Presentation Abstract
The WebFOCUS Python Adapter, introduced in WebFOCUS 8.2.05, turns Python scripts into computation engines for WebFOCUS COMPUTE expressions. In this hands-on lab we will learn how to configure the Python Adapter and create wrappers (synonyms) for Python scripts. We study and run example Python scripts to match and extract text using regular expressions (patterns), apply a linear regression model, and use TensorFlow for image classification. Code available on GitHub (https://github.com/ibi)
## Contents

Business Need .................................................................................................................. 4  
What We Will Do Today ..................................................................................................... 5  
A Crash Course in Python, Highly Abridged .................................................................... 7  
  Starting the Python Interpreter ......................................................................................... 7  
  Variable Assignment ......................................................................................................... 9  
  Whitespace Formatting ....................................................................................................... 9  
  Modules ............................................................................................................................... 10  
  Functions ........................................................................................................................... 10  
  Strings ............................................................................................................................... 11  
  Lists .................................................................................................................................. 11  
  Dictionaries ....................................................................................................................... 12  
  Control Flow ...................................................................................................................... 12  
Example 1: Python Regular Expressions for Extracting Email Addresses ...................... 14  
  Running the Example ........................................................................................................ 15  
  The Python Script ............................................................................................................. 17  
  The WebFOCUS Procedure, aka Focexec or Fex ............................................................ 22  
  Creating the Synonym ..................................................................................................... 23  
Example 2: Predicting the auction price of Bordeaux wine .............................................. 30  
  Running the Example ........................................................................................................ 32  
  The Python Script 1 .......................................................................................................... 34  
  The Python Script 2 .......................................................................................................... 36  
  The WebFOCUS Procedure, aka Focexec or Fex ............................................................ 38  
  Creating the Synonym ..................................................................................................... 38  
Example 3: TensorFlow for Image Classification .............................................................. 39  
  Running the Example ........................................................................................................ 40  
  The WebFOCUS Procedure, aka Focexec or Fex ............................................................ 43  
  Creating the Synonym ..................................................................................................... 45  
We Have Completed Our Lab! ............................................................................................ 45  
Appendix ................................................................................................................................ 46  
  Configuring the WebFOCUS Adapter for Python ............................................................ 46  
  Python Adapter Requirements ......................................................................................... 46
Python Adapter Configuration for Windows .......................................................... 46
Python Adapter Configuration for Linux .................................................................... 47
Python Adapter Configuration for z/OS ....................................................................... 48
Business Value ........................................................................................................... 48
Python Tutorial ............................................................................................................ 48
A Little Python Humor ................................................................................................. 49
Python Integrated Environments (IDE) ...................................................................... 50
Running Anaconda 3 and Python.org Distributions on the Same PC ....................... 50
Installing Python 3.6.8 ............................................................................................... 53
Business Need

The First Industrial Revolution began in Britain's textile industry starting in the second half of the 18th century, powered by steam engines. The Second Industrial Revolution spanned the period between 1870 and 1914 and its defining characteristics included the expansion of railroads, machinery driven manufacturing, extensive adoption of the telegraph, the beginning of electrification, the assembly line, and mass production. The Information Age and the Third Industrial Revolution, aka the Digital Revolution, are overlapping and underway; their drivers include computers and clever software (artificial intelligence - AI, and machine learning - ML).

What is Python's role in the Digital Revolution? It is the programmer's connection to high quality software libraries and web services for Artificial Intelligence and Machine Learning. Complex AI and ML algorithms and processes, written in C/C++ or Java, are often packaged with Python APIs or SDKs (application programming interface and software development kit), simplifying their use because Python is easy to learn for basic use and extremely powerful. Examples include scikit-learn (https://scikit-learn.org), TensorFlow (https://www.tensorflow.org/), and Amazon Web Services - AWS (https://aws.amazon.com/sdk-for-python/).

But why use Artificial Intelligence and Machine Learning in business or education or law enforcement or your organization? Let's listen to Erik Brynjolfsson and Andrew McAfee, Director and Co-Director of the MIT Initiative on the Digital Economy:

The most important general-purpose technology of our era is artificial intelligence, particularly machine learning (ML) — that is, the machine’s ability to keep improving its performance without humans having to explain exactly how to accomplish all the tasks it’s given. Within just the past few years machine learning has become far more effective and widely available. We can now build systems that learn how to perform tasks on their own.

Why is this such a big deal? Two reasons. First, we humans know more than we can tell: We can’t explain exactly how we’re able to do a lot of things — from recognizing a face to making a smart move in the ancient Asian strategy game of Go. Prior to ML, this inability to articulate our own knowledge meant that we couldn’t automate many tasks. Now we can.

Second, ML systems are often excellent learners. They can achieve superhuman performance in a wide range of activities, including detecting fraud and diagnosing disease. Excellent digital learners are being deployed across the economy, and their impact will be profound.

What We Will Do Today

In today's lab we will explore three examples: a regular expression to extract email addresses from sentences, a linear regression model to predict the price of Bordeaux wine at auction, and TensorFlow to identify pictures of clothing and footwear. The first example shows how we use Python to extend WebFOCUS' capabilities without using AI or ML. The second example illustrates how a predictive model, built outside of WebFOCUS, is consumed or used by WebFOCUS. And the third example, extending the ideas from the second example, explains how we adapt an online tutorial presented as a Jupyter notebook to the WebFOCUS Adapter for Python and how to handle multiple values returned from a function call.

Using a Python regular expression to extract email addresses
Predicting the auction price of Bordeaux wine using a linear regression model

Image classification using TensorFlow
A Crash Course in Python, Highly Abridged


Warning: this is an extremely short introduction to Python and lacks all details and nuance. It's enough to get us through the lab. To learn more, a good next step is the official Python Tutorial written by Guido van Rossum, the creator of Python.  [https://docs.python.org/3/tutorial/index.html](https://docs.python.org/3/tutorial/index.html)

Starting the Python Interpreter

1. Click the Windows key, scroll down to Python 3.6, open the folder, and click "Python3.6 (64-bit)"

2. The bare-bones Python REPL (read, evaluate, print, loop) window opens. Python is interactive.

   Type \(2+4\) and press [Enter]. We have a simple calculator.
3. Type `import this` and press [Enter]. The "Zen of Python" appears.

```
>>> import this
The Zen of Python, by Tim Peters
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than "right" now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```
What is the obvious way? Writing Pythonic code. To learn more about Pythonic, see:

- PEP 8 — the Style Guide for Python Code [https://pep8.org/]
- The Hitchhiker’s Guide to Python! [https://docs.python-guide.org/]
- Search for "pythonic" in the Wikipedia entry, "Python (programming language)" [https://en.wikipedia.org/wiki/Python_%28programming_language%29]

If you like using Python interactively, check out IPython ([https://ipython.org/]) and Jupyter Notebooks ([https://jupyter.org/]). Both come with the Anaconda distribution.

To save time, the remainder of this crash course will not be hands on during the lab. Of course, you can try the examples at home.

**Variable Assignment**

Python uses duck typing, it if quacks like a duck and walks like a duck... There is no variable type declaration. The equals sign, =, is used for assignment and a double equals sign, ==, is used for equality testing.

```python
>>> foo = 5
>>> bar = 6
>>> foo == bar
False
```

**Whitespace Formatting**

Python uses indentation, white space (use spaces, not tabs!), to delimit blocks of code (four spaces for each level of indentation is most common), and not curly braces as in other languages. Whitespace is ignored inside parentheses, braces, and brackets.

The example below will not work easily in the interactive Python interpreter; a Python IDE such as PyCharm ([https://www.jetbrains.com/pycharm]) or Spyder (included with the Anaconda Python distribution [https://www.anaconda.com/distribution]) is better. Note that Anaconda does not currently work with the Python adapter. However, you can use Anaconda for development and Python from [www.python.org](http://www.python.org) for configuring and using the Python adaptor; be sure to use the same packages and package releases.
for i in [1, 2, 3, 4, 5]:
    print(i)                      # first line in "for i" block
for j in [1, 2, 3, 4, 5]:
    print(j)                  # first line in "for j" block
    print(i + j)              # last line in "for j" block
print(i)                      # last line in "for i" block
print("done looping")

Example from Data Science from Scratch: First Principles with Python by Joel Grus, Chapter 2 -- A Crash Course in Python. O'Reilly Media Inc., 2015. Updated for Python 3, using `print(foo)` and not Python 2.7's `print foo`.

**Modules**

To learn more: [https://docs.python.org/3/tutorial/modules.html](https://docs.python.org/3/tutorial/modules.html)

Python has a batteries included philosophy ([https://www.python.org/dev/peps/pep-0206/](https://www.python.org/dev/peps/pep-0206/)):

The Python source distribution has long maintained the philosophy of "batteries included" -- having a rich and versatile standard library which is immediately available, without making the user download separate packages. This gives the Python language a head start in many projects.

Modules contain features (classes, objects, functions, executable statements, etc.) not in the core Python language or loaded by default. And sometimes third party modules, e.g., scikit-learn and pandas, are best for your project. The `import` statement is used to import an entire module, a specific object from a module, and to assign an alias to a module.

```python
import re  # import entire regular expressions module
from pandas import dataframe  # import only dataframe object from pandas
import matplotlib as plt  # import entire module and assign alias 'plt'
from sklearn.linear_model import LinearRegression  # import submodule
```

What is the dot (.) in `sklearn.linear_model` about? A package is a collection of modules. `sklearn` is a module and `linear_model` is a submodule. Think of the dot as meaning "is a member of"; `linear_model` is a member of `sklearn`. Note: `sklearn` is the package name of scikit-learn library ([https://scikit-learn.org/](https://scikit-learn.org/)).

**Functions**

A function is a rule for taking zero or more inputs and returning a corresponding output. In Python, we typically define functions using `def` (Data Science from Scratch: First Principles with Python by Joel Grus, p. 18).
def double(x):
    """this is where you put an optional docstring
    that explains what the function does.
    for example, this function multiplies its input by 2"""
    return x * 2
>>>
number_doubled = double(5)
>>>
number_doubled
10

Strings

Strings are delimited by a pair or single or double quotes.

>>> foo = 'this is a string'
>>> bar = "and so is this"
>>> bat = "it's ok to embed a single quote inside a double quoted string"
>>> etc = 'and "this is also" permitted'

Lists

Python has a number of compound data types, used to group together other values. The most versatile is the list, which can be written as a list of comma-separated values (items) between square brackets. Lists might contain items of different types, but usually the items all have the same type.

https://docs.python.org/3/tutorial/introduction.html

>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
>>> squares.append(36)
>>> squares
[1, 4, 9, 16, 25, 36]
>>> squares[0]
# index access
1
>>> squares[0:3]
# slicing
[1, 4, 9]
>>> squares[1] = 9999
>>> squares
[1, 9999, 9, 16, 25, 36]
>>>len(squares)
# use built-in function on lists
6
>>> sum(squares)
# use built-in function on lists
91
Dictionaries
https://docs.python.org/3/tutorial/datastructures.html#dictionaries
Dictionaries are sometimes found in other languages as “associative memories” or “associative arrays”. Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys.

```python
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'jack': 4098, 'sape': 4139, 'guido': 4127}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'jack': 4098, 'guido': 4127, 'irv': 4127}
>>> list(tel)
['jack', 'guido', 'irv']
>>> sorted(tel)
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

Control Flow

If Statement
https://docs.python.org/3/tutorial/controlflow.html#if-statements
Perhaps the most well-known statement type is the if statement. For example:

```python
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
```

There can be zero or more `elif` parts, and the `else` part is optional. The keyword `elif` is short for `else if`, and is useful to avoid excessive indentation. An `if ... elif ... elif ...` sequence is a substitute for the `switch` or `case` statements found in other languages.

For Statement

https://docs.python.org/3/tutorial/controlflow.html#for-statements

The `for` statement in Python differs a bit from what you may be used to in C or Pascal. Rather than always iterating over an arithmetic progression of numbers (like in Pascal), or giving the user the ability to define both the iteration step and halting condition (as C), Python’s `for` statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence. For example:

```python
>>> # Measure some strings:
... words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print(w, len(w))
...  
cat 3
window 6
defenestrate 12
```
Example 1: Python Regular Expressions for Extracting Email Addresses

The scenario: You have been sent a list of sentences with embedded email addresses and your boss wants the email addresses extracted yesterday. Trust me (okay, don't trust me) that using regular expressions is the best solution. Most modern programming languages support regular expressions. What are regular expressions? A mini-language for matching patterns in text.

From A.M Kuchling's Regular Expression HOWTO:
https://docs.python.org/3.6/howto/regex.html

Regular expressions (called REs, or regexes, or regex patterns) are essentially a tiny, highly specialized programming language embedded inside Python and made available through the re module. Using this little language, you specify the rules for the set of possible strings that you want to match; this set might contain English sentences, or e-mail addresses, or TeX commands, or anything you like. You can then ask questions such as “Does this string match the pattern?” or “Is there a match for the pattern anywhere in this string?” You can also use REs to modify a string or to split it apart in various ways.

Here is a simple example matching any five letter string starting with 'a' and ending with 's'.

```
^a...s$
```

And here is a sample from the sentences to be processed. Yes, I know the words are gibberish but the email addresses are properly structured.

```
Lorem ipsum JohnNTorres@dayrep.com dolor sit amet, consectetur adipiscing elit.
Integer non velit CarlosKSettle@dayrep.com ex.
Nullam viverra mauris ac metus semper CharlesKSolomon@armyspy.com dapibus.
Donec congue vitae nisi MichaelMBriggs@fleckens.hu convallis maximus.
Vestibulum at KristinHJames@dayrep.com ornare nibh.
Aenean eu ChristinaDSchwartz@gustr.com venenatis justo.
Nam porttitor enim sed justo MarieLCausey@einrot.com semper, ac tempor erat aliquam.
Donec fermentum, eros ut dignissim sollicitudin, ex magna interdum mauris,
GeraldineRWilliams@cuvox.de et auctor justo mauris eget metus.
In ullamcorper pulvinar BrendaBBarnhart@cuvox.de arcu, ac faucibus quam efficitur eu.
Pellentesque vitae risus MaryFMungia@cuvox.de enim.
```
Running the Example

1. Open the Google Chrome web browser if not already open (the icon is on the desktop) and enter the URL for the WebFOCUS Reporting Server console: \texttt{http://localhost:8121}

2. Let’s run \texttt{consume_regex.fex} \textit{(the file extension does not appear)}.
   a. Expand the folder \texttt{regex}
   b. Right-click \texttt{consume_regex} and select Run.

3. Hey, it worked! I know we should not be surprised. But this is a hands-on lab and, well, stuff happens.
4. How did we get here? No, I don't mean traveling to Orlando, I mean getting the Python script to work with WebFOCUS.

First, close the `consume_regex` tab by clicking its 'X'.

5. Right-click `regex_example.py` and select `View`. 
The Python Script

1. Let's take a look at the code.

```python
import csv
import re

# Example using input and output csv files with header row

def get_email_address(csvin, csvout):
    """Retrieve an email address from a sentence."""

    pattern = r'\b[A-Z0-9_.%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}\b'
    prog = re.compile(pattern, flags=re.IGNORECASE)

    with open(csvin, 'r', newline='') as file_in, 
         open(csvout, 'w', newline='') as file_out:
        reader = csv.DictReader(file_in, 
                                  quoting=csv.QUOTE_NONNUMERIC)
        writer = csv.DictWriter(file_out, fieldnames=['email_address'], 
                                 quoting=csv.QUOTE_NONNUMERIC)
        writer.writeheader()

        for row in reader:
            m = prog.search(row['SENTENCE'])
            result = m.group(0) if m else 'NA'
            writer.writerow({'email_address': result})

if __name__ == '__main__':
    csvin = 'sentences.csv'
    csvout = 'test_out.csv'
    get_email_address(csvin, csvout)
```

**Line 1:** Import the `csv` module for reading and writing CSV files.

**Line 2:** Import the `re` module for regular expressions.

**Line 4:** Python comments begin with the hash character `#`. Python does not have multi-line comments and doc strings can be used as an alternative.
Line 7: Define the function `get_email_address` with parameters (not arguments) `csvin` and `csvout`. Later, we will use `csvin` (think 'csv file in') as it contains the name of the temporary file (in foccache) holding the records to process in our Python script. We do not manually create the csv hold file; it is automatically created for us by the Reporting Server (more details later).

Line 8: A doc string. Can be multiple lines and used by tools for automatically generating documentation from source code. See https://www.python.org/dev/peps/pep-0257/

Line 10: This is the regex pattern. 'r' means 'raw'; back slashes, '\', are treated as literal characters and not as the beginning of an escape sequence. Let's break down the regular expression:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b[A-Z0-9._%+-]+@[A-Z0-9.-]+.[A-Z]{2}\b</td>
<td>A word boundary. This is a zero-width assertion that matches only at the beginning or end of a word. A word is defined as a sequence of alphanumeric characters, so the end of a word is indicated by whitespace or a non-alphanumeric character.</td>
</tr>
<tr>
<td>\b[A-Z0-9._%+-]+@[A-Z0-9.-]+.[A-Z]{2}\b</td>
<td>A character class, which is a set of characters that you wish to match. Characters inside the square brackets can be listed individually, or as a range of characters indicated by giving two characters and separating them by a '-' . The square brackets are not characters to be matched, but meta-characters defining the search pattern.</td>
</tr>
<tr>
<td>\b[A-Z0-9._%+-]+@[A-Z0-9.-]+.[A-Z]{2}\b</td>
<td>The characters A through Z. Any single one of them. The + meta-character outside the closing ] indicates we can have one or more matches of any of the characters inside the character class.</td>
</tr>
<tr>
<td>\b[A-Z0-9._%+-]+@[A-Z0-9.-]+.[A-Z]{2}\b</td>
<td>The characters 0 through 9.</td>
</tr>
</tbody>
</table>
The dot inside a character class is a literal character, a dot. Outside a character class, or if not escaped as in '\.', the dot is a meta-character indicating any character, like a wildcard in card games.

Literal characters to be matched.

The + meta-character outside the closing ] indicates we can have one or more matches of any of the characters inside the character class.

Literal character to be matched. Just one occurrence because there is no following meta-character indicating 'how many.'

A character class, very similar to the one previously described, followed by the + meta-character. See the above explanations.

An escaped dot, indicating to match a literal dot. Without the back slash the dot is a meta-character indicating any character, like a wildcard in card games.

A character class, the letters A through Z.

This repeated qualifier is \{m,n\}, where m and n are decimal integers. This qualifier means there must be at least m matches and at most n. \{2,\} means a lower limit of 2 matches and an upper limit of infinity matches of any letter from A through Z (does not have to be the same letter). \{, 2\} means a lower limit of zero matches and an upper limit of two matches.

A word boundary. See above for definition.
Line 11: Although not required, compiling a pattern results in faster execution. The flag `re.IGNORECASE` means just what is says. Email addresses for matching are treated as case insensitive. By ignoring case, the character class `[A-Z]` will match any characters in the ranges a-z and A-Z. Alternatively, we could have written our pattern as

```
\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\b
```

and omitted `re.IGNORECASE`, but resulting in a longer, more difficult to read pattern, instead of the original

```
\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}\b
```

Lines 13 - 14: We are opening our input and output files in read ("r") mode and write mode ("w"), respectively. The `with` context manager ensures the files are closed when reading and writing have completed. And the `as` assigns alias names to the file streams. A better way to write these lines (the line is wrapped in this document; it should not be wrapper in an actual Python script) is:

```python
with open(csvin, 'r', newline='') as file_in, open(csvout, 'w', newline='') as file_out:
```

Line 16: We are instantiating a CSV file reader that assumes the file has a header record (column titles). When we iterate through `reader` (line 23), it will read a row into a Python dictionary, process it in the code block below, loop, read the next row, etc. In our example, the CSV file column header is "SENTENCE" and the value is "Lorem ipsum JohnNTorres@dayrep.com dolor sit amet, consectetur adipiscing elit."

As a Python dictionary, it is:

```
{"SENTENCE": "Lorem ipsum JohnNTorres@dayrep.com dolor sit amet, consectetur adipiscing elit."}
```

There is no single or official definition for CSV files. However, the Internet Engineering Task Force’s Request for Comment # 4180 comes close (IETF and RFC):


The `csv.DictReader` argument `quoting=csv.QUOTE_NONNUMERIC` means all values that are not numbers (strings) will be quoted.

For more about the `csv` module, see [https://docs.python.org/3.6/library/csv.html](https://docs.python.org/3.6/library/csv.html)

Line 18: We are instantiating a CSV file writer that will write a header record with column titles, `email_address` in our example. `csv.DictWriter` reads a
dictionary and writes a row to a CSV file. The `csv.DictWriter` argument `quoting=csv.QUOTE_NONNUMERIC` means all values that are not numbers (strings) will be quoted.

**Line 21:** Write the header record to the csv file.

**Line 23:** We are looping or iterating (for) through each row in the CSV file's input stream, `reader`. The variable name following for, `row`, is assigned the read row as a dictionary. The variable name does not have to be `row`; `foo` is just as good. Python convention is to use `row`.

**Line 24:** `prog` contains our compiled regular expression object. We call its `search` function or method and pass it the string to search for an email address, the value of the 'SENTENCE' entry in the dictionary row: `row['SENTENCE']`. The result of the search is assigned to `m`, as in `match`, another Python convention.

**Line 25:** If there is a match, then the result is in `m.group(0)` . If there is no match then `m` is `None`, meaning no value. The Python adapter requires we return a value for each row in `csvin`. In our example we return 'NA' if there is no match. If returning a numeric value, you might return -1, 9999, etc. to signify missing data. Line 25 uses the Python ternary operator (see https://stackoverflow.com/questions/394809/does-pytho...ternary-conditiona...operator). Although compact, it can be confusing and I probably should not have used it for the lab. Line 25 can be rewritten as:

```python
if m:
    result = m.group(0)
else:
    result = 'NA'
```

For a detailed explanation of `m.group(0)`, see https://docs.python.org/3.6/library/re.html

**Line 26:** Let’s write the result to the CSV file. `csv.DictWriter` objects, e.g., `writer`, requires a dictionary as input.

**Line 29:** Python scripts are evaluated and executed from top to bottom. When a function definition, `def`, is encountered the function is instantiated as an object in memory but not executed; it is not called.

`__name__` is a special Python variable, pronounced "dunder name dunder." dunder is Python-speak for double underscore. When a Python script is run directly, not imported as a module, then `__name__` is assigned the value 'main'.

But when our script is executed by the Python adapter, it is imported and treated as a module -- which it is in this scenario. The script's name is `regex_example.py`
and when it is imported, `regex_example.__name__` is assigned the value 'regex_example', inside the module's namespace.

When our script is run directly, the indented statements (meaning a block of code) following
```python
if __name__ == '__main__':
```
are executed. This means we can test our script outside of WebFOCUS, which makes development and testing much easier. Then, when ready to use our script with the Python adapter, no changes need to be made as the statements following
```python
if __name__ == '__main__':
```
will not be executed because importing the script will set `__name__` to 'regex_example' inside `regex_example`'s namespace and the `if` statement's test will fail.


To learn more about regular expressions:

- [https://docs.python.org/3.6/library/re.html](https://docs.python.org/3.6/library/re.html)
- [https://docs.python.org/3.6/howto/regex.html](https://docs.python.org/3.6/howto/regex.html)
- [https://www.amazon.com/Mastering-Regular-Expressions-Jeffrey-Friedl/dp/0596528124](https://www.amazon.com/Mastering-Regular-Expressions-Jeffrey-Friedl/dp/0596528124)

The WebFOCUS Procedure, aka Focexec or Fex

How do we call the Python script from WebFOCUS? By using the WebFOCUS function `PYTHON` in a WebFOCUS `COMPUTE` (This requires licensing the WebFOCUS Adapter for Python; contact your Information Builders account executive for details).

Below is `consume_regex.fex` with nonessential statements, e.g., styling, removed. The full procedure is in the app folder `..\ibi\apps\regex`.

```
TABLE FILE SENTENCES
PRINT
  SENTENCE AS Sentence
  COMPUTE Email/A100 = PYTHON(regex_example, SENTENCE, EMAIL_ADDRESS); AS 'Email'
END
```

`regex_example`, although the name of our Python script minus the file extension, as an argument to the `PYTHON` function in the example above, is actually the name of the Python script's WebFOCUS synonym or master file description (.mas) and the access file (.acx). We will cover creating the synonym later.
SENTENCE is the name of the field being passed to the Python script. More than one field can be passed; simply separate them by commas. EMAIL_ADDRESS is the name of the returned value and matches the field name used when writing the results to a CSV file.

But how were the contents of csvin, the name of the CSV file read by the Python script, created? No hold file was explicitly created in our focexec.

The WebFOCUS Reporting server parses the focexec, bundles all the rows or records of one or more fields to be sent to our Python script into one CSV file, whose name is in csvin. How did the variable csvin get its value and passed to the Python script? It was not explicitly set in the script. The Python adapter uses the Python/C API (https://docs.python.org/3/c-api/index.html) and creates csvin and csvout as global variables available (in the same scope) to our imported Python script.

Why pass all the rows at one time? For greater performance and to make the entire file available for data frame operations, etc.

Creating the Synonym
Creating the synonym requires a file of one or more sample records as a CSV file with header (no synonym is required for this sample file). The file contains only input fields for the Python script.

And our Python script must have a user defined function to call. Recall the Python adapter imports our Python script; it must have a starting point other than __main__. Python scripts used with the Python adapter can have multiple user defined functions; one serves as the starting point. In our example the Python user defined function is get_email_address.

The sample record(s) should represent the maximum value for each field. For integer and floating point or double numeric fields, the largest possible number that could be sent to the Python script. For floating point or double fields, be sure to include a decimal point and as many decimal places as needed. For string or alphanumeric fields, the longest string possible.

Our sample file, sentences_sample_data.csv, is below. Note it contains a header and a single record (wrapped in this document).

"SENTENCE"
"Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Praesent purus elit, lacinia ac AdelePThomas@dayrep.com gravida sed, ornare ut diam."

1. In the Reporting Server console, in the Applications tree...
   a. Right-click the regex folder
b. Select **New**

c. Select **Synonym**

2. We are now at the tree of Configured Adapters.

   a. Right-click **PYTHON Python 3.6**

   b. Select **Create metadata objects**. A synonym or master file description is the same thing as a metadata object.
3. We are at the Create Synonym for PYTHON form.
   
   a. Click the ellipsis (…) for the field Python Script. Navigate to the regex app folder and select regex_example.py.
   
   b. The Function Name's list box will automatically populate. Select the starting point function. In our example it is get_email_address.
   
   c. Click the ellipsis (…) for the field File with sample input data for the PYTHON Script. Navigate to the regex app folder and select sentences_sample_data.csv
   
   d. Check both Input and Output for the field CSV files with header.
   
   e. The Application field will be automatically set to regex because we began the create synonym process (step 1, above) from the regex folder.
   
   f. We already have a Synonym Named regex_example. Do not overwrite it; use regex_example_2 instead.
4. Click Create Synonym in the ribbon bar.

5. No success message will appear. The Application Directories/Files panel will show the new synonym, regex_example_2.
6. Right-click `regex_example_2` and select Sample Data.

7. The Python adapter will execute `regex_example.py` and pass to it data from `sentences_sample_data.csv`, a single record or row along with a header.

8. But how did the Python adapter "know" where to find the sample data? By parsing the synonym or metadata.
   a. Select the Applications tab
   b. Right-click `regex_example`
   c. Select Metadata Management
   d. Select Edit as Text
9. regex_example.mas is displayed.

This is the result of the create synonym process. Note the SUFFIX is PYTHON. We have two SEGMENTS, INPUT_DATA and OUTPUT_DATA, corresponding to the structure of the sample data file sentences_sample_data.csv and the output of regex_example.py, respectively.

10. Let's look at the access file, regex_example.acx.

   a. Select the Applications tab
b. Right-click `regex_example`

c. Select Metadata Management

d. Select Edit Access File as Text

11. `regex_example.acx` is displayed.

Here we find the values we had entered in the Create Synonym form: the Python script, the starting point function, the sample data file, and if input and output headers are used.

We have completed our first example, Python Regular Expressions for Extracting Email Addresses.
Example 2: Predicting the auction price of Bordeaux wine

Can we predict the future quality and auction price of a wine without tasting it? Professor Orley Ashenfelter, Princeton University economist, can.

Wine Equation Has Noses Out of Joint

Continued From Page 1

The economic Review is convinced he has found a better way.

It is widely agreed that weather influences wine quality. What few understand, he argues, is that a mere handful of facts about the local weather tell almost all there is to know about a vintage. And using the same techniques to forecast, say, the effect of a change in weather on employment in the auto industry, he has gone a long way toward proving his audacious theory.

It's Laptop vs. Nose

Weather-based vintage prediction is not a new idea. What is new is the notion that laptop computers can outperform the most sophisticated noses and palates.

The critical concept, Professor Ashenfelter acknowledges, came from Bruno Prus, owner of Château Cos d'Estournel, in the St.-Estéphe region of Bordeaux. Mr. Prus characterized both the average temperature during the growing season and rainfall during the harvest months to make systematic comparisons between vintages.

Professor Ashenfelter added data for weather rainfall and then rigorously measured their statistical relationships to the most objective measure of quality he could devise: an index of auction prices for about 80 wines after they had been sold.

According to this "multivariate regression analysis," heavy rains in the winter followed by a hot summer improve wine quality, while rainfall before the harvest damages it. The statistical fit from 1952 through 1989 is remarkably good for the red wines of Bordeaux as well as Bordeaux.

Year-to-Year Variations

Michael Broadbent, director of wines at Christie's auction house in London and a leading authority on Bordeaux, points out that the system cannot predict quality differences between estate-bottled wines of the same vintage. And it would yield systematic errors in judgments made by tasters primed primarily on the Cabernet Sauvignon grapes with the minority based on the softer-tasting Merlot grapes, which matures earlier.

But as Steve Ross, a wine enthusiast who is a professor of finance at the Yale School of Management, points out, the simple statistical model does explain basic year-to-year variations without major slips. "If you did not tell them a machine did it," Mr. Ross said, "I think it would pass muster with the wine heads in the business."

Interest in the system has mostly been confined to the small group of wine buffs who are comfortable with econometric techniques. Circulation of Mr. Ashenfelter's semimonthly newsletter, "Liquid Assets" ($38 for a yearly subscription), numbers about 600 - a tiny fraction of the readership of The Wine Advocate (338, with a circulation of more than 27,000), which Mr. Parker publishes from his home in Maryland. But that could change if two of the Princeton economist's predictions prove accurate.

For most young Bordeaux vintages, the Ashenfelter forecasts are within

What is different about this second example as compared to our first example? We will be using a predictive model, linear regression, which was trained (built) and saved using Python. In this example we will not cover topics from the previous example, e.g., creating a Python script's synonym, but new topics such as serializing a model using the Python pickle module.

**Running the Example**

1. Open the Google Chrome web browser if not already open (the icon is on the desktop) and enter the URL for the WebFOCUS Reporting Server console: [http://localhost:8121](http://localhost:8121)

2. Let's run `wine_consume_model.fex` (the file extension does not appear).
   a. Expand the folder `wine`.
   b. Right-click `wine_consume_model` and select Run.

3. Let’s look at data similar to what Orley Ashenfelter used. AGST: Average Growing Season Temperature in degrees Celsius/centigrade (for the grapes), Harvest Season Rainfall in milliliters, Winter Rainfall in milliliters, Age of the wine in years, and the predicted price (dollars? Francs? Euros?) as log base 2.
In the first row, Price is 7.60. 194 log base 2 is 7.6. 2 raised to the 7.6\textsuperscript{th} power (2\textsuperscript{7.6}) is equal to 194.

<table>
<thead>
<tr>
<th>AGST</th>
<th>HarvestRain</th>
<th>WinterRain</th>
<th>Age</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>17.1167</td>
<td>160</td>
<td>600</td>
<td>31</td>
<td>7.60</td>
</tr>
<tr>
<td>16.7333</td>
<td>80</td>
<td>690</td>
<td>30</td>
<td>7.79</td>
</tr>
<tr>
<td>17.1500</td>
<td>130</td>
<td>502</td>
<td>28</td>
<td>7.62</td>
</tr>
<tr>
<td>16.1333</td>
<td>110</td>
<td>420</td>
<td>26</td>
<td>6.97</td>
</tr>
<tr>
<td>16.4167</td>
<td>187</td>
<td>582</td>
<td>25</td>
<td>6.91</td>
</tr>
<tr>
<td>17.4833</td>
<td>187</td>
<td>485</td>
<td>24</td>
<td>7.48</td>
</tr>
<tr>
<td>16.4167</td>
<td>290</td>
<td>763</td>
<td>23</td>
<td>6.56</td>
</tr>
<tr>
<td>17.3333</td>
<td>38</td>
<td>830</td>
<td>22</td>
<td>8.34</td>
</tr>
<tr>
<td>16.3000</td>
<td>52</td>
<td>697</td>
<td>21</td>
<td>7.50</td>
</tr>
<tr>
<td>15.7167</td>
<td>155</td>
<td>608</td>
<td>20</td>
<td>6.56</td>
</tr>
<tr>
<td>17.2667</td>
<td>96</td>
<td>402</td>
<td>19</td>
<td>7.60</td>
</tr>
<tr>
<td>15.3667</td>
<td>267</td>
<td>602</td>
<td>18</td>
<td>5.78</td>
</tr>
<tr>
<td>16.5333</td>
<td>86</td>
<td>819</td>
<td>17</td>
<td>7.52</td>
</tr>
<tr>
<td>16.2333</td>
<td>118</td>
<td>714</td>
<td>16</td>
<td>7.08</td>
</tr>
<tr>
<td>16.5500</td>
<td>244</td>
<td>575</td>
<td>14</td>
<td>6.53</td>
</tr>
<tr>
<td>16.6667</td>
<td>89</td>
<td>622</td>
<td>13</td>
<td>7.35</td>
</tr>
</tbody>
</table>

4. How did we get here? In two steps: one Python script to build the model and another to run or consume the model.

5. Select the Applications tab, right-click wine_build_model.py (near the bottom of the files listing), and select View.
The Python Script

1. Let's take a look at the code.

```python
# wine_build_model.py
# ira Kaplan
# april 24, 2019

import pickle
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

dataset = pd.read_csv('wine.csv')
```
Learning how to build or train a linear regression model is beyond the scope of this lab. What is important to note is saving the model to be used later, for predicting prices on new data.

Line 18: Instantiate the model.

Line 19: Train the model.

Line 21: Choose a filename and extension for our model to be saved.

Line 22: Using the Python pickle module, write the file to disk in binary format. Pickling is also more generally known as serialization.

To learn about the Python pickle module: [https://docs.python.org/3/library/pickle.html](https://docs.python.org/3/library/pickle.html)
The Python Script 2

1. Select the Applications tab, right-click wine_run_model.py (near the bottom of the files listing, and select View.

   ![File Explorer Screenshot]

   Applications
   - wine_consume_model
   - ...
   - /wine
   - wine_run_model.py

  Application Directories/Files
  
<table>
<thead>
<tr>
<th>Application</th>
<th>Size</th>
<th>Date Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>wine_consume_model</td>
<td>220</td>
<td>2019/03/06 08:31:04</td>
</tr>
<tr>
<td>wine_create_synonym</td>
<td>220</td>
<td>2019/05/06 11.55.36</td>
</tr>
<tr>
<td>wine_model</td>
<td></td>
<td>2019/05/06 16.14.04</td>
</tr>
<tr>
<td>wine_sample_data</td>
<td></td>
<td>2019/08/27 16.29.11</td>
</tr>
<tr>
<td>wine_test</td>
<td></td>
<td>2019/05/07 07.51.16</td>
</tr>
<tr>
<td>wine_training</td>
<td></td>
<td>2019/03/19 11.16.20</td>
</tr>
<tr>
<td>wine</td>
<td></td>
<td>2019/05/06 16.14.04</td>
</tr>
<tr>
<td>wine_sample_data</td>
<td></td>
<td>2019/08/27 16.19.18</td>
</tr>
<tr>
<td>wine_test</td>
<td></td>
<td>2019/05/06 07.54.26</td>
</tr>
<tr>
<td>wine_training</td>
<td></td>
<td>2019/03/19 11.16.20</td>
</tr>
<tr>
<td>wine_build_model.py</td>
<td></td>
<td>2019/05/06 08.55.32</td>
</tr>
<tr>
<td>wine_run_model.py</td>
<td>637</td>
<td>2019/05/06 15.19.13</td>
</tr>
<tr>
<td>wine_run_model_with_trace.py</td>
<td>892</td>
<td>2019/05/06 15.47.58</td>
</tr>
<tr>
<td>wine_model.sav</td>
<td>562</td>
<td>2019/05/06 08.55.49</td>
</tr>
</tbody>
</table>

2. Let's take a look at the code.

```python
1  # wine_run_model.py
2  # ira kaplan
3  # april 24, 2019
4  # version 02
5
6  import pickle
7  import pandas as pd
8```
def run_model(csvin, csvout):
    dataset = pd.read_csv(csvin)
    x = dataset[['AGST', 'HarvestRain', 'WinterRain', 'Age']].values
    filename = r'C:\ibi\apps\wine\wine_model.sav'
    regressor = pickle.load(open(filename, 'rb'))
    y_pred = regressor.predict(x)
    df = pd.DataFrame({'Price': y_pred})
    df.to_csv(csvout, index=None, header=True)

if __name__ == '__main__':
    run_model(csvin='wine_test.csv', csvout='wine_test_out.csv')

**Line 11:** This should look familiar; our function definition has `csvin` and `csvout` as its two parameters.

**Line 13:** `pd` (pandas) is reading the entire `csvin` file into a data frame (pandas DataFrame); we are not iterating or looping through the rows. This is common when training, testing, and using models.

**Line 16:** When we built the model we saved (pickled) it as `wine.sav`. Because it was most likely developed in a Python IDE outside of WebFOCUS, the programmer determined to where `wine.sav` was written. **WE MUST** copy it to a folder on the Reporting Server’s host machine and accessible to the Reporting Server. I recommend using the application’s APP folder, e.g., `...\ibi\apps\wine`, but this is not required.

**AND WE MUST** use an absolute path to the file. When in WebFOCUS our current working directory is foccache, not a good location for a Python script because foccache stores temporary files and is deleted after your WebFOCUS session ends. WebFOCUS APP PATH is
not used for Python files. Our Python script, `wine_run_model.py`, is reading `wine.sav`, not WebFOCUS.

**Lines 20 – 21:** We are converting the results into a pandas DataFrame with a column header and then saving it as a CSV file, writing to `csvout`.

### The WebFOCUS Procedure, aka Focexec or Fex

In our first example we described how the WebFOCUS procedure, in a **COMPUTE**, calls the Python adapter. The **COMPUTE** is in bold, below.

```fex
/* wine_consume_model.fex

TABLE FILE wine_training
PRINT
AGST
HarvestRain
WinterRain
Age
COMPUTE Price/D18.2 MISSING ON ALL = PYTHON(wine/wine_model, AGST, HarvestRain, WinterRain, Age, Price); AS 'Price'
END
```

### Creating the Synonym

See our first example for instructions on creating the synonym for the Python script. If you want to rebuild the synonym (I recommend using a new name until you are familiar with the process), the files are:

- **Python script:** `wine_run_model.py`
- **Sample data file:** `wine_sample_data.csv`
Example 3: TensorFlow for Image Classification


We will take code snippets from the TensorFlow tutorial, “Train your first neural network: basic classification,” and adapt them to work with the Python adapter. https://www.tensorflow.org/tutorials/keras/basic_classification

What is different about this third example as compared to our second example? We will not be reviewing the adapted Python code (after Summit it will be posted with comments on the Information Builders GitHub site – https://github.com/ibi) and saving the trained model for later use will require a keras function because pickling is not sufficient.

Before using the Python code from this example back at home, do work through the entire TensorFlow tutorial. This example is a proof of concept and the adapted code is not production quality.

Tutorial: https://www.tensorflow.org/tutorials/keras/basic_classification.

Jupyter notebook: https://github.com/tensorflow/docs/blob/master/site/en/tutorials/keras/basic_classification.ipynb
Running the Example

1. Open the Google Chrome web browser if not already open (the icon is on the desktop) and enter the URL for the WebFOCUS Reporting Server console: http://localhost:8121

2. Let's run image_report_v03.fex (the file extension does not appear).
   a. Expand the folder tensorflow.
b. Right-click image_report_v03 and select Run.

! Warning: It may take 40 seconds or more to run the example.

3. What are we looking at? A photograph of a boot I saved from the Web. It is in color, 1000 x 1000 pixels, in JPEG format (.jpg). The second image is the boot in grayscale at 28 x 28 pixels. This is the format used for model training in the TensorFlow tutorial. I used the Python Pillow module for image operations (https://python-pillow.org/). And the bar chart displays the probabilities this image is classified as one of the following: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, or Ankle boot. How well did we do? Not so good. Our classifier “thought” it is a bag, as in handbag. Seems my color photograph was so different from the NMIST fashion training set, even transformed to grayscale 28 x 28 pixels using Pillow, that accurate classification was not possible. We do a bit better with some of the other photographs.
4. How did we get here? In two steps: adding code to the TensorFlow tutorial (Jupyter Notebook) to save the model and another to run or consume the model. This is a screen shot from the Jupyter notebook building the model.

```python
In [8]: model = keras.Sequential([  
        keras.layers.Flatten(input_shape=(28, 28)),  
        keras.layers.Dense(128, activation=tf.nn.relu),  
        keras.layers.Dense(10, activation=tf.nn.softmax)  
    ])  

WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\ops\resource_variable_ops.py:642: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.

In [9]: model.compile(optimizer='adam',  
        loss='sparse_categorical_crossentropy',  
        metrics=['accuracy'])

In [10]: model.fit(train_images, train_labels, epochs=5)
```

Epoch 1/5
60000/60000 [==============================] - 6s 102us/sample - loss: 0.4983 - acc: 0.8249
Epoch 2/5
60000/60000 [==============================] - 5s 82us/sample - loss: 0.3756 - acc: 0.8651
Epoch 3/5
60000/60000 [==============================] - 5s 82us/sample - loss: 0.3400 - acc: 0.8754
Epoch 4/5
60000/60000 [==============================] - 5s 82us/sample - loss: 0.3164 - acc: 0.8841
Epoch 5/5
60000/60000 [==============================] - 5s 82us/sample - loss: 0.2955 - acc: 0.8919
5. Line 11 is what I had added to save the model. You can ignore the error message “Unable to create link (name already exists).”

```python
try:
    keras.models.save_model(model=model,
                            filepath=r'<path>/fashion_model.h5',
                            overwrite=True,
                            include_optimizer=True)
except RuntimeError:
    print(sys.exc_info()[1])

Unable to create link (name already exists)
```

```python
In [12]:

    test_loss, test_acc = model.evaluate(test_images, test_labels)

    print('Test accuracy:', test_acc)

10000/10000 [==============================] - 0s 45us/sample - loss: 0.3397 - acc: 0.8787
Test accuracy: 0.8787
```

The WebFOCUS Procedure, aka Focexec or Fex

Note that changes were made to the style sheet in the tensorflow app folder, warm.sty, to support image display.

```plaintext
*- image_report

DEFINE FUNCTION get_prob(key/A15, prob_string/A255)
    token_index/I4 = DECODE key(
        'T-shirt/top' 2
        'Trouser'     4
        'Pullover'    6
        'Dress'       8
        'Coat'        10
        'Sandal'      12
        'Shirt'       14
        'Sneaker'     16
```

Python Advanced Analytics and Data Science Computing with WebFOCUS
Copyright © 2019 Information Builders
Page 43 of 58
prob_alpha/A20 = GETTOK(prob_string, 255, token_index, ' ', 20, 'A20');
get_prob/D17.15 = TO_NUMBER(prob_alpha) ;
END

TABLE FILE images_tensorflow
PRINT
  COMPUTE IMAGE_URL1/A100 = 'C:\IBI\APPS\TENSORFLOW\' || IMAGE_NAME ; NOPRINT
  COMPUTE IMAGE_URL2/A100 = 'C:\IBI\APPS\TENSORFLOW\' || IMAGE_NAME || '_grayscale.jpg' ; NOPRINT
  COMPUTE IMAGE_URL3/A100 = 'C:\IBI\APPS\TENSORFLOW\' || IMAGE_NAME || '_probability.jpg' ; NOPRINT

  COMPUTE Prob/A255 = PYTHON(predict_fashion, IMAGE_NAME, probability) ;
  NOPRINT

  COMPUTE t_shirt_prob/D17.15L = get_prob('T-shirt/top', Prob) ; OVER
  COMPUTE trouser_prob/D17.15L = get_prob('Trouser', Prob) ; OVER
  COMPUTE pullover_prob/D17.15L = get_prob('Pullover', Prob) ; OVER
  COMPUTE dress_prob/D17.15L = get_prob('Dress', Prob) ; OVER
  COMPUTE coat_prob/D17.15L = get_prob('Coat', Prob) ; OVER
  COMPUTE sandal_prob/D17.15L = get_prob('Sandal', Prob) ; OVER
  COMPUTE shirt_prob/D17.15L = get_prob('Shirt', Prob) ; OVER
  COMPUTE sneaker_prob/D17.15L = get_prob('Sneaker', Prob) ; OVER
  COMPUTE bag_prob/D17.15L = get_prob('Bag', Prob) ; OVER
  COMPUTE ankle_boot_prob/D17.15L = get_prob('Ankle boot', Prob) ; OVER

  BY IMAGE_NAME NOPRINT

  ON IMAGE_NAME SUBHEAD
  "<IMAGE_NAME"
  " " ** " " repeated 20 more times to provide space for images

  ON TABLE SET PAGE-NUM OFF
  ON TABLE SET STYLE Warm
END
The `COMPUTE` statement via the Python adapter accepts a single return value. But we evaluate each image for a match (classification) to each of the ten articles of clothing, footwear, and accessory (handbag). How do we overcome this limitation? But returning a record with all ten probabilities (wrapped example below) and then retrieving the values with a combination of WebFOCUS `DEFINE FUNCTION` and `COMPUTE`. The key is using `DECODE` to return the position of each probability value and `GETTOK` to review the value using the position.

"probability"

"T-shirt/top 0.488679289817810 Trouser 0.001886021811515 Pullover 0.055431481450796 Dress 0.273793578147888 Coat 0.005938739050180 Sandal 0.016042292118073 Shirt 0.138800099492073 Sneaker 0.000780666712672 Bag 0.016587505117059 Ankle boot 0.002060273196548"

"T-shirt/top 0.000848557800055 Trouser 0.897442460060120 Pullover 0.012338738888502 Dress 0.075209274888039 Coat 0.00466026103823 Sandal 0.000021230509446 Shirt 0.00010703733489 Sneaker 0.00000636701202 Bag 0.013547712936997 Ankle boot 0.000018389671823"

"T-shirt/top 0.001044584903866 Trouser 0.982053756713867 Pullover 0.00021075824810 Dress 0.016632648184896 Coat 0.000010000795555 Sandal 0.00001002734962 Shirt 0.000045244378271 Sneaker 0.0000000562312 Bag 0.00002065984290 Ankle boot 0.000011560675"

"T-shirt/top 0.342396706342697 Trouser 0.000240113266045 Pullover 0.013339115306735 Dress 0.01293902168982 Coat 0.00275962683722 Sandal 0.00008691117251 Shirt 0.625206232070923 Sneaker 0.00004837329470 Bag 0.014641292393208 Ankle boot 0.000031227307772"

**Creating the Synonym**

See our first example for instructions on creating the synonym for the Python script. If you want to rebuild the synonym (I recommend using a new name until you are familiar with the process), the files are:

Python script: `predict_fashion_v04.py`
Sample data file: `images.csv` Contains names of .jpg files to classify or process.
Starting point function: `predict_fashion`

**We Have Completed Our Lab!**

Thank you for attending our lab. We hope you enjoyed it as much as we did creating the content and sharing it with you – Ira and Tim.
Appendix

Configuring the WebFOCUS Adapter for Python
Refer to this manual and the instruction in the sections below:
WebFOCUS Adapter Administration
WebFOCUS Reporting Server Release 8205
DataMigrator Server Release 7709

Chapter 78, Using the Adapter for Python, page 1931


Python Adapter Requirements

The Adapter requires:

1. Python must be installed locally on same box as the Reporting Server (WFRS).
2. Only Windows, Linux and z/OS are currently supported.
3. On Windows, Linux install Python from Python Download (general download page).
4. On z/OS install Python from Rocket Software Downloads and follow instructions.
5. Python bit size (32-bit or 64-bit) must match the Reporting Server.
6. Only Python 3.6 is currently supported, below are specific minimum releases by platform.
7. Additional Python packages must be installed (steps below) before adapter use.
8. Server must be recycled (Workspace -> Server Actions -> Restart) between supplying Python installation directory and use of adapter test button or adapter features.

Python Adapter Configuration for Windows
(Prior to Reporting Server Start)

1. Minimum recommended Python 3.6.x release level to install is Python 3.6.5 (download page).
   1.1. If doing custom install, do not unselect pip option.
   1.2. If doing custom install, be sure to note the install location for later steps that refer to {pythoninstallpath}.
   1.3. Checking add to PATH option is not required, select only if you want this feature for other purposes.
   1.4. The default Windows install directory is displayed on the initial screen of the Python GUI installer and is referred to in this documentation as {pythoninstallpath}.
   1.5. If Python is already installed and the location is unknown, the default is the resolved value of %USERPROFILE%\appdata\local\programs\python\python36.
2. On the adapter configuration page supply the Python installation directory ({pythoninstallpath}) and use where indicated in the steps below.

   2.1. On Windows, the {pythoninstallpath} directory is specifically the directory containing the python.exe command line executable and python*.dll dlls.

3. Install (before adapter use) the following additional Python packages (in DOS Command Window) using:

   {pythoninstallpath}\python.exe -m pip install numpy scipy scikit-learn pandas

4. Confirm installed Python packages (in DOS Command Window) using:

   {pythoninstallpath}\python.exe -m pip list

5. Any additional Python environment sets as per the Python Documentation (such as PYTHONPATH) are allowed and may be set at the system level before server start or in server's edaenv.cfg file.

*Generally report servers are started as a service and service start requires system level variables to be used.

Python Adapter Configuration for Linux
(Prior to Reporting Server Start)

1. Minimum recommended Python 3.6.x release level to install is Python 3.6.3 (download page).
2. Note that Linux Python releases are typically either source build or "prepackaged" from the Linux vendor and as such require being built-installed by the system's administrator.
3. As of this writing, typically Linux systems ship with Python 2.x and as such system administrators will typically NOT want to disturb the default python version with a Python 3.6.x release so will install build/install to alternate locations not on the PATH such as /usr/local/python36 and is referred to in this documentation as {pythoninstallpath}.
4. Build steps are outside the scope of this document, see Python Documentation for build specifics.
5. On the adapter configuration page supply the Python installation directory ({pythoninstallpath}) and use where indicated in the steps below.
6. On UNIX/Linux {pythoninstallpath} directory is specifically the parent directory to the bin/python executable and the lib directory (or the lib64 depending on your python configuration) containing the .so Python DLLs (as well as other directories within the parent directory).
7. Install (before adapter use) the following additional Python packages using:

   {pythoninstallpath}/bin/python -m pip install numpy scipy scikit-learn pandas

8. Confirm installed Python packages using:

   {pythoninstallpath}/bin/python -m pip list

9. Any additional Python environment sets as per the Python Documentation (such as PYTHONPATH) are allowed and may be set before server start or in server's edaenv.cfg file.
Python Adapter Configuration for z/OS (Prior to Report Server Start)

1. Minimum recommended Python release level to install is Python 3.6.1 (download page).
2. Please follow instructions while installing.
3. Python installation directory, typically /usr/lpp/python36, is referred to in this documentation as `{pythoninstallpath}`.
4. On the adapter configuration page supply the Python installation directory `{pythoninstallpath}`.
5. If the server starts with Python configured, it automatically
   a. alters PATH and LIBPATH to make Python executables and libraries available; `{pythoninstallpath}`/bin is added at the end of PATH to avoid names collision with system utilities
   b. sets _BPX_AUTOCVT to ON
   c. sets FFI_LIB to `{pythoninstallpath}`/lib/ffi
   d. sets JOBLIB_MULTIPROCESSING to 0 to suppress Python warnings on multiprocessing on forced cpus = 1
6. As Python expects all text files to be in ASCII, all Python source and data files have to be correctly tagged in z/OS HFS (ZFS) system.
   All files produced by Python are tagged as ASCII and are automatically converted to EBCDIC when the server reads them.
   All files created by Python and the server are tagged automatically.
   All manually created files intended to be read by Python have to be tagged as EBCDIC with chtag command.
   Example: chtag -tc IBM-1047 my_file.py
   To verify tags run ls -T

Business Value

https://www.economist.com/leaders/2012/04/21/the-third-industrial-revolution
https://en.wikipedia.org/wiki/Information_Age

Python Tutorial
Tim recommends https://realpython.com/
Topics on basics, syntax, tricks can be found here - [https://realpython.com/tutorials/basics/](https://realpython.com/tutorials/basics/) Most of the articles are well written and free.

If one is interested in advanced topics or tutorials on specific packages, they can become a paid member of the site and gain access to all the videos. This site provides a pretty comprehensive guide to Python.

**A Little Python Humor**

From the Python interactive prompt, type: `import antigravity [Enter]`

[https://xkcd.com/353/](https://xkcd.com/353/)
Python Integrated Environments (IDE)
https://www.jetbrains.com/pycharm/
https://wiki.python.org/moin/IntegratedDevelopmentEnvironments
https://realpython.com/python-ides-code-editors-guide/

Running Anaconda 3 and Python.org Distributions on the Same PC
https://www.anaconda.com
https://www.python.org

Currently (I write this on May 30th, 2019) the Python adapter works with Python 3.6.5 - 3.6.8 from Python.org. We are working on compatibility with the Anaconda distribution (a pathing issue) and upgrading the adapter to Python 3.7.

A programmer typically will have one Python distribution installed on a computer. And using more than one release of the same distribution in virtual environments (on the same computer) is not uncommon.

Can the Anaconda and Python.org distributions coexist on the same PC? Tentatively, yes. The workaround described below assumes Anaconda was installed first and Python.org installed next, simply for the purpose of using the WebFOCUS Python adapter. Anaconda for development work, Python.org for the Python adapter.

1. Install Anaconda first.
2. Follow the instructions in the next section, "Installing Python 3.6.8", to install Python 3.6.8 from Python.org
3. After installing Python 3.6.8 from Python.org, add to the Windows Path if not already set:
   C:\Python\Python36
   C:\Python\Python36\Scripts
   or use the path set during installation
4. Temporarily disable the Anaconda entries in the Windows path by prefixed the entries with XXXX_:  
XXXX_C:\ProgramData\Anaconda3  
XXXX_C:\ProgramData\Anaconda3\Scripts

5. Open a Windows Command Prompt window as Administrator

6. Run Python to confirm the Python.org (plain) distribution comes up and not Anaconda. Type Ctrl+Z to exit Python but do not close the command prompt window
7. Use pip to install scipy, scikit-learn, numpy, and pandas for the Python.org distribution.

8. Configure the WebFOCUS Python adapter and set the Python installation directory. Run Test after clicking Configure. You will need to reopen the adapter's Properties dialog by right-clicking the adapter icon.

9. Restore the Windows Path entries for Anaconda.
Installing Python 3.6.8

https://www.python.org/

1. Select Downloads > All releases

![Python Downloads](https://www.python.org/downloads/)

2. Scroll down to Python 3.6.8 and click it.
3. Scroll down to and click "Windows x86-64 executable installer"

4. Navigate to your Downloads folder, right-click python-3.6.8-amd64.exe and select "Run as administrator"
5. Check "Add Python 3.6 to PATH", click "Customize installation."

6. Accept default settings, click "Next"
7. These are my preferences. In my work I often have both Python 2.7 and 3.x installed, so I create a parent Python directory. Make changes appropriate for your work environment. Click Next.

8. Wait while Python is installed. Click "Close" when done.
9. Disable the path length limit if desired. Click "Close." The online tutorial and documentation are quite good.
Socialize to Win.
Daily Prizes Awarded!

Don’t forget to tag #IBSummit in your posts on Twitter and Instagram – you just might make it on the Leaderboard!

Check our social channels for contests, updates, photos, and announcements.