Full Name: _____

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

CSE 160 Winter 2025 - Final Exam

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.
- 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**
Expressions		
Functions		
Nested Structures		
CSVs and File I/0		
Debugging		
Classes		

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'` `x = cse[1] + str(160 * 1.0)`	's160.0'	str
`list = [("b", 3), ("a", 1), ("d", 5), ("t", 1)]` `sorted(list, key=itemgetter(1))`	[('a', 1), ('t', 1), ('b', 3), ('d', 5)]	list
`t = ("a", "b", "c", "d")` `t.append("g")`	Error	

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

`a = {'1.0', '3', '6', 6/3}`
`b = {'six', '1', '3', 2, '2.0'}`
`x = a & b`
	{2.0, '3'}	set

2 (x 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. 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 return value of the function call: `get_flight_paths(flight_schedule)`

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

2b. 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.

```
# 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 (x 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. Imagine there was an error in the original dataset and the average temperature for Miami in May was actually 85. Write one line of code that would update the dataset to be correct.

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

3b. 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 (x 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.

For example, if make the function call `student_grades("grades.csv", 65)` the function will return the dictionary:

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

```
# Write your code here

import csv
def student_grades(filename, passing_grade):
    dict = {}
    infile = csv.DictReader(open(filename))
    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 dict:
                dict[full_name] = [course]
            else:
                dict[full_name].append(course)
    return dict
```

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

```
def lets_break_this(input_list, input_set, input_int):
      '''
      This function takes in a list, a set, and an integer and
      removes all occurrences (if any) of the integer in the original
      set. Then, every element of the set is added to the original
      list, integer amount of times.

      Example:
            After calling lets_break_this([1, 2], {3, 5}, 5),
            input_set becomes {3} and input_list becomes [1, 2, 3, 3,
3,
            3, 3]. The function call should return None.
      '''
      input_set.discard(input_int)
      result_list = []
      for item in input_list:
            result_list.append(item)
      for item in input_set:
            for i in range(input_int):
                  result_list.append(item)

input_list = [1, 2]
input_set = {3, 5}
input_int = 5

result = lets_break_this(input_list, input_set, input_int)

assert input_list == [1, 2, 3, 3, 3, 3, 3]
assert input_set == {3}
assert input_int == 5
assert result is None
```

5a. Fill in the following table on whether the assert statement will fail or not

Statement	Will this fail? True or False
`assert input_list == [1, 2, 3, 3, 3, 3, 3]`	True
`assert input_set == {3}`	False
`assert input_int == 5`	False

`assert result is None`	False

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

[1, 2]

5c. Why does the code produce that received value instead of the expected value?

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

5d. 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 `input_list.append(item)`.

Above and beyond answer: You can also remove the first for loop and `result_list = []`. This will not change the output of the code and removes code that contributes nothing to final solution.

6 (15 pts). Read the class written on the following Trainer 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 checkings and savings accounts and
     purchases made
    """

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

    def income(self, amount, account):
        """
        Adds the money amount passed into the
        function to the specified account:
        checkings or savings
        """

        if ____________________________________________:

            ____________________________________________

        elif ____________________________________________:

            ____________________________________________

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

```
def make_purchase(self, amount, item):
    """
    Makes a purchase and adds a tuple of the form
    (item, amount) into the purchases attribute
    """

    if (_____) >= 0:

        self.checking-= amount

        _____

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

def max_purchase(self):
    """
    Returns the name of the item that has the
    highest purchasing cost in the account.
    If no purchases has been made, replies with
    the message: 'No purchases made'
    """

    if _____:
        print("No purchases made!")
    else:
        item = ""
        max_purchase = 0
        for purchases in _____:

            if _____ > max_purchase:

                item = _____
```

```
                max_purchase =_____________________________


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



def display_account(self):

        print("Account name:", _____________________________)

        print("Checkings:", "$" + _____________________________)

        print("Savings:", "$" + _____________________________)

        print("# purchases made:", _____________________________)
```

SOLUTION

```
class BankAccount:

    """
    This BankAccount class will keep track of your checking
    and savings accounts, purchases made, and credit score
    """

    def __init__(self, name, checking, savings):
        self.name = name
        self.checking = checking
        self.savings = savings
        self.purchases = []

    def income(self, amount, account):
        """
        Adds the money amount passed into the function
        to the specified account (checkings or savings)
        """
        if account == "checkings":
            self.checking+=amount
        elif account == "savings":
            self.savings+=amount
        else:
            print("Invalid account")

    def make_purchase(self, amount, item):
        """
        Makes a purchase and adds a tuple of the form
        (item, amount) into the purchases attribute
        """
        if (self.checking - amount) >= 0:
            self.checking-= amount
            self.purchases.append((item, amount))
        else:
            print("Not enough money in checkings!")

    def max_purchase(self):
```

```
    """
    Returns the name of the item that has the
    highest purchasing cost in the account.
    If no purchases has been made, replies with
    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 display_account(self):
    """
    Displays account information starting
    with the name, then checkings and savings amount,
    and finally with total number of purchases made
    """

    print("Account name:", self.name)
    print("Checkings:", '$' + str(self.checking))
    print("Savings:", '$' + str(self.savings))
    print("# of 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, “checkings”)

>>> 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 checkings!

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

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

Extra Credit (1 point):