CSC 111:
Intro to Computer Science through Programming

Spring 2017
Prof. Sara Mathieson

Admin

- Homework 4 due Tuesday
- Office hours Monday 3-5pm in Ford 355 or across the hall
- Midterm: Tues-Fri right before Spring Break
  - Take in Neilson Library
  - 2 hours
  - Pencil and paper exercises
  - You may bring a cheat sheet (front and back)
  - No technology or books

Outline: 2/24

WEEK 4: all about FUNCTIONS

- Recap last time
- More about mutable vs. immutable types
- Function and debugging practice
- (if time) Go over Lab 3

Recap

Informal quiz (discuss with a partner)

1) Describe the different **goals** of these two programs.
2) Why is **return** used in only one of these programs?
3) How does this relate to the difference between **ints** and **lists**?
4) Where is the concept of **inside out** design used here?

```
# INTEGER version
def add_interest(balance, rate):
    new_balance = balance * (1+rate)
    return new_balance

def main():
    amount = 100
    rate = 0.05

    amount = add_interest(amount, rate)
    print("New amount is", amount)

main()
```

A

```
# LIST version
def add_interest(balances, rate):

    for i in range(len(balances)):
        balances[i] = balances[i] * (1+rate)

def main():

    amounts = [1000, 2200, 800, 360]
    rate = 0.05

    add_interest(amounts, rate)
    print(amounts)

main()
```

B

Informal quiz (discuss with a partner)

1) Describe the different **goals** of these two programs.

   The goal of program A is to modify one balance, and program B should modify a list of balances.

2) Why is **return** used in only one of these programs?

3) How does this relate to the difference between **ints** and **lists**?

4) Where is the concept of **inside out** design used here?

Informal quiz (discuss with a partner)

1) Describe the different **goals** of these two programs.

   The goal of program A is to modify one balance, and program B should modify a list of balances.

2) Why is **return** used in only one of these programs?

   We cannot modify **amount** by modifying the placeholder variable and formal parameter **balance**, so we must return the computed value and reassign **amount**. Lists are mutable so modifying **balances** will modify **amounts** (they point to the same structure in memory.

3) How does this relate to the difference between **ints** and **lists**?

4) Where is the concept of **inside out** design used here?

Informal quiz (discuss with a partner)

1) Describe the different **goals** of these two programs.

   The goal of program A is to modify one balance, and program B should modify a list of balances.

2) Why is **return** used in only one of these programs?

   We cannot modify **amount** by modifying the placeholder variable and formal parameter **balance**, so we must return the computed value and reassign **amount**. Lists are mutable so modifying **balances** will modify **amounts** (they point to the same structure in memory.

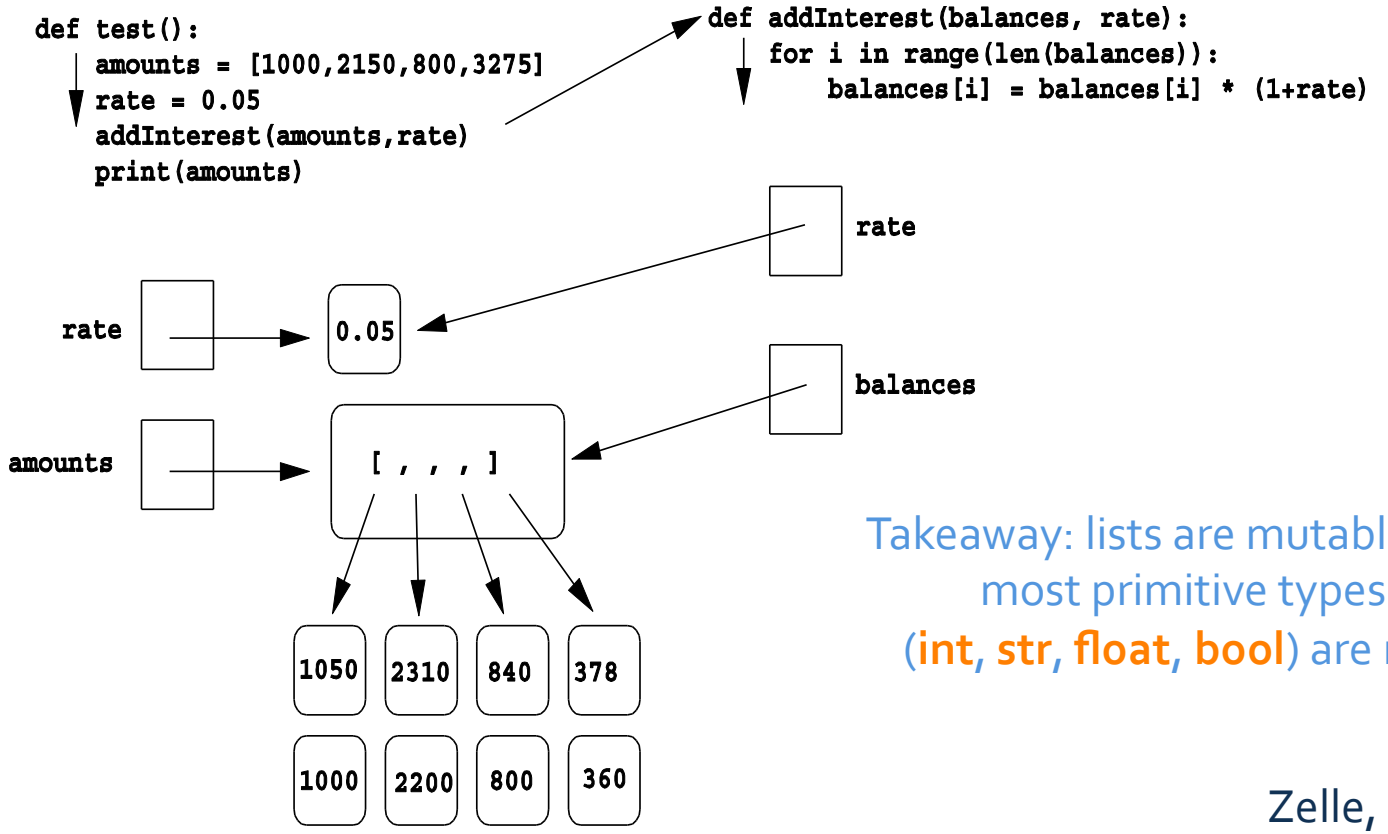
3) How does this relate to the difference between **ints** and **lists**?

   Updating the value of an **int** variable will not change other variables that point to the same **int**. But modifying a **list** will modify all variables that point to that **list**.

4) Where is the concept of **inside out** design used here?

Amounts and balances point to same structure

Balances and **amounts** still point to the same structure,
but the elements have been changed.



Takeaway: lists are mutable, but most primitive types (**int**, **str**, **float**, **bool**) are not.

Zelle, Figure 6.9

Informal quiz (discuss with a partner)

Where is the concept of **inside out** design used here?

```
# INTEGER version
def add_interest(balance, rate):
    new_balance = balance * (1+rate)
    return new_balance

def main():
    amount = 100
    rate = 0.05

    amount = add_interest(amount, rate)
    print("New amount is", amount)

main()
```

A

```
# LIST version
def add_interest(balances, rate):

    for i in range(len(balances)):
        balances[i] = balances[i] * (1+rate)

def main():

    amounts = [1000, 2200, 800, 360]
    rate = 0.05

    add_interest(amounts, rate)
    print(amounts)

main()
```

B

Mutable vs. Immutable Types

Quick demo of lists and strings

- Lists are mutable

```
>>> list1 = [8,9,10]
>>> list2 = list1
>>> list2
[8, 9, 10]
>>> list2[0] = 13
>>> list2
[13, 9, 10]
>>> list1
[13, 9, 10]
>>> # AH!
```

- Strings are not (they are immutable)

```
>>> string1 = "Happy weekend"
>>> string2 = string1
>>> string2
'Happy weekend'
>>> string2[0] = "S"
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    string2[0] = "S"
TypeError: 'str' object does not support item assignment
>>> # cannot change a string!
```

More function practice

Goal: complete the functions below to create a range method

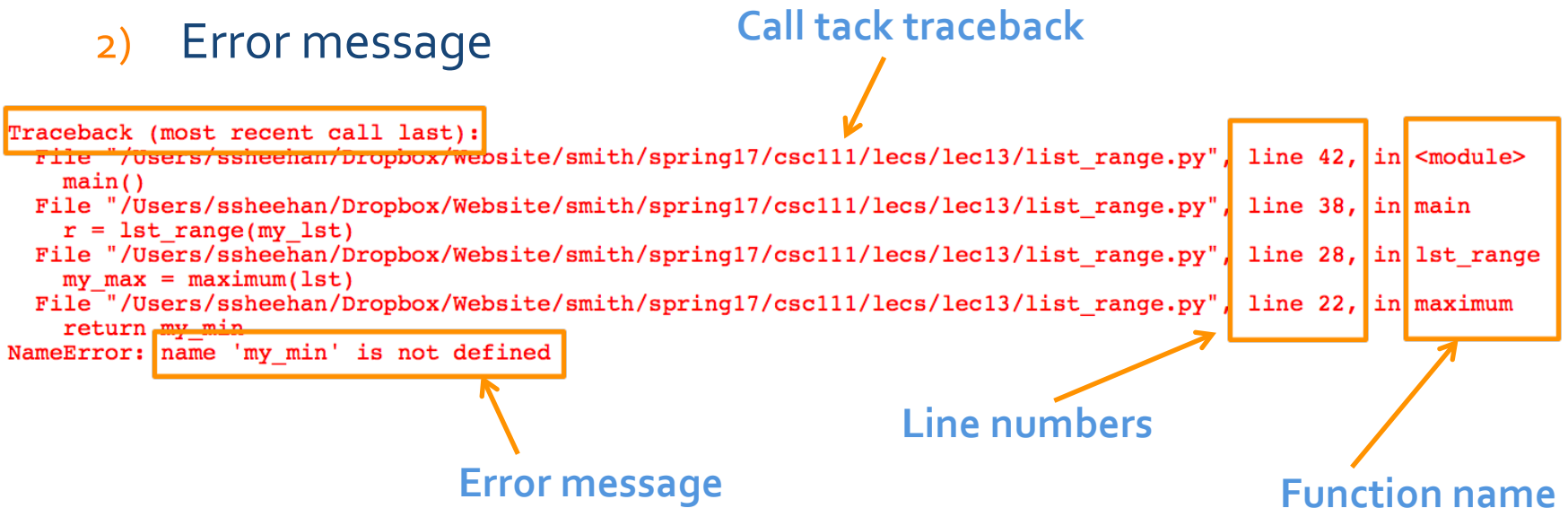
Work with a partner!

```
# return the minimum element of the list
def minimum(lst):


# return the maximum element of the list
def maximum(lst):


# return the range of the list
def lst_range(lst):


def main():

    my_lst = [17, 10, 1, 12, 5, 18, 15, 16, 6, 14]

    r = lst_range(my_lst)

    print("Range is", r)

main()
```

Debugging demo: 3 types of bugs

1) Syntax error

2) Error message



3) No errors but your program is not doing what you want

START PRINTING!

Go over Lab 3

Lab 3, Part D Solution

```
# CSC 111, Lab 3, Part D Solution
# Author: Sara Mathieson
# A program to generate random strings of DNA

import random

def main():

    bases = ["A","C","G","T"]

    num_genes = eval(input("Enter the number of genes: "))
    num_bases = eval(input("Enter the number of bases in each gene: "))
    print()

    for i in range(num_genes):
        gene = ""
        for j in range(num_bases):
            b = random.randint(0,len(bases)-1)
            gene = gene + bases[b]
        print("gene" + str(i) + ": " + gene)

main()
```

Lab 3, Part E Solution

```
# CSC 111, Lab 3, Part E Solution
# Author: Sara Mathieson
# A program compare two DNA sequences

import random

def main():

    human = "ATA?CAAGACCTCGTTATTAATACGGCGCCATGTGAGTAATCCTATC?GA"
    chimp = "ATAACAAGAGCTAGTTATTA?TACTGCGCCATGTGAGAAATCCTATAGGA"

    n = len(human)
    diff = 0
    same = 0
    unknown = 0

    for i in range(n):
        if human[i] == "?" or chimp[i] == "?":
            unknown = unknown + 1
        elif human[i] != chimp[i]:
            diff = diff + 1
        else:
            same = same + 1

    print(same, "bases are the same.")
    print(diff, "bases are different.")
    print(unknown, "bases: either human or chimp or both unknown.")

main()
```