Tutorial Week 5

Recommend that your finish any remaining while loop exercises from last week before starting the following.

Week 5 - Experiment with ***for*** loops and handling **exceptions**. Make sure you know when it is best to use *while* or *for* loop.

1. Write a program to print the times table for a number entered by the user. Example Output:

```
Enter the number to print the tables for: 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

2. Write a program that uses a *for* loop to read in 5 numbers and output the total.
3. Write a *for* loop that will output all odd numbers between 0 and 50.
4. Write a program which asks the user how many stars are required and then make it display that number of stars across the screen.
5. Write a program that simulates the rolling of two six-sided dice and then counts the number of times that the dice generate a double (e.g. both dice shown the same value). Roll the dice 100 times and then print the number of doubles. Example output., Out of 100 you rolled 20 doubles
6. **Nested for loops** - A loop within a loop.
   Try the following program and check you understand how the program runs the nested loop code.

```
for number in range(3) :
    print("----------------------------------------")
    print("Outer loop iteration " + str(number))
    # Inner loop
    for another_number in range(4):
       print("**************************")
       print("In inner loop iteration " + str(another_number))
```

7. A) Try the following nested for loop.

```
for x in range(1,4):  # print 3 rows
   for y in range(1,4):  # 3 asterisks in each row
      print('*', end='')
   print()
```

   The output is:

```
***
***
***
```

   B) Amend the program so that it prints the following:

```
*
**
***
```

8. Exception Handling in Python

A) When we think that some commands may fail, we create an exception handler for the errors we expect using the try…except construct.

```
try:
    Python commands
except:
    Exception handler
```

The above construct does not specify a specific exception (error) to handle, so the exception handler is user for **any** errors that occur while executing the Python commands.

```
try:
    Python commands
except ExceptionType:
    Exception handle
```

If the exception handler only handles certain exceptions (specified in the ExceptionType) and the program produces an exception that the handler wasn't written to deal with, Python's default exception handler will handle that error.

In the following example we ask the user to enter an integer number and cast the string input from the input() method to an integer. If the input entered is not been an integer and cannot be converted (cast) to an integer it will generate a ValueError.

B) For example, in the following the user enters 10.5 instead of a valid integer.

```
>>> n = int(input("Please enter a number: "))
Please enter a number: 10.5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '10.5'
```

With the aid of exception handling, we can write robust code for reading an integer from input:

```
n = input("Please enter an integer: ")
try:
   n = int(n)
except ValueError:
   print("Requires a valid integer!")
```

Example: The program accepts a value from the user. In the try clause the value in n is converted (cast) to an integer. If an exception occurs (the value cannot be converted to an integer) the except clause will be executed. The error, in this case a ValueError, has to match the name after except.

C) This code would be more useful in a loop:

```
while True:
    n = input("Please enter an integer: ")
    try:
       n = int(n)
       break
    except ValueError:
       print("Requires a valid integer! Please try again.")
print("You successfully entered an integer.")
```

Example output:

```
Please enter an integer: r
Requires a valid integer! Please try again.
Please enter an integer: 10.6
Requires a valid integer! Please try again.
Please enter an integer: 5
You successfully entered an integer.
```

9. Test what type of error the following will produce.

```
>>> x = 2/0
```

- Then create a try except construct that will give the message ‘Cannot divide by zero’ when appropriate.