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
  • 😊 😞

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

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.”

duolingo

I’m not a robot

Forbes

Bill Gates

"EL PROFESOR"
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

1. Work on Material
2. Stuck
3. Has it been an hour?
   - No
   - Yes, Get Help

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

What does the animal say?

```python
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!

Function Name Format

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Returns</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>part_DESCRIPTION</td>
<td>No parameters</td>
<td>A string</td>
<td>part_smiling_mouth</td>
</tr>
<tr>
<td>DESCRIPTION_face</td>
<td>No parameters</td>
<td>No return value, only prints</td>
<td>happy_face</td>
</tr>
<tr>
<td>face_with_DESCRIPTION</td>
<td>1 or 2 parameters of type function</td>
<td>No return value, only prints</td>
<td>face_with_mouth</td>
</tr>
<tr>
<td>faces_DESCRIPTION</td>
<td>No parameters</td>
<td>No return value, calls face functions</td>
<td>faces_fixed, faces_selfie, faces_random</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…

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.

Selection Syntax

- 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

Boolean condition (True/False)

- See type(3 < 5)
- Relational operators: < <= > >= == !=
- Boolean operators: and or not

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
Console on Booleans

Boolean Operations

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

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

```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: third

Example with And and Or

```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: first
WOTO-1 Review Functions and Booleans

• In your groups:
  • Come to a consensus

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>A and B</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>A and B</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

When is a 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

• Hint: Is 1900 a leap year?

```python
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
```

```python
def is_leap_two(year):
    if year % 4 == 0:
        return True
    if year % 100 == 0:
        return False
    if year % 400 == 0:
        return True
    return False
```
Which LeapYear correct?

• Is 1900 a leap year?

  - Not divisible by 400
  - 1900/4 = 475
  - 1900/100 = 19
  - 1900/400 = 4.75

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

Which LeapYear correct?

• Is 1900 a leap year? NO

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 % 400 == 0:
        return True
    if year % 100 == 0:
        return False
    if year % 4 == 0:
        return True
    return False
```

Flowchart: if vs if...elif...else

```python
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
```
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

- \( x = \text{"chair"} \)
- \( y = \text{"desk"} \)
- \( z = x[2] + y[2] + y[3] \)  \( z \) 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 \( \text{ERROR !!!!!!!} \)
Lists

- Syntax: `[ITEM_1, ITEM_2, ITEM_3, ...]`
  - Starts and ends with square brackets: `[ ... ]`
  - 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

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

• What is `len(s)`?
  • What is `len(l)`?
```

```python
• 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

- \texttt{s=“hello world”} \texttt{lst=[“hello”, “world”]}
  - What is an element for the string \texttt{s}? List \texttt{lst}?
    - What is ‘h’ in \texttt{s}?  
    - What is ‘h’ in \texttt{lst}?  
    - “hello” in \texttt{lst}?  

**Indexing Python Sequences**

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

- \texttt{“h” vs “hello”}
- \texttt{“d” vs “world”}
Slicing Python Sequences

- `s = "hello world"
- `lst=["my", "big", "beautiful", "world"]`
- Slicing provides sub-sequence (string or list)
  - `seq[n:m]` – all elements \(i, \text{s.t. } n \leq i < m\)
  - 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 "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"
WOTO-3 Sequence Length Indexing

• In your groups:
  • Come to a consensus

• DIDN’T DO THIS ONE!