Sometimes newer is better, sometimes newer is just newer.

How to pursue new technologies rationally!

Presenter: Hui Liu
GPS A&R Standards, MSD
**New** is a relative term. It doesn’t always bring excitement.
New COVID Variant arrives
New SDTM 3.3 Standard is now required while our macros are SDTM 3.2 compliant
Enrico Fermi

In 1938, Nobel Prize in Physics for his "demonstrations of the existence of new radioactive elements produced by neutron irradiation, and ...".

It was later found out that element is not a new one.
Are there any *new/newish* concepts or technology which can be borrowed for solving problem in Pharmaceutical industry? The following slides will show some examples.
Scenario 1:

Each Pharma/CRO might have similar macros for analysis & reporting but they are designed in different taste.

```sas
%procMeans(
  dataset=ADSL,
  lib=ADAM,
  var=Age,
  by=trt01,
  layout=vertical);
```

```sas
%stat0means0report(
  ds=analysis.ADLS,
  v=Age,
  group=,
  landport=L
  );
```
Problems

- New employees need to re-learn similar things with different details for macro name/parameter options.

- Each company is also concerned about the productivity loss due to employees’ learning curve.
A market-neutral strategy from hedge funds aims to generate returns that are independent of the market’s (mainstream stocks and bonds’ value) swings.
Solution #1: A syntax-neutral strategy is a form of programming strategy that aims to develop code which is independent of macro-library (normally depending on who your employer is).
Your code will be mainly using original SAS syntax which is the common knowledge of developers at almost any level.

A virtual machine interface to SAS can potentially translate your original SAS code and generate various output widely accepted by Pharmaceutical industry. Meanwhile it generates the calling program for specific macro utilities.
How does an virtual machine look like to a user?
%let statloop=0;
%let _win=c:hui;
/*\"\v\vware-host\Shared Folders\Downloads\_win;*/
libname win "&_win";

%macro vm(str);
%local proc var by data;
%ahgincr(statloop);

%ahgpm(str);
%let str=%sysfunc(prxchange(s/%str(\"%\")\%/,-1,%bquote(&str)));%
%ahgpm(str);

%local i one;
%do i=1 %to 20;
/**AHGcount(%bquote(&str),dlm=%str(\;));*/
%let one=%bquote(%scan(%bquote(&str),i,%str(\;)));
%if not %AHGblank(&one) %then
  %do;
    %if %sysfunc(prxmatch(m:\s*proc\s/i, &one)) %then %let proc=%sysfunc(prxchange(s:\s*proc\s+(\w+)).*/1,
    %if %sysfunc(prxmatch(m:\s*proc\s/i, &one)) %then %let data=%sysfunc(prxchange(s:\.*data=(\[\w\].)+).*/\:
    %if %sysfunc(prxmatch(m:\s*by\s/i, &one)) %then %let by=%sysfunc(prxchange(s:\s*by\s+(.*\w+)).*/1/i,1,
    %if %sysfunc(prxmatch(m:\s*var\s/i, &one)) %then %let var=%sysfunc(prxchange(s:\s*var\s+(.*\w+)).*/1/:
    %end;
cat(ntmi_out);
sink();

vm<-function(command){
  ### real code
  print(command)
  realCode(command)
  |
  vm('proc means data=sashelp.class;
  var height;
  by sex age;
  run;
  ')
}

R Script
It penetrates the language barrier. A SAS programmer can generate R code by coding in SAS syntax on common tasks. vice versa.

The hedge fund concept brings some ideas for use within statistical programming.
Pharmaceutical company works with functional service providers closely and it is common the macro library is shared from sponsor FSPs. Are there any concerns?

- It is not rare seeing validation programmers use original code as double programming code.
- Even though we trust our partners, an individual unethical misconduct can cause a leak. Or, our partners’ network being hacked is a risk we cannot ignore.
Non-preferred Solution #2: There are some non-code sharing solutions like compiled macro catalogs but with limitations.

- It is platform dependent. Unix and windows versions are not exchangeable.
- It is a black-box without any flexibility to debug. Not even inserting a %put statement or mprint option.
- A specific library is not usually as waterproof as SAS PROC. Remaining the minimal debugging option is necessary.
Preferred Solution #2: obfuscating your code
Obfuscation is the act of creating source or machine code that is difficult for humans to understand. Programmers may deliberately obfuscate code to conceal its purpose or its, primarily, in order to deter reverse engineering.[–wikipedia]

There are a plenty of tools for many languages like C, C++, Obj–C Java, Kotlin, Perl, Python, Swift. But you rarely see tools for SAS
Articles for SAS code obfuscating

- There are some technical tricks to make SAS code strange. A famous SAS guru Arthur L. Carpenter has even series papers for this topic.

Most of these tricks in these articles are fun but they harm the code. This means you do not have an original well-formed code. It makes maintenance difficult in the long run.

**INCONSISTENCY**

Subtle inconsistencies are very difficult to detect and can be very useful, especially in a program that has what seem to be clearly defined parameters.

- YES/NO variables should take on the values of YES=0 and NO=1 (of course never Y & N), except somewhere for some variable that has YES=1 and NO=0.

  A useful variation has `ANSWER='N'` when the response is YES and `ANSWER='Y'` when the response is NO.

- Variable and data set names should not have unique definitions throughout the program.

- Variables should on occasion disappear and reappear later with different definitions.
Our highlights of obfuscating

- The original macros remain well formatted without including dirty code.
- The obfuscated version’s functionalities are reserved without losing any usability.
- It can be done in batch for even a whole library.
- The obfuscated code is workable but lacks readability and maintainability. Likelihood of code being leaked is minimal.
Code before and after obfuscating

* TODO:
* Conditionally use submit(“statement;”) for systask, proc s:
* use it from the command bar. If we do this we should also:
* option on the %macro.

```sas
%local sasRoot sasExe configArg workDir;
%local sasArgs objServerArgs;
%local oNotes oQuoteLenMax host isSDSonWin rc;

%let host=%substr(%upcase(&syscsp), 1, 2); /* Get platform */
%let workDir = %sysfunc(getoption(work)); /* work dir, temp dir */
```

```sas
%macro getSasExe;
%local tmpPath tmpExe tmpSasExe sasHost path indx done wtkPathList;

%let embedQQ = %str(“%”);

%if (&isSDSonWin=1) %then /* sdssas wx6, w32 ? (old DNT
%do;
   %let tmpPath=%sysfunc(getoption(PATH));
   /* strip (, ) and ” */
   %let delims=%str(%) %str(%) %str(”);
```

```sas
%macro getSasExe;
%local Goat Rooster Horse Bear Jaguar Fox Hippopotamus

%let Crocodile= %str(”%”);

%if (&Buffalo=1) %then /* sdssas wx6, w32 ? (old DNT
%do;
   %let Goat=%sysfunc(getoption(PATH));
   %let Tiger=%str(%) %str(%) %str(”);
```

```sas
%let Cow= %str(”%”);

%if (&Camel=1) %then /* sdssas wx6, w32 ? (old DNT
%do;
   %let Cow=%sysfunc(getoption(PATH));
   %let Sheep=%str(%) %str(%) %str(”);
```

```sas
%let Chicken= %str(”%”);

%if (&Turkey=1) %then /* sdssas wx6, w32 ? (old DNT
%do;
   %let Chicken=%sysfunc(getoption(PATH));
   %let Turkey=%str(%) %str(%) %str(”);
```

```sas
%let Fish= %str(”%”);

%if (&Duck=1) %then /* sdssas wx6, w32 ? (old DNT
%do;
   %let Fish=%sysfunc(getoption(PATH));
   %let Fish=%str(%) %str(%) %str(”);
```
It can be done by SAS code but Python code is just more concise with dictionary object.

```python
def replacement(match):
    return r'\1'.lower()
line = re.sub(r'\\(\|
        ', replacement, line, flags=re.IGNORECASE)
line = re.sub(r'\ahgpm\((\|:)+\)', '', line, flags=re.IGNORECASE)

def replacement(match):
    return match.group(1).lower()
line = re.sub(r'\(|\|\)', replacement, line, flags=re.IGNORECASE)

if re.search(r'\&\&', line):
    for key in newLineDict.keys():
        line = re.sub(key+r'\b', str(newLineDict[key])+rdm, line)

for key in resDict.keys():
    line = re.sub(key+r'\b', str(resDict[key])+rdm, line)
line = re.sub(r'\&\&\|', r'', line)
#sys.stdout.write(line)
outFile.write(line)

def x():
    for line in lineList:
        pass
        sys.stdout.write(line)
        # for file in *.sas
        # do
        # obsas $file
        # done
        # cat *.sas >ahg.txt
```
def replacement(match):
    return r'\*\*\*+match.group(1).lower()'
line=re.sub(r'\*\*\*+\s*:\(\w+\)',replacement,line,flags=re.IGNORECASE)

def replacement(match):
    return r'\*\*\*+match.group(1).lower()'
line=re.sub(r'\*\*\*+\s*\(\w+\)',replacement,line,flags=re.IGNORECASE)

def replacement(match):
    return r'\*\*\*+match.group(1).lower()'
line=re.sub(r'\*\*\*+\s*\(\w+\)',replacement,line,flags=re.IGNORECASE)

def replacement(match):
    return r'\*\*\*+match.group(1).lower()'
line=re.sub(r'\*\*\*+\s*\(\w+\)',replacement,line,flags=re.IGNORECASE)

line=re.sub(r'\*\*\*\(\{\&\}\)+',',',line,flags=re.IGNORECASE)

def replacement(match):
    return match.group(1).lower()
line=re.sub(r'\&\&',replacement,line,flags=re.IGNORECASE)

if re.search(r'\&\&',line):
    for key in NewLineDict.keys():
        line=re.sub(key+r'\b',str(NewLineDict[key])+rdm,line)

for key in resDict.keys():
    Line=re.sub(key+r'\b',str(resDict[key])+rdm,line)
line=re.sub(r'\&\&',',',line)

#sys.stdout.write(line)
outFile.write(line)

def x():
    for line in lineList:
        pass
        sys.stdout.write(line)
        # for file in *.sas
        # do
        # obfsas $file
        # done
        # cat *.sas >ahg.txt
Another level of irretrievability if necessary. Efforts of reverse engineering is extremely high.

* option on the %macro.
***************************/

%local _mrk_merckm_759564 _mrk_erckme_759564 _mrk_rckmer_759564 _mrk_ckmerc_759564 ;
%local _mrk_kmerck_759564 _mrk__merck_759564 ;
%local _mrk_merckm_759564 _mrk__erckme_759564 _mrk__rckmer_759564 _mrk__ckmerc_759564 _mrk__kmerck_759564 ;

*/
%let _mrk__kmerck_759564=1;
%let _mrk____merck_759564=0;
%do %until (&mrk____merck_759564);
  %let _mrk__ckmerc_759564= %scan(&mrk____merck_759564, &mrk__kmerck_759564, &mrk____merck_759564);
  %let _mrk__kmerck_759564= %eval (&mrk__kmerck_759564 + 1);
  %if (%length(&mrk__ckmerc_759564) > 0) %then
    %let _mrk__ckmerc_759564= %sysfunc(strip(&mrk__ckmerc_759564));

  %if (%index(&mrk__ckmerc_759564, !) > 0) %then /* Has !SASROOT */
    %do;
      %let _mrk__ckmerc_759564= %substr(&mrk__ckmerc_759564, 9); /* strip !sasroot */
      %let _mrk__ckmerc_759564= &mrk_merckm_759564&mrk__ckmerc_759564; /* prefix the kn
All you need to is to change key/value pairs from animal names to randomized noisy strings.

```python
loop=0
newLineDict={}
resDict={}

for key in paraDict.keys():
    loop+=1;
    div=divmod(loop,6)
    lead='_'*div[0]
    allAnimal='Donkey Pig Bee Sheep Fish Camel Rabbit Chicken Cow Turtu
    rmus Tiger Crocodile Monkey Zebra Deer Lion Elephant Gorilla Panda Squa
    ran Duck Sparrow Eagle Rooster Owl Swan Hummingbird Parrot Peacock Bat

    []huige='__merckmerck__'
    ahg='__mrk__'+lead+ahuige[div[1]:div[1]+6]

    arrAni=allAnimal.split(' ')
    # ahg=arrAni[loop]
    newLineDict[key]=ahg
    resDict[key[2:]]=('&'+ahg
    # print(key,loop,ahg)
    print(newLineDict)

for line in lineList:
    def replacement(match):
        return r'\{}&{}\{}+match.group(1).lower()+'}"
Conclusion:

New/Newish technology is everywhere and ever existing.

All you need is to keep an open mind.

Coding in Pharmaceutical industry is not an easy thing but also with much fun. Hope you all enjoy it.