Compsci 101
Selection, Lists, Sequences, Totem

Susan Rodger
January 20, 2022
E is for …

- **Escape Sequence**
  - Why \n is newline and \t is a tab

- **Encryption**
  - From Caesar Ciphers to SSL and beyond

- **Enumerate**
  - Iterating over data, counting

- **Emoticon**
  - 😊 😞
Luis von Ahn, Guatemalan entrepreneur
Duke BS Math 2000, CMU PhD CS

“I build systems that combine humans and computers to solve large-scale problems that neither can solve alone. I call this Human Computation, but others sometimes call it crowdsourcing.”

"In college, I thought my goal in life was to get a good GPA, but it's equally important to get involved with a good professor doing good research. Take advantage of what's going on around you."
Announcements

• APT-1 is due tonight! 11:30pm
  • Run each APT on the APT tester, 1 grace day
• QZ01-05 extended to Sat night 11:30pm (drop/add)
  • Remaining reading quizzes turn off 10:15am on due date
• Assignment 1 Faces is out, due Jan 27
  • Read the whole thing
  • Take assignment 1 quiz on Sakai – Due Jan 25
• Lab 2 Friday
  • Prelab 2 do before attending lab
• Always, Reading and Sakai quiz before next class
Why is this person so important to this course?
Why is this person so important to this course?

- Brad Miller, Runestone
- He built the Runestone infrastructure for online textbooks.
- Our Textbook is on his Runestone platform!
- Have you donated yet?
Top 10 list for surviving in CompSci 101

10. Read the book and Ask questions
9. Eat lots of pizza
8. Learn how to spell Rodger
7. Understand what you turn in
6. Visit your prof in her office hours on zoom and the UTAs in consulting hours
Top 10 list (cont)

5. Check Ed Discussion every day
4. Learn how to debug your programs
3. Follow the 7 step process
2. Seek help (One Hour Rule!)
1. Start programming assignments early
One Hour Rule for Getting Help

Work on Material -> Stuck

Has it been an hour?

Yes -> Get Help

No
PFTD

• Assignment 1
• Selection continued
• Strings
  • Sequence of characters, “CompSci 101”
• Lists
  • Heterogenous sequences
• Sequences
  • len(...), indexing, and slicing
Finish WOTO-3 from last time
import random

s += "What does a " + animal + " say?\n"
which = random.randint(0,1)

if which == 1:
    s += otherSound1 + "? No. "
    s += otherSound2 + "? No. "
else:
    s += otherSound2 + "? No. "
    s += otherSound1 + "? No. "

s += sound + "? Yes!\n"
Assignment 1: Faces
Learning Goals: Faces

• Understand differences and similarities:
  • Function definitions vs function calls
  • Functions with return statements vs those without
  • Functions with parameters vs those without
  • Functions can be arguments

• Be creative and learn lesson(s) about software design and engineering
  • Create a small, working program, make incremental improvements.
  • Read the directions and understand specifications!
<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Returns</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>part_DESCRIPTION</code></td>
<td>No parameters</td>
<td>A string</td>
<td><code>part_smiling_mouth</code></td>
</tr>
<tr>
<td><code>DESCRIPTION_face</code></td>
<td>No parameters</td>
<td>No return value, only prints</td>
<td><code>happy_face</code></td>
</tr>
<tr>
<td><code>face_with_DESCRIPTION</code></td>
<td>1 or 2 parameters of type function</td>
<td>No return value, only prints</td>
<td><code>face_with_mouth</code></td>
</tr>
<tr>
<td><code>faces_DESCRIPTION</code></td>
<td>No parameters</td>
<td>No return value, calls face functions</td>
<td><code>faces_fixed, faces_selfie, faces_random</code></td>
</tr>
</tbody>
</table>

`selfie_band, face_random` – helper functions!
Creating your program

- Start small and build incrementally
- Use seven steps! Plan what to do!
With functions grow by...

```python
def part_hair_pointy():
    a1 = r"012345678901234567"  
a2 = r" /\///\\\\\\\\"
    return a2

def happy_face():
    print(part_hair_pointy())

def faces_fixed():
    pass

def faces_selfie():
    pass

def faces_random():
    pass

if __name__ == '__main__':
    print("Fixed group of three faces")
    faces_fixed()

    print("Group of three self faces")
    faces_selfie()

    print("Group of three random faces")
    faces_random()
```

Minimal code that does run and can be submitted

Where go from here?

- Add face part functions to create happy_face()
- Create the next face function for faces_fixed and any new face part functions
- Try a face_with function
- Go to the next group of faces
- etc.
Faces Assignment
What should you do …

• Read the assignment
• Do the Assignment 1 reading quiz
• Create project and start writing code (do not need to finish)

• Goal: Find your first question about how to do this assignment then ask on Ed Discussion (anonymously) or at consulting/office hours
Selection Syntax

```python
if BOOLEAN_CONDITION:
    CODE_BLOCK_A
else:
    CODE_BLOCK_B
elif BOOLEAN_CONDITION:
    CODE_BLOCK_B
else:
    CODE_BLOCK_C
```

- What is similar and different?
  - What other variations could work?
  - Could only `elif...else` work?
- `if` – required
- `elif` – optional, as many as needed
- `else` – optional, no condition

Could this else not be here?
Boolean condition (True/False)

if BOOLEAN_CONDITION:
    CODE_BLOCK_A

• See `type(3 < 5)`
• Relational operators: `< <= > >= == !=`
• Boolean operators: `and or not`
Console on Booleans

```python
import sys; print('Python %s on %s'
sys.path.extend(['C:\Users\Susan'])

Python Console

>>> |
```
## Boolean Operations

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>A and B</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>A and B</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>A and B</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>A and B</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>A or B</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>A or B</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>A or B</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>A or B</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>not A</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>not A</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

**Example Statements:**

- IF it is raining **OR** it might rain today, I will carry an umbrella.
- IF my cat is hungry **AND** she likes the food, she will eat dinner.
- IF I did **NOT** have dessert yesterday, I may have dessert today.
Example with And and Or

```
x = 3
y = 8
if x < 2 or y > 2:
    print("first")
elif x > 2 and y < 2:
    print("second")
else:
    print("third")
```

OUTPUT:
```
x = 3
y = 2
if x < 2 or y > 2:
    print("first")
elif x > 2 and y < 2:
    print("second")
else:
    print("third")
```

OUTPUT:
Example with And and Or

```python
x = 3
y = 8
if x < 2 or y > 2:
    print("first")
elif x > 2 and y < 2:
    print("second")
else:
    print("third")
```

OUTPUT:
first

```python
x = 3
y = 2
if x < 2 or y > 2:
    print("first")
elif x > 2 and y < 2:
    print("second")
else:
    print("third")
```

OUTPUT:
third
WOTO-1 Review Functions and Booleans

• In your groups:
  • Come to a consensus

<table>
<thead>
<tr>
<th>A and B</th>
<th>A</th>
<th>B</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
When is a leap year?

- [https://en.wikipedia.org/wiki/Leap_year](https://en.wikipedia.org/wiki/Leap_year)

- "years which are multiples of four (with the exception of years divisible by 100 but not by 400)"

- 2004/4 = 501, 2004/100 = 20.04, 2004/400 = 5.01
  - Leap year
- 2200/4 = 550, 2200/100 = 22, 2200/400 = 5.5
  - Not Leap Year
- 2000/4 = 500 and 2000/100 = 20, 2000/400 = 5
  - Leap Year
WOTO-2: Which LeapYear correct?

• is_leap_one
• is_leap_two
WOTO-2: Which LeapYear correct?

- is_leap_one
- is_leap_two
- Hint: Is 1900 a leap year?

<table>
<thead>
<tr>
<th>def is_leap_one(year):</th>
</tr>
</thead>
<tbody>
<tr>
<td>if year % 400 == 0:</td>
</tr>
<tr>
<td>return True</td>
</tr>
<tr>
<td>if year % 100 == 0:</td>
</tr>
<tr>
<td>return False</td>
</tr>
<tr>
<td>if year % 4 == 0:</td>
</tr>
<tr>
<td>return True</td>
</tr>
<tr>
<td>return False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>def is_leap_two(year):</th>
</tr>
</thead>
<tbody>
<tr>
<td>if year % 4 == 0:</td>
</tr>
<tr>
<td>return True</td>
</tr>
<tr>
<td>if year % 100 == 0:</td>
</tr>
<tr>
<td>return False</td>
</tr>
<tr>
<td>if year % 400 == 0:</td>
</tr>
<tr>
<td>return True</td>
</tr>
<tr>
<td>return False</td>
</tr>
</tbody>
</table>
Which LeapYear correct?

• Is 1900 a leap year?

• Which program is correct?
• What is wrong with the program that is not correct?
Which LeapYear correct?

• Is 1900 a leap year? **NO**
  • $1900/4 = 475$
  • $1900/100 = 19$
  • $1900/400 = 4.75$  Not divisible by 400

• Which program is correct?  **Is_leap_one**
• What is wrong with the program that is not correct?
  • If statements in a different order!
  • The order matters!
Wikipedia Leap Year Algorithm

• See algorithm section
  • https://en.wikipedia.org/wiki/Leap_year

```python
def is_leap(year):
    if year % 4 != 0:
        return False       # not leap
    elif year % 100 != 0:  # 1968
        return True
    elif year % 400 != 0:
        return False       #1968
    else:
        return True           #2000
```
def is_leap_one(year):
    if year % 400 == 0:
        return True
    if year % 100 == 0:
        return False
    if year % 4 == 0:
        return True
    return False

def is_leap(year):
    if year % 4 != 0:
        return False
    elif year % 100 != 0:
        return True
    elif year % 400 != 0:
        return False
    else:
        return True
if's or if…elif…else?

- Remember steps 1-4 do not involve code!
- After have plan, choose based on what works best
  - There could be multiple ways to implement it
Strings

- $x = \text{“chair”}$
- $y = \text{“desk”}$
- $w = \text{len}(x)$
- $v = x[\text{len}(y)]$
- $t = x[\text{len}(x)]$
Strings

- $x = \text{“chair”}$
- $y = \text{“desk”}$
  
  - $z$ is “ask”
- $w = \text{len}(x)$
  
  - $w$ is 5
- $v = x[\text{len}(y)]$
  
  - $v$ is “r”
- $t = x[\text{len}(x)]$
  
  - $t$ is ERROR !!!!!!!
Strings

- \(x = \text{“chair”}\)
- \(y = \text{“desk”}\)
- \(z = x[2] + y[2] + y[3]\) \(z\) is \text{“ask”}
- \(w = \text{len}(x)\) \(w\) is 5
- \(v = x[\text{len}(y)]\) \(v\) is \text{“r”}
- \(t = x[\text{len}(x)]\) \(t\) is \text{ERROR !!!!!!!!}

```
\[
\begin{array}{c}
\text{‘c’} \\
\text{‘h’} \\
\text{‘a’} \\
\text{‘l’} \\
\text{‘r’}
\end{array}
\]
```

0 1 2 3 4
Lists

- Syntax: \([\text{ITEM}_1, \text{ITEM}_2, \text{ITEM}_3, \ldots]\)
  - Starts and ends with square brackets: \([\ldots]\)
  - Elements in the list are divided by commas ","
- Lists can be *heterogenous* sequence
  - Strings, ints, lists, anything

\[
[1, 2, 3] \\
["hello", "world"] \\
["count", "off", 1, 2, 3.0, "done"]
\]
Python Sequences

- Types String and List are both sequences
- A sequence in Python has
  - Length - `len(…)`
  - Membership – `in`
  - Indexing and slicing – `[n]`, `[n:m]`
- Difference:
  - String is immutable – cannot change
  - List is mutable – can change
len(…) for Python Sequences

- Length – the number of *elements* in a sequence
- `len(…)` – returns the length of a sequence

- `s="hello world"`  `l=["hello", "world"]`
  - What is `len(s)`?
  - What is `len(l)`?
len(...) for Python Sequences

• Length – the number of *elements* in a sequence
• len(...) – returns the length of a sequence

• s="hello world"    l=["hello", "world"]
  • What is len(s)?
    • 11
  • What is len(l)?
    • 2
in for Python Sequences

- *in* checks for membership in the sequence
  - True/False – if element in seq

- `s="hello world"  lst=["hello", "world"]`
  - What is an element for the string `s`? List `lst`?
    - What is ‘h’ in `s`?
    - What is ‘h’ in `lst`?
    - “hello” in `lst`?
in for Python Sequences

- `in` checks for membership in the sequence
  - True/False – if `element in seq`

- `s="hello world" lst=["hello", "world"]`
  - What is an element for the string `s`? List `lst`?
    - `s` has ‘h’, ‘e’, etc, `lst` has “hello”, “world”
  - What is ‘h’ in `s`? `True`
  - What is ‘h’ in `lst`? `False`
  - “hello” in `lst`? `True`
Indexing Python Sequences

- `s="hello world" l=["hello", "world"]`
- Indexing provides access to individual elements
  - Compare `s[0]` and `l[0]`
    - Start with 0 offset, what is last valid positive index?
  - Compare `s[-1]` and `l[-1]`
    - What is negative index of second to last element?
  - Index `−n` is the same as index `len(seq) − n`

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H</td>
<td>E</td>
<td>L</td>
<td>L</td>
<td>O</td>
<td>W</td>
<td>O</td>
<td>R</td>
<td>L</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-11</td>
<td>-10</td>
<td>-9</td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>
Indexing Python Sequences

- `s="hello world"` `l= ["hello", "world"]`

- Indexing provides access to individual elements
  - Compare `s[0]` and `l[0]` “h” vs “hello”
    - Start with 0 offset, what is last valid positive index?
  - Compare `s[-1]` and `l[-1]` “d” vs “world”
    - What is negative index of second to last element?
  - Index \(-n\) is the same as index `len(seq) - n`

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H</td>
<td>E</td>
<td>L</td>
<td>L</td>
<td>O</td>
<td>W</td>
<td>O</td>
<td>R</td>
<td>L</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>-11</td>
<td>-10</td>
<td>-9</td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>
Slicing Python Sequences

- `s="hello world"
- `lst=['my', 'big', 'beautiful', 'world']`
- Slicing provides sub-sequence (string or list)
  - `seq[n:m]` – all elements i, s.t. `n <= i < m`
  - Compare `s[0:2]` and `lst[0:2]`
    - `s[0:2]` is "he"
    - `lst[0:2]` is ['my', 'big']
- What is length of subsequence? `len(lst[1:3])`
  - `lst[1:3]` is ['big', 'beautiful']
  - `len(lst[1:3])` is 2
Slicing Python Sequences

- `s = "hello world"
- `lst = ["my", "big", "beautiful", "world"]`
- Slicing provides sub-sequence (string or list)
  - `seq[n:m]` – all elements `i`, s.t. `n <= i < m`
  - Compare `s[0:2]` and `lst[0:2]`
    - `s[0:2]` is "he"
    - `lst[0:2]` is ["my", "big"]
- What is length of subsequence? `len(lst[1:3])`
  - `lst[1:3]` is ["big", "beautiful"]
  - `len(lst[1:3])` is 2
Slicing Python Sequences (more)

- \( s = "hello\ world" \)
- \( lst=['my', 'big', 'beautiful', 'world'] \)
- Slicing provides sub-sequence (string or list)
  - Compare \( s[4:-1] \) and \( lst[2:-1] \)
    - \( s[4:-1] \) is
    - \( lst[2:-1] \) is
  - Is last index part of subsequence?

- Omit last value. Compare \( s[2:] \), \( s[:3] \)
  - \( s[2:] \) is
  - \( s[:3] \) is
Slicing Python Sequences (more)

- s = "hello world"
- lst=["my", "big", "beautiful", "world"]
- Slicing provides sub-sequence (string or list)
  - Compare s[4:-1] and lst[2:-1]
    - s[4:-1] is  "o worl"
    - lst[2:-1] is  ["beautiful"]
  - Is last index part of subsequence?
    - NO, in s[2:4] we go up to but not including 4
  - Omit last value. Compare s[2:] , s[:3]
    - s[2:] is "llo world"
    - s[:3] is "hel"
• In your groups:
  • Come to a consensus

• DIDN’T DO THIS ONE!