CS-361L Artificial Intelligence

Python Assignment **Weightage: 10%**

❖ Part 1

Question 1:

What is the output of the following program?

```
i = 0
while i < 3:
       print i
       i++
       print i+1
```

Output:

Question 2:

```
String1 = '''I'm a "Alaric"'''
print("Initial String with use of Triple Quotes: ")
print(String1)
String1 = 'I\'m a "Alaric"'
print("\nEscaping Single Quote: ")
print(String1)
String1 = "I'm a \"Alaric\""
print("\nEscaping Double Quotes: ")
print(String1)
String1 = "C:\\Python\\Alaric\\"
print("\nEscaping Backslashes: ")
print(String1)
String1 = "This is \x47\x65\x65\x6b\x73 in \x48\x45\x58"
print("\nPrinting in HEX with the use of Escape Sequences: ")
print(String1)
```

```
String1 = r"This is \x47\x65\x65\x6b\x73 in \x48\x45\x58"
print("\nPrinting Raw String in HEX Format: ")
print(String1)
```

Write output:

Question 3:

```
matrix = [
     [8, 3, 2],
     [3, 2, 1],
    [4, 4, 5]
]
Print(matrix[0][2])
```

Output:

Question 4:

```
numbers = [12,3,18,15,20,16]
max = numbers[0]
for no in numbers:
    if no > max:
        max = no
print(max)
```

Output:

Question 5:

```
coordinates = (1, 2, 3)
x, y, z = coordinates
print(y)
```

Output:

Question 6:

```
phone =input("phone_no: ")
digit_mapping = {
    "1": "one",
     "2": "two",
    "3":"three"
}
output =""
for ch in phone:
   output += digit_mapping.get(ch, "!") + ""
print(output)
```

Output:

Question 7:

```
customer = {
    "name": "jack smith",
    "age": 20,
    "is verified": True
}
customer["name"] = "john smith"
customer["birthdate"] = "jan 1 1999"

print(customer["name"])
print(customer["birthdate"])
```

Output:

Question 8:

```
my_set = {1,3}
print(my_set)
my_set.add(2)
print(my_set)
my_set.update([2,3,4])
print(my_set)
my_set.update([4,5], {1,6,8})
print(my_set)
```

Output:

❖ Part 2

Coding Questions:

1. Write a Python program to create a tuple with different data types.
2. Write a Python program to print all unique values in a dictionary. Go to the editor
   Sample Data : [{"V":"S001"}, {"V": "S002"}, {"VI": "S001"}, {"VI": "S005"}, {"VII":"S005"}, {"V":"S009"},{"VIII":"S007"}]
   Expected Output : Unique Values: {'S005', 'S002', 'S007', 'S001', 'S009'}
3. Write a Python program to create and display all combinations of letters, selecting each letter from a different key in a dictionary. Go to the editor
   Sample data: {'1': ['a', 'b'], '2': ['c', 'd']}
   Expected Output:
   ac
   ad
   bc
   bd
4. Find the series of Fibonacci numbers using lambda function.
5. Write a list comprehension which, from a list, generates a lowercased version of each string that has length greater than five.
6. Let we have a dictionary D which contain roll number of students as key and its values shall be a list. The list has tuples as its item. Each tuple represents the name of the subject and gpa of that subject. You can visualize the data structure as of following:

Key	**Value**
2016-CS-700	[(DSA,3), (Algo,2.5), (AI,3)]
2016-CS-701	[(LA,3), (Algo,3.5), (PF,2.8)]
…	…
…	..
2016-CS-710	[(OOP,3), (DB,3.5), (PF,2.8)]

a) Write a program which shall print GPA of all students. Assume all subject have 3 credit hours. (with and without using for loop)
b) Write a program that print the highest number obtained in DSA (with and without using for loop)
c) Write a program that print the total number of students those have less than 2.5 GPA in AI? (with and without using for loop)

❖ Part 3

Task 01:

Take a list, say for example this one:

a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

and write a program that prints out all the elements of the list that are less than 5.

Extras:

1.1. Instead of printing the elements one by one, make a new list that has all the elements less than 5 from this list in it and print out this new list.

1.2. Write this in one line of Python.

1.3. Ask the user for a number and return a list that contains only elements from the original list a that are smaller than that number given by the user.

Task 02:

Create a program that asks the user for a number and then prints out a list of all the divisors of that number. (If you don't know what a *divisor* is, it is a number that divides evenly into another number. For example, 13 is a divisor of 26 because 26 / 13 has no remainder.)

Task 03:

Take two lists, say for example these two:

```
a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
```

and write a function that returns a list that contains only the elements that are common between the lists (without duplicates). Make sure your function works on two lists of different sizes.

Extras:

3.1. Randomly generate two lists to test this.

3.2. Write this in one line of Python (don't worry if you can't figure this out at this point - we'll get to it soon)

Task 04:

4.1. Ask the user for a string and print out whether this string is a palindrome or not. (A **palindrome** is a string that reads the same forwards and backwards.)

4.2. Let's say I give you a list saved in a variable: `a = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]`. Write one line of Python that takes this list `a` and makes a new list that has only the even elements of this list in it.

Task 05:

Write a program (using functions!) that asks the user for a long string containing multiple words. Print back to the user the same string, except with the words in backwards order. For example, say I type the string:

My name **is** Michele

Then I would see the string:

Michele **is** name My

shown back to me.

Task 06:

Make a two-player Rock-Paper-Scissors game. (*Hint: Ask for player plays (using input), compare them, print out a message of congratulations to the winner, and ask if the players want to start a new game*)

Remember the rules:

- Rock beats scissors
- Scissors beats paper
- Paper beats rock

Task 07:

Write a password generator in Python. Be creative with how you generate passwords - strong passwords have a mix of lowercase letters, uppercase letters, numbers, and symbols. The passwords should be random, generating a new password every time the user asks for a new password. Include your run-time code in a main method.

Extra:

- Ask the user how strong they want their password to be. For weak passwords, pick a word or two from a list.

Task 08:

a) In this exercise, the task is to write a function that picks a random word from a list of words below. `listOfWords = [APPLE, BILBO, CHORUSED, DISIMAGINE, ENSURING, FORMALISING, GLITCHES, HARMINE, INDENTATION, JACKED, KALPACS, LAUNDRY, MASKED, NETTED, OXFORD, PARODY, QUOTIENTS, RACERS, SADNESS, THYREOID, UNDUE, VENT, WEDGED, XERIC, YOUTHHOOD, ZIFFS]`

Hint: use the Python random library for picking a random word.

b) Let’s continue building Hangman. In the game of Hangman, a clue word is given by the program that the player has to guess, letter by letter. The player guesses one letter at a time until the entire word has been guessed. (In the actual game, the player can only guess 6 letters incorrectly before losing).

Let’s say the word the player has to guess is “EVAPORATE”. For this exercise, write the logic that asks a player to guess a letter and displays letters in the clue word that were guessed correctly. For now, let the player guess an infinite number of times until they get the entire word. As a bonus, keep track of the letters the player guessed and display a different message if the player tries to guess that letter again. Remember to stop the game when all the letters have been guessed correctly! Don’t worry about choosing a word randomly or keeping track of the number of guesses the player has remaining - we will deal with those in a future exercise.

An example interaction can look like this:

```
>>> Welcome to Hangman!
_ _ _ _ _ _ _ _ _
>>> Guess your letter: S
Incorrect!
>>> Guess your letter: E
E _ _ _ _ _ _ _ E
...
```

And so on, until the player gets the word

c) In this exercise, we will finish building Hangman. In the game of Hangman, the player only has 6 incorrect guesses (head, body, 2 legs, and 2 arms) before they lose the game.

In Part 1, we loaded a random word list and picked a word from it. In Part 2, we wrote the logic for guessing the letter and displaying that information to the user. In this exercise, we have to put it all together and add logic for handling guesses.

Copy your code from Parts 1 and 2 into a new file as a starting point. Now add the following features:

- Only let the user guess 6 times and tell the user how many guesses they have left.
- Keep track of the letters the user guessed. If the user guesses a letter they already guessed, don’t penalize them - let them guess again.

Optional additions:

- When the player wins or loses, let them start a new game.
- Rather than telling the user "You have 4 incorrect guesses left", display some picture art for the Hangman. *This is challenging - do the other parts of the exercise first!*

Your solution will be a lot cleaner if you make use of functions to help you!