1. **[14 points]** Warming up

   (a) **[2 points]** Briefly indicate the respective audiences for docstrings and inline/block comments (i.e. after the # symbol)?

   Docstrings are intended for programmers who will use the (your) functions, but not necessarily read the underlying code, while inline/block comments (starting with #) are for programmers who are reading the underlying code.

   (b) **[4 points]** Quick coding: Write a function `twodice` that returns an integer simulating the sum of two dice rolled simultaneously, for example:

```
>>> twodice()
10
>>> twodice()
2
```

```
from random import randint
def twodice():
    return randint(1,6) + randint(1,6)
```

   (c) **[8 points]** Quick coding: Write two functions, one using a `for` loop and the other using a `while` loop, to print every other number from 10 to 0, starting with 10, one number per line.

```
def forloop():
    for i in range(10, -1, -2):
        print(i)
```

```
def whileloop():
    i = 10
    while i >= 0:
        print(i)
        i -= 2
```

2. [**12 points**] Slice and dice

   Given the variables `s` and `t` with the following values:

```
>>> s
'simon says'
>>> t
'jump around'
```

   Evaluate the following expressions and provide the resulting value.

   (a) `t[:5] + s[:5]`

```
'jump simon'
```

   (b) `s[:6] + s[-9:-11:-1] + s[5:8] + t[-1]`

```
'simon is sad'
```

   (c) `((t.capitalize() + "! ") * 3).strip()`

```
'Jump around! Jump around! Jump around!'
```

   (d) `(t.split()[0] + " " + t[1:4:2] + "!").upper()`

```
'JUMP UP!'
```

3. [**8 points**] Function calls

Consider the following Python code:

```
def bar(s):
    print(s)
    return len(s)

def foo(s):
    r = 1
    for i in range(len(s)):
        bar(s[i:])
        r = r * (-bar(s[i:]))
    return r

y = foo("test")
```

After execution what is the value of `y` and what, if anything, is printed in the console?

Printed in the shell:

```
test
test
est
est
st
st
t
t
```

After execution `y` will be 24.

4. [**14 points**] Coding

   Write a function called `swap_every_pair` that "encrypts" a string by swapping every pair of letters. For example:

```
>>> swap_every_pair("banana")
'abanan'
>>> swap_every_pair("Hello")
'eHllo'
```

```
def swap_every_pair(message):
    result = ""
    for i in range(1, len(message), 2):
        result += message[i] + message[i-1]
    if len(message) % 2 == 1:
        result += message[-1]
    return result
```

5. **[14 points]** T/F

For each of the statements below state whether they are true or false.

True ___ If `s` is an even-length string, `s[len(s)//2:]` evaluates to the second half of the string.

True ___ `7%3 + 5//2 + 7/4` evaluates to `4.75`.

False ___ `"hello".capitalize().replace("E", "A").lower()` would evaluate to `"hallo"`.

False ___ `[["a", "b", "c"], "def"][1][1]` evaluates to `"b"`.

True ___ The following two functions do the same thing:

```
def f(num):
    if num // 2 < 10:
        return False
    else:
        return True

def g(num):
    return num >= 20
```

True ___ The following function would return the last line in a non-empty file:

```
def last_line_of_file(filename):
    with open(filename, "r") as file:
        for line in file:
            last_line = line
        return last_line
```

True ___ In Python all `for` loops can be replaced with an equivalent `while` loop.

6. [**12 points**] We've got problems

(a) [**6 points**] The function below has two boolean parameters, `a` and `b`. The function works correctly, however, it is too long and it uses bad coding style. Rewrite the function in good style to be as concise as possible.

```
def could_be_better(a, b):
    if a == False and b == False:
        return True
    if a == False and b == True:
        return False
    if a == True and b == False:
        return False
    if a == True and b == True:
        return True
```

```
def better(a, b):
    return a == b
```

(b) [**6 points**] The following function was designed to return a `bool` indicating if two uppercase strings of DNA bases ("A", "C", "G", "T") are identical (defined as have same length and DNA bases). Sometimes we don't know the DNA base and so use "N" to stand for "any base", that is "N" is considered identical to any of "A","G","C","T". Describe the two different situations where this implementation will produce an invalid result or even an error (that is Python would stop and report an error). *Note*, your two situations should highlight different problems not variations of the same issue.

```
def is_identical(dna1, dna2):
    i = 0
    while i < len(dna1) or i < len(dna2):
        if dna1[i] != "N" and dna2[i] != "N" and dna1[i] != dna2[i]:
            return False
        i += 1
```

i. If `dna1` or `dna2` is a substring of the other we will get an index error because `i` will increment beyond the length of the shorter string. It should return `False` in this context.

ii. If `dna1` and `dna2` are identical it will return `None` instead of `True` since the final return statement is missing.

7. [**16 points**] Coding, take 2

Write a Python function named `equalizer` that will identify students who should switch sections to equalize the number of students in two sections of a class. The section rosters will be provided as text files with one student name per line (as Lastname, Firstname). Your function should have two parameters, the filenames for the section 1 and section 2 rosters, and should print the names of students who should switch and the direction as shown below. You can pick the specific students in any way you want. If the sections have an equal number of students, print "No switching required".

```
>>> equalizer("sectionA.txt", "sectionB.txt")
Smith, John switch to section 1
```

```
def file_to_list(filename):
    names = []
    with open(filename, "r") as file:
        for line in file:
            names.append(line.strip())
    return names

def print_switches(names1, names2):
    delta = len(names1) - len(names2)

    if (delta < -1):
        for i in range((-delta) // 2):
            print(names2[i], "switch to section", 1)
    elif (delta > 1):
        for i in range(delta // 2):
            print(names1[i], "switch to section", 2)
    else:
        print("No switching required.")

def equalizer(section1, section2):
    names1 = file_to_list(section1)
    names2 = file_to_list(section2)
    print_switches(names1, names2)
```

8. [**10 points**] Turtle fun

```
from turtle import *

def shape(side):
    forward(side)
    for i in range(5):
        right(90)
        forward(side)

base = 0
side = 50
while side < 400:
    shape(side)
    current = side + base
    base = side
    side = current

done()
```

Draw the image that would be created by the above code. Assume that the turtle is initially at the origin, facing right.