

SAS and Python for Code-blooded Programmers

Chris Hemedinger, SAS



Why Python is likely to pass Java in popularity

According to RedMonk programming languages analysis, Python has a way to ...
1 m



TNW

Could Python topple Java as the web's most popular coding language? Signs say it could happen.

TLDR: Python is on the rise — and with the training in The Absolute Python Programmin
2 weeks ago



JAXenter

Python is on its way to become the top programming language ...

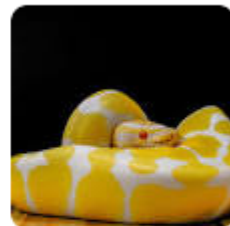
The TIOBE Index for December 2019 reaffirms the status quo. Java, C, Python,
Dec 9, 20



Dice Insights (blog)

Python Keeps Swallowing the Programming Universe: TIOBE

Python leapt from fourth to third place in TIOBE's monthly rankings of programming-language popularity, placing it just behind Java and C (and ...
Jun 11, 2019





Developers appreciate programming
languages for what the
languages can do for **them**.

It's not about the syntax.
It's about the **framework**.

Agenda

SAS 9.4 and Python: Ways of working

SASPy

Open source
code library to
connect Python
to SAS

PROC FCMP

Functional hook
to call Python
code from SAS
programs

Python <--> SAS

Run Python.
Run SAS.
Hand off using
data exchange.



SASPy

Python access to your SAS environment

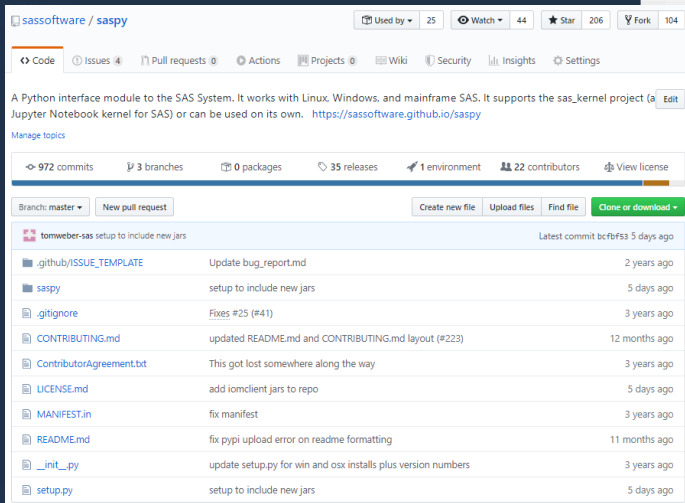
SASPy

Python code to access SAS

- Python library on GitHub:
github.com/sassoftware/saspy
- **Requires** a SAS installation – local or remote
- Simple to install with pip:

```
pip install saspy
```

- Think of it as just another SAS client (similar to SAS Enterprise Guide or SAS Add-In for Microsoft Office)
- Getting started: <https://sassoftware.github.io/saspy/>



What can you do?

Access SAS data

Run SAS code

Use Python functions
for certain PROCs

Exchange data with
pandas DataFrames

Use in SAS Grid
environment

Learn SAS!

SASPy features

```
In [1]: import saspy

In [2]: sas = saspy.SASsession()
Using SAS Config named: default
SAS Connection established. Workspace UniqueIdentifier is 9152FF02-B744-4369-9407-309C51AB3DC8

In [3]: cars=sas.sasdata('cars', 'sashelp')

In [4]: cars.head()
Out[4]:
```

	Make	Model	Type	Origin	...	MPG_Highway	Weight	Wheelbase	Length
0	Acura	MDX	SUV	Asia	...	23.0	4451.0	106.0	189.0
1	Acura	RSX Type S 2dr	Sedan	Asia	...	31.0	2778.0	101.0	172.0
2	Acura	TSX 4dr	Sedan	Asia	...	29.0	3230.0	105.0	183.0
3	Acura	TL 4dr	Sedan	Asia	...	28.0	3575.0	108.0	186.0
4	Acura	3.5 RL 4dr	Sedan	Asia	...	24.0	3880.0	115.0	197.0

```
[5 rows x 15 columns]

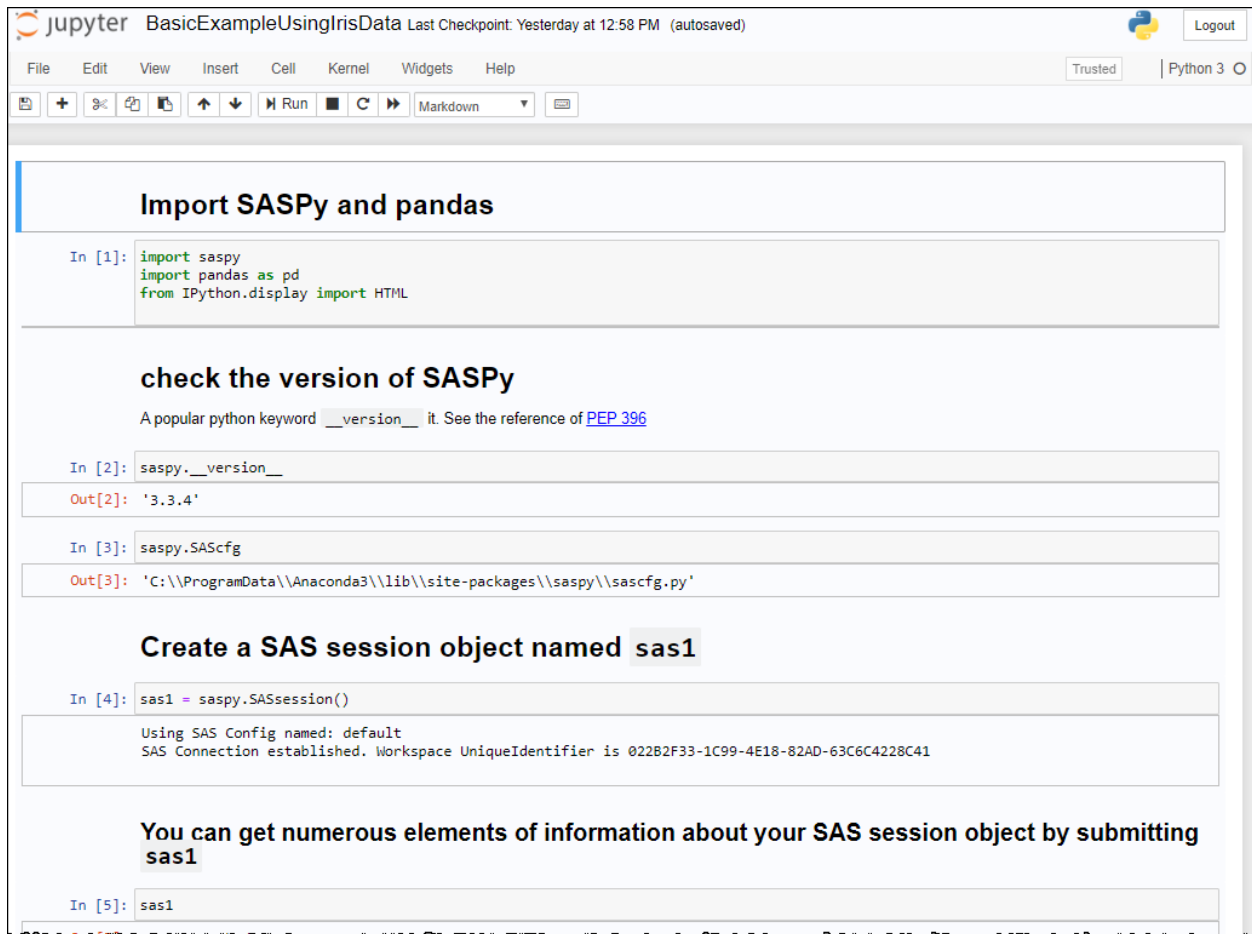
In [6]: c = cars.means()

In [7]: c
Out[7]:
```

	Variable	Label	N	...	P50	P75	Max
0	MSRP		428.0	...	27635.0	39215.0	192465.0
1	Invoice		428.0	...	25294.5	35732.5	173560.0
2	EngineSize	Engine Size (L)	428.0	...	3.0	3.9	8.3
3	Cylinders		426.0	...	6.0	6.0	12.0
4	Horsepower		428.0	...	210.0	255.0	500.0
5	MPG_City	MPG (City)	428.0	...	19.0	21.5	60.0
6	MPG_Highway	MPG (Highway)	428.0	...	26.0	29.0	66.0
7	Weight	Weight (LBS)	428.0	...	3474.5	3978.5	7190.0
8	Wheelbase	Wheelbase (IN)	428.0	...	107.0	112.0	144.0
9	Length	Length (IN)	428.0	...	187.0	194.0	238.0

```
[10 rows x 12 columns]
```

DEMO - SASPy



The screenshot shows a Jupyter Notebook titled "BasicExampleUsingIrisData" with a last checkpoint from yesterday at 12:58 PM. The interface includes a top menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running cells, and markdown. The notebook content is organized into sections with blue headers.

Import SASPy and pandas

```
In [1]: import saspy
import pandas as pd
from IPython.display import HTML
```

check the version of SASPy

A popular python keyword `__version__` it. See the reference of [PEP 396](#)

```
In [2]: saspy.__version__
```

```
Out[2]: '3.3.4'
```

```
In [3]: saspy.SAScfg
```

```
Out[3]: 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\saspy\\sascfg.py'
```

Create a SAS session object named `sas1`

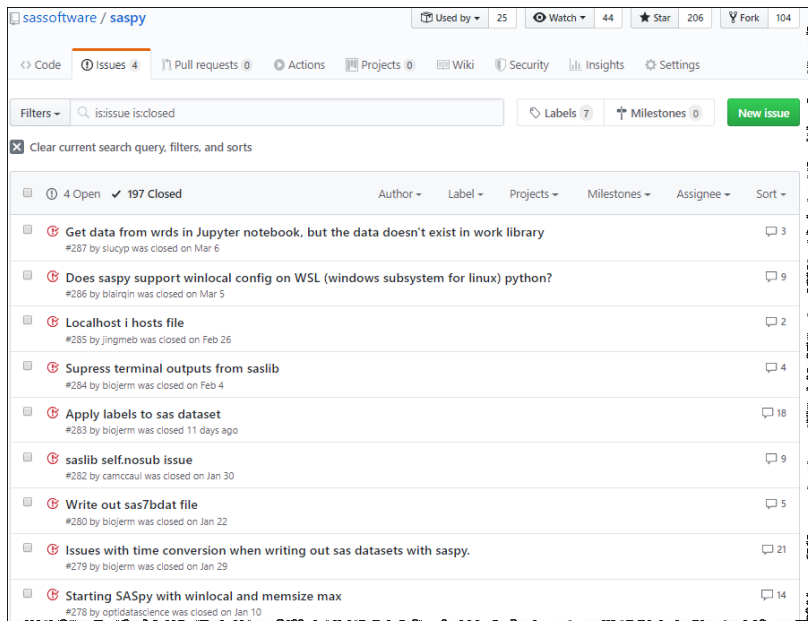
```
In [4]: sas1 = saspy.SASsession()
```

Using SAS Config named: default
SAS Connection established. Workspace UniqueIdentifier is 022B2F33-1C99-4E18-82AD-63C6C4228C41

You can get numerous elements of information about your SAS session object by submitting `sas1`

```
In [5]: sas1
```


Open Source and actively maintained



- SASPy dev team responds quickly to issues
- New ideas and examples are welcome
- **Contribute!**
Bug fixes, enhancements, new extensions for SAS PROCs, etc.
- **Try it!**
Built into SAS University Edition, launch Jupyter Notebook

The History and Evolution of SASPy, Including an Overview of What It Can Do and How to Use It





PROC FCMP and Python

Call Python functions from your SAS programs

PROC FCMP

This SAS procedure extends the SAS language by allowing user-written functions.

SAS 9.4M6 allows Python code in these functions.

Add Python code as functions in SAS

- Embed and import Python functions into SAS programs
- Python code runs in the Python interpreter
- Introduced with SAS Micro Analytic Service to score Python models...but now part of Base SAS.

Some special notes for setup

Environment variables to point SAS to Python and the SAS Micro Analytic Service

PROC FCMP and Python

Workflow example

```
proc fcmp outlib=work.fcmp.pyfuncs;
declare object py(python);
submit into py;
def PyProduct(var1, var2):
    "Output: MyKey"
    newvar = var1 * var2
    return newvar,
endsubmit;
rc = py.publish();
rc = py.call("PyProduct", 5, 10);
MyResult = py.results["MyKey"];
put MyResult=;
run;
```

- PROC FCMP wrapper
- Declare the Python object
- Inline Python code with **submit into**
- **Or**, use INFILE method to import a Python script
- Publish the code to Python
- (Optional) Return Python output

DEMO – Python object in SAS

```
Start Page | F:\scmsamples.sas x
```

Run ■ Cancel | [Icons] | < Share - | [Icon] Debug | [Icon] Local ▾

```
Code  
1 /* Creating a PROC FCMP function that calls a Python function */  
2 proc fcmp outlib=work.myfuncs.pyfuncs;  
3  
4 /* Create the outer FCMP function */  
5 /* These arguments are passed to the inner Python function */  
6 function FCMP_blackscholescall(stockprice, strikeprice, timeremaining, vol)  
7  
8 /* Create the inner Python function call */  
9 /* Declare Python object */  
10 declare object py(python);  
11  
12 /* Use the INFILE method to import Python code from a file */  
13 rc = py.infile("C:\Projects\fcmp\blackscholes.py");  
14  
15 /* Publish the code to the Python interpreter */  
16 rc = py.publish();  
17  
18 /* Call the Python function from SAS */  
19 /* Since this is the inner function, instead of values in the call  
20 /* you will pass the outer FCMP function arguments to the Python f  
21 rc = py.call("black_scholes_call", stockprice, strikeprice, timere  
22  
23 /* Store the inner function Python output in a SAS variable  
24 FCMP_out = py.results["optprice"];  
25  
26 /* Return the Python output as the output for outer FCMP function  
27 return(FCMP_out);  
28  
29 /* End the FCMP function  
30 endsub;  
31 run;  
32  
33 /* Specify the function library you want to call from  
34 options cmplib=work.myfuncs;  
35  
36 /*Use the DATA step to call your FCMP function and examine the res  
37 data blackscholes;  
38 stockprice = 132.58;  
39 strikeprice = 137;  
40 timeremaining = 0.041095;  
41 volatility = .2882;  
42 rate = .0222;  
43 result = FCMP_blackscholescall(stockprice, strikeprice, timeremaining,  
44
```

```

1  def internal_black_scholes_call(stockPrice, strikePrice, timeRemaining, volatility, rate):
2      import numpy
3      from scipy import stats
4      import math
5      if ((strikePrice != 0) and (volatility != 0)):
6          d1 = (math.log(stockPrice/strikePrice) + (rate + (volatility**2)\
7                  / 2) * timeRemaining) / (volatility*math.sqrt(timeRemaining))
8          d2 = d1 - (volatility * math.sqrt(timeRemaining))
9          callPrice = (stockPrice * stats.norm.cdf(d1)) - \
10                     (strikePrice * math.exp((-rate) * timeRemaining) * stats.norm.cdf(d2))
11      else:
12          callPrice=0
13      return (callPrice)
14
15  def black_scholes_call(stockPrice, strikePrice, timeRemaining, volatility, rate):
16      "Output: optprice"
17      import numpy
18      from scipy import stats
19      import math
20      optPrice = internal_black_scholes_call(stockPrice, strikePrice,\
21                                             timeRemaining, volatility, rate)
22      callPrice = float(optPrice)
23      return (callPrice,)

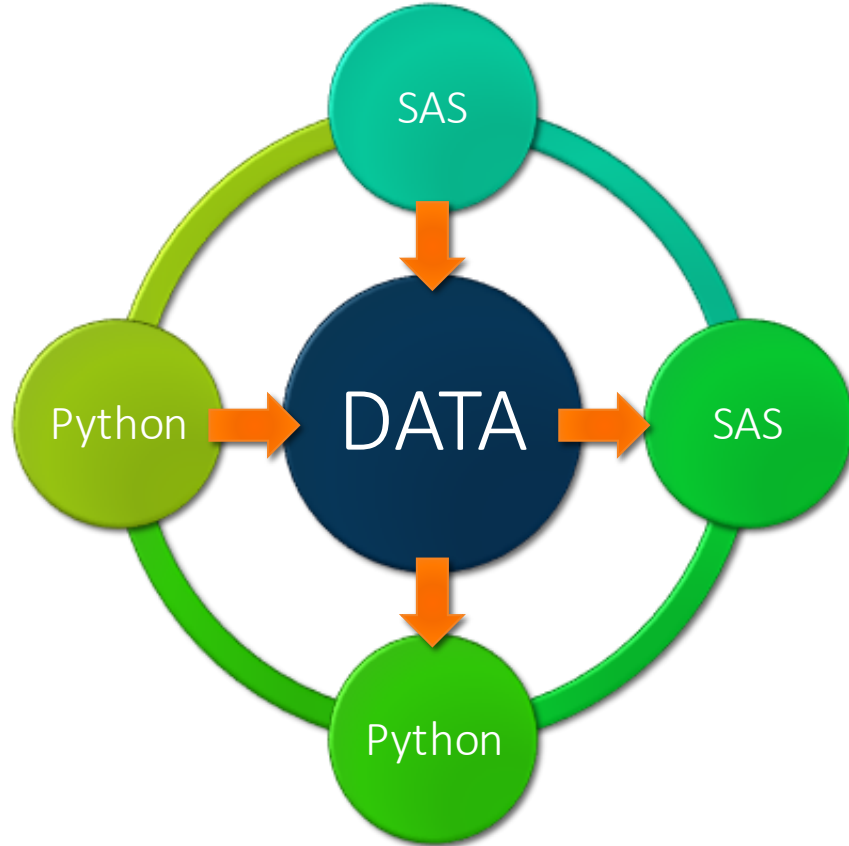
```



Python ↔ SAS

Use data and services to link processes together

This might seem obvious, but...



- **Data** is the primary currency in data science
- **Python** and **SAS** can easily hand off to one another, using **data** as the interchange

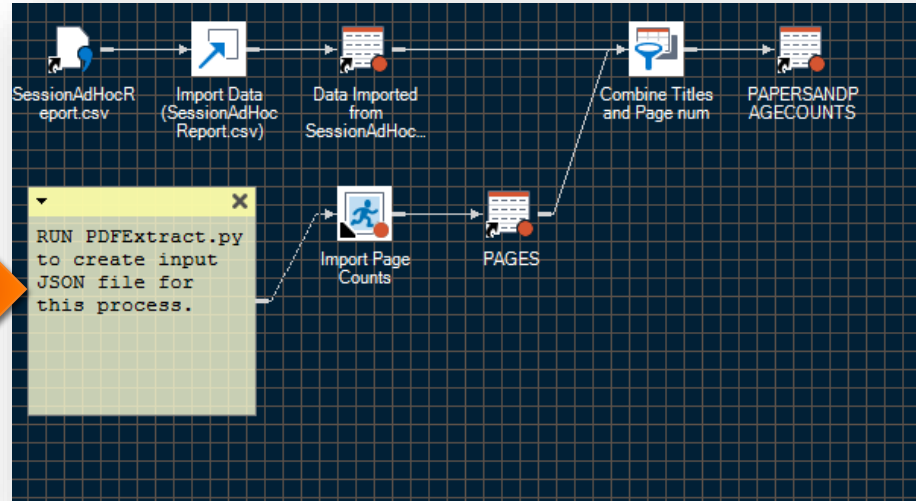
DEMO – Python-to-SAS handoff

```
from PyPDF2 import PdfFileReader
import os
import json

def get_info(path):
    with open(path, 'rb') as f:
        pdf = PdfFileReader(f)
        # info = pdf.getDocumentInfo()
        info = pdf.getNumPages()

    return info

if __name__ == '__main__':
    allfiles = {}
    directory = 'C:\\Projects\\SASGF2020_Proceedings'
    for filename in os.listdir(directory):
        if filename.endswith(".pdf"):
            info = get_info(os.path.join(directory,filename))
            allfiles[filename] = info
            continue
        else:
            continue
    paper_json = json.dumps(allfiles, indent=4)
    outfile = open("c:\\projects\\fcmp\\pagecounts.json", 'w')
    rc = outfile.write(paper_json)
    outfile.close()
```



Learn more

SAS 9.4 and Python: Ways of working

SASPy

Open source code library to connect Python to SAS

PROC FCMP

Functional hook to call Python code from SAS programs

Python <--> SAS

Run Python.
Run SAS.
Hand off using data exchange.

- [Introducing SASPy: Use Python code to access SAS](#)
- [SAS or Python? Why not use both? Using Python functions inside SAS programs](#)
- [Developers forum on communities.sas.com](#)
- [Python integration on developer.sas.com](#)



Thank you

sas.com