Everything is Better with Friends

Using SAS in Python Applications with SASPy and Open-Source Tooling (Getting Started)

A few notes before we get started...

1. Please enable line numbers using the Tools menu: **Tools** -> **Settings** -> **Editor** -> **Show line numbers** -> **Save**
2. To execute code examples, you'll need credentials for the following accounts:
    - Google. (If you're not already signed in, you should see a **Sign In** button in the upper right corner. You can also visit https://accounts.google.com/signup to create an account for free.)
    - SAS OnDemand for Academics. (You can create an account for free at https://welcome.oda.sas.com/ using an existing SAS Profile account. If you don't already have a SAS Profile account, you can create one for free by clicking on the link near the bottom of the ODA login page under the heading "Get Started".)
3. To save a copy of this notebook, along with any edits you make, please use the File menu: **File** -> **Save a copy in Drive**
4. We also recommend enabling the Table of Contents using the View menu: **View** -> **Table of contents**
5. Some useful Zoom Reactions:
    - 👍 (Thumbs Up) when you're done with a section
    - ✋ (Raise Hand) when you need tech support
    - ☕ (I'm Away) to let us know you've stepped away
6. Looking for "extra credit"? Please let us know if you spot any typos!

Section 0. Setup and Connect to SAS OnDemand for Academics (ODA)

Example 0.1. Install the SASPy package

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter.

```
1 !pip install saspy
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting saspy
  Downloading saspy-4.7.0.tar.gz (9.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.9/9.9 MB 31.8 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: saspy
  Building wheel for saspy (setup.py) ... done
  Created wheel for saspy: filename=saspy-4.7.0-py3-none-any.whl size=9938036 sha256=9d8241567970355a2fb097cb
  Stored in directory: /root/.cache/pip/wheels/c1/ab/9f/ffe82d792f6a8314e07be05a27f8fa0b8459e0110a682cf8c6
Successfully built saspy
Installing collected packages: saspy
Successfully installed saspy-4.7.0
```

Line-by-Line Code Explanation:

- Line 1: Install the Python package `saspy` inside the current Google Colab session.

Notes about Example 0.1:

1. Google Colab is based on JupyterLab, which is a popular open source platform for programming in Python and other languages.
2. The exclamation mark is used in JupyterLab to pass a command to the underlying operating system, which is Ubuntu Linux for Google Colab sessions.
3. `pip` is the standard command line tool for installing Python packages. (Fun fact: The name "pip" is a recursive acronym meaning "pip installs packages.")

Step 2. Connect to SAS OnDemand for Academics (ODA) and start a SAS session

Instructions:

1. Determine the Region for your ODA account by logging into https://welcome.oda.sas.com/. You should see a value like `Asia Pacific 1`, `Asia Pacific 2`, `Europe 1`, `United States 1`, or `United States 2` at the top of screen near your username. (For more information about Regions and using Python in Jupyter Notebooks, please see the ODA documentation at https://support.sas.com/ondemand/caq_new.html#region and https://support.sas.com/ondemand/saspy.html.)
2. If your ODA account is associated with a Region other than `United States 2`, comment out Line 38 by adding a number sign (`#`) at the beginning of the line, and then uncomment the list of servers corresponding to your Region.

   Note: As of the time of creation of this Notebook, only the Regions listed below were available. If your SAS ODA account is associated with a Region that's not listed, you will need to manually add the appropriate servers.
3. Click anywhere in the code cell, and run the cell using Shift-Enter.
4. At the prompt `Please enter the OMR user id`, enter either your SAS ODA user ID or the email address associated with your ODA account.
5. At the prompt `Please enter the password for OMR user`, enter the password for your SAS ODA account.

```
 1 # import standard library packages
 2 import io
 3 import pathlib
 4 import zipfile
 5
 6 # import third-party libraries
 7 import requests
 8 import saspy
 9
10 # because of recent changes to SAS ODA, we may need to install some files in our Colab session
11 zip_file_url = 'https://drive.google.com/uc?id=1vQ6oVgky8UcLAvhct7CL8Oc5I9Mctiw5&export=download'
12 expected_zip_file_contents = {'sas.rutil.jar', 'sas.rutil.nls.jar', 'sastpj.rutil.jar'}
13 jar_file_installation_path = '/usr/local/lib/python3.8/dist-packages/saspy/java/iomclient/'
14
```

```
# check the JAVA config files currently available in the SASPy installation of our Colab session
current_saspy_jar_files = {
    file.name
    for file
    in pathlib.Path(jar_file_installation_path).glob('*.jar')
}

# if any of three specific .jar files aren't found, download and install them in our Colab session
if not expected_zip_file_contents.issubset(current_saspy_jar_files):
  zip_file_url_response = requests.get(zip_file_url)
  zip_file_contents = zipfile.ZipFile(io.BytesIO(zip_file_url_response.content))
  zip_file_contents.extractall('/usr/local/lib/python3.8/dist-packages/saspy/java/iomclient/')

# with the preliminaries out of the way, we can now establish a connection from Colab to SAS ODA
sas = saspy.SASsession(
    java='/usr/bin/java',
    iomport=8591,
    encoding='utf-8',

    # For Region "United States 1", uncomment the line below.
    #iomhost = ['odaws01-usw2.oda.sas.com','odaws02-usw2.oda.sas.com','odaws03-usw2.oda.sas.com','odaws04-usw2

    # For Region "United States 2", uncomment the line below.
    iomhost = ['odaws01-usw2-2.oda.sas.com','odaws02-usw2-2.oda.sas.com'],

    # For Region "Europe 1", uncomment the line below.
    #iomhost = ['odaws01-euw1.oda.sas.com','odaws02-euw1.oda.sas.com'],

    # For Region "Asia Pacific 1", uncomment the line below.
    #iomhost = ['odaws01-apse1.oda.sas.com','odaws02-apse1.oda.sas.com'],

    # For Region "Asia Pacific 2", uncomment the line below.
    #iomhost = ['odaws01-apse1-2.oda.sas.com','odaws02-apse1-2.oda.sas.com'],

)
print(sas)
```

```
Using SAS Config named: default
Please enter the OMR user id: isaiah.p.lankham@kpchr.org
Please enter the password for OMR user : ··········
SAS Connection established. Subprocess id is 549

Access Method         = IOM
SAS Config name       = default
SAS Config file       = /usr/local/lib/python3.8/dist-packages/saspy/sascfg.py
WORK Path             = /saswork/SAS_work5D04000135DF_odaws01-usw2-2.oda.sas.com/SAS_work94D3000135DF_odaws01
SAS Version           = 9.04.01M7P08062020
SASPy Version         = 4.7.0
Teach me SAS          = False
Batch                 = False
Results               = Pandas
SAS Session Encoding  = utf-8
Python Encoding value = utf-8
SAS process Pid value = 79327
```

Line-by-Line Code Explanation:

- Lines 1-2: Load the standard library modules `io` and `zipfile`.
- Lines 4-5: Load the third-party modules `requests` (pre-installed in Colab) and `saspy` (installed into your Google Colab session in Example 0.1 above).
- Lines 7-10: Download a .zip archive to the Google Colab environment and extract its contents into the SASPy installation, which is necessary for SASPy to connect to SAS ODA. (See the Notes below for an explanation.)
- Lines 12-32: Connect to SAS ODA and establish a SAS session. A Python object named `sas` is created, which will be used in most subsequent examples.
- Line 33: The `print` function is used to display attributes of the `sas` object.

Notes about Example 0.2:

1. If your SAS session times out or terminates (e.g., by closing this notebook or using the `sas.endsas()` command), you'll need to run this cell again and re-enter your ODA login credentials.

2. In this notebook, we're using the "IOM using Java" method to connect to ODA. The SASPy documentation lists methods for several other scenarios, including a local Python installation connecting to SAS running either on the same machine or a remote server.
3. If an error is displayed, an incompatible kernel has been chosen. This Notebook was developed using the Python 3.8 kernel provided in Google Colab as of February 2023.
4. ODA was recently upgraded to SAS version 9.40M7, which is why SASPy needs the contents of a .zip file to be downloaded and installed inside Colab before SASPy can connect to SAS ODA. If Python and SAS were installed on the same machine, the contents of the .zip file would already be available as part of the SAS installation, per https://sassoftware.github.io/saspy/configuration.html

Example 0.3. Test the SAS connection

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter.

```
1 sas.submitLST("ods text='Hello, SAS ODA!';")
```

```
                              The SAS System

Hello, SAS ODA!
```

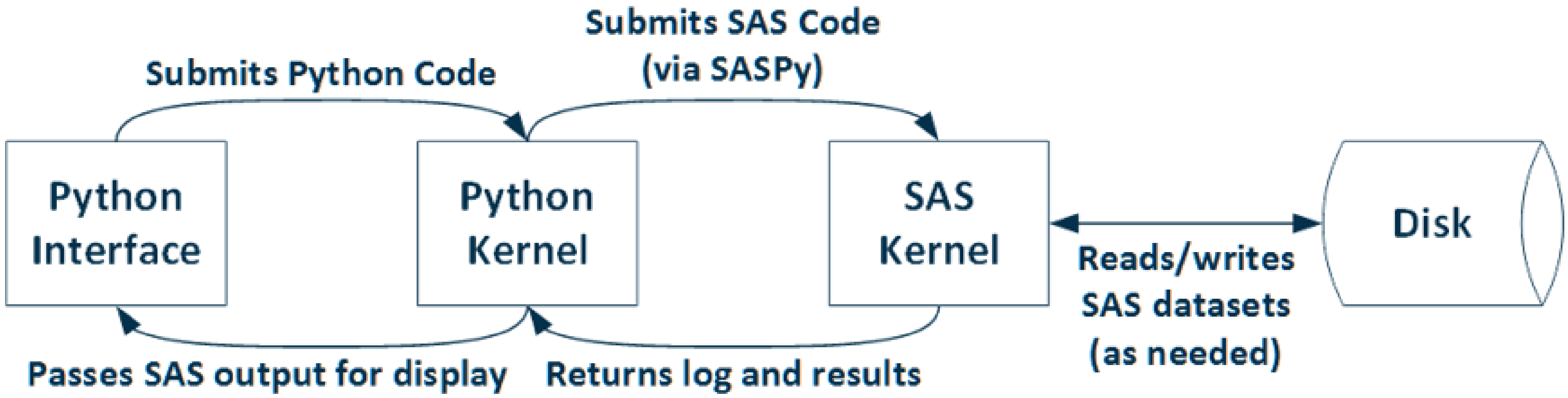
Line-by-Line Code Explanation:

- Line 1: Use the `submitLST` method to run some SAS code and display the results. There should be a short message displayed as SAS HTML output.

Notes about Example 0.3:

1. If everything runs successfully up to this point, it proves that SAS and Python are communicating!

2. This brief example demonstrates the basic purpose of `saspy`, which is to use Python to submit SAS code to a SAS Kernel and get SAS logs and output back. The following figure illustrates the basic architecture, with Google Colab providing the Python Interface/Kernel and SAS OnDemand for Academics providing both the SAS Kernel and the Disk to store SAS datasets:


Submits Python Code
Submits SAS Code (via SASPy)
Python Interface
Python Kernel
SAS Kernel
Disk
Reads/writes SAS datasets (as needed)
Passes SAS output for display
Returns log and results

Example 0.4. Install and import additional packages

```
# Install the rich module for colorful printing, limiting the version for compatibility with Colab
!pip install 'rich<13.3'

# We'll use IPython to display DataFrames or HTML content
from IPython.display import display, HTML

# We'll use the pandas package to create and manipulate DataFrame objects
from pandas import DataFrame

# We'll use the platform package to get information about our Python environment.
import platform

# We're overwriting the default print function with rich.print
from rich import print

```

```
# We're also setting the maximum line width of rich.print to be a bit wider (to avoid line wrapping)
from rich import get_console
console = get_console()
console.width = 165
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting rich<13.3
  Downloading rich-13.2.0-py3-none-any.whl (238 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 238.9/238.9 KB 8.1 MB/s eta 0:00:00
Collecting markdown-it-py<3.0.0,>=2.1.0
  Downloading markdown_it_py-2.2.0-py3-none-any.whl (84 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.5/84.5 KB 9.3 MB/s eta 0:00:00
Requirement already satisfied: typing-extensions<5.0,>=4.0.0 in /usr/local/lib/python3.8/dist-packages (from
Requirement already satisfied: pygments<3.0.0,>=2.6.0 in /usr/local/lib/python3.8/dist-packages (from rich<13
Collecting mdurl~=0.1
  Downloading mdurl-0.1.2-py3-none-any.whl (10.0 kB)
Installing collected packages: mdurl, markdown-it-py, rich
Successfully installed markdown-it-py-2.2.0 mdurl-0.1.2 rich-13.2.0
```

Section 1. Python Code Conventions and Data Structures

Example 1.1. Meet the Python environment

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
!cat /etc/*release*
print('\n')
print('Python Version:', platform.sys.version)
print('\n')
print(sorted(list(platform.sys.modules)))
```

```
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=20.04
DISTRIB_CODENAME=focal
DISTRIB_DESCRIPTION="Ubuntu 20.04.5 LTS"
NAME="Ubuntu"
VERSION="20.04.5 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.5 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
```

```
Python Version: 3.8.10 (default, Nov 14 2022, 12:59:47)
[GCC 9.4.0]
```

```
[
    'IPython',
    'IPython.core',
    'IPython.core.alias',
    'IPython.core.application',
    'IPython.core.async_helpers',
    'IPython.core.autocall',
    'IPython.core.builtin_trap',
    'IPython.core.compilerop',
    'IPython.core.completer',
    'IPython.core.completerlib',
    'IPython.core.crashhandler',
    'IPython.core.debugger',
    'IPython.core.display',
    'IPython.core.display_trap',
    'IPython.core.displayhook',
    'IPython.core.displaypub',
    'IPython.core.error',
    'IPython.core.events',
    'IPython.core.excolors',
    'IPython.core.extensions',
    'IPython.core.formatters',
    'IPython.core.getipython',
    'IPython.core.history',
    'IPython.core.hooks'
```

```
 'IPython.core.hooks',
 'IPython.core.inputsplitter'
```

Line-by-Line Code Explanation:

- Lines 1-2: Display information about the underlying operating system using a standard Linux sysadmin command line trick, followed by a blank line.
- Lines 3-4: Print information about the Python version, followed by a blank line.
- Line 5: Print a sorted list of all modules currently available to be loaded by the Python kernel.

```
 'IPython.core.magics.basic',
```

Exercise 1.1.1. True or False: Changing Line 3 to `PRINT(PLATFORM.SYS.VERSION)` would result in an execution error.

```
 'IPython.core.magics.display',
```

True. Because Python is case-sensitive, `PRINT(PLATFORM.SYS.VERSION)` would result in an error.

```
 'IPython.core.magics.logging',
```

Exercise 1.1.2. True or False: The example code should result in an execution error because there are no terminating semicolons.

```
 'IPython.core.magics.packaging',
```

False. Semicolons are not required to terminate a Python statement.

```
 'IPython.core.page',
```

Notes about Example 1.1:

1. This example illustrates four ways Python syntax differs from SAS:
    - Unlike SAS, capitalization matters in Python. Changing Line 3 to `PRINT(PLATFORM.SYS.VERSION)` would produce an error.
    - Unlike SAS, semicolons are optional in Python, and they are typically only used to separate multiple statements placed on the same line. E.g., Lines 3-5 could be combined into the single, super-long line `print(platform.sys.version); print('\n'); print(sorted(list(platform.sys.modules)))`
    - Unlike SAS, dot-notation has a consistent meaning in Python and can be used to reference objects nested inside each other at any depth. E.g., on Line 3, the `platform` module object invokes the sub-module object `sys` nested inside of it, and `sys` invokes the object `version` nested inside of it. (Think Russian nesting dolls or turduckens.)

2. To increase performance, only a small number of modules in Python's standard library are available to use directly by default, which is why the `platform` module was explicitly loaded in Example 0.4 above.
3. Python comes with a large standard library because of its "batteries included" philosophy, and numerous third-party modules are also actively developed and made freely available through sites like https://github.com/ and https://pypi.org/. For the examples in this notebook, we're using these five third-party modules:
    - `IPython`, which stands for "Interactive Python." JupyterLab builds upon `IPython`, so it's already available by default in Google Colab.
    - `pandas`, which provides `DataFrame` objects. DataFrames can be found in other languages, like R, and are similar to SAS datasets. Because `pandas` is a fundamental package for working with data in Python, it's already available by default in Google Colab.
    - `requests`, which is the standard go-to package for making HTTP requests and downloading files. Because `requests` has become so ubiquitous, it's already available by default in Google Colab.
    - `rich`, which has recently become the standard go-to package for creating beautiful text-based output. Because `rich` doesn't come pre-installed in Google Colab sessions, we had to manually install it in Section 0 above.
    - `saspy`, which is a Python package developed by the SAS Institute for connecting to a SAS kernel. Because `saspy` doesn't come pre-installed in Google Colab sessions, we had to manually install it in Section 0 above.

```
    'IPython.utils.ipstruct',
```

Example 1.2. Hello, data!

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
    'IPython.utils.sysinfo',
```

```
hello_world_str = 'Hello, Colab!'
print(hello_world_str)
print('\n')
if hello_world_str == 'Hello, Colab!':
    print(type(hello_world_str))
```

```
6 else:
```

```
Hello, Colab!

<class 'str'>
```

```
        'PIL._imaging',
        'PIL._util'
```

Line-by-Line Code Explanation:

- Lines 1-3: Create a string object (`str` for short) named `hello_world_str`, and print its value, followed by a blank line.
- Lines 4-7: Check to see if `hello_world_str` has the expected value. If so, print its type. Otherwise, print an error message.

```
        '_bisect',
```

Exercise 1.2.1. Which of the following changes to the above example would result in an error? (Select all that apply.)

a. Removing an equal sign (`=`) so that Line 4 becomes `if hello_world_str = 'Hello, Colab!'`
b. Removing Line 3 (`print('\n')`)
c. Unindenting Line 5 (`print(type(hello_world_str))`)

```
        '_contextvars',
        '_csv'
```

Changes a and c would result in errors.

```
        '_cython_0_29_21',
```

Exercise 1.2.2. Write several lines of Python code to produce the following output:

```
42


<class 'int'>
```

```
        '_heapq',
        '_imp'
```

```
hello_world_int = 42
print(hello_world_int)
print('\n')
print(type(hello_world_int))
```

```
42    _pydev_bundle',
     '_pydev_bundle._pydev_calltip_util'
```

Notes about Example 1.2:

1. This example illustrates four more ways Python differs from SAS:
   - Unlike SAS, variables are dynamically typed in Python. After Line 1 has been used to create `hello_world_str`, it can be assigned a new value later with a completely different type. E.g., we could change Line 3 to be `hello_world_str = 42` so that `type(hello_world_str)` becomes `<class 'int'>`.
   - Unlike SAS, single-equals (`=`) only ever means assignment, and double-equals (`==`) only ever tests for equality, in Python. E.g., changing Line 4 to `if hello_world_str = 'Hello, Colab!'` would produce an error.
   - Unlike SAS, indentation is significant and used to determine scope in Python. E.g., unindenting Line 5 would produce an error since the `if` statement would no longer have a body.
   - Unlike SAS, single and double quotes always have identical behavior in Python. E.g., `'Hello, Colab!'` is treated exactly the same as `"Hello, Colab!"`.

```
     '_pydevd_bundle.pydevd_additional_thread_info'
```

Example 1.3. Python lists and indexing

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
      _pydevd_bundle.pydevd_console',
     '_pydevd_bundle.pydevd_constants'
```

```
1 hello_world_list = ['Hello', 'list']
2 print(hello_world_list)
3 print('\n')
4 print(type(hello_world_list))
```

```
['Hello', 'list']

<class 'list'>
```

```
      _pydevd_bundle.pydevd_extension_utils',
     '_pydevd_bundle.pydevd_filtering'
```

Line-by-Line Code Explanation:

- Line 1: Create a list object named `hello_world_list`, which contains two strings.

- Lines 2-4: Print the contents of `hello_world_list`, followed by a blank line and its type.

```
'_pydevd_bundle.pydevd_net_command_factory_json',
```

Exercise 1.3.1. Would the Python statement `print(hello_world_list[1])` display the value `'Hello'` or `'list'`?

```
'_pydevd_bundle.pydevd_process_net_command_json'
```

The value `'list'` would be displayed.

```
'_pydevd_bundle.pydevd_save_locals',
```

Exercise 1.3.2. True or False: A Python list may only contain values of the same type.

```
'_pydevd_bundle.pydevd_timeout'
```

False. A list may contain values of different types.

```
'_pydevd_bundle.pydevd_traceproperty',
```

Notes about Example 1.3.

1. Values in lists are always kept in insertion order, meaning the order they appear in the list's definition, and they can be individually accessed using numerical indexes within bracket notation:
   - `hello_world_list[0]` returns `'Hello'`
   - `hello_world_list[1]` returns `'list'`.
2. The left-most element of a list is always at index `0`. Unlike SAS, customized indexing is only available for more sophisticated data structures in Python (e.g., a dictionary, as in Example 1.4 below).
3. Lists are the most fundamental Python data structure and are related to SAS data-step arrays. However, unlike a SAS data-step array, a Python list object may contain values with different types, such as `str` and `int`. (Processing the values of a list without checking their types, though, may cause errors if the list contains unexpected values.)

```
'_sysconfigdata__x86_64-linux-gnu'
```

Example 1.4. Python dictionaries

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
'argparse',
'array'
```

```
1 hello_world_dict = {
2     'salutation'      : ['Hello'        , 'dict'],
3     'valediction'     : ['Goodbye'      , 'list'],
4     'part of speech'  : ['interjection', 'noun'],
5 }
6 print(hello_world_dict)
7 print('\n')
8 print(type(hello_world_dict))
```

```
{'salutation': ['Hello', 'dict'], 'valediction': ['Goodbye', 'list'], 'part of speech': ['interjection', 'nou
```

```
    'asyncio.coroutines',
```

Line-by-Line Code Explanation:

- Lines 1-5 : Create a dictionary object (`dict` for short) named `hello_world_dict` , which contains three key-value pairs, where each key is a string and each value is a list of two strings.
- Lines 6-8: Print the contents of `hello_world_dict` , followed by a blank line and its type.

```
    'asyncio.runners',
```

Exercise 1.4.1. What would be displayed by executing the statement `print(hello_world_dict['salutation'])` ?

```
    'asyncio.staggered',
```

The value `['Hello', 'dict']` would be displayed.

```
    'asyncio.transports',
```

Exercise 1.4.2. Write a single line of Python code to print the initial element of the list associated with the key `valediction` .

```
    'atexit',
```

```
print(hello_world_dict['valediction'][0])
```

```
Goodbye
```

```
    'attr._make',
```

Notes about Example 1.4:

1. Dictionaries are another fundamental Python data structure, which map keys (appearing before the colons in Lines 2-4) to values (appearing after the colons in Lines 2-4). The value associated with each key can be accessed using bracket notation:

    - `hello_world_dict['salutation']` returns `['Hello', 'dict']`
    - `hello_world_dict['valediction']` returns `['Goodbye', 'list']`
    - `hello_world_dict['part of speech']` returns `['interjection', 'noun']`

2. Whenever indexable data structures are nested in Python, indexing methods can be combined. E.g., `hello_world_dict['salutation'][0] == ['Hello', 'dict'][0] == 'Hello'`.
3. Dictionaries are more generally called *associative arrays* or *maps* and are related to SAS formats and data-step hash tables.

```
        'certifi.core',
```

Example 1.5. Introduction to DataFrames

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
        'chardet.big5prober',
```

```
hello_world_df = DataFrame(
    {
        'salutation'     : ['Hello'      , 'DataFrame'],
        'valediction'    : ['Goodbye'    , 'dict'],
        'part of speech' : ['exclamation', 'noun'],
    }
)
display(hello_world_df)
print('\n')
print(hello_world_df.shape)
print('\n')
hello_world_df.info()
```

	salutation	valediction	part of speech
0	Hello	Goodbye	exclamation
1	DataFrame	dict	noun

```
(2, 3)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 3 columns):
```

Line-by-Line Code Explanation:

- Lines 1-7: Create a DataFrame object (`df` for short) named `hello_world_df` with dimensions 2x3 (2 rows by 3 columns), with each key-value pair in the dictionary in Lines 2-6 becoming a column that is labelled by its key. (Think of a DataFrame as a rectangular array of values, like a SAS dataset, with all values in a column having the same type.)
- Lines 8-12: Print the contents of `hello_world_df`, following by the number of rows and columns it has, and then some additional information about it.

```
    'configparser',
```

Exercise 1.5.1. Write a single line of Python code to display the column labelled by `salutation`.

```
    'copy',
```

```
display(hello_world_df['salutation'])
```

```
0        Hello
1    DataFrame
Name: salutation, dtype: object
```

```
    'dataclasses',
```

Exercise 1.5.2. Write a single line of Python code to print the final element of the column labeled by `valediction`.

```
    'dateutil._common',
    'dateutil._version'
```

```
print(hello_world_df['valediction'][1])
```

```
dict
```

```
    'dateutil.relativedelta',
```

Notes About Example 1.5:

1. The `DataFrame` object type is not built into Python, which is why we had to import its definition from the `pandas` module in Example 0.4 above.
2. The columns in a DataFrames can be indexed like the keys in a dictionary. E.g., `hello_world_df['salutation'][0] == ['Hello', 'dict'][0] == 'Hello'`
3. A DataFrame is a tabular data structure with rows and columns, similar to a SAS data set. However, while SAS datasets are typically accessed from disk and processed row-by-row, DataFrames are loaded into memory all at once. This means values in DataFrames can be randomly accessed, but it also means the size of DataFrames can't grow beyond available memory.
4. The dimensions of the DataFrame are determined as follows:
    - The keys `'salutation'`, `'valediction'`, and `'part of speech'` of the dictionary passed to the `DataFrame` constructor function become column labels.
    - Because each key maps to a list of length two, each column will be two elements tall (with an error occurring if the lists aren't all the same length).
5. The `DataFrame` constructor function can also accept many other object types, including another `DataFrame`. Please see [pandas documentation]() for more information.

```
        'email.base64mime',
```

Section 2. SASPy Data Round Trip

```
        'email.header',
```

Example 2.1. Load a SAS dataset into a pandas DataFrame

<u>**Instructions**</u>: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
        'encodings.ascii',
```

```
fish_df_smelt_only = sas.sasdata2dataframe(
    table='fish',
    libref='sashelp',
 4     dsopts={
 5         'where' : ' Species = "Smelt" ',
 6         'obs'    : 10,
 7     },
 8 )
 9 print(type(fish_df_smelt_only))
10 print('\n')
11 print(fish_df_smelt_only.shape)
12 print('\n')
13 display(fish_df_smelt_only.head())
```

```
<class 'pandas.core.frame.DataFrame'>

(10, 7)
```

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Smelt	6.7	9.3	9.8	10.8	1.7388	1.0476
1	Smelt	7.5	10.0	10.5	11.6	1.9720	1.1600
2	Smelt	7.0	10.1	10.6	11.6	1.7284	1.1484
3	Smelt	9.7	10.4	11.0	12.0	2.1960	1.3800
4	Smelt	9.8	10.7	11.2	12.4	2.0832	1.2772

```
'google.colab._interactive_table_helper',
```

Line-by-Line Code Explanation:

- Lines 1-8: Create a DataFrame object named `fish_df_smelt_only` with dimensions 10x7 (10 rows and 7 columns) from the SAS dataset sashelp.fish by subsetting to the first ten rows where the column `Species` has the value `Smelt`.
- Line 9: Print the type of the object `fish_df_smelt_only`.
- Line 11: Print the number of rows and columns in `fish_df_smelt_only`.
- Line 13: Print the first 5 rows of `fish_df_smelt_only`.

```
'google.colab.drive'
```

Exercise 2.1.1. By default, the `head` method returns the first ________ rows in a dataset.

```
'google.colab.output._is_builder',
```

By default, the `head` method returns the first <u>five</u> rows in a dataset.

```
'google.colab.output._util',
```

Exercise 2.1.2. True or False: The `head` method (without an argument) always returns the same number of rows in a dataset.

```
'google.colab.widgets',
```

False. If there are fewer than five rows in a dataset, the `head` method may not return as many rows as expected.

```
'google.colab.widgets._widget',
```

Exercise 2.1.3. Write several lines of Python code to create and display a DataFrame object from the SAS dataset sashelp.fish, but limit the rows using a different value for `species`. (Hint: If Lines 4-7 in the Example were commented out or removed, you'd be able to view a different part of the dataset. Alternatively, you could also try running Example 2.2 below to see the full list of species names.)

```
'html',
```

```
fish_df_bream_only = sas.sasdata2dataframe(
    table='fish',
    libref='sashelp',
    dsopts={
      'where' : ' Species = "Bream" ',
      'obs'   : 10,
    },
)
display(fish_df_bream_only.head())
```

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340

```
'ipykernel',
'ipykernel_version'
```

Notes About Example 2.1:

1. `sasdata2dataframe` is a method of a `SASsession` object, which allows the contents of a SAS dataset (meaning a physical file living on disk in SAS ODA) to be copied into memory in Colab as a DataFrame object.
2. The resulting DataFrame has rows labelled by non-negative integers: The first row is labelled as 0, the second as 1, and so on, just like elements in a list. However, as we'll see below, the row labels can also be given by an index column.
3. The `dsopts` argument for `sasdata2dataframe` allows dataset options to be passed to SAS, which subset the dataset before it's converted to a DataFrame. In the above example, we've only used the dataset options `where` and `obs`, but it's also possible to pass through additional options like `keep` and `drop`, which also accept lists of columns. Notice that each option is specified as a key-value pair inside a dictionary.
4. When used without an argument, the `head` method returns the first 5 rows of a DataFrame (or the entire DataFrame, if there are 5 or fewer rows). We can also control the number of rows; e.g., `fish_df_smelt_only.head(3)` returns the first 3 rows.
5. A parallel method called `tail` can also be used to return the last few rows in a DataFrame.
6. The `sas` object represents a connection to a SAS session, and was created in section 0 above.

```
'ipython_genutils.path',
```

Example 2.2. Manipulate a DataFrame

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
'jupyter_client._version',
```

```
fish_df     = sas.sasdata2dataframe(table='fish',libref='sashelp')
fish_df_g   = fish_df.groupby('Species')
fish_df_gs  = fish_df_g['Weight']
fish_df_gsa = fish_df_gs.agg(['count', 'std', 'mean', 'min', 'max'])
display(fish_df_gsa)
```

Species	count	std	mean	min	max
Bream	34	206.604585	626.000000	242.0	1000.0
Parkki	11	78.755086	154.818182	55.0	300.0
Perch	56	347.617717	382.239286	5.9	1100.0
Pike	17	494.140765	718.705882	200.0	1650.0
Roach	20	88.828916	152.050000	0.0	390.0

```
    'kiwisolver',
```

Line-by-Line Code Explanation:

- Line 1: Create a DataFrame object named `fish_df` with dimensions 159x7, comprising all 159 rows and 7 columns of the SAS dataset sashelp.fish.
- Line 2: Group the rows of `fish_df` by the values in column `Species`. (This can be thought of as follows: For each possible value of `Species`, create a DataFrame having just the corresponding rows.)
- Line 3: Subset to just the column `Weight` in each grouping.
- Line 4: Apply aggregation functions for counting number of records, standard deviation, mean, minimum, and maximum to the values of `Weight` that have been grouped by values of `Species`.
- Line 5: Print the results of the aggregations, which uses `Species` as a row index.

```
    'matplotlib._path',
```

Exercise 2.2.1. The DataFrame `groupby` method is like a ______________________ statement in a PROC MEANS step in SAS.

```
    'matplotlib.afm',
```

The DataFrame `groupby` method is like a CLASS statement in a PROC MEANS step in SAS.

```
    'matplotlib.axes._axes',
```

Exercise 2.2.2. After a `groupby` method has been used on a DataFrame, subsetting to a specific column is like a ______________________ statement in a PROC MEANS step in SAS.

```
    'matplotlib.backend_bases',
    'matplotlib.backend_tools'
```

After a `groupby` method has been used on a DataFrame, subsetting to a specific column is like a VAR statement in a PROC MEANS step in SAS.

Exercise 2.2.3. Write several lines of Python code to create a DataFrame object from the SAS dataset sashelp.class, and then imitate a PROC MEANS step to get the median of `Height` when grouped by `Sex`.

```
class_df     = sas.sasdata2dataframe(table='class',libref='sashelp')
class_df_g   = class_df.groupby('Sex')
class_df_gs  = class_df_g['Height']
class_df_gsa = class_df_gs.agg(['count', 'std', 'mean', 'min', 'max'])
display(class_df_gsa)
```

```
     count       std       mean   min   max
Sex
F        9  5.018328  60.588889  51.3  66.5
M       10  4.937937  63.910000  57.3  72.0
```

Notes about Example 2.2:

1. When the `sasdata2dataframe` method is used without an `dsopts` argument, the entire SAS dataset is copied into memory as a DataFrame.
2. In the output, notice that the left-most column `Species` is actually an index column, which is a byproduct of using the `groupby` method. In other words, we can think of the rows of `fish_df_gsa` as being labelled by values of `Species`. This is different from the output in Example 2.1, where the rows of `fish_df_smelt_only` were labelled by non-negative integers since `fish_df_smelt_only` doesn't have an index column.
3. The SAS equivalent of this example is as follows:

```
PROC MEANS DATA=sashelp.fish STD MEAN MIN MAX;
    CLASS species;
    VAR Weight;
RUN;
```

However, while PROC MEANS operates on SAS datasets row-by-row from disk, DataFrames are stored entirely in main memory. This allows any number of DataFrame operations to be combined for on-the-fly reshaping using "method chaining." In other words, `fish_df_gsa` could have instead been created with the following one-liner, which avoids the need for intermediate DataFrames (and thus executes much more quickly):

```
fish_df_gsa = fish_df.groupby('Species')['Weight'].agg(['count', 'std', 'mean', 'min', 'max'])
```

4. Example 2.3 below assumes `fish_df_gsa` exists.

5. The `sas` object represents a connection to a SAS session, and was created in section 0 above.

```
    'mpl_toolkits.mplot3d.proj3d',
    'petsc'
```

Example 2.3. Load a DataFrame into a SAS dataset

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
    'numexpr.utils',
```

```
sas.dataframe2sasdata(
    fish_df_gsa.reset_index(),
    table="fish_sds_gsa",
    libref="Work"
)
sas_submit_return_value = sas.submit(
    '''
        PROC PRINT DATA=fish_sds_gsa;
        RUN;
    '''
)
```

```
sas_submit_log = sas_submit_return_value['LOG']
print(sas_submit_log)
sas_submit_results = sas_submit_return_value['LST']
display(HTML(sas_submit_results))
```

```
62                                                   The SAS System                    Wednesday, Februa

450         ods listing close;ods html5 (id=saspy_internal) file=_tomods1 options(bitmap_mode='inline') device
450       ! ods graphics on / outputfmt=png;
451
452
453                 PROC PRINT DATA=fish_sds_gsa;
454                 RUN;
455
456
457
458         ods html5 (id=saspy_internal) close;ods listing;
459

63                                                   The SAS System                    Wednesday, Februa

460
```

The SAS System

Obs	Species	count	std	mean	min	max
1	Bream	34	206.605	626.000	242.0	1000.0
2	Parkki	11	78.755	154.818	55.0	300.0
3	Perch	56	347.618	382.239	5.9	1100.0
4	Pike	17	494.141	718.706	200.0	1650.0
5	Roach	20	88.829	152.050	0.0	390.0
6	Smelt	14	4.132	11.179	6.7	19.9
7	Whitefish	6	309.603	531.000	270.0	1000.0

```
    'numpy.lib.index_tricks',
```

Line-by-Line Code Explanation:

- Lines 1-5: Convert the pandas DataFrame `fish_df_gsa` (a memory-resident rectangular array of values created in Example 2.2 above) into a SAS dataset (a physical file on disk), and store the result in the Work library (a physical location on disk in the remote SAS OnDemand for Academics session). Only columns are transferred, not indexes, which is why the `reset_index` method is used to convert `Species` values to a column.
- Lines 6-11: Apply the SAS PRINT procedure to SAS dataset `fish_df_gsa`. (Note: In Lines 7-10, triple quote marks are used to create a single string object with embedded line breaks.)
- Lines 12-13: Extract the SAS log (a string containing plain text) from the dictionary returned by the `submit` method, and display it using the `print` function.
- Lines 14-15: Extract the SAS output (a string containing HTML) from the dictionary returned by the `submit` method, and render it using the `display` and `HTML` functions.

```
     'numpy.polynomial.laguerre',
```

Exercise 2.3.1. True or False: If `reset_index` were not used in example 2.3, information could be lost when the `dataframe2sasdata` method is used to transform a DataFrame into a SAS dataset.

```
     'numpy.random._bounded_integers',
```

True, `dataframe2sasdata` *only transfers regular DataFrame columns, not indexes.*

```
     'numpy.random._mt19937',
```

Exercise 2.3.2. True or False: The `submit` method of a `SASsession` object allows arbitrary SAS code to be submitted directly to the SAS kernel.

```
     'numpy.random.bit_generator',
```

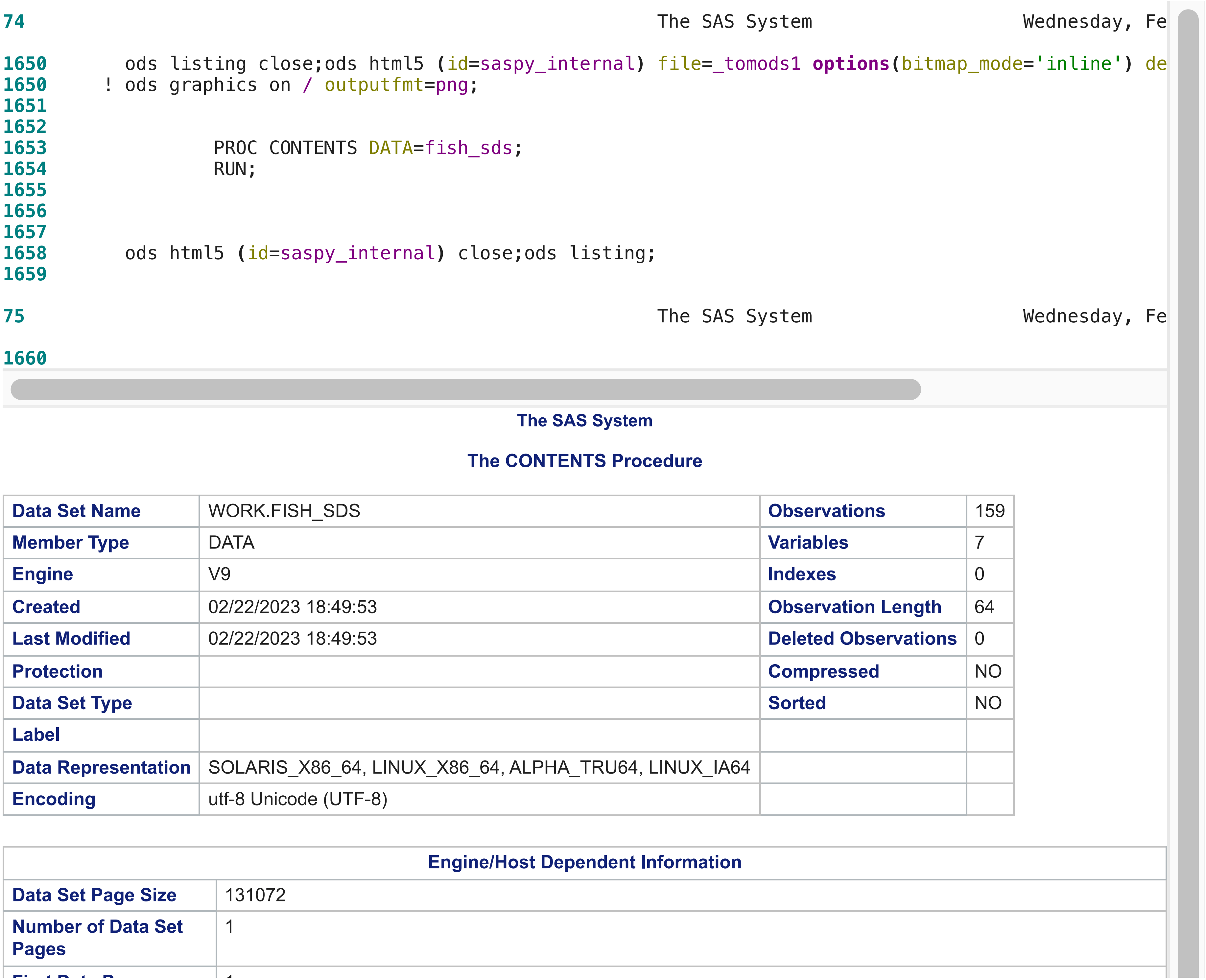
True. The `submit` *method submits the value of any string to the SAS kernel for execution.*

```
     'operator',
```

Exercise 2.3.3. Write several lines of Python code to convert the DataFrame `fish_df` created in Example 2.2 to a SAS dataset, and then use the SAS CONTENTS procedure directly on the result.

```
     'pandas._config.config',
```

```
sas.dataframe2sasdata(
    fish_df,
    table="fish_sds",
    libref="Work"
)
sas_submit_return_value = sas.submit(
    '''
        PROC CONTENTS DATA=fish_sds;
        RUN;
    '''
)
sas_submit_log = sas_submit_return_value['LOG']
print(sas_submit_log)
sas_submit_results = sas_submit_return_value['LST']
display(HTML(sas_submit_results))
```



```
74                                                        The SAS System                      Wednesday, Fe

1650         ods listing close;ods html5 (id=saspy_internal) file=_tomods1 options(bitmap_mode='inline') de
1650       ! ods graphics on / outputfmt=png;
1651
1652
1653                 PROC CONTENTS DATA=fish_sds;
1654                 RUN;
1655
1656
1657
1658         ods html5 (id=saspy_internal) close;ods listing;
1659

75                                                        The SAS System                      Wednesday, Fe

1660
```

The SAS System

The CONTENTS Procedure

Data Set Name	WORK.FISH_SDS	**Observations**	159
Member Type	DATA	**Variables**	7
Engine	V9	**Indexes**	0
Created	02/22/2023 18:49:53	**Observation Length**	64
Last Modified	02/22/2023 18:49:53	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Set Page Size	131072
Number of Data Set Pages	1
First Data Page	1

First Data Page	1
Max Obs per Page	2043

Notes about Example 2.3:

1. If `reset_index` is not used when exporting to SAS, the index column `Species` will be lost. Properties of a DataFrame without a SAS equivalences are not preserved when the method `dataframe2sasdata` is used to convert a pandas DataFrame to a SAS dataset.

2. Python strings can be defined with one of four quoting conventions:

   - single quotes, as in `'Hello, World!'`
   - double quotes, as in `"Hello, World!"`
   - triple quotes, as in `'''Hello, World!'''` or `"""Hello, World!"""`

   All four styles are interchangeable for single-line strings. However, unlike single- and double-quoted strings, triple-quoted strings can contain embedded line breaks.

3. The `submit` method can be used to pass arbitrary SAS code directly to a SAS kernel. After the SAS kernel executes the code, a dictionary is returned with the following two key-value pairs:

   - `sas_submit_return_value['LST']` is a string comprising the results of executing the PROC PRINT step. Because SAS returns HTML by default, the `HTML` function needs to be used to render the results, and the `display` function needs to be used to display the rendered HTML. (You could also use `print(sas_submit_return_value['LST'])` to view the raw, underlying HTML.)

   - `sas_submit_return_value['LOG']` is a string comprising the plain-text log resulting from executing the PROC PRINT step, which can be displayed with the `print` function.

4. Alternatively, adding the argument `results='TEXT'` to the `submit` method would replace the HTML output with plain-text viewable using the `print` function.

5. The `sas` object represents a connection to a SAS session, and was created in section 0 above.

```
    'pandas.core.groupby.groupby',
```

Section 3. Executing SAS Procedures with Convenience Methods

```
        'pandas.core.indexes'.
```

Example 3.1. Connect directly to a SAS dataset

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
        'pandas.core.indexes.frozen'
```

```
fish_sds = sas.sasdata(table='fish', libref='sashelp')
print(type(fish_sds))
print('\n')
display(fish_sds.columnInfo())
print('\n')
display(fish_sds.describe())
```

```
<class 'saspy.sasdata.SASdata'>

        Member  Num  Variable  Type  Len  Pos
```

Line-by-Line Code Explanation:

- Lines 1-2: Create a file pointer to the SAS dataset sashelp.fish (a physical file on disk) and print its type.
- Line 4: Use the `columnInfo` method to view metadata about the variables in sashelp.fish, and use the `display` function to render it as HTML output.
- Line 6: Use the `describe` method to view summary statistics for the numeric variables in the sashelp.fish, and use the `display` function to render it as HTML output.

Exercise 3.1.1. True or False: The `sasdata` method imports a SAS dataset and returns a Python DataFrame.

False. the `sasdata` method creates a pointer to a SAS dataset, and by itself does not import or export data.

Exercise 3.1.2. Can you guess which SAS procedures are invoked by the `columnInfo` and `describe` methods?

```
3  'pandas.io.common',  0.0  29.4000  31.227044  11.610246  8.8000  23.1000  29.4000  39.7000  68.000
```

The `columnInfo` method invokes PROC CONTENTS, and `describe` invokes PROC MEANS.

Notes About Example 3.1:

1. The `sasdata` method creates a file pointer, meaning a direct connection to a disk-based SAS dataset, whereas the `sasdata2dataframe` method used in Section 2 examples loads a SAS dataset into memory as a pandas DataFrame.
2. `columnInfo` and `describe` are examples of "convenience methods" that implicitly invoke SAS procedures. Specifically, `columnInfo` invokes the CONTENTS procedure, and `describe` invokes the MEANS procedure.
3. Additional convenience methods are listed in the SASPy documentation at https://sassoftware.github.io/saspy/api.html#sas-data-object.

4. The `sas` object represents a connection to a SAS session, and was created in section 0 above.

'pandas.io.gbq',

Example 3.2 Display generated SAS code

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

'pandas.io.parsers',

```
1 sas.teach_me_SAS(True)
2 fish_sds.describe()
3 sas.teach_me_SAS(False)
```

```
proc means data=sashelp.'fish'n  stackodsoutput n nmiss median mean std min p25 p50 p75 max;run;
```

'pandas.io.sas.sasreader'

Line-by-Line Code Explanation:

- Line 1: Set `teach_me_SAS` to `True`. (This is a global effect that will apply to all `saspy` method calls submitted from this point forward.)
- Line 2: Invoke the `describe` method. (Because `teach_me_SAS` is set to `True`, the SAS code generated by the `describe` method is displayed, but not executed.)
- Line 3: Set `teach_me_SAS` to `False`, reversing the effect in Line 1.

'pandas.util'

Exercise 3.2.1. Imagine `teach_me_SAS` had been set to True when the `columnInfo` method was called in Example 3.1. What SAS code might have been displayed instead?

'pandas.util._validators'

```
ODS SELECT Variables; /* used to restrict output to only the list of columns in the dataset */
PROC CONTENTS DATA=sashelp.fish;
RUN;
```

'pexpect.pty_spawn',

Exercise 3.2.2. Write several lines of Python code to execute your SAS code from Exercise 3.2.1 and display the results.

```
'pickleshare',
```

```
1 sas_submit_return_value = sas.submit(
2     '''
3         ODS SELECT Variables; /* used to restrict output to only the list of columns in the dataset */
4         PROC CONTENTS DATA=sashelp.fish;
5         RUN;
6     '''
7 )
8 sas_submit_results = sas_submit_return_value['LST']
9 display(HTML(sas_submit_results))
```

The SAS System

The CONTENTS Procedure

Alphabetic List of Variables and Attributes			
#	**Variable**	**Type**	**Len**
6	Height	Num	8
3	Length1	Num	8
4	Length2	Num	8
5	Length3	Num	8
1	Species	Char	9
2	Weight	Num	8
7	Width	Num	8

```
'prompt_toolkit.application.current',
```

Notes About Example 3.2:

1. Lines 1 and 3 can be thought of as a "Teach Me SAS" sandwich, similar to how an "ODS Sandwich" can be used to toggle output to a specific destination.

2. `True` and `False` are standard Python objects. Like their SAS equivalents, they are interchangeable with the values `1` and `0`, respectively. They are also case sensitive; e.g., `False` is not the same as `false`.

3. The `teach_me_SAS` method allows us to extract (and modify) the SAS code generated by convenience methods. For example, the `describe` convenience method doesn't allow us to set classification variables for PROC MEANS. However, we can use `teach_me_SAS` to generate the underlying SAS code, add a CLASS statement, and then execute the modified SAS code using the `submit` method (see Example 2.3).

4. The `sas` object represents a connection to a SAS session, and was created in section 0 above.

```
'prompt_toolkit.eventloop.event',
```

Example 3.3 Adding variables to a SAS dataset

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
'prompt_toolkit.formatted_text.ansi',
```

```
class_sds = sas.sasdata(
    table='class',
    libref='sashelp'
)
class_bmi_sds = sas.sasdata(
    table='class_bmi',
    libref='work'
)
class_sds.add_vars(vars = {'bmi':'(Weight/Height**2)*703'}, out = class_bmi_sds)

display(class_bmi_sds.head())
print('\n')
display(class_bmi_sds.means())
```

```
Table work.class_bmi does not exist. This SASdata object will not be useful until the data set is created.

126                                                         The SAS System                   Wednesday, Februa

1977
1978         data work.'class_bmi'n ; set sashelp.'class'n ;
1979         bmi = (Weight/Height**2)*703;
1980         ; run;
1981
1982
1983

127                                                         The SAS System                   Wednesday, Februa

1984
```

	Name	Sex	Age	Height	Weight	bmi
0	Alfred	M	14.0	69.0	112.5	16.611531
1	Alice	F	13.0	56.5	84.0	18.498551
2	Barbara	F	13.0	65.3	98.0	16.156788
3	Carol	F	14.0	62.8	102.5	18.270898
4	Henry	M	14.0	63.5	102.5	17.870296

	Variable	N	NMiss	Median	Mean	StdDev	Min	P25	P50	P75	Max
0	Age	19.0	0.0	13.000000	13.315789	1.492672	11.000000	12.000000	13.000000	15.000000	16.00000

Line-by-Line Code Explanation:

- Lines 1-4: Create a file pointer to the SAS dataset sashelp.class (a physical file on disk).
- Lines 5-8: Create a file pointer to a SAS dataset that does not exist (yet), naming the SAS dataset work.class_bmi.
- Line 9: Starting with the SAS dataset represented by class_sds (i.e., sashelp.class), calculate the new variable BMI, and store the resulting dataset in the SAS dataset represented by class_bmi_sds (i.e., work.class_bmi). (Note: The `add_vars` method also prints the log of the corresponding SAS DATA step, by default.)

- Lines 11-13: Display the first 5 rows of the result, along with some summary statistics.

```
        'prompt_toolkit.validation',
        'prompt_toolkit.widgets'
```

Exercise 3.3.1. Copy the code from Example 3.3, paste it below and add a `dsopts=` parameter to the `sasdata` method in Lines 1-4. (Just like the `sasdata2dataframe` method used in Example 2.1, the `sasdata` method has a `dsopts` argument that allows dataset options like `where` and `obs` to be specified as key-value pairs in a dictionary.)

```
        'psutil',
```

```
 1 class_input_dsopts = {
 2     'where' : ' Age > 13',
 3     'keep'  : ['Name','Age','Height','Weight'],
 4 }
 5 class_sds = sas.sasdata(
 6     table='class',
 7     libref='sashelp',
 8     dsopts=class_input_dsopts
 9 )
10 class_bmi_sds = sas.sasdata(
11     table='class_bmi',
12     libref='work'
13 )
14 class_sds.add_vars(vars = {'bmi':'(Weight/Height**2)*703'}, out = class_bmi_sds)
15 
16 display(class_bmi_sds.head())
```

```
271                                                        The SAS System                   Wednesday, Februa

2864
2865          data work.'class_bmi'n ; set sashelp.'class'n (where=( Age > 13) keep=Name Age Height Weight );
2866          bmi = (Weight/Height**2)*703;
2867          ; run;
2868
2869
2870

272 'pygments_filters'                                     The SAS System                   Wednesday, Februa
```

Exercise 3.3.2. Copy the code from Example 3.3, paste it below, and create one or more additional variables in the SAS dataset work.class_bmi by adding additional key-value pairs to the dictionary used for the `vars=` parameter of the `add_vars` method.

```
class_sds = sas.sasdata(
    table='class',
    libref='sashelp'
)
class_bmi_sds = sas.sasdata(
    table='class_bmi',
    libref='work'
)
class_sds.add_vars(
    vars = {
        'bmi':'(Weight/Height**2)*703',
        'first_initial':'substr(Name,1,1)'
    },
    out = class_bmi_sds
)

display(class_bmi_sds.head())
```

```
246                                                          The SAS System                    Wednesday, Februa

2716
2717         data work.'class_bmi'n ; set sashelp.'class'n ;
2718         bmi = (Weight/Height**2)*703;
2719         first_initial = substr(Name,1,1);
2720         ; run;
2721
2722
2723

247                                                          The SAS System                    Wednesday, Februa

2724
```

	Name	Sex	Age	Height	Weight	bmi	first_initial
0	Alfred	M	14.0	69.0	112.5	16.611531	A
1	Alice	F	13.0	56.5	84.0	18.498551	A
2	Barbara	F	13.0	65.3	98.0	16.156788	B

Notes About Example 3.3:

1. The `sasdata` method can be used to create a pointer to a dataset that does not exist. This can still be useful if you intend to create the dataset later.
2. The `add_vars` method uses a SAS DATA step to assign values to new variables. If no `out=` argument is specified, the dataset will be modified in place.
3. Just like the `sasdata2dataframe` method used in Section 2 examples, the `sasdata` has a `dsopts` argument allowing dataset options to be passed to SAS. However, because `sasdata` only creates a file pointer, the dataset options only affect the values returned when the SAS dataset is read from disk. The underlying SAS dataset itself will not be changed unless `dsopts` is specified for an output dataset.
4. Just like its pandas counterpart, the `head` method returns the first 5 rows of a `sasdata` object (or the corresponding dataset in its entirety, if there are 5 or fewer observations). We can also control the number of rows; e.g., `class_bmi_sds.head(3)` returns the first 3 rows.

5. The `sas` object represents a connection to a SAS session, and was created in section 0 above.

```
    'requests.packages.chardet.sjisprober'
```

Section 4. Staying D.R.Y.

```
    'requests.packages.idna',
```

Example 4.1. Imitate the SAS Macro Processor

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercise that follows, only looking at the explanatory notes for hints when needed.

```
    'requests.packages.urllib3.connectionpool',
```

```
sas_code_fragment = 'TITLE "Dataset: sashelp.{dsn}"; PROC MEANS DATA=sashelp.{dsn}; RUN; TITLE;'
for dataset_name in ['fish', 'class', 'iris']:
    sas_submit_return_value = sas.submit(
        sas_code_fragment.format(dsn=dataset_name)
    )
    print(sas_submit_return_value['LOG'])
    display(HTML(sas_submit_return_value['LST']))
```

```
342                                                  The SAS System                Wednesday, Fe

3272         ods listing close;ods html5 (id=saspy_internal) file=_tomods1 options(bitmap_mode='inline') de
3272       ! ods graphics on / outputfmt=png;
3273
3274         TITLE "Dataset: sashelp.fish"; PROC MEANS DATA=sashelp.fish; RUN; TITLE;
3275
3276
3277         ods html5 (id=saspy_internal) close;ods listing;
3278

343                                                  The SAS System                Wednesday, Fe

3279
```

Dataset: sashelp.fish

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
Weight Length1 Length2 Length3 Height Width	158 159 159 159 159 159	398.6955696 26.2471698 28.4157233 31.2270440 8.9709937 4.4174855	359.0862037 9.9964412 10.7163281 11.6102458 4.2862076 1.6858039	0 7.5000000 8.4000000 8.8000000 1.7284000 1.0476000	1650.00 59.0000000 63.4000000 68.0000000 18.9570000 8.1420000

```
344                                                  The SAS System                Wednesday, Fe

3282         ods listing close;ods html5 (id=saspy_internal) file=_tomods1 options(bitmap_mode='inline') de
3282       ! ods graphics on / outputfmt=png;
3283
3284         TITLE "Dataset: sashelp.class"; PROC MEANS DATA=sashelp.class; RUN; TITLE;
3285
3286
3287         ods html5 (id=saspy_internal) close;ods listing;
3288

345                                                  The SAS System                Wednesday, Fe

3289
```

Dataset: sashelp.class

Line-by-Line Code Explanation:

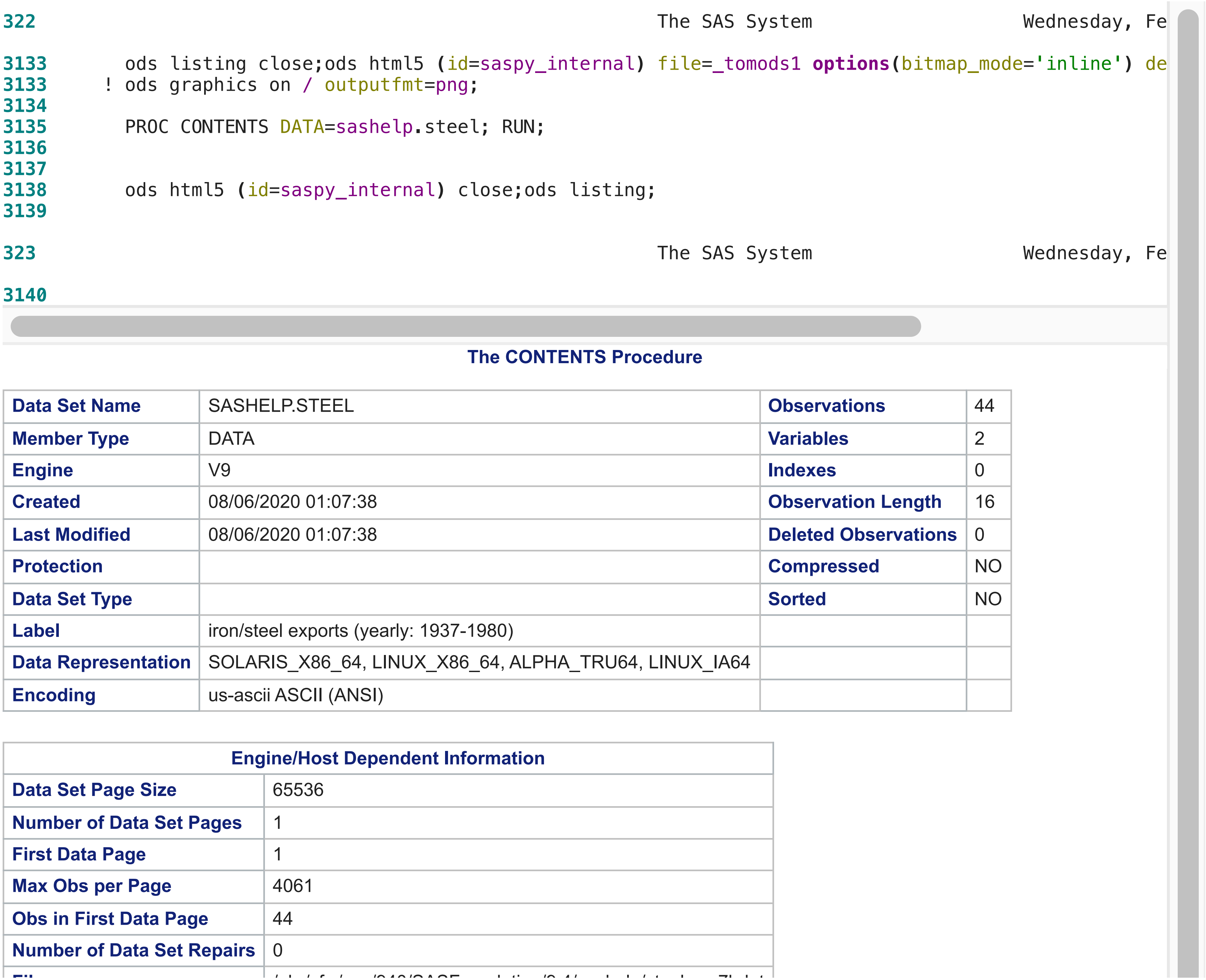
- Line 1: Create a string object named `sas_code_fragment` with a templating placeholder `{dsn}` in curly brackets. The portion in brackets will be replaced with other strings in subsequent uses of `sas_code_fragment`.
- Line 2: Initiate a for-loop over the three values in the list `['fish', 'class','iris']`. (In other words, the body of the for-loop, meaning all subsequent lines that are indented, will be executed three time. The first time, the value of the index variable `dataset_name` will be `'fish'`, the second time `dataset_name` will be `'class'`, and the third time `dataset_name` will be `'iris'`)
- Lines 3-7: Define the body of the for-loop to use the `submit` method on `sas_code_fragment`. The `format` method is used to replace the placeholder `{dsn}` with the current value of the index variable `dataset_name`. SAS logs and output are also printed for each iteration of the loop, per Lines 6-7.

```
3297 ods html5 (id=saspy internal) close;ods listing;
```

Exercise 4.1.1. Write several lines of Python code to accomplish the following: Using the above example as a model, print out the results of applying the SAS CONTENTS procedure to the SAS datasets sashelp.steel and sashelp.tourism, and output the results.

```
3299 'secrets'
```

```
sas_code_fragment = 'PROC CONTENTS DATA=sashelp.{dsn}; RUN;'
for dataset_name in ['steel','tourism']:
    sas_submit_return_value = sas.submit(
        sas_code_fragment.format(dsn=dataset_name)
    )
    print(sas_submit_return_value['LOG'])
    display(HTML(sas_submit_return_value['LST']))
```



```
322                                                     The SAS System                Wednesday, Fe

3133         ods listing close;ods html5 (id=saspy_internal) file=_tomods1 options(bitmap_mode='inline') de
3133       ! ods graphics on / outputfmt=png;
3134
3135         PROC CONTENTS DATA=sashelp.steel; RUN;
3136
3137
3138         ods html5 (id=saspy_internal) close;ods listing;
3139

323                                                     The SAS System                Wednesday, Fe

3140
```

The CONTENTS Procedure

Data Set Name	SASHELP.STEEL	**Observations**	44
Member Type	DATA	**Variables**	2
Engine	V9	**Indexes**	0
Created	08/06/2020 01:07:38	**Observation Length**	16
Last Modified	08/06/2020 01:07:38	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label	iron/steel exports (yearly: 1937-1980)		
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	us-ascii ASCII (ANSI)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	4061
Obs in First Data Page	44
Number of Data Set Repairs	0

Filename	/pbr/sfw/sas/940/SASFoundation/9.4/sashelp/steel.sas7bdat

Notes About Example 4.1.

1. The end result of this Example is to construct and submit the following SAS code to the SAS kernel:

   ```
   TITLE "Dataset: sashelp.fish"; PROC MEANS DATA=sashelp.fish; RUN; TITLE;
   TITLE "Dataset: sashelp.class"; PROC MEANS DATA=sashelp.class; RUN; TITLE;
   TITLE "Dataset: sashelp.iris"; PROC MEANS DATA=sashelp.iris; RUN; TITLE;
   ```

   While it may have been fewer keystrokes to submit this directly, the Python code illustrates the general software engineering principle "Don't Repeat Yourself" (aka D.R.Y.). Think about how much easier it would be to extend the Example to a list of one hundred datasets --- and how much less error prone it would be.

2. The same outcome could also have been achieved with the following SAS macro code:

   ```
   %MACRO loop(dsn_list);
       %LET list_length = %sysfunc(countw(&dsn_list));
       %DO i = 1 %TO &list_length;
           %LET dsn = %scan(&dsn_list.,&i.);
           TITLE "Dataset: sashelp.&dsn.";
           PROC MEANS DATA=sashelp.&dsn.;
           RUN;
           TITLE;
       %END;
   %MEND;
   %loop(fish class iris)
   ```

   However, note the following differences:

  - Python allows us to concisely repeat an arbitrary block of code by iterating over a list using a for-loop. In other words, the body of the for-loop (meaning everything indented underneath it, since Python uses indentation to determine scope) is repeated for each string in the list `['fish','class','iris']`.
  - The SAS macro facility only provides do-loops based on numerical index variables (the macro variable `&i.` above), so clever tricks like implicitly defined arrays (macro parameter `dsn_list` above) need to be used together with functions like `%SCAN` to extract a sequence of values. A combination of the macro function `%SYSFUNC` and the data-step function `countw` also need to be used to determine the "length" of `dsn_list`.

3. The `sas` object represents a connection to a SAS session, and was created in section 0 above.

Number of Data Set Pages	1

Wrapping Up: Call to Action!

Obs in First Data Page	29

Want some ideas for what to do next? Here are our suggestions:

1. Continue learning Python.
   - For general programming, we recommend starting with these:
     - Automate the Boring Stuff with Python, a free online book with numerous beginner-friendly hands-on projects
     - Fluent Python, which provided a deep dive into Intermediate to Advanced Python concepts
   - For data science, we recommend starting with these:
     - A Whirlwind Tour of Python, a free online book with coverage of essential Python features commonly used in data science projects
     - Python for Data Analysis, which provided a deep dive into the `pandas` package by its creator, Wes McKinney
   - For web development in Python, we recommend starting with this:
     - The Flask Mega-Tutorial, a freely accessible series of blog posts covering essential features of developing dynamic websites with the `flask` web framework
2. Consider taking our Beyond the Basics class on 10MAR2023.