Applied Math DUG!

The Applied Math DUG is kicking off for the 2021-2022 school year! Please fill out the form here to join our listserv, and be notified of upcoming events this semester. We encourage first-years and any students who are considering Applied Math or related joint concentrations or who just have an interest in Applied Math subjects to join. We are looking forward to meeting you all!

Lecture 12
Loops

“Life is just one damn thing after another.”
- Mark Twain

“Life isn’t just one damn thing after another… it’s the same damn thing over and over and over again.”
- Edna St. Vincent Millay

Outline

- Turtle
- Looping
- while Loops
- for Loops
- Choosing the Right Loops
Introduction to Turtle (1/2)

• Before we see loops, we need some tools
  - We will use a Turtle to help us understand loops
  - Turtles are based on Seymour Papert's Logo, a language for beginners
• Turtles are imaginary pens that when given instructions can draw shapes for us


Introduction to Turtle (2/2)

• Turtles know where they are and what direction they are facing and can move and turn
• Turtles can draw lines behind them as they move around the screen or just move without drawing
• PaneOrganizer holds instructions for the turtle
  - reminiscent of our first Robot example...

Turtle's Methods (1 of 2)

public class Turtle {
  // instance variables elided
  /* constructor for Turtle instantiates a Polygon representing the turtle graphically */
  public Turtle() {
    // some code here
  }
  /* reset turtle to center of pane */
  public void home() {
    // some code here
  }
  /* turn right a specified number of degrees */
  public void right(double degrees) {
    // some code here
  }
  /* turn left a specified number of degrees */
  public void left(double degrees) {
    // some code here
  }
  // continued
Turtle's Methods (2 of 2)

```java
public class PaneOrganizer {
    private Turtle turtle;
    private Pane root;
    public PaneOrganizer() {
        this.root = new Pane();
        this.turtle = new Turtle();
        this.root.getChildren().add(this.turtle.getNode());
    }
    public Pane getRoot() {
        return this.root;
    }
    // methods for each geometric pattern to follow...
}
```

getNode() just returns the triangle contained in Turtle class so it can be added to the Scene Graph.

Note: Because this is a very small program, our logic is also in our PaneOrganizer rather than in a top-level class like we do in CS15 projects.

Drawing with Turtle (2/2)

- How will we code it?
  - create PaneOrganizer class which defines methods for drawing each shape
  - PaneOrganizer also instantiates the root Pane that the Turtle will draw on and contains the Turtle. The root is returned in getRoot().
  - Turtle is a wrapper class that contains a polygon (a triangle) and defines methods for how the Turtle will move; it can also return its polygon as a node via getNode().

Drawing with Turtle (1/2)

- Need class to tell Turtle how to draw some basic shapes
  - will contain a Pane and a Turtle
  - will have methods for each shape we want to draw
- First, determine what shapes we want
  - this lecture: square, random walk
A Repetitive Solution (1/2)

- Let's write `drawSquare` method in the `PaneOrganizer` class
- Brute force: write line of code for each side of the square

```java
public void drawSquare(int sideLen) {
    this.turtle.forward(sideLen);
    this.turtle.right(90);
    this.turtle.forward(sideLen);
    this.turtle.right(90);
    this.turtle.forward(sideLen);
    this.turtle.right(90);
    this.turtle.forward(sideLen);
    this.turtle.right(90);
}
```

A Repetitive Solution (2/2)

- What if we wanted to make a more general method that handles regular shapes such as pentagons or octagons?
  - need to call `forward()` and `right()` for each side
  - cannot fix how many sides we need in generic method
  - note that we're using the `Turtle`'s primitive methods to generate higher-level shapes that are normally already defined in JavaFX
- There must be an easier way!

Outline

- Turtle
- Looping
  - `while` Loops
  - `for` Loops
- Choosing the Right Loops
Looping (1/2)

- Execute a section of code repeatedly
  - uses booleans (true and false) as loop conditions; continues looping as long as condition is true, but when boolean is false, loop condition equals exit condition and loop is terminated
  - as with conditionals, code in loop can be a single line or many lines enclosed in curly braces
  - section of code executed is called loop’s body

Looping (2/2)

- Three loop structures in Java
  - while loop
  - do while loop
  - for loop
- Differ in relation between body and loop condition, as well as length of execution
- Let’s look at while loop first

Outline

- Turtle
- Looping
- while Loops
- for Loops
- Choosing the Right Loops
The **while** loop (1/2)

- **Executes** **while** stated condition is true
  - **tests loop condition before** executing body
  - if loop condition is **false** first time through, body is not executed at all

```java
while (<loop condition>) {
    <loop body>
}
```

16

The **while** loop (2/2)

- **Examples of loop conditions:**
  ```java
  numClasses < 6
  peopleStanding <= maxPeople
  this.checkAmount() <= acctBalance
  this.isSquare() //predicate, a method that returns a boolean
  ```
- **Follows the same rules as conditions for if-else statements**
- **Multiple conditions can be combined using logical operators** (`and (&&)`, `or (||)`, `not (!)`)

```java
(numClasses >= 3) && (numClasses <= 5) 
(peopleStanding <= maxPeople) || (maxPeople < 50)
```

17

**while** loop Flowchart (1/2)

- **while** loops continue while the loop condition is **true**
- **<loop condition> can be any Boolean expression**

18
**while loop Flowchart (2/2)**

- **while** loops continue while the loop condition is true
- `<loop condition>` can be any Boolean expression

```
while loop Flowchart (2/2)

- while loops continue
  while the loop condition is true
- <loop condition> can be any Boolean expression

Walk into the Ratty.

Is The Ratty open?

- Yes → Get more food.
- No → Go to Jo's.
```

---

**All Flow of Control Structures: 1-in, 1-out**

- Benefits of **predictable** flow of control:
  - much easier debugging
  - compiler can optimize much better
- Different from "spaghetti" code (unorganized and difficult to maintain code) by having a **goto** which allows the program to jump to another line of code (based on h/w’s unconditional and conditional “jump” instructions)
  - Go To Statement Considered Harmful letter by Edsger Dijkstra, CACM, 1968
  - IF-ELSE, etc., are “structured flow-of-control”

**So, just how bad is goto?**

```
So, just how bad is goto?

I could reconstruct the program flow OR use one little goto instead

EX SREDDO PRACTICE:
YOU END ON IT BUT
you can waste

[Source: https://xkcd.com/292/](https://xkcd.com/292/)
```
Syntax: Random Walk Using while

- Method of PaneOrganizer class:
  - draws random lines while this.turtle is within its pane

```java
public void randomWalk() {
  while (this.turtle.getPosition() is inside its pane, move this.turtle randomly
  // this.turtle's initial location set to (0,0)
  while (this.root.contains(this.turtle.getPosition())) {
    this.turtle.forward((int) (Math.random() * 15)); // cast to [0-14]
    this.turtle.right((int) (Math.random() * 360)); // cast to [0-359]
  }
}
```

- On last step of walk, turtle will move forward out of pane
  - the line is clipped by JavaFX since we don’t explicitly tell it to wrap around
  - no point in continuing to walk outside the pane

TopHat Question 1

What is the value of tempSum after this while loop is terminated?

```java
int tempSum = 0;
while (tempSum < 10) {
  tempSum += 3;
}
```

A. 10  
B. 9  
C. 12  
D. The loop will never terminate

The do while Loop

- do while always executes loop body at least once by switching order of test and body
- <loop condition> is Boolean expression
Example: Another Random Walk

- Method of PaneOrganizer class:
  - draws random lines while turtle is within pane
  - this.turtle starts in center of root pane, so first step guaranteed to be within pane

```java
public void centeredRandomWalk() {
    // move turtle to pane’s center
    this.turtle.home();
    // move turtle randomly within pane
do {
        this.turtle.forward((int)(Math.random()*15));
        this.turtle.right((int)(Math.random()*360));
    } while (this.root.contains(this.turtle.getLocation()));
}
```

Note the semicolon at the end of while statement

---

**do while vs. while (1/2)**

- In both loops:
  - stops executing body if loop condition is \textit{false}
  - must make sure loop condition becomes \textit{false} by some computations to avoid an “infinite loop”
  - \textit{infinite loop} means your loop condition will never turn \textit{false} – i.e., exit condition never occurs (and your program “freezes up!”)

---

**do while vs. while (2/2)**

- \textbf{do while}
  - body always executes at least once
  - loop condition tested at bottom of loop body

- \textbf{while}
  - body may not execute at all
  - loop condition tested before body; loop condition variables must be set before loop entry
  - useful for screening bad data that might cause statements within loop to fail (e.g. \texttt{while (ref != null)})
TopHat Question 2
What's the difference between these two loops?

Loop 1:
while(andyIsAway()) {
    this.tas.takeADayOff();
    this.tas.takeADayOff();
} while (andyIsAway());

A. In the second loop, the condition is tested before the body
B. In the second loop, the TAs always take at least 1 day off
C. In the first loop, the body is executed before the condition is tested.
D. There is no difference between the two loops

Outline
• Turtle
• Looping
• while Loops
• for Loops
• Choosing the Right Loops

for loops (1/4)
• Most specialized loop construct (and the first high-level, goto-less loop in FORTRAN): typically used to execute loop body a predetermined number of times
  o while and do while loops can execute body for undetermined number of times; based on boolean
• This is the syntax for a for loop:
  for (<init-expr>; <loop condition>; <update>) {
    <loop body>
  }
for loops (2/4)

for (init-expr; <loop condition>; <update>) {
  <loop body>
}

- init-expr
  - expression for setting initial value of loop counter (traditionally use single char. identifier; e.g., i)
  - executed at start of loop code, only once, not for each time through the loop

for loops (3/4)

for (init-expr; <loop condition>; <update>) {
  <loop body>
}

- loop condition
  - true or false
  - test involves loop counter to determine if loop should execute
  - checked at start of every loop (including the first)

for loops (4/4)

for (init-expr; <loop condition>; <update>) {
  <loop body>
}

- update
  - expression that modifies loop counter
  - run at end of every <loop body>, just before returning to the top of the loop
**drawSquare Revisited**

- Better way of drawing square than explicitly drawing each side:

```java
public void drawSquare(int sideLen) {
    /* start with integer i initialized to 0; 
       execute as long as i < 4; each execution 
       increments i by 1 */
    for (int i = 0; i < 4; i++) {
        this.turtle.forward(sideLen);
        this.turtle.right(90);
    }
}
```

**for Flowchart**

- for loop has four parts
  - initialize value of counter
  - test loop condition
  - loop body
  - update counter

**for Flowchart**

- We can use an example of a student reading books on different floors of the SciLi.

```java
Student student = new Student("Huey");
student.goToSciLi();
for (int floor = 1; floor < 14; floor++) {
    student.readBook();  // read a new book
}
student.goHome();
```

Note: For this example, we use the old SciLi, where every floor had books!
Outline

- Turtle
- Looping
  - while Loops
  - for Loops
  - Choosing the Right Loops

Choosing the Right Loop (1/2)

- **for** loop is called a **definite** loop because you can typically predict how many times it will loop
- **while** and **do while** loops are **indefinite** loops, as you do not know when they will end
- **for** loop is typically used for math-related loops like counting finite sums

Choosing the Right Loop (2/2)

- **while** loop is good for situations where **boolean** condition could turn **false** at any time
- **do while** loop is used in same type of situation as **while** loop, but when code should execute at least once
- When more than one type of loop will solve problem, use the cleanest, simplest one
TopHat Question 3
What is the value of \texttt{sum} at the end of the following loop?
\begin{verbatim}
sum = 0;
for (int \texttt{i} = 0; \texttt{i} <= 10; \texttt{i}+=2) {
    sum++;
}
\end{verbatim}

A. 10  B. 11  C. 5  D. 6
Looping to Make a Filled-in Square

• 3D Printing a Pizza
  https://www.youtube.com/watch?v=IgXqC-YPnpc

Decrementing Counter

● We can count backwards in our loop too
  ○ in fact, we can update however we want
    public void countDownSeconds(){
      /* change counter to decrement, and change the loop condition accordingly */
      for(int i = 5; i > 0; i--){
        System.out.print(i);
      }
    }

  ○ for loops end in one of two ways
    ○ when counter value equals limit (for < or >)
      ○ when counter value "goes past" limit (for <= or >=)
    ○ thus, countDownSeconds() would also print 0 if used i >= 0
      ○ beware of such "off-by-one" errors! → hand simulation really helps!

    Output: 54321

break

● break causes immediate exit from a flow-of-control structure
  (e.g., switch, while, do while, for)

● Example:
  for (int i = 0; i < 10; i++) {
    if (this.cookieJar.getNumberOfCookies() == 0) {
      break; // if there are no cookies left, we should break out of the loop!
    }
    this.eatACookie();
  }

  // execution continues here after loop is done or after break statement is executed

  ● Execution continues with first line of code after structure
  ● There are other ways to do this loop…
**continue**

- When used in `while`, `for`, or `do while` structures, `continue` skips remaining statements in body of that structure and proceeds with next iteration of loop
  - useful if there is list of data that you are looping over and you want to skip processing of data that is somehow "not legal"
- In `while` and `do while` structures, execution continues by evaluating loop-continuation condition
- In `for` structure, execution continues by incrementing counter and then evaluating loop condition

**continue** Example

```java
// We'd like to try on swimsuits that hang on a rack
for (int i = 0; i < 20; i++) {
  if (!rack.isSwimsuitOnHanger(i)) {
    // If there's no swimsuit on the current hanger,
    // skip to the next iteration
    continue;
  }
  // Only do this if there's a swimsuit on the hanger
  this.tryOnSwimsuit(rack.getSwimsuitOnHanger(i)); // Get swimsuit and try it on
}
// more code here
```

**Boolean Predicates and Flags**

- A **Boolean predicate** is a method that returns a `boolean` (e.g., `isLeft()`, `isAvailable()`, `isSwimsuitOnHanger(i)`)
- A **Boolean flag** records the result of a predicate: set and saved in one place, used later in different place
- Example (implementing a `for` loop, using `while`):
```java
boolean isDone = false;
int i = 0;
while (!isDone) {
  i++;
  if (i == 5) { isDone = true; }
}
// Note: Here, the Boolean flag is set within loop, which, though legal, is not practical.
```
TopHat Question 4

In the loop to the right, what is the value of i upon exit?

A. 4  
B. 5  
C. 6  
D. Infinite loop

```
boolean isDone = false;
int i = 0;
while (!isDone){
  i++;
  if(i == 5){
    isDone = true;
  }
}
```

Empty Intervals

- Example scenario: we want to keep a running sum of a sequence of numbers
- What happens if we try to add integers in this loop?

```
public int sum() {
  int tempSum = 0;
  for (int i = 1; i < 1; i++) {
    tempSum += i;
  }
  return tempSum;
}
```

- Answer: body of loop is not executed
- Why?
  - loop condition is false for initial counter value

Correct Example

- What about this loop?

```
/*This method sums all numbers from 1 up to and including 10 */
public int sum() {
  int tempSum = 0;
  for (int i = 1; i <= 10; i++) {
    tempSum += i;
  }
  return tempSum;
}
```

- It will work!
Off-by-one Errors

- These errors occur when loop executes one too many or one too few times
  - example: add even integers from 2 to some `number`, inclusive
    ```
    count = 2;
    result = 0;
    while (count < number) {
        result += count;
        count += 2;
    }
    ```
    - should be:
      ```
      while (count <= number) {
      ...
      }
      ```
  - Produces incorrect result if `number` is assigned an even value. Values from 2 to `number-2` will be added (i.e., `number` is excluded)
  - Now, value of `number` is included in summation

Syntax: Other Loop Errors (1/2)

- Make sure test variables have proper values before loop is entered
  ```
  int product = 0;
  do { product *= 2; } while (product < 100);
  ```
- Make sure tests check proper conditions
  ```
  for (int i = 1; i < 100; i += 2) {
    // do something here
  }
  ```

TopHat Question 5

Given the following code:
```java
int num = 2021;
do {
    num--;
} while (num < 2021);
```
What do you expect will happen?
A. Loop will never end
B. Loop will run 2021 times (until `num` is 0), then end
C. Loop will run only once
Syntax: Other Loop Errors (2/2)

- ALWAYS HAND SIMULATE first, last, and typical cases through a loop to avoid off-by-one or infinite loop errors
  - the first and last cases of a loop's execution are called boundary conditions or edge cases or corner cases
  - hand simulation doesn't just apply to loops – use it for everything! Trust us – it saves debugging time!

Which loop to use?

- You want to stack 17 blocks
- Your job is to stand at the end of the bowling alley and pick up all the pins, one by one, that have been knocked over
- Sleep until your clock reads 7:51AM or later

Announcements

- Collaboration Policy [Phase 2 & Quiz](#)
- Clarification on Debugging Hours
  - Not restricted to terminal-produced errors!
- Cartoon Deadlines
  - Early due Thursday 21/10
  - On-time due Saturday 23/10
  - Late due Monday 25/10
- Doodle Jump Partner Signups [form](#) released today!
  - Due Friday 22/10
  - Lab 5 – GitHub and Debugging this week
General Data Protection Regulation (GDPR)

- Set of privacy regulations in the EU, meant to harmonize laws between member countries
- Applies to any organization operating within the EU (including most large American tech companies)
- Limits on how data can be collected and what is collected
- Strengthen the ‘right to be forgotten’ — process to remove your data from services completely
- Data breaches must be reported to governments within 72 hrs
  - users have right to know when their data has been leaked
  - max fine: 4% of global annual revenue or €20M, whichever is greater
- Approved by European Parliament in April 2016 and came into force in 2018

European commission’s evaluation of GDPR in 2020:

- "the GDPR ‘set an example […] inspiring similar measure elsewhere’"
- "the GDPR has successfully, met its objectives of strengthening the protection of the individual’s right to personal data protection and guaranteeing the free flow of personal data within the EU"

Improvements

- not enough collaboration on governments for investigations
- diverging approaches / fragmentation on enforcement
- many individuals do not know about / exercise their rights
- hard for small and medium businesses to comply
- Tough to apply to new technologies
California Consumer Privacy Act (2020)

- Enforced starting January 1, 2020
- “The Golden State officially has the strongest consumer data protections in the US” (WIRED, 2020)
- Applies to businesses established in California
- New rights
  - businesses must tell consumers when data is collected about them
  - know when data is sold/disclosed and to whom
  - must have way to opt out of the sale of personal data
  - right to access info collected about you
  - right to equal service even if exercising privacy rights
- Estimated $55 billion in initial costs for CCPA compliance
- Industry group, “The Internet Association” claims not enough public debate
- Compared with GDPR, no notification for data breaches, harder to enforce penalties

American Privacy Laws

- “Currently, privacy laws are a cluttered mess of different sectoral rules.” —Thorin Klosowski, The New York Times (2021)
- legislation that is state-specific or specific to sectors (i.e., education data, healthcare data, credit reporting data, etc)
  - In most states, companies can use, share, sell any data they collect about you
  - No standardized notification for data breaches
  - Companies can sell your data to third parties who can further share/sell it without telling you
- Numerous attempts to legalize
  - Specific agencies filing lawsuits: FTC successfully sued Facebook for $5 billion in 2019 for making information public it told users was private
  - Privacy Bill of Rights Act (Sen. Ed Markey), Consumer Online Privacy Rights Act (Sen. Mark Warner), United States Consumer Data Privacy Act (Sen. Roger Wicker), all introduced & referred to committee (Commerce, Science, and Transportation)
  - Senate Commerce Committee started holding hearings about consumer privacy this September

How is tech policy shaped?

- In 117th (current) Congress of 535 members, 3 scientists, 9 engineers, 8 software company executives
- The Internet Association (right) — consortium of industry players that makes tech policy suggestions
- Lobbying + vacuum of knowledge about issues — often is just what is best for industry!
- Consider tech policy!
More reading that may be of interest!

- "WTF is GDPR?" — Natasha Lomas, TechCrunch (2018)
- "What is GDPR?" and "GDPR: What's really changed so far?" — Danny Palmer, ZDNet (2018)
- "15 Unexpected Consequences Of GDPR" — Forbes (2018)
- "California's new privacy law, explained" — Sara Morrison, Vox (2019)
- European Commission's Report on GDPR (2020)
- "Senate Commerce launches long-awaited privacy hearings" — Benjamin Din, Politico (2021)
- "California's new data privacy law the toughest in the US" — Laura Hautala, CNET (2018)
- "Membership of the 117th Congress: A Profile"
- "Europe's Data Law Is Broken, Departing Privacy Chief Warns" — Stephanie Bodoni, Bloomberg (2021)
- "FTC Imposes $5 Billion Penalty and Sweeping New Privacy Restrictions on Facebook" — Federal Trade Commission
- "The Internet Association"
- "Federal data privacy regulation is on the way — That’s a good thing" — Karen Schuler, Industry Association of Privacy Professionals
- "A quick reference for CCPA compliance" — Deloitte
- NOYB.eu (None of Your Business, nonprofit taking on GDPR civil suits)