True/False (10 Q, 20 points, 12 min)

1. A, T
2. B, F
3. A, T
4. A, T
5. B, F
6. A, T
7. B, F
8. B, F
9. B, F
10. B, F

Short Answers (8 Q, 50 points, 32 min):

1. (6 points), write printed result

```
import math
print("Pi is " + str(float("{0:.1f}".format(math.pi))))
```

Should print:

```
Pi is 3.1
```

2. (6 points) Reasoning over code (string, list), come up with input that returns True

```
def Func(number):
      total = 0
      while number > 0:
            total = total + number*(number-1)
            number = number - 1
      return total == 40
```

Answer: `5`

3. (6 points) Give three reasons for writing functions instead of one long python script.

Sample answers:

- Modular/does one thing
- Easier to debug
- Code security (don't access what you don't need)
- Easier to share (can call in another file)
- Readability
- Reusability, prevent repeated code
- Parameterization allows for abstraction (treat implementation as a black-box)

4. (6 points) Code Tracing [List], write printed result

```
A = [1,2,3]
B = [ ]
for element in A:
      B.append(element)
A = B + [5]
B[1] = 4
print(A)
print(B)
```

Answer:

```
[1, 2, 3, 5]
[1, 4, 3]
```

5. (6 points) Code Tracing [String], write printed result

```
s = (“abcde” * 2).replace(“e”, “b”)
while (s.startswith(“b”) or s.endswith(“b”)):
      print(len(s))
      s = s[1:-1]
print(s)
```

Answer:

```
10
8
cdbabc
```

6. (7 points) Code Tracing [String], write printed result

```
def ct1(s, p):
      result = ””
      for character in s:
            new = ord(character) + p
            result += chr(new) + result
            p += 1
      return result

print(ct1(“abcd”, 2))
```

Answer:

```
igec
```

7. (6 points) String decompression, fill in the blank

```
def decompress_string(s):
    decompressed_string = ""
    num = “”

    for c in s:
        if c.isdigit():
            num = num + c
        else:
            if num == “”:
                  new_string = c
            else:
                  new_string = c * int(num)

             decompressed_string += new_string
             num = “”

    return decompressed_string
```

8. (7 points) Code Tracing, write printed result

```
def f(x):
    while (x > 0):
        print(x)
        for y in range(1, x):
            print(y)
            if (y % 3 > 0):
                print("A")
            else:
                return("C")
      y -= 1
      x -= 1
    return(‘B’)
print(f(5))
```

Ans:

```
5
1
A
2
A
3
C
```

Writing Code (2 Q, 30 points, 26 minutes):

Slicing Strings

a) **[12 POINTS]** Write a function generate_sliced_string_list(s) which takes some string s (composed only of characters from the alphabet) and generates a list of strings such that the first string is composed of the last character of s, the second string is composed of the last two characters of s (in their original order), and so on. The last string in this list is the full string s.

```
def generate_sliced_string_list(s):
      result = []
      for i in range(len(s)-1,-1,-1):
            result.append(s[i:])
      return result
```

b) **[8 POINTS]** Write a function, write_sliced_strings_to_file(s, filename), which takes a string s and then writes to the file called filename such that:

The first line written to filename will be s itself
The second line will contain all the characters in s from s[1] onward (omitting s[0])
The third line will contain all the characters in s from s[2] onward (omitting s[0] and s[1])
And so on, until
The second to last line will contain the last two characters of s (in order their original order)
The last line will contain the last character of s

The function itself will not return anything.

For example, upon running write_sliced_strings_to_file(“monty”, “my_solution.txt”), my_solution.txt would look like:
monty
onty
nty
ty
y

```
def write_sliced_strings_to_file(s, filename):
      f = open(filename,“w”)
      #call the function in previous part and reverse it:
      for i in generate_sliced_string_list(s)[::-1]:
            f.write(i+“\n”)
            f.close()
```

Fibonacci sequence **[10 POINTS]**

The fibonacci sequence is a sequence of numbers such that the first number is 1, the second number is 1, and every number thereafter is equal to the sum of the previous two numbers in the sequence. The first eight numbers in the sequence are: 1, 1, 2, 3, 5, 8, 13, 21

Write a function, create_fibonacci_sequence(n) which will generate a list composed of the first n elements of the fibonacci sequence and return that list.

For example, create_fibonacci_sequence(8) would return [1, 1, 2, 3, 5, 8, 13, 21]

```
def create_fibonacci_sequence(n):
      if n<=0:
            return []
      elif n==1:
            return [1]
      else:
            result = [1,1]
            for i in range(2,n):
                  result.append(result[-1] + result[-2])
            return result
```

Bonus (5 points)

Ada wrote a function: `do_integer_division_on_list(L,n)` that takes a list L and a number n, performs integer division on each element of the input list L, and then returns L.

For example, `do_integer_division_on_list([3,6,9],3)` will return [1,2,3].

Ada writes the following code:

```
A = [3, 6, 9]
B = do_integer_division_on_list(A,3)
C = do_integer_division_on_list(A,3)
print(C)
```

However, the print statement prints [0,0,1], rather than [1,2,3], which is what she was expecting.

Explain what is happening and how to fix it. (No need to write code)

Reason: `do_integer_division_on_list` is destructive. When calling it for the first time, it changes A to [1,2,3].

Fix: (Many ways)

Make a deep copy before doing the integer division to keep A untouched.