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
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

```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!!
- Assignment makes a copy
  - \( b = a \)
  - \( b \) gets a copy of \( a \)
- Let's look at an example
  - Example with integers

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

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

```python
val is 0
```
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
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
    - \[ val = 0 \]
      - \[ bee = val \]
      - \[ val = val + 20 \]
      - \[ val \text{ is } 20 \]
      - \[ bee \text{ is } 0 \]

\[ \text{val changing, doesn't affect bee} \]
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"
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"
```
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"
```

bee is a copy of val

```
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!!
- Assignment makes a copy
  - b = a
  - b gets a copy of a
- Here is another example!
  - With strings!

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

val is "applesauce"
bee is "apple"
```
Let's see how the memory works in Python Tutor
Compare assign with integers, strings and lists

<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>

Python 3.6 (known limitations)

Edit this code

- \( \rightarrow \) line that just executed
- \( \rightarrow \) next line to execute
Compare assign with integers, strings and lists

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

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

<table>
<thead>
<tr>
<th>x</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>6</td>
</tr>
</tbody>
</table>

y gets a copy of the value of x

Edit this code

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

---

Python 3.6

(known limitations)

<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>

Frames

Global frame

| 3 | x |
| 6 | y |

Objects

x gets a new value

---

Edit this code

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

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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;pink&quot;</td>
</tr>
</tbody>
</table>

Objects

Edit this code

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

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

<table>
<thead>
<tr>
<th>Global frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>x 3</td>
</tr>
<tr>
<td>y 6</td>
</tr>
<tr>
<td>m &quot;pink&quot;</td>
</tr>
<tr>
<td>n &quot;pink&quot;</td>
</tr>
</tbody>
</table>

n gets a copy of the value of m
Compare assign with integers, strings and lists

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

<table>
<thead>
<tr>
<th>Global frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>m</td>
</tr>
<tr>
<td>n</td>
</tr>
</tbody>
</table>

Objects

m gets a new value

Edit this code

- line that just executed
- next line to execute
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

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

Objects

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

Edit this code

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

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"

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

a and b refer to the same list!

b gets a copy of the value of a
Compare assign with integers, strings and lists

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 "ant"

Edit this code

- line that just executed
- next line to execute

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**

- Global frame
- lst1

**Objects**

```
0 | "a"  | 1 | "b"  | 2 | 3 | 2
```

2/2/23

Compsci 101, Spring 2023
List Cloning (or copying)

\[
\text{lst1} = ['a', 'b', 1, 2] \\
\text{lst2} = \text{lst1} \\
\text{lst3} = \text{lst1}[:]
\]
List Cloning (or copying)

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

![Diagram showing list cloning](image)
List Cloning (or copying)

```python
lst1 = ['a', 'b', 1, 2]
lst2 = lst1
lst3 = lst1[:]
lst1[-1] = "SUN"
```
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 \( \text{lst0} = [1, 2] \)
2 \( \text{lst1} = [3, 4, 5] \)
3 \( \text{lst2} = \text{lst0} + \text{lst1} \)
Concatenation:
length, create, copy, copy, assign

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

```python
1  lst0 = [1, 2]
2  lst1 = [3, 4, 5]
3  lst2 = lst0 + lst1
```

4. Copy right list

5. Assign lst2
Concatenation: Makes new List

```python
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. \( \text{lst0} = [1, 2] \)
2. \( \text{tmp} = \text{lst0} \)
3. \( \text{lst0} = \text{lst0} + [4] \)
Concatenation: Makes new List

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

\[
\text{lst0} = [1, 2] \\
\text{tmp} = \text{lst0} \\
\text{lst0} = \text{lst0} + [4]
\]
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
1 lst0 = [1, ['b', 3.0]]
2 lst1 = [4]
3 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?

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(...)`

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

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

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

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

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

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

```
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?

```python
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?

```python
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

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

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

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

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

- num gets first value in list

Frames

Objects

Global frame

- lst
- sum
- num

list

0 1 2
5 3 2
Trace through for loop

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

```
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

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

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

num gets third value in list
Trace through for loop

```python
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

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

Frames

No more values in lst

The for loop is done!
Trace through for loop

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

Print output (drag lower right corner to resize)

10

Print result

Frames

<table>
<thead>
<tr>
<th>Global frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>lst</td>
</tr>
<tr>
<td>sum</td>
</tr>
<tr>
<td>num</td>
</tr>
</tbody>
</table>

Objects

<table>
<thead>
<tr>
<th>list</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  1  2</td>
</tr>
<tr>
<td>5  3  2</td>
</tr>
</tbody>
</table>
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

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

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

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

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

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

- ch gets first character in word

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

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

Add ch to end of word

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

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

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

- ch gets second character in word

Global frame

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>animal</td>
<td>&quot;cat&quot;</td>
</tr>
<tr>
<td>word</td>
<td>&quot;catc&quot;</td>
</tr>
<tr>
<td>ch</td>
<td>&quot;a&quot;</td>
</tr>
</tbody>
</table>
Trace through for loop

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

Add ch to end of word
Trace through for loop

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

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

ch gets third character in word

Global frame
animal | "cat"
word | "catca"
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)
```

Add ch to end of word

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

1. \(\text{animal} = '\text{cat}'\)
2. \(\text{word} = \text{animal}\)
3. \(\text{for ch in animal:}\)
   
   \(\text{word} = \text{word} + \text{ch}\)
4. \(\text{print}()\)

Iterate over what is left in copy of word:

No more characters in word to process

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)`

Execute code after for loop

Global frame

<table>
<thead>
<tr>
<th>animal</th>
<th>&quot;cat&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
<td>&quot;catcat&quot;</td>
</tr>
<tr>
<td>ch</td>
<td>&quot;t&quot;</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
  ['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 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
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 for Comp Sci 101, Exam 1, Spring 2023

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>$6 / 3 = 2.0$</td>
</tr>
<tr>
<td>%</td>
<td>mod/remainder</td>
<td>$5 % 3 = 2$</td>
</tr>
<tr>
<td>**</td>
<td>exponentiation</td>
<td>$3 ** 2 = 9$, $2 ** 3 = 8$</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;x0&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>$4 \geq 3$ is True</td>
</tr>
<tr>
<td>&lt;=</td>
<td>is less than or equal to</td>
<td>$4 \leq 3$ 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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>not</td>
<td>flips/negates the value of a bool</td>
<td>(not x == 5) is False</td>
</tr>
</tbody>
</table>