# CSC 110
# Midterm Exam:
# Thursday, 2 December 2021

**Name**:____________________________________________(please print clearly!)

**UVic ID number**:_____________________________________(please print clearly!)

**Signature**:______RUBIRC____________________________________________

**Exam duration:** 50 minutes

**Instructor:** Celina Berg

**Students must check the number of pages in this examination paper before beginning to write, and report any discrepancy immediately.**

- We will not answer questions during the exam. If you feel there is an error or ambiguity, write your assumption and answer the question based on that assumption.
- Answer all questions on this exam paper.
- The exam is closed book. No books or notes are permitted.
- **Electronic devices, including calculators, are not permitted**.
- The marks assigned to each part are printed within brackets. Partial marks are available.
- There are ten (10) pages in this document, including this cover page.
- Page 8 is left blank for scratch work. If you write an answer on that page, clearly indicate this for the grader under the corresponding question.
- Clearly indicate only one answer to be graded. Questions with more than one answer will be given a **zero grade**.
- It is strongly recommended that you read the entire exam through from beginning to end before beginning to answer the questions.

**Part 1 (21 marks)**

For the following questions, write your final answer in the box provided.

**For questions a-h, if the code given generates an error, in the box provided, write "invalid" followed by a BRIEF explanation of what caused the error.**

**GRADING: don't deduct for quotations shown in output**

a) Given `text_file.txt` contains the following text:

```
apple juice
bread
milk
corn flakes
```

What is the output of the following program?

```
file_handle = open('text_file.txt', 'r')

line = file_handle.readline()
while line != '':
    if line != 'milk':
        print(line, end='')
    line = file_handle.readline()

file_handle.close()
```

```
apple juice
bread
milk
corn flakes
```

1 mark for printing milk

`'milk' != 'milk\n'`

b) Given `text_file.txt` contains the following text:

```
apple juice
bread
milk
corn flakes
```

What does `text_file.txt` contain after the following program executes?

```
file_handle = open('text_file.txt', 'w')
file_handle.write('bananas')
file_handle.write('apples')
file_handle.close()
```

```
bananasapples
```

1 mark for overwrite

1 mark for write bananasapples

-½ if on separate lines

Given the following function definition (documentation and meaningful variable names omitted intentionally), answer questions **c** and **d**.

```
def foo(d, x, y):
    d[y] = x
```

c) What is the output of the following code segment:

```
a = {}
foo(a, 'bananas', [4,5,6,9])
foo(a, 'apples', [2,3,1])
foo(a, 'oranges', [5,8])

print(len(a))
print(a[ [5,8] ])
```

Invalid

Key cannot be a mutable type

1 mark for invalid

1 mark for reason

d) What is the output of the following code segment:

```
a = {}
foo(a, 7, 'bananas')
foo(a, 9, 'apples')
foo(a, 5, 'oranges')
foo(a, 6, 'bananas')

print(len(a))
print(a['bananas'])
print(a['apples'])
print(a['oranges'])
```

3

6

9

5

1 mark for 3

1 mark for 6

Given the following function definition (documentation and meaningful variable names omitted intentionally), answer questions **e** and **f**.

```
def bar(d, x, y):
    if y in d:
       d[y].append(x)
    else:
       d[y] = x
```

e) What is the output of the following code segment:

```
a = {}
bar(a, 'bananas', 1)

print(len(a))
print(a[1])
```

1

bananas

1 mark for 1

1 mark for bananas

f) What is the output of the following code segment:

```
a = {}
bar(a, 'bananas', 1)
bar(a, 'apples', 1)

print(len(a))
print(a[1])
```

Invalid

Cannot append a string to the initial string

added to the dictionary (d[y] = x)

1 mark for invalid

Given the following class, answer questions **g** and **h.**

```
class Flight:
    """ Flight class with departure and arrival airport codes
    and duration in minutes
    """
    def __init__(self, departure: str, arrival: str, duration: int) -> None:
        """
        initialize an instance of Flight with departure, arrival and duration
        >>> f = Flight('YVR', 'YYJ', 19)
        """
        self.__departure = departure
        self.__arrival = arrival
        self.__duration = duration

    def __str__(self) -> str:
        if self.__duration > 600:
            return f'arriving in {self.__arrival} late'
        else:
            return f'arriving in {self.__arrival} fast'
```

Assume the following methods have been implemented within this class and that they take expected arguments and return the expected values (implementation omitted intentionally):

- get_departure
- set_departure
- get_arrival
- set_arrival
- get_duration
- set_duration

g) What is the output of the following program:

```
f1 = Flight('YYJ' , 'LAX', 450)
f1.set_arrival('JFK')
print(f1)
```

arriving in JFK fast

1 mark JFK

h) What is the output of the following program:

```
f2 = Flight('YYJ' , 'LAX', 450)
f3 = Flight('YVR', 'YYJ', 19)
f4 = f2
f2.set_arrival('JFK')
f3.set_arrival('YYZ')
f4.set_arrival('KOA')

print(f2.get_arrival())
print(f3.get_arrival())
print(f4.get_arrival())
```

KOA

YYZ

KOA

1 mark each

Given the following code, answer questions **i**, **j** and **k** below:

```
print(1, end=' ')

try:
    print(2, end=' ')
    # inserted line of code
    print(3, end=' ')
except ErrorTypeOne:
    print(4, end=' ')

print(5, end=' ')
```

i) What is the output if the inserted line of code generates an `ErrorTypeOne` exception but not an `ErrorTypeTwo` exception?

```
1 2 4 5
```

j) What is the output if the inserted line of code generates an `ErrorTypeTwo` exception but not an `ErrorTypeOne` exception?

```
1 2
```

k) What is the output if the inserted line of code does not generate any errors (exceptions)?

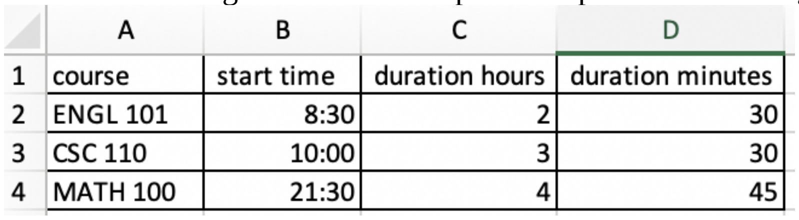
```
1 2 3 5
```

**Part 2 (19 marks)**

Consider the following `Time` and `Exam` classes. Assume `__str__` and `__repr__` methods have been implemented as expected (implementations omitted intentionally – do not complete).
Do not add to or change these classes.

```
class Time:
    """ Time class with hour and minute on a 24 hour clock
    Precondition: 0 <= hour < 24, 0 <= minute < 60
    """

    def __init__(self, hour: int, minute: int) -> None:
        """ initialize an instance of Time with hour and minute
        >>> t = Time(13, 8)
        """
        self.__hour = hour
        self.__minute = minute

    def add_time(self, hours: int, minutes: int) -> None:
        """ Adds given hours and minutes to this instance of Time ensuring
        precondition on hour and minute attributes are maintained.
        >>> t = Time(12, 20)
        >>> t.add_time(2, 15)
        >>> t
        Time(14, 35)
        >>> print(t)
        14:35

        >>> t = Time(23, 20)
        >>> t.add_time(2, 45)
        >>> t
        Time(2, 5)
        >>> print(t)
        2:05
        """
        # Implementation omitted intentionally.
        # DO NOT complete but it can be called.

from time import Time
class Exam:
    """ Exam class with course name and start and end Time instances.
    """
    def __init__(self, course: str, start: Time, end: Time) -> None:
        """ initialize an instance of Exam with the given course and
        start and end Time instances
        >>> strt = Time(17, 00)
        >>> end  = Time(19, 30)
        >>> e = Exam('CSC 110', strt, end)
        """
        self.__course = course
        self.__start  = start
        self.__end    = end
```

Consider the image below of a sample csv input file containing a header row and 3 lines of data:



| | A | B | C | D |
|---|---|---|---|---|
| 1 | course | start time | duration hours | duration minutes |
| 2 | ENGL 101 | 8:30 | 2 | 30 |
| 3 | CSC 110 | 10:00 | 3 | 30 |
| 4 | MATH 100 | 21:30 | 4 | 45 |

Complete the design of the function below according to the documentation provided.

```
from exam import Exam
from time import Time

COURSE_INFILE    = 0
TIME_INFILE      = 1
HOURS_INFILE     = 2
MINUTES_INFILE   = 3

def create_exam_list(filename: str) -> list[Exam]:
    """ creates and returns a list of Exam instances populated from filename.
    Precondition: The first line of file is a header row and it is ignored.
    Each line contains expected values separated by commas according to header row.
    >>> create_exam_list('0lines.csv')
    []
    >>> create_exam_list('input.csv')
    [Exam('ENGL 101', Time(8, 40), Time(11, 10)), \
Exam('CSC 110', Time(10, 0), Time(13, 30)), \
Exam('MATH 100', Time(21, 30), Time(2, 15))]
    """
    result_list = []
    file_handle = open(filename)
    file_handle.readline()

    for line in file_handle:
        line = line.strip()
        course, start, dur_hours, dur_minutes = line.split(',')
        hour, minute = start.split(':')

        hour        = int(hour)
        minute      = int(minute)
        dur_hours   = int(dur_hours)
        dur_minutes = int(dur_minutes)

        start_time = Time(hour, minute)
        end_time   = Time(hour, minute)

        end_time.add_time(dur_hours, dur_minutes)

        new_exam = Exam(course, start_time, end_time)
        result_list.append(new_exam)

    file_handle.close()
    return result_list
```

**Rubric on following page**

**Page left blank intentionally for scratch work if needed. If you write an answer to a question here, be sure to make note of it on the question itself so graders will see it.**

**RUBRIC:**

**2 marks – init/return list**

- **½ redundant conditional check to return empty list**

**3 marks – open/close file, skip first line (1 mark each)**

**-1 for opening 'input.csv'**

**1 mark – loops through rest of lines in file**

**-0.5 for each error**

**2 marks – split on comma**

**-1 for each error**

**Errors:**

- **Assignment statement backwards**
- **split called on wrong variable**

**2 marks – split on colon, convert data to int when necessary**

**1 mark for conversion**

**1 mark for split**

**-1 for each error**

**4 marks – create 2 time instances**

**2 marks for each instance created**

**-1 for each error (only deduct once if same mistake made twice)**

**Errors:**

- **Wrong number of arguments**

**No deduction for value error on arguments due to faulty split**

**2 marks – adds time to end time instance**

**-1 for each error**

**-1 if don't call add_time**

**-1 if calls a method that does not exist in the Time class**

**-1 if wrong number of arguments**

**Not deducting for storing None return value to an end time instance**

**3 marks – create and append exam instance to result list**

**2 marks – create Exam**

**-1 for each error**

**No deduction for value error on arguments due to faulty split**

**1 mark – append to result list**

**-2 hardcoded answer**

**Part 3 (5 marks)**

The following function has errors in it.

Find and fix the errors by crossing out incorrect code and writing the code that would replace it.

NOTE: there are no errors in the documentation

```
def make_grocery_list(recipe_list: list[str],
                      recipe_to_ingredients: dict[str, list[str]]) -> list[str]:
    """ Create a shopping list with unique values needed for given recipe_list
    - dict[recipe name, list of recipe ingredients]

    >>> recipes = ['spaghetti', 'chilli']

    >>> recipe_to_ingredients ={'chilli': ['beans', 'onions', 'tomato sauce', 'corn'],\
    'chicken soup': ['chicken', 'carrots', 'broth', 'onions'], \
    'spaghetti': ['noodles', 'tomato sauce', 'beef']}

    >>> make_grocery_list([], {})
    []
    >>> make_grocery_list(recipes, {})
    []
    >>> make_grocery_list([], recipe_to_ingredients)
    []
    >>> make_grocery_list(recipes, recipe_to_ingredients)
    ['noodles', 'tomato sauce', 'beef', 'beans', 'onions', 'corn']
    """


    grocery_list = []                   (1 mark)
    for recipe in recipe_list:

      if recipe in recipe_to_indredients: (2 marks: 1 find the error, 1 correct fix)

        ingredients = recipe_to_ingredients[recipe]

        ~~grocery_list = []~~

        for ingredient in ingredients:

          if ingredient not in grocery_list: (2 marks: 1 find the error, 1 correct fix)

            grocery_list.append(ingredient)

    return grocery_list
```

**END OF EXAM**

**For grading purposes, do not fill in:**

| **Part** | **Value** | **Mark** |
|---|---|---|
| 1 | 21 | |
| 2 | 19 | |
| 3 | 5 | |
| **Total** | **45** | |