Top-down Design

Announcements

- Lab 6 due Saturday
- Quiz 3 on Friday

Today's plan

- Go over quiz topics
- Top-down design

- (Design patterns pushed to Friday after quiz)

Quiz 3 topics

- Topics from quiz 1 and 2
- No graphics, methods, or mutation
- `while` loops
  - Infinite loops
  - Condition updated within loop (to avoid infinite loop)
  - More general than `for` loops

Quiz 3 topics

- Functions
  - Defining functions (with parameters)
  - Return values
  - Calling functions (with arguments)
- The stack (without mutation)
  - Drawing diagrams
  - The **scope** of a variable

Practice stack diagram

- Show what stack would look like at position indicated
- Show output from the program (on next slide)

```
def isBigger(n, m):
  print("Control has passed to isBigger")
  if n > m:
    bigger = True
  else:
    bigger = False
  # DRAW STACK AT THIS POINT
  return bigger

def main():
  num1 = 12
  num2 = 15
  result = isBigger(num1, num2)
  if result:
    print("%d is bigger than %d" % (num1, num2))
  else:
    print("%d is bigger than %d" % (num2, num1))

main()
```

Circle click correction

- We should have considered the radius when calculating which circle was nearest.

Top-down design

- Write `main()` function before you do anything else, creating a “wish list” of functions that would make the job of writing `main()` easier.

- Think about functions in terms of inputs and outputs, don’t worry about implementation details.

- `main()` needs to orchestrate the program’s data flow—initializing data, saving return values, passing arguments, etc.

Top-down design example

```
def main():
    width = 600
    height = 400
    window = createWindow(width, height)
    text = drawText(window, "Click for sun center")
    sun = drawSun(window)
    text.setText("Click for horizon")
    drawHorizon(window)
    text.setText("Click twice for beach")
    drawBeach(window)
    text.setText("Click to animate")
    animateSun(window, sun)
    text.setText("Click to quit")
    window.getMouse()
```

`main()` manages

- `main()` is like a manager, asking employees to prepare reports, perform tasks.

- A good manager gives employees the information they need to do their jobs, and explains what information they should report back when the job is done. Similarly, `main()` gives functions the input data they need to perform a task, and sets up an expectation for what the function should return.

- Like a good manager, `main()` doesn't micro-manage, letting functions decide how they want to accomplish the requested task.

- This hierarchy can have many levels, i.e. you can write functions that call other functions and use top-down design again.

Top-down design + Incremental Development

- Once you've written `main()`, create a **function stub** for each of the function calls that appear in this `main()`.

- These functions shouldn't be fully implemented, but they should allow the still-incomplete program to ***run without errors***.

- That way, you can incrementally implement one function at a time, testing it before moving on to the next function.

Revisiting rps