[KEY] – CSE 160 Summer 2026 - Midterm Exam – [KEY]

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. Tracing	10
Q3. Function Writing	10
Q4. File I/O	10
Q5. Nested Lists	10

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

`a = 2`
`b = 5`
`a * b - 3`
	7	int

`("CSE"*2) + 160 + "A"`	Error	

`a = " hello "`
`if len(a) <= 5:`
`  a = "hola"`
`a`
	“ hello “	str

`n = [1, 2, 3]`
`n.extend([4,5])`
`n.pop(4)`
`n = n[2:5]`
`n`
	[3,4]	list

`d = {"a": 1, "b": 2}`
`d["c"] = 3`
`d[3] = "c"`
`d[1] = "a"`
`d`
	d = {“a”: 1, “b”: 2, “c”: 3, 3: “c”, 1: “a”}	dict

2. Tracing (10 points). Consider the following functions:

```
def climb(energy, water, length):
    if ( (energy > length) and (water > 7) ):
        spent = length // 3
        energy -= length
        water -= spent
        print("Energy:", energy, "and Water:", water)
    else:
        print("Re-energize!")
        for i in range (0, length, 3):
            energy = eat(energy)
        water = hydrate(water)
        print("Energy:", energy, "and Water:", water)

def eat(x):
    x += x % 2
    return(x)

def hydrate(y):
    bottles = [2, 4, 5, 7, 4, 5, 6]
    y += bottles[y - 1]
    return(y)
```

What is printed to the console when each of the following statements is executed? Place each new line of output on a separate line.

Statement	**Terminal Output**
`[1pt] print(eat(10))`	10
`[1pt] print(hydrate(3))`	8
`[2pts] climb(3, 2, 9)`	Re-energize! Energy: 4 and Water: 6
`[2pts] climb(7, 8, 5)`	Energy:2 and Water: 7

`[4pts]`
`energy = eat(5)`
`water = hydrate(5)`
`climb(energy, water, 5)`
	Energy: 1 and Water: 8

[page intentionally left blank]

3. Function Writing (10 points). Write a function `replace_vowels(word, replacement)` that takes in two parameters: a String named `word` and a character (i.e. a String of size 1) named `replacement`. This function will return a new string where every vowel character (a, e, i, o, u) in `word` is replaced with the character specified by `replacement` You can assume all characters and words will be written in lowercase. You may assume only strings will be passed into the function.

Example:

`replace_vowels("banana", "X") will return the string "bXnXnX"`

`replace_vowels("apple pie", "8") will return the string "8ppl8 p88"`

```
# Write your solution here
def replace_vowels(word, replacement):
    result = ""
    for c in word:
        if c in "aeiou":
            result += replacement
        else:
            result += c
    return result
```

4. File I/O (10 points). Adrian wants to search through a file containing various items to determine which he can afford to buy. He writes a function called `fill_cart` that takes in three arguments:

1. **`filename`**, the name of a CSV file containing information about the items.
2. **`category`**, the category of a particular item Adrian is searching to fill his cart with (e.g. `"Snacks"`).
3. **`max_price`**, the maximum amount of money that a single item in the cart can be.

Each CSV file Adrian uses with this function will have the following information on each line, separated by commas:

```
item_name,category,price
```

For example, the `items.csv` file looks like this:

```
Chips,Snacks,2.99
Switch 2,Gaming,449.99
Coffee Machine,Kitchen,79.99
Cookies,Snacks,7.49
```

The `fill_cart` function should search the CSV file (indicated `filename`) for items in the corresponding `category` (e.g. `"Snacks"`, `"Gaming"`, `"Kitchen"`) and only add items to a list if the item's price is less than or equal to the `max_price`. **The function should return a list of which items can be added to the cart. If no items can be added, the function should return an empty list**

Below are three examples of the intended functionality of `fill_cart`, using the `items.csv` file:

Function Call	**Return Value**
`fill_cart("items.csv", "Snacks", 10)`	`["Chips", "Cookies"]`
`fill_cart("items.csv", "Kitchen", 50)`	`[]`
`fill_cart("items.csv", "Kitchen", 100)`	`["Coffee Machine"]`

Adrian's implementation of the `fill_cart` function (on the next page) **contains 5 bugs!**

Your task is to **annotate** (write on) the code below to fix the bugs so that the program behaves as expected. You may add, delete, or move code.

```
def fill_cart(filename, category, max_price):

    file = open(filename)

    for line in file:

        cart = []  # Move before for-loop

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

        if ( (float(tokens[2]) <= max_price) or and (tokens[1] == category) ):

              cart.insert append(tokens[1 0])

    file.close()

    return cart
```

5. Nested Lists (10 points). Write a function called `show_grades(student, scores)` that takes in a String called `student` and a nested list of integers called `scores`. Each inner list in `scores` represents all the scores that `student` has earned in one of their classes. Your function should calculate the average of each inner list and print the average in the following format:

```
<name> averages: <list of average scores>
```

Suppose we have the following `student` and `scores`:

```
student = "Peter Parker"
scores = [[100, 100, 100], [80, 80, 50, 50], [100, 90, 90, 0]]
```

The function call `show_grades(student, scores)` should print the following:

```
"Peter Parker averages: [100, 65, 70]"
```

because the average of the first inner list is 100, the average of the second inner list is 65, and the average of the third inner list is 70

You may use either float (/) or integer (//) division in your calculations. Do not worry about matching the output spacing exactly as shown.

```
# Write your solution here

def show_grades(student, scores):

    result = []

    for inner in scores:

        total = 0

        for s in inner:

            total += s

        result.append(total / len(inner))

    print(student + " averages: " + str(result))
```

[page intentionally left blank]

OPTIONAL (1 point). Draw us a picture :)

CSE 160 26su Midterm Exam Cheat 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
```

Function	Description
range([*start*,] *stop* [, *step*])	Returns a sequence of numbers from *start* (inclusive) to *stop* (exclusive) incremented by *step*
len(*lst*)	Returns the number of elements in *lst*

Lists

Function	Description
lst = []	Creates an empty list
lst[*idx*]	Returns the element in *lst* at index *idx*
lst[*start* : *end*]	Returns a sublist of *lst* from index *start* to index *end* (exclusive)
lst[**start : end : step**]	Returns a sublist of *lst* from index *start* (default 0) to index *end* (exclusive, default *len(lst)*), incrementing by *step*
lst.append(*elmt*)	Adds the element *elmt* to the end of *lst*. Returns **None**.
lst.extend(*other*)	Adds each of the elements in the list *other* to the end of *lst*. Returns **None**.
lst.index(*elmt*)	Returns index of the first occurrence of *elmt* in *lst*, error if *elmt* is not in lst
lst.count(*elmt*)	Returns the number of times *elmt* occurs in *lst*
lst.remove(*elmt*)	Removes first occurrence of *elmt* from *lst*, error 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 an element *elmt* in *lst* at index *idx.* Returns **None**.
lst.sort()	Sorts the given list *lst*. Returns **None**.
lst.reverse()	Reverses the order of elements in the list *lst*. Returns **None**.

File I/O

Function	Description
`my_file = open(`***`filepath`***`)`	Opens the file with given ***`filepath`*** for reading, returns a file object
`my_file.close()`	Closes file `my_file`
`with open(`***`filepath`***`) as `***`f`***`:` `  # read file`	Opens the file with given ***`filepath`*** for reading via the file object ***`f`*** in the body of the "with" statement.
`str.strip()`	Given a string `str`, remove any trailing or ending whitespace.
`str.split([separator])`	Given a string `str`, splits the string on the optional separator (will split on spaces if no separator is given) and returns a list of separated strings.

```
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

Function	Description
`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`***`]`	Removes the ***`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)`***

```
# Process each key-value pair together:
for key, value in my_dict.items():
    # process key and value
```

```
# Process one key at a time
for key in my_dict:
    # use dictionary's key
```