Compsci 101
Lists, Mutation, Objects

Susan Rodger
September 15, 2022

F is for …

- Function
  - Key to all programming
- Floating Point
  - Decimal numbers aka Python float
- File
  - Sequence of stored bits

Genesis Bond ‘16

- Struggled at Duke
  - 5 years
- Revature
  - Trainer Full Stack Development
  - She worked smarter
- Meta (Facebook)
  Engineer, big success!

“Poor preparation promotes poor performance. In anything you do, your preparation will show.”

Announcements

- Assign 1 Faces, Sakai QZ due Sept 20 (no grace day)
  - Program is due Sept 22!
- Lab 2 Friday, Do Prelab 2 before lab
- Sakai QZ due by lecture time each day
- Exam 1 – Tuesday, Sept 27
  - In person during class
  - Go over next time, see old exams
- Need SDAO letters for exams!
  - Email them to Ms. Velasco
    yvelasco@cs.duke.edu
Sage Learning Communities (Optional)

- STEM Advancement through Group Engagement (SAGE) Learning Communities are now available for CompSci 101!
- Strengthen programming skills!
- Hands on collaborative programming activities
- Sign up at: https://arc.duke.edu/sage

PFTD

- Strings, Lists, Sequences, Slicing
- Functions as Parameters
- Debugging
- List concatenation and nesting
- Mutability
- Objects and what that means

Strings

- x = "chair"
- y = "desk"
- w = len(x)
- v = x[ len(y) ]
- t = x[ len(x) ]

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

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

### 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`?

### Indexing Python Sequences

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

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<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>
</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
    - `lst[0:2]` is
  - What is length of subsequence? `len(lst[1:3])`
    - `lst[1:3]` is
    - `len(lst[1:3])` is

WOTO-1 Sequence Length Indexing

- In your groups:
  - Come to a consensus

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?
    - NO
  - Omit last value. Compare `s[2:]` , `s[:3]`
    - `s[2:]` is
    - `s[:3]` is

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!
Name vs Value vs Type

What are the arrows?

- Name: Enzo’s Pizza Co.
- Address (arrow): 2608 Erwin Rd # 140, Durham, NC 27705
- Value: Physical Store

Pizza.py

```
def enzospizzaco():
    print("Pizza!")
    return "2608 Erwin Rd # 140, Durham, NC 27705"
def eatfood(where):
    print("Let's go eat!")
    address = where()
    print("The address is", address)
if __name__ == '__main__':
    eatfood(enzospizzaco)
```

Functions can be arguments
Pizza2.py - Pass multiple functions to eatfood

```python
7     def naanstop():
8         print("Indian cuisine!")
9     return "2812 Erwin Road, Durham, NC 27705"
10
11     def enzospizzaco():
12         print("Pizza!")
13     return "2608 Erwin Rd # 140, Durham, NC 27705"
14
15     def eatfood(where):
16         print("Let's go eat!")
17         address = where()
18         print("The address is", address)
19
20     if __name__ == '__main__':
21         eatfood(enzospizzaco)
22         eatfood(naanstop)
```

In Assignment 1 Faces

```python
def face_with_mouthAndEyes(mouthfunc, eyefunc):
    print(part_hair_squiggly())
    print(eyefunc())
    print(part_nose_up())
    print(mouthfunc())
    print(part_chin_simple())
```

In Assignment 1 Faces

```python
def face_random():
    eyefunc = part_eyes_sideways
    mouthfunc = part_mouth_oh
    x = random.randint(1,4)
    if x == 1:
        mouthfunc = part_mouth_frown
        eyefunc = part_eyes_ahead
    < code not shown >

# now call the function
face_with_mouthAndEyes(mouthfunc, eyefunc)
```

Debugging

- **Finding what is wrong + fixing it**
  - Finding is its own skill set, and many find difficult
  - Fixing: revisit Step 1—5
**Debugging Steps**

1. Write down what is happening
2. Brainstorm
3. Go through list
4. Found problem? Yes! Fix it!
5. No

---

**Relate W’s to Debugging**

- Who was involved?
- What happened?
- Where did it take place?
- When did it take place?
- Why/How did it happen?

Translate these questions to debugging

---

**Step 7 -> Steps 1-4 or 5**

**Which year is a leap year?**

- A Leap Year must be divisible by four.
- But Leap Years don't happen every four years ... there is an exception.
  - If the year is also divisible by 100, it is not a Leap Year unless it is also divisible by 400.
List Concatenation

- String concatenation:
  - “hi” + “ there” == “hi there”

- List concatenation:
  - [1, 2] + [3, 4] == [1, 2, 3, 4]

List examples

- [1, 2] + [3, 4]
- lst1 = ['a', 'b']
- lst2 = [5, 6]
- lst1 + lst2
- lst1 + “c”
- lst1 + ['c''

Nested Lists

- Lists are heterogenous, therefore!
  - lst = [1, ‘a’, [2, ‘b’]] is valid
  - len(lst) == 3
    - [2, ‘b’] is one element in list lst

- How to index?
  - [...] all the way down
  - lst[2][1] returns ‘b’
### Nested Lists with Python Tutor

- `lst = ['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
1 lst1 = ['Hello', 'world']
2 print(lst1)
3 lst2 = [lst1[0]] + ['Ashley']
4 print(lst2)
5 print(lst1)
6 lst1[1] = 'Ashley'
7 print(lst1)
8
9 lst3 = [1, ‘a’, [2, ‘b’]]
10 print(lst3)
11 lst3[2][1] = ‘c’
12 print(lst3)
```

### Mutating Lists

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