Top Down Design

Methodology for writing larger programs

Step 1: Divide problem into smaller, easy-to-solve subproblems

- Compile list of features
- Sketch high-level algorithm on paper
- Sketch program (in code) using **stubs** (e.g placeholder functions)

Step 2: Implement program bottom-up

- Use incremental development
- Refine design in step 1 and re-stub as necessary

Goals of TDD

Good design makes your program

- easy to build and test incrementally
- easy to debug
- well-organized
- easy for a human to read
- resistant to bugs

Rules of thumb:

NO CUT AND PASTED CODE BLOCKS!

Functions should do a single, clearly defined task

Algorithms should be clear from function and variable names

Top-down design - Analogies

Approach is the same as any you would take with a large project:

- Applying to schools
- Organizing an event
- Building a piece of furniture
- Writing a paper

TDD Example - checkbook

Step 1: List features

- Keep track of current balance
- Allow user to make deposits
- Allow user to make withdrawals
- Prevent user from withdrawing more than they have
- Print summary with current balance
- Press ‘q’ to quit

TDD Example - checkbook

Step 2: Sketch high-level algorithm on paper

Goal: subdivide program into small, easy steps

```
Get the starting balance from the user
while not timeToQuit:
    Ask the user what they want to do (withdraw,deposit,quit)
    if withdraw:
        withdraw
    elif deposit:
        deposit
    elif quit:
        timeToQuit = True
    else:
        Report an unrecognized command
```

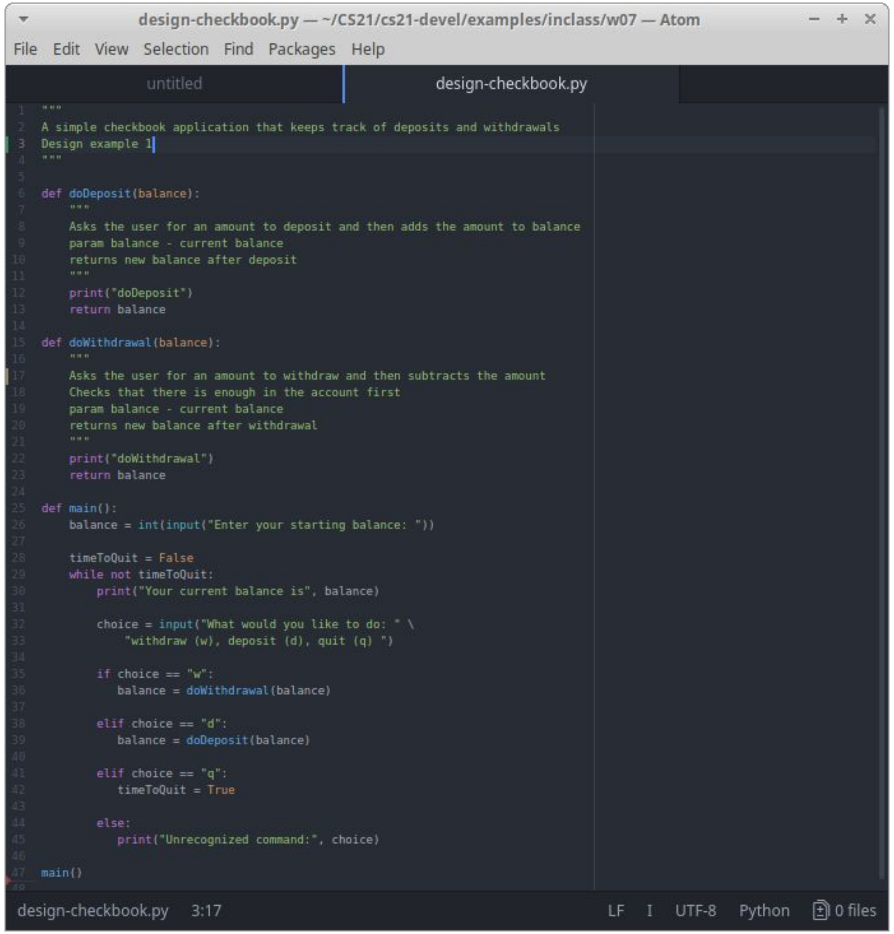
This is a small example, so we have only two functions we need to stub: withdraw() and deposit()

TDD Example - checkbook

Step 3: Sketch program using stubs

NOTE: There are multiple good potential designs (but watch out because also many bad designs that will make your life miserable!)

NOTE: This program runs!

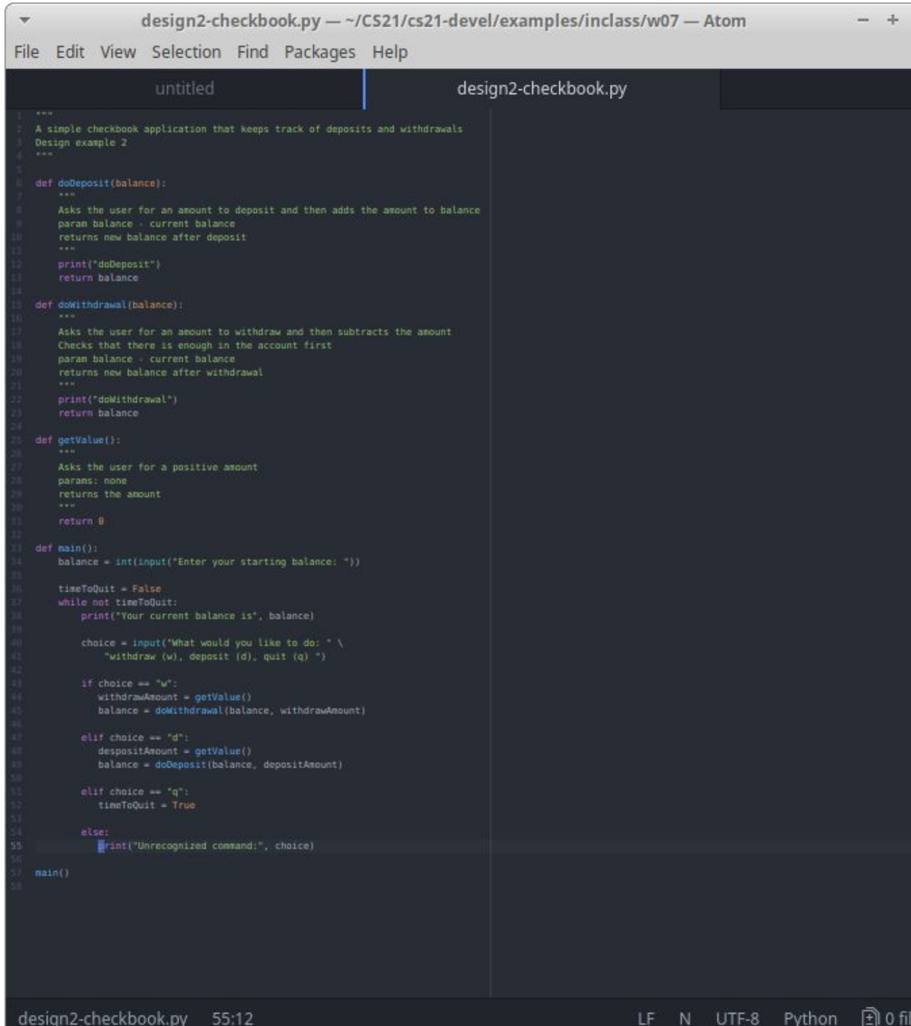

design-checkbook.py — ~/CS21/cs21-devel/examples/inclass/w07 — Atom
File Edit View Selection Find Packages Help
untitled
design-checkbook.py
design-checkbook.py 3:17
LF UTF-8 Python 0 files

```python
"""
A simple checkbook application that keeps track of deposits and withdrawals
Design example 1
"""

def doDeposit(balance):
    """
    Asks the user for an amount to deposit and then adds the amount to balance
    param balance - current balance
    returns new balance after deposit
    """
    print("doDeposit")
    return balance

def doWithdrawal(balance):
    """
    Asks the user for an amount to withdraw and then subtracts the amount
    Checks that there is enough in the account first
    param balance - current balance
    returns new balance after withdrawal
    """
    print("doWithdrawal")
    return balance

def main():
    balance = int(input("Enter your starting balance: "))

    timeToQuit = False
    while not timeToQuit:
        print("Your current balance is", balance)

        choice = input("What would you like to do: " \
            "withdraw (w), deposit (d), quit (q) ")

        if choice == "w":
           balance = doWithdrawal(balance)

        elif choice == "d":
           balance = doDeposit(balance)

        elif choice == "q":
           timeToQuit = True

        else:
           print("Unrecognized command:", choice)

main()
```

TDD Example - checkbook

Step 3: Sketch program using stubs

> NOTE: There are multiple good potential designs (but watch out because also many bad designs that will make your life miserable!)
>
> NOTE: This program runs!


design2-checkbook.py — ~/CS21/cs21-devel/examples/inclass/w07 — Atom
File Edit View Selection Find Packages Help
untitled
design2-checkbook.py
design2-checkbook.py 55:12
LF N UTF-8 Python

```python
"""
A simple checkbook application that keeps track of deposits and withdrawals
Design example 2
"""

def doDeposit(balance):
    """
    Asks the user for an amount to deposit and then adds the amount to balance
    param balance - current balance
    returns new balance after deposit
    """
    print("doDeposit")
    return balance

def doWithdrawal(balance):
    """
    Asks the user for an amount to withdraw and then subtracts the amount
    Checks that there is enough in the account first
    param balance - current balance
    returns new balance after withdrawal
    """
    print("doWithdrawal")
    return balance

def getValue():
    """
    Asks the user for a positive amount
    params: none
    returns the amount
    """
    return 0

def main():
    balance = int(input("Enter your starting balance: "))

    timeToQuit = False
    while not timeToQuit:
        print("Your current balance is", balance)

        choice = input("What would you like to do: " \
            "withdraw (w), deposit (d), quit (q) ")

        if choice == "w":
           withdrawAmount = getValue()
           balance = doWithdrawal(balance, withdrawAmount)

        elif choice == "d":
           despositAmount = getValue()
           balance = doDeposit(balance, depositAmount)

        elif choice == "q":
           timeToQuit = True

        else:
           print("Unrecognized command:", choice)

main()
```

Code stubs - best practices

Your program with stubs should still **run**

Your stubs should have **comments** describing their function

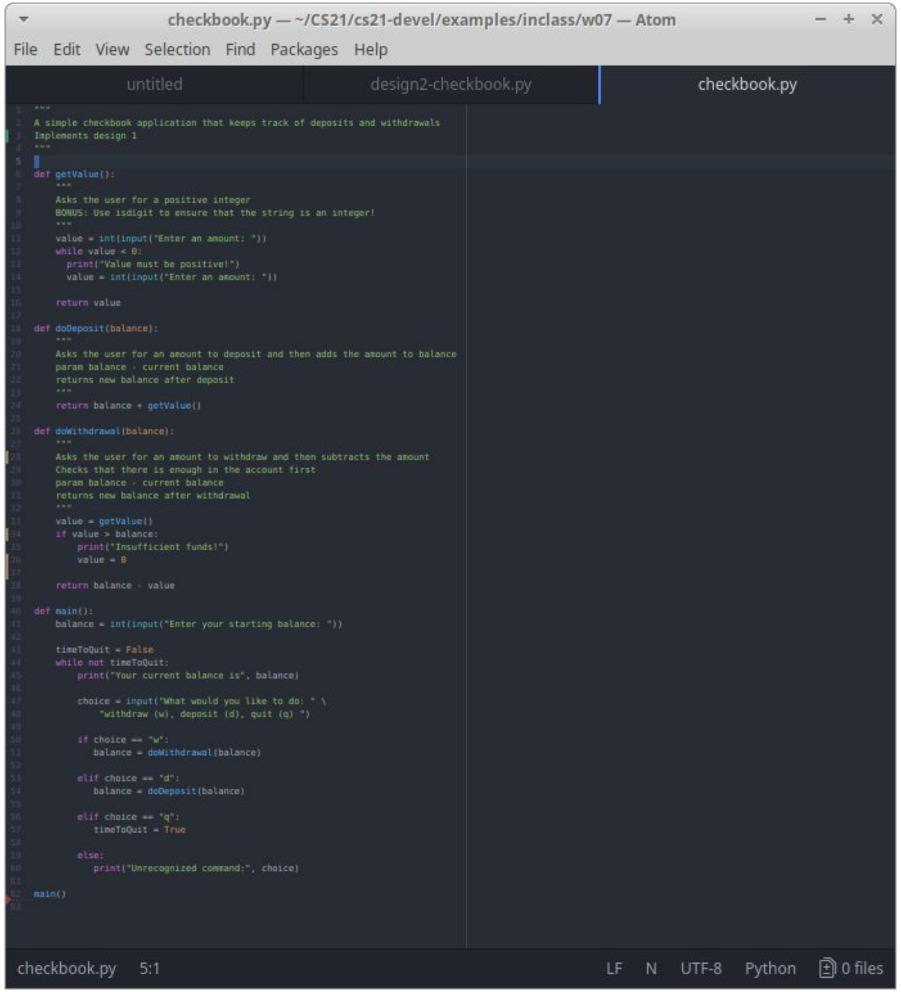
Your stubs should have the **arguments** and **return type** that you expect it to use

All the stubs you define should be used somewhere in your program

TDD Example - checkbook

Bottom-up Implementation

Implement and test each stub one at a time! In class, we started with deposit and then implemented withdrawal


checkbook.py — ~/CS21/cs21-devel/examples/inclass/w07 — Atom
File Edit View Selection Find Packages Help
untitled
design2-checkbook.py
checkbook.py
checkbook.py 5:1
LF N UTF-8 Python 0 files

```python
"""
A simple checkbook application that keeps track of deposits and withdrawals
Implements design 1
"""

def getValue():
    """
    Asks the user for a positive integer
    BONUS: Use isdigit to ensure that the string is an integer!
    """
    value = int(input("Enter an amount: "))
    while value < 0:
      print("Value must be positive!")
      value = int(input("Enter an amount: "))

    return value

def doDeposit(balance):
    """
    Asks the user for an amount to deposit and then adds the amount to balance
    param balance - current balance
    returns new balance after deposit
    """
    return balance + getValue()

def doWithdrawal(balance):
    """
    Asks the user for an amount to withdraw and then subtracts the amount
    Checks that there is enough in the account first
    param balance - current balance
    returns new balance after withdrawal
    """
    value = getValue()
    if value > balance:
        print("Insufficient funds!")
        value = 0

    return balance - value

def main():
    balance = int(input("Enter your starting balance: "))

    timeToQuit = False
    while not timeToQuit:
        print("Your current balance is", balance)

        choice = input("What would you like to do: " \
            "withdraw (w), deposit (d), quit (q) ")

        if choice == "w":
           balance = doWithdrawal(balance)

        elif choice == "d":
           balance = doDeposit(balance)

        elif choice == "q":
           timeToQuit = True

        else:
           print("Unrecognized command:", choice)

main()
```