Why can't elephants use computers?

Because they're scared of the MOUSE!!

Source: https://heresajoke.com/computer-jokes/

Thank you Hayden!
Last Lecture

• Continued investigating the topic of Cryptography and Encryption
  1. By finishing the implementation of our OddEvenEncryption program
     • Implementing the encryption algorithm
  2. By adding decryption to our OddEvenEncryption program

• Introduced functions
Today’s Menu

• Finish implementing the decryption algorithm

• Create functions in our OddEvenEncryption program
  • Encrypt()
  • Decrypt()

• We shall look at another way of iterating Python statements in our programs
  • while loop
Transposition algorithm \textit{odd\&even}:

1. Find the middle of \textit{cipherMsg}.
2. Store the first half of \textit{cipherMsg} into \textit{String3}.
   - This first half contains the characters originally located in \textit{odd positions} of \textit{plainMsg}.
3. Store the last half of \textit{cipherMsg} into \textit{String4}.
   - This second half contains the characters originally located in \textit{even positions} of \textit{plainMsg}.
4. Lastly, merge \textit{String3} with \textit{String4} to recreate \textit{plainMsg}.

**Definition:** Algorithm that shuffles elements from their original positions in a sequence to new positions!
So far …

… we have used functions that were already built into Python by calling them

• Built-in functions (some came from modules)
  • For example: `print(...), input(...), type(...), random.randint(1, 10)`

• Built-in methods
  • For example: `<string>.upper( ), <string>.isalpha( )`
Why creating functions?

Functions make our program easier to ...

1. Implement and test -> Incremental development
   • Dividing a long program into functions allows us to implement, test and debug the parts one at a time and then assemble them into a working program

2. Read
   • Encapsulate code fragment that does one thing (functionality) in one location, i.e., a “module” (function) and give this location a descriptive name

3. Modify
   • If we need to make a change to our program, we know where to go, i.e., where to find the code fragment we need to change

4. Reuse
   • Once we write, test and debug a function, we can reuse it in other programs that need this particular functionality

5. No more repeated code
   • Functions can make a program smaller by eliminating repeated code - Repeated code is very error-prone
Review - Function

Syntax of function definition

\textbf{def} -> means "here is the definition of a function"

\begin{verbatim}
def <functionName>( [parameter(s)] ) :
    < 1 or more statements >
    return [expression]
\end{verbatim}

- **GPS about <functionName>**
  - Function name is descriptive -> it describes the purpose of the function
  - Function name syntax: same as for variable name syntax

Function header

Body of the function

1 return statement
Execution flow and functions

• Let’s examine what happens to the execution flow when we call functions
  • using the Python code visualizer
From last lecture: Your turn

• **Problem Statement:**
  • Write a program that encrypts and decrypts messages using the transposition algorithm odd&even

• **Requirement:**
  • Your program must go on encrypting and decrypting messages entered by the user until the user only presses the ENTER key.
Review - Syntax of a \textbf{while} loop

\begin{verbatim}
<stmt before loop:
    initialize condition variable>

while <Boolean condition> :
    <first statement to be repeated>
    <second statement to be repeated>
    ...
    <stmt: modify condition variable>

<statement outside (after) the loop>
\end{verbatim}
Review - Syntax of a `while` loop

```plaintext
<stmt: before loop: initialize condition variable> >
while <Boolean condition> :
  <first statement to be repeated>
  <second statement to be repeated>
  ...
  <stmt: modify condition variable>
<statement outside (after) the loop>
```

- **Important** – About Indentation
  - *Statements inside the loop* (i.e., statements executed at each iteration of the loop) are the statements indented with respect to the `while` keyword
  - *Statements outside the loop* (before and after the loop) are the statements that are *not* indented with respect to the `while` keyword – these statements are considered to be at the same level of indentation as the `while` loop
Review - Difference between while and for loops

while <Boolean condition> :
  <first statement to be repeated>
  <second statement to be repeated>
  ...
  <stmt: modify condition variable>
  <statement outside (after) the loop>

for <iterating variable> in <sequence> :
  <first statement to be repeated>
  <second statement to be repeated>
  ...
  <last statement to be repeated>
  <statement outside (after) the loop>
When best to use a `while` loop

- If there is a **condition** that will occur during the execution of our program and when this condition occurs, the execution of a set of statements in our program needs to stop, then we use a `while` loop
- This **condition** is often called a **sentinel** or **flag**
  - Examples:
    - User termination
    - User presses the ENTER key without typing anything -> empty string
    - User enters yes/no or some special value
    - User selects ‘X’ to eXit from a menu (menu-driven program)
    - Occurrence of an error
    - Reading data from a file -> EOF
When best to use a for loop

- If we know exactly how many times we must iterate a set of statements in our program, then we use a for loop.
GPS: We cannot use a **while** loop with a **True** condition:

```python
while True :
    break
    exit()
```

Can you see why?
Next Lecture

• Practice Exam #3