CPSC 128

# Final Examination

**Instructions:**

- You have **three** (3) hours to complete the examination.
- The exam is in two parts.
- You may not use the computer while working on Part I, and **must** hand in Part I before turning the computer on.
- You **may**, but do not need to, use the computer for Part II of the exam.
- If you choose to work on a computer it is your responsibility to: 1) save your solutions to the Moodle **within** the three-hour time limit; 2) backup frequently — power failures happen!
- You are allowed to reference online resources, but you are not allowed to contact other people online for help or use AI tools like chatGPT to create or check the code.

**Advice:**

- The value of all questions is shown — **use your time wisely!** (The exam will be marked out of 180 and there are 180 minutes to complete it, so the weight assigned to a question indicates the longest you should spend on it.)
- **Don't panic.** Even the most daunting problem can be broken into small pieces. Stay calm and think your way through it.
- Rough work, e.g. pseudocode, can be worth substantial part marks — **don't erase or delete!**
- Your code should display the elements of good programming style, e.g. meaningful variable names and careful documentation.
- You may only make use of standard library functions covered in class. If you are unsure if a particular function is allowed, **ASK**!
- Note carefully whether a problem requests a *program*, a *program fragment*, a *function* or a *class* and respond accordingly.
- The time allocated for the problems in Part II should be sufficient for you to sketch out correct (perhaps even elegant!) solutions. However it is probably **not** sufficient to produce working, debugged Python programs (of course you can always prove me wrong on this!).
- **If you are unsure of anything, ASK**!

## Part I

**Answer these problems on this sheet.**

1. (*10 marks*) What output does the following code fragment produce?

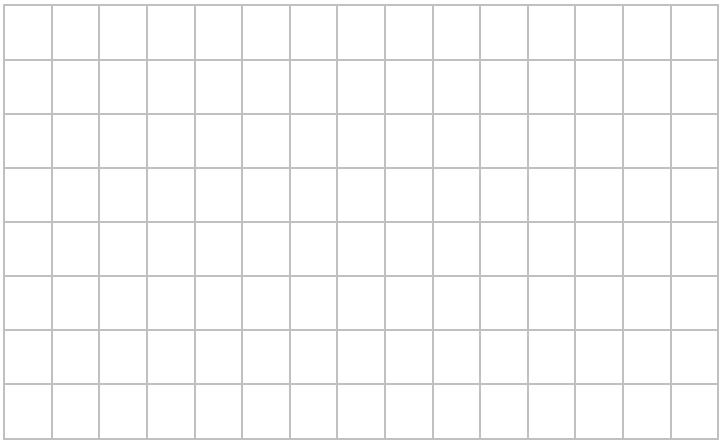
```
def mystery( num ):
    i = num
    while i > 0:
        if num%i == 0:
            print(i)
        i = i - 1

mystery( 16 )
```

2. (*10 marks*) What output does the following code fragment produce?

```
d = { 'b':0, 'o':2, 't':1 }
lst = ['b','o','t']
lst[d[lst[1]]] = lst[0]
print(lst)
```

3. (*10 marks*) What output does the following code fragment produce? (Each cell in the grid below corresponds to one character output position).

```
size = 5
for i in range( size ):
    print((size-i-1)*' ' + '*' + (2*i)*' ' + '*')
```



## Part II

**Write Python *code fragments* to do each of the following. (Use the names in courier bold as variable names or string literals).**

4. (*5 marks*) Increment the value of **`tens`** if the value of **`test`** is divisible by ten (with no remainder).

5. (*5 marks*) Create a dictionary called **`students`** representing the students in a school with the students' last names as the keys and a list each student's classes as the matched values. Create the dictionary with one student (and their classes) and add another student (and their classes) using indexing.

6. (*5 marks*) Convert a measurement in tablespoons to cups (1 tablespoon == 0.0625 cups).

7. (*10 marks*) Use **`if`** statements to set **`a`** to the median value of **`b`**, **`c`**, and **`d`**.

8. (*10 marks*) **`num`** is a four-digit number. Add the number formed by the first two digits to the number formed by the second two digits, and square the resulting sum. If the result is the original number display the message **`Got one!`** For example, here's a number that fails the test: 1430. Add 14 to 30 to get 44, square 44 to get 1936, but this is not what we started with (i.e. 1430) so we do not print **`Got one!`**

**Fill in the blanks**

9. (*10 marks*) Fill in the blanks in the program below so that it will work as the documentation specifies.

```
# Read a grade of A, B, C, D, or F.
# Keep trying until a valid grade is entered.

________ = ['A','B','C','D','F']

grade = input( 'Enter grade: ')

if grade __________ valid_grades:

    valid = ________
else:
    valid = ________

while not ________:

    print('Error: Grade must be one of: ', ________)

    ________ = input( '__________________ ')

    if grade ________ valid_grades:

        _________ = ________
```

**Testing**

10. (*10 marks*) For your summer job you're working on the testing team of a software development project. Your first task is to flesh out the tests for a **`smallest`** function that should return the smallest value of three values it is passed. *Insert your recommended tests* into the docstring below.

```
def smallest(a, b, c):
    ''' returns the smallest value of a, b and c
    >>> smallest(1, 2, 3)
    1









    '''
if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

**Write a function/method/program**

11. (*45 marks*) In assignment 11 we looked at using an adjacency matrix to represent a cave system in our hunt the Wumpus game. In this representation 1 represents a tunnel from the room indicated by the row position to the room number indicated by the column position. **Note** that if you are struggling with one sub problem you can still solve the other two because they don't build on each other directly.

    a) These matrices can be created and stored in tab separated files. Write a function called **`load_cave`** that reads a matrix from a tab separated file and converts it to a list of lists. The function should take the file name as an argument and return the list of lists. Make sure that the list elements are integers. Below is an example file and the list format it should be in after loading.

**File**

```
0    1    0    0    1
0    0    1    0    1
1    0    0    1    0
1    1    0    0    0
0    0    1    1    0
```

**List representation in Python**

```
cave_system = [[0,1,0,0,1],
               [0,0,1,0,1],
               [1,0,0,1,0],
               [1,1,0,0,0],
               [0,0,1,1,0]]
```

    b) Given a cave system as described above write a function called **`bidirectional`** that determines if all of the tunnels are 2-directional. That means that if a tunnel goes from room 5 to room 8 there should be an equivalent tunnel from room 8 to room 5. Cave systems that have 2-directional doors are symmetrical along the diagonal, so cave_system2 below id bidirectional, but cave_system1 is not. The function should take a cave system as the argument and return a Boolean value.

    c) Given a cave system as described above write a function called **`portal`** that adds a tunnel from room A to room B. The function should take two room numbers and the cave system list as arguments and return a modified cave system (list of lists).

```
cave_system1 = [[0,1,0,0,1],
                [0,0,1,0,1],
                [1,0,0,1,0],
                [1,1,0,0,0],
                [0,0,1,1,0]]
```

```
cave_system2 = [[0,1,0,0,1],
                [1,0,1,0,0],
                [0,1,0,1,0],
                [0,0,1,0,1],
                [1,0,0,1,0]]
```

**Class-based code**

12. (*5 marks*) A straight flush in poker is a hand that is both a straight (all the face values are consecutive) and a flush (all the suits are the same). Assume we already have a class **`Hand`** that has the methods **`isStraight()`** and **`isFlush()`** defined. Write a new class method **`isStraightFlush()`** that returns True if the given hand is a straight flush.

13. (*45 marks*) Duration times

    a) Design a class named **`HMSTime`** to represent represent a duration time in hh:mm:ss format, e.g. 10:30:25 represents 10 hours, 30 minutes and 25 seconds. Write the method(s) necessary for the code,

```
t1 = HMSTime(10, 30, 25)
t2 = HMSTime(15, 45, 5)
print('t1 =', t1)
print('t1 =', t2)
t2 = t2 - t1
print('t2 - t1 =', t2)
```

    to run and produce the output,

```
t1 = 10:30:25
t2 = 15:45:05
t2 - t1 = 5:14:40
```

    b) Define the equality operator with __eq__ that returns True when two **`HMSTime`** objects store the same amount of time.

    c) Write a method called **`clean_up`** that makes sure the time is stored in the cleanest format possible, so that if there are more than 60 seconds or more than 60 minutes it converts it to that is incorporated into the appropriate format (i.e. 10:45:76 should become 10:46:16).

    d) Write a function called **`days`** that takes a **`HMSTime`** object and returns a float that represents the same amount of time in days.