Full Name: [Edited for CSE 160 26wi] _____

Email Address (UW Net ID): _____ @uw.edu

Section: _____

CSE 160 Winter 2025 - Final Exam - Key

Instructions:

- You have **the entire testing period (110 minutes)** to complete this exam.
- The exam is **closed book**, including no calculators, computers, phones, watches or other electronics.
- You are allowed a single sheet of notes for yourself.
- We also provide a syntax reference sheet.
- **You will only be graded on what you include in the answer boxes/empty lines.**
- Turn in ***all sheets*** of this exam, together and in the same order when you are finished.
- When time has been called, you must put down your pencil and stop writing.
    - **Points will be deducted if you are still writing after time has been called.**
- 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.

Good luck!

Question	**Topic**	**Points**
Question 1	Expressions	10
Question 2	Functions	5
Question 3	Nested Structures	5
Question 4	CSVs and File I/O	10
Question 5	Debugging	5
Question 6	Classes	15

1 (10 pts). Given the table below, fill in the correct values and type for the matching expression. In other words, what will be outputted if this code is run in the Python interpreter. If there is an error, write "Error" in the value column. (You may leave the type column blank, and you do *not* have to explain the error.)

Expression	**Value**	**Type**
`cse = "cse160"` `cse[1] + str(160 * 1.0)`	`'s160.0'`	`string`

`lst = [("b", 3), ("a", 1),`
`("d", 5), ("t", 1)]`
 	
`lst[1]`
	`('a', 1)`	`tuple`

`t = ("a", "b", "c", "d")` `t.append("g")`	`Error!`	`Error!`

`d = {"a":{"a": 1, "b": 2, "c": 3},`
`"b":{"a": 5, "b": 4, "c": 3},`
`"c":{"a": 1, "b": 2, "c": 3}}`
 	
`d["c"]["b"]`
	`2`	`integer`

2 (5 pts). Suppose you are also given a list representing a flight schedule where each element in the list is a tuple of the form (airline, origin, destination, time). For example:

```
flight_schedule = [
    ("Alaska", "Seattle", "Los Angeles", "0600"),
    ("Alaska", "Seattle", "Los Angeles", "1200"),
    ("Delta", "Seattle", "Los Angeles", "1500"),
    ("Delta", "Los Angeles", "Phoenix", "1300"),
    ("American", "Los Angeles", "Phoenix", "0700"),
    ("United", "Phoenix", "Denver", "0500"),
    ("Southwest", "Phoenix", "Denver", "1400"),
    ("Frontier", "Los Angeles", "Seattle", "1200"),
    ("Delta", "Denver", "Seattle", "1800"),
    ("United", "Denver", "Seattle", "1700"),
    ("Southwest", "Denver", "Seattle", "1000")]
```

2a **(3 pts)**. Suppose you are given the following code

```
def get_flight_paths(schedule):
    paths = {}
    for flight in flight_schedule:
        if( (flight[1], flight[2]) in paths):
            if flight[0] not in paths[(flight[1], flight[2])]:
                paths[(flight[1], flight[2])].append(flight[0])
        else:
            paths[(flight[1], flight[2])] = [flight[0]]
    return paths
```

What is the returned value of the function call: `get_flight_paths(flight_schedule)`. You may use the following empty page to write out any scratch work.

```
{('Seattle', 'Los Angeles'): ['Alaska', 'Delta'],
 ('Los Angeles', 'Phoenix'): ['Delta', 'American'],
 ('Phoenix', 'Denver'): ['United', 'Southwest'],
 ('Los Angeles', 'Seattle'): ['Frontier'],
 ('Denver', 'Seattle'): ['Delta', 'United', 'Southwest']}
```

You may use this page for scratch work

2b **(2 pts)**. Suppose the flight `("Delta", "Seattle", "Los Angeles", "1500")` was delayed from `"1500"` to `"1700"` due to bad weather conditions. Write code that would update the flight schedule to reflect this change. In other words, write code so that `("Delta", "Seattle", "Los Angeles", "1500")` no longer appears in the list and is instead replaced with `("Delta", "Seattle", "Los Angeles", "1700")`. The updated tuple **DOES NOT** need to appear in the same index in the list. The original list is below for reference:

```
flight_schedule = [
    ("Alaska", "Seattle", "Los Angeles", "0600"),
    ("Alaska", "Seattle", "Los Angeles", "1200"),
    ("Delta", "Seattle", "Los Angeles", "1500"),
    ("Delta", "Los Angeles", "Phoenix", "1300"),
    ("American", "Los Angeles", "Phoenix", "0700"),
    ("United", "Phoenix", "Denver", "0500"),
    ("Southwest", "Phoenix", "Denver", "1400"),
    ("Frontier", "Los Angeles", "Seattle", "1200"),
    ("Delta", "Denver", "Seattle", "1800"),
    ("United", "Denver", "Seattle", "1700"),
    ("Southwest", "Denver", "Seattle", "1000")]
```

```
# Write your code here


# Remove tuple
flight_schedule.remove(("Delta", "Seattle", "Los Angeles", "1500"))

# Add new tuple
flight_schedule.append(("Delta", "Seattle", "Los Angeles", "1700"))
```

3 (5 pts). Consider the following dictionary that maps cities to their monthly average temperatures

```
city_temps = {"Seattle": {
              "January": 44, "February": 48, "March": 52, "April": 58,
              "May": 66, "June": 69, "July": 77, "August": 76,
              "September": 69, "October": 62, "November": 53, "December": 44},
              "Miami": {
              "January": 69, "February": 72, "March": 77, "April": 77,
              "May": 84, "June": 85, "July": 88, "August": 88,
              "September": 85, "October": 80, "November": 77, "December": 73 }}
```

3a **(1 pt)**. Imagine there was an error in the original dataset and the average temperature for Miami in the month of May was actually 85 and not 84. Write **<u>one line of code</u>** that would update the dataset to make this correction.

```
city_temps['Miami']['May'] = 85
```

3b **(4 pts)**. Write an average_annual_temp function that takes a city name and dataset and outputs the average annual temperature for that city as an integer. If the city is not in the dictionary, return the string `"No data found"`. Here are some examples:

```
print(average_annual_temp("Seattle", city_temps)) → 59
print(average_annual_temp("Miami", city_temps)) → 79
print(average_annual_temp("Los Angeles", city_temps)) → "No data found"
```

```
# Write your code here
def average_annual_temp(city, data):
    if city in data:
       total = 0
       count = 0
       temps = data[city]
       for month in temps.keys():
          total += temps[month]
          count += 1
       return int(total / count)
    else:
       return "No data found"
```

4 (10 pts). Consider you are given the following csv file named `grades.csv`: The contents of the file are shown below:

`grades.csv`

```
firstname,lastname,course,grade
John,Cena,CSE160,100
John,Cena,PHIL200,60
Jennifer,Coolidge,MATH100,78
Jennifer,Coolidge,MATH103,89
Emma,Stone,GH350,55
Ariana,Grande,EPI201,45
Cynthia,Erivo,CHEM301,76
```

Write a function `student_grades` that takes in parameters `filename` and an integer `passing_grade`. The function should return a dictionary where the keys are the student's full name (first and last name) and the values are a list of courses where the student has a grade that is greater than or equal to the passing grade.

Some sample function calls and outputs are shown below:

```
>>> student_grades("grades.csv", 30)

{'John Cena': ['CSE160', 'PHIL200'], 'Jennifer Coolidge': ['MATH100',
'MATH103'], 'Emma Stone': ['GH350'], 'Ariana Grande': ['EPI201'], 'Cynthia
Erivo': ['CHEM301']}

>>> student_grades("grades.csv", 65)

{'John Cena': ['CSE160'], 'Jennifer Coolidge': ['MATH100', 'MATH103'], 'Cynthia
Erivo': ['CHEM301']}


>>> student_grades("grades.csv", 90)

{'John Cena': ['CSE160']}


>>> student_grades("grades.csv", 1000)

{}
```

Write your code in the box on the following page

```
# Write your code here

import csv

# Using CSV DictReader
def student_grades(filename, passing_grade):
    grade_dict = {}
    file = open(filename)
    infile = csv.DictReader(file)
    for row in infile:
        full_name = row['firstname'] + ' ' + row['lastname']
        course = row['course']
        grade = int(row['grade'])

        if grade >= passing_grade:
            if full_name not in grade_dict:
                grade_dict[full_name] = [course]
            else:
                grade_dict[full_name].append(course)
    file.close()
    return grade_dict
```

```
# Not using CSV DictReader
def student_grades(filename, passing_grade):
    infile = open(filename)
    data = []
    for line in infile:
        data.append(line.strip().split(","))

    # Remove column names (first line)
    data = data[1:]

    grade_dict = {}

    for row in data:
        full_name = row[0] + ' ' + row[1]
        course = row[2]
        grade = int(row[3])


        if grade > passing_grade:
            if full_name not in grade_dict:
                grade_dict[full_name] = [course]
            else:
                grade_dict[full_name].append(course)
    infile.close()
    return grade_dict
```

5 (5 pts). You are given the following code:

```
def lets_break_this(lst1, lst2, remove_and_dup):

    """
    This function takes in two lists of integers and an integer.
    It first removes all occurrences (if any) of remove_and_dup in lst2.
    Then, every element of the new lst2 is appended to lst1,
    remove_and_dup times.

    Example:
       After calling: lets_break_this([1, 2], [3, 5], 5),
       lst2 becomes [3] and
       lst1 becomes [1, 2, 3, 3, 3, 3, 3].
    The function call should return None.
    """

    for val in lst2:  # This should be enough times!
        lst2.remove(remove_and_dup)

    result_list = []
    for item in lst1:
        result_list.append(item)

    for item in lst2:
        for i in range(remove_and_dup):
            result_list.append(item)


input_list = [1, 2]
input_list2 = [3, 5]
input_int = 5

result = lets_break_this(input_list, input_list2, input_int)

assert input_list == [1, 2, 3, 3, 3, 3, 3]
assert input_list2 == [3]
assert input_int == 5
assert result is None
```

(continued on next page)

5a **(2 pts)**. Fill in the following table on whether the assert statement will fail or not

Statement	Will this assertion raise an error? Yes or no.
`assert input_list == [1, 2, 3, 3, 3, 3, 3]`	YES
`assert input_list2 == [3]`	NO
`assert input_int == 5`	NO
`assert result is None`	NO

5b **(1 pt)**. What is the received value that makes the assert statement(s) from 5a fail?

The received value for input_list is [1, 2] which does not match the expected input_list of [1, 2, 3, 3, 3, 3, 3]

5c **(1 pt)**. Why does the code produce that received value instead of the expected value?

You are appending to the result_list instead of lst1 inside of the function. Therefore, you are not updating the original lst1 with the duplicate values (i.e., the duplicate 3s).

(continued on next page)

5d **(1 pt)**. How would you fix the code so that it produces the correct expected value?

Change the last line of the code from result_list.append(item) to lst1.append(item).

There is actually no need for result_list, so you can also remove the loop that appends values from lst1.

6 (15 pts). Read the class written on the following BankAccount class written on the following two pages. It is missing some necessary code indicated by the (_____). Using the code already written and the print output on the last page, fill in the blank lines of code to finish the class. Hint: Read through the entire class and comments before starting to fill in the blanks.

```
class BankAccount:
    """
    This BankAccount class will keep track of the
    money in your checking and savings accounts and
    purchases made
    """

    def __init__(self, name, checking, savings):
        self.name = name            # string
        self.checking = checking    # integer
        self.savings = savings      # integer
        self.purchases = []         # list


    def income(self, amount, account):
        """
        Adds the money amount passed into the
        function to the specified account:
        "checking" or "savings". If no valid account is
        passed, then print the statement 'Invalid account'
        """
        if account == "checking":

            self.checking += amount

        elif account == "savings":

            self.savings += amount

        else:
            print("Invalid account")
```

```
def max_purchase(self):
    """
    Prints the name and the cost of the
    most expensive purchase in the BankAccount.
    If no purchases have been made, print
    the message: 'No purchases made!'
    """

    if len(self.purchases) == 0:
        print("No purchases made!")

    else:
        item = ""
        max_purchase = 0
        for purchases in self.purchases:

            if purchases[1] > max_purchase:

                item = purchases[0]

                max_purchase = purchases[1]

      print("Max purchase is", item, "which cost",
            "$" + str(max_purchase))
```

```
def make_purchase(self, amount, item):
    """
    Makes a purchase and adds a tuple of the form
    (item, amount) into the purchases attribute. The
    purchase is only made if there is enough money
    in the checking account.
    Otherwise print the statement 'Not enough
    money in checking!'
    """

    if (self.checking - amount) >= 0:

        self.checking -= amount

        self.purchases.append((item, amount))

    else:
        print("Not enough money in checking!")

def display_account(self):
    """
    Prints all information related to the account
    object. A sample output is shown below:
    Account name: Bob
    Checking: $100
    Savings: $200
    # purchases made: 5
    """

      print("Account name:", self.name)

      print("Checking:", "$" + str(self.checking))

      print("Savings:", "$" + str(self.savings))

      print("# purchases made:", len(self.purchases))
```

Prompts and example output for Question 6

An example usage of the class is listed below. Each line of code (indicated by >>>) was run one line at a time and the output (if there was any) is printed immediately below.

```
>>> person = BankAccount("Joe", 200, 100)

>>> person.income(50, "checking")

>>> person.income(100, "retirement")
Invalid account

>>> person.max_purchase()
No purchases made!

>>> person.make_purchase(5, "chips")

>>> person.make_purchase(10, "burger")

>>> person.make_purchase(1000000, "car")
Not enough money in checking!

>>> person.max_purchase()
Max purchase is burger which cost $10

>>> person.display_account()
Account name: Joe
Checking: $235
Savings: $100
# of purchases made: 2
```

Extra Credit (1 point): Write a motivational quote about coding.