Uber said to use “sophisticated” software to defraud drivers, passengers

Uber has devised a "clever and sophisticated" scheme in which it manipulates navigation data used to determine "upfront" rider fare prices while secretly short-changing the driver, according to a proposed class-action lawsuit against the ride-hailing app.

When a rider uses Uber's app to hail a ride, the fare the app immediately shows to the passenger is based on a slower and longer route compared to the one displayed to the driver. The software displays a quicker, shorter route for the driver. But the rider pays the higher fee, and the driver's commission is paid from the cheaper, faster route, according to the lawsuit.

Administrivia

- Assignments:
  - Events due Tuesday (4/11)
  - Animal Functions due Wednesday (4/12)

- Make sure to ask for help *before* deadlines
  - All assignments (except Reading Checks) due at 11:59pm
  - If posting code, make the post private; see Piazza Usage Guidelines
  - Check rubrics – incomplete assignments submitted on time still eligible for partial credit

- “Big Ideas” this week: Algorithms
Outline

- Expressions & Operators
- Conditionals
  - if-statement
- Loops
  - while-loop
  - for-loop
Expressions

“An expression is a combination of one or more values, constants, variables, operators, and functions that the programming language interprets and computes to produce another value.”


Expressions are evaluated and resulting value is used

- Assignment: \( x = x + 1; \)
- Assignment: \( x_{\text{pos}} = \min(x_{\text{pos}} + 3, 460); \)
- Argument: \( \text{ellipse}(50+x, 50+y, 50, 50); \)
- Argument: \( \text{mouse}(\text{rowX}+4*\text{sp}, \text{rowY}, \text{rowC}); \)
Operators

- **Built-in “functions” in Processing that use special symbols:**
  - Multiplicative: \( * \) \( \div \) \( \% \) \( \text{modulus} \)
  - Additive: \( + \) \( - \) \( \text{add} \) \( \text{sub} \)
  - Relational: \( < \) \( > \) \( \leq \) \( \geq \) \( \text{less than} \) \( \text{greater than} \) \( \text{less than or equal to} \) \( \text{greater than or equal to} \)
  - Equality: \( == \) \( != \) \( \text{equal to} \) \( \text{not equal to} \)
  - Logical: \( \&\& \) \( || \) \( \text{and} \) \( \text{or} \) \( ! \) \( \text{not} \)

- **Operators can only be used with certain data types and return certain data types**
  - Multiplicative/Additive: give numbers, get number
  - Relational: give numbers, get Boolean
  - Logical: give Boolean, get Boolean
  - Equality: give same type, get Boolean
Operators

- Built-in “functions” in Processing that use special symbols:
  - Multiplicative: $*$, $/$, $\%$
  - Additive: $+$, $-$
  - Relational: $<$, $>$, $\leq$, $\geq$
  - Equality: $==$, $!=$
  - Logical: $\&\&$, $| |$, $!$

- In expressions, use parentheses for evaluation ordering and readability
  - e.g. $x + (y \times z)$ is the same as $x + y \times z$, but easier to read
Modulus Operator: %

- $x \% y$ is read as “$x$ mod $y$” and returns the remainder after $y$ divides $x$
  - For short, we say “mod” instead of modulus
    - $0/3 = 0$ remainder $0$
    - $1/3 = 1$ remainder $1$

Practice:

- $0 \% 3$ is ___
- $1 \% 3$ is ___
- $2 \% 3$ is ___
- $3 \% 3$ is ___
- $4 \% 3$ is ___
- $5 \% 3$ is ___
- $6 \% 3$ is ___
Modulus Operator: %

- \( x \mod y \) is read as “\( x \mod y \)” and returns the remainder after \( y \) divides \( x \)
  - For short, we say “mod” instead of modulus

- Example Uses:
  - Parity: Number \( n \) is even if \( n \mod 2 == 0 \)
  - Leap Year: Year \( year \) is a leap year if \( year \mod 4 == 0 \)
  - Chinese Zodiac: \( year1 \) and \( year2 \) are the same animal if \( year1 \mod 12 == year2 \mod 12 \)
Modulus Example in Processing

- Use mod to “wrap around”
  - Replace min/max function to “connect” edges of drawing canvas

- \( x_{\text{pos}} = \min(x_{\text{pos}} + 3, 460); \)
- \( x_{\text{pos}} = (x_{\text{pos}} + 3) \mod 460; \)
Control Flow

- The order in which instructions are executed

- We typically say that a program is executed in sequence from top to bottom, but that’s not always the case:
  - Function calls and return calls
  - Conditional/branching statements
  - Loops

- Curly braces `{ }` are used to group statements
  - Help parse control flow
  - Remember to use indentation!
Outline

- Expressions & Operators
- **Conditionals**
  - if-statement
- Loops
  - while-loop
  - for-loop
If-Statements

- Sometimes you don’t want to execute every instruction
  - Situationally-dependent

- Conditionals give the programmer the ability to make decisions
  - The next instruction executed depends on a specified condition
    - The condition must evaluate to a boolean (i.e. true or false)
    - Sometimes referred to as “branching”
  - This generally lines up well with natural language intuition
If-Statements

- Basic form:

```java
if (condition) {
    // “then”
    // statements
}
```

- Example conditions:
  - Variable: `if (done == true )`
  - Variable: `if (done )`
  - Expression: `if (x_pos > 460 )`
  - Expression: `if (x_pos > 100 && y_pos > 100 )`
If-Statements

- With `else` clause:

```plaintext
if (condition) {
    // "then"
    // statements
} else {
    // "otherwise"
    // statements
}
```

Diagram:
- Start
- Condition?
- True: "Then" Statements (branch 1)
- False: "Otherwise" Statements (branch 2)
- End
If-Statements

With else if clause:

```java
if (cond1) {
    // “then”
    // statements
} else if (cond2) {
    // “otherwise if”
    // statements
}
```

![Flowchart diagram](attachment:image.png)
If-Statements

- Notice that conditionals *always* go from Start to End
  - Choose one of many *branches*
  - A conditional must have a single `if`, as many `else if` as desired, and at most one `else` \(\hookleftarrow\) “catch all”

- Can nest and combine in interesting ways:
Processing Demo: Drawing Dots

```java
void draw() {
    if (mousePressed) {
        fill(0, 0, 255); // blue if mouse is pressed
    } else {
        fill(255, 0, 0); // red otherwise
    }
    ellipse(mouseX, mouseY, 5, 5); // draw circle
}
```
Outline

- Expressions & Operators
- Conditionals
  - if-statement
- Loops
  - while-loop
  - for-loop
Looping

- Sometimes we want to do the same (or similar) things over and over again
  - Looping saves us time from writing out all of the instructions

- Loops control a sequence of *repetitions*
While-Loop

- Basic form:
  
  ```
  while (condition) {
    // loop
    // body
  }
  ```

- Repeat loop body until condition is **false**
  - Must make sure to update conditional variable(s) in loop body, otherwise you cause an infinite loop

- `draw()` is basically a **while**(true) loop
While-Loop

- More general form:
  ```
  // init cond var(s)
  while(condition) {
    // loop body
    // update var(s)
  }
  ```

- This occurs so commonly that we create a separate syntax for it!
For-Loop

```java
for (init; cond; update) {
    // loop body
}
```

- First runs `init` expression(s)
- Then checks `cond`
- If `true`, runs loop body followed by update statement(s)
For-Loop Example

Without loop:

```python
line(20, 40, 80, 80);
line(80, 40, 140, 80);
line(140, 40, 200, 80);
line(200, 40, 260, 80);
line(260, 40, 320, 80);
line(320, 40, 380, 80);
line(380, 40, 440, 80);
```

With loop:

```python
for (int i = 20; i < 400; i = i + 60) {
    line(i, 40, i + 60, 80);
}
```
Understanding the For-Loop

- **Choice of variable name(s) is not critical**
  - Represent the value(s) that vary between different executions of the loop body
  - Think of as temporary variable(s)

- **Variable scope:** variable $i$ only exists within this loop
Understanding the For-Loop

- Condition evaluated before the loop body and must evaluate to **true** or **false**

  - Reminder:
    - `>` greater than
    - `<` less than
    - `>=` greater than or equal to
    - `<=` less than or equal to
    - `==` equal to
    - `!=` not equal to
Understanding the For-Loop

- Update is an assignment that is executed after the loop body.

- Loop body is enclosed by curly braces {} and should be indented for readability.
Processing Demo: Circles on Canvas Edge

```
size(480, 120);
background(255);
noStroke();
fill(75, 47, 131);

// loop for circles along the top edge
for(int x = 0; x <= width; x = x + 40){
    ellipse(x, 0, 40, 40);
}

// loop for circles along the left edge
for(int y = 0; y <= height; y = y + 40){
    ellipse(0, y, 40, 40);
}
```