Lab 2: `input()`, `if ... else`, String Operations, Variable Assignment

Ling 1330/2330: Intro to Computational Linguistics
Na-Rae Han
Objectives

- Learn Python basics
  - Taking input from keyboard with `input()`
  - Commenting
  - Comparison operators
  - Variable assignment, augmented assignment
  - String operations: `.split()`, `len()`, `.upper()`, etc.

- Control flow
  - Conditional statement: `if... elif... else`
  - Python indentation
Taking input from keyboard

- `input('prompt')`
  - Prompts for user keyboard input, stores value

- IDLE script editor window:

```
n = input('What is your name? ')  
print('Hello,', n)
```

- IDLE shell:

```
What is your name? Charlie Brown
Hello, Charlie Brown
```
Try it out

- `input('prompt ')`

- A sentence-making script (name it "noun_verb.py"):

  ```python
  noun = input('What is your noun? ')
  verb = input('What is your verb? ')
  sent = noun + ' ' + verb
  print('Your sentence is:')
  print(sent)
  ```

- Execution:

  ```
  What is your noun? cats
  What is your verb? sleep
  Your sentence is:
  cats sleep
  ```
# Takes noun and verb from user input
# and outputs a sentence as a single string

noun = input('What is your noun? ')
verb = input('What is your verb? ')
sent = noun + ' ' + verb  # a space in between
print('Your sentence is:')
print(sent)
Commenting

- **Comments** start with "#" ('pound' or 'hash')
  - They are ignored by Python interpreter
  - Works on a single line only
  - Good for notes, explanation, or other information for human readers

```python
# Takes noun and verb from user input
# and outputs a sentence as a single string

noun = input('What is your noun? ')
verb = input('What is your verb? ')
sent = noun + ' ' + verb  # a space in between
print('Your sentence is: ')
print(sent)
```

- Good comments make programs more readable and will help you diagnose problems. Get in the habit of commenting!
Multi-line comments

- Multi-line strings (in """" ... """""") are often used for a long comment spanning across multiple lines:

```python
""""This is a script that prompts for two user inputs: a noun and a verb. It then concatenates them into a sentence with a space in between and then prints it out."""

noun = input('What is your noun? ')
verb = input('What is your verb? ')
sent = noun + ' ' + verb  # a space in between
print('Your sentence is:')
print(sent)
```

- NOTE: These multi-line strings are NOT a true comment in that they are *not* ignored by the interpreter.
if and indentation

"If it is snowing shovel the sidewalk and do your homework."

- Is it:
  - If it is snowing
    - shovel the sidewalk
    - and do your homework

  or:
  - If it is snowing
    - shovel the sidewalk
    - and do your homework

Python uses indentation to define code blocks.
Conditional statement

if keyword

TEST

Don't forget colon :!!

```python
snowing = False
if snowing:
    print('Bart must shovel sidewalk')
else:
    print('Bart can go outside and play')
```
if ... else ...

```python
snowing = False
if snowing:
    print('Bart must shovel sidewalk')
else:
    print('Bart can go outside and play')
```

Indented block under `if`:
Executes only when `TEST` evaluates to `True`
```python
snowing = False
if snowing:
    print('Bart must shovel sidewalk')
else:
    print('Bart can go outside and play')
```

*else* keyword

Don't forget :!!
if ... else ...

```python
snowing = False
if snowing:
    print('Bart must shovel sidewalk')
else:
    print('Bart can go outside and play')
```

Indented block under **else**: Executes only when **TEST** evaluates to **False**
if ... elif ... else ...

- **elif**  \(\rightarrow\) "else if"

```python
snowing = False
raining = True
if snowing:
    print('Bart must shovel sidewalk')
elif raining:
    print('Bart must fix his umbrella')
else:
    print('Bart can go outside and play')
```
if ... elif ... else ...

- elif

  "else if"

```
# Assume snowing and raining are boolean variables
snowing = False
raining = True

if snowing:
    print('Bart must shovel sidewalk')
elif raining:
    print('Bart must fix his umbrella')
else:
    print('Bart can go outside and play')
```
if ... elif ... elif ... else ...

```python
snowing = False
raining = True
windy = True
if snowing :
    print('Bart must shovel sidewalk')
elif raining :
    print('Bart must fix his umbrella')
elif windy :
    print('Bart must close every window')
else :
    print('Bart can go outside and play')
```

What will be the output?

```
Bart must fix his umbrella
>>>```
Conditional and Boolean values

- if TEST :

  TEST can be any expression that evaluates to a Boolean value: **True** or **False**.
  - if True :
  - if 'cat'.endswith('at') :
  - if 'et' in 'scattered' :
  - if mary == mary2 :
  - if mary == mary2 or mary == mary3 :
  - ... and many more!
## Comparison operations

<table>
<thead>
<tr>
<th>Expression</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
</tbody>
</table>
Numeric comparisons

>>> 3 == 4
False

>>> 3*4 == 12
True

>>> 3 == 4
False

>>> 3 != 4
True

>>> 9**2 == 81
True

>>> 3 < 4
True

>>> 3 <= 3
True

>>> 3 >= 4
False
String comparisons

>>> 'coffee' == 'tea'
False
>>> 'coffee' < 'tea'
True
>>> 'Coffee' < 'coffee'
True
>>> 'coffee' < 'Tea'
False
>>> len('coffee') < len('tea')
False

- String comparisons are based on the ordering of character code points in Unicode
- All uppercase letters precede all lowercase letters!
Let's modify our noun_verb.py script so that the verb has number agreement

- How to do this? See if noun ends in 's', and if it does not, attach 's' suffix to the verb
  
  \[ \text{--- use if... : else : !} \]  

```python
# Takes noun and verb from user input
# and outputs a sentence as a single string
noun = input('What is your noun? ')
verb = input('What is your verb? ')
sent = noun + ' ' + verb    # a space in between
print('Your sentence is: ')  
print(sent)
```
# Takes noun and verb from user input
# and outputs a sentence as a single string

noun = input('What is your noun? ')
verb = input('What is your verb? ')

if noun.endswith('s'):  # if noun is plural
    sent = noun + ' ' + verb  # no verbal inflection needed
else:  # noun is singular
    sent = noun + 's'  # add inflection 's'

print('Your sentence is:')
print(sent)

>>> 'cats'.endswith('s')
True
>>> 'Mary'.endswith('s')
False
# Takes noun and verb from user input
# and outputs a sentence as a single string
noun = input('What is your noun? ')
verb = input('What is your verb? ')

if noun.endswith('s'):
    sent = noun + ' ' + verb  # no verbal inflection needed
else:
    sent = noun + ' ' + verb + 's'  # add inflection 's'

print('Your sentence is: ')
print(sent)
Let's polish up our sentence-generating program a little bit more:

```python
>>> What is your noun? snow
What is your verb? fall
Your sentence is:
"Snow falls."
```

- Capitalize the initial letter  `-- use `.capitalize()` method
- Add a period at the end
- Enclose the sentence in quotation " "

1/10/2017
# Takes noun and verb from user input
# and outputs a sentence as a single string
noun = input('What is your noun? ')
verb = input('What is your verb? ')

if noun.endswith('s') :
    sent = noun + ' ' + verb
    # no verbal inflection needed
else :
    sent = noun + ' ' + verb + 's'
    # add inflection 's'

print('Your sentence is:')
print(sent.capitalize()) # "Snow falls."
# Takes noun and verb from user input
# and outputs a sentence as a single string
noun = input('What is your noun? ')
verb = input('What is your verb? ')

if noun.endswith('s'):    # if noun is plural
    sent = noun + ' ' + verb    # no verbal inflection needed
else:    # noun is singular
    sent = noun + ' ' + verb + 's'    # add inflection 's'

print('Your sentence is:')
print('"" + sent.capitalize() + "."')    # "Snow falls."
Irregular plurals?

- But there are plural nouns that do not end with 's':
  - *people, mice, geese*

  ```
  >>>
  What is your noun?  people
  What is your verb?  walk
  Your sentence is:
  "People walks."  X
  >>>
  ```

- How to modify the script further so that we get correct number agreement with these nouns?
  <= expand if ... : to include tests to see if noun equals one of the words above
Accounting for irregular plurals

```python
# Takes noun and verb from user input
# and outputs a sentence as a single string
noun = input('What is your noun? ')  
verb = input('What is your verb? ')  

if noun.endswith('s') or noun == 'people' or noun == 'mice' or noun == 'geese':  
    sent = noun + ' ' + verb  
else:  # noun is singular
    sent = noun + ' ' + verb + 's'  # add inflection 's'

print('Your sentence is:')  
print('""' + sent.capitalize() + '""')  # "Snow falls."
```

noun_verb.py

Use \ to break a very long line!
Accounting for irregular plurals

```python
# Takes noun and verb from user input
# and outputs a sentence as a single string
noun = input('What is your noun? ')
verb = input('What is your verb? ')

if noun.endswith('s') or noun == 'people' or noun == 'mice' 
    or noun == 'geese':
    sent = noun + ' ' + verb
else:
    # noun is singular
    sent = noun + ' ' + verb + 's'  # add inflection 's'

print('Your sentence is:')
print('"" + sent.capitalize() + "."')  # "Snow falls."
```

noun_verb.py
Irregular nouns: simpler

# Takes noun and verb from user input
# and outputs a sentence as a single string

noun = input('What is your noun? ')
verb = input('What is your verb? ')

if noun.endswith('s') or noun in ['people', 'mice', 'geese']:
    sent = noun + ' ' + verb
else:
    sent = noun + ' ' + verb + 's'  # add inflection 's'

print('Your sentence is: ')
print('' + sent.capitalize() + '.')  # "Snow falls."
Python indentation: CAUTION

- Usually 4 SPACES per block: 

- Sometimes a single TAB is used instead: 

- "Intelligent" text editors optimized for Python programming provide indentation help:
  - Auto-indentation after if... : or while... :
  - Auto-convert 1 tab into 4 spaces
  - Default setting still differs editor to editor; you must pay attention
    - IDLE, Notepad++ (Win), TextWrangler (Mac), JEdit (any OS)

- Other plain text editors do not!
  - You have to manually indent, keep your indentation CONSISTENT
    - Notepad & Wordpad (Win), TextEdit (Mac)
Unfortunately, IDLE **shell** uses different indentation from IDLE **script editor**
- IDLE shell uses actual **TAB** character (= size of **8 spaces**)
- IDLE shell also has the >>> prompt...

Be careful when copying-and-pasting indented blocks between IDLE editor and shell
String operations

- **String**: A single piece of text, composed of a sequence of letters.

- **Operations on string objects**:
  - `print()` ← prints string
  - `len()` ← returns integer
  - `.endswith()`, `.startswith()` ← returns True/False
  - `in` ← returns True/False
  - `.upper()`, `.lower()` ← returns string
  - `.replace()` ← returns string
  - `+`, `+=` ← returns string
  - `.split()` ← splits a string into a list, returns it
Concatenation and variable assignment

```python
>>> v = 'walk'
>>> v + 'ed'
'walked'
>>> v
'walk'
>>> vd = v + 'ed'
>>> vd
'walked'
>>> v
'walk'
>>> v = v + 'ed'
>>> v
'walked'
>>> v == vd
True
```
Concatenation does not change original

>>> v = 'walk'
>>> v + 'ed'
'walked'

>>> v
'walk'

>>> vd = v + 'ed'
>>> vd
'walked'

>>> v
'walk'

>>> v = v + 'ed'
>>> v
'walked'

>>> v == vd
True
New variable

```python
>>> v = 'walk'
>>> v + 'ed'
'walked'
>>> v
'walk'
>>> vd = v + 'ed'
>>> vd
'walked'
>>> v
'walk'
>>> v = v + 'ed'
>>> v
'walked'
>>> v == vd
True
```

New variable `vd` is assigned to the output of concatenation

`v` is still unaffected
Changing the original variable

```python
>>> v = 'walk'
>>> v + 'ed'
'walked'
>>> v
'walk'
>>> vd = v + 'ed'
>>> vd
'walked'
>>> v
'walk'
>>> v = v + 'ed'
>>> v
'walked'
>>> v == vd
True
```

Here, `v` is assigned a new value: its former self suffixed with 'ed'
==: Equality comparison

```python
>>> v = 'walk'
>>> v + 'ed'
'walked'
>>> v
'walk'
>>> vd = v + 'ed'
>>> vd
'walked'
>>> v
'walk'
>>> v = v + 'ed'
>>> v
'walked'
>>> v == vd
True
```

*Double equal sign** `==` *tests if v and vd have an equal value!*
Assignment vs. equality comparison

=  

name = value

Attaches a name to a value

==

value₁ == value₂

Compares two values, returns True or False
Right Hand Side (RHS) first

```python
>>> num = 5
>>> num = num + 2
>>> num
7
```

- `num` has different values in one statement. How could this be?
Right Hand Side (RHS) first

```python
>>> num = 5
>>> num = num + 2
>>> num
7
```

- `num` has different values in one statement. How could this be?
- Answer: Right hand side is evaluated first, *and then* variable assignment happens.
+ and +=

```python
>>> homer = 'doh'
>>> print(homer)
doh
>>> homer = homer + '!
>>> print(homer)
doh!
>>> homer = homer + '!
>>> print(homer)
doh!!
>>> homer += '!
>>> print(homer)
doh!!!
>>> homer += '!
>>> print(homer)
doh!!!!
```
>>> homer = 'doh'
>>> print(homer)
doh
>>> homer = homer + '!
>>> print(homer)
doh!

>>> homer = homer + '!
>>> print(homer)
doh!!

>>> homer += '!

>>> homer += '!

>>> homer += '!

>>> homer += '!

>>> print(homer)
doh!!!!

Do exactly the same thing: suffixing homer with '!'
Augmented assignments

foo = foo + 10
foo -= 10

foo = foo - 10
foo &= 10

foo = foo * 10
foo *= 10

foo = foo / 10
foo /= 10

foo = foo ** 10
foo **= 10

foo = foo % 10
foo %= 10
Repeated commands:
Use **Ctrl + p/n** (Mac)
**Alt+ p/n** (Win)!

```python
>>> chor = 'tra'
>>> chor = chor + '-la'
>>> print(chor)
tra-la
>>> chor += '-la'
>>> print(chor)
tra-la-la
>>> chor += '-la'
>>> print(chor)
tra-la-la-la
```
Splitting a string into a list

>>> mary = 'Mary had a little lamb'
>>> mary
'Mary had a little lamb'
>>> print(mary)
Mary had a little lamb
>>> mary.split()
['Mary', 'had', 'a', 'little', 'lamb']
>>> mary.split(' ')  
['Mary', 'had', 'a', 'little', 'lamb']
>>> mary.split('a')
['M', 'ry h', 'd ', ' little l', 'mb']
>>> len(mary)
22
>>> len(mary.split())
5
Splitting a string with .split()

```python
>>> mary = 'Mary had a little lamb'

```  
```python
>>> mary.split(' ')
['Mary', 'had', 'a', 'little', 'lamb']

```  
```python
>>> mary.split('a')
['M', 'ry h', 'd ', ' little l', 'mb']

```
Splitting a string with \texttt{.split()}

```python
>>> mary = 'Mary had a little lamb'

>>> mary.split()
['Mary', 'had', 'a', 'little', 'lamb']

>>> mary.split(' ')
['Mary', 'had', 'a', 'little', 'lamb']
```

No separator given: splits on whitespace

Splits on every ' ':

Same result in this case, but not always
**len() works with strings and lists**

```python
>>> mary = 'Mary had a little lamb'

>>> mary.split()
['Mary', 'had', 'a', 'little', 'lamb']

>>> len(mary)
22
>>> len(mary.split())
5
```

- `len(string)` returns the length of string: # of characters
- `len(list)` returns the # of items in a list
Splitting and whitespace characters

```python
>>> hi = 'Hello mother,\nHello father.'
>>> print(hi)
Hello mother,
Hello father.
>>> hi.split()
['Hello', 'mother,', 'Hello', 'father.]
>>> hi.split('\n')
['Hello mother,', 'Hello father.]
```

A list of lines in hi

```python
>>> record = 'Lisa Simpson\nBorn 8/12 2002\nSpringfield'
>>> print(record)
Lisa Simpson  Born 8/12 2002 Springfield
>>> record.split('\\t')
['Lisa Simpson', 'Born 8/12 2002', 'Springfield']
```
.upper(), .lower(), .capitalize()

```python
>>> 'hello!'.upper()
'HELLO!'
>>> 'hello!'.upper().lower()
'hello!'
>>> 'hello!'.capitalize()
'Hello!'
>>> 
```

uppercases 'hello!' and then lowercases it back.
.replace()

```python
>>> foo = 'abracadabra'
>>> foo.replace('b', 'a')
'aaracadaara'
```

```python
>>> foo.replace('b', 'a').replace('a', 'b')
'bbrbcbdbbrb'
```

```python
>>> faa = 'colourful rumour'
>>> faa.replace('our', 'or')
'colorful rumor'
```
`.replace()`

```python
>>> foo = 'abracadabra'
>>> foo.replace('b', 'a')
'aaracadaara'
>>> foo.replace('b', 'a').replace('a', 'b')
'bbrbcdbbbrb'

>>> faa = 'colourful rumour'
>>> faa.replace('our', 'or')
'colorful rumor'

>>> mary = 'Mary had a little lamb'
>>> print(mary)
Mary had a little lamb
>>> mary.replace(' ', '')
'Maryhadalittlelamb'
```
.replace()

```python
>>> foo = 'abracadabra'
>>> foo.replace('b', 'a')
'aaracadaara'
>>> foo.replace('b', 'a').replace('a', 'b')
'bbrbcdbbbrb'

>>> faa = 'colourful rumour'
>>> faa.replace('our', 'or')
'colorful rumor'

>>> mary = 'Mary had a little lamb'
>>> print(mary)
Mary had a little lamb
>>> mary.replace(' ', '')
'Maryhadalittlelamb'
```

Removing every space: achieved by replacing ' ' with the empty string ''
.count()

```python
>>> foo = 'abracadabra'
>>> foo.count('a')
5
>>> foo.count('ab')
2

>>> faa = 'Six sick hicks nick six slick bricks'
>>> faa.count('ks')
2
>>> faa.replace('x', 'ks')
'Siks sick hicks nick siks slick bricks'
>>> faa.replace('x', 'ks').count('ks')
4
```

Stacked application: Counting 'ks' is done AFTER 'x' is replaced with 'ks'
>>> sim = 'Homer, Marge'
>>> sim += ', Bart'
>>> print(sim)

>>> sim.split()
['Homer,', 'Marge,', 'Bart']

>>> sim.split(2)
['Homer', 'Marge', 'Bart']

>>> sim.lower()
'homer, marge, bart'

>>> sim.lower().split()

>>> sim.split().lower()
>>> sim = 'Homer, Marge'
>>> sim += ', Bart'
>>> print(sim)
Homer, Marge, Bart
>>> sim.split()
['Homer,', 'Marge,', 'Bart']
>>> sim.split(', ')
['Homer', 'Marge', 'Bart']
>>> sim.lower()
'homer, marge, bart'
>>> sim.lower().split()
['homer,', 'marge,', 'bart']
>>> sim.split().lower()
Traceback (most recent call last):
  File "<pyshell#113>", line 1, in <module>
    sim.split().lower()
AttributeError: 'list' object has no attribute 'lower'
```python
>>> print(sim)
Homer, Marge, Bart
>>> sim += ' Lisa'
>>> print(sim)
Homer, Marge, Bart, Lisa
>>> sim2 = sim.replace(', ', '
')
>>> print(sim2)
Homer
Marge
Bart
Lisa
>>> sim2
```
>>> print(sim)
Homer, Marge, Bart
>>> sim += ', Lisa'
>>> print(sim)
Homer, Marge, Bart, Lisa
>>> sim2 = sim.replace(', ', ', 
')
'Homer
Marge
Bart
Lisa'
>>> print(sim2)
Homer
Marge
Bart
Lisa
>>> sim2
'Homer
Marge
Bart
Lisa'
Practice 3

```python
>>> sim2
'Homer\nMarge\nBart\nLisa'

>>> ['Homer', 'Marge', 'Bart', 'Lisa']

>>> len(sim2)
21

>>> len(sim2.split())

>>> print('There are', len(sim2.split()), 'Simpsons.')

>>> sim2.upper()

>>> print(sim2.upper())
```
Practice 3

```python
>>> sim2
'Homer
Marge
Bart
Lisa'
>>> sim2.split()
['Homer', 'Marge', 'Bart', 'Lisa']
>>> len(sim2)
21
>>> len(sim2.split())
4
>>> print('There are', len(sim2.split()), 'Simpsons.')
There are 4 Simpsons.
>>> sim2.upper()
'HOMER
MARGE
BART
LISA'
>>> print(sim2.upper())
HOMER
MARGE
BART
LISA
>>>```
Wrap-up

- **Office hours**
  - Na-Rae: Mon 3-5pm, Wed 3-4pm (may change)
  - Reed: to be announced

- **Next class:**
  - *Language and Computers*, Ch.1 Encoding language
  - More Python

- **Exercise #2**
  - [http://www.pitt.edu/~naraehan/ling1330/ex2.html](http://www.pitt.edu/~naraehan/ling1330/ex2.html)
  - Due Thursday 10:30am