CMPT 120
Lecture 9 – Cryptography and Encryption –
The realm of secret codes
Python – Arithmetic operators, order of evaluation (operator precedence) and string/list manipulation
Last Lecture

Improving our guessing game

• Wouldn’t it be nice to play our guessing game many times without having to press Run over and over again?

• New Problem Statement
  • Write a guessing game, which allows a player to guess a number between 1 and 10 in 3 guesses!

• Let’s get coding!
Review - range() function

• Very useful in for loop
• Syntax: `range([start,] stop [,step])`
• Produces a list of integers
• How does it work?
  • If `theLength = 5`
  • Then `range(theLength)` produces the following sequence: `0, 1, 2, 3, 4`
Repeated code

-> Bad idea!

- What do you mean by repeated code?
- If the problem statement is: List some movies, then Solution 1 would solve the problem using repeated code -> bad idea!
  - Solution 1:
    ```python
    print("Superman")
    print("Frozen")
    print("X-Men")
    ```
- Solution 2 would not -> good idea!
  - Solution 2:
    ```python
    movies = ["Superman", "Frozen", "X-Men"]
    for movie in movies:
        print(movie,"!")
    ```

If you need to repeat some of your code in your program, do not repeat the code itself by copying and pasting it, use a loop instead!

This semester, let’s not do this! 😞

This semester, let’s do this! 😊
Todays’ Menu

• Introducing **Cryptography and Encryption**

• Can we build programs that create secret(encrypted) messages using
  • Arithmetic operators
  • String and List indexing and slicing mechanism
  • etc...

• Let’s see 😊
Cryptography and Encryption - Secret codes

• When you log onto your SFU mail account, purchase goods on the Internet, check your grades or your bank account online, you are using cryptography.

• **Cryptography** is a field of study involving many disciplines such as mathematics, **computing science**, and engineering.

• It encompasses various aspects such as data confidentiality, data integrity, and authentication.

• **Encryption** is one of the aspects of cryptography that deals with the process of encoding a message using an algorithm.
Cryptography and Encryption - Secret codes

- In this unit, we'll see how we can encrypt/decrypt messages using Python programs.
- To do so, we’ll need to learn:
  - How to manipulate characters in a string
  - How to manipulate elements in a list
  - How to calculate
  - How to loop in various ways
  - How to write our own functions
Encryption - Secret codes

- Makes messages confidential, secret
- It does this by using an algorithm that transforms messages we can read into messages we can no longer read ... and back to messages we can read

- We shall call
  - messages we can read `plainMsg`
  - messages we cannot read `cipherMsg`
Let’s give it a go!

• **Step 1 - Problem Statement:**
  • Write a Python program that *encrypts* messages using a *transposition algorithm* called *odd&even*
Transposition algorithm odd&even:

1. Get **plainMsg** from user
2. Create a **cipherMsg** that is made of 2 strings
   - **String1** contains the characters located in odd positions in **plainMsg**
   - **String2** contains the characters located in even positions in **plainMsg**
3. Lastly, concatenate these two strings:
   
   $$\text{cipherMsg} = \text{String1} + \text{String2}$$
An example:

• Let’s encrypt this message "Hello, World!"
Let’s give it a go! (cont’d)

- **Step 2 – Design**
  Transposition algorithm **odd&even**:
  1. Get **plainMsg** from user
  2. Create a **cipherMsg**:
     For each character in the **plainMsg**
     - If the character is at an odd position in **plainMsg**
     - Then this character goes into **String1**
     - Otherwise it goes into **String2**
  3. Lastly, concatenate these two strings:
     
     **cipherMsg = String1 + String2**
Let’s give it a go! (cont’d)

- **Step 3 – Implementation**
  In order to implement our encryption algorithm, we need to know ...
  1. String concatenation
  2. Arithmetic operator -> modulo operator
      - Example:
  3. Order of evaluation (precedence)
      - Example:

  4. Running count (accumulator) algorithm
Let’s give it a go! (cont’d)

• Step 4 - Testing
Your turn: Let's decrypt our message!

**Step 1 - Problem Statement:**
- Write a program that **decrypts** messages that have been **encrypted** using the **transposition algorithm** **odd&even**
Review - String indexing: positive indexing

How to access one string character at a time?

Answer: Use the index associated with the character as illustrated below:

```
positive indexing-→ index: 0 1 2 3 4 5 6 7 8 9 10 11 12
                      ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑
```

Example: `message = "Hello, World!"`

- So if we wish to access
  - The 1ˢᵗ character of the string, we use the index 0
  - The 2ⁿᵈ character of the string, we use the index 1
  - etc...
Review - String indexing: positive indexing examples

```
>>> message = "Hello, World!"
>>> message
'Hello, World!'
>>> message[0]
'H'
>>> message[12]
'!'
>>> message[6]
'
>>> message[15]
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    message[15]
IndexError: string index out of range
>>> message[-2]
'd'
>>> 
```

Careful: Positive index starts at 0
There is another way we can use to access one string character at a time: negative indexing:

Example: `message = "Hello, World!"

negative indexing: \( \text{index: } -13 \rightarrow -12 \rightarrow -11 \rightarrow -10 \rightarrow -9 \rightarrow -8 \rightarrow -7 \rightarrow -6 \rightarrow -5 \rightarrow -4 \rightarrow -3 \rightarrow -2 \rightarrow -1 \)

So if we wish to access

- The 1\(^{st}\) character of the string, we use the index -13
- The last character of the string, we use the index -1,
- etc...
Review - String indexing: negative indexing examples

Careful: Negative index starts at -1, not 0
How to access a section (slice) of a string at a time?

Answer: use indices to indicate the string slice

```
positive indexing-> index:    0 1 2 3 4 5 6 7 8 9 10 11 12
```

Example: `message = "Hello, World!"`

Syntax: `<aString>[start : stop : step]`

- start
- stop
- step

Example: So if we wish to access the string slice "Hello", we use `message[0:5]`
Review - How does String slicing works?

Positive indexing

<table>
<thead>
<tr>
<th>index:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
</table>

Example: message = "Hello, World!"

- **message[0:5]**
  - We use index 0 to indicate the **start** of the string slice
    - **Inclusive** -> the character at index 0 is included in the string slice
  - We use index 5 to indicate the **stop** of the string slice
    - **Non-inclusive** -> the character at index 5 is ***not*** included in the string slice
Note what happens when `stop` represents an index that is **out of range**, i.e., the index 25 no longer correspond to a character of the string `message` since this string only has 13 characters, i.e., from index 0 to index 12. So, Python interprets the index 25 to mean “all the way to the end of the string”. Therefore, it creates a slice of the string `message` from its character at index 7 all the way to its last character (because the index of this last character is < 25).
## Review - Strings (sequence) manipulation

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Operator/function</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>concatenation</td>
<td>+</td>
<td>Combine strings together</td>
</tr>
<tr>
<td>repetition</td>
<td>*</td>
<td>Concatenate a string that is being repeated a number of times</td>
</tr>
<tr>
<td>indexing</td>
<td>[n]</td>
<td>Access an element of a string</td>
</tr>
<tr>
<td>slicing</td>
<td>[ :: ]</td>
<td>Extract a part of a string</td>
</tr>
<tr>
<td>length</td>
<td>\textit{len(}aString\textit{)}</td>
<td>Determine the number of characters in a string \textit{aString}</td>
</tr>
</tbody>
</table>
Review: Arithmetic operators

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Floor division: //
- Modulus: %
- Exponentiation: **

Syntax: <operand> <operator> <operand>

Running count (accumulator) algorithm
- Example: charCount = charCount + 1
  OR charCount += 1
# Review: Order of Evaluation

<table>
<thead>
<tr>
<th>Highest precedence</th>
<th>Parentheses</th>
<th>Parentheses P</th>
</tr>
</thead>
<tbody>
<tr>
<td>(expressions...)</td>
<td>Parentheses</td>
<td>Parentheses P</td>
</tr>
<tr>
<td>x[index],</td>
<td>Indexing</td>
<td>Indexing P</td>
</tr>
<tr>
<td>x[index : index],</td>
<td>(aka</td>
<td>(aka P</td>
</tr>
<tr>
<td>x(arguments...)</td>
<td>Subscription)</td>
<td>Subscription)</td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>*, /, //, %</td>
<td>Multiplication, division</td>
<td>Multiplication, division E</td>
</tr>
<tr>
<td>+, -</td>
<td>Addition and subtraction</td>
<td>Addition and subtraction AS</td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;, &gt;=, !=, ==</td>
<td>Relational operators</td>
<td>Relational operators AS</td>
</tr>
<tr>
<td>not</td>
<td>Logical operator</td>
<td>Logical operator</td>
</tr>
<tr>
<td>and</td>
<td>Logical operator</td>
<td>Logical operator</td>
</tr>
<tr>
<td>or</td>
<td>Logical operator</td>
<td>Logical operator</td>
</tr>
<tr>
<td></td>
<td>Lowest precedence</td>
<td>Lowest precedence</td>
</tr>
<tr>
<td>Lowest precedence</td>
<td>Logical operator</td>
<td>Logical operator</td>
</tr>
<tr>
<td></td>
<td>Logical operator</td>
<td>Logical operator</td>
</tr>
<tr>
<td></td>
<td>Logical operator</td>
<td>Logical operator</td>
</tr>
</tbody>
</table>
Next Lecture

• Practice Exam #2 😊