Expressions and Control Structures

About Tutorials

The aim of the CSSE1001/CSSE7030 tutorial (T) sessions is to give additional practice at writing computer programs. The tasks are designed to be completed by a group of two or three students working together, so it is strongly recommended that you sit with other students, and use a lab machine instead of a laptop. Making use of the course notes and lecture material is also encouraged. If you are on a lab machine ensure that you open up the Python 3.6 interpreter, **IDLE 3.6**, and not one of the other Python interpreters that may be installed on the system.

Tutors are present at the session to provide assistance and insight into the tasks, so ask them questions when you get stuck. Note that tutors will not provide assistance for the assignment tasks in the T sessions; for that, you should attend any of the timetabled P sessions.

Attendance at these sessions is not marked, however you will miss out on learning opportunities by not attending. Students often find that the learning experiences from the tutorials are directly helpful in completing the assignments and the final exam.

Hello, World!

The purpose of this task is to become familiar with writing, saving, and running programs from IDLE. Write a program which, when run, outputs the text `Hello, World!`, and save it as `hello.py`.

Then, close IDLE, and navigate the file-system (in Windows Explorer or equivalent) to find the program you just wrote, open it in IDLE, and run it.

Next, modify the program to first ask the user what their name is (using `input`), and then output a greeting using their name. For example (the blue text is output from the program, the black text is input from the user):

```
>>>
 RESTART: H:/CSSE1001/Python Code/Week2/hello.py
What is your name? Sir Lancelot of Camelot
Hello, Sir Lancelot of Camelot!
>>>
```

Hint:

- Create a Python source code file by selecting "New" from the File menu in IDLE, or using the Ctrl-N shortcut.
- Save the file in a location where you will be able to find it. On the lab machines, the `H:` drive is your personal network drive, which can be accessed from any EAIT machine (`U:` is the equivalent of the science `H:`). Files saved to the `C:` drive might not be accessible after logging off.
- By default, IDLE will **not** add a file extension to your files. You must manually add `.py` to the end of the filename in order to keep the syntax highlighting.

Guess the Expression

For each of the below expressions, make a guess on what the result of the expression should be. Discuss your guesses with the other people at your table, or with a tutor, **before** running the code in IDLE. After running the code, if you didn't guess correctly, discuss why it produced the result that it did, or experiment with some other examples yourself.

```
>>> 3--2

>>> 5 / 2

>>> -5 / 2

>>> 5 // 2

>>> -5 // 2

>>> 5.0 / 2

>>> 5.0 // 2

>>> 3.0 - 5 / 2

>>> 3.0 - 5 // 2

>>> '3' + '12'

>>> 3 + '12'

>>> 3 * '12'

>>> 'ba' + 'na' * 2

>>> 'hello' - 'o'

>>> 'ababab' / 3

>>> 2 ** 3

>>> 9 ** 1/2

>>> 11 % 4

>>> -11 % 4

>>> 'Tim' > 'Tom'

>>> 'A' < 'a'
```

Leap Years

A leap year occurs when the year is a multiple of four, unless the year is a multiple of 100. However, if the year is a multiple of 400, then it is a leap year. For instance, 2016 and 2000 are leap years, but 1900 is not.

Create a new Python source file, and write a program `leap_year.py` which asks the user to enter an integer (representing a year), and prints `True` if the year is a leap year, and `False` if not.

```
>>>
 RESTART: H:/CSSE1001/Python Code/Week2/leap_year.py
Please enter a year: 2018
False
>>>
 RESTART: H:/CSSE1001/Python Code/Week2/leap_year.py
Please enter a year: 2020
True
>>>
 RESTART: H:/CSSE1001/Python Code/Week2/leap_year.py
Please enter a year: 2100
False
>>>
 RESTART: H:/CSSE1001/Python Code/Week2/leap_year.py
Please enter a year: 2000
True
>>>
```

> **Challenge 1: Days of the Week**
>
> Research Zeller's congruence, and write a program which takes as input, the day of the month, the month, and the year, and prints the day of the week corresponding to that date. Design a set of example cases to check you've implemented the logic correctly.

Powers of Two

Write a program that takes a positive integer as an input. Output a simple table that lists the powers of two up to the integer value passed to the program. The output should look as follows:

```
>>>
 RESTART: H:/CSSE1001/Python Code/Week2/powers_of_two.py
Enter the power: 0
0 1
>>>
 RESTART: H:/CSSE1001/Python Code/Week2/powers_of_two.py
Enter the power: 1
0 1
1 2
>>>
 RESTART: H:/CSSE1001/Python Code/Week2/powers_of_two.py
Enter the power: 3
0 1
1 2
2 4
3 8
>>>
```

Guess a Number

Write a program that asks a user to guess a number between 1 and 100. The program should generate a random number and repeatedly ask the user to input a number until they guess the correct number. When they guess the correct number, the program should output `Congratulations!` and then terminate. The user may give up by entering -1 and the program will output the number the user was trying to guess and then terminate. The output of the program, if run in IDLE, should look like:

```
>>>
 RESTART: H:/CSSE1001/Python Code/Week2/guess_number.py
Try to guess the number I am thinking of between 1 and 100.
Please enter your guess: 23
Sorry that is not correct.
Please enter your guess: 42
Congratulations! You have guessed correctly.
>>>
 RESTART: H:/CSSE1001/Python Code/Week2/guess_number.py
Try to guess the number I am thinking of between 1 and 100.
Please enter your guess: 42
Sorry that is not correct.
Please enter your guess: -1
The number you were trying to guess was 2.
>>>
```

Hint:

To generate random integers, first put `import random` at the top of the file, then use the function call `random.randint(1, 100)` to generate a random integer between 1 and 100.

```
>>> import random
>>> random.randint(1, 100)
73
>>> x = random.randint(1, 100)
>>> x
22
```

You can also type `>>> help(random.randint)` at the prompt to get more information. Type `help(random)` or `dir(random)` to find more functions in the random module, or look at the online documentation.

Alice's Addition Academy

Your 5-year old cousin Alice is learning addition, and you have decided to write a program she can use to practice with. The program will generate two random 2-digit integers, and ask the user what the sum of the two integers is, and display whether or not the user is correct. If the user is incorrect, also display the correct answer.

Some example interactions with the program are as follows:

```
>>>
 RESTART: H:/CSSE1001/Python Code/Week2/arithmetic_academy.py
70 + 17 = 87
Correct!
>>>
 RESTART: H:/CSSE1001/Python Code/Week2/arithmetic_academy.py
99 + 56 = 156
No, the correct answer is 155
>>>
```

Challenge 2: Advanced Academy

Modify the above program so that it asks a set of 10 questions, rather than just one question.

Challenge 3: Advanced Academy (Hard!)

Modify the above program so that it also randomly selects an operator from the six options `+ - * // ** %`, to test Alice's abilities with all of them. If the `**` operator is being used, the exponent should be randomly chosen between 0 and 3, instead of between 10 and 99.