• Reminder: Working on Lab 6 this afternoon
• Intermission
• [link](http://www.thedailyshow.com/full-episodes/thu-october-11-2012-paul-thomas-anderson)
• 6:30-9:00
def change(b):
    print("In change1: ", b)
    b = b + 1
    print("In change2: ", b)

def main():
    a = 2
    print("In main 1: ", a)
    change(a)
    print("In main 2: ", a)

main()
Functions that Modify Parameters

• Return values are the main way to send information from a function back to the caller
• Can communicate back to the caller by making changes to the function parameters
Bank Account Management

• One function: accumulate interest on account

```python
def addInterest(balance, rate):
    newBalance = balance * (1 + rate)
    balance = newBalance

def test():
    amount = 1000
    rate = 0.05
    addInterest(amount, rate)
    print(amount)
```
Functions that Modify Parameters

• We hope that that the 5% will be added to the amount, returning 1050.

```python
>>> test()
1000
```
def addInterest(balance, rate):
    newBalance = balance * (1 + rate)
    balance = newBalance

def test():
    amount = 1000
    rate = 0.05
    addInterest(amount, rate)
    print(amount)
def addInterest(balance, rate):
    newBalance = balance * (1 + rate)
    balance = newBalance

def test():
    amount = 1000
    rate = 0.05
    addInterest(amount, rate)
    print(amount)
Functions that Modify Parameters

• The formal parameters of a function only receive the values of the actual parameters
• Function does not have access to variable that holds the actual parameter

• Python is said to pass all parameters by value
Other Languages

• Some programming languages (C++ & many more) allow variables themselves to be sent as parameters to a function
• This mechanism is said to pass parameters by reference
• When new value is assigned to the formal parameter, the value of the variable in the calling program actually changes
Option 1: return

def addInterest(balance, rate):
    newBalance = balance * (1 + rate)
    return newBalance

def test():
    amount = 1000
    rate = 0.05
    amount = addInterest(amount, rate)
    print(amount)

test()
• First, suppose we are writing a program for a bank that deals with many accounts

• Could store the account balances in a list, then add accrued interest to each balance in list

• Update the first balance in the list with code like:

  \[ \text{balances}[0] = \text{balances}[0] \times (1 + \text{rate}) \]
balances[0] = balances[0] * (1 + rate)

• “Multiply value in the 0\textsuperscript{th} position of list by (1 + rate) and store result back into 0\textsuperscript{th} position of the list”

• More general: loop over 0, 1, …, length – 1

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

def test():
    amounts = [1000, 2200, 800, 360]
    rate = 0.05
    addInterest(amounts, rate)
    print(amounts)

test()
Functions that Modify Parameters

• Original code had these values:
  [1000, 2200, 800, 360]

• Program returns:
  [1050.0, 2310.0, 840.0, 378.0]

• Python passes parameters by value, but it looks like amounts has been changed
Functions that Modify Parameters

• First two lines of main create the variables amounts and rate

```python
def addInterest(balances, rate):
    for i in range(len(balances)):
        balances[i] = balances[i] * (1 + rate)

def test():
    amounts = [1000, 2200, 800, 360]
    rate = 0.05
    addInterest(amounts, 0.05)
    print(amounts)
```

• Value of the variable amounts is a list object that contains four int values
Functions that Modify Parameters

```python
def test():
    amounts = [1000, 2150, 800, 3275]
    rate = 0.05
    addInterest(amounts, rate)
    print(amounts)

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

rate → 0.05

amounts → [1000, 2150, 800, 3275]
Functions that Modify Parameters

- **Next**, `addInterest` executes

- Loop goes through each index in the range 0, 1, ..., length –1 and updates that value in `balances`

```python
def addInterest(balances, rate):
    for i in range(len(balances)):
        balances[i] = balances[i] * (1 + rate)

def test():
    amounts = [1000, 2200, 800, 360]
    rate = 0.05
    addInterest(amounts, 0.05)
    print(amounts)
```
Functions that Modify Parameters

```python
def test():
    amounts = [1000, 2150, 800, 3275]
    rate = 0.05
    addInterest(amounts, rate)
    print(amounts)

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

Diagram:
- `rate`: 0.05
- `amounts`: [1050, 2310, 840, 378]
- `balances`: [1000, 2200, 800, 360]
Functions that Modify Parameters

• In diagram, old values are left hanging around.

```python
def addInterest(balances, rate):
    for i in range(len(balances)):
        balances[i] = balances[i] * (1 + rate)
```

• Numbers in the boxes have not changed, but the new values were created and assigned into list

```python
def test():
    amounts = [1000, 2200, 800, 360]
    rate = 0.05
    addInterest(amounts, 0.05)
    print(amounts)
```

• Old values will be destroyed during garbage collection
Functions that Modify Parameters

- When `addInterest` terminates, list stored in `amounts` now contains new values
- `amounts` wasn’t changed (it’s still a list), but the state of that list has changed
- This change is visible to calling program
Functions that Modify Parameters

• Parameters are always passed by value.

• If the value of the variable is a mutable object (like a list or graphics object), then changes to the state of the object will be visible to the calling program.

• Another example of the aliasing issue
Functions and Program Structure

def main():
    # Introduction
    print("This program plots the growth of a 10 year investment.")

    # Get principal and interest rate
    principal = eval(input("Enter the initial principal: "))
    apr = eval(input("Enter the annualized interest rate: "))

    # Create a graphics window with labels on left edge
    win = GraphWin("Investment Growth Chart", 320, 240)
    win.setBackground("white")
    win.setCoords(-1.75,-200, 11.5, 10400)
    Text(Point(-1, 0), ' 0.0K').draw(win)
    Text(Point(-1, 2500), ' 2.5K').draw(win)
    Text(Point(-1, 5000), ' 5.0K').draw(win)
    Text(Point(-1, 7500), ' 7.5k').draw(win)
    Text(Point(-1, 10000), '10.0K').draw(win)

    # Draw bar for initial principal
    drawBar(win, 0, principal)

    # Draw a bar for each subsequent year
    for year in range(1, 11):
        principal = principal * (1 + apr)
        drawBar(win, year, principal)

    input("Press <Enter> to quit.")
    win.close()
Functions and Program Structure

```python
def createLabeledWindow():
    window = GraphWin("Investment Growth Chart", 320, 240)
    window.setBackground("white")
    window.setCoords(-1.75, -200, 11.5, 10400)
    Text(Point(-1, 0), ' 0.0K').draw(window)
    Text(Point(-1, 2500), ' 2.5K').draw(window)
    Text(Point(-1, 5000), ' 5.0K').draw(window)
    Text(Point(-1, 7500), ' 7.5K').draw(window)
    Text(Point(-1, 10000), '10.0K').draw(window)
    return window

def main():
    print("This program plots the growth of a 10 year investment.")
    principal = eval(input("Enter the initial principal: "))
    apr = eval(input("Enter the annualized interest rate: "))
    win = createLabeledWindow()
    drawBar(win, 0, principal)
    for year in range(1, 11):
        principal = principal * (1 + apr)
        drawBar(win, year, principal)
    input("Press <Enter> to quit.")
    win.close()
```

Python Programming, 2/e 24