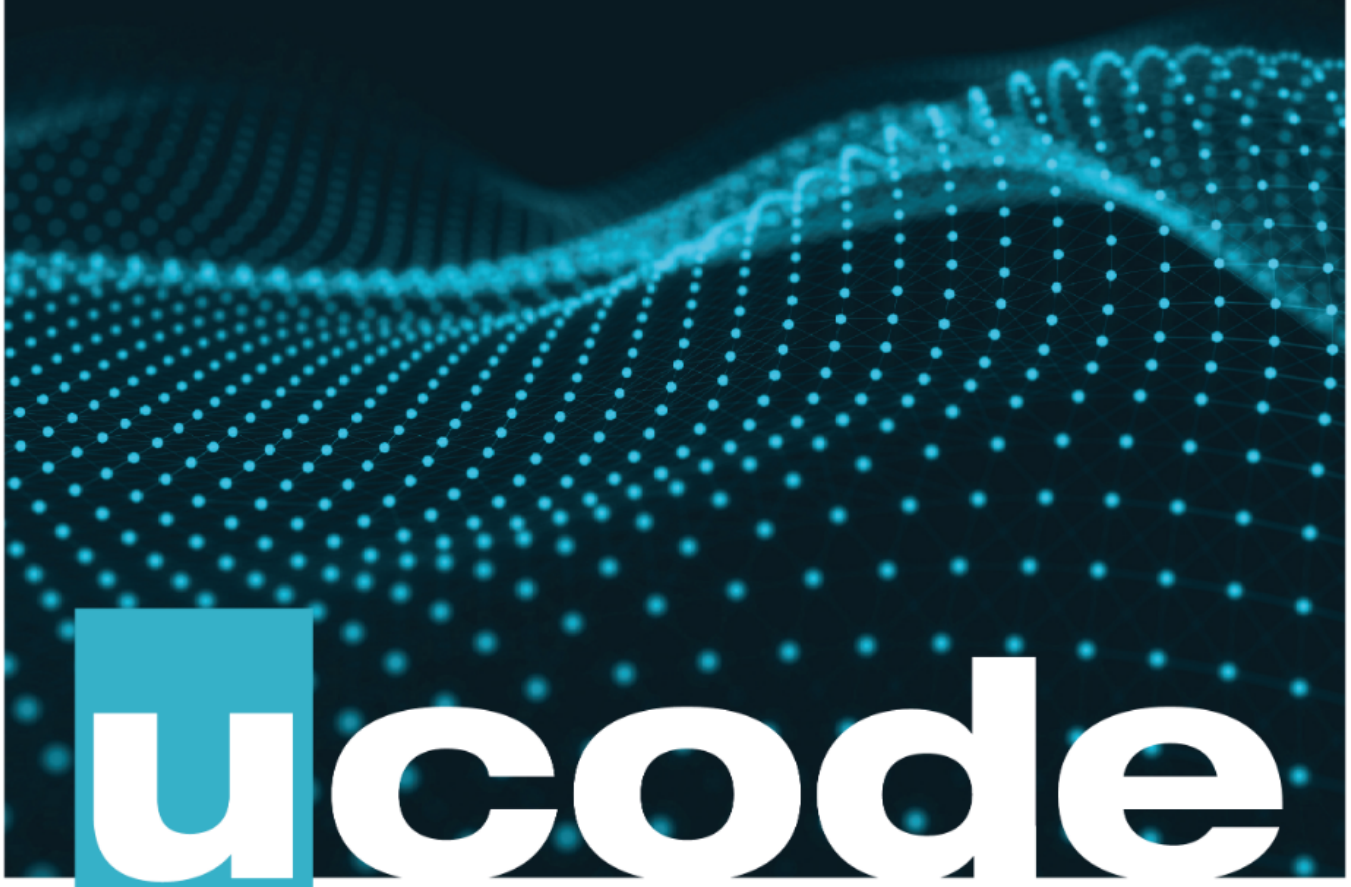
Sprint 04

Marathon Python

May 13, 2021



ucode

Contents

ucode

Engage

DESCRIPTION

This challenge focuses on several topics and you have a lot to learn.

You will work with date and time, learn what a URL is, learn to "communicate" with the browser, get acquainted with modules and packages.

Also, in this challenge, you will learn how to connect to database management systems and work with databases. Regardless of the particular area of programming, storage and manipulation of data is a vital topic. So far in your challenges, you have stored data, if any, within the program, or sometimes in text or JSON format. It's time to learn a more advanced way to work with data, using databases.

Databases offer a convenient, logical, and efficient way to store data. You can quickly and easily find, add, edit, or sort data.

Well, let's get started?

BIG IDEA

Various Python features.

ESSENTIAL QUESTION

What interesting things can you do using Python?

CHALLENGE

Learn to use various Python features.

Investigate

GUIDING QUESTIONS

We invite you to find answers to the following questions. By researching and answering them, you will gain the knowledge necessary to complete the challenge. To find answers, ask the students around you and search the internet. We encourage you to ask as many questions as possible. Note down your findings and discuss them with your peers.

- What is a URL?
- How to use a module? What are the uses?
- How is a module different from a package in Python? What are the benefits of using a package?
- What is a relational database?
- What is SQL?
- What are databases used for?
- What is a Python virtual environment?

GUIDING ACTIVITIES

Complete the following activities. Don't forget that you have a limited time to overcome the challenge. Use it wisely. Distribute tasks correctly.

1. Get acquainted with the basics of a URL.
2. Learn how to organize your code with modules and packages in this article.
3. Learn more about the history of the databases.
4. Read about database attacks.
5. Find information about the areas where databases are used.
6. Investigate how to use SQL in Python.
7. Clone your git repository issued on the challenge page in the LMS.
8. Proceed with the tasks.

ANALYSIS

Analyze your findings. What conclusions have you made after completing guiding questions and activities? In addition to your thoughts and conclusions, here are some more analysis results.

- Be attentive to all statements of the story.
- All tasks are divided into Act Basic and Act Advanced. You need to complete all basic tasks to validate the Sprint. But to achieve maximum points, consider accomplishing advanced tasks also.
- Analyze all information you have collected during the preparation stages. Try to define the order of your actions.
- Perform only those tasks that are given in this document.
- Submit only those files that are described in the story. Only useful files allowed, garbage shall not pass!

- Run the scripts using `python3`.
- Make sure that you have `Python` with a `3.8` version, or higher.
- Use the standard library available after installing `Python`. You may use additional packages/libraries that were not previously installed only if they are specified in the task.
- To figure out what went wrong in your code, use PEP 553 -- Built-in breakpoint().
- Complete tasks according to the rules specified in the PEP8 conventions.
- The solution will be checked and graded by students like you. Peer-to-Peer learning.
- Also, the challenge will pass automatic evaluation which is called `Oracle`.
- If you have any questions or don't understand something, ask other students or just Google it.

Act Basic: Task 00

NAME

Read URL

DIRECTORY

`t00_read_url/`

SUBMIT

`html_save.py, helper.py`

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- What is a `URL`?
- What elements does a `URL` consist of?
- What is the difference between `standard error` and `standard output` streams?
- How to create your own module in Python?
- What is a request (in programming)?
- How to make an HTTP request using the `requests` library?
- What HTTP methods are there?
- What is the result of an HTTP GET request?
- What status codes can a response have?
- How to get the content at a certain URL?

DESCRIPTION

Create a script with function a `html_save()` that:

- takes a `URL` address as the first argument and a file path as the second argument
- validates the `URL` that is passed to it using a validation regex. Note that the `URL` validation must cover the different parts of the address and their different variations (scheme, domain, port, etc.). Also, pay attention that some URL parts are required and some are optional. There is a list in the `resources` of valid and invalid URLs for you to test your code.
- downloads and saves the web page content located at the provided `URL` to the specified file
- displays logs describing the execution of each step (see the `EXAMPLE` for more details)

The file path may be either absolute or relative.

All errors and information during the process must be printed to either the `stderr` or `stdout` respectively.
For it, create a file-module `helper.py` that contains two functions:

- `print_stderr()` prints a given error message to stderr in the following format `'ERROR| [message].'` and exits the program
- `print_stdout()` prints a given info message to stdout in the following format `'INFO| [message].'`

Don't forget to catch all errors that may happen when making a request. Print them using the `print_stderr()` function.

In this task you must use `re`, `requests` and your own `helper` modules.

The script in the EXAMPLE tests your function. If everything is correct, it should generate output as seen in the CONSOLE VIEW. The PYTHON INTERPRETER contains an additional example. Pay attention that you must submit two files, html_save.py and helper.py.

EXAMPLE

```
from html_save import html_save


if __name__ == '__main__':

    print('*** Test 1 ***')
    html_save('https://www.bbc.com/', 'bbc.html')

    print('*** Test 2 ***')
    html_save('bbc.com', 'bbc.html')
```

CONSOLE VIEW

```
>python3 s04t00_read_url_main.py
*** Test 1 ***
INFO| Sending the request to the 'https://www.bbc.com/'.
INFO| Request to the 'https://www.bbc.com/' has been sent.
INFO| <Response [200]>.
INFO| Parsing page data.
INFO| Page data has been parsed.
INFO| Saving page data to 'bbc.html'.
INFO| Page data has been saved.
*** Test 2 ***
ERROR| The site URL format is invalid.
>head -10 bbc.html
    <!DOCTYPE html>
<html class="b-header--black--white b-pw-1280 b-reith-sans-font">

    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <meta name="description" content="Breaking news, sport, TV, radio and a whole
↪   lot more.
```

```
        The BBC informs, educates and entertains - wherever you are, whatever your
↪   age.">
        <meta name="keywords" content="BBC, bbc.co.uk, bbc.com, Search, British
↪   Broadcasting Corporation, BBC iPlayer, BBCi">
        <title>BBC - Homepage</title>

>
```

PYTHON INTERPRETER

```
>python3
>>> from html_save import html_save
>>> html_save('https://www.google.com.ua/', 'google.html')
INFO| Sending the request to the 'https://www.google.com.ua/'.
INFO| Request to the 'https://www.google.com.ua/' has been sent.
INFO| <Response [200]>.
INFO| Parsing page data.
INFO| Page data has been parsed.
INFO| Saving page data to 'google.html'.
INFO| Page data has been saved.
>>> html_save('google', 'google.html')
ERROR| The site URL format is invalid.
>python3
>>> from html_save import html_save
>>> html_save('https://www.google.com.ua/qwe', 'google.html')
INFO| Sending the request to the 'https://www.google.com.ua/qwe'.
INFO| Request to the 'https://www.google.com.ua/qwe' has been sent.
INFO| <Response [404]>.
ERROR| Request failed.
>python3
>>> from html_save import html_save
>>> html_save('http://142.42.1.1:8080/', 'test')
INFO| Sending the request to the 'http://142.42.1.1:8080/'.
ERROR| HTTPConnectionPool(host='142.42.1.1', port=8080): Max retries exceeded with url: /
↪   (Caused by NewConnectionError('<urllib3.connection.HTTPConnection object at
↪   0x7fa94800e2b0>: Failed to establish a new connection: [Errno 65] No route to
↪   host')).
>python3
>>> from helper import print_stderr, print_stdout
>>> print_stdout('Some output')
INFO| Some output.
>>> print_stderr('Some error')
ERROR| Some error.
>
```

SEE ALSO

System-Specific Parameters and Functions
What is a URL?

Response Methods - Python requests
Python Modules

ucode

Act Basic: Task 01

NAME

YouTube

DIRECTORY

`t01_youtube/`

SUBMIT

`player.py, helper.py`

BEFORE YOU BEGIN

Read the documentation of the webbrowser module.

In order to complete this task, you must be able to answer the following questions:

- How to open a URL in the browser using the webbrowser module?
- How to make sure that a web page will open specifically in a new tab or window?
- What is the role of the `BROWSER` environment variable?

DESCRIPTION

Create a script that does the following:

- takes a YouTube link as a command-line argument (if more than one argument is given, just use the first one and ignore the rest)
- validates the link in the following way:
  - same validation rules as in the task Task 00 > Read URL
  - the link must be a YouTube link (the domain name must be 'youtube')
- uses the method `webbrowser.open()` to open the browser and play the YouTube video at the URL

All actions must be logged as messages printed to the default `stdout` or `stderr` (depending on the message). Use the module you've created earlier. The exact messages can be seen in the CONSOLE VIEW section (also available in the resources).

Use the webbrowser module.

CONSOLE VIEW

```
>python3 player.py https://www.youtube.com/watch\?v\=yKQ_sQKBASM
INFO| Opening the YouTube video at https://www.youtube.com/watch?v=yKQ_sQKBASM.
INFO| YouTube video opened.
>python3 player.py
ERROR| The site URL was not passed.
>python3 player.py https://pymotw.com/2/webbrowser/
ERROR| The URL is invalid.
>python3 player.py https://www.youtube.com/watch\?v\=yKQ_sQKBASM some random arguments
INFO| Opening the YouTube video at https://www.youtube.com/watch?v=yKQ_sQKBASM.
INFO| YouTube video opened.
>
```

SEE ALSO

Convenient Web-Browser Controller

Act Basic: Task 02

NAME

Knight factory

DIRECTORY

`t02_knight_factory/`

SUBMIT

`knight_factory.py, generator/__init.py__, generator/names.py, generator/titles.py`

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- How is a module different from a package in Python? What is the benefit of using a package?
- What is the role of `__init__.py` in a package?
- What standard module is used to work with logging?
- What is the role of a `Handler` object in logging? How to print a log to the error or output stream?
- How can a `Formatter` object help? What are `LogRecord` attributes and what are they used for?

DESCRIPTION

Create a package with the name `generator`.

Create the `__init__.py` file for the package, where you should import all necessary packages/modules.

Also, inside the package create two modules: `names.py` and `titles.py`.

The `names` module has a function `names()` that generates a random name and the `titles` module has a function `titles()` that generates a random title from predefined lists. You can find examples of lists to use in the file s0402_knight_factory_example.txt from the resources. You can modify the lists or use different ones, as long as there are at least three values in each.

Create a `knight_factory.py` script and use the package there. The script must create a knight by generating a name and a title and log the knight's information in the following format:
'Sir [name] the [title]'
Generate five knights.

Also, your script must work with `logging` (place the logging process in your `__init__.py`):

- import the required module
- create a logger instance
- use the INFO level messages

- messages must be printed to the stdout
- the format of the logs must be: '...:Knight Generator::.. [process]-[levelname]-[message]' (the process may vary). There are helpful resources in the SEE ALSO section on how to print meta information that is in square brackets

EXAMPLE

```
# a list of names

'Galahad',
'Lancelot',
'Gawain',
'Percivale',
'Lionell',
'Tristram de Lyones',
'Gareth',
'Bedivere',
'Bleoberis',
'Lacotemale Taile',
'Lucan',
'Palomedes',
'Lamorak',
'Bors de Ganis',
'Safer',
'Pelleas',
'Kay',
'Ector de Maris',
'Dagonet',
'Degore',
'Brunor le Noir',
'Lebius Desconneu',
'Alymere',
'Mordred'

# a list of titles

'Tiny',
'Good-Enough',
'Thrice-Divorced',
'Not-so-Mighty',
'Hungry',
'Puzzled',
'Never-There',
'Tall-One'
```

CONSOLE VIEW

```
>python3 knight_factory.py
..::Knight Generator::.. 8672-INFO-Package __init__ executed.
..::Knight Generator::.. 8672-INFO-[Name chosen.]
..::Knight Generator::.. 8672-INFO-[Title chosen.]
..::Knight Generator::.. 8672-INFO-Sir Pelleas the Tall-One
..::Knight Generator::.. 8672-INFO-[Name chosen.]
..::Knight Generator::.. 8672-INFO-[Title chosen.]
..::Knight Generator::.. 8672-INFO-Sir Dagonet the Tiny
..::Knight Generator::.. 8672-INFO-[Name chosen.]
..::Knight Generator::.. 8672-INFO-[Title chosen.]
..::Knight Generator::.. 8672-INFO-Sir Lebius Desconneu the Thrice-Divorced
..::Knight Generator::.. 8672-INFO-[Name chosen.]
..::Knight Generator::.. 8672-INFO-[Title chosen.]
..::Knight Generator::.. 8672-INFO-Sir Lamorak the Never-There
..::Knight Generator::.. 8672-INFO-[Name chosen.]
..::Knight Generator::.. 8672-INFO-[Title chosen.]
..::Knight Generator::.. 8672-INFO-Sir Percivale the Thrice-Divorced
>
```

SEE ALSO

Create a Python package with __init__.py
Modules and Packages
logging - Logging facility for Python

Act Basic: Task 03

NAME

SQLite manager

DIRECTORY

`t03_sqlite_manager/`

SUBMIT

`manager.py`

BEFORE YOU BEGIN

In this and next several tasks you will be working with databases. Before you get started, research the key concepts of relational databases, SQL, and SQLite. SQLite is a good starting point in learning SQL, because it's lightweight, easy-to-use, and doesn't require a server or any configurations to get started. Here are some articles with brief theoretical background on the topics:

- Understanding Relational Databases
- What is SQL?
- sqlite3 - Embedded Relational Database

In order to complete this task, you must be able to answer the following questions:

- What is a relational database?
- What is SQL?
- What are the pros and cons of SQL?
- What is a query?
- How to start working with SQLite in Python?
- How to create a connection to a database?
- How to close a connection to a database?
- What is a cursor object in sqlite3?
- How to execute a SQL statement using sqlite3?
- How to commit changes to a database?

DESCRIPTION

Create a program that manages the work with SQLite databases using user input. The program must be able to create connections to databases, close the connections, and execute a given SQL statement.

In order to manage all the connections, create a dict `connections`, where you will store the filenames of the databases as the keys, and the connection objects as the values. Everytime a connection is closed, remove the corresponding dictionary item.

Valid commands are:

- help
  Shows the list of available commands. See the required format of the output in the CONSOLE VIEW.
- connect [database]
  Tries to create a connection to the given database file. Possible scenarios:
  - the connection to this database already exists (check using your dict connections).
    In this case, print 'Already connected to database "[database]".'
  - there was a `sqlite3.Error` when attempting to create the connection.
    In this case, print out the error.
  - the connection was successfully created.
    In this case, print 'Created connection to database "[database]".'. Also, add the created connection to the dict connections.
- close [database]
  Tries to close a connection to the given database file. Possible scenarios:
  - the connection to this database doesn't exist.
    In this case, print 'Cannot close connection to "[database]". Not connected.'
  - there was a `sqlite3.Error` when attempting to close the connection.
    In this case, print out the error.
  - the connection was successfully closed.
    In this case, print 'Closed connection to database "[database]".'. Also, remove the corresponding item from the dict connections.
- execute [database] "[SQL statement]"
  Tries to execute the given SQL statement onto the given database file. (You will need to create a cursor object, use it to execute the statement, and then commit changes). Possible scenarios:
  - the connection to this database doesn't exist.
    In this case, print 'Cannot execute SQL statement. Not connected to "[database]".'
  - there was a `sqlite3.Error` when attempting to execute the statement.
    In this case, print out the error.
  - the statement was successfully executed.
    In this case, print 'Executed SQL statement.'.
- show
  Shows the list of currently open connections as a list of database files. See the exact format of the output in the CONSOLE VIEW. If there are no connections, prints 'No connections.'.
- exit
  Closes all open connections and exits the program.

In case of an invalid command, the program prints:
'Invalid command. Try "help" to see available commands.'
Here are some examples of invalid commands:

- 'commit' (completely wrong command)
- 'help me' (command 'help' given too many arguments)

- 'connect' (command 'connect' not given database filename)
- 'connect my_database.db another_database.db' (command 'connect' given too many arguments)
- 'execute my_database.db "SELECT name FROM users' (SQL statement must have double quotes on both sides)

Test your program as shown in the CONSOLE VIEW. Your outputs must be identical.

Pay attention that as a result of all the commands in the CONSOLE VIEW, your working directory will contain the files: my_database, another_database.db, cats.db, etc. Don't forget to delete them before submitting your solution.

CONSOLE VIEW

```
>python3 manager.py
Enter command: help
Available commands:
- help
- connect [database]
- close [database]
- execute [database] "[SQL statement]"
- show
- exit
Enter command: show
No connections.
Enter command: exit
>python3 manager.py
Enter command: connect my_database
Created connection to database "my_database".
Enter command: show
Connected to:
['my_database']
Enter command: connect another_database.db
Created connection to database "another_database.db".
Enter command: show
Connected to:
['my_database', 'another_database.db']
Enter command: exit
Closed connection to database "my_database".
Closed connection to database "another_database.db".
>python3 manager.py
Enter command: connect cats.db
Created connection to database "cats.db".
Enter command: connect dogs.db
Created connection to database "dogs.db".
Enter command: connect birds.db
Created connection to database "birds.db".
Enter command: connect dogs.db
Already connected to database "dogs.db".
Enter command: show
Connected to:
```

```
['cats.db', 'dogs.db', 'birds.db']
Enter command: close snakes.db
Cannot close connection to "snakes.db". Not connected.
Enter command: close dogs.db
Closed connection to database "dogs.db".
Enter command: close dogs.db
Cannot close connection to "dogs.db". Not connected.
Enter command: show
Connected to:
['cats.db', 'birds.db']
Enter command: execute cats.db "CREATE TABLE my_cats(id integer, name text)"
Executed SQL statement.
Enter command: execute cats.db "INSERT INTO my_cats VALUES(4, 'Tom')"
Executed SQL statement.
Enter command: execute cats.db "not an sql statement"
near "not": syntax error
Enter command: exit
Closed connection to database "cats.db".
Closed connection to database "birds.db".
>
```

SEE ALSO

sqlite3 cheatsheet for Python 3

Act Basic: Task 04

NAME

ZooLite

DIRECTORY

`t04_zoolite/`

SUBMIT

`zoolite.py`

BEFORE YOU BEGIN

In this task you will start practicing various SQL statements. In order to complete this task, you must be able to answer the following questions:

- What is the syntax of an SQL statement that creates a table?
- Which three magic keywords can you add after `'CREATE TABLE'` to tell the database to ignore the whole statement if the table already exists?
- How to get the number of rows in a table?
- When creating a table, how to specify that the value for a certain column cannot be left empty?
- How to set a `DEFAULT` value for a column?
- What is a `PRIMARY KEY` in SQL and why is it useful?
- How to use sqlite3 command-line tool?
- What is the syntax of an SQL statement that inserts a new row into a table?
- What is the syntax of a `DELETE` query?
- How to increment a value of a column?
- What keyword is missing in this statement:
  `'DELETE my_table WHERE status = "done" AND year = 1991;'`?

Follow the steps shown below to practice the basics:

SQLITE COMMAND LINE

Create a table

```
>sqlite3
SQLite version 3.31.1 ...
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open practice.db
sqlite> CREATE TABLE test_results (
   ...> student_id INTEGER,
   ...> result INTEGER NOT NULL,
```

```
   ...> comment TEXT);
sqlite> .tables
test_results
```

Insert rows

```
sqlite> INSERT INTO test_results (student_id, result, comment)
   ...> VALUES (257, 50, "average understanding of the topic");
sqlite> INSERT INTO test_results (student_id, result)
   ...> VALUES (12, 95);
sqlite> INSERT INTO test_results (student_id, result, comment)
   ...> VALUES (458, 90, "excellent work");
sqlite> INSERT INTO test_results (student_id, result, comment)
   ...> VALUES (4, 75, "very promising");
```

Select rows

```
sqlite> .headers on
sqlite> .mode column
sqlite> SELECT * FROM test_results;
student_id  result      comment
----------  ----------  --------------------------------
257         50          average understanding of the topic
12          95
458         90          excellent work
4           75          very promising
sqlite> SELECT result FROM test_results;
result
----------
50
95
90
75
sqlite> SELECT * FROM test_results WHERE result < 80;
student_id  result      comment
----------  ----------  --------------------------------
257         50          average understanding of the topic
4           75          very promising
```

Delete rows

```
sqlite> DELETE FROM test_results WHERE student_id < 20;
sqlite> SELECT * FROM test_results;
student_id  result      comment
```

```
----------  ----------  --------------------------------
257         50          average understanding of the topic
458         90          excellent work
```

Update rows

```
sqlite> UPDATE test_results SET comment = "Amazing!" WHERE result = 90;
sqlite> SELECT * FROM test_results;
student_id  result      comment
----------  ----------  --------------------------------
257         50          average understanding of the topic
458         90          Amazing!
sqlite> .quit
```

DESCRIPTION

Create a function `update_zoo()` that takes a dictionary `data` as an argument. The main idea is to update the database of a zoo according to some news about its animals.

The dictionary `data` will contain fields `'database'`, `'table'`, and `'news'`. The value of the field `'database'` is the path to the database you have to connect to. This file may already exist or not.
The value of the field `'table'` is the name of the table that you will make changes to. The function must create the table if it doesn't exist yet. The table has the following columns:

- `species`
  will contain text data, cannot be left empty
- `name`
  will contain text data, cannot be left empty
- `age`
  will contain integer data, if not specified, defaults to `0`

There can be many animals of the same species, and can be many with the same name, but the combination of the two must always be unique. For example, there can be only one alligator Paul, but there can be many different Pauls and many different alligators in the zoo. Therefore, when creating the table, you will need to add a line that instructs the table to use the combination of the two columns as the `PRIMARY KEY`.

The value of the field `'news'` is a list of events that happened. Each item in the list contains the fields `'species'`, `'name'`, and `'event'`. The values of `'species'` and `'name'` will help you find the correct animal in the database, and the value of `'event'` determines what should be done with the record of the animal.

The value of `'event'` can be one of the following options:

- `'born'`
  insert a new record to the table

- 'died'
  delete a record from the table
- 'birthday'
  increment the age of the animal by one

Your function will have to construct a correct SQL statement based on the event and the animal's info, and then execute and commit the changes.

Your function must print short messages about the successfully processed events. See the exact formatting in the section CONSOLE VIEW. By success we mean that not only the statement's execution did not cause an error, but also that a row has been modified. For example, if you try to update the age of an animal that is not in the zoo, there won't be any errors, but the table will not be changed in any way. To check if any rows have been affected, you can use the cursor.rowcount method.

Error handling:

- if there has been an `sqlite3.Error` when trying to connect to a database, to create a table, or to close the connection, then print the error and return from the function
- if there has been an `sqlite3.Error` when trying to execute a statement for one of the events, print the error (in a particular format), but continue processing the remaining events
  Format of output in this case:
  'Failed to process event: [event]. Error: [error].'
  (see example in the CONSOLE VIEW)

Here is a general outline for the order and logic of actions of the function:

- connect
- create the table if necessary
- print the number of rows in the table (in the format visible in the CONSOLE VIEW)
- for each of the news try to execute a relevant statement and commit, and if:
  - no errors and the table has been modified -> print a message about what was done
  - no errors but the table has not been modified -> don't print anything
  - there was an error -> print the error message as described in the Error handling description
- print the number of rows in the table (in the format visible in the CONSOLE VIEW)
- close the connection

The script in the EXAMPLE will help you test your solution. If your function works correctly, the output will match the one shown in the CONSOLE VIEW.

EXAMPLE

```
from zoolite import update_zoo
import json
import sys

if __name__ == '__main__':
    filename = sys.argv[1]
    with open(filename) as data_file:
        update_zoo(data=json.load(data_file))
```

CONSOLE VIEW

```
>python3 s04t04_zoolite_main.py s04t04_zoolite_fill.json
*** BEFORE: 0 animals in the zoo.
Inserted Gorilla Lucy into table animals of zoo.db.
Inserted Anaconda Ziggy into table animals of zoo.db.
Inserted Wolf Oscar into table animals of zoo.db.
Inserted Penguin Coco into table animals of zoo.db.
Failed to process event: {'species': 'Anaconda', 'name': 'Ziggy', 'event': 'born'}.
↪  Error: UNIQUE constraint failed: animals.species, animals.name
Inserted Zebra Coco into table animals of zoo.db.
Inserted Seal Jack into table animals of zoo.db.
Inserted Flamingo Ziggy into table animals of zoo.db.
Inserted Iguana Lucy into table animals of zoo.db.
Inserted Gorilla Max into table animals of zoo.db.
Inserted Giraffe Daisy into table animals of zoo.db.
*** AFTER: 10 animals in the zoo.
>sqlite3 zoo.db
SQLite version 3.31.1 ...
Enter ".help" for usage hints.
sqlite> .tables
animals
sqlite> .mode column
sqlite> .headers on
sqlite> SELECT * FROM animals;
species     name        age
----------  ----------  ----------
Gorilla     Lucy        0
Anaconda    Ziggy       0
Wolf        Oscar       0
Penguin     Coco        0
Zebra       Coco        0
Seal        Jack        0
Flamingo    Ziggy       0
Iguana      Lucy        0
Gorilla     Max         0
Giraffe     Daisy       0
sqlite> .quit
```

```
>python3 s04t04_zoolite_main.py s04t04_zoolite_random.json
*** BEFORE: 10 animals in the zoo.
Updated Iguana Lucy in table animals of zoo.db.
Failed to process event: {'species': 'Giraffe', 'name': 'Daisy', 'event': 'born'}.
↪  Error: UNIQUE constraint failed: animals.species, animals.name
Inserted Lemur Zeus into table animals of zoo.db.
Inserted Flamingo Coco into table animals of zoo.db.
Updated Gorilla Max in table animals of zoo.db.
Inserted Giraffe Ziggy into table animals of zoo.db.
Inserted Hyena Max into table animals of zoo.db.
Inserted Panda Daisy into table animals of zoo.db.
Updated Gorilla Max in table animals of zoo.db.
Inserted Cheetah Ziggy into table animals of zoo.db.
Deleted Hyena Max from table animals of zoo.db.
Inserted Flamingo Ruby into table animals of zoo.db.
*** AFTER: 16 animals in the zoo.
>sqlite3 zoo.db
SQLite version 3.31.1 ...
Enter ".help" for usage hints.
sqlite> .mode column
sqlite> .headers on
sqlite> SELECT * FROM animals;
species     name        age
----------  ----------  ----------
Gorilla     Lucy        0
Anaconda    Ziggy       0
Wolf        Oscar       0
Penguin     Coco        0
Zebra       Coco        0
Seal        Jack        0
Flamingo    Ziggy       0
Iguana      Lucy        1
Gorilla     Max         2
Giraffe     Daisy       0
Lemur       Zeus        0
Flamingo    Coco        0
Giraffe     Ziggy       0
Panda       Daisy       0
Cheetah     Ziggy       0
Flamingo    Ruby        0
sqlite> .quit
>
```

PYTHON INTERPRETER

```
>python3
>>> from zoolite import update_zoo
>>> data = {'database': 'animal_land', 'table': 'birds',
...         'news': [ {'species': 'eagle',
...                    'name': 'Stan',
...                    'event': 'born'},
...                   {'species': 'condor',
...                    'name': 'Sally',
...                    'event': 'born'}]}
>>> update_zoo(data)
*** BEFORE: 0 animals in the zoo.
Inserted eagle Stan into table birds of animal_land.
Inserted condor Sally into table birds of animal_land.
*** AFTER: 2 animals in the zoo.
>>> data = {'database': 'animal_land', 'table': 'reptiles',
...          'news': [ {'species': 'king cobra',
...                     'name': 'Leon',
...                     'event': 'born'}]}
>>> update_zoo(data)
*** BEFORE: 0 animals in the zoo.
Inserted king cobra Leon into table reptiles of animal_land.
*** AFTER: 1 animals in the zoo.
>>> data = {'database': 'animal_land', 'table': 'birds',
...          'news': [ {'species': 'eagle',
...                     'name': 'Stan',
...                     'event': 'birthday'}]}
>>> update_zoo(data)
*** BEFORE: 2 animals in the zoo.
Updated eagle Stan in table birds of animal_land.
*** AFTER: 2 animals in the zoo.
>>>
```

SEE ALSO

SQL As Understood By SQLite
Command Line Shell For SQLite
SQLite Python

Act Advanced: Task 05

NAME

Connect MySQL

DIRECTORY

`t05_connect_mysql/`

SUBMIT

`database.py, cfg.yaml`

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- What is MySQL?
- What are the differences between MySQL and standard SQL?
- What modules need to be imported in order to work with MySQL in Python?
- What information do you need to provide to create a database connection?
- Which method returns information about the MySQL server?
- How to open and close a connection to a MySQL server?

Prepare for completing tasks with MySQL. Install:

- the latest stable version of MySQL server
- `mysql-connector-python`
- `pyyaml`

DESCRIPTION

Create a `cfg.yaml` file with MySQL connection settings. You can find an example in the EXAMPLE section.

Create a script that:

- gets configs for connection from a `cfg.yaml` file
- creates a connection
- if successful, prints a message that includes MySQL server information (your version of the MySQL)
- closes the connection
- prints the appropriate messages at each step (look for them in the CONSOLE VIEW)
- uses `mysql.connector.Error` to catch all errors in case of connection problems

The CONSOLE VIEW shows two runs of the program, in one of them the password was not specified (caused an error).

EXAMPLE

```
settings:
  host: your_host_here
  user: your_user_name_here
  password: your_pass_here
```

CONSOLE VIEW

```
>python3 database.py
MySQL connection Error: 1045 (28000): Access denied for user 'root'@'localhost' (using
↪  password: NO)
>python3 database.py
Connected to MySQL server (version 8.0.24).
MySQL connection is closed.
>
```

SEE ALSO

Python MySQL
MySQL Connector/Python Developer Guide
errors.Error Exception

Act Advanced: Task 06

NAME

Database

DIRECTORY

`t06_database/`

SUBMIT

`database.py, cfg.yaml`

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- What is the syntax of queries in MySQL?
- How to create a database?
- What does it mean to drop a database?
- How to see all existing databases?

DESCRIPTION

Create a program that reads command-line arguments, and performs certain actions depending on those arguments.

If the command-line arguments are valid, the program must connect to the MySQL server using the settings given in the file `cfg.yaml`. It may optionally contain the name of the database to connect to. If it does, connect to that database. If the connection has been established successfully (there is a dedicated method for checking that), print a message in the format

'Connected to MySQL server (version [version]), database [database].' Get the name of the database currently connected to using a query.

The program must close the connection before exiting.

If the command-line argument is 'databases', the program must print all existing databases. See format of the output in the CONSOLE VIEW.

If the command-line argument is 'tables', the program must print all the tables that exist in the database that the program is currently connected to. If the program is not connected to any database, print a corresponding message.

If the command-line arguments are 'create [database]', the program must create a new database with that name.

If the command-line arguments are 'drop [database]', the program must drop the database with that name.

If the arguments are not valid, print 'Error: command-line arguments are invalid.'. Here

are some examples of invalid usage of the program:
python3 database.py create
python3 database.py tables my_database
python3 database.py Databases
python3 database.py drop my_database hello
python3 database.py

Test your program as shown in the CONSOLE VIEW. Your outputs must follow the same logic.
Catch all possible errors, and print them in the format given in the console section.

CONSOLE VIEW

```
>cat cfg.yaml
settings:
  host: [hidden]
  user: [hidden]
  password: [hidden]
>python3 database.py
Error: command-line arguments are invalid.
>python3 database.py databases
Connected to MySQL server (version 8.0.22), database None.
Database list:
        - information_schema
        - mysql
        - performance_schema
        - sys
        - user
MySQL connection is closed.
>python3 database.py create my_db
Connected to MySQL server (version 8.0.22), database None.
Database 'my_db' is created.
MySQL connection is closed.
>python3 database.py create my_db
Connected to MySQL server (version 8.0.22), database None.
MySQL connection Error: 1007 (HY000): Can't create database 'my_db'; database exists
>python3 database.py tables
Connected to MySQL server (version 8.0.22), database None.
Error: not connected to a database.
MySQL connection is closed.
>echo "  database: my_db" >> cfg.yaml
>cat cfg.yaml
settings:
  host: [hidden]
  user: [hidden]
  password: [hidden]
  database: my_db
>python3 database.py tables
Connected to MySQL server (version 8.0.22), database my_db.
No tables.
MySQL connection is closed.
>sed -i'' -e 's/my_db/mysql/g' cfg.yaml
```

```
>cat cfg.yaml
settings:
  host: [hidden]
  user: [hidden]
  password: [hidden]
  database: mysql
>python3 database.py tables
Connected to MySQL server (version 8.0.22), database mysql.
Table list:
        - columns_priv
        - component
        - db
        - default_roles
        - engine_cost
        - func
        - general_log
        [more lines]
MySQL connection is closed.
>python3 database.py databases
Connected to MySQL server (version 8.0.22), database mysql.
Database list:
        - information_schema
        - my_db
        - mysql
        - performance_schema
        - sys
        - user
MySQL connection is closed.
>python3 database.py drop my_db
Connected to MySQL server (version 8.0.22), database mysql.
Database 'my_db' is dropped.
MySQL connection is closed.
>python3 database.py drop my_db
Connected to MySQL server (version 8.0.22), database mysql.
MySQL connection Error: 1008 (HY000): Can't drop database 'my_db'; database doesn't exist
>
```

SEE ALSO

Python MySQL Create Database
Getting Started with MySQL in Python

Act Advanced: Task 07

NAME

MyZoo

DIRECTORY

`t07_myzoo/`

SUBMIT

`myzoo.py, cfg.yaml`

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- What query lets you select the database you want to work with (when you're already connected to the MySQL server)?
- How to create a database only if it doesn't exist yet?
- What can be used to change the columns of an existing table?
- How to get the column names of a table?
- How to print a table in a pretty way?

DESCRIPTION

In this task you will create a function similar to the one in Task 04 - ZooLite, but this time using MySQL. Read the task description very carefully, because, even though the function is similar, there are some significant changes (e.g., event types, outputs, etc.).

Create a function `update_zoo()` that takes a dictionary `data` as an argument. The dictionary will contain fields `'database'`, `'table'`, and `'news'`. The value of the field `'database'` is the name of the database you have to connect to. You have to create the database if it doesn't exist yet. The value of the field `'table'` is the name of the table that you will make changes to. The function must create the table if it doesn't exist yet.

The table must have the following columns:

- `species`
  will contain text data of at most 100 length, cannot be left empty
- `name`
  will contain text data of at most 100 length, cannot be left empty
- `age`
  will contain integer data with values ranging from 0 to 100, if not specified, defaults to 0

The species and name together are the `PRIMARY KEY`.

The value of the field `'news'` is a list of events that happened. Each item in the list contains a field `'event'` and some other fields. In most cases the fields `'species'` and `'name'` are given. Other fields may also be given (depends on event).

The value of 'event' determines what should be done with the record of the animal. It can be one of the following options:

- 'born'
  insert a new record to the table (initialize age to default value)
  'species' and 'name' always given, and may optionally be given additional fields (apart from 'age')
- 'died'
  delete a record from the table
  only one animal; 'species' and 'name' will always be given
- 'birthday'
  increment the age of the animal by one
  'species' and 'name' always given, additional fields (apart from 'age') may be given (need to update these fields as well)
- 'moved in'
  insert a new record to the table (initialize age to given value)
  'species', 'name', and 'age' always given, additional fields of information may be given
- 'moved out'
  delete records from the table (zero or more animals, depending on given fields)
  find what records to delete using provided fields

Your function will have to construct a correct query based on the event and the animal's info, and then execute and commit the changes.

As already mentioned, news items may contain additional fields. For example, this news item:
"species": "Giraffe", "name": "Daisy", "event": "born", "weight": 68
contains an additional field "weight". If a column with that name doesn't exist yet, add it to the table. Determine the necessary data type from the value of the field. Your solution must decide between three data types to give to the column. Here are the requirements for the SQL data types that you will need:

- a data type that supports integers
- a data type that supports strings of at most 300 length
- a data type that supports boolean values

Error handling:

- if there has been an `mysql.connector.Error` when trying to connect to a database, to create a table, or to close the connection, then print the error and return from the function
- if there has been an error when trying to execute a statement for one of the events, print the error (in a particular format), but continue processing the remaining events
  Format of output in this case:
  'Failed to process event: [event]. Error: [error].'
  (see example in the CONSOLE VIEW)

Here is a general outline for the order and logic of actions of the function:

- connect to MySQL server
- create the necessary database if it doesn't exist yet
- use that database
- create the necessary table if it doesn't exist yet
- print the table's contents (including headers)
- for each of the news try to execute a relevant statement, and if:
  - no errors and the table has been modified -> print a message about what was done
  - no errors but the table has not been modified -> don't print anything
  - there was an error -> print the error message as described in the Error handling description
- print the table's contents (including headers)
- if the table is empty, drop it, and print a corresponding message
- close the connection

The script in the EXAMPLE will help you test your solution. See the CONSOLE VIEW for format and content of outputs of your function. You are not required to format the table in the exact same way as shown in the CONSOLE VIEW, but all rows and headers must be printed.

EXAMPLE

```
from myzoo import update_zoo
import json
import sys


if __name__ == '__main__':
    filename = sys.argv[1]
    with open(filename) as data_file:
        update_zoo(data=json.load(data_file))
```

CONSOLE VIEW

```
>python3 s04t07_myzoo_main.py s04t07_myzoo_fill.json
Connected to MySQL server (version 8.0.22).
Using database 'zoo'.
Table animals: 0 rows, 3 columns.
+-----------+--------+-------+
| species   | name   | age   |
|-----------+--------+-------|
+-----------+--------+-------+
Inserted Bear Angela into table animals of zoo.
Column weight added to table animals.
Column area added to table animals.
```

```
Inserted Bear Walter into table animals of zoo.
Inserted Hyena Jay into table animals of zoo.
Inserted Horse Jack into table animals of zoo.
Column endangered added to table animals.
Inserted Giraffe Lisa into table animals of zoo.
Inserted Koala Max into table animals of zoo.
Inserted Panda Rocky into table animals of zoo.
Inserted Penguin Ollie into table animals of zoo.
Inserted Bat Red into table animals of zoo.
Inserted Wolf Max into table animals of zoo.
Inserted Bear Luke into table animals of zoo.
Inserted Bear Rocky into table animals of zoo.
Inserted Gorilla Oscar into table animals of zoo.
Inserted Lion Ollie into table animals of zoo.
Inserted Giraffe Jay into table animals of zoo.
Inserted Hyena Oscar into table animals of zoo.
Failed to process event: {'species': 'Penguin', 'name': 'Ollie', 'event': 'born'}.
↪  Error: 1062 (23000): Duplicate entry 'Penguin-Ollie' for key 'animals.PRIMARY'
Failed to process event: {'species': 'Giraffe', 'name': 'Jay', 'event': 'moved in',
↪  'age': 25, 'area': 'A2'}. Error: 1062 (23000): Duplicate entry 'Giraffe-Jay' for key
↪  'animals.PRIMARY'
Inserted Tortoise Ozzy into table animals of zoo.
Inserted Horse Lisa into table animals of zoo.
Inserted Bear Max into table animals of zoo.
Inserted Giraffe Luke into table animals of zoo.
Table animals: 20 rows, 6 columns.
+-----------+--------+-------+----------+--------+--------------+
| species   | name   |   age |   weight | area   | endangered   |
|-----------+--------+-------+----------+--------+--------------|
| Bat       | Red    |     0 |          | A1     |              |
| Bear      | Angela |     0 |          |        |              |
| Bear      | Luke   |     0 |          |        | False        |
| Bear      | Max    |     0 |          |        |              |
| Bear      | Rocky  |     0 |       20 |        |              |
| Bear      | Walter |     0 |       15 | A2     |              |
| Giraffe   | Jay    |     2 |       90 |        | False        |
| Giraffe   | Lisa   |     5 |          |        |              |
| Giraffe   | Luke   |    12 |          |        |              |
| Gorilla   | Oscar  |     0 |          | A1     |              |
| Horse     | Jack   |     6 |          |        |              |
| Horse     | Lisa   |     8 |          |        |              |
| Hyena     | Jay    |    10 |          | A2     |              |
| Hyena     | Oscar  |     0 |          |        |              |
| Koala     | Max    |     4 |          | A1     |              |
| Lion      | Ollie  |    10 |      105 |        | False        |
| Panda     | Rocky  |    10 |          |        | True         |
| Penguin   | Ollie  |     0 |          | A2     | False        |
| Tortoise  | Ozzy   |    80 |          |        |              |
| Wolf      | Max    |    10 |          |        |              |
+-----------+--------+-------+----------+--------+--------------+
```

```
MySQL connection is closed.
>python3 s04t07_myzoo_main.py s04t07_myzoo_random.json
Connected to MySQL server (version 8.0.22).
Using database 'zoo'.
Table animals: 20 rows, 6 columns.
+-----------+--------+-------+----------+--------+--------------+
| species   | name   |   age |   weight | area   | endangered   |
|-----------+--------+-------+----------+--------+--------------|
| Bat       | Red    |     0 |          | A1     |              |
| Bear      | Angela |     0 |          |        |              |
| Bear      | Luke   |     0 |          |        | False        |
| Bear      | Max    |     0 |          |        |              |
| Bear      | Rocky  |     0 |       20 |        |              |
| Bear      | Walter |     0 |       15 | A2     |              |
| Giraffe   | Jay    |     2 |       90 |        | False        |
| Giraffe   | Lisa   |     5 |          |        |              |
| Giraffe   | Luke   |    12 |          |        |              |
| Gorilla   | Oscar  |     0 |          | A1     |              |
| Horse     | Jack   |     6 |          |        |              |
| Horse     | Lisa   |     8 |          |        |              |
| Hyena     | Jay    |    10 |          | A2     |              |
| Hyena     | Oscar  |     0 |          |        |              |
| Koala     | Max    |     4 |          | A1     |              |
| Lion      | Ollie  |    10 |      105 |        | False        |
| Panda     | Rocky  |    10 |          |        | True         |
| Penguin   | Ollie  |     0 |          | A2     | False        |
| Tortoise  | Ozzy   |    80 |          |        |              |
| Wolf      | Max    |    10 |          |        |              |
+-----------+--------+-------+----------+--------+--------------+
Failed to process event: {'event': 'moved out', 'species': 'Horse', 'color': 'brown',
↪  'weight': 86}. Error: 1054 (42S22): Unknown column 'color' in 'where clause'
Failed to process event: {'species': 'Penguin', 'name': 'Ollie', 'event': 'moved in',
↪  'age': 20}. Error: 1062 (23000): Duplicate entry 'Penguin-Ollie' for key
↪  'animals.PRIMARY'
Column color added to table animals.
Inserted Puma Ty into table animals of zoo.
Deleted 4 animals from table animals of zoo.
Inserted Gorilla Ty into table animals of zoo.
Inserted Seal Jay into table animals of zoo.
Inserted Panda Alex into table animals of zoo.
Deleted 1 animal from table animals of zoo.
Inserted Tortoise Ollie into table animals of zoo.
Deleted 1 animal from table animals of zoo.
Inserted Zebra Tina into table animals of zoo.
Inserted Penguin Rocky into table animals of zoo.
Inserted Zebra Oscar into table animals of zoo.
Inserted Horse Jay into table animals of zoo.
Inserted Koala Ruby into table animals of zoo.
Inserted Penguin Daisy into table animals of zoo.
Table animals: 25 rows, 7 columns.
```

```
+-----------+--------+-------+----------+--------+--------------+---------+
| species   | name   |   age |   weight | area   | endangered   | color   |
|-----------+--------+-------+----------+--------+--------------+---------|
| Bear      | Angela |     0 |          |        |              |         |
| Bear      | Luke   |     0 |          |        | False        |         |
| Bear      | Max    |     0 |          |        |              |         |
| Bear      | Rocky  |     0 |       20 |        |              |         |
| Bear      | Walter |     0 |       15 | A2     |              |         |
| Giraffe   | Jay    |     2 |       90 |        | False        |         |
| Giraffe   | Lisa   |     5 |          |        |              |         |
| Giraffe   | Luke   |    12 |          |        |              |         |
| Gorilla   | Ty     |     6 |          |        |              |         |
| Horse     | Jack   |     6 |          |        |              |         |
| Horse     | Jay    |     0 |          | A1     | True         |         |
| Horse     | Lisa   |     8 |          |        |              |         |
| Hyena     | Jay    |    10 |          | A2     |              |         |
| Hyena     | Oscar  |     0 |          |        |              |         |
| Koala     | Ruby   |     6 |          |        | False        |         |
| Panda     | Alex   |    24 |          |        |              | black   |
| Penguin   | Daisy  |    11 |       50 | A1     |              |         |
| Penguin   | Ollie  |     0 |          | A2     | False        |         |
| Penguin   | Rocky  |     0 |       45 | A2     |              |         |
| Seal      | Jay    |     2 |          | A1     |              |         |
| Tortoise  | Ollie  |     0 |          | A1     |              | black   |
| Tortoise  | Ozzy   |    80 |          |        |              |         |
| Wolf      | Max    |    10 |          |        |              |         |
| Zebra     | Oscar  |     0 |       92 |        |              |         |
| Zebra     | Tina   |     0 |          | A1     | False        |         |
+-----------+--------+-------+----------+--------+--------------+---------+
MySQL connection is closed.
>python3 s04t07_myzoo_main.py s04t07_myzoo_clear.json
Connected to MySQL server (version 8.0.24).
Using database 'zoo'.
Table animals: 25 rows, 7 columns.
+-----------+--------+-------+----------+--------+--------------+---------+
| species   | name   |   age |   weight | area   | endangered   | color   |
|-----------+--------+-------+----------+--------+--------------+---------|
| Bear      | Angela |     0 |          |        |              |         |
| Bear      | Luke   |     0 |          |        | False        |         |
| Bear      | Max    |     0 |          |        |              |         |
| Bear      | Rocky  |     0 |       20 |        |              |         |
| Bear      | Walter |     0 |       15 | A2     |              |         |
| Giraffe   | Jay    |     2 |       90 |        | False        |         |
| Giraffe   | Lisa   |     5 |          |        |              |         |
| Giraffe   | Luke   |    12 |          |        |              |         |
| Gorilla   | Ty     |     6 |          |        |              |         |
| Horse     | Jack   |     6 |          |        |              |         |
| Horse     | Jay    |     0 |          | A1     | True         |         |
| Horse     | Lisa   |     8 |          |        |              |         |
| Hyena     | Jay    |    10 |          | A2     |              |         |
```

```
| Hyena     | Oscar  |     0 |          |        |              |         |
| Koala     | Ruby   |     6 |          |        | False        |         |
| Panda     | Alex   |    24 |          |        |              | black   |
| Penguin   | Daisy  |    11 |       50 | A1     |              |         |
| Penguin   | Ollie  |     0 |          | A2     | False        |         |
| Penguin   | Rocky  |     0 |       45 | A2     |              |         |
| Seal      | Jay    |     2 |          | A1     |              |         |
| Tortoise  | Ollie  |     0 |          | A1     |              | black   |
| Tortoise  | Ozzy   |    80 |          |        |              |         |
| Wolf      | Max    |    10 |          |        |              |         |
| Zebra     | Oscar  |     0 |       92 |        |              |         |
| Zebra     | Tina   |     0 |          | A1     | False        |         |
+-----------+--------+-------+----------+--------+--------------+---------+
Deleted 1 animal from table animals of zoo.
Deleted Tortoise Ozzy from table animals of zoo.
Deleted 3 animals from table animals of zoo.
Deleted 3 animals from table animals of zoo.
Deleted 2 animals from table animals of zoo.
Deleted 2 animals from table animals of zoo.
Deleted 7 animals from table animals of zoo.
Deleted 2 animals from table animals of zoo.
Deleted 1 animal from table animals of zoo.
Deleted 1 animal from table animals of zoo.
Deleted 2 animals from table animals of zoo.
Table animals: 0 rows, 7 columns.
+-----------+--------+-------+----------+--------+--------------+---------+
| species   | name   | age   | weight   | area   | endangered   | color   |
|-----------+--------+-------+----------+--------+--------------+---------|
+-----------+--------+-------+----------+--------+--------------+---------+
Table animals is dropped.
MySQL connection is closed.
>
```

SEE ALSO

Python MySQL Create Table
Getting Started with MySQL in Python
Python with MySQL: Connect, Create Database, Table, Insert

Act Advanced: Task 08

NAME

Datetime

DIRECTORY

`t08_datetime/`

SUBMIT

`dt_helper.py`

LEGEND

ANNOUNCER ONE: Well, it's five past nine and nearly time for six past nine. On BBC 2 now, it'll shortly be six and a half minutes past nine. Later on this evening, it'll be ten o'clock and at 10:30 we'll be joining BBC 2 in time for 10:33, and don't forget tomorrow when it'll be 9:20. Those of you who missed 8:45 on Friday will be able to see it again this Friday at a quarter to nine. Now, here is a time check. It's six and a half minutes to the big green thing.
ANNOUNCER TWO: You're a looney.
ANNOUNCER ONE: I get so bored. I get so bloody bored.

-- Monty Python's Flying Circus

DESCRIPTION

Create a script that contains three functions: `create_datetime()`, `print_formatted_datetime()` and `print_difference()`.

The `create_datetime()` function must do the following:

- take as the first argument all parameters that can then be passed as arguments to the Python `datetime` function from the `datetime` package.
  Use `*args` (asterisk construction) to pack parameters that are dynamic in size
- take as the second argument the `timezone_str` argument to adopt the default datetime to the passed timezone (default value is `None`)
- print the datetime in the specified format of the given timezone

The `print_formatted_datetime()` must do the following:

- take two arguments: the datetime object to print and the string format to print in
- print the datetime to the console in the given format

The `print_difference()` function must do the following:

- calculate the difference between two datetime objects and print it to the console
- take three arguments:
  - the first two arguments are the `datetime` objects, between which you need to find the difference

- the third one is the string `timezone_str` (default value is None) to cast previous two arguments, because the datetimes must be translated into one timezone to find the difference between them

PYTHON INTERPRETER

```
>python3
>>> from dt_helper import create_datetime, print_formatted_datetime, print_difference
>>> ukraine_timezone = 'Etc/GMT+3'
>>> datetime_1 = create_datetime(2015, 5, 21, 12, 0)
>>> datetime_2 = create_datetime(2016, 6, 14, 3, 0, timezone_str='Asia/Shanghai')
>>> datetime_2
datetime.datetime(2016, 6, 14, 3, 0, tzinfo=<DstTzInfo 'Asia/Shanghai' LMT+8:06:00 STD>)
>>> datetime_3 = create_datetime(2016, 6, 14, 3, 0, timezone_str=ukraine_timezone)
>>> datetime_3
datetime.datetime(2016, 6, 14, 3, 0, tzinfo=<StaticTzInfo 'Etc/GMT+3'>)
>>> datetime_1_format = '%d.%m.%y %H:%M:%S'
>>> datetime_2_format = '%y-%m-%d %H:%M'
>>> datetime_3_format = '%y : %H, %M, %S'
>>> print_formatted_datetime(datetime_1, datetime_1_format)
21.05.15 12:00:00
>>> print_formatted_datetime(datetime_2, datetime_2_format)
16-06-14 03:00
>>> print_formatted_datetime(datetime_3, datetime_3_format)
16 : 03, 00, 00
>>> print_difference(datetime_1, datetime_1)
0:00:00
>>> print_difference(datetime_3, datetime_2)
0:00:00
>>> print_difference(datetime_1, datetime_3, timezone_str=ukraine_timezone)
-390 days, 9:00:00
>>> print_difference(datetime_3, datetime_1, timezone_str=ukraine_timezone)
389 days, 15:00:00
>>>
```

SEE ALSO

Basic Date and Time Types
World Timezone Definitions, Modern and Historical

Act Advanced: Task 09

NAME

ls

DIRECTORY

`t09_ls/`

SUBMIT

`ls.py`

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- What is the `ls` command used for?
- How to use the `ls` command in UNIX?
- What flags does the `ls` command have?

DESCRIPTION

Create a script that acts as the UNIX `ls` command.

The script must support only two options available with `ls` command:

- `-l` - allows listing all information in the provided file path
- `-R` - tells the script to recursively analyze and print info of directories under the provided path

Also, the script must allow combining flags as `-lR/-Rl`.

The script takes the path of the directory to analyze in `ls` way and one of the options: `-l/-R/-lR/-Rl`.
If the script is run without arguments, it analyzes the current directory.

CONSOLE VIEW

```
>python3 ls.py
<dir1|file1> <dir2|file2> <dir3|file3> ...
>python3 ls.py <path>
<dir1|file1> <dir2|file2> <dir3|file3> ...
>python3 ls.py <path> -R
<path>:
<dir1|file1> <dir2|file2> <dir3|file3> ...

<path/dir1>:
<dir1|file1> <dir2|file2> <dir3|file3> ...
>python3 ls.py <path> -l
-rw------- <owner> <group> <number of bytes> <date and time of last modification>
↪  <dir1|file1>
```

```
-r-------- <owner> <group> <number of bytes> <date and time of last modification>
↪  <dir2|file2>
-rw-r--r-- <owner> <group> <number of bytes> <date and time of last modification>
↪  <dir3|file3>
>python3 ls.py <path> -lR
<path>:
-rw------- <owner> <group> <number of bytes> <date and time of last modification>
↪  <dir1|file1>
-r-------- <owner> <group> <number of bytes> <date and time of last modification>
↪  <dir2|file2>
-rw-r--r-- <owner> <group> <number of bytes> <date and time of last modification>
↪  <dir3|file3>

<path/dir1>:
-rw------- <owner> <group> <number of bytes> <date and time of last modification>
↪  <dir1|file1>
-r-------- <owner> <group> <number of bytes> <date and time of last modification>
↪  <dir2|file2>
>
```

SEE ALSO

man ls
Python printy Library
Interpreting stat() Results
The Password Database
The Group database

Act Advanced: Task 10

NAME

Venv

DIRECTORY

`t10_venv/`

SUBMIT

`venv.sh`

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- What is a virtual environment in Python?
- Why to use a virtual environment? What problems can it solve?
- How to "activate" the created environment?
- How to establish the necessary requirements to an environment?

DESCRIPTION

Create a `.sh` script that must do the following (each command must start on a new line):

1. create a virtual environment with the name environment in the task directory
2. activate it
3. install the `requests` Python library
4. freeze all dependencies to a `requirements.txt` file

Note: the console will print a message about the status of installing `requests`. Your script must be four lines long.

CONSOLE VIEW

```
>sh venv.sh
***some installing***
>
```

SEE ALSO

Creation of Virtual Environments
A Guide to Python's Virtual Environments

Act Advanced: Task 11

NAME

Soup

DIRECTORY

`t11_soup/`

SUBMIT

`soup.py`

BEFORE YOU BEGIN

Make sure to install Beautiful Soup.

In order to complete this task, you must be able to answer the following questions:

- What is the purpose of the Beautiful Soup library?
- What is web scraping?
- How to find all occurrences of a certain tag in a web page using Beautiful Soup?
- How to specify the class of the elements you're looking for?
- How to access the text content of a found HTML element?
- What is the purpose of the ensure_ascii parameter in the `json.dump()` method?

DESCRIPTION

Create a program that generates a JSON 'reading list' from a web page of the Python documentation website. Use the library Beautiful Soup.

Your program must do the following:

- take a URL as a command-line argument (assume you will only receive valid URLs from https://docs.python.org/3/)
- make a request (using requests library) to that URL
- parse the web page and find all occurrences of external references (they are marked by a particular class)
- for each of the found references find the following:
  - the link text (the text content of the a tag)
  - the URL (found within the href attribute)
  - the title (text content of the title tag) of the web page that the URL leads to (None if it's not possible to get the title)
    For this step you will have to make a request to the URL.
- find the heading (text content of the h1 tag) of the web page
- save all the found references' info as JSON in a file with a name generated from the found heading (more info below)

To generate a filename for the output file from the heading, do the following to the found heading:

- remove any leading and trailing whitespaces
- remove all characters that are not alphanumeric characters or spaces
- replace all spaces with '_'
- append '.json'

The CONSOLE VIEW section shows some examples of the results. Test with the same URLs to make sure your program generates the output files correctly.

CONSOLE VIEW

```
>python3 soup.py https://docs.python.org/3/library/unicodedata.html
>cat unicodedata__Unicode_Database.json
[
 {
  "link_text": "UCD version 13.0.0",
  "url": "https://www.unicode.org/Public/13.0.0/ucd",
  "title": "Index of /Public/13.0.0/ucd"
 },
 {
  "link_text": ""Unicode Character Database"",
  "url": "https://www.unicode.org/reports/tr44/",
  "title": "UAX #44: Unicode Character Database"
 },
 {
  "link_text": "https://www.unicode.org/Public/13.0.0/ucd/NameAliases.txt",
  "url": "https://www.unicode.org/Public/13.0.0/ucd/NameAliases.txt",
  "title": null
 },
 {
  "link_text": "https://www.unicode.org/Public/13.0.0/ucd/NamedSequences.txt",
  "url": "https://www.unicode.org/Public/13.0.0/ucd/NamedSequences.txt",
  "title": null
 }
]%
>
>python3 soup.py https://docs.python.org/3/library/itertools.html
>cat itertools__Functions_creating_iterators_for_efficient_looping.json
[
 {
  "link_text": "recurrence relations",
  "url": "https://en.wikipedia.org/wiki/Recurrence_relation",
  "title": "Recurrence relation - Wikipedia"
 },
 {
  "link_text": "more-itertools project",
  "url": "https://pypi.org/project/more-itertools/",
  "title": "more-itertools · PyPI"
 }
```

```
]%
>
>python3 soup.py https://docs.python.org/3/library/functions.html
>cat Builtin_Functions.json
[
 {
  "link_text": "guide to using super()",
  "url": "https://rhettinger.wordpress.com/2011/05/26/super-considered-super/",
  "title": "Python's super() considered super! | Deep Thoughts by Raymond Hettinger"
 }
]%
>
```

SEE ALSO

Beautiful Soup Documentation
Guide to Parsing HTML with BeautifulSoup in Python

Share

PUBLISHING

Last but not least, the final stage of your work is to publish it. This allows you to share your challenges, solutions, and reflections with local and global audiences. During this stage, you will discover ways of getting external evaluation and feedback on your work. As a result, you will get the most out of the challenge, and get a better understanding of both your achievements and missteps.

To share your work, you can create:

- a text post, as a summary of your reflection
- charts, infographics or other ways to visualize your information
- a video, either of your work, or a reflection video
- an audio podcast. Record a story about your experience
- a photo report with a small post

Helpful tools:

- Canva - a good way to visualize your data
- QuickTime - an easy way to capture your screen, record video or audio

Examples of ways to share your experience:

- Facebook - create and share a post that will inspire your friends
- YouTube - upload an exciting video
- GitHub - share and describe your solution
- Telegraph - create a post that you can easily share on Telegram
- Instagram - share photos and stories from ucode. Don't forget to tag us :)

Share what you've learned and accomplished with your local community and the world. Use #ucode and #CBLWorld on social media.