[KEY] CSE 160 Summer 2026 - Final Exam

Full Name: _____

Student UW **Email**: _____@uw.edu

Do not turn the page until you are instructed to do so.

Instructions:

- You have **the entire class period (60 minutes)** to complete this exam.
- You must not begin working before time begins, and you **must stop working promptly when time is called**. Any modifications to your exam (writing or erasing) before time begins or after time is called will be reported as academic misconduct to the university.
- The exam is **closed book**, including no calculators, computers, phones, watches or other electronics.
- You are allowed a **single sheet of notes**, no larger than 8.5 x 11 inches, for yourself.
- You may also use the reference sheet (last sheet of the exam packet)
- Turn in ***all sheets*** of this exam—except the reference sheet, which you can tear off—together and in the same order when you are finished.
- You may ***only*** use parts and features of Python that have been covered in class up to this point.
- You may ask questions by raising your hand, and a TA will come over to you.

Initial here to indicate you have read and agreed to these rules: | |

Question	Points
Q1. Expressions	10
Q2. Debugging	5
Q3. File I/O	10
Q4. Nested Structures	10
Q5. Classes and Objects	15

Good luck!

1. Expressions (10 pts). In the table below, fill in the correct value and type (e.g., “string”) for each expression. In cells with multiple lines of code, evaluate the last line of code after the preceding lines are run. In other words, what does the expression on the last line *evaluate* to? If the code in a cell causes an error, write "Error" in the value column and leave the type column blank.

Expression	Value	Type

`lst1 = [16]`
`lst2 = lst1`
`lst2[0] = 0`
`lst1[0]`
	0	int

`uw = {}`
`uw["bio"] = 700`
`uw["cse"] += 20`
`uw["cse"]`
	Error	

`animals = ["dog"]`
`animals.append(["Tabby", "Bengal"])`
`animals`
	`["dog",["Tabby", "Bengal"]]`	list

`d = {"A": 1, "B: 2}`
`for i in d.items:`
`  result = i`
`result`
	drop because of typo on “B”	

`x = (1, 2, 3, 4)`
`for i in range(len(x)):`
`  x[i]*=2`
`x`
	Error	

2. Debugging & Testing (5 points).

2a (3 points) Consider the following **incorrect** code that is meant to take in a list of integers and update the list so all elements are in descending order.

```
def sort_list(lst):
      for i in range(len(lst)):
            for j in range(i + 1, len(lst)):
                  lst[i] = lst[j]
                  lst[j] = lst[i]
```

In order to track down the bug, you have been provided 3 test lists to run on this buggy method. **What is the resulting list if lst is passed into this function?** If you believe an Error will occur, write “Error”.

List Test	**Sort_List Result**
`lst = []`	[ ]
`lst = [1, 2, 3]`	[3, 3, 3]
`lst = [5, 8, 9, 1, 3]`	[3, 3, 3, 3, 3]

2a (2 points) In the box below, update the method so that it sorts the list correctly in descending order. You may add, remove, or edit any existing code.

```
def sort_list(lst):


     for i in range(len(lst)):

           for j in range(i + 1, len(lst)):

                 if lst[i] < lst[j]:

                   temp = lst[i]

                   lst[i] = lst[j]

                   lst[j] = temp

                 ~~lst[i] = lst[j]~~

                 ~~lst[j] = lst[i]~~
```

[page intentionally left blank]

3. File I/O (10 points). Given a csv file, `seasonal-fruits.csv`, you are asked to write a program that parses the file into a dictionary containing all the seasons as keys with the values associated with each season being a list of all fruits that are available within that season. Your program should NOT hardcode any of the seasons in your dictionary.

```
{'season': [fruit1, fruit2,...]}
```

For example, given the following csv:

```
Winter,January,oranges,grapefruits,lemons
Winter,February,oranges,grapefruits,lemons
Winter,March,oranges,grapefruits
Spring,April,strawberries
Spring,May,strawberries
Summer,June,cherries,raspberries,blueberries
Summer,July,cherries,blackberries,peaches
Summer,August,peaches,blackberries,watermelon
Fall,September,apples,pears,grapes,figs
Fall,October,apples,pears,grapes,pomegranates
Fall,November,cranberries
```

Your program should generate the following dictionary:

```
{'Winter': ['oranges', 'grapefruits', 'lemons'],
 'Spring': ['strawberries'],
 'Summer': ['cherries', 'raspberries', 'blueberries', 'peaches', 'watermelon'],
 'Fall': ['apples', 'pears', 'grapes', 'figs', 'pomegranates', 'cranberries']}
```

```
# Write your solution here

file = open("seasonal-fruits.csv")

d = {}

for line in file:

    data = line.strip().split(",")

    season = data[0]

    fruits = data[2:]


    if season not in d:

       d[season] = fruits

    else:

       for fruit in fruits:

          if fruit not in d[season]:

             d[season].append(fruit)

file.close()

# Could print, return, or simply do nothing here
# Problem only asks for you to create the dictionary
```

4. Nested Structures (10 points). Write a function, `find_hottest_day`, that takes in a daily temperature calendar in the form of a dictionary of dictionaries with lists of integers as values, and returns the date of the hottest day (**bolded and <u>underlined</u>** for convenience). Each month's days are split across 5 weeks, the first item in week 1 is the temperature on the 1st of the month, and so on. For example,

`find_hottest_day(calendar)` should return `July 17`

```
calendar = {"June": {"Week 1": [66, 70, 71, 71, 70, 68, 72],
                     "Week 2": [71, 71, 73, 71, 72, 74, 72],
                     "Week 3": [73, 74, 77, 85, 90, 88, 87],
                     "Week 4": [90, 88, 85, 82, 78, 80, 81],
                     "Week 5": [79, 77]},
            "July": {"Week 1": [76, 78, 77, 75, 79, 80, 79],
                     "Week 2": [83, 84, 86, 90, 92, 94, 96],
                     "Week 3": [94, 97, 98, 95, 93, 94, 91],
                     "Week 4": [89, 90, 87, 83, 84, 81, 79],
                     "Week 5": [77, 77, 75]},
            "August": {"Week 1": [77, 78, 75, 71, 72, 73, 73],
                       "Week 2": [75, 79, 74, 76, 78, 75, 71],
                       "Week 3": [74, 73, 76, 79, 81, 74, 74],
                       "Week 4": [72, 88, 75, 71, 68, 65, 70],
                       "Week 5": [67, 66, 64]}
```

Please write your solution on the next page…

```
# Write your solution here

    def find_hottest_day(calendar):

        hottest_month = ""

        hottest_day = 0

        hottest_temp = 0    # Alternatively: hottest_temp = None

        for month, week in calendar.items():

            count = 0

            for days in week.values():

                for day in days:

                    count += 1

                    if hottest_temp == 0 or day > hottest_temp:

                        hottest_temp = day

                        hottest_day = count

                        hottest_month = month

        return hottest_month, hottest_day # or any type of return/print
```

5. Classes and Objects (15 points). Consider the following Restaurant class:

```
class Restaurant:

    menu = {"Salad": 10,
          "Water": 0,
          "Coke": 5,
          "Noodles": 15}

    def __init__(self, location, size):
        self.location = location
        self.tables = {}
        for i in range(size):
            self.tables[i] = "Open"

    def reserve(self, party_size):
        if party_size >= 6:
            print("We cannot accommodate a party of this size.")
        else:
            for table, status in self.tables.items():
                if status == "Open":
                    self.tables[table] = "Reserved"
                    success = True
                    print("Table", table, "reserved.")
                    return table
            print("We have no available tables.")

    def start_tab(self, table):
        self.tables[table] = []

    def place_order(self, table, items):
        for item in items:
            if item in self.menu:
                self.tables[table].append(item)
            else:
                print("We do not sell this item.")

    def calculate_pay(self, table):
        total = 0
        for item in self.tables[table]:
            total += self.menu[item]
        return total

    def add_tip(self, amount, table):
        return self.calculate_pay(table) * amount
```

5a (10 points): What would be the output in the terminal if you ran the following lines of code?

```
>>> u_dist = Restaurant("U-District", 6)
>>> slu = Restaurant("SLU", 4)

>>> table = u_dist.reserve(5)

>>> for i in range(6, 1, -1):
      slu.reserve(i)

>>> u_dist.start_tab(table)

>>> u_dist.place_order(table, ["Soup", "Noodles", "Coke", "Coke"])
```

```
# Write your solution here

Table 0 reserved.
We cannot accommodate a party of this size.
Table 0 reserved.
Table 1 reserved.
Table 2 reserved.
Table 3 reserved.
We do not sell this item.
```

5b (5 points): You want to expand the Restaurant class by adding a new method called `get_bill` that takes in a `table` and `tip` parameters. The method should display all the items for the table along with each item's price and then display the total of the table with tip included.

For example, if `get_bill(table, 0.2)` were called with the items in part a, the output should look like the following:

```
Noodles: $15
Coke: $5
Coke: $5
Total: $30.0
```

The total bill comes to $30 because a 20% tip on a total of $25 is $5.

```
# Write your solution here


    def get_bill(self, table, tip):

        for item in self.tables[table]:

            print(str(item) + ": $" + str(self.menu[item]))

        total = self.calculate_pay(table)

        total += self.add_tip(tip, table)

        print("Total: $" + str(total))
```

OPTIONAL (1 point). Draw us one last picture :) Have a good rest of your summer!!!

CSE 160 26su Final Exam Reference Sheet

```
# if / elif / else syntax
if condition1:
    # statements
elif condition2:
    # other statements
else:
    # more statements
```

```
# for loop syntax
for i in sequence:
    # statements

# function definition syntax
def function_name(param1, param2, …):
    # statements
```

Sequences and Lists

`range([start,] stop [, step])`	Returns a sequence of numbers from ***start*** (inclusive) to ***stop*** (exclusive) incrementing by ***step***
`len(lst)`	Returns the number of elements in ***lst***
`lst = []`	Creates an empty list
`lst[idx]`	Returns the element in ***lst*** at index ***idx***
`lst[start:end:step]`	Returns a slice of ***lst*** from index ***start*** (optional) to index ***end*** (exclusive), incrementing by ***step*** (optional)
`lst.append(elmt)`	Adds the ***elmt*** to the end of ***lst***, returns **None**
`lst.extend(sequence)`	Adds each of the elements in ***sequence*** to the end of ***lst***, returns **None**
`lst.index(elmt)`	Returns the index of the first occurrence of ***elmt*** in ***lst***, errors if ***elmt*** is not in ***lst***
`lst.count(elmt)`	Returns the number of times ***elmt*** occurs in ***lst***
`lst.remove(elmt)`	Removes the first occurrence of ***elmt*** from ***lst***, errors if ***elmt*** is not in ***lst***, returns **None**
`lst.pop(idx)` `lst.pop()`	Removes and returns the element at index ***idx*** in ***lst*** With no parameter, removes the last element in ***lst***
`lst.insert(idx, elmt)`	Inserts ***elmt*** into ***lst*** at index ***idx***, returns **None**

File I/O

```
file = open(filepath, "r")

# read entire file at once
file.read()
file.close()
```

```
file = open(filepath, "r")

# read line by line
for line in file:
    # process line
```

```
file = open(filepath, "w")

# write entire file at once
file.write(string_to_write)
file.close()
```

Dictionaries

`my_dict = {}` `my_dict = dict()`	Creates a new, empty dictionary
my_dict`[`***key***`]`	Returns the value associated with the given ***key*** in ***my_dict***

`del` ***my_dict*`[`*****key***`]`	Remove ***key*** (and its associated value) from ***my_dict***
`list(`***my_dict***`.keys())`	Returns a list of keys in ***my_dict***
`list(`***my_dict***`.values())`	Returns a list of values in ***my_dict***
`list(`***my_dict***`.items())`	Returns a list of tuples of the form (***key***, ***value***)
`sorted(`***my_dict***`)`	Returns a sorted list of the keys in ***my_dict***

Sorting

`sorted(`***collection*** `[, `***key***`, `***reverse***`])`	Returns a sorted copy of ***collection***, based on the optional sort key (***key***) and optional order preference (***reverse***)
`lst.sort([key, reverse])`	Sorts the given list (***lst***), based on the optional sort key (***key***) and optional order preference (***reverse***), and returns **None**
key	A function to determine how to compare two values

Classes

`class` ***Name***`:`
`    # class methods`
`    def method(self [, args]):`
`        # method body`
	Defines a new class named ***Name*** with the subsequently defined methods (and attributes)

`def __init__(self [, args]):` `    # method body`	Function that is called during the creation of an instance of the class (e.g. ***Name()***)
self	Required parameter for all methods of a class. Refers to the specific instance of the class. Can hold any number of arbitrary variables (attributes), as in ***self*`.`my_attribute**
n `= Name()`	Instantiates (creates) a new instance of the ***Name*** class and assigns a reference to it to the variable ***n***
n*`.`**method***`([`*****args***`])`	Calls ***method*** on the instance of the class (***n***), optionally passing in any required arguments. Inside the function, **self** is assigned to ***n***