COM S 1270 Exam #1 PRACTICE VERSION KEY

Fall 2025

Name: _____ **Student ID #: _____**

ISU username/ netid: _____**@iastate.edu**

General Instructions:

- Please **look over the exam carefully** before you begin.
- **READ ALL OF THE INSTRUCTIONS CAREFULLY – THIS IS THE POINT OF THE EXAM.**
- **NO, REALLY, READ ALL OF THE INSTRUCTIONS VERY CAREFULLY.**
- The problems are **not** necessarily **in order of difficulty**.
- **Closed book/ notes, Closed internet/ email, Closed friend/ talking**.
- **NO ELECTRONIC DEVICES/ NO HEADPHONES**.
- **Time Limit**: 75 minutes.
- **Use correct Python syntax** for writing any code – including the use of whitespace.
- If you like, you may draw vertical straight lines to denote different levels of whitespace for clarity.
- You are **not required** to **write comments** for your code.
- **DO NOT WRITE ANY CODE NOT ASKED FOR IN THE QUESTION. Yes, that includes *if __name__ == "__main__":***
- There are five (5) questions on the exam.
- **Mark the three (3) questions you wish to have graded with a star on the question letter** (★).
- If you do *not* mark three (3) questions with a star (★), questions will be graded starting at question A.
- The questions you select to be graded will be worth thirty (30) points each.
- You **must** at least *attempt* the other two (2) questions.
- Regardless of the correctness of the answers for the two (2) questions you do not select to be graded, so long as you *try* you will receive five (5) points each.
    - Here, 'trying' means providing code for a full solution. Meaning – a partial solution or just comments is insufficient for the purposes of 'trying.'
- If you do *not* attempt one or more of the other two (2) non-graded questions, you will *not* receive the points for each question you do not attempt.

Question	Student's Score	Max Score
GR #1:		30
GR #2:		30
GR #3:		30
AT #1:		5
AT #2:		5
TOTAL:		100

Python Built-in Functions

Function	Description
abs(x)	Returns the absolute value of *x*
float(x)	Converts *x* to a float
input(prompt)	Reads input from the user and returns it as a string
int(x)	Converts *x* to an integer
len(s)	Returns the length of string or collection *s*
max(iterable)	Returns the largest item in *iterable*
min(iterable)	Returns the smallest item in *iterable*
print(x)	Prints value to console with a newline.
print(x, end='')	Prints value to console without a newline
sorted(iterable)	Returns a sorted list from elements of *iterable*
str(x)	Return a str version of *x*
sum(iterable)	Returns the sum of elements in *iterable*
type(obj)	Returns the type of object *obj*

Python String Methods

Method	Description
str.capitalize()	Returns a new string with the first letter of the original string capitalized
str.endswith(suffix)	Returns True if string ends with *suffix*
str.find(sub)	Returns index of first occurrence of *sub*, or -1 if not found
str.isalnum()	Returns True if all characters are alphabetic or digits
str.isalpha()	Returns True if all characters are alphabetic
str.isdigit()	Returns True if all characters are digits
str.join(iterable)	Joins elements in *iterable* with str as separator
str.lower()	Converts all characters to lowercase
str.replace(old, new)	Replaces all occurrences of *old* with *new*
str.split()	Splits the string into a list based on whitespace
str.split(sep)	Splits the string into a list at the separator *sep*
str.startswith(prefix)	Returns True if string starts with *prefix*
str.strip()	Removes leading and trailing whitespaces
str.upper()	Converts all characters to uppercase

Python List Methods

Method	Description
list.append(x)	Adds *x* to the end of the list
list.clear()	Removes all elements from the list
list.copy()	Returns a shallow copy of the list
list.count(x)	Returns number of occurrences of *x* in list
list.extend(iterable)	Extends list by appending all the elements from *iterable*
list.index(x)	Returns the index of first occurrence of *x* in list
list.insert(i, x)	Inserts *x* at index *i*
list.pop(i)	Removes and returns element at index *i*
list.remove(x)	Removes the first occurrence of *x* from the list
list.reverse()	Reverses the elements in place
list.sort()	Sorts the list in ascending order

Python Dictionary Methods

Method	Description
dict.clear()	Removes all items from dictionary
dict.copy()	Returns a shallow copy of the dictionary
dict.get(key)	Returns value for *key*, or None if not found
dict.items()	Returns a view of all key-value pairs
dict.keys()	Returns a view of all keys
dict.pop(key)	Removes and returns value for *key*
dict.popitem()	Removes and returns random (key, value) pair
dict.setdefault(key, default)	Returns value for *key*, or inserts *key* with *default* value
dict.update(other_dict)	Updates dictionary with key-value pairs from *other_dict*
dict.values()	Returns a view of all values

A) Shipping Cost Calculator

A store charges shipping based on package weight:

- Up to 2 kg: $10
- Over 2 kg: $10 + $8 per kg beyond 2
- Maximum charge: $25

Write a function **shippingCost** that

- Takes one parameter, **weight** (the weight of the package in kg). You may assume that **weight** will always be a positive integer greater than zero.
- Returns the total shipping cost.

Examples:

shippingCost(2) returns 10, because 2 kg costs $10.00

shippingCost(3) returns 18, because 3 kg costs $10.00 + $8.00 = $18.00

shippingCost(20) returns 25, because the maximum charge is $25.00

```
def shippingCost(weight):
    if weight <= 2:
        cost = 10.0
    else:
        cost = 10.0 + 8.0 * (weight - 2)
    if cost > 25.0:
        cost = 25.0
    return cost
```

Syntax: _____ + Logic: _____ + Output: _____ = Total: _____

B) Buses Needed

Each bus can carry 50 passengers. Write a function **busesNeeded** that

- Takes one parameter, **total_people**. You may assume that **total_people** will always be a positive integer greater than zero.
- Returns how many buses are needed so everyone gets a seat.

Examples:

busesNeeded(30) returns 1, because a bus holds 50 people, and 30 < 50

busesNeeded(60) returns 2, because buses hold 50 people each, and partial busses do not exist

```
def busesNeeded(total_people):

    if total_people % 50 == 0:
        buses = total_people // 50
    else:
        buses = (total_people // 50) + 1

    return buses
```

Syntax: _____ + Logic: _____ + Output: _____ = Total: _____

C) Credit Score Category

Write a function called **creditCategory** that

- Takes one parameter, **score**. You may assume that **score** will always be a positive integer greater than zero.
- Returns a string describing the credit rating using the criteria below.

Credit Score	Category
Below 580	Poor
580 – 669	Fair
670 – 739	Good
740 – 799	Very Good
800 +	Excellent

Examples:
creditCategory(710) : Good
creditCategory(805) : Excellent

```
def creditCategory(score):
    if score > 800:
        return "Excellent"
    elif score >= 740 and score <= 799:
        return "Very Good"
    elif score >= 670 and score <= 739:
        return "Good"
    elif score >= 580 and score <=669:
        return "Fair"
    else:
        return "Poor"
```

Syntax: _____ + Logic: _____ + Output: _____ = Total: _____

D) Mask Digits

Write a function **maskDigits** that

- Takes one parameter, **text**. You may assume that **text** will always be a string with a length greater than zero.
- Returns a new string where every digit(0-9) in text is replaced by the '#' symbol.

NOTE: You may not use any built-in string methods to accomplish this task.

Examples:
maskDigits('My pin is 1234') returns the string “My pin is ####”
maskDigits('Room 204') returns the string “Room ###”

```
def maskDigits(text):
    result = ""
    for ch in text:
        if ch in '0123456789':
            result += '#'
        else:
            result += ch
    return result
```

Syntax: _____ + Logic: _____ + Output: _____ = Total: _____

E) Sum of Squares and Cubes

Write a function **sumSquareCube** that:

- Takes one parameter, **n**. You may assume that **n** will always be a positive integer greater than zero.
- Loops from 1 to **n** (inclusive) and, to a sum, adds:
  - i^2 , if i is even
  - i^3, if i is odd
- Returns the total sum following the pattern below:

$$sum = 1^3 + 2^2 + 3^3 + 4^2 + \cdots .. n^2, if\ n\ is\ even$$
$$sum = 1^3 + 2^2 + 3^3 + 4^2 + \cdots .. n^3, if\ n\ is\ odd$$

- NOTE: Here, i is the loop variable holding the values from 1 to **n**

Examples:
sumSquareCube(3) returns 32, because $1^3 + 2^2 + 3^3 = 32$
sumSquareCube(4) returns 48, because $1^3 + 2^2 + 3^3 + 4^2 = 48$

```
def sumSquareCube(n):
    total = 0
    for i in range(1, n + 1):
        if i % 2 == 0:
            total += i ** 2
        else:
            total += i ** 3
    return total
```

Syntax: _____ + Logic: _____ + Output: _____ = Total: _____

Scratch paper

Scratch paper