEFORE PREPROCESSING) + PYTHON PREPROCESSOR CODE

APPENDIX – EMOJI USE WITH SAS & SQL (INPUT & OUTPUT SAS CODE COMPARISON)

INPUT – SAS CODE (TEXT WITH EMOJIS)	OUTPUT – SAS CODE (TEXT ONLY)

💾 dt_tm(keep=dt tm);
❓ a=b ✅ c=d; ❎ e=f;
😳 g=h 👍 i=j; 👎 k=🎀(L, 1, 3);
dt=📅(); tm=🕛();
format dt 📅10. tm 🕛8.; 🏃
	data dt_tm(keep=dt tm);
	if a=b then c=d; else e=f;
	if g=h then i=j; else k=substr(L, 1, 3);
	dt=date(); tm=time();
	format dt date10. tm time8.; run;

proc 🖨️; 🏃	proc print; run;
proc sql noprint; 🧵 car_summary as 🛒 make, model, max(dt) as max📅 format=📅10., min(tm) as min🕛 format=🕛8. ⬅️ sashelp.cars t1 🔗 dt_tm t2 on 1=1 ∑ make, model ↕️ make, model ⬇️;	proc sql noprint; create table car_summary as select make, model, max(dt) as maxdate format=date10., min(tm) as mintime format=time8. from sashelp.cars t1 join dt_tm t2 on 1=1 group by make, model order by make, model desc;
proc 🖨️ 💾=car_summary(obs=5); 🏃	proc print data=car_summary(obs=5); run;
proc sql; 🗑️ car_summary;	proc sql; drop table car_summary;
%macro 📅🕛; * Macro to print current date/time; 📅🕛=put(📅(),date10.)||" "||compress(put(🕛(),🕛8.)); file print; put 📅🕛; %mend;	%macro datetime; * Macro to print current date/time; datetime=put(date(),date10.)||" "||compress(put(time(),time8.)); file print; put datetime; %mend;
💾 _null_; * Get current time twice, sleep 5s between; %📅🕛; call sleep(5,1); %📅🕛; 🏃	data _null_; * Get current time twice, sleep 5s between; %datetime; call sleep(5,1); %datetime; run;

SAS CODE – BEFORE AND AFTER PREPROCESSING