

A portrait of Hermione Granger from the Harry Potter series, with long blonde hair, wearing a dark blue robe, standing against a dark background.

Python with Hermione: Unleash your Inner Coding Witch and Dragon

Charu Shankar & Jim Box

PharmaSUG 2025 OS-323



Author(s) Bio

Charu Shankar, Jim Box SAS® Institute,



With a background in computer systems management. SAS Senior Technical Training Consultant Charu Shankar engages with logic, visuals, and analogies to spark critical thinking since 2007. Charu curates and delivers unique content on SAS, SQL, Viya, etc. to support users in the adoption of SAS software. When not coding, Charu teaches yoga and loves to explore Canadian trails with her husky Miko.

Jim Box is a Principal Data Scientist at the SAS Institute, where he has been supporting customers implementation of machine learning for the past seven years. Prior to that he spent 18 years in Clinical Research Organizations primarily as a statistician and programming director. He has Masters Degrees in Statistics and in Analytics.



Agenda

1. Introduction

What Is Python

Python application in analytics industry

Why should you learn Python to work with SAS

2. Basic Python

Python vs SAS

Python Programming Language Basics

Functions

Data structures

3. Python Data Manipulation

Popular Python Packages in SAS: SASPy, NumPy, Pandas

PROC Python

1 Introduction

What Is Python

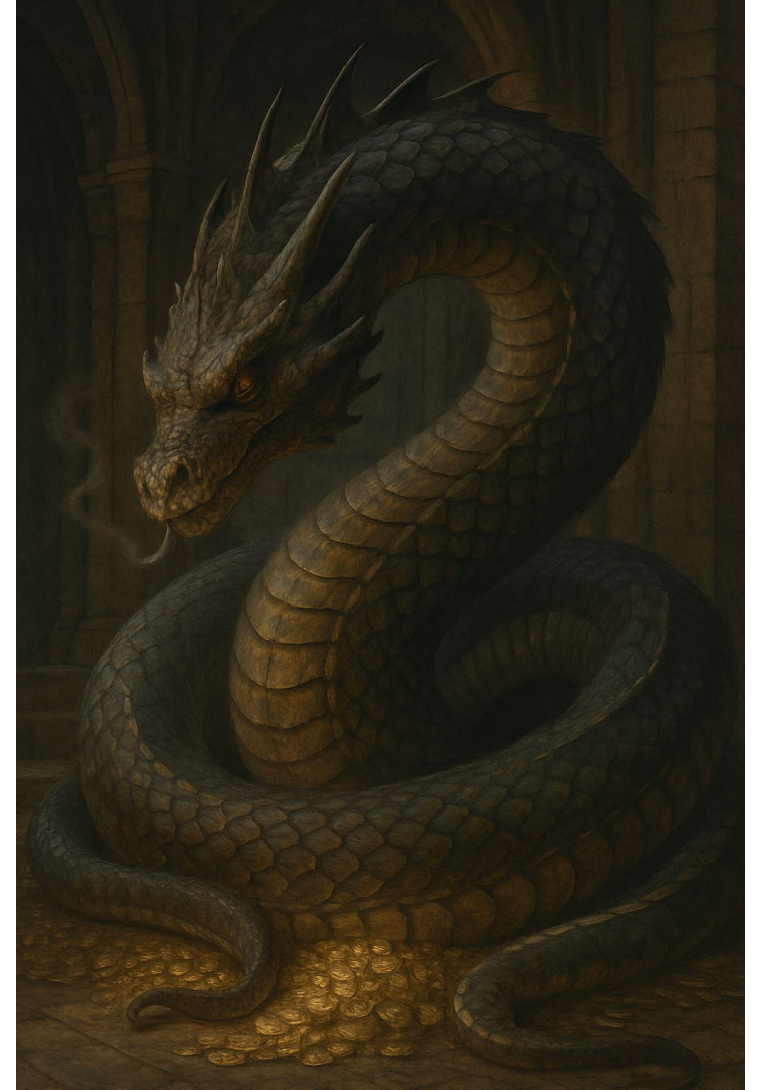
Python application in analytics industry

Why should you learn Python to work with SAS

What Is Python?

Python is a powerful, yet simple language, much like Hermione's precision in casting spells. It's versatile, easy to read, and is used for everything from data analysis to web development.

With Python, you can quickly solve problems and automate tasks, much like casting a well-planned charm to get results in the wizarding world.




Python application in analytics industry

- Python supports object-oriented programming, structured programming, and functional programming patterns.
- Python is generally considered ‘the second best language for everything’. As a jack of all trades, python is not well-suited to statistical analysis, but organizations see advantage in standardizing & using it. standard libraries exceed 72,000 & growing daily
- Python’s inherent readability and simplicity with the vast number of libraries available helps data scientists find packages tuned to their needs.

Why Should You Learn Python to Work with SAS?



Learning Python alongside SAS is like mastering both a wand and a broomstick—each has its strengths. Python's libraries, like pandas and NumPy, extend SAS's capabilities, giving you more control over your data, while allowing you to automate tasks and perform complex analyses.

Together, they make for a powerful combination, much like Hermione and her quick thinking in the face of a dragon. 

2 Basic Python

Python vs SAS

Python Programming Language Basics

Functions

Data structures

Python vs. SAS basics

	Python	SAS Data step
Case Sensitivity	Python respects casing of variable names or functions. E.g, Python will view City and city as two completely different variables, despite them having the same spelling because the casing of each letter is inconsistent.	SAS Data step will treat both City and city as the same variable. Variable names are case insensitive
Order of first occurrence	Python generally processes the code from top to bottom, so if it encounters a variable not yet defined, it would throw a NameError.	SAS also processes code from top to bottom. During execution if it encounters a variable not yet defined, it assigns a missing value to that variable
Slicing	Slicing in Python allows you to extract a subset of a sequence (like a list, tuple, or DataFrame) using indices and range notation	Slicing in SAS typically involves using PROC SQL or data step filtering techniques to select specific rows and columns from a dataset based on conditions or positions.
Storage	Data Frame	Data Set/Table
End of program statement	No symbols are used to end a program statement. The end-of-line character is used to end a Python statement	semicolon

Python Basics – Comments: A Spell for Clarity

Documentation is an important step in programming. Comments tell a computer to ignore that part of the program during execution and helps users provide content on why something is written the way it is.

A comment in Python starts with the # (pound symbol/hash sign/hash tag) and a space:

```
# don't run this bit - it's a comment  
print('Today is Tuesday')
```

```
Today is Tuesday
```



The shortcut for adding (or removing) comment lines in Python editors is the same as in SAS Studio: CTRL + /

Python Programming Basics – Indenting

Indentation in Python is used to identify lines of code executed together. The block of code that is part of **for** loop has to be indented so that Python knows what to execute for each iteration of the loop.

The following code will help convert Fahrenheit temperatures to Celsius for visiting Canadians.

```
[52]: #2.1 indenting
      chictemp = [59, 60, 61, 62, 63]
      for i in chictemp:
          celsius = (i - 32) * 5/9
          print('Temp for a canadian is:', celsius)
```

```
Temp for a canadian is: 15.0
Temp for a canadian is: 15.555555555555555
Temp for a canadian is: 16.111111111111111
Temp for a canadian is: 16.666666666666668
Temp for a canadian is: 17.222222222222222
```

Indenting Matters: A Dragon's Breath of Code

The following 2 code snippets are identical except in the 1st snippet, The line after the **for** block is **not** indented. This results in a error being raised in the interpreter.

```
[51]: #2.1 indenting
chictemp = [59, 60, 61, 62, 63]
for i in chictemp:
celsius = (i - 32) * 5/9
print('Temp for a canadian is:', celsius)
```

```
File "<ipython-input-51-72d5f868c653>", line 4
```

```
celsius = (i - 32) * 5/9
```

```
^
```

```
IndentationError: expected an indented block
```

Python Basics – ending program statement

No symbols are used to end a program statement. The end-of-line character is used to end a Python statement. This also helps to enforce legibility by keeping each statement on a separate physical line.

Coincidentally, like SAS, Python will also accept a semi-colon as an end of statement terminator. This is rarely seen because multiple statements on the same physical line is considered to be unacceptable.

Inconsistent Casing: The Dragon's Roar of Chaos

```
#2.1 python basics
city = "Chicago"
print(City)
# This will output a NameError because
# the casing is not consistent.
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-27-6205f02d6b52> in <module>
      1 #2.1 python basics
      2 city = "Chicago"
----> 3 print(City)
      4 # This will output a NameError because
      5 # the casing is not consistent.
```

```
NameError: name 'City' is not defined
```


Order of First Occurrence: Tracking the Dragon's Footsteps

```
[29]: #2.1 order of first occurrence  
print(year)  
year=2018
```

```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-29-c62089a3fe2d> in <module>  
      1 #2.1 order of first occurrence  
----> 2 print(year)  
      3 year=2018  
  
NameError: name 'year' is not defined
```

```
[ ]: year
```

Order of first occurrence - SAS

```
42      /*2.1 order of first occurrence*/  
43      data _null_;  
44          x=year;  
45          year=2018;  
46          put x=;  
47      run;  
  
x=.
```

Functions: The Magic of Reusable Spells

- Python has several built-in functions.
- Function names are generally typed in lowercase.
- Defining your own functions is an important part of programming in Python.

Some of Python's built-in functions include the following:



Function	Description	Run this	Get this
<code>print()</code>	Prints to the screen	<code>print('Hello')</code>	Hello
<code>type()</code>	Returns the type of an object	<code>a = 'Test'</code> <code>print(type(a))</code>	<class 'str'>
<code>sorted()</code>	Returns a sorted list ¹	<code>a = [5,1,0,3,2,4]</code> <code>print(sorted(a))</code>	[0, 1, 2, 3, 4, 5]

Python Data Structures: Organizing Your Magical Artifacts

Data structures are containers for data and other objects.

They form building blocks and aid the creation of Dataframes and other useful objects.

DataFrames are built on top of Python and Numpy modules.

Python data structures can be broken down into:

- Basic
- Advanced

Python Basics – Variables

- A Variable is just a way to store data, objects, lists etc.
- Variable names can contain letters, numbers, and underscores¹, and must begin with a letter or an underscore.
- Case matters: the variables 'a' and 'A' are not the same:
- Python has some reserved words including these (again, case matters):

False	class	is	in	return
True	and	if	not	while
None	as	or	else	import

```
a=12
A='Twelve'
print(a,A)

12 Twelve
```

```
υψος = 10
```

```
print(υψος)
```

```
10
```

¹Unicode characters are also allowed. Unicode is an encoding standard used to represent letters from various languages and scripts. For example, the Greek word for height, υψος, is allowable as a variable name in Python:

Python data structures – Basic data types

The Building Blocks of Your Magical Code"

There are several basic data types in Python, including the following¹:

Data Type
Str
Int
Float
Bool

¹Other data types include list, tuple, dict.

Python Data Structures – String: Crafting Charms with Words

A String can be defined as the sequence of characters within quotation marks. In python, single, double, or triple quotes can be used to define a string. Consider the string examples below:

```
[23]: #2.2 The operator + is used to concatenate two strings as the operation returns "hello world"  
      "hello"+" world"
```

```
[23]: 'hello world'
```

```
[25]: #2.2 The operator * is known as the repetition operator as the operation returns "Hello Hello"  
      "Hello " *2
```

```
[25]: 'Hello Hello '
```

Python Data Structures - string in SAS

Here is how we would concatenate a string in SAS

```
data _null_;  
  x='hello';  
  y=' world';  
  z= catt(x,y);  
  put z;  
run;
```

Python Data Structures - String Slicing & Indexing: A Sharp Spell for Extracting Words

Python provides indexing methods for a number of objects, including lists. SAS has a similar construct `_n_` for the Data Step. Both indexing methods act as keys providing access to an individual item (or set of items). In Python's case, the default start index position is 0, and for SAS it is 1.

The slice operator `[n:m]` returns the part of the string from the *n*'th character to the *m*'th character, including the first but excluding the last. In other words, start with the character at index *n* and go up to but do not include the character at index *m*.

If you omit the first index (before the colon), the slice starts at the beginning of the string. If you omit the second index, the slice goes to the end of the string.

Python data structures - string indexing

Indexing of python strings starts from 0. For example, The string "HELLO" is indexed as shown below.

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

```
[53]: #2.2 string indexing
      str='Hello'
      print (str[0]);
      print (str[1]);
      print (str[2]);
      print (str[3]);
      print (str[4]);
```

H
e
l
l
o

Python data structures – string slicing

The slice operator `[]` is used to access individual characters of the string. However, the `:` (colon) operator can be used in python to access the substring as shown below.

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

```
[54]: #2.2 string slicing
print (str[:]);
print (str[0:]);
print (str[:5]);
print (str[:3]);
print (str[0:2]);
print (str[1:4]);

Hello
Hello
Hello
Hel
He
ell
```

Python Data Structures – List: A Dragon's Hoard of Data



A list is an ordered set of objects in Python.

In Python, items in a list are separated by commas and enclosed in square brackets. Lists are mutable, meaning you can add, remove, or alter items without changing their identity. Lists can be recognized by their square brackets [and] that hold elements, separated by a comma. Lists are built into Python, you do not need to invoke them separately.

Lists are similar to arrays in C. However; Lists can contain data of different types.

Python Data Structures – printing index positions

```
[58]: #printing index position 1  
      print(c[1])
```

10

```
[59]: #printing index position 0  
      print(c[0])
```

fall

Python Data Structures - list slicing cheat sheet

Slice `[:]` operators can be used to access data in a list. The concatenation operator `(+)` and repetition operator `(*)` works with the list in the same way that they work with strings

`a[start:stop]` # items start through stop-1

`a[start:]` # items start through the rest of the array

`a[:stop]` # items from the beginning through stop-1

`a[:]` # a copy of the whole array

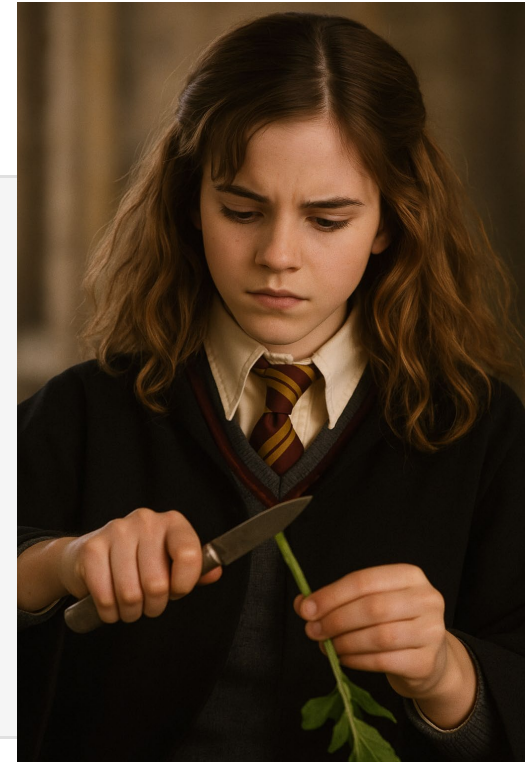
List Slicing: A Sharp Spell to Cut Through Data

Just like Hermione uses her keen intellect to slice through complex spells, Python's list slicing allows you to cut through sequences of data with precision. Whether you're choosing a specific range or a particular element, it's a powerful tool to get exactly what you need—no dragons to face here!

```
[61]: l = [1, "hi", "class", 2]
```

```
print (l[3:]);  
print (l[0:2]);  
print (l);  
print (l + l);  
print (l * 3);
```

```
[2]  
[1, 'hi']  
[1, 'hi', 'class', 2]  
[1, 'hi', 'class', 2, 1, 'hi', 'class', 2]  
[1, 'hi', 'class', 2, 1, 'hi', 'class', 2, 1, 'hi', 'class', 2]
```



Python List Comparison with Data Step: A Dragon's Test of Strength

[63]: *#Python code comparison with data step*

```
numbers = [2, 4, 6, 8, 10]
product = 1
for i in numbers:
    product = product * i
print('The product is:', product)
```

The product is: 3840

Python List Comparison with Data Step: A Dragon's Test of Strength

```
42      /*data step comparison with python*/
43
44      data _null_;
45          retain product 1;
46          do i = 2 to 10 by 2;
47              product=product*i;
48          end;
49      put 'The product is: ' product;
50      run;
```

The product is: 3840

Python Basic Data Types

Boolean: The Dragon's Choice Between True and False"

Boolean operations are valuable in data cleansing and transformations.

The standard two Boolean values are True and False with the capitalization as shown. In numeric contexts, they behave like the integers with values 0 and 1, respectively.

SAS does not have a boolean data type, so Data Step code is often constructed as a series of cascading IF-THEN/DO code blocks for testing True/False.

3 PROC Python Example

Importing Messy Lab Data

SAS Pandas comparison

pandas	SAS
DataFrame	data set
column	variable
row	observation
groupby	BY-group
NaN	.

NaN is an acronym for Not a Number

Pandas



Python has long been great for data wrangling and preparation, but less so for data analysis and modeling.

Pandas helps fill this gap, enabling you to carry out your entire data analysis workflow in Python without having to switch to a more domain specific language like R.

Pandas Data structures

Pandas data structure	Definition
Series	A 1-dimensional labelled array capable of holding any data type
DataFrame	A 2-dimensional labeled data structure with columns of potentially different types

Panda components vs SAS

A DataFrame looks a great deal like a SAS data set (or relational table). The table below compares panda components to those found in SAS.

Pandas	SAS
DataFrame	SAS data set
row	observation
column	variable
groupby	BY-Group
NaN	.
slice	sub-set
axis 0	observation
axis 1	column

Importing From LabExtract.xlsx

Columns Exactly as in Import File

The CONTENTS Procedure

Data Set Name	WORK.IMPORT	Observations	3
Member Type	DATA	Variables	20
Engine	V9	Indexes	0
Created	04/18/2025 11:16:54	Observation Length	160
Last Modified	04/18/2025 11:16:54	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64, LINUX_POWER_64		
Encoding	utf-8 Unicode (UTF-8)		

Variables in Creation Order					
#	Variable	Type	Len	Format	Label
1	ID	Num	8	BEST.	ID
2	Hematology_Measurement_Mean_Meas	Num	8	BEST.	Hematology Measurement;Mean;Measurement;Meas. Type: White Blood Cell;Units: E9/L
3	Hematology_Measurement_Mean_Mea1	Num	8	BEST.	Hematology Measurement;Mean;Measurement;Meas. Type: Neutrophil;Units: E9/L
4	Hematology_Measurement_Mean_Mea2	Num	8	BEST.	Hematology Measurement;Mean;Measurement;Meas. Type: Lymphocytes;Units: E9/L
5	Hematology_Measurement_Mean_Mea3	Num	8	BEST.	Hematology Measurement;Mean;Measurement;Meas. Type: Monocytes;Units: E9/L
6	Hematology_Measurement_Mean_Mea4	Num	8	BEST.	Hematology Measurement;Mean;Measurement;Meas. Type: Eosinophil;Units: E9/L
7	Hematology_Measurement_Mean_Mea5	Num	8	BEST.	Hematology Measurement;Mean;Measurement;Meas. Type: Basophil;Units: E9/L
8	Hematology_Measurement_Mean_Mea6	Num	8	BEST.	Hematology Measurement;Mean;Measurement;Meas. Type: Red Blood Cells;Units: E12/L

Summon the Python

```
proc python;
submit;

import pandas as pd

def update_column_names(input_file, output_file):
    # Read the Excel file
    df = pd.read_excel(input_file)

    # Update column names
    new_columns = []
    for column in df.columns:
        # Find the index of the first ":"
        index_of_colon = column.find(":")
        if index_of_colon != -1:
            # Find the index of the next ";"
            index_of_semicolon = column.find(";", index_of_colon)
            if index_of_semicolon != -1:
                new_column_name = column[index_of_colon + 1:index_of_semicolon].strip()
            else:
                new_column_name = column[index_of_colon + 1:].strip()
            new_columns.append(new_column_name)
        else:
            new_columns.append(column)

    # Update column names in the DataFrame
    df.columns = new_columns

    # Save the modified DataFrame to a new Excel file
    df.to_excel(output_file, index=False)

update_column_names('/nfsshare/sashls2/data/Labs/LabExtract.xlsx', '/nfsshare/sashls2/data/Labs/LabExtract_Clean.xlsx')

endsubmit;
run;
```

Magic Achieved

Columns after Python Code Execution

The CONTENTS Procedure

Data Set Name	WORK.IMPORT	Observations	3
Member Type	DATA	Variables	20
Engine	V9	Indexes	0
Created	04/18/2025 11:20:18	Observation Length	160
Last Modified	04/18/2025 11:20:18	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64, LINUX_POWER_64		
Encoding	utf-8 Unicode (UTF-8)		

Variables in Creation Order

#	Variable	Type	Len	Format	Label
1	ID	Num	8	BEST.	ID
2	White_Blood_Cell	Num	8	BEST.	White Blood Cell
3	Neutrophil	Num	8	BEST.	Neutrophil
4	Lymphocytes	Num	8	BEST.	Lymphocytes
5	Monocytes	Num	8	BEST.	Monocytes
6	Eosinophil	Num	8	BEST.	Eosinophil
7	Basophil	Num	8	BEST.	Basophil
8	Red_Blood_Cells	Num	8	BEST.	Red Blood Cells

Thanks! Visit us in Booth 301

Jim Box
Charu Shankar
SAS Institute

EMAIL	Charu.shankar@sas.com
BLOG	https://blogs.sas.com/content/author/charushankar/
TWITTER	CharuYogaCan
LINKEDIN	https://www.linkedin.com/in/charushankar/

