Compsci 101
List and String Operations, For loop

Susan Rodger
September 20, 2022
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

• Sample item:
  • Duke Women in Tech looking for new members and to get our mailing lists. Fill out this form: https://tinyurl.com/witspring22
Announcements

• **Assignment 1 Faces**
  • Assign 1 QZ due today 11:30pm (no grace day!)
  • Program due Thursday (has one grace day)
  • Also REFLECT Form due same time
  • Remember, no consulting hours on Friday

• **APT-2 out today, due Sept 29**

• **Lab 3 Friday**
  • Do prelab 3 before attending!

• **Exam 1 on Tuesday, Sept 27**
Lecture for Thursday Sept 22

• Lecture will be asynchronous
  • There is no in-person lecture
• Watch the videos (there are 4), do WOTOs
• Complete it before going to lab
• WOTOs will turn off by Friday night 11:30pm

• Prof. Rodger is out of town Tues-Friday, no office hours today or the rest of the week
• We have fantastic UTAs in consulting hours! And there is always Ed Discussion!
PFTD

• Exam 1
• Lists continued
• String methods and more
• For Loops
Exam 1 – Sept 27, 2022

• All lecture/reading topics through Tues. Sept 20
  • Topics on Sept 20 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:
  • In person, in lecture
Exam 1 – Sept 27, 2022 (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!**
  • 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 for Compsci 101, Exam 1, Fall 2022

On this page we'll keep track of the Python types, functions, and operators that we've covered in class. You can also review the online Python References for more complete coverage. BUT NOTE there is way more python in the there then we will cover! The reference page below is all you should need to complete the exam.

### Mathematical Operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>4 + 5 = 9</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>9 - 5 = 4</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>3*5 = 15</td>
</tr>
<tr>
<td>/ and //</td>
<td>division</td>
<td>6/3 = 2.0 6/4 = 1.5 6/4 = 1</td>
</tr>
<tr>
<td>%</td>
<td>mod/remainder</td>
<td>5 % 3 = 2</td>
</tr>
<tr>
<td>**</td>
<td>exponentiation</td>
<td>3<strong>2 = 9, 2</strong>3 = 8</td>
</tr>
</tbody>
</table>

### String Operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>concatenation</td>
<td>&quot;ab&quot;+&quot;cd&quot;=&quot;abcd&quot;</td>
</tr>
<tr>
<td>*</td>
<td>repeat</td>
<td>&quot;xo&quot;*3 = &quot;xoxoxo&quot;</td>
</tr>
</tbody>
</table>

### Comparison Operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>is equal to</td>
<td>3 == 3 is True</td>
</tr>
<tr>
<td>!=</td>
<td>is not equal to</td>
<td>3 != 3 is False</td>
</tr>
<tr>
<td>&gt;=</td>
<td>is greater than or equal to</td>
<td>4 &gt;= 3 is True</td>
</tr>
<tr>
<td>&lt;=</td>
<td>is less than or equal to</td>
<td>4 &lt;= 3 is False</td>
</tr>
<tr>
<td>&gt;</td>
<td>is strictly greater than</td>
<td>4 &gt; 3 is True</td>
</tr>
<tr>
<td>&lt;</td>
<td>is strictly less than</td>
<td>3 &lt; 3 is False</td>
</tr>
</tbody>
</table>

### Boolean Operators
Nested Lists

- Lists are heterogenous, therefore!
  - \( \text{lst} = [1, 'a', [2, 'b']] \) is valid
  - \( \text{len(lst)} == 3 \)
    - \([2, 'b']\) is one element in list \( \text{lst} \)

- How to index?
  - [...] all the way down
  - \( \text{lst}[2][1] \) returns ‘b’
Nested Lists with Python Tutor

```
1  lst = [1, 'a', [2, 'b']]
2  print(len(lst))
3  print(type(lst[2]))
4  print(lst[2])
5  print(lst[2][1])
```
Nested Lists with Python Tutor

```
1 lst= [1, 'a', [2, 'b']]
2 print(len(lst))
3 print(type(lst[2]))
4 print(lst[2])
5 print(lst[2][1])
```

OUTPUT:

3
<class 'list'>
[2, 'b']
b
Mutating Lists

• \(lt = ['Hello', 'world']\)
  • Change to: ['Hello', 'Ashley']

• Two ways: 1. Build new list or 2. modify list
  1. Concatenation: \(lt = [lt[0]] + ['Ashley']\)
  2. Index: \(lt[1] = 'Ashley'\)

• How change ‘b’ in \(lt = [1, 'a', [2, 'b']]\)?
  • \(lt[2][1] = 'c'\)
Mutating Lists code

```python
lst1 = ['Hello', 'world']
print(lst1)
lst2 = [lst1[0]] + ['Ashley']
print(lst2)
print(lst1)
lst1[1] = 'Ashley'
print(lst1)

lst3 = [1, 'a', [2,'b']] 
print(lst3)
lst3[2][1] = 'c'
print(lst3)
```
Mutating Lists code

```python
lst1 = ['Hello', 'world']
print(lst1)
lst2 = [lst1[0]] + ['Ashley']
print(lst2)
print(lst1)
lst1[1] = 'Ashley'
print(lst1)

lst3 = [1, 'a', [2,'b']]  
print(lst3)
lst3[2][1] = 'c'
print(lst3)
```

OUTPUT:

```
['Hello', 'world']
[Hello, Ashley]
['Hello', 'world']
```

```
[1, 'a', [2, 'c']]
```
Mutating Lists code

```python
lst1 = ['Hello', 'world']
print(lst1)
lst2 = [lst1[0]] + ['Ashley']
print(lst2)
print(lst1)
lst1[1] = 'Ashley'
print(lst1)

lst3 = [1, 'a', [2,'b']]  
print(lst3)
lst3[2][1] = 'c'
print(lst3)
```
Mutating Lists code

```python
lst1 = ['Hello', 'world']
print(lst1)
lst2 = [lst1[0]] + ['Ashley']
print(lst2)
print(lst1)
lst1[1] = 'Ashley'
print(lst1)

lst3 = [1, 'a', [2,'b']]  
print(lst3)
lst3[2][1] = 'c'
print(lst3)
```

OUTPUT:

```
['Hello', 'world']
['Hello', 'Ashley']
['Hello', 'world']
```

![Frames and Objects diagram]
Mutating Lists code

```python
lst1 = ['Hello', 'world']
print(lst1)
lst2 = [lst1[0]] + ['Ashley']
print(lst2)
print(lst1)
lst1[1] = 'Ashley'
print(lst1)

lst3 = [1, 'a', [2,'b']]  # Tuples are not mutable
print(lst3)
lst3[2][1] = 'c'
print(lst3)
```

**OUTPUT:**

```
['Hello', 'world']
['Hello', 'Ashley']
['Hello', 'world']
['Hello', 'Ashley']
```

```
[(1, ('a', [2, 'b']))]
```
Mutating Lists code

```python
lst1 = ['Hello', 'world']
print(lst1)
lst2 = [lst1[0]] + ['Ashley']
print(lst2)
print(lst1)
lst1[1] = 'Ashley'
print(lst1)

lst3 = [1, 'a', [2,'b']]
print(lst3)
print(lst3)
lst3[2][1] = 'c'
print(lst3)
```

OUTPUT:

```
[1, 'a', [2, 'b']]
```
Mutating Lists code

```python
lst1 = ['Hello', 'world']
print(lst1)
lst2 = [lst1[0]] + ['Ashley']
print(lst2)
print(lst1)
lst1[1] = 'Ashley'
print(lst1)

lst3 = [1, 'a', [2, 'b']]
print(lst3)
lst3[2][1] = 'c'
print(lst3)
```

OUTPUT:

```
[1, 'a', [2, 'b']]  
[1, 'a', [2, 'c']]  
```

9/20/22  
CompSci 101, Fall 2022
Immutable built-in Types

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

• PythonTutor gets this wrong
  • Everything should be in Objects area

• Objects don’t change
  • Value associated with variable changes

```python
val = 0
bee = val
val = val + 20
```
Immutable built-in Types

- In python string, int, float, boolean - Immutable
  - Once created cannot change
  - These are still objects in Python3!!
- PythonTutor gets this wrong
  - Everything should be in Objects area
- Objects don’t change
  - Value associated with variable changes

```python
val = 0
bee = val
val = val + 20
val is 0
```
Immutable built-in Types

• In python string, int, float, boolean - Immutable
  • Once created cannot change
  • These are still objects in Python3!!
• PythonTutor gets this wrong
  • Everything should be in Objects area
• Objects don’t change
  • Value associated with variable changes
Immutable built-in Types

- In python string, int, float, boolean - Immutable
  - Once created cannot change
  - These are still objects in Python3!!
- PythonTutor gets this wrong
  - Everything should be in Objects area
- Objects don’t change
  - Value associated with variable changes

```
val = 0
bee = val
val = val + 20
```

```
val is 20
bee is 0
```
Immutable built-in Types

- In python string, int, float, boolean - Immutable
  - Once created cannot change
  - These are still objects in Python3!!
- PythonTutor gets this wrong
  - Everything should be in Objects area
- Objects don’t change
  - Value associated with variable changes

```
val = "apple"
bee = val
val = val + "sauce"
```
Immutable built-in Types

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

• PythonTutor gets this wrong
  • Everything should be in Objects area

• Objects don’t change
  • Value associated with variable changes

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

```python
val is "apple"
```
Immutable built-in Types

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

• **PythonTutor gets this wrong**
  • Everything should be in Objects area

• **Objects don’t change**
  • Value associated with variable changes

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

val is "apple"
bee is "apple"
```
Immutable built-in Types

- In python string, int, float, boolean - Immutable
  - Once created cannot change
  - These are still objects in Python3!!
- PythonTutor gets this wrong
  - Everything should be in Objects area
- Objects don’t change
  - Value associated with variable changes

```python
val = "apple"
bee = val
val = val + "sauce"
val is "applesauce"
bee is "apple"
```
Compare assign with integers, strings and lists – 1

### Python 3.6 (known limitations)

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

**Edit this code**

- ➡️ line that just executed
- ➡️ next line to execute
Compare assign with integers, strings and lists – 2

Python 3.6 (known limitations)

```
1  x = 6
2  y = x
3  x = 3
4  m = "pink"
5  n = m
6  m = "red"
7  a = ["pig", "cow", "dog"]
8  b = a
9  a[-1] = "ant"
```

Frames

Objects

Global frame

| x | 6 |

Edit this code

- line that just executed
- next line to execute
Compare assign with integers, strings and lists – 3

Python 3.6 (known limitations)

```
1 x = 6
2 y = x
3 x = 3
4 m = "pink"
5 n = m
6 m = "red"
7 a = ["pig", "cow", "dog"]
8 b = a
9 a[-1] = "ant"
```

Frames

- Global frame
  - x: 6
  - y: 6

Objects

y gets a copy of the value of x
Compare assign with integers, strings and lists – 4

```
Python 3.6
(known limitations)

1 x = 6
2 y = x
3 x = 3
4 # m = "pink"
5 n = m
6 m = "red"
7 a = ["pig", "cow", "dog"]
8 b = a
9 a[-1] = "ant"
```

Frames
```
Global frame
x 3
y 6
```

x gets a new value

Edit this code

- ➡️ line that just executed
- ➡️ next line to execute
Compare assign with integers, strings and lists – 5

Python 3.6 (known limitations)

```
1  x = 6
2  y = x
3  x = 3
4  m = "pink"
5  n = m
6  m = "red"
7  a = ["pig", "cow", "dog"]
8  b = a
9  a[-1] = "ant"
```

Frames

```
Global frame
x 3
y 6
m "pink"
```

Objects

Edit this code

- line that just executed
- next line to execute
Compare assign with integers, strings and lists – 6

```python
Python 3.6
(known limitations)

1 x = 6
2 y = x
3 x = 3
4 m = "pink"
5 n = m
6 m = "red"
7 a = ["pig", "cow", "dog"]
8 b = a
9 a[-1] = "ant"

Frames

Global frame
x 3
y 6
m "pink"
n "pink"

Objects

n gets a copy of the value of m
```

Edit this code

- line that just executed
- next line to execute
Compare assign with integers, strings and lists – 7

```python
# Python 3.6 (known limitations)
1 x = 6
2 y = x
3 x = 3
4 m = "pink"
5 n = m
6 m = "red"
7 a = ["pig", "cow", "dog"]
8 b = a
9 a[-1] = "ant"

Frames | Objects
-------|---------
Global frame
| x | 3 |
| y | 6 |
| m | "red" |
| n | "pink" |

m gets a new value

Edit this code
- line that just executed
- next line to execute
```
Compare assign with integers, strings and lists – 8

```
1 x = 6
2 y = x
3 x = 3
4 m = "pink"
5 n = m
6 m = "red"
7 a = ["pig", "cow", "dog"]
8 b = a
9 a[-1] = "ant"
```

Edit this code

- line that just executed
- next line to execute
Compare assign with integers, strings and lists – 9

Python 3.6 (known limitations)

```
1  x = 6
2  y = x
3  x = 3
4  m = "pink"
5  n = m
6  m = "red"
7  a = ["pig", "cow", "dog"]
8  b = a
9  a[-1] = "ant"
```

Frames

```
Global frame
  x 3
  y 6
  m "red"
  n "pink"
  a
  b
```

Objects

```
list
  0 "pig"
  1 "cow"
  2 "dog"
```

- b gets a copy of the value of a
- a’s value is the address of its list, the address is copied!
- a and b refer to the same list!
Compare assign with integers, strings and lists - 10

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x = 6</td>
</tr>
<tr>
<td>2</td>
<td>y = x</td>
</tr>
<tr>
<td>3</td>
<td>x = 3</td>
</tr>
<tr>
<td>4</td>
<td>m = &quot;pink&quot;</td>
</tr>
<tr>
<td>5</td>
<td>n = m</td>
</tr>
<tr>
<td>6</td>
<td>m = &quot;red&quot;</td>
</tr>
<tr>
<td>7</td>
<td>a = [&quot;pig&quot;, &quot;cow&quot;, &quot;dog&quot;]</td>
</tr>
<tr>
<td>8</td>
<td>b = a</td>
</tr>
<tr>
<td>9</td>
<td>a[-1] = &quot;ant&quot;</td>
</tr>
</tbody>
</table>

**Edit this code**

- line that just executed
- next line to execute

**Frames**

Global frame

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>6</td>
</tr>
<tr>
<td>m</td>
<td>&quot;red&quot;</td>
</tr>
<tr>
<td>n</td>
<td>&quot;pink&quot;</td>
</tr>
<tr>
<td>a</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

**Objects**

- list
  - 0: "pig"
  - 1: "cow"
  - 2: "ant"

‘dog’ changed to ‘ant’

Changing list a also changes list b As they are 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[:]
```

**Frames**

**Objects**

- Global frame
  - `lst1`

- `list`
  - 0: "a"
  - 1: "b"
  - 2: 1
  - 3: 2
List Cloning (or copying)

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

![Diagram of Global frame and list representation]
List Cloning (or copying)

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

Frames

Global frame

lst1
lst2
lst3

Objects

<table>
<thead>
<tr>
<th>list</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;a&quot;</td>
<td>&quot;b&quot;</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>list</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;a&quot;</td>
<td>&quot;b&quot;</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
List Cloning (or copying)

```python
lst1 = ['a', 'b', 1, 2]
lst2 = lst1
lst3 = lst1[:]
lst1[-1] = "SUN"
```
WOTO-1 Cloning
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:
length, create, copy, copy, assign

1  lst0 = [1,2]
2  lst1 = [3, 4, 5]
3  lst2 = lst0 + lst1
Concatenation:
length, create, copy, copy, assign

1. Calculate length
2. Create new list
3. Copy left list
Concatenation:
length, create, copy, copy, assign

1. \( \text{lst0} = [1, 2] \)
2. \( \text{lst1} = [3, 4, 5] \)
3. \( \text{lst2} = \text{lst0} + \text{lst1} \)

4. Copy right list

5. Assign lst2
Concatenation: Makes new List

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

What will Python Tutor Display? How many lists will there be?
Concatenation: Makes new List

```
1  lst0 = [1,2]
2  tmp = lst0
3  lst0 = lst0 + [4]
```
1. Calculate length
2. Create new list
3. Copy left list

lst0 = [1, 2]
tmp = lst0
lst0 = lst0 + [4]
Concatenation: Makes new List

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

4. Copy right list
5. Assign lst0
Concatenation:
length, create, copy, copy, assign

• How is the inner list copied?

```python
lst0 = [1, ['b', 3.0]]
lst1 = [4]
lst2 = lst0 + lst1
```

What will Python Tutor Display? How many copies of ['b', 3.0] will be present?
Concatenation:
length, create, copy, copy, assign

• How is the inner list copied?

```python
1 lst0 = [1, ['b', 3.0]]
2 lst1 = [4]
3 lst2 = lst0 + lst1
```

1. Calculate length
2. Create new list
3. Copy left list
Concatenation:
length, create, copy, copy

• How is the inner list copied?

```python
1 lst0 = [1, ['b', 3.0]]
2 lst1 = [4]
3 lst2 = lst0 + lst1
```

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)

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

- `x` is `[6, 2, 4]`
- `x` is `[6, 2, 4, 3]`
- `x` is `[6, 2, 4, 3, [5, 2]]`
List Mutation: .append(...) 

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

What will Python Tutor Display? One or two lists?
List Mutation: .append(...)
List Mutation: `.append(...)`

```
1   lst0 = [1, 2, 3]
2   tmp = lst0
3   lst0.append(4)
```
List Mutation: `.append(…)`

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

Same list! No new list
List Mutation: .append(...)
List Mutation: `.append(…)`

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

Same list! No new list
WOTO-2 – Mutable and Append

Anatomy of a for loop

```python
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?
Anatomy of a for loop

```python
for v in seq:
    CODE_BLOCK
```

Start

More elements in seq?

Yes

\[ v = \text{next element in seq} \]

CODE_BLOCK

No

Finished
Example for loop with a list

• What does this for loop do?

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

• What is first value of `num`?

• What is final value of `num`?
Example for loop with a list

• What does this for loop do?

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

• What is first value of `num`?

5

• What is final value of `num`?

2

Adds the numbers in the list
Trace through for loop – 1

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

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

```python
lst = [5, 3, 2]
sum = 0
for num in lst:
    sum = sum + num
print(sum)
```
Trace through for loop – 4

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

num gets first value in list
Trace through for loop – 5

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

Add num to sum
Trace through for loop – 6

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

num gets second value in list
Trace through for loop – 7

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

Add num to sum
Trace through for loop – 8

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

- num gets third value in list
- Global frame:
  - lst: 2
  - sum: 8
  - num: 2
- Objects:
  - list:
    - 0: 5
    - 1: 3
    - 2: 2
Trace through for loop – 9

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

Add num to sum

Frames

Objects

Global frame

<table>
<thead>
<tr>
<th>lst</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>10</td>
</tr>
<tr>
<td>num</td>
<td>2</td>
</tr>
</tbody>
</table>

list

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
Trace through for loop – 10

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

No more values in lst

The for loop is done!
Trace through for loop – 11

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

Print output (drag lower right corner to resize)

```
10
```

Frames

- Global frame
  - lst
  - sum: 10
  - num: 2

Objects

- list
  - 0: 5
  - 1: 3
  - 2: 2
Example for loop with a string

• What does this for loop do?

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

• What is first value of `ch`?
  ‘c’

• What is final value of `ch`?
  ‘t’
Trace through for loop – 1

1. `word = 'cat'
2. `for ch in word:
3.     word = word + ch
4. print(word)`
Trace through for loop – 2

```
1  word = 'cat'
2  for ch in word:
3      word = word + ch
4  print(word)
```
Trace through for loop – 3

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

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

Frames

Global frame

<table>
<thead>
<tr>
<th>word</th>
<th>&quot;cat&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch</td>
<td>&quot;c&quot;</td>
</tr>
</tbody>
</table>
Trace through for loop – 4

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

Frames

Global frame

<table>
<thead>
<tr>
<th>word</th>
<th>&quot;catc&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch</td>
<td>&quot;c&quot;</td>
</tr>
</tbody>
</table>

Add ch to end of word
Trace through for loop – 5

1. `word = 'cat'`
2. `for ch in word:`
   - `ch` gets second character in `word`
   - `word = word + ch`
3. `print(word)`
4. Iterate over what is left in copy of `word`: ‘a’ ‘t’

Frames:
- Global frame
  - `word`:
    - "catc"
  - `ch`:
    - "a"
Trace through for loop – 6

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

Add ch to end of word

Frames

Global frame

<table>
<thead>
<tr>
<th>word</th>
<th>&quot;catca&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch</td>
<td>&quot;a&quot;</td>
</tr>
</tbody>
</table>
Trace through for loop – 7

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

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

Frames

<table>
<thead>
<tr>
<th>Global frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
</tr>
<tr>
<td>ch</td>
</tr>
</tbody>
</table>

ch gets third character in word
Trace through for loop – 8

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

Frames

Global frame

| word  | "catcat" |
| ch    | "t"      |

Add ch to end of word
Trace through for loop – 9

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

The for loop is done!

No more characters in word to process

Frames

<table>
<thead>
<tr>
<th>Global frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
</tr>
<tr>
<td>ch</td>
</tr>
</tbody>
</table>
Trace through for loop – 10

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

Print output (drag lower right corner to resize)

```
catcat
```

Frames

<table>
<thead>
<tr>
<th>Global frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
</tr>
<tr>
<td>ch</td>
</tr>
</tbody>
</table>
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

• What did it divide the string by?
  • When no parameter, default whitespace
  'one fish, two fish'.split(',,')
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
    ```python
    ['one', 'fish', 'two', 'fish']
    ```
  - What did it divide the string by?
    - When no parameter, default whitespace
  - `'one fish, two fish'.split(',',)`
    ```python
    ['one fish', ' two fish']
    ```
String’s join(...) 

- \textsc{type}\_\textsc{string}.join(\textsc{seq}\_\text{of}\_\text{strings})
  - Opposite of .split()
  - Creates string from sequence’s items separated by the string to the left of join

\begin{align*}
  &\text{'} '.join(['\text{one}', '\text{fish}', '\text{two}', '\text{fish}']) \\
  &\text{'+}'.join(['\text{one}', '\text{fish}', '\text{two}', '\text{fish}']) \\
  &\text{'ish'}.join(['\text{f}', '\text{w}', '\text{d}', '\text{end}'])
\end{align*}
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 <code>s</code></td>
</tr>
<tr>
<td><code>.rfind(s)</code></td>
<td>index of last occurrence of <code>s</code> (from Right)</td>
</tr>
<tr>
<td><code>.upper()</code>/<code>.lower()</code></td>
<td>uppercase/lowercase version of string</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 <code>s</code> in string</td>
</tr>
<tr>
<td><code>.startswith(s)</code></td>
<td>bool of whether the string begins with <code>s</code></td>
</tr>
<tr>
<td><code>.endswith(s)</code></td>
<td>bool of whether the string ends with <code>s</code></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 <code>lst</code></td>
</tr>
<tr>
<td><code>max(lst)</code></td>
<td>maximum value of <code>lst</code></td>
</tr>
<tr>
<td><code>min(lst)</code></td>
<td>minimum value of <code>lst</code></td>
</tr>
<tr>
<td><code>.append(elm)</code></td>
<td>Mutates the list by adding <code>elm</code> to the end of the list</td>
</tr>
<tr>
<td><code>.count(elm)</code></td>
<td>Number of times see <code>elm</code> in the list</td>
</tr>
</tbody>
</table>
WOTO-3 – Split and Join