

COSC 101 Homework 3

Spring 2016

Due date: **Friday, February 26, 5:00 p.m.**

The first program will improve your skills with string indexing.

The other programs will improve your skills creating functions with `for` loops and `if` statements.

Your task is to complete three programs as described below:

- `hw3_email.py`
- `hw3_farm.py`
- `hw3_leap.py`

Upload on moodle by the deadline the three python files `hw3_email.py`, `hw3_farm.py` and `hw3_leap.py` by zipping a folder named `username_hw3` which contains them. See me if you are unsure how to do a zip.

Program 1: Email Address Validator

Valid internet email addresses must have a particular format, their sequences of characters must follow a specific pattern. Forms on the web often check if email addresses are valid. This exercise imitates such validation, but is a simplification of reality.

In the file `hw3_email.py` you will write a program that takes apart a string to determine the validity of an email address.

A valid email address consists of a *local* part, followed by `@`, followed by a *domain* part. We define an email address to be valid if the string has the following three properties.

1. One `@` is present (zero or more than one `@` are invalid email addresses).
2. At least one character forms the local part.
3. At least one character forms the domain part.

For example:

String	Valid?	local	domain
<code>user@localserver</code>	Yes	<code>user</code>	<code>localserver</code>
<code>niceandsimple@example.com</code>	Yes	<code>niceandsimple</code>	<code>example.com</code>
<code>Abc.example.com</code>	No	---	---
<code>A@b@c@example.com</code>	No	---	---
<code>@host</code>	No	---	---

Your program first asks the user to enter an email address and then returns if the string entered is a valid email address or not.

If it is valid the program should also print the following information:

- the local part,
- the domain part,
- if it is from the colgate domain, i.e., the domain part is equal to `colgate.edu`

For example:

Think about how to break down the problem into manageable parts so as to implement and test one at the time. You should build your program incrementally.

You can use two `for` loops (the second can be useful to determine the top-level string to print). You are not required to use string indexing but we recommend using string indexing as it makes the program easier to write.

Your program's output should look **exactly** like the examples provided below.

```
Enter an email: ef@col@gate.edu
Invalid email
```

```
Enter an email: @colgate.edu
Invalid email
```

```
Enter an email: joe@
Invalid email
```

```
Enter an email: joe
Invalid email
```

```
Enter an email: efourquet@colgate.edu
Valid email
Local part: efourquet
Domain part: colgate.edu
From COLGATE!
```

```
Enter an email: e@colgate.com
Valid email
Local part: e
Domain part: colgate.com
```

```
Enter an email: efourquet@colgate.ca
Valid email
Local part: efourquet
Domain part: colgate.ca
```

```
Enter an email: e@colgate
Valid email
Local part: e
Domain part: colgate
```

Program 2: Farm song

Write a program `hw3_farm.py` that prints the lyrics of the song [Old MacDonald](#).

This program should have a similar structure to the first program: you will define a few functions including one called `main`. At the end of the program, you should have a call to `main`.

The program first asks the user the name of the farmer and the number of the various animals the farmer keeps on the farm. Then the user is prompted to enter the name for each animal type and the noise it makes. The program prints the lyrics of the song accordingly.

Your program should use at least two functions. One function that collects the animal types and noises and one function that builds a single verse of the song. **Lists should not be used in your solution.**

An example of the program execution is as follows:

```
What is the farmer name? Maturin
How many different animals does Maturin keeps on the farm? 2
```

```
Enter the name of animal 1: cow
Enter the noise of animal 1: moo
Enter the name of animal 2: duck
Enter the noise of animal 2: quack
```

```
Maturin had a farm, E-I-E-I-O.
And on that farm he had a cow, E-I-E-I-O.
With a moo moo here and a moo moo there
Here a moo, there a moo, everywhere a moo moo
Maturin had a farm, E-I-E-I-O.
```

```
Maturin had a farm, E-I-E-I-O.
And on that farm he had a duck, E-I-E-I-O.
With a quack quack here and a quack quack there
Here a quack, there a quack, everywhere a quack quack
Maturin had a farm, E-I-E-I-O.
```

In the above example, the user entered six inputs Maturin, 2, cow, moo, duck, and quack. Your program must match the format above *exactly*.

Program 3: Leap Year

Your task is to complete the program `hw3_leap.py`. Included with in the .zip file for this homework is a *partial* implementation of this program. Your job is to finish it.

This program should ask the user for a range of years and then prints for each year within that range if it is a leap or a normal year. Since 1582, a [leap year](#) (click to get the algorithm) occurs according to the following formula: a leap year is divisible by four, but not by one hundred, unless it is divisible by four hundred.

First complete the definition of the function `is_leap_year(y)` that returns `True` if the the parameter `y` is a leap year and `False` if `y` is a normal year. Test this function on its own to make sure it is correct.

Now complete the function called `main()` that takes no arguments. Instead, it uses `raw_input` to ask the user for a year range and first checks that the range starts at or after 1582. If the range is valid, your program should output the type for each year in ascending order using the `is_leap_year` function you wrote. Important detail: you must handle the case when the user enters the years “out of order” in the sense that the first input is the end of the range (the later year) and and the second input is the start of the range (the earlier year).

Finally, at the very end of `hw3_leap.py` you should see that the main function is called. Thus, when a user runs your program, they will run the main function.

Your program should reproduce the following three executions. In the first exexecution the user enters 2008 followed by 2024.

```
Enter a year: 2008
Enter a second year: 2024
```

```
2008 is a leap year
2009 is a normal year
2010 is a normal year
2011 is a normal year
2012 is a leap year
2013 is a normal year
2014 is a normal year
2015 is a normal year
2016 is a leap year
2017 is a normal year
2018 is a normal year
2019 is a normal year
2020 is a leap year
2021 is a normal year
2022 is a normal year
2023 is a normal year
2024 is a leap year
```

In this example, the user enters the years “out of order” but the program still correctly prints the range.

```
Enter a year: 2400
Enter a second year: 2392
```

```
2392 is a leap year
2393 is a normal year
2394 is a normal year
2395 is a normal year
2396 is a leap year
2397 is a normal year
2398 is a normal year
2399 is a normal year
2400 is a leap year
```

Lastly, the user enters an invalid range.

```
Enter a year: 1000
Enter a second year: 1004
The range must start after or at 1582
```