Compsci 101
List and String Operations, For loop

Susan Rodger
February 2, 2023

G is for …

- Google
  - How to find the answer to everything
- Global Variable
  - Accessible everywhere, typically do not do
- GIGO
  - Garbage In, Garbage Out
- Git
  - Working Together or Solo

Sir Tim Berners-Lee

- Invented World Wide Web
  - Turing award 2016
- HTTP vs. TCP/IP
  - Just protocols?

“The Web as I envisaged it, we have not seen it yet. The future is still so much bigger than the past.”

“We need diversity of thought in the world to face the new challenges.”

Did you sign up for compsci@duke.edu mailing list?

- Mailing list to get the CompSci weekly newsletter
  - Events, research and job opportunities
- To add yourself:
  - Go to lists.duke.edu
  - Authenticate and then add compsci@duke.edu

This Photo by Unknown Author is licensed under CC BY-SA
Announcements

• Assignment 1 Faces
  • Program due Tonight (has one grace day)
  • Also REFLECT Form due same time
  • Remember, no consulting hours on Friday
• APT-2 out today, due Feb 9
  • Some good practice for the exam
• Lab 3 Friday
  • Do prelab 3 before attending!
• Exam 1 on Tuesday, Feb 7

PFTD

• Immutable Types
• Objects and what that means
• Lists continued
• String methods and more
• For Loops
• Exam 1

Immutable built-in Types

• In python string, int, float, boolean - Immutable
  • Once created cannot change
  • These are still objects in Python3!!
• Assignment makes a copy
  • \( b = a \)
  • \( b \) gets a copy of \( a \)
• Let's look at an example
  • Example with integers

\[
\begin{align*}
\text{val} &= 0 \\
\text{bee} &= \text{val} \\
\text{val} &= \text{val} + 20
\end{align*}
\]
Immutable built-in Types

- In python string, int, float, boolean - Immutable
  - Once created cannot change
  - These are still objects in Python3!!

- Assignment makes a copy
  - b = a
  - b gets a copy of a

- Let's look at an example
  - Example with integers
    - val = 0
    - bee = val
    - val = val + 20
    - bee is a copy of val

Let's look at another example!
- With strings!
  - val = "apple"
  - bee = val
  - val = val + "sauce"

val is 20
bee is 0

val changing, doesn't affect bee

val is "apple"
Immutable built-in Types

- In python string, int, float, boolean - Immutable
  - Once created cannot change
  - These are still objects in Python3!!

- Assignment makes a copy
  - b = a
  - b gets a copy of a

- Here is another example!
  - With strings!

```
val = "apple"
bee = val
val = val + "sauce"
```

```
val is "apple"
bee is "apple"
```

```
val changing, doesn't affect bee
```

Let's see how the memory works in Python Tutor

Compare assign with integers, strings and lists
Compare assign with integers, strings and lists

<table>
<thead>
<tr>
<th>Python 3.6 (known limitations)</th>
<th>Frames</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x = 6</td>
<td>Global frame</td>
<td>x 6</td>
</tr>
<tr>
<td>2 y = x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 x = 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 m = &quot;pink&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 n = m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 m = &quot;red&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 a = [&quot;pig&quot;, &quot;cow&quot;, &quot;dog&quot;]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 b = a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 a[-1] = &quot;ant&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

y gets a copy of the value of x

x gets a new value
Compare assign with integers, strings and lists

What about lists?

What happens when a and b are list variables?

b = a

It is a copy! Of what?
Compare assign with integers, strings and lists

List Cloning (or copying)

```
lst1 = ['a', 'b', 1, 2]
lst2 = lst1
lst3 = lst1[:]
```

Changing list `a` also changes list `b` As they are the same list!

‘dog’ changed to ‘ant’

a’s value is the address of its list, the address is copied!

b gets a copy of the value of a

a and b refer to the same list!
List Cloning (or copying)

```python
lst1 = ['a', 'b', 1, 2]
lst2 = lst1
lst3 = lst1[:]
```

List Cloning (or copying)

```python
lst1 = ['a', 'b', 1, 2]
lst2 = lst1
lst3 = lst1[:]
```

```python
lst1[-1] = "SUN"
```

List Cloning (or copying)

```python
lst1 = ['a', 'b', 1, 2]
lst2 = lst1
lst3 = lst1[:]
```

```python
lst1[-1] = "SUN"
```
List Concatenation Steps

1. Calculate the **length** of the new list
2. **Create** list of that length
3. **Copy** values from first list
4. **Copy** values from second list
5. **Assign the variable to the new list**

```
1   lst0 = [1,2]
2   lst1 = [3, 4, 5]
3   lst2 = lst0 + lst1
```
Concatenation: Makes new List

1. Calculated length
2. Create new list
3. Copy left list
4. Copy right list
5. Assign list2

What will Python Tutor Display? How many lists will there be?

Concatenation: Makes new List

1. $\text{lst0} = [1, 2]$
2. $\text{tmp} = \text{lst0}$
3. $\text{lst0} = \text{lst0} + [4]$
**Concatenation: Makes new List**

1. `lst0 = [1, 2]`
2. `tmp = lst0`
3. `lst0 = lst0 + [4]`

**Concatenation: length, create, copy, copy, assign**

4. Copy right list
5. Assign `lst0`

**How is the inner list copied?**

- What will Python Tutor Display? How many copies of `[b, 3.0]` will be present?

---

**Concatenation: length, create, copy, assign**

1. Calculate length
2. Create new list
3. Copy left list

**Concatenation: length, create, copy, copy, assign**

4. Copy right list
5. Assign `lst2`

**How is the inner list copied?**

- This is a shallow copy! Don’t copy inner lists
List Mutation: `.append(...)`

- `.append()` – list function that adds element to end of list
  - Mutates list to left of “.”
  - “.” – call function to the right of the dot on the thing to the left of the dot (LEFT . RIGHT)

```python
x = [6, 2, 4]
x.append(3)
x.append([5,2])
```

2/2/23 Compsci 101, Spring 2023 45

```
x is [6, 2, 4]
x is [6, 2, 4, 3]
x is [6, 2, 4, 3, [5, 2]]
```

Same list!

List Mutation: `.append(...)`

- `.append()` – list function that adds element to end of list
  - Mutates list to left of “.”
  - “.” – call function to the right of the dot on the thing to the left of the dot (LEFT . RIGHT)

```
x = [6, 2, 4]
x.append(3)
x.append([5,2])
```

2/2/23 Compsci 101, Spring 2023 46

```
x is [6, 2, 4]
x is [6, 2, 4, 3]
x is [6, 2, 4, 3, [5, 2]]
```

List Mutation: `.append(...)`

```
lst0 = [1, 2, 3]
tmp = lst0
lst0.append(4)
```

What will Python Tutor Display? One or two lists?

```
lst0 = [1, 2, 3]
tmp = lst0
lst0.append(4)
```

```
frame
lst0
```

list
0 1 2 3
List Mutation: `.append(...)`

1. `lst0 = [1, 2, 3]`
2. `tmp = lst0`
3. `lst0.append(4)`

Same list! No new list

1. `lst0 = [1, 2, 3]`
2. `tmp = lst0`
3. `lst0.append(4)`

List Mutation: `.append(...)`

`lst0 = [1, 2, 3]`
`tmp = lst0`
`lst0.append(4)`
`lst0.append([5, 6])`

Same list! No new list

1. `lst0 = [1, 2, 3]`
2. `tmp = lst0`
3. `lst0.append(4)`
4. `lst0.append([5, 6])`
Anatomy of a for loop

```
for VARIABLE in SEQUENCE:
    CODE_BLOCK
```

- Think of as:
  - “For each element in the SEQUENCE put it in the VARIABLE and execute the CODE_BLOCK.”
- Also called: Iterate over the sequence
- What type(s) are sequences?
  - Strings, Lists
- Will VARIABLE likely be in CODE_BLOCK?

Example for loop with a list

```
1  lst = [5, 3, 2]
2  sum = 0
3  for num in lst:
4      sum = sum + num
5  print(sum)
```

- What does this for loop do?
- What is first value of num?
- What is final value of num?
Example for loop with a list

• What does this for loop do?

```python
lst = [5, 3, 2]
sum = 0
for num in lst:
    sum = sum + num
print(sum)
```

• What is first value of `num`?
  5

• What is final value of `num`?
  2

Trace through for loop

```
lst = [5, 3, 2]
sum = 0
for num in lst:
    sum = sum + num
print(sum)
```

Frames

Global frame

Objects

```
lst
0 1 2
list
5 3 2
```

Global frame

```
list
0 1 2
lst
5 3 2
sum
0
```
Trace through for loop

```
1 lst = [5, 3, 2]
2 sum = 0
3 for num in lst:
   4   sum = sum + num
5 print(sum)
```

Frames

Objects

num gets first value in list

```
num

Global frame
lst
sum
num
```

```
list
0 1 2
5 3 2
```

Add num to sum

```
num

Global frame
lst
sum
num
```

```
list
0 1 2
5 3 2
```

Trace through for loop

```
1 lst = [5, 3, 2]
2 sum = 0
3 for num in lst:
   4   sum = sum + num
5 print(sum)
```

Frames

Objects

num gets second value in list

```
num

Global frame
lst
sum
num
```

```
list
0 1 2
5 3 2
```

Add num to sum

```
num

Global frame
lst
sum
num
```

```
list
0 1 2
5 3 2
```
Trace through for loop

1. `lst = [5, 3, 2]`
2. `sum = 0`
3. `for num in lst:
   sum = sum + num`
4. `print(sum)`

Frames
- Global frame
  - `lst`:
    - 0: 5
    - 1: 3
    - 2: 2

Objects
- List:
  - 0: 5
  - 1: 3
  - 2: 2

Trace through for loop

1. `lst = [5, 3, 2]`
2. `sum = 0`
3. `for num in lst:
   sum = sum + num`
4. `print(sum)`

Frames
- Global frame
  - `lst`:
    - 0: 5
    - 1: 3
    - 2: 2

Objects
- `sum`:
  - 0: 10

Trace through for loop

1. `lst = [5, 3, 2]`
2. `sum = 0`
3. `for num in lst:
   sum = sum + num`
4. `print(sum)`

Frames
- Global frame
  - `lst`:
    - 0: 5
    - 1: 3
    - 2: 2

Trace through for loop

1. `lst = [5, 3, 2]`
2. `sum = 0`
3. `for num in lst:
   sum = sum + num`
4. `print(sum)`

Frames
- Global frame
  - `lst`:
    - 0: 5
    - 1: 3
    - 2: 2

Objects
- `sum`:
  - 0: 10

Print result

No more values in lst
The for loop is done!
Example for loop with a string

- What does this for loop do?

```python
1 animal = 'cat'
2 word = animal
3 for ch in animal:
   4   word = word + ch
5 print(word)
```

- What is first value of `ch`?

- What is final value of `ch`?

Example for loop with a string

- What does this for loop do?

```python
1 animal = 'cat'
2 word = animal
3 for ch in animal:
   4   word = word + ch
5 print(word)
```

- What is first value of `ch`?
  
  - ‘c’

- What is final value of `ch`?
  
  - ‘t’

Trace through for loop

```python
1 animal = 'cat'
2 word = animal
3 for ch in animal:
   4   word = word + ch
5 print(word)
```

Trace through for loop

```python
1 animal = 'cat'
2 word = animal
3 for ch in animal:
   4   word = word + ch
5 print(word)
```

Global frame

- `animal` = "cat"
Trace through for loop

```python
1 animal = 'cat'
2 word = animal
3 for ch in animal:
4     word = word + ch
5 print(word)
```

Global frame:
- animal: "cat"
- word: "cat"

Trace through for loop

Iterate over copy of word: ‘c’ ‘a’ ‘t’

Add ch to end of word

Global frame:
- animal: "cat"
- word: "catc"
- ch: "c"

Trace through for loop

Iterate over what is left in copy of word: ‘a’ ‘t’

ch gets first character in word

Global frame:
- animal: "cat"
- word: "catc"
- ch: "c"

Trace through for loop

Iterate over what is left in copy of word: ‘a’ ‘t’

ch gets second character in word

Global frame:
- animal: "cat"
- word: "catc"
- ch: "a"
Trace through for loop

1. `animal = 'cat'`
2. `word = animal`
3. `for ch in animal:`
   - `word = word + ch`
4. `print(word)`

Add ch to end of word

Global frame:

- `animal`: "cat"
- `word`: "catca"
- `ch`: "a"

Trace through for loop

1. `animal = 'cat'`
2. `word = animal`
3. `for ch in animal:`
   - `word = word + ch`
4. `print(word)`

Add ch to end of word

Global frame:

- `animal`: "cat"
- `word`: "catca"
- `ch`: "a"

Trace through for loop

1. `animal = 'cat'`
2. `word = animal`
3. `for ch in animal:`
   - `word = word + ch`
4. `print(word)`

Add ch to end of word

Global frame:

- `animal`: "cat"
- `word`: "catcat"
- `ch`: "t"

Trace through for loop

1. `animal = 'cat'`
2. `word = animal`
3. `for ch in animal:`
   - `word = word + ch`
4. `print(word)`

No more characters in word to process

Global frame:

- `animal`: "cat"
- `word`: "catcat"
- `ch`: "t"

The for loop is done!
**Trace through for loop**

```
1 animal = 'cat'
2 word = animal
3 for ch in animal:
   4   word = word + ch
5 print(word)
```

**Print output**

```
catcat
```

**Global frame**

- `animal` -> "cat"
- `word` -> "catcat"
- `ch` -> "t"

**String’s split(…)**

- Strings have functions too!
- **TYPE_STRING.FUNCTION(PARAMETERS)**
  - "." means apply function to what is on the left
  - 'one fish two fish'.split() returns a list

  ```
  ['one', 'fish', 'two', 'fish']
  ```

- What did it divide the string by?
  - When no parameter, default whitespace
  - 'one fish, two fish'.split(',')

  ```
  ['one fish', ' two fish']
  ```

**String’s join(…)**

- **TYPE_STRING.join(SEQ_OF_STRINGS)**
  - Opposite of .split()
  - Creates string from sequence’s items separated by the string to the left of join
  - ' '.join(['one', 'fish', 'two', 'fish'])
  - '+'.join(['one', 'fish', 'two', 'fish'])
  - 'ish'.join(['f', 'w', 'd', 'end'])

**String’s split(…)**

- Strings have functions too!
- **TYPE_STRING.FUNCTION(PARAMETERS)**
  - "." means apply function to what is on the left
  - 'one fish two fish'.split() returns a list

  ```
  ['one', 'fish', 'two', 'fish']
  ```

- What did it divide the string by?
  - When no parameter, default whitespace
  - 'one fish, two fish'.split(',')

  ```
  ['one fish', ' two fish']
  ```
String’s join(...)  

- `TYPE_STRING.join(SEQ_OF_STRINGS)`  
  - Opposite of .split()  
  - Creates string from sequence’s items separated by the string to the left of join

  - `' .join(['one','fish','two','fish'])`  
    - `one fish two fish`  
  - `' +.join(['one','fish','two','fish'])`  
    - `one+fish+two+fish`  
  - `'ish'.join(['f','w','d','end'])`  
    - `fishwishdishend`

---

More Methods

**String**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>find(s)</code></td>
<td>index of first occurrence of s</td>
</tr>
<tr>
<td><code>rfind(s)</code></td>
<td>index of last occurrence of s (from Right)</td>
</tr>
<tr>
<td><code>upper()</code></td>
<td>uppercase/lowercase version of string</td>
</tr>
<tr>
<td><code>lower()</code></td>
<td></td>
</tr>
<tr>
<td><code>strip()</code></td>
<td>remove leading/trailing whitespace</td>
</tr>
<tr>
<td><code>count(s)</code></td>
<td>number of times see s in string</td>
</tr>
<tr>
<td><code>startswith(s)</code></td>
<td>bool of whether the string begins with s</td>
</tr>
<tr>
<td><code>endswith(s)</code></td>
<td>bool of whether the string ends with s</td>
</tr>
</tbody>
</table>

**List**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sum(lst)</code></td>
<td>sum of the elements in lst</td>
</tr>
<tr>
<td><code>max(lst)</code></td>
<td>maximum value of lst</td>
</tr>
<tr>
<td><code>min(lst)</code></td>
<td>minimum value of lst</td>
</tr>
<tr>
<td><code>append(elm)</code></td>
<td>Mutates the list by adding elm to the end of the list</td>
</tr>
<tr>
<td><code>count(elm)</code></td>
<td>Number of times see elm in the list</td>
</tr>
</tbody>
</table>

---

WOTO-3 – Split and Join


---

APT2 out today – Due Feb 9

Do early - practice for exam

- 5 problems
  - Write code on paper first  - good practice!
  - Then type in and debug

---

One of these uses a loop

- ReadQuizScore
- RemoveMiddle
- PortManteau
- TotalWeight
- SentenceLength
Exam 1 – Feb 7, 2023

- All lecture/reading topics through today
  - Topics today at simpler level
    - Loop over list, loop over characters in a string
- Understand/Study
  - Reading, lectures
  - Assignment 1, APT-1, (APT-2 helpful, not required)
  - Labs 0-3
  - Very Important! Practice writing code on paper
- Logistics:
  - Exam in person, in lecture

Exam 1 – Feb 7, 2023 (cont)

- What you should be able to do
  - Read/trace code
  - Determine output of code segment
  - Write small code segments/function
- Look at old test questions
  - We will look at some in Lab 3
- Exam 1 is your own work!
  - Only bring a pen or a pencil!
  - Do not consult with anyone else.
  - Closed book, no notes, no paper, no calculators
  - See Exam 1 Reference sheet (will be on exam)

Python Reference Sheet, is attached to your exam
(see link on calendar page, under 2/7)