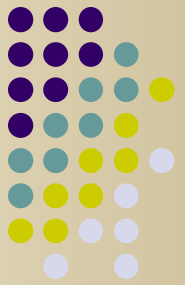
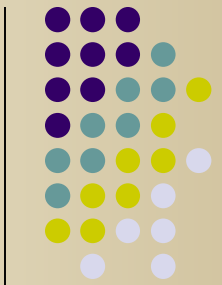
ENGR/CS 101 CS Session Lecture 6



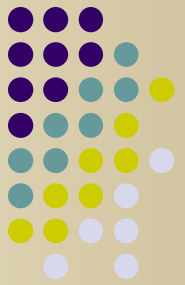
- Log into Windows/ACENET (reboot if in Linux)
- Start Python, create a New File, and Save As
- Reminder: Homework 1 is due today by 4:30pm via the submission system.

Outline



- Problem: Encode and decode a text file using the Caesar shift cipher
  - Today: encode user entered word in uppercase
- New Python concepts
  - Characters
  - For loops
  - Characters and strings as user input

Problem Specification

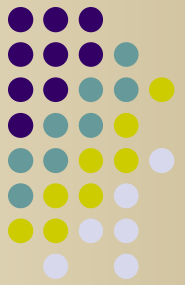


- Will work our way up to a program to do the Caesar shift cipher on messages in a file. Today's program will do the following:
  - Allow the user to enter a shift key letter in uppercase
  - Repeat until the user enters '#' for a word
    - Allow the user to enter a plaintext word in uppercase to be enciphered
    - Compute and display the corresponding ciphertext word

Sample Run

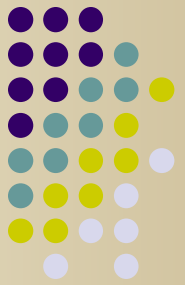
```
>>> main()
Enter a shift key in uppercase: I
Enter a word in uppercase to encrypt (# to quit): GO
The encrypted word is: OW
Enter a word in uppercase to encrypt (# to quit): ACES
The encrypted word is: IKMA
Enter a word in uppercase to encrypt (# to quit): #
All done
>>>
```

Characters



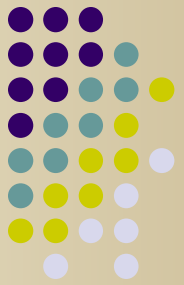
- In programming languages, ***character*** is a common data type.
- Characters are the alphabet, digits, punctuation, and whitespace. As noted in Lecture 1, the encoding of characters to binary digits is called ASCII.

Characters



- Typically, character literal values are enclosed in single quotes.
  - E.g., `'A', '3', '?'`
- Special characters are 'escaped'
  - Newline – `'\n'`
  - Tab – `'\t'`
  - Single quote – `'\''`

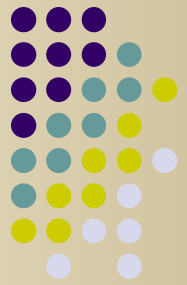
Character Functions



- Often it is useful to use the underlying number that represents a character.
- In Python, there are two functions for this
  - **`ord(ch)`** – return ASCII numeric equivalent of **`ch`**
  - **`chr(num)`** – return character corresponding to ASCII value **`num`**

```
>>> ord('a')
97
>>> chr(103)
'g'
```

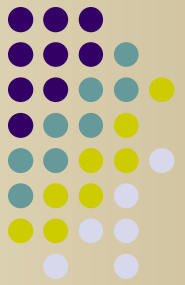
Computing the Cipher Letter



- Assume that variable shiftKey holds the key letter and variable plainLetter holds the letter to be enciphered.
- Since the alphabetic characters have sequential mapping (i.e., 'A' is first, followed by 'B', etc.), the number of places to shift is the key letter minus 'A'. In Python code, this is:

```
shiftNumber = ord(shiftKey) - ord('A')
```

Computing the Cipher Letter

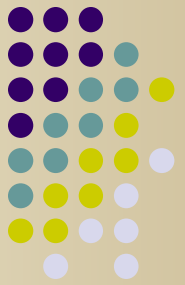


- To find the cipher letter, we determine the ***index*** of plaintext letter (i.e., where in the alphabet it is when we start counting at 0) using a similar method, then add the shift number.
- This will be the index of the ciphertext letter, except that the number may be greater than 26. To make it circular, we compute the ***modulus*** with respect to 26. In code, this is:

```
index = ((ord(plainLetter) - ord('A')
          + shiftNumber) % 26)
```

- The modulus operator symbol is %

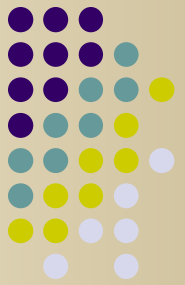
Computing the Cipher Letter



- Now we add this new index back to 'A' to find the ciphertext letter.

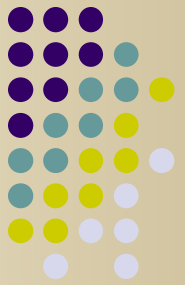
```
cipherLetter = chr(ord('A') + index))
```

For Loops



- For-loops are used when the number of iterations are known.
- Typically, they are used for counting or for accessing each element of a collection.
- Since a string is a sequence of characters, Python allows a string to be used as a collection. The first iteration accesses the first character, etc.

For-Loops



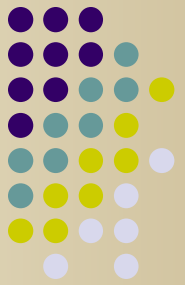
- The syntax of a for-loop in Python is:

```
for <var> in <collection>:
      <steps to be repeated>
```

- Each value in **<collection>** is assigned to **<var>** in turn
- For our program, we would write:

```
for plainLetter in plaintext:
    # steps to be repeated
```

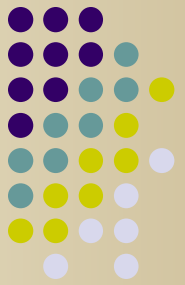
String Concatenation



- To create the ciphertext string, the cipher letters must be concatenated together.
- The + operator on strings can be used to append the cipher letter to the end of the ciphertext string
- To accumulate the letters, assign the result back to the ciphertext variable

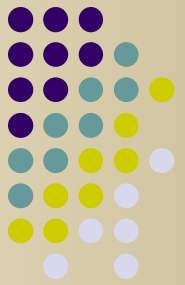
```
ciphertext = ciphertext + cipherLetter
```

encodePlaintext Function



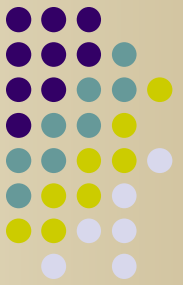
- Specification
  - Write a function the encodes a plaintext word into a ciphertext word using a shift key letter and return the ciphertext word
- Analysis – Identify the data being used
  - Receives: plaintext word, shift key letter
  - Returns: ciphertext word

encodePlaintext Function



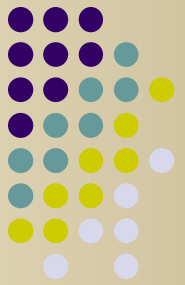
- Design – Write the steps of an algorithm
  1. Initialize ciphertext word to the empty string
  2. Compute the shift number
  3. For each letter in the plaintext word
     a. Compute the index of the letter
     b. Compute the cipher letter
     c. Append the cipher letter to the ciphertext word
  4. Return the ciphertext word

encodePlaintext Function Code



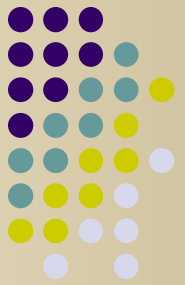
```
# function to encode plaintext using Caesar
# shift cipher

def encodePlaintext (plaintext, shiftKey):
    ciphertext = ""
    shiftNumber = ord(shiftKey) - ord('A')
    for plainLetter in plaintext:
        index = ((ord(plainLetter) - ord('A')
                  + shiftNumber) % 26)
        cipherLetter = chr(ord('A') + index)
        ciphertext = ciphertext + cipherLetter
    return ciphertext
```

Main Program Function



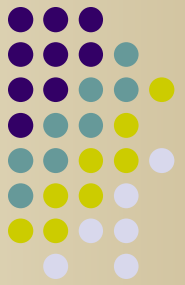
- Analysis - Identify the data being used
  - Input: shift key, plaintext string
  - Output: ciphertext string

Main Program Function



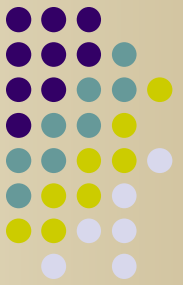
- Design - Write the steps of an algorithm
  1. Ask the user for the shift key letter
  2. Ask the user for a plaintext word
  3. While the plaintext word is not '#'
     a. Compute the ciphertext word using the function
     b. Display the ciphertext word
     c. Ask the user for another plaintext word

Raw Input



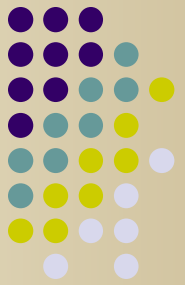
- The Python **`input`** function requires input to be in the format expected by the interpreter, which requires characters and strings to have surrounding quotes.
- The Python **`raw_input`** function reads the entered data to the newline and returns a string. This is exactly what we need for this program.

Main Program Function Code



```
# Main program - ask user for a shift key and
# repeatedly encode words until user wants to quit

def main():
    key = raw_input ('Enter a shift key in uppercase: ')
    message = raw_input('Enter a word in uppercase \
                         to encrypt (# to quit): ')
    while message != '#':
        secret = encodePlaintext(message, key)
        print 'The encrypted word is:', secret
        message = raw_input('Enter a word in uppercase \
                         to encrypt (# to quit): ')

    print 'All done'
```

Test the Program



- When you have completed writing the program, try running it with different keys and words.
- What happens if you try to input multiple words or lowercase letters?
- Next class we will enhance the project to handle encoding entire sentences.