Answer: Answers, inline, preceded by red boxes. See exam for full questions and formatting.

Mock Final Exam
CSci 127: Introduction to Computer Science
Hunter College, City University of New York

Fall 2025

1. (a) Fill in the code below to produce the output on the right:

```
day_string = "Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday"
```

	Python Code:	Output:
i.	`print(day_string[ ☐ ])`	`Mon`

ii.	`counts = {}`
	`for c in day_string.lower():`
	`  if ☐ :`
	`    counts[c] = counts[c]+1`
	`  else:`
	`    counts[c]=1`
	`print("t appears", counts['t'], "times.")`
		`t appears 3 times.`

iii.	`days_list = day_string. ☐`
	`print(days_list[-1].upper())`
		`SUNDAY`

iv.	`short_days = [ ☐ for d in days_list]`
	`print(short_days[-2:])`
		`['Sa', 'Su']`

v.	`weekdays = days_list[ ☐ ]`
	`print(len(weekdays), "weekdays.")`
		`5 weekdays.`

Answer:

```
day_string = "Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday"
print(day_string[:3])
counts = {}
for c in day_string.lower():
    if c in counts:
        counts[c] = counts[c]+1
    else:
        counts[c] = 1
print("The letter t appears", counts['t'], "times.")
days_list = day_string.split(',')
print(days_list[-1].upper())
short_days = [d[:2] for d in days_list]
```

```
print(short_days[-2:])
weekdays = days_list[:5]
print("There are", len(weekdays), "weekdays.")
```

(b) The commands below are **run sequentially**, what is the output after each has run:

```
$ ls
baruch.png    ccny.mp4        hunter.png    queens.png
$ pwd
/tmp/mock
```

i.
```
$ mkdir pix
$ mv *.png pix
$ ls
```

Answer:

```
ccny.mp4        pix
```

ii.
```
$ ls | grep cc
```

Answer:

```
ccny.mp4
```

iii.
```
$ cd pix
$ echo "Picture folder:"
$ pwd
```

Answer:

```
Picture folder:
/tmp/mock/pix
```

iv.
```
$ cp ../ccny.mp4 ccny2.mp4
$ ls | wc -l
```

Answer:

```
4
```

2. (a) Check all that apply:

Answer:

i. What color is `tess` after this command? `tess.color("#ABABAB")`
☐ black ☐ red ☐ white ✓ gray ☐ green

ii. Select all the **even** binary numbers:
☐ 1011 ☐ 1101 ☐ 0111 ✓ 1010 ✓ 1110

iii. Select the hexadecimal number **larger than 160**:
✓ AA ☐ 11 ✓ FF ☐ 55 ✓ DD

(b) Fill in the code to produce the output on the right:

i.
```
nums = [ 1, 4, 9, 16, 25, 36, 49, 64]
```

Answer:

```
for i in range(  1, 5  ):
    print(nums[i], end=" ")
```

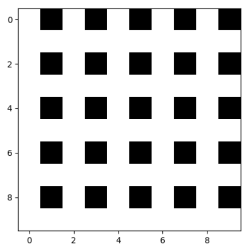
Output:

```
4 9 16 25
```

ii. **Answer:**

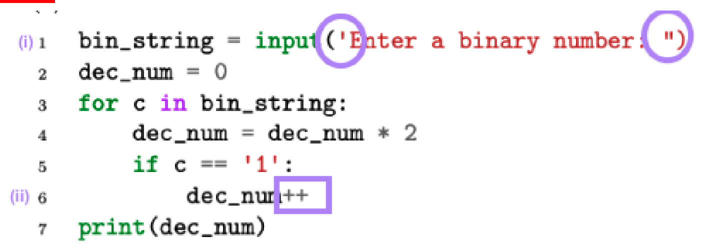
```
import numpy as np
import matplotlib.pyplot as plt
img = np.ones( (10,10,3) )

img[0::2,1::2,:] = 0
plt.imshow(img)
plt.show()
```

Output:



(c) Consider the code:

Answer:



```
(i)  1  bin_string = input('Enter a binary number: ")
     2  dec_num = 0
     3  for c in bin_string:
     4      dec_num = dec_num * 2
     5      if c == '1':
(ii) 6          dec_num++
     7  print(dec_num)
```

i. **Circle** the code above and mark line with **(i)** that caused this error:

```
    bin_string = input('Enter a binary number: ")
                       ^
SyntaxError: unterminated string literal (detected at line 1)
```

Write the code that would fix the error:

Answer:

```
bin_string = input("Enter a binary number:")
```

ii. **Box** the code above and mark line with **(ii)** that caused this error:

```
    dec_num++
             ^
SyntaxError: invalid syntax
```

Write the code that would fix the error:

Answer:

```
dec_num = dec_num + 1
```

3. (a) What is the value (True/False) of out:

i.
```
in1 = True
in2 = False
out = in1 and in2
```

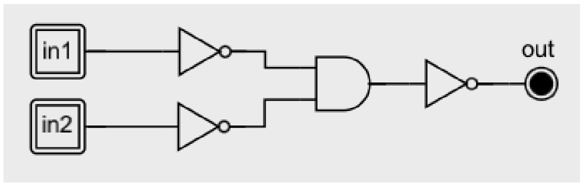
Answer:

```
out = False
```

ii.
```
in1 = False
in2 = True
out = not in2 and (in2 or not in1)
```

Answer:

```
out = False
```

iii. 

```
in1 = False
in2 = False
```

Answer:

```
out = False
```

(b) Fill in the values to yield the output:

i.
```
in1 = 
```
Answer: False

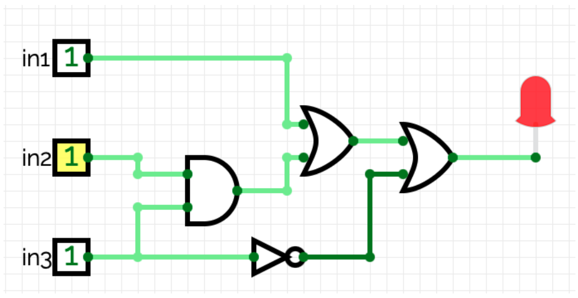
```
in2 = 
```
Answer: False

```
out = in1 or (not in1 and in2)
```

(c) Design a circuit that **exactly implements** the logical expression:

```
(in1 or (in2 and in3)) or (not in3)
```

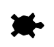
Answer:



4. (a) Draw the output for the function calls:

i. `ramble(tess,5)`

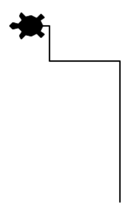
Answer:



```python
import turtle
tess = turtle.Turtle()
tess.shape('turtle')

def ramble(t, len):
    if len <= 10:
        t.stamp()
    elif len%2 == 0:
        t.left(90)
        t.forward(len)
        ramble(t, len//2)
    else:
        t.right(90)
        t.forward(len)
        ramble(t, len//2)
```

ii. `ramble(tess,100)`

Answer:



(b) What are the formal parameters for `ramble()`:

Answer: `t, len`

(c) If you call `ramble(tess,5)`, which branches of the function are tested (check all that apply):

Answer:

X The block of code at Line 7.
☐ The block of code at Lines 9-11.
☐ The block of code at Lines 13-15.
☐ None of these blocks of code (lines 7, 9-11, 13-15) are visited from this invocation (call).

(d) If you call `ramble(tess,100)`, which branches of the function are tested (check all that apply):

Answer:

X The block of code at Line 7.
X The block of code at Lines 9-11.
X The block of code at Lines 13-15.
☐ None of these blocks of code (lines 7, 9-11, 13-15) are visited from this invocation (call).

5. Write a function `both_cases()` that takes a string, removes all punctuation and spacing, and returns the characters that occurs both in upper and lower case in the string. For example:

```
both_cases("A man, a plan, a canal: Panama")
```

would return `A` and `P` since both `A` and `P` occur both as upper and lower case letters in the inputted string.

Answer:

Libraries:	No additional– just core Python
Input:	a string
Output:	the characters that occur in both upper and lower cases.

Design Pattern:

Answer:

☐ Accumulator ☐ Max/Min ✓ Finding Duplicates ☐ Searching

Principal Mechanisms (select all that apply):

Answer:

✓ Single Loop ☐ Nested Loop ✓ Conditional (if/else) ☐ Recursion
☐ Indexing/slicing ✓ Dictionary ☐ List Comprehension ☐ Regular Expressions

Process (as a concise and precise LIST OF STEPS / pseudocode):

(Assume libraries have already been imported.)

Answer:

Note: the answer is considered correct with either precise steps or pseudocode (actual Python code not needed).

(a) Remove punctuation from the string, `s`.

(b) Set up an empty dictionary, `new_dict`.

(c) Add the lowercase characters to the dictionary. That is,

```
for c in the_string:
    if c == c.lower():
        new_dict[c] = 1
```

(d)

(e) Add the uppercase characters to the dictionary for their corresponding lowercase letter:

```
for c in the_string:
    if c == c.upper() and c.lower() in new_dict:
        new_dict[c] = 2
```

(f) Return all the keys in the dictionary with value 2:

```
return( [k for (k,v) in new_dict.items() if v == 2] )
```

6. Fill in the Python code below for the function, `animate()`, that animates a hurricane tracker using the Turtle library.

Answer:

```
def animate(t,lat,lon,wind):
    """
    Takes a turtle, a location, and windspeed.
    Moves the turtle to the location, adjusts color \& pensize based on windspeed
    """
    #Lift the pen up:
    t.penup()
    #Move the turtle to (lat, lon) location:
    t.goto(lon,lat)
    #Check if wind stronger than 156. If so, change pen size to 5 and color to red:
    if wind > 156:
        t.pensize(5)
        t.color("red")
    #Else, check if wind > 129. If so, change pen size to 4 and color to orange:
    elif wind > 129:
        t.pensize(4)
        t.color("orange")
    #Else, check if wind > 110. If so, change pen size to 3 and color to yellow:
    elif wind > 110:
        t.pensize(3)
        t.color("yellow")
    #Else, check if wind > 95. If so, change pen size to 2 and color to green:
    elif wind > 95:
        t.pensize(2)
        t.color("green")
    #Else, check if wind > 73. If so, change pen size to 2 and color to blue:
    elif wind > 72:
        t.pensize(2)
        t.color("blue")
    #Else change pen size to 1 and color to white:
    else:
        t.pensize(1)
        t.color("white")
```

7. Write a **complete Python program** that creates a DataFrame. Your program should ask the user for:

- A list of place names,
- A list of populations, and
- A name for the output (CSV) file.

Your program should filter the DataFrame to contain only places with populations larger than 100000 and save the resulting DataFrame to the specified CSV file.

Hint: Create a dictionary from the inputted lists and then select rows with large populations. To cast an entire column of a DataFrame to be an integer, the following command is useful:

```
df['population'] = df['population'].astype(int)
```

Answer:

```
#Mock Exam, F25, #7
import pandas as pd
#Read in data:
name_str = input('Enter names, separated by spaces: ')
pop_str = input('Enter populations, separated by spaces: ')
#Need to split up the inputted strings into lists:
names = name_str.split(' ')
pops = pop_str.split(' ')
#Set up a dictionary of the lists (used to make df):
data = {'name': names, 'population' : pops}
#Make a DataFrame of the dictionary:
df = pd.DataFrame(data)
#Make sure that populations are integers (not strings):
df['population'] = df['population'].astype(int)
#Filter for large populations:
df = df[ df['population'] > 100000 ]
#Save the output:
file_name = input('Enter output file name: ')
df.to_csv(file_name)
```

8. (a) Consider the following MIPS program:

```
ADDI $s0, $zero, 10
ADDI $s1, $zero, 3
SUB $s2, $s1, $s0
ADD $s3, $s1, $s0
```

After the program runs, what is the value stored in:

$s0 register	$s1 register	$s2 register	$s3 register
Answer: 10	**Answer:** 3	**Answer:** -7	**Answer:** 13

(b) Consider the MIPS code:

```
1  ADDI $sp, $sp, -5
2  ADDI $t0, $zero, 48
```

```
3  ADDI $s2, $zero, 60
4  SETUP: SB $t0, 0($sp)
5  ADDI $sp, $sp, 1
6  ADDI $t0, $t0, 3
7  BEQ $t0, $s2, DONE
8  J SETUP
9  DONE: ADDI $t0, $zero, 0
10 SB $t0, 0($sp)
11 ADDI $sp, $sp, -4
12 ADDI $v0, $zero, 4
13 ADDI $a0, $sp, 0
14 syscall
```

Answer:

i) How many characters are printed?	**4**
ii) What is the first character printed?	**0**
iii) What is the whole message printed?	**0369**
iv) Detail the changes needed to the code to print the message in reverse:	**Line 1: Change sp to sp - 3.** **Line 3: Change s2 at 51.** **Line 11: Subtract 2 from sp.**

9. (a) What is the output:

```cpp
#include <iostream>
using namespace std;
int main()
{
  cout << "Motto:"
       << endl << "Mihi ";
  cout << "Cura,\nFuturi\n";
}
```

Answer:

```
Motto:
Mihi Cura,
Futuri
```

(b) Fill in the missing code to yield the output:

```
#include <iostream>
using namespace std;
int main()
{
    int myst = 5, quest = 10;
    while ((myst < 15) && quest > 0 )
    {
        cout << myst << "\t" << quest << endl;
```

Answer:

```
        myst++;
        quest -=2;

    }
    return 0;
}
```

(c) What is the output:

```
#include <iostream>
using namespace std;
int main()
{
    for (int i = 1; i <= 5; i += 2)
    {
        for (int j = 0; j < (5 - i)/2; j++)
            cout << "|";
        for (int j = 0; j < i; j++)
            cout << "#";
        for (int j = 0; j < (5 - i)/2; j++)
            cout << "|";
        cout << endl;
    }
    return 0;
}
```

Answer:

```
||#||
|###|
#####
```

10. (a) Translate the Python into a **complete** C++ program:

C++ program:

Answer:

```
#include <iostream>
using namespace std;
int main()
{
  int i;
  for (i = 100; i > 0; i=i-5)
  {
    cout << i << endl;
  }
  return 0;
}
```

Python program:

```
for i in range(100,0,-5)
    print(i)
```

(b) Write a C++ program that asks the user for the starting population and prints out the yearly population until it reaches 1000. Each year the population doubles in size.

A sample run:

```
Please enter the starting population: 50
Year 0  50
Year 1  100
Year 2  200
Year 3  400
Year 4  800
```

Answer:

```
#include <iostream>
using namespace std;

int main()
{
    int year = 0;
    int pop;
    cout<< "Enter starting population: ";
    cin >> pop;

    while (pop < 1000)
    {
        cout<< "Year " << i <<"\t" << total << "\n";
        balance = pop*2;
        year++
    }
}
```