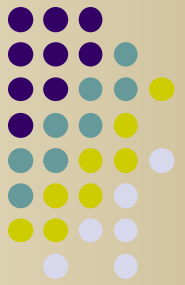
ENGR/CS 101 CS Session Lecture 8

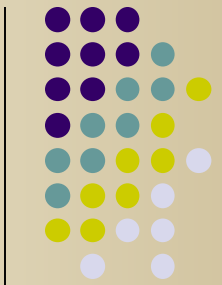


- Log into Windows/ACENET (reboot if in Linux)
- Start Python, open program from last time.
- Has everyone finished the program from last class so that it can encipher an arbitrary message?
- Create a New File and Save As it, copy the **`encodePlaintext`** function from previous program.

Outline

- Go over Homework 1
- Menu-driven loops
  - Functions without parameters
- Problem: Deciphering secret messages
- Error checking

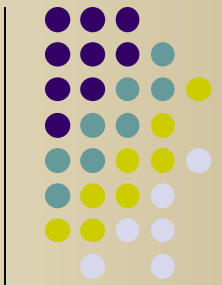
Menu-Driven Loops



- One common loop structure is a menu-driven loop. The user is presented with a menu of choices and asked to enter one. The choice determines what action is taken.

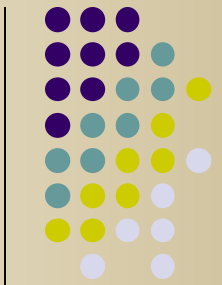
```
Choose an option:
   1. Encrypt a message
   2. Decrypt a message
   3. Quit
Enter your choice: 1
Enter a message to encrypt: Go Aces!
The encrypted message is: Ow Ikma!
Choose an option:
   1. Encrypt a message
   ...
```

getUserChoice function



- To keep the main program from becoming cluttered, usually it is best to have a function that displays the menu, asks the user for her choice, and returns that choice
- Analysis: no parameters
- Design
  1. Display menu
  2. Ask user for a choice
  3. Return choice

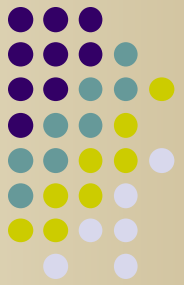
In-class Exercise, Part 1



- Write a Python function that implements the **`getUserChoice`** function.
- Run the module and test the function

```
>>> getUserChoice() # note, need parens to call
Choose an option:
   1. Encrypt a message
   2. Decrypt a message
   3. Quit
Enter your choice: 1
'1'
>>>
```

In-class Exercise, Part 1

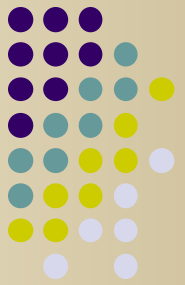


- For the main program, write a menu-driven loop using **`getUserChoice`**, that repeats until the user enters 3.

```
choice = getUserChoice()
while choice != '3':
    choice = getUserChoice()
```

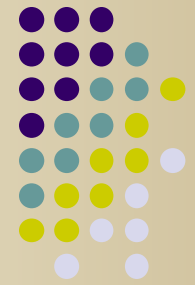
- Inside the loop, write an if-statement to test if choice is '1', and if so, asks the user for a message, computes the encrypted message, and displays it.

Problem: Deciphering



- One of the reasons the Caesar shift cipher is not a good method for keeping secrets is that it is easy to decipher when you know the key.
- We could decipher by subtracting the shift number from the cipherletter index rather than adding it.
- Unfortunately, the % (modulus) operator only works reliably on positive numbers.

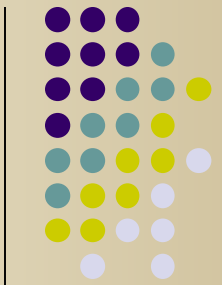
Complementary Shift Number



- But in modular arithmetic, subtracting is the same as adding its complementary number (modulus base minus the number). In our case, this will be 26 minus the shift number.
- For example, shift key 'I', gives a shift number of 8. The complementary shift number for deciphering is 26 - 8 = 18. (This is equivalent to a shift key of 'S'.)

```
>>> encodePlaintext('Ow Ikma!', 'S')
>>> Go Aces!
```

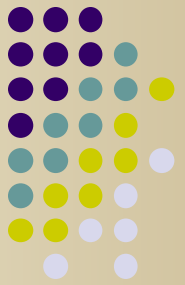
Complementary Shift Number



- This means that we can decipher using the exact same algorithm as for enciphering, except that the shift number computation becomes:

```
shift = 26 - (ord(key) - ord('A'))
```

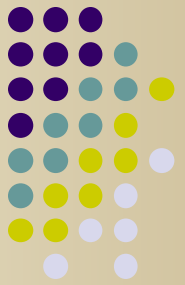
One Function for Both



- Since deciphering is exactly the same as enciphering, we can use the same function as long as we pass in the shift number rather than the shift key.
- First, we move the computation of the shift number from the function to the main program in the if-statement for encrypting.

```
if choice == '1':
    shift = ord(key) - ord('A')
    ...
```

One Function for Both



- Then we change the second parameter of the function to **`shiftNumber`**

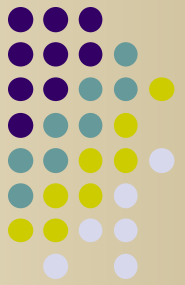
```
>>> encodePlaintext('Ow Ikma!', 18)
>>> Go Aces!
```

- Finally, we change the second argument to the function call to be the shift number (rather than the shift key letter)

```
secret = encodePlaintext(message, shift)
```

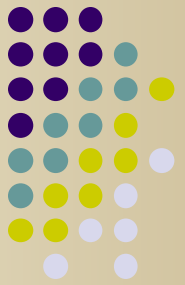
- Run the program. It should behave the same as before.

In-class Exercise, Part 2



- In the main program loop, add a branch to the if-statement to process a choice of '2' to decrypt a secret message. The steps are
  a. Compute the ***complementary*** shift number
  b. Ask the user for a ciphertext message
  c. Compute the plaintext message by calling the function with the secret message and the complementary shift number as arguments
  d. Display the plaintext message

In-class Exercise, Part 2

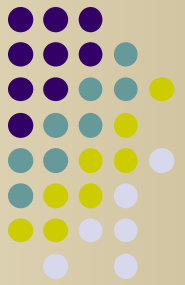


- Run the program and test the decrypt choice

```
Choose an option:
   1. Encrypt a message
   2. Decrypt a message
   3. Quit
Enter your choice: 2
Enter a message to decrypt: Ow Ikma!
The decrypted message is: Go Aces!
```

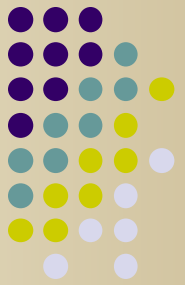
- At this point, the name of the function should be something generic like **`shiftText`**

Error handling



- When a user provides an illegal input, a program should try to handle such an error so that the user is informed that the input is illegal and so that the program doesn't provide erroneous results or crash.
- E.g., what happens if the user inputs a non-uppercase alphabetic character for the shift key? What happens when user enters 4 for the menu choice?

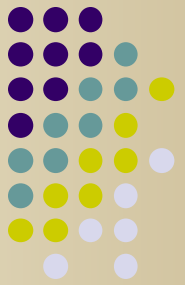
In-class Exercise, Part 3



- Add code to the main program so that if the user does not enter an uppercase letter for the shift key, the program displays an error message and quits

```
Enter a shift key in uppercase: i
The shift key must be in uppercase!
All done
```

In-class Exercise, Part 3



- Add an else section to the if-statement in the main program loop that displays an error message if the user does not enter 1, 2, or 3

```
Choose an option:
   1. Encrypt a message
   2. Decrypt a message
   3. Quit
Enter your choice: 4
Bad choice, try again
Choose an option:
   1. Encrypt a message
   ...
```