CMPT 120

Topic: Python strings
Last Lecture

• Continue learning Python statements (Python’s building blocks)
  • Categories of Statements
    4. Operational statements
      • Mathematical/arithmetic operators
      • Order of operations

as we practice Step 5 of the software development process -> Testing and Debugging

• 3 kinds of errors
Learning outcomes

At the end of this course, a student is expected to:

- Create (design) simple algorithms:
  - ...
  - Solve problems by designing simple algorithms, e.g., basic calculations, searching in strings and lists, counting, calculating running sums and products
- Create (design) small size programs using Python:
  - ...
  - Create programs that search or construct lists and strings
Today’s Menu

Python’s building blocks:
• Python statements
  • Categories of Statements
    4. Operational statements
      • Mathematical/arithmetic operators
        • Augmented assignment operators
      • String manipulation operators
      • Function terminology
  • Python strings
Python statements

• Categories:
  1. Assignment statement
  2. Input statement
    • Conversion function
  3. Output statement
  4. Operational statements
    • Mathematical/arithmetic operators
    • String manipulation operators
  5. Conditional statement
  6. Iterative statement

Some of them are built-in function or method
4. Operational statements

• String (sequence) manipulation operators:
  • **Syntax**: `<operand> <operator> <operand>`
  • **Operators**:
    • **Concatenation operator**: `+`
      • Both operands must be of `str` data type
    • **Repetition operator**: `*`
      • One operand must be of `str` data type
      • The other operand must be of `int` data type
    • **Format operator**: `%`
      • String formatting
  • **Examples**:
    ```
    >>> dessertToday = "ice" + "cream"
    >>> guffaw = "ha" * 3     # OR  3 * "ha"
    >>> age = 26
    >>> message = "Paul is %i years of age." %age
    >>> print(message)
    Paul is 26 years of age.
    ```
Strings

**Definition:** Sequence of characters

1. How to create a string variable?
   **Answer:** Use the name of the string variable
   • For example:

2. How to access a whole string?
   **Answer:** Use the name of the string variable
   • For example:
3. 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 1\textsuperscript{st} character of the string, we use the index 0
  • The 2\textsuperscript{nd} character of the string, we use the index 1
  • etc...
String indexing: positive indexing examples

When does the "IndexError: string index out of range" error occur?

Careful: Positive index starts at 0
String indexing: negative indexing

- There is another way we can use to access one string character at a time: negative indexing:

Example: message = "Hello, World!"

negative indexing -> index: -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1

- So if we wish to access
  - The 1st character of the string, we use the index -13
  - The last character of the string, we use the index -1,
  - etc...
String indexing: negative indexing examples

Careful: Negative index starts at -1, not 0
String slicing (using positive indices)

4. How to access a section (slice) of a string at a time?

   Answer: use indices to indicate the string slice

   \[
   \text{positive indexing} \rightarrow \text{index: } 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12
   \]

   \[
   \text{Example: } \text{message} = "Hello, World!"\]

   \[
   \text{Syntax: } \langle \text{aString} \rangle [\text{start} : \text{stop} : \text{step}]
   \]

   \[
   \begin{align*}
   &\text{• start} \\
   &\text{• stop} \\
   &\text{• step}
   \end{align*}
   \]

   \[
   \text{• Example: } \text{So if we wish to access the string slice "Hello", we use message[0:5]}
   \]
How does String Slicing works?

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

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
How does String Slicing works?

The Python Interpreter would execute the following as:

• message[4]
  • where \texttt{start} = 4, \texttt{stop} = ___ and \texttt{step} = ___

• message[4:9]
  • where \texttt{start} = 4, \texttt{stop} = 9 and \texttt{step} = ___

• message[4:9:2]
  • where \texttt{start} = 4, \texttt{stop} = 9 and \texttt{step} = 2
String Slicing - Direction

• The **direction** of slicing dictated by sign of **step**
  • If **step** is + positive -> the slicing is done from left to right
  • For example:
    ```
    message[3:11:2]
    ```
    produces the slice -> 'l,Wr'
  
  • If **step** is - negative -> the slicing is done from right to left
  • For example:
    ```
    message[-2:-9:-3]
    ```
    produces the slice -> 'do,'
Let’s try!

>>> "123456789"[2:8:3]   ->

>>> "123456789"[2:8:-3]   ->

>>> "123456789"[-2:-8:-3]   ->

>>> "123456789"[-2:-8:3]   ->

>>> "123456789"[8:2:3]   ->

>>> "123456789"[8:2:-3]   ->

>>> "123456789"[-8:-2:-3]   ->

>>> "123456789"[-8:-2:3]   ->
String slicing - Examples

```
>>> message = "Hello, World!"
>>> message[0:5]
'Hello'
>>> message[7:9]
'Wo'
>>> message[:5]
'Hello'
>>> message[-2:5]
'

>>> message[7:12]
'World'
>>> message[7:]
'World!'
>>> message[7: 25]
'World!'
```

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).
String slicing (using negative indices)

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

negative indexing -> index: -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1
Let’s try!
Strings are immutable!

• Definition of **Immutable**:

• Can we modify a string?
  • For example, let’s try:
    ```
    message[12] = "?"
    ```

• Here is what happened when we tried?
  ```
  >>> message = "Hello, World!"
  >>> message[12] = "?"
  Traceback (most recent call last):
    File "<pyshell#4>", line 1, in <module>
    message[12] = "?"
  TypeError: 'str' object does not support item assignment
  ```

• Why?
Strings are immutable!

• But we can create another string that is a modified version of the first string

• Let’s try this Python program:
  ```python
  message = "Hello, World!"
  question = message[:12] + "?"
  print("message is still: ", message)
  print("question is: ", question)
  ```

• What is the output of the above program?
## Summary: 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>len(aString)</td>
<td>Determine the number of characters in a string aString</td>
</tr>
</tbody>
</table>
Built-in functions for strings

• `len()` is an example of a **built-in function** for strings in Python
• Let’s try it on `message`

```
>>> len(message)
13
```

• Notice that the valid range of the indices for the string `message` is **0 to 12**
• So …
  • If the length of a string is `x` then we know that the valid range of the indices for that string is **0 to `x-1`**
  • If the valid range of the indices for a string is **0 to `y`** then we know that the length of that string is **`y+1`**
Other useful built-in functions

<table>
<thead>
<tr>
<th>Method name</th>
<th>Use</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximum within a string</td>
<td>max(aString)</td>
<td>Returns the maximum value among the characters in the string aString</td>
</tr>
<tr>
<td>minimum within a string</td>
<td>min(aString)</td>
<td>Returns the minimum value among the characters in the string aString</td>
</tr>
</tbody>
</table>

We can find more built-in functions at:
https://docs.python.org/3.5/library/functions.html
Methods

• Aside from **built-in functions**, Python also has **methods**, which are a specific type of functions that are related to classes
  • **Class** is a mechanism that allows us to create our own type of objects (variables)

• **Syntax for methods:**
  
  `<string> . <method>(<arguments>)`

  **dot notation**

  Arguments are not always needed. How do we know if they are? Answer: We must look at a description of the method.
Methods for strings

- Methods are called using the **dot notation**, and are applied to a string that already exist.

- Let’s try this Python program:

```python
greetings = "Hello"
newGreetings = greetings.upper()
print(newGreetings)
print(greetings)
```

The result is:
```
HELLO
Hello
```
# Some string methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Use</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>center</td>
<td><code>aString.center(w)</code></td>
<td>Returns a new string, with [the contents of] <code>aString</code> centered in a field of size <code>w</code></td>
</tr>
<tr>
<td>count</td>
<td><code>aString.count(item)</code></td>
<td>Returns the number of occurrences of <code>item</code> in <code>aString</code></td>
</tr>
<tr>
<td>ljust</td>
<td><code>aString.ljust(w)</code></td>
<td>Returns a new string, with <code>aString</code> left justified in a field of size <code>w</code></td>
</tr>
<tr>
<td>rjust</td>
<td><code>aString.rjust(w)</code></td>
<td>Returns a new string, with <code>aString</code> contents right justified in a field of size <code>w</code></td>
</tr>
<tr>
<td>upper</td>
<td><code>aString.upper()</code></td>
<td>Returns a new string, with <code>aString</code> contents all in upper cases</td>
</tr>
<tr>
<td>lower</td>
<td><code>aString.lower()</code></td>
<td>Returns a new string, with <code>aString</code> contents all in lower cases</td>
</tr>
<tr>
<td>find</td>
<td><code>aString.find(item)</code></td>
<td>Returns the index of the first occurrence of <code>item</code> in <code>aString</code></td>
</tr>
</tbody>
</table>
A few more string methods

- title()
- strip()
- rstrip()
- rfind()
- isdigit()
- isalpha()
- isnumeric()
- islower()
- isupper()
- isspace()
- capitalize()
- etc.
More information about string methods

• You can find a lot more string methods at: https://docs.python.org/3.5/library/stdtypes.html#index-31

• Even though we may not talk about all the Python string methods during our lectures, our task, as a Python software developer (i.e., CMPT 120 students), is to become familiar with Python string methods as they will be very useful to us in solving our problems/tasks throughout the semester
Summary

Python’s building blocks:

• Python statements
  • Categories of Statements
    4. Operational statements
      • Mathematical/arithmetic operators
        • Augmented assignment operators
      • String manipulation operators
      • Function terminology
  • Python strings
Next Lecture

- Python Lists