ANA-522-OL1 Spring 2022
Mod01 Week01 HW: Python Part One
Due: Sunday January 16th at midnight

0.1 Caesar Cipher Wheel

The action of a Caesar cipher is to replace each plaintext letter with a different one a fixed number of places down the alphabet. Take a look at an example on Wikipedia to see how it works. You can also play around this Online Caesar Cipher Wheel to get the ideas.

0.2 Exploring strings and character codings

Character encoding is the process of representing individual characters using a corresponding encoding system made up of other symbols and types of data. The numerical values that make up a character encoding are known as "code points".

It is convenient to query about the numerical value of a character or symbol with the ord function. We can use help symbol (?) to lookup usages of a function such as ord.

[1]:
```
?ord
```

```
Signature: ord(c, /)
Docstring: Return the Unicode code point for a one-character string.
Type:      builtin_function_or_method
```

[2]:
```
# Note: the '/' above is as an argument marks the end of arguments that are␣
 ↪positional only


# Q: How to lookup code points for characters, say 'A','B','C' etc as Python␣
 ↪statements?
print("Unicode for character 'A' is", ord('A'))
print("Unicode for character 'B' is", ord('B'))
print("Unicode for character 'C' is", ord('C'))
print("Unicode for character 'a' is", ord('a'))
print("Unicode for character 'b' is", ord('b'))
print("Unicode for character 'c' is", ord('c'))
```

```
Unicode for character 'A' is 65
Unicode for character 'B' is 66
Unicode for character 'C' is 67
Unicode for character 'a' is 97
Unicode for character 'b' is 98
Unicode for character 'c' is 99
```

[3]:
```
#All right then. But, how to print alphabet of letters from 'A' to 'Z', or from␣
 ↪'a' to 'z', without having to type each letter in the statements?

#Q:Is there a convenient way to print all letters?
#A:The solution idea is to specify the code point one at a time.
#
#  We are using loop statement with the help of range() function to automate the␣
 ↪process.
#  Before getting there, firstly test the for lool with numbers first
#
for idx in range(26):
    print( idx, end=" ")
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

[4]:
```
# Now we want to adjust the range of the number to meet the numerical values of␣
 ↪the code points.
#
# From previous Unicode printouts we knew that code point ranges for upper and␣
 ↪lower alphabets are:
#  A - Z  : starting at 65
#  a - z  : starting at 97

#
# chr() function can be used to retrive a character by its code point. See the␣
 ↪usage first.
?chr
```

```
Signature: chr(i, /)
Docstring: Return a Unicode string of one character with ordinal i; 0 <= i <=␣
 ↪0x10ffff.
Type:      builtin_function_or_method
```

[5]:
```
# From all the above, we can print out all letters if specifying their code␣
 ↪points

for idx in range(26):
    print( chr(idx + 65), end= " ")
```

```
print()

for idx in range(26):
    print( chr(idx + 97), end= " ")
```

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

[6]:
```
# Now we have the tools needed to explore the Caesar Cipher.
#
# Let's try a Caesar Cipher with offet value, say 13, for all upper case letters␣
 ↪as an example
#
# Print out the original uppercase alphabet
#
for idx in range(26):
    print( chr(idx + 65), end= " ")

print()
#
# Print out with the uppercase alphabet with offset 13
#
offset = 13
for idx in range(26):
    print( chr(idx + 65 + offset), end= " ")
```

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
N O P Q R S T U V W X Y Z [ \ ] ^ _ ` a b c d e f g
```

It turned out that the result was not totally correct, since there were non-alphabet symbols came in to the mapping (look up ASCII code chart to see the table.)

[7]:
```
# let's fix it by focusing on the cycle of only uppercase letters. It means␣
 ↪after done with Z, the next letter should be back to A and continuing.
# To do so, we use % (mod) operator with 26, which is the size of total␣
 ↪uppercase letters.
#
# Print out with the offset 13
#
offset = 13
for idx in range(26):
    print(chr( (idx+offset)%26 + 65), end= " ")

# We fixed it by applying the offset and the cycle of only 26 upppercase␣
 ↪letters, while applying the starting code point, where A is 65, to every␣
 ↪letter.
```

N O P Q R S T U V W X Y Z A B C D E F G H I J K L M

[8]:
```
## Given above, now it's time to demystify some secret messages using Caesar Cipher.

## Use the above, can you solve the encryped message.

# What does this string say?

'Gur Mra bs Clguba, ol Gvz Crgref'

# The string was encryped with an offset equals to 13 using Caesar Cipher Wheel
# Can you implement Python statements to retrieve the original message?
```

[8]:
```
'Gur Mra bs Clguba, ol Gvz Crgref'
```

0.3 Encryption and Decryption with Caeser Ciher

[9]:
```
###
### Homework Question One.
###
# Your work starts here, given the secret message is stored in the secret variable in the following.

secret = 'Gur Mra bs Clguba, ol Gvz Crgref'
```

[10]:
```
###
### Homework Question Two.
###

# Once the previous secret message is retrieved,
# use the secret to create a new secret message using caesar cipher with offset equals to 11
# Then print out the new encrypted message
#
```