Introduction to Python

Data Structures
Data Structures

- Encapsulation & Notion of an Object
  - Data + a set of methods (functions) that operate on the data
    - A.foo()
- Linear Data Structure:
  - List, Strings, sequences
Lists and ranges

- A **list** is a sequence of values enclosed in brackets
  - Example: `courses = ['CIT 591', 'CIT 592', 'CIT 593']`

- You can refer to an individual value by putting a bracketed number (starting from 0) after the list
  - Example: `courses[2]` is 'CIT 593'

- The **len** function tells you how many things are in a list
  - Example: `len(courses)` is `3`

- **range** is a function that creates a list of integers, from the first number up to *but not including* the second number
  - Example: `range(0, 5)` creates the list `[0, 1, 2, 3, 4]`

- If you give **range** a third number, it is used as the step size
  - Example: `range(2, 10, 3)` creates the list `[2, 5, 8]`
Lists

• Construction
  - Syntax: [elem1, elem2, ...]
  - Heterogeneous, ordered sequence
  - Mutable
  - Example:
    >>> list1 = [1, 'hello', 4+2j, 123.12]
    >>> list1
    [1, 'hello', (4+2j), 123.12]
    >>> list1[0] = 'a'
    >>> list1
    ['a', 'hello', (4+2j), 123.12]
Lists - Operations

- **Concatenation (+)**
  - Syntax: `list1 + list2`
  - Example:
    ```python3
    >>> [1, 'a', 'b'] + [3, 4, 5]
    [1, 'a', 'b', 3, 4, 5]
    ```

- **Repetition (*)**
  - Syntax: `list * number`
  - Example:
    ```python3
    >>> [23, 'x'] * 4
    [23, 'x', 23, 'x', 23, 'x', 23, 'x']
    ```
Indexing

- Indexing operator: [ ]
- Positive indices count from the left
- Negative indices count from the right

```
sequence[0] == a     sequence[-7] == a
sequence[6] == g     sequence[-1] == g
sequence[2] == c     sequence[-5] == c
```
Slices

• **Two indices separated by a colon**
  - Available for both strings and lists
  - Example
    ```python
    >>> sequence = [0, 1, 2, 3, 4, 5, 6, 7]
    >>> sequence[1:4]
    [1, 2, 3]
    >>> sequence[2:-1]
    [2, 3, 4, 5, 6]
    - Missing Index implies end point
      ```python
      >>> sequence[:2]
      [0, 1]
      >>> sequence[3:]
      [3, 4, 5, 6, 7]
    ```
Built-in Function: len

- **Syntax:** `len(object)`
  - Return the length of `object`
  - Example
    ```python
def example_usage(list1):
    return len(list1)

def example_usage(string1):
    return len(string1)
```

```python
>>> list1 = [1, 2, 3, 4, 5]
>>> len(list1)
5

>>> string1 = "length of a string"
>>> len(string1)
18
```
Built-in Function: range

- **Syntax:** `range([start,] stop[, step])`
  - Generate a list of numbers from `start` to `stop` stepping every `step`
  - `start` defaults to 0, `step` defaults to 1
  - Example

    >>> range(5)
    [0, 1, 2, 3, 4]
    >>> range(1, 9)
    [1, 2, 3, 4, 5, 6, 7, 8]
    >>> range(2, 20, 5)
    [2, 7, 12, 17]
Lists - Methods

- **pop**
  - Syntax: `list.pop([index])`
  - Remove and return item at position `index` from `list`
  - Default is to remove last item
  - Example:
    ```python
    >>> list1 = [3, '10', 2, 9, 11]
    >>> list1.pop()
    11
    >>> list1
    [3, '10', 2, 9]
    ```
Lists - Methods

- **insert**
  - Syntax: `list.insert(index, element)`
  - Insert element into list at position index
  - Example:
    ```python
    >>> list2 = [0, 1, 2, 3, 4, 5]
    >>> list2.insert(3, 'new')
    >>> list2
    [0, 1, 2, 'new', 3, 4, 5]
    ```
Extend and append

- Syntax: `list.append(element), list.extend(list2)`
- Insert element/concatenate list2 into list at end

Example:

```python
>>> list1 = [0, 1, 2, 3, 4, 5]
>>> list1.append( 'new')
>>> list1
[0, 1, 2, 3, 4, 5, 'new']
>>> list2 = [8, 9, 10]
>>> list1.append( list2 )
>>> list1
[0, 1, 2, 3, 4, 5, 'new', 8, 9, 10]
```
Lists - Methods

• **remove**
  - Syntax: `list.remove(element)`
  - Removes the first occurrence of `element` in `list`
  - Example:
    ```python
    >>> list2 = [0, 1, 3, 4, 3, 5]
    >>> list2.remove(3)
    >>> list2
    [0, 1, 4, 3, 5]
    ```
Lists - Methods

- reverse
  - Syntax: `list.reverse()`
  - Reverse elements of `list` in place
  - Example:
    ```python
    >>> list3 = [4, 12, 3, 9]
    >>> list3.reverse()
    >>> list3
    [9, 3, 12, 4]
    ```
Lists - Methods

- **sort**
  - Syntax: `list.sort(cmpfunc])`
  - Sort list in place
  - Example:
    ```python
    >>> list3 = [4, 12, 3, 9]
    >>> list3.sort()
    >>> list3
    [3, 4, 9, 12]
    ```
The String Data Type

• The most common use of personal computers is word processing.
• Text is represented in programs by the string data type.
• A string is a sequence of characters enclosed within quotation marks (""" or apostrophes (')).
The String Data Type

```python
>>> str1="Hello"
>>> str2='spam'
>>> print(str1, str2)
Hello spam
>>> type(str1)
<class 'str'>
>>> type(str2)
<class 'str'>
```
The String Data Type

• Getting a string as input

```python
>>> firstName = input("Please enter your name: ")
Please enter your name: John
>>> print("Hello", firstName)
Hello John
```

• Notice that the input is not evaluated. We want to store the typed characters, not to evaluate them as a Python expression.
The String Data Type

• We can access the individual characters in a string through *indexing*.

• The positions in a string are numbered from the left, starting with 0.

• The general form is `<string>[(<expr>)]`, where the value of expr determines which character is selected from the string.
The String Data Type

```
>>> greet = "Hello Bob"
>>> greet[0]
'H'
>>> print(greet[0], greet[2], greet[4])
H l o
>>> x = 8
>>> print(greet[x - 2])
B
```
The String Data Type

- In a string of \( n \) characters, the last character is at position \( n-1 \) since we start counting with 0.
- We can index from the right side using negative indexes.

```python
>>> greet[-1]
'b'
>>> greet[-3]
'B'
```
The String Data Type

- Indexing returns a string containing a single character from a larger string.
- We can also access a contiguous sequence of characters, called a *substring*, through a process called *slicing*. 
The String Data Type

• Slicing:
  `<string>[<start>:<end>]`
• start and end should both be ints
• The slice contains the substring beginning at position start and runs up to **but doesn’t include** the position end.
The String Data Type

>>> greet[0:3]
'Hel'
>>> greet[5:9]
' Bob'
>>> greet[:5]
'Hello'
>>> greet[5:]
' Bob'
>>> greet[:]
'Hello Bob'
The String Data Type

- If either expression is missing, then the start or the end of the string are used.
- Can we put two strings together into a longer string?
- **Concatenation** “glues” two strings together (+)
- **Repetition** builds up a string by multiple concatenations of a string with itself (*)
The String Data Type

- The function `len` will return the length of a string.

```python
>>> "spam" + "eggs"
'spameggs'
>>> "Spam" + "And" + "Eggs"
'SpamAndEggs'
>>> 3 * "spam"
'spamspamspam'
>>> "spam" * 5
'spamspamspamspamspam'
>>> (3 * "spam") + ("eggs" * 5)
'spamspamspameggseggseggseggseggsegs'
```
# The String Data Type

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Concatenation</td>
</tr>
<tr>
<td>*</td>
<td>Repetition</td>
</tr>
<tr>
<td><code>&lt;string&gt;[:]</code></td>
<td>Indexing</td>
</tr>
<tr>
<td><code>&lt;string&gt;[:,:]</code></td>
<td>Slicing</td>
</tr>
<tr>
<td><code>len(&lt;string&gt;)</code></td>
<td>Length</td>
</tr>
</tbody>
</table>
Simple String Processing

- Usernames on a computer system
  - First initial, first seven characters of last name

```python
# get user's first and last names
first = input("Please enter your first name (all lowercase): ")
last = input("Please enter your last name (all lowercase): ")

# concatenate first initial with 7 chars of last name
uname = first[0] + last[:7]
```
Simple String Processing

• Another use – converting an int that stands for the month into the three letter abbreviation for that month.
• Store all the names in one big string: “JanFebMarAprMayJunJulAugSepOctNovDec”
• Use the month number as an index for slicing this string:
  monthAbbrev = months[pos:pos+3]
Simple String Processing

<table>
<thead>
<tr>
<th>Month</th>
<th>Number</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Feb</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Mar</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Apr</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

- To get the correct position, subtract one from the month number and multiply by three
# month.py  
# A program to print the abbreviation of a month, given its number

def main():

    # months is used as a lookup table
    months = "JanFebMarAprMayJunJulAugSepOctNovDec"

    n = eval(input("Enter a month number (1-12): "))

    # compute starting position of month n in months
    pos = (n-1) * 3

    # Grab the appropriate slice from months
    monthAbbrev = months[pos:pos+3]

    # print the result
    print ("The month abbreviation is", monthAbbrev + ".")

main()
Simple String Processing

```python
>>> main()
Enter a month number (1-12): 1
The month abbreviation is Jan.
>>> main()
Enter a month number (1-12): 12
The month abbreviation is Dec.
```

- **One weakness** – this method only works where the potential outputs all have the same length.
- **How could you handle spelling out the months?**