Compsci 101
Lists, Mutation, Objects
Live Lecture

Susan Rodger
January 25, 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
- Facebook Engineer, big success!

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

Announcements

- Assign 1 Faces, due Thursday, January 27
  - Assignment quiz due tonight!
- Lab 3 Friday, Do Prelab 3 before lab
- Sakai QZ due by lecture time each day
- Exam 1 – Tuesday, Feb 1
  - This exam will be online
  - Other exams in person, likely

- Need SDAO letters for exams!
  - Email them to Ms. Velasco
    yvelasco@cs.duke.edu
Exam 1 – Feb 1, 2022

- All lecture/reading topics through Tues. Jan 25
- Understand/Study
  - Reading, lectures
  - Assignment 1, APT-1,
  - Labs 0-2, Lab 3 Part 3 (review questions)
- Logistics:
  - Online, on Gradescope
  - Pick time to take it on Feb 1
  - Once you start, you have 90 minutes
    - Ms. Velasco will contact you if you get accommodations

Exam 1 – Feb 16, 2021 (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
  - See Exam 1 Reference sheet
- Exam 1 is your own work! Do not consult with anyone else.
  - Rules posted in Sakai Announcement
  - Read the rules before taking the exam

WOTO-1 Sequence Length Indexing
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

<table>
<thead>
<tr>
<th>Names</th>
<th>Values</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Address</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What are the arrows?

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

Pizza.py

```python
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

```python
def enzospizzaco():
    return "2608 Erwin Rd # 140, Durham, NC 27705"

def eatfood(where):
    address = where()
    print("The address is", address)
    if __name__ == '__main__':
        eatfood(enzospizzaco)

if __name__ == '__main__':
    eatfood(enzospizzaco)
```

Pizza2.py - Pass multiple functions to eatfood

```python
def naanstop():
    print("Indian cuisine!")
    return "2812 Erwin Road, Durham, NC 27705"

def enzospizzaco():
    print("Pizza!")
    return "2608 Erwin Rd # 140, Durham, NC 27705"

def eatfood(where):
    address = where()
    print("The address is", address)
    if __name__ == '__main__':
        eatfood(enzospizzaco)
```

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())
```

```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

How Not To Debug

• Bad (but tempting) way to debug
  - Change a thing. Does it work now?
  - No … another change … how about this?
• Trust doctor if they say?
  - “Ok try this medicine and see what happens?”
• Trust mechanic if they say?
  - “Let’s replace this thing and see what happens”

Debugging Steps

1. Write down exactly what is happening
   1. input, output, what should be output
   2. ____ happened, but ____ should happen
2. Brainstorm possible reasons this is happening
   1. Write down list of ideas
3. Go through list
4. Found it?
   1. Yes, fix it using the 7-steps
   2. No, go back to step 2

Remember: One-hour rule
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.

WOTO: Buggy Leap Year

Input: 1900
Output: True
Should be: False

def is_leap_year(year):
    if year % 4 == 0:
        return True
    if year % 100 == 0:
        return False
    if year % 400 == 0:
        return True
    return False
List Concatenation

- **String concatenation:**
  - "hi" + " there" == "hi there"

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

List examples

```python
[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

```python
lst[2][1] = 'b'
```

- How to index?
  - [...] all the way down
  - lst[2][1] returns 'b'

Nested Lists with Python Tutor

```
1 lst = [1, 'a', [2, 'b']]
2 print(len(lst))
3 print(type(lst[2]))
4 print(lst[2])
5 print(lst[2][1])
```
Mutating Lists

- `lt = ['Hello', 'world']`
  - Change to: ['Hello', 'Ashley']
- Concatenation: `lt = [lt[0]] + ['Ashley']`
- Index: `lt[1] = 'Ashley'`

- How change ‘b’ in `lt = [1, ‘a’, [2, ‘b’]]`?
  - `lt[2][1] = ‘c’`

---

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

---

```python
lst1 = ['Hello', 'world']
print(lst1)
lst2 = [lst1[0]] + ['Ashley']
print(lst2)
print(lst1)
lst1[1] = 'Ashley'
print(lst1)
lst3 = [1, 'a', [2, 'b']]
print(lst3)
lst3[2][1] = 'c'
print(lst3)
```
WOTO-3 List Mutation