Python Training

Labs

CLARITY INSIGHTS

Lab 1 – Basic Functions

- Create a function which creates a random integer value between `lowValue` and `highValue` parameters.
  - If no value is provided, the integer should be between 0 and 100 inclusive.
  - hint: the function randint in the random module will be helpful. randint takes two parameters (a, b) and returns a random integer N such that a <= N <= b
- Using mathematical operators, write a function that calculates the area of a triangle, given a base and height. Default base and height should be 10
  - Example: passing 2 as base and 5 as height should return 5

Lab 2 – Control Structures

- Write a program which calculates a student’s letter grade based on their score. Default score should be 100.
- Use the following logic. if …
  - score is greater than or equal to 90, then it is an A
  - score is greater than or equal to 80, then it is a B
  - score is greater than or equal to 70, then it is a C
  - score is greater than or equal to 60, then it is a D
  - else for all other score, it is an F

Lab 3 – Error Handling

- Enhance the divideTwoNumbers(num, denom) function to handle the Exception when num or denom is a string. Handle that case by returning the string “Cannot divide strings”
  - hint: the Exception you want to catch is TypeError

Lab 4 – Debugging

- For this lab, refer to the following script:

```
import sys
try:
 print("Argument 1: " + sys.argv[1])
 print("Argument 2: " + sys.argv[2])
 arg1 = int(sys.argv[1])
 arg2 = int(sys.argv[2])
except TypeError:
 print("Error: Cannot perform addition on non-numeric values" )
 sys.exit(1)
except IndexError:
 print("Error: This script requires two arguments")
 sys.exit(1)
else:
 average = (arg1 + arg2) / len(sys.argv[1:])
 print("Average   : " + str(average))
```

Lab 4 – Debugging

- This script appears to be working when testing the ideal scenario (2 arguments passed), but what happens if it is used improperly?
- Is this script catching enough common error types? What error types can you experience in testing?
- How can you correct this script such that it averages numbers correctly?

Lab 5 – Create Your Own Class

- Take 15 minutes to create a class in Python.
- Remember to include properties and methods.
- Then, take 5 minutes to share your class with your classmates.
- If you defined your class well, your classmates will be able to create and do things with objects of your class.

Lab 6 – Literate Programming

- Make the following more readable with literate programming:

```
class aClassName():
      def __init__(self):
            return
      def method1(self, arg1, arg2):
            return arg1 + arg2
      def method2(self, arg1, arg2):
            return arg1 - arg2
      def method3(self, arg1, arg2):
            return arg1 * arg2
      def method4(self, arg1, arg2):
            return arg1 / arg2
if __name__ == "__main__":
      object = aClassName()
      print(object.method1(1,2))
      print(object.method2(1,2))
      print(object.method3(1,2))
      print(object.method4(1,2))
```

Hint: What object do you know of that performs mathematical computations such as +, -, *, and /?

Lab 7 – Problem Solving Approach

- As a class, solve the following problem using the problem solving approach discussed in the lecture slides:
    - Write a function that takes in a string and returns the same string but in all uppercase.

Lab 8 – Working With Lists 1

- Write a function that returns the number of items in a list.

Lab 9 – Working With Lists 2

- Write a function that determines if a list contains duplicate elements.

Lab 10 – Working with Dictionary

- Given this list of student names:

```
      ["morgan", "david", "jeff", "david",
"clara", "morgan", "david"]
```

- Write a function that prints out how many times a name occurs in the list

Lab 11 – Working With Sets

- Create and output a set, exam_grades_under_50, containing the prime numbers under 50.
- Check for correctness as follows:

```
        >>> print(exam_grades_under_50)

        >>> {2, 3, 5, 37, 7, 41, 11, 43, 13, 47,
17, 19, 23, 29, 31}
```

Lab 11 – Working With Sets

- Create and output a set, exam_grade_under_20, containing the prime numbers under 20, derived from the set in step (1).
- Check for correctness as follows:

```
>>> print(exam_grade_under_20)

>>> {2, 3, 5, 7, 11, 13, 17, 19}
```

Lab 11 – Working With Sets

- Create and output a set, exam_grades_under_50_over_20, containing the prime numbers under 50 and over 20, by utilizing the sets from steps (1) and (2).
- Check for correctness as follows:

```
>>> print(exam_grades_under_50_over_20)
>>> {37, 41, 43, 47, 23, 29, 31}
```

Lab 12 – Working With Tuples

- Create a tuple, `dean_metadata_shermer_il`, containing three values: the integer `1986`, the string `Dean`, and the string `Rooney`.
- Check for correctness as follows:

```
>>> print(dean_metadata_shermer_il)
>>> (1986, 'Dean', 'Rooney')
```

Lab 12 – Working With Tuples

- Create a tuple, `dean_metadata_faber_college`, containing three values: the integer `1978`, the string `Dean`, and the string `Wormer`.
- Check for correctness as follows:

```
>>> print(dean_metadata_faber_college)
>>> (1978, 'Dean', 'Wormer')
```

Lab 12 – Working With Tuples

- Create a tuple, format, containing three values: the strings `year`, `role`, and `name`.
- Check for correctness as follows:

```
>>> print(format)
>>> ('year', 'role', 'name')
```

Lab 12 – Working With Tuples

- Using the data from steps (1) and (2), and utilizing the order information available from step (3), find and output the years and the names of the dean data, omitting the role data.
- Check for correctness by checking that output matches the information in the following lines:

```
1986 Rooney
1978 Wormer
```

Lab 13 – Working With Files

- Read in the contents of the file `my_hidden.txt` and replace all occurrences of the word "`hidden`" with your favorite subject in school. Write the results to a new file called `my_favorite_subject.txt`.
- Hint: There is a string method called replace that you may find helpful in this lab.

Lab 13 – Working With Files

- Contents of myhidden.txt (3 lines):

```
My favorite subject is hidden. There is nothing I love more than to come home to
a wonderful book about hidden. Why do I love hidden so much? It's just the most interesting
subject ever in my opinion.
```

Lab 14 – Working with Databases

Using the same SQLite installation, perform the following tasks. Write your code in a Python file and run it.

1) create a database called inventory
2) create a table called fruits with the following fields: name varchar(100), quantity integer, cost double
3) insert the following records into the fruit table:
   a) "apple", 20, 1.01
   b) "oranges", 100, 2.01
   c) "bananas", 0, 1.50
4) update the fruit table so that there are 0 apples
5) delete from the fruit table all fruits that are not in our inventory (quantity equals 0)
6) save your changes and close the connection

Lab 15 – Working with JSON and CSV

Given the following JSON string, parse it and write the data records to a csv file called fruits.csv.

```
json_string_fruits = “””
{
        “total”: 3
        , “data”: [
                  { “id”: 1, “name”: “grapes”, “quantity”: 100, “cost”: 1.01 }
                , { “id”: 2, “name”: “strawberries”, “quantity”: 50, “cost”: 3.71 }
                , { “id”: 3, “name”: “lemon”, “quantity”: 10, “cost”: 2.17 }
        ]
}
“””
```

Lab 16 – Regular Expressions

- What is the regex pattern to match the following:
  - Email (such as `name@email.com`)
  - Email with `.com`, `.edu` or `.gov` address
  - Email with no numbers allowed in mailbox name

Lab 16 – Regular Expressions

- Write a function which tests if a password is >= 8 characters and meets the at least 3 of the following requirements:
  - Has 1+ English uppercase alphabet character (A–Z)
  - Has 1+ English lowercase alphabet character (a–z)
  - Has 1+ Base-10 digits (0–9)
  - Has 1+ Non-alphanumeric characters (for example, !$#,%)
- This function should accept one parameter.

Lab 17 – Pythonic ETL & Apache Airflow

- Note: This is an advanced topic. Apache Airflow is interesting because it is *very similar* to Dataswarm, the Python ETL engine used at Facebook.
- As an alternative to doing this lab with the `airflow` module, consider using Python without `airflow` to create the reports requested by this lab later on:
- Optionally replace “airflow pipeline” with “Python script” later on in the deliverable sentence, “create an airflow pipeline that reads from the provided database and reports the following”

Lab 17 – Pythonic ETL & Apache Airflow

- Install Python and the Apache Airflow module if they are not already installed.
  - Install openssl if needed, e.g. per https://github.com/saghul/pythonz
  - Install pythonz if needed for easy Python version management, e.g. per https://github.com/saghul/pythonz

CLARITY INSIGHTS

Lab 17 – Pythonic ETL & Apache Airflow

- Sample python environment setup:

```
$ pythonz install 2.7.13
$ cd ~/Documents/clarity/python_training/
$ virtualenv -p $(pythonz locate 2.7.13) airflow_project
$ source airflow_project/bin/activate
```

- If needed, install dependencies so that `airflow` can be installed, e.g. with `'sudo pip install six'`, etc.

Lab 17 – Pythonic ETL & Apache Airflow

- Get Apache Airflow set up and running.
  - Initialize the db and start the airflow webserver as in in http://pythonhosted.org/airflow/start.html?highlight=sqlite
  - Create a home directory for airflow:

```
$ mkdir ~/airflow
$ airflow initdb
$ airflow webserver -p 8080
```

  - See the airflow server message on the screen. Keep this process running.

Lab 17 – Pythonic ETL & Apache Airflow

- See the airflow web UI at http://localhost:8080/admin/
- Review the airflow configuration:

```
$ vim ~/airflow/airflow.cfg
```

- See in `~/airflow/airflow.cfg:`

```
dags_folder = /Users/benfranklin/airflow/dags
```

- Set the following in ~/airflow/airflow.cfg to clear out the examples if desired:

```
load_examples = False
```

- See the DAGs list is empty :

```
airflow list_dags
```

Lab 17 – Pythonic ETL & Apache Airflow

- For basic familiarity, create a pipeline using operators
  - Read about DAGs, Operators, Tasks, Dependencies at http://airflow.readthedocs.io/en/latest/concepts.html and http://pythonhosted.org/airflow/tutorial.html
  - Read about setting default task parameters at http://pythonhosted.org/airflow/tutorial.html#default-arguments

Lab 17 – Pythonic ETL & Apache Airflow

- Operators suggested for use in an initial mini project including the following:
    - BashOperator
    - PythonOperator
    - SqliteOperator

Lab 17 – Pythonic ETL & Apache Airflow

- Run your pipeline from the command line.
  - Read about the airflow cli commands at http://airflow.readthedocs.io/en/latest/cli.html
- Observe your pipeline's execution history in the web UI.
  - See the airflow web UI at http://localhost:8080/admin/

Lab 17 – Pythonic ETL & Apache Airflow

- Mini Project
  - A sample SQLite database file, `working_with_airflow_lab.db`, is located at https://drive.google.com/open?id=0Bx4ift37FbCWRmt4Vjh0WlFBUVE

Lab 17 – Pythonic ETL & Apache Airflow

- The mini project db contains the following tables:

```
$ sqlite3 working_with_airflow_lab.db

sqlite> .tables

course_instructors    courses                     students
course_registrations  professors
```

- Configure your instance of airflow to connect to the provided SQLite database file.

Lab 17 – Pythonic ETL & Apache Airflow

- Create an airflow pipeline that reads from the provided database and reports the following:
  - List names of students not registered for any courses.
    - Target table name: `unregistered_students`
  - List names of students registered for each course name.
    - Target table name: `rollcall_by_course`
  - List groups of students who have with 3 or more courses together.
    - Target table name: `cohort_groups`
  - List names of students enrolled in 300 level or higher lasses who are also enrolled in 100 or 200 level courses.
    - Target table name: `accelerated_students`

Lab 17 – Pythonic ETL & Apache Airflow

- The pipeline is required to create tables containing the reported data, and to refresh the tables with update data when the pipeline is executed.
- The pipeline is also required to produce text files which report the data contained in the resulting tables.

Lab 17 – Pythonic ETL & Apache Airflow

- Extensions
    - Set up MySQL, Postgres, and Hive on your system, and load some data into tables. Then, test out the following additional operators:
        - MySqlOperator
        - PostgresOperator
        - HivePartitionSensor

Lab 17 – Pythonic ETL & Apache Airflow

- Gain practical experience with the following:
  - Using the "`airflow list_tasks <task_name> --tree`" command to review a pipeline's DAG at the command line
  - Using the "`airflow test`" command to test pipelines
  - Using the "`airflow backfill <task_name> -s <start_date> -e <end_date>`" command to backfill pipeline runs for prior date's schedules

End of Labs