CompSci 101
Fall 2021

Lecture 7

Reminders

- **Identity & Computing Lecture Series**
  - https://identity.cs.duke.edu/speakerSeries.html
  - 9/20-Dr. Safiya Noble
  - 9/27-Dr. Michele Williams

- **Assignments**
  - Exam 1 Review Questions
    - ***Must be posted in Ed thread***
  - **Exam 1**
    - **3 points extra credit (2 survey responses)**

Key instructions

- **Input**
- **Output**
- **Assignments* ✓**
- **Math/Logic ✓**
- **Conditionals✓**
- **Repetition**

not listed in book

Python Data Types

- **int, float, bool ✓**
- **Collections**
  - Strings ←
  - Lists ←
  - Tuples
  - Sets
  - Dictionaries

PFTD

- **Debugging**
  - PAY ATTENTION TO ERROR MESSAGES
- **Mutating Lists**

“The mere imparting of information is not education.”

- Dr. Carter G. Woodson

People to Know: Dr. Clarence "Skip" Ellis

- Beloit (BS, Math/Physics)
- University of Illinois (MS-Math, PhD-CS)
- 1st Black person to earn a PhD in CS
- Fellow, ACM

Types of Errors

- **Syntax**
  - Structure of program and rules to follow
  - E.g., forget a ‘:’ or indentation (won’t compile)
- **Runtime**
  - Don’t appear until executing program
  - print(greeting)→ greeting undefined
- **Semantic**
  - Program runs, but won’t do the right thing
  - *This is why DESIGN FIRST matters*

Section 3.4
(Types of Error Messages)

- **ParseError**
  - Error in syntax
- **TypeError**
  - Combine two incompatible objects
- **NameError**
  - Use variable before assigning value
- **ValueError**
  - Function expects certain value type and receives incompatible one

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!

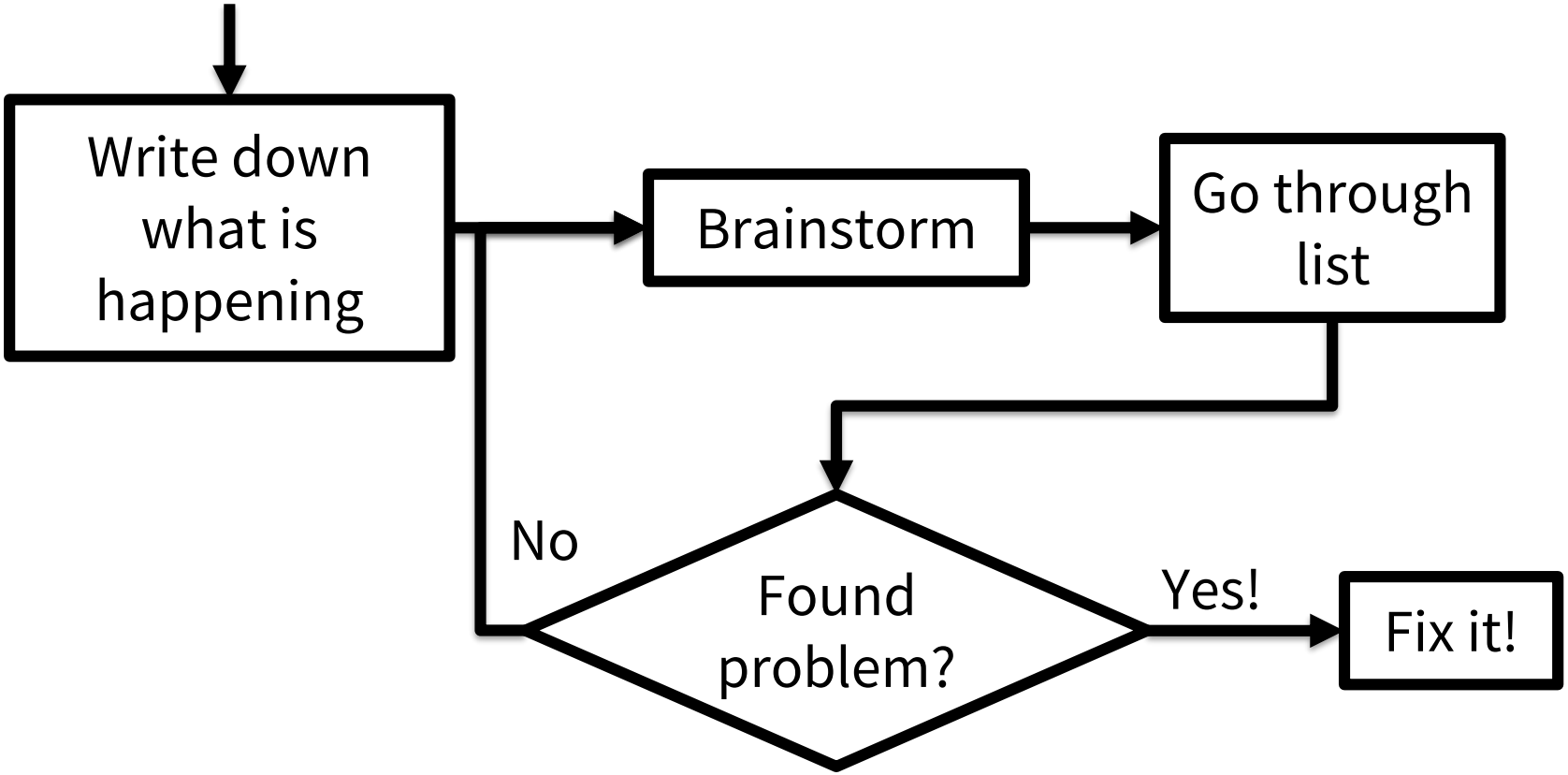
Debugging Steps

1. **Write down exactly what is happening**
   1. input, expected output, actual output
   2. ____ happened, but ____ should happen
2. **Brainstorm possible reasons this is happening**
   1. Write down 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



Debugging: 5 W's

- **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?

Activity 1: W's of withCutOff
http://bit.ly/101f21-09-14-1

Bug Example: Score

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

```
7   def withCutOff(total, possible):
8       denominator = int(possible*0.75)
9       percent = total/denominator
10      if percent > 1:
11          percent = 1.0
12      return percent
```

Input: (1,1)
Output: Error
Should be: 1.0

Bug Example: Score

- **Who? (Which variables)**
  - total, denominator
- **What kind of bug is it?**
  - Runtime error
- **Where in the code?**
  - Line 9
- **When does it happen?**
  - Input (1,1), but not (75,100) nor (50,134)
- **Why/How did it happen?**
  - Divide by zero, so denominator variable is zero

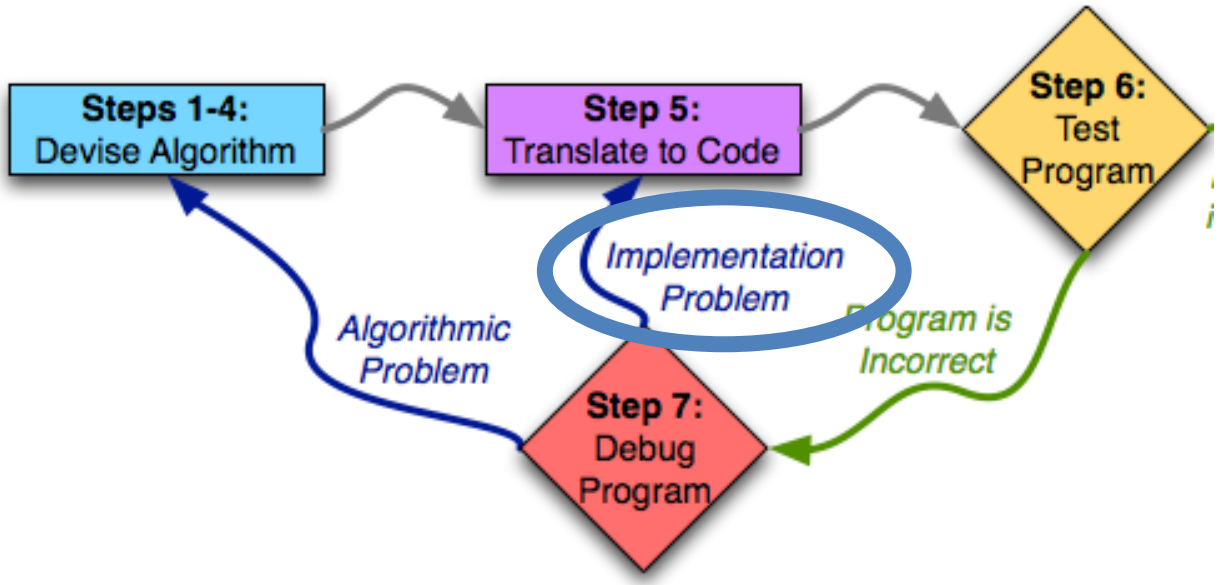
```
7   def withCutOff(total, possible):
8       denominator = int(possible*0.75)
9       percent = total/denominator
10      if percent > 1:
11          percent = 1.0
12      return percent
```

Why is it 0? Where does it get its value?

Input: (1,1)
Output: Error
Should be: 1.0

Why Is Bug Present?

- **Why: Not accounting for possibility of rounding down to 0**
- **Solution: Check if denominator is 0 and have special case**



Is this code correct?

```
7   def calculateAge(birthYear, currentYear):
8       age=currentYear-birthYear
9       return age
```

Strings are IMMUTABLE

```
name="Tim Johnson"
print(name)
name[0]='K'
```

```
name="Tim Johnson"
print(name)
name="Kyla Johnson"
```

List Concatenation/Repetition

- **String concatenation:**
  - `“hi” + “ there” == “hi there”`
- **List concatenation:**
  - `[1, 2] + [3, 4] == [1, 2, 3, 4]`
- **`What about these?`**
  - `[1, 2] + “yellow”`
  - `[1, 2] + 3`

Appending to List

- ***list_name*.append(*item*)**

- **Appends *item* to end of *list_name***
  - *list_name* now contains *item*
- **Doesn't require assignment to a variable**

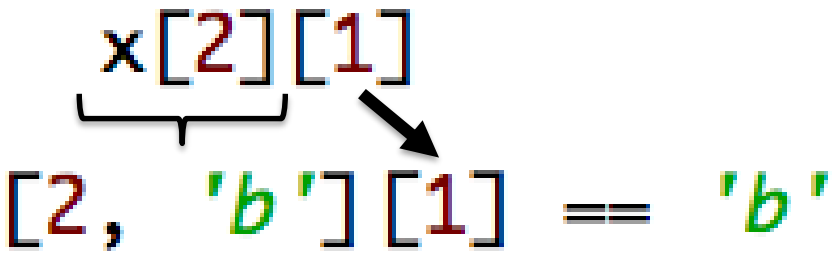
list1 = [1, 2]

list1.append("yellow")

Nested Lists

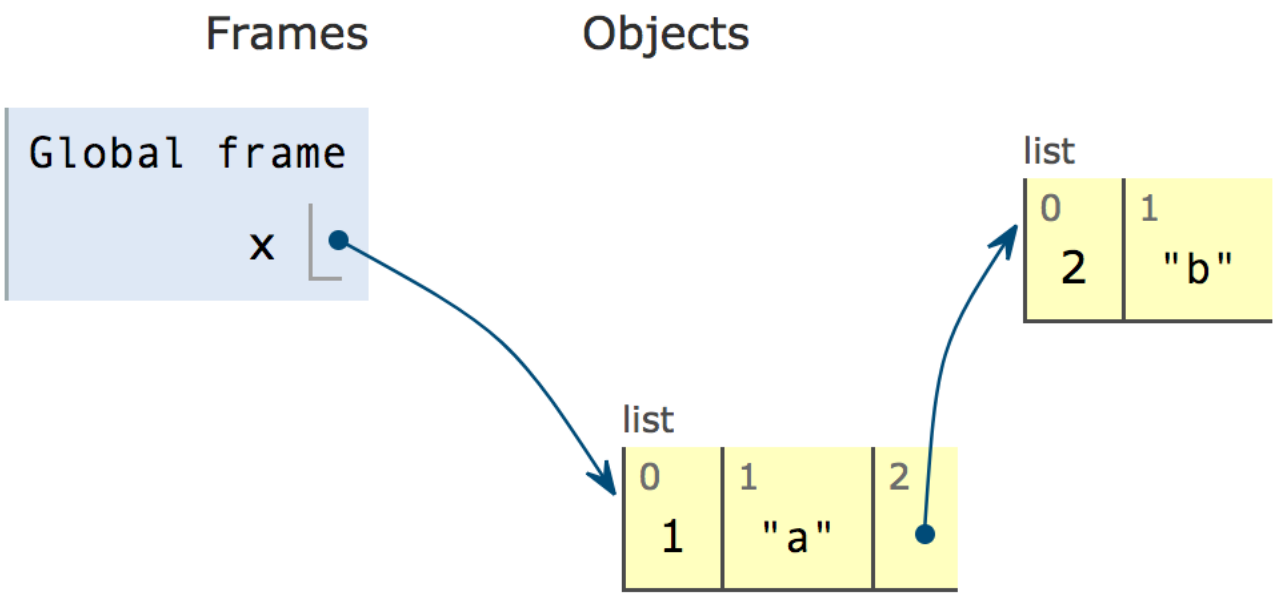
- **Lists are heterogenous, therefore!**
  - `x = [1, ‘a’, [2, ‘b’]]` is valid
  - `len(x) == 3`
    - `[2, ‘b’]` is one element in list x

- **How to index?**
  - [...] all the way down
  - x[2][1] returns ‘b’



Nested Lists

- **Python Tutor:**
- `x = [1, 'a', [2, 'b']]`



Lists are MUTABLE!

- **x = [‘Hello’, ‘world’]**
  - Change to: [‘Hello’, ‘Ashley’]
  - x[1] = ‘Ashley’
  - Is there another solution?

- **How change ‘b’ in x = [1, ‘a’, [2, ‘b’]] to ‘c’?**
  - x[2][1] = ‘c’
  - Is there another solution?

Examples

- items = [5, [“red”, “blue”], “13”]

- **What is the value of items after each of these?**
  - items.append(13)
    - [5, [“red”, “blue”], “13”, 13]
  - items[1][0] = 4
    - [5, [4, “blue”], “13”, 13]

Objects

- **Sometimes it helps to know how things "work"**
  - Sometimes it's wonderful to be oblivious (abstraction)

- **Object – a "thing" in memory/object space**
  - Everything in Python is an object (state, behavior, identity)

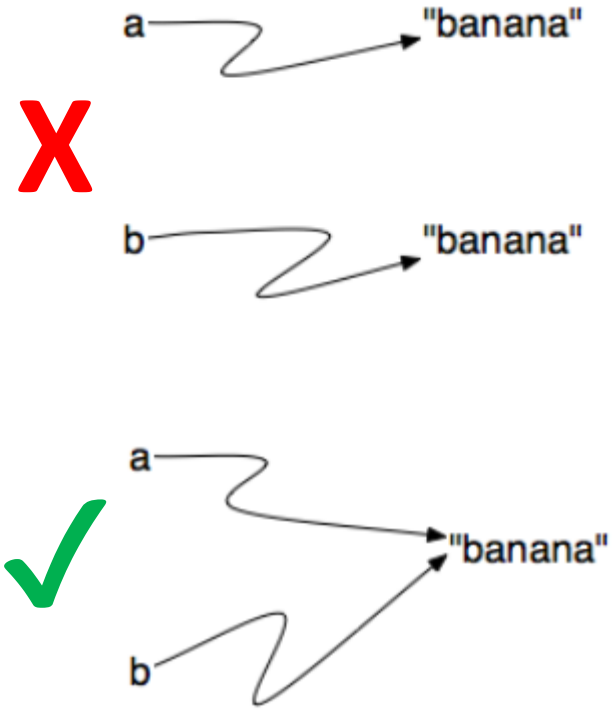
Arrow / Computer memory address

- **Python variables are references**
  - Label that refers to object
  - Label is small, object is big

is operator

- **True→ two references (i.e., variables) are to the same object)**

```
a="banana"
b="banana"
```



NOTE: PythonTutor doesn't always make everything an object(why not always correct)

Aliasing vs. Cloning
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)
```

- **Does print(b) include ‘bat’ or ‘ant’?**
- **How do we keep these two lists separate?**
  - Cloning (slicing)
  - b=a[:]

Note: Reference and Repetition

```
origlist=[45,76,34]
print(origlist*3)
```

```
[45, 76, 34, 45, 76, 34, 45, 76, 34]
```

```
origlist=[45,76,34]
newlist=origlist*3
print(newlist)
newlist = [origlist] * 3
print(newlist)
```

```
[45, 76, 34, 45, 76, 34, 45, 76, 34]
[[45, 76, 34], [45, 76, 34], [45, 76, 34]]
```

Reminders

- **Work smarter, not harder**
- **Design first**
- **Try to identify where you are stuck**
  - Identify resources to help solve problem
- **Leverage your design and PythonTutor to understand program flow of control**
  - http://pythontutor.com