Homework 4: Looping statements

CS-UY 1114 NYU Tandon

Your submission for this assignment must be via Gradescope.

Submit your solution to all problems in a single file named *YourNetID*`_hw4.py`

Your submitted file must start with the mandatory comment header, containing your name, the class number, the date, and the homework number.

Your submitted code may use only the material that we've covered so far this semester.

Problem 1
Consider the following short programs. Without using a computer, determine what their output will be. Write what each program will **print** in a comment in your code file.

1.
```
i=0
j=0
while i<5:
    i+=1
    j+=i
    print(j*"@")
```

2.
```
for x in range(0,5):
    for y in range(x, 0, -1):
        print(x, y)
```

3.
```
foo = ""
x = 0
while x < 10:
    foo += str(x)
    x += 3
print(foo)
```

Problem 2
Consider the following function:

```
def contains_two_fives(n):
    """
    sig: int -> bool
    Give a four-digit positive decimal integer,
    the function will return True if the number
    contains at least two fives, or False otherwise.
    """
    if n % 10 == 5 and n % 100 // 10 == 5:
        return True
    if n % 10 == 5 and n % 1000 // 100 == 5:
        return True
    if n % 10 == 5 and n % 10000 // 1000 == 5:
        return True

    if n % 100 // 10 == 5 and n % 1000 // 100 == 5:
        return True
    if n % 100 // 10 == 5 and n % 10000 // 1000 == 5:
        return True

    if n % 1000 // 100 == 5 and n % 10000 // 1000 == 5:
        return True

    return False
```

The function determines if a four-digit number contains at least two fives. For example:

- `contains_two_fives(1234) == False` because the number contains no fives
- `contains_two_fives(1534) == False` because the number contains only one five
- `contains_two_fives(1535) == True` because the number contains exactly two fives
- `contains_two_fives(5505) == True` because the number contains at least two fives

As you can see, the function uses modulus and division to extract each digit individualy. For example, the expression `n % 100 // 10` evaluates to the second least-significant digit in `n`. Read the program and make sure you underestand how it works.

The function is correct, however it is too verbose and repetitive. A lot of code in it is nearly the same thing, repeated. As programmers, we strive to avoid repetitive code. This principle is referred to as *DRY*: "Don't repeat yourself."

Rewrite the function `contains_two_fives` so that the code is less repetitive. Use a loop. Your improved version should behave exactly the same as the above version; that is, given the same input, it should produce the same result.

Hint: try using a loop to look at the least-significant digit of `n`, then remove that digit, and repeat until `n` becomes zero. Count the number of fives that you encounter.

Your professor's solution is only six lines. Can you do as well? Even better, can you ensure that your function will work for any arbitrary positive integer `n` (i.e. potentially more than four digits)?

Problem 3

Using a **`while`** loop, write a function `multiples_of_three_while` that takes a positive integer parameter n, and then prints the first n multiples of 3 (starting from 3).

The header of the function is as follows:

```
def multiples_of_three_while(n):
    # sig: int -> NoneType
```

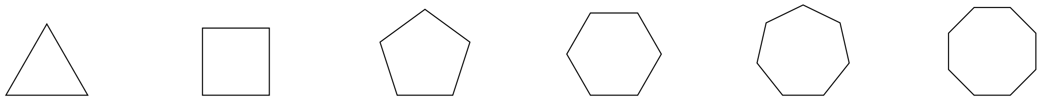
For example, running `multiples_of_three_while(3)` should give:

```
3
6
9
```

Problem 4
Write a function `multiples_of_three_for`, which is identical to the previous function, but use a **`for`** loop, instead of a **`while`** loops. Your function should behave identically.

The **`for`** `i` **`in range`** syntax may be helpful.

Problem 5
A **regular polygon** is a shape with a given number of sides, where each side has the same length, and where all the angles between the sides have the same measure:



Write a function named `polygon` that will draw a regular polygon using `turtle`. It should takes two integer parameters: `size`, specifying the length of a side of the polygon; and `sides`, specifying the number of sides.

Recall that the angles of a regular polygon must sum to 360.

The header of the function is as follows:

```
def polygon(size, sides):
    # sig: int, int -> NoneType
```

Problem 6
Write a function `hourglass` that takes a positive integer parameter n, and then, using **`print`** statements, draw an hourglass shape of $2n$ lines, made of asterisks.

For example, the call `hourglass(4)` should display the following:

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

Some hints:

- To make the hourglass shape, you'll have to print an appropriate number of spaces (to indent the stars), followed by an appropriate number of asterisks. If you know how many spaces and how many stars, you can use a statement like the following:

  ```
  print(numspaces * " " + numasterisks * "*")
  ```

- You'll have to carefully consider how many spaces and how many asterisks to print at each line. What is the relationship between n and the number of asterisks printed at the widest point? What is the relationship between the numbers of asterisks at a given line and the number of asterisks at the next line? What is the relationship between the number of spaces an a given line and the number of spaces at the next line?

It may be helpful to draw an hourglass by hand for $n = 5$ in order to help visualize the mathematical relationships of the various parts.

- Because of the symmetric nature of the picture (i.e. it gets narrower, then wider), you may want to consider using two separate loops: one for the narrowing part, another for the widening part. First write one, then the other.

Problem 7
Write a function named `guessing_game` that implements a number-guessing game.

The computer will randomly select a secret number between 1 and 100. The player (that is, the user) will then be prompted to guess the secret number. They will be given the opportunity to enter a guess until they guess the correct number. Each time the user guesses, the computer will say whether the guess was too low, too high or exactly right. If the user enters a non-number, an appropriate message will be displayed, and the user will be prompted again.

When the player guesses the secret number correctly, the game is won.

Here is an example of how the game is played:

```
I'm thinking of a number between 1 and 100, try to guess it.
Please enter a number between 1 and 100: what?
Sorry, but "what?" is not a number between 1 and 100.
Please enter a number between 1 and 100: 50
your guess was too low, try again.
Please enter a number between 1 and 100: 75
your guess was too low, try again.
Please enter a number between 1 and 100: 88
your guess was too low, try again.
Please enter a number between 1 and 100: 94
your guess was too high, try again.
Please enter a number between 1 and 100: 91
your guess was too high, try again.
Please enter a number between 1 and 100: 90
Congratulations, you guessed the number!
```

Some suggestions:

1. At the beginning of your function, it should print an introductory message (“`I'm thinking of a number between 1 and 100, try to guess it.`”) and choose the secret number.

   A secret number between 1 and 100 can be chosen randomly by calling the function `random.randint (1 , 100)`. Each time this function is called, it will return a random number between 1 and 100. You should call this function only once, and store the result into a variable.

   Don’t forget to include the `import random` statement at the beginning of your code file or else you won’t be able to use `random.randint`.

2. The main loop part of your function will be constructed as a **`while`** loop. You must carefully design the loop’s condition so that the loop ends when the user guesses the secret number correctly.

In the loop's body, you will ask the user to enter a number and compare their response to the secret number. To get the number form the user, you should use the `input` function. You can use the `isdigit` function to make sure that what the user types is a valid number. For example:

```
v = input("Enter a number: ")
if not v.isdigit():
    print("not a good number")
```

3. The main loop ends when the user guesses the secret number. In the coda, you can print a congratulatory message.

Problem 8
Write a function `mul_table` that will print out a table with 5 rows and 10 columns, such that the value of the cell at row i, column j is the integer j^i. For example, the first row should read, from left to right, 1^1, 2^1, 3^1, 4^1, etc; the second row should read 1^2, 2^2, 3^2, etc. Your program's output should look like this:

```
1       2       3       4       5       6       7       8       9       10
1       4       9       16      25      36      49      64      81      100
1       8       27      64      125     216     343     512     729     1000
1       16      81      256     625     1296    2401    4096    6561    10000
1       32      243     1024    3125    7776    16807   32768   59049   100000
```

You must use *nested loops* in your function; that is, you will need one loop for rows, containing a loop for columns.

Your function takes no parameters are return `None`.

Hint: to get the numbers nicely lined up in columns, use tabs. Printing the special character `"\t"` will move the cursor to the next 8-character column, so you don't need to count spaces.