**CSCI 4333.1**

**CSCI 4333 and DASC 5333 DCM Web Hosting**

Fall 2024

Dear CSCI 4333 and DASC 5333 students,

Your Web-Python assignment needs to be uploaded to the DCM server for grading.

[1] A directory has been created for each student in DCM: fa24<<course>>s<<section in one digit>><<full last name>><<first character of first name>>. For examples:

|  |  |
| --- | --- |
| **Student** | **DCM directory for the student** |
| Bun Yue in CSCI 4333 Section 1 | fa24c4333s1yueb |
| Jane Jacob in CSCI 4333 Section 2 | fa24c4333s2jacobj |
| Mounika Patel in DASC 5333 Section 1 | fa24d5333s1patelm |

[2] Use FTP to upload the assignment (h6.py) to your directory.

* FTP server: dcm.uhcl.edu
* Login username and password: Your PCLAB account credential.

[3] Your file(s) (e.g., h6.py) should be uploaded to the “pages” folder of your DCM directory, e.g., fa24c4333s1yueb/pages. It can then be accessed using the URL: http://dcm.uhcl.edu/fa24c4333s1yueb /h6.py.

[4] A MySQL guest account ‘dbguest’ (with the password ‘uhcl\_\_dbguest’) has been created with read privileges to the swim and toyu databases. Thus, your uploaded Python program should use the following credentials:

|  |
| --- |
| Host: localhost user: dbguest password: uhcl\_\_dbguest database: swim |

[5] If you have any questions about your DCM account, you may contact the student systems administrators: Charan Gedipudi or Harini Navari, Email: [gedipudic1346@uhcl.edu](mailto:gedipudic1346@uhcl.edu)

Email: [navarih8507@uhcl.edu](mailto:navarih8507@uhcl.edu). Please copy your email to me and the faculty system administrator (Abeysekera, Krishani at <Abeysekera@uhcl.edu>)

[6] For questions about the homework, you may ask my TA (Pavan Kodavali at [kodavali@uhcl.edu](mailto:kodavali@uhcl.edu)). Do not send questions about the assignments or the course materials to the systems administrators.

[7] For additional information about the DCM server, see: <http://sce.uhcl.edu/support/>.

**10/28/2024**

**An Introduction to Python**

by K. Yue

**1. Resources**

* Python: <https://www.python.org/>
* Version 3.11: (navigate to your version).
  + Manual: <http://docs.python.org/3.11/reference/index.html>
  + Tutorial: <https://docs.python.org/3.11/tutorial/interpreter.html>
  + Standard library: <https://docs.python.org/3.11/library/index.html>
* Installation notes:
  + It is recommended that you should install Python*using customization*:
    1. Put Python in the *top level* of your C drive. For example: in c:\Python311\.
    2. Select to let the installer set Python path environment variable.
    3. Select to let the installer disable Windows path length limit (if not, limit is 260 characters). This option is shown after the successful installation screen.
* Python's Beginner Guide for programmers: <https://wiki.python.org/moin/BeginnersGuide/Programmers>
* Style Guide for Python Code:
  + <https://www.python.org/dev/peps/pep-0008/> (dated but still containing many useful tips).
  + [Google Python Guideline](https://google.github.io/styleguide/pyguide.html)
* For experienced programmers: <https://learnxinyminutes.com/docs/python3/>
* For all: <https://www.w3schools.com/python/>

**2. Basics**

* The course assumes that you are reasonably comfortable with Python for program development.
* Python is a high-level, open source, general-purpose, object-oriented, extensible, interpreted scripting programming language.
* Python is a scripting language:
  1. Especially good for scripting and rapid application development
  2. Object-oriented language (more so than many other OO languages)
  3. Dynamic typing: an object bound to a variable can change type dynamically.
  4. Strongly typed: less implicit type conversion.
  5. Native high level data structures: list, tuple, set, dictionary, etc.
  6. Interpreted
  7. Automatic garbage collection
  8. Open source
  9. Use good software engineering principles.
* Using the Python's Integrated Development and Learning Environment (IDLE) interpreter is a good way to learn Python.
  1. Use Control-z or quit() to exit Python interpreter.

**2.1 Python programs: some basic concepts**

* A Python program contains *logical lines*.
* A logical line usually contains one *physical line* but can be extended by '\' to contain many physical lines.
* Implicit line joining is performed for (), [] and {} expressions. It is preferred to using '\'.
* Do not use ';' as a statement separator.
* Indentation (white spaces at the beginning of lines) has meanings. Proper indentations are mandatory.
* *Indentation level* is used to group statements.
* Python 2: suggest to not mix tabs and spaces.
* Python 3:
  + Disallow mixing of tabs and spaces.
  + Spaces are preferred.
* Comments start with a '#'. You may also use a multiple line string, such as ''' a comment. '''
* ...

**2.2 Running Python in Windows**

* To start the Python interpreter, open a command line prompt in your working directory and execute, for example:

set path=%path%;C:\Python311  
python

* You may need to replace "C:\Python311" by the location of your python installation.
* You may set the environmental variable PATH in your OS. If so, you will not need to execute the set path command.
* To run a Python program, helloworld.python, use:

python helloworld.py

**2.3 Concepts to learn in a new language**

1. syntax
2. concepts that are familiar to you: note any difference.
3. concepts that are new to you
4. resources and libraries
5. design patterns
6. best practices

**2.4 Some basics to get you interested (hopefully):**

* No ;
* No i++
* Multiple assignment: a,b = b,a
* strings are automatically concatenated: print("ab" "cd")
* strings are automatically indexed:
  + a='12345'
  + print(a[1:3])
* id() to find out id of an object.
* Python does not support the traditional array data type. Instead, lists can be viewed as a more general array type.
* ...

**2.5 Examples of concepts that may be new to you:**

* Immutable or mutable?
  + The *value* of an immutable object cannot be changed.
  + Use the id() function for experimentation.
  + Some data types require immutable objects. E.g., member of a set and key of a dict.
* Automatic garbage collection: "Objects are never explicitly destroyed; however, when they become unreachable they may be garbage-collected."
* Built-in Python data structures: list, set, tuple and dictionary.

***Example:***

Print first 100 Fibonacci numbers: [fib.py.txt](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/notes/emSQL/fib.py.txt) (remove .txt when saving)

#   print first 100 Fibonacci numbers  
a, b, count = 0, 1, 1  
while count <= 100:  
   print(b, ' ', end='')  
   a, b, count = b, a+b, count+1  
print()

Note the use of multiple assignment statements. In other languages, such as Java, you may need to replace:

a, b, count = b, a+b, count+1

by

temp=b;  
b=a+b;  
a=temp;  
count++;

Other examples will be discussed in the class.

**3. A Non-trivial Example**

***Example:***

Consider the following weather information file: [201401daily\_sample.csv](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/notes/emSQL/201401daily_sample.csv). Write a Python program to read the 'codeSum' column (#23) and output the count of each codeSum. Note that the codeSum column may contain more than one codeSum separated by white spaces. Example:

SN FG+ FZFG BR UP

has five codeSum:

* SN
* FG+
* FZFG
* BR
* UP

[codeSum.py](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/notes/emSQL/codeSum.py.txt):

import sys, getopt  
import re  
from operator import itemgetter  
  
#   getopt: C-style parser for command line options.  
#   sys: System-specific parameters and functions.  
#   re: regular expression operation  
#   operator: methods for built-in operators.  
#       (useful when passing the function as a parameter.)  
  
#   Read and process a weather information file.  
#   It parse the codeSum column (#23) in the CSV file  
#   and show the count of each codeSum.  
  
f = open(sys.argv[1], 'r')  
result = [];  
heading = f.readline().split(',')  
num\_line = 0;  
  
for line in f:  
    line = line.rstrip()    #   strip trailing white spaces  
    #   result is an array of arrays:  
    #   Add the array contains columns of the current line  
    #   to result.  
    result.append(line.split(','))  
    num\_line = num\_line + 1  
f.close()  
  
#   Debug:  
#   for i in range(len(result)):  
#       print (str(i) + ":" + str(result[i][22]))  
  
#   count is a dictionary with the key being the individual CodeSummary  
count = {};  
for i in range(len(result)):  
    #   process one reading.  
    line = result[i][22].strip()  
    if line:  
        #   Get all codeSummary and update their counts.  
        summary = re.split('\s+', line)  
        for j in range(len(summary)):  
            #   Debug:  
            #   print (str(j) + ":" + str(summary[j]) + "---")  
            if summary[j] in count.keys():  
                count[summary[j]] += 1  
            else:  
                count[summary[j]] = 1  
  
#   Print result in the sorted order of codeSum.                 
for key, value in sorted(count.items(), key=itemgetter(0)):  
    print (key + ": " + str(value))  
  
             
Running the program:

>...python codeSum.py 201401daily\_sample.csv  
BR: 18  
FG: 2  
FG+: 5  
FZFG: 8  
HZ: 10  
RA: 2  
SN: 14  
UP: 4  
  
Version 2: more Python-style with some interesting Python features:

[codeSum2.py](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/notes/emSQL/codeSum2.py.txt):

from collections import defaultdict  
import sys, getopt  
import re  
from operator import itemgetter  
  
#   Read and process a weather information file.  
#   It parse the codeSum column (#23) in the CSV file  
#   and show the count of each codeSum.  
  
f = open(sys.argv[1], 'r')  
heading = f.readline().split(',')  
result = [line.strip().split(',') for line in f.readlines()]  
f.close()  
  
#   count is a dictionary with the key being the individual CodeSummary  
count = defaultdict(int)  
for field in filter(lambda a: a, map(lambda w: w[22].strip(), result)):  
   for sym in re.split('\s+', field):  
      count[sym] += 1  
    
#   Print result in the sorted order of codeSum.                 
for key, value in sorted(count.items(), key=itemgetter(0)):  
    print (key + ": " + str(value))

**4. Some useful tools**

**4.1 Logging**

* Python logging: <https://docs.python.org/3/howto/logging.html>
* "Logging is a means of tracking events that happen when some software runs. The software’s developer adds logging calls to their code to indicate that certain events have occurred."
* Logging can be useful in debugging programs.

***Example:***

[logExample.py.txt](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/notes/emSQL/logExample.py.txt):

import logging  
  
# logging  
logLevel = logging.DEBUG  
logPath = "."  
  
logger = logging.getLogger()  
logger.setLevel(logLevel)  
  
logFormatter = logging.Formatter("%(asctime)s::%(threadName)-12.12s::[%(levelname)-5.5s] %(message)s")  
  
consoleHandler = logging.StreamHandler()  
consoleHandler.setFormatter(logFormatter)  
logger.addHandler(consoleHandler)  
  
logger.debug("Program starts.")  
logger.debug("Program continues.")

**4.2 Jupyter Notebook (especially for DASC 5333)**

* <https://jupyter.org/>: "The Jupyter Notebook is the original web application for creating and sharing computational documents."
* Installation and launching:
  + Jupyter Notebook:
    - Installation: using pip: "pip install notebook"
    - Launching server: open a terminal in the work directory and execute the command: "jupyter notebook".
    - Stopping server: [1] crtl-c, or [2] open anther command terminal, and execute: "jupyter notebook stop 8888", where 8888 is the (default) port of the server.
  + JupyterLab Notebook: next generation notebook.
    - Installation: using pip: "pip install jupyterlab"
    - Launching: open a terminal in the work directory and execute the command "jupyter-lab".
    - Stopping server: crtl-c
* If there is time to cover this topic, we will use JupyterLab, which is preferred.

***Example:***

Try out: [CodeSumNB.zip](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/notes/emSQL/CodeSumNB.zip). Unzipp to get CodeSumNB.ipynb. Put it in your notebook working directory and open it with juypter-lab.

**MySQL Programming in Python**

by K. Yue

**1. MySQL Drivers for Python**

* There are many MySQL drivers for Python. Some examples:
  + MySQLDB Python: <http://sourceforge.net/projects/mysql-python/>
  + MySQL Connector Python: <https://dev.mysql.com/downloads/connector/python/>: written with pure Python.
  + MySQL Connector Python's Developer Guides: <http://dev.mysql.com/doc/connector-python/en/index.html>
* Selecting the right driver is important and not straightforward. Examples:
  + An example comparing driver performance: <https://gist.github.com/methane/90ec97dda7fa9c7c4ef1>
  + PyMySQL evaluation: <https://wiki.openstack.org/wiki/PyMySQL_evaluation>
* Because of some problems of MySQL Connector in Python 3, we use*PyMySQL.*

**2. PyMySQL:**

* Installation:
  + Install pip (Package Installer for Python), if necessary: <https://pip.pypa.io/en/stable/installing/>.
  + Command: "pip install PyMySQL"
* Documentation: <http://pymysql.readthedocs.io/en/latest/>
* Compliant with Python Database API Specification v2.0: <https://www.python.org/dev/peps/pep-0249/#module-interface>.

|  |
| --- |
| **Additional Materials: MySQL Connector Python (*not* used in this course, but it is a good resource)**   * API reference: <http://dev.mysql.com/doc/connector-python/en/connector-python-reference.html> * Read MySQL Connector Python's Guideline for developer: [http://dev.mysql.com/doc/connector-python/en/connector-python-coding.htm](http://dev.mysql.com/doc/connector-python/en/connector-python-coding.html) * Examples of good tips and insight:   1. "Use config.py module to store database connection information."   2. "Any application that accepts input must expect to handle bad data."   3. "Data that you choose to store in MySQL instead is likely to have special characteristics."   4. "you can use Python's triple-quoting mechanism to enclose the entire statement."   5. "Oracle recommends the ENGINE=INNODB clause for most tables, and makes InnoDB the default storage engine in MySQL 5.5 and up." |

**3. Python Programming with MySQL**

* Embedded SQL: embedding SQL in a host language.
* SQL is a DB language, not a general purpose language.
* Host languages:
  1. general purpose languages
  2. usually have vast libraries and tools.
  3. usually have a good community in a specific application area.
* Examples of popular host languages: Java, JavaScript, ts, GOLANG, C#, C, C++, PHP, Perl, Python, Ruby, etc.
* Preparation: create a MySQL account in your computer for development if you have not already done so. (You may use PHPMyAdmin or MySQL Workbench)

**3.1 Basic steps**

* Basic steps of embedded SQL programming.
  1. Create connections to the database.
  2. Prepare and execute SQL statements in the connections by using a cursor.
  3. Use the results of SQL statements.
  4. Perform house cleaning and close connections.
* Step (2) => learn your SQL well.
* Some issues in step (3):
  1. Data type mismatch (impedance mismatch): relations/tables are usually not supported natively in the host languages.
  2. Performance consideration: DB operations are usually much more expensive (take longer time) than CPU operations. Examples:
     1. Minimize DB connections, especially active ones.
     2. Optimize the numbers of SQL statements and their efficiencies.
  3. Security consideration: DB breaches are serious.

**3.2 Connection and Cursor Objects**

* To solve the data type mismatch problem, the concept of *cursor* is usually used.
* A cursor allows the programmer to *iterate* through the result set, one row at a time.
* Cursor reference of PyMySQL: <http://pymysql.readthedocs.io/en/latest/modules/cursors.html>
* There may be different cursor types for performance, security, and access method considerations.
* PyMySQL's cursor types:
  1. SSCursor: unbuffered cursor
  2. DictCursor: return a dictionary
  3. SDictCursor: an unbuffered cursor returning a dictionary

***Example:***

[student1.py](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/notes/emSQL/student1.py.txt): (All .py and .ini files have an extra extension, .txt, which should be removed when downloading).

import pymysql  
  
# [1] Making connection to the MySQL server  
cnx = pymysql.connect(user='demo', password='abcdef',  
    host='localhost',  
    database='toyu')  
  
# Create a cursor using the connection.  
cursor = cnx.cursor()  
  
# [2] Prepare a SQL query for the problem  
query = '''  
SELECT CONCAT (s.fname, ' ', s.lname) AS student,  
    d.deptName,  
    CONCAT(f.fname, ' ', f.lname) as advisor  
FROM student AS s LEFT JOIN department AS d  
        ON (s.major = d.deptCode)  
     LEFT JOIN faculty AS f  
        ON (s.advisor = f.facId);  
'''  
  
# Execute the query  
cursor.execute(query)  
  
# [3] Use the result in the query  
for (student, major, advisor) in cursor:  
    print("{}: major={}; advisor={}".format(student, major, advisor))  
  
# [4] Housekeeping  
cursor.close()  
cnx.close()

The output of running this program:

C:\...>python student1.py  
Tony Hawk: major=Computer Science; advisor=Paul Smith  
Mary Hawk: major=Computer Science; advisor=Paul Smith  
David Hawk: major=Computer Science; advisor=Mary Tran  
Catherine Lim: major=Information Technology; advisor=None  
Larry Johnson: major=Information Technology; advisor=Deborah Gump  
Linda Johnson: major=Computer Information Systems; advisor=Daniel Kim  
Lillian Johnson: major=Computer Information Systems; advisor=Andrew Byre  
Ben Zico: major=None; advisor=None  
Bill Ching: major=Arts; advisor=None  
Linda King: major=Arts; advisor=Art Allister  
Cathy Johanson: major=None; advisor=Art Allister

* Some tips:
  1. Test your SQL statements thoroughly first.
  2. Use typical input parameters for testing.
  3. Be mindful of:
     1. security, especially SQL injection
     2. special characters of the languages involved
     3. performance

**3.3 Configuration files**

* It is better to use configuration module and data ini file (e.g. dbconfig.py and dbconfig.ini). Why?
  1. Separation of data and program
  2. Reuse and maintenance
  3. Security
* In general, using a configuration data file is a good programming practice.

[dbconfig.py](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/notes/emSQL/dbconfig.py.txt):

import configparser  
  
#  simplistic and no error handling.  
def get\_mysql\_param(filename='dbconfig.ini', section='mysql'):  
  
    config = configparser.ConfigParser()  
    config.read(filename)  
  
    return config[section]

If running in Windows IIS (such as in the dcm server):

import configparser  
from pathlib import Path  
  
#  simplistic and no error handling.  
def get\_mysql\_param(filename='dbconfig.ini', section='mysql'):  
    config = configparser.ConfigParser()  
    file\_path = (Path(\_\_file\_\_).parent / filename).resolve()  
    config.read(file\_path)  
  
    return config[section]

[dbconfig.ini](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/notes/emSQL/dbconfig.ini.txt):

[mysql]  
host = localhost  
port = 3306  
database = toyu  
user = your\_account  
password = your\_password

In your Python program:

from dbconfig import \*  
import pymysql  
  
db = get\_mysql\_param()  
cnx = pymysql.connect(user=db['user'],  
      password=db['password'],  
      host=db['host'],  
      database=db['database'],  
      port=int(db['port']))  
cursor = cnx.cursor()

If you use the default port (i.e., 3306), you may adjust by

1. remove the line "port=3306" in dbconfig.ini, and
2. remove the argument "port" in the call of pymysql.connect() in the main program.

In your Python program,

[student2.py](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/notes/emSQL/student2.py.txt):

from dbconfig import \*  
import pymysql  
  
db = get\_mysql\_param()  
  
print (db)  
cnx = pymysql.connect(user=db['user'], password=db['password'],  
      host=db['host'],  
      database=db['database'],  
      port=int(db['port']))  
  
cursor = cnx.cursor()  
  
# [2] Prepare a SQL query for the problem  
query = '''  
SELECT CONCAT (s.fname, ' ', s.lname) AS student,  
    d.deptName,  
    CONCAT(f.fname, ' ', f.lname) as advisor  
FROM student AS s LEFT JOIN department AS d  
        ON (s.major = d.deptCode)  
     LEFT JOIN faculty AS f  
        ON (s.advisor = f.facId);  
'''  
  
# Execute the query  
cursor.execute(query)  
  
# [3] Use the result in the query  
for (student, major, advisor) in cursor:  
    print(f"{student}: major={major}; advisor={advisor}")  
  
# [4] Housekeeping  
cursor.close()  
cnx.close()

* See Format Specification Mini-Language: <https://docs.python.org/3.11/library/string.html#formatspec>
* Your SQL statement may use placeholder parameters: %s.
* Parametrized statements are preferred.

Example from <http://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor-execute.html>:

insert\_stmt = (  
  "INSERT INTO employees (emp\_no, first\_name, last\_name, hire\_date) "  
  "VALUES (%s, %s, %s, %s)"  
)  
data = (2, 'Jane', 'Doe', datetime.date(2012, 3, 23))  
cursor.execute(insert\_stmt, data)  
  
select\_stmt = "SELECT \* FROM employees WHERE emp\_no = %(emp\_no)s"  
cursor.execute(select\_stmt, {'emp\_no': 2})

***Example:***

Get the school code as the command line argument.

[student3.py](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/notes/emSQL/student3.py.txt)

from dbconfig import \*  
import pymysql  
import sys  
  
# Use the school code in command line arguments  
# to list all students majoring in a department  
# in the school.  
# [1] Making connection to the MySQL server  
  
db = get\_mysql\_param()  
cnx = pymysql.connect(user=db['user'], password=db['password'],  
      host=db['host'],  
      database=db['database'],  
      port=int(db['port']))  
cursor = cnx.cursor()  
  
# [2] Prepare a SQL query for the problem  
query = '''  
SELECT CONCAT (s.fname, ' ', s.lname) AS student,  
    d.deptName,  
    CONCAT(f.fname, ' ', f.lname) as advisor  
FROM student AS s LEFT JOIN department AS d  
        ON (s.major = d.deptCode)  
     LEFT JOIN faculty AS f  
        ON (s.advisor = f.facId)  
WHERE d.schoolCode = %s;  
'''  
  
# [2b] Get input values  
school\_code = 'CSE'  
if len(sys.argv) > 1:  
    school\_code = sys.argv[1]  
  
# Execute the query  
cursor.execute(query, school\_code)  
  
# [3] Use the result in the query  
for (student, major, advisor) in cursor:  
    print(f"{student}: major={major}; advisor={advisor}")  
  
# [4] Housekeeping  
cursor.close()  
cnx.close()  
  
Result:

C:\...>python student3.py CSE  
Linda Johnson: major=Computer Information Systems; advisor=Daniel Kim  
Lillian Johnson: major=Computer Information Systems; advisor=Andrew Byre  
Tony Hawk: major=Computer Science; advisor=Paul Smith  
Mary Hawk: major=Computer Science; advisor=Paul Smith  
David Hawk: major=Computer Science; advisor=Mary Tran  
Catherine Lim: major=Information Technology; advisor=None  
Larry Johnson: major=Information Technology; advisor=Deborah Gump  
   
Notes:

* %s is used as a placeholder.