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

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

What are the arrows?

Pizza.py

```python
6 def enzospizzaco():
7     print("Pizza!")
8     return "2608 Erwin Rd # 140, Durham, NC 27705"
9
10 def eatfood(where):
11     print("Let's go eat!")
12     address = where()
13     print("The address is", address)
14
15 if __name__ == '__main__':
16     eatfood(enzospizzaco)
```
Functions can be arguments

```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)
```

Output of Pizza2.py

```
C:\Users\Susan\AppData\Local\Programs\Python\Python3\
Let's go eat!
Pizza!
The address is 2608 Erwin Rd # 140, Durham, NC 27705
Let's go eat!
Indian cuisine!
The address is 2812 Erwin Road, Durham, NC 27705
```

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):
    print("Let's go eat!")
    address = where()
    print("The address is", address)

if __name__ == '__main__':
    eatfood(enzospizzaco)
    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())
```
Two parameters that are functions!

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

Variables whose values are function names

```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)
```

Change their value

Pass functions as arguments

```python
<code not shown>
# now call the function
face_with_mouthAndEyes(mouthfunc, eyefunc)
```

Find 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”

It may be easy, but that doesn’t make it a good idea!

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

This is what experts do!
Remember: One-hour rule

Debugging Steps

Write down what is happening

Brainstorm

Go through list

No

Found problem?

Yes!

Fix it!

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
Relate W’s to Debugging

- Who was involved?
  - Which variables are involved?
- What happened?
  - What kind of error/bug is it?
- Where did it take place?
  - Where in the code did this happen?
- When did it take place?
  - Does it happen every time? For certain cases?
- Why/How did it happen?
  - Given the answers to the above, how did the error/bug happen?

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

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

Input: 1900
Output: True
Should be: False
WOTO: Buggy Leap Year

- Who? (Which variables)
- What kind of bug is it?
- Where in the code?
- When does it happen?
- Why/How did it happen?

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

Input: 1900
Output: True
Should be: False

1/25/22 Compsci 101, Spring 2022 29

WOTO: Buggy Leap Year

- Who? (Which variables)
- What kind of bug is it?
- Where in the code?
- When does it happen?
- Why/How did it happen?

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

Input: 1900
Output: True
Should be: False

1/25/22 Compsci 101, Spring 2022 30

Buggy Leap Year – add print tests

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

if __name__ == '__main__':
    print('Is 2016 a leap year? (should be True)', is_leap_year(2016))
    print('Is 2019 a leap year? (should be False)', is_leap_year(2019))
    print('Is 1900 a leap year? (should be False)', is_leap_year(1900))

Output:
Is 2016 a leap year? (should be True) True
Is 2019 a leap year? (should be False) False
Is 1900 a leap year? (should be False) True

Buggy Leap Year – Which “return true”?

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

# This is the correct return statement
if year % 400 == 0:
    print("DEBUG: if year % 400 == 0:")
    return True

Output:
DEBUG: if year % 4 == 0:
Is 2016 a leap year? (should be True) True
Is 2019 a leap year? (should be False) False
DEBUG: if year % 4 == 0:
Is 1900 a leap year? (should be False) True

1/25/22 Compsci 101, Spring 2022 31
Correct Leap Year – ifs correct order

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

Output:
- Is 2016 a leap year? (should be True) True ✓
- Is 2019 a leap year? (should be False) False ✓
- Is 1900 a leap year? (should be False) False ✓

Why Leap Year Buggy?

- **Why:** Should not always return True if year is divisible by 4
- **Solution:** Check first for %400, then %100, and finally %4

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”]`
List examples

\[ [1, 2] + [3, 4] \quad [1, 2, 3, 4] \]
\[ \text{lst1} = ['a', 'b'] \]
\[ \text{lst2} = [5, 6] \]
\[ \text{lst1} + \text{lst2} \quad ['a', 'b', 5, 6] \]
\[ \text{lst1} + "c" \quad \text{ERROR} \]
\[ \text{lst1} + ['c'] \quad ['a', 'b', 'c'] \]

Nested Lists

- Lists are heterogenous, therefore!
  - \( \text{lst} = [1, 'a', [2, 'b']] \) is valid
  - \( \text{len(lst)} == 3 \)
    - \([2, 'b']\) is one element in list \( \text{lst} \)

- How to index?
  - \([...)\) all the way down
  - \( \text{lst}[2][1] \) returns 'b'

Mutating Lists

- \( \text{lt} = ['Hello', 'world'] \)
  - Change to: \( ['Hello', 'Ashley'] \)
- Concatenation: \( \text{lt} = [\text{lt}[0]] + ['Ashley'] \)
- Index: \( \text{lt}[1] = 'Ashley' \)

- How change 'b' in \( \text{lt} = [1, 'a', [2, 'b']]? \)
  - \( \text{lt}[2][1] = 'c' \)
### 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)
```

### 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
```

```
val is 0
bee is 0
```
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

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

val is 20
bee is 0
```

---

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

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

val is "apple"
bee is "apple"
```
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

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

val is "applesauce"
bee is "apple"
```

bat or ant?

Python 3.6
```
1  a = ["pig", "cow", "dog", "bat"]
2  b = a
3  print(a)
4  a[-1] = "ant"
5  print(a)
6  print(b)
```

`print(b)` -> has ‘bat’ or ‘ant’?