**Lab 4 - Strings**

**Content:**

Sstrings, string slicing, and string methods.

## Conceptual Practice

**1.** Write a Python program to print each character of a string on a single line.

**my\_str = input("Enter a string: ")**

**for char in my\_str:**

**print(char)**

**2.** Given the string "Monty Python":

1. Write an expression to print the first character**. print(my\_str[0])**
2. Write an expression to print the last character. **print(my\_str[-1])**

(c) Write an expression including len to print the last character. **print(my\_str[len(my\_str) - 1])**

(d) Write an expression that prints "Monty". **print(my\_str[:5])**

**3.** Given a variable S containing a string of odd length:

(a) Write an expression to print the middle character. **print(my\_str[len(my\_str) // 2])**

(b) Write an expression to print the string up to but not including the middle character

(i.e., the first half of the string). **print(my\_str[:len(my\_str) // 2])**

(c) Write an expression to print the string from the middle character to the end (not

including the middle character). **print(my\_str[(len(my\_str) // 2) + 1:])**

**4.** Experiment with the count method. What does it count? For example,

some\_string = "Hello world!"

some\_string.count("o")

**Counts the amount of occurrences of that substring in original string**

**5.** Experiment with the strip method. What does it do? For example,

some\_string = "Hi!......"

some\_string.strip(".!")

**Removes any leading and trailing whitespaces from a string. It can also remove specified characters from the beginning and end**

## Critical Thinking Tasks

**6.** Write a Python program that will reverse a string (using a loop, not using slicing)

**some\_string = "Hello!"**

**reversed\_str = ""**

**for char in some\_string:**

**reversed\_str = char + reversed\_str**

**print(reversed\_str)**

\*\*\*(prepending rather than appending)\*\*\*

OR

for i in range(len(some\_string) – 1, -1, -1):

reversed += some\_string[i]

OR

some\_string[::-1]

**7.** A palindrome is a string that reads the same forward and backwards. “Same” in a sense that: 1) case does not matter; and 2) punctuation is ignored. "Madam I'm Adam" is a palindrome. Write a program that checks if a string is palindrome. Steps:

1. Get a string from an input
2. Convert it to lowercase
3. Remove bad characters (punctuation and whitespaces)
4. Check whether the reverse of the resulting string is equal the string

Every letter is converted using the lower method. import string, brings in a series of predefined sequences:

string.digits

string.punctuation

string.whitespace

To remove all non-wanted characters use the replace method. First arg is what to replace, the second the replacement. The beginning of the solution is given below. Try to finish it...

**import** string

original\_str = input(**"Input a string: "**)

modified\_str = original\_str.lower()

bad\_chars = string.whitespace + string.punctuation

**import string**

**original\_string = input("Enter a string: ")**

**modified\_string = original\_string.lower()**

**bad\_chars = string.whitespace + string.punctuation**

**for char in bad\_chars:**

**modified\_string = modified\_string.replace(char, "")**

**if modified\_string == modified\_string[::-1]:**

**print("The string is a palindrome.")**

**else:**

**print("The string is not a palindrome.")**

**8.** Write a Python program that will “encrypt” a string. The encryption algorithm we’ll use is to add 1 to the ASCII code, so ‘a’ becomes ‘b’, ‘b’ becomes ‘c’, etc. The string ‘abc’ becomes ‘bcd’. You’ll need to use the functions ord() and chr() discussed in class

Hint: To encrypt the letter ‘a’ take the ASCII code of ‘a’ 97, add 1 (98) and find the

character with ASCII code 98 (‘b’). So ‘a’ encrypted becomes ‘b’

**input\_string = input("Enter a string to encrypt: ")**

**encrypted\_string = ""**

**for char in input\_string:**

**# Get the ASCII code, add 1, and convert back to character**

**encrypted\_char = chr(ord(char) + 1)**

**encrypted\_string += encrypted\_char**

**print("Encrypted string:", encrypted\_string)**

**Encrypting using the index of each letter in the array:**

for i, char in enumerate(password):

encrypted += chr(ord(char) + i)

**9.** (a) Suppose you want to print a line full of'#' characters. For simplicity, let’s say that a

line can have only 80 characters. One way is to create a long string to be printed. How would you do it more elegantly in Python using the plus operation (+) of strings?

**line = ""**

**for char in range(80):**

**line += '#'**

**print(line)**

(b) Suppose you want to print a column full of '#' characters. For simplicity, let’s

say that a column could have only 30 characters. Similar to (a), how would you do

it more elegantly in Python using the multiply operation (\*) of strings? Hint: Use

the newline character (‘\n’).

**line = ('#\n' \* 30)**

**print(line)**

**10.** Although Python’s formatted printing can be cumbersome, it can often drastically

improve the readability of output. Try creating a table out of the following values:

Melting and Boiling Points of Alkanes

Name Melting Point (deg C) Boiling Point (deg C)

Methane -162 -183

Ethane -89 -172

Propane -42 -188

Butane -0.5 -135

**print(“Melting and Boiling Points of Alkanes”)**

**print("{:<10s} {:<30s} {:<30s}".format(“Name”, "Melting Point (deg C)", "Boiling Point (deg C)")**

**print("{:<10s} {:<30d} {:<30d}".format(("Methane", -162, -183))**

**print("{:<10s} {:<30d} {:<30d}".format(("Ethane", -89, -172))**

**print("{:<10s} {:<30d} {:<30d}".format(("Propane", -42, -188))**

**print("{:<10s} {:<30.1f} {:<30d}".format(("Butane", -0.5, -135))**

**11. Pig Latin**

Pig Latin is a game of alterations played on words. To make the Pig Latin form of an

English word the initial consonant sound is transposed to the end of the word and an

“ay” is affixed. Specifically there are two rules:

(a) If a word begins with a vowel, append “yay” to the end of the word.

(b) If a word begins with a consonant, remove all the consonants from the beginning

up to the first vowel and append them to the end of the word. Finally, append “ay”

to the end of the word.

For example:

* dog ⇒ ogday
* scratch ⇒ atchscray
* is ⇒ isyay
* apple ⇒ appleyay

Write a program that repeatedly prompts for an English word to translate into Pig

Latin and prints the translated word. If the user enters a period, halt the program.

Hints:

* Slicing is your friend: it can pick off the first character for checking, and you can slice

off pieces and concatenate to yield the new word.

* Making a string of vowels allows the use of the **in** operator: vowels = 'aeiou' .

**vowels = “aeiou”**

**word = input(“Please enter a word: “)**

**while word != “.”:**

**word\_pig\_latin = “”**

**if word[0] in vowels:**

**word\_pig\_latin = word + “yay”**

**else:**

**first\_vowel = none**

**for i in range(len(word)):**

**if word[i] in vowels:**

**first\_vowel = i**

**if first\_vowel is not none:**

**word\_pig\_latin = word[first\_vowel:] + word[:first\_vowel] + “ay”**

**else:**

**word\_pig\_latin = word + “ay”**

**print(word\_pig\_latin)**

**word = input(“Please enter a word: “)**