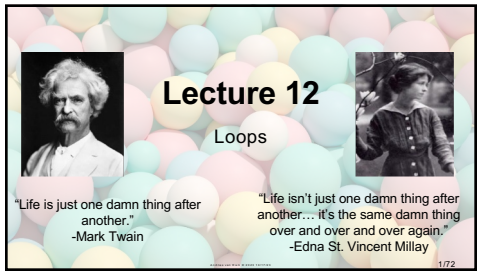
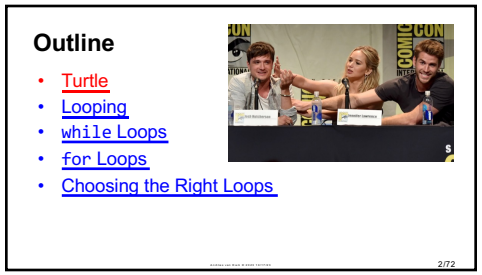
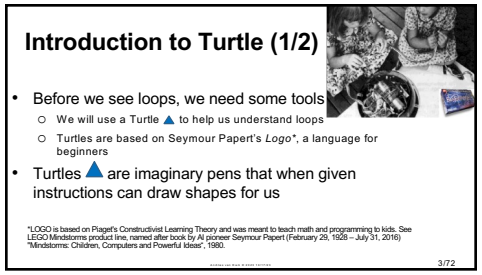

Lecture 12
Loops
"Life is just one damn thing after another."
-Mark Twain
"Life isn't just one damn thing after another… it's the same damn thing over and over and over again."
-Edna St. Vincent Millay
1/72

1


Outline
Turtle
Looping
while Loops
for Loops
Choosing the Right Loops
2/72

2


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
*LOGO is based on Piaget's Constructivist Learning Theory and was meant to teach math and programming to kids. See LEGO Mindstorms product line, named after book by AI pioneer Seymour Papert (February 29, 1928 – July 31, 2016) "Mindstorms: Children, Computers and Powerful Ideas", 1980.
3/72

3

Introduction to Turtle (2/2)

- Turtles know where they are, what direction they are facing, and how to 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…

4/72

4

Turtle's Methods (1 of 2)

TAs have written a `Turtle` class

```
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
```

5/72

5

Turtle's Methods (2 of 2)

```
   /* move forward a specified distance, drawing a line as the turtle
   moves */
   public void forward(int distance) {
      // some code here
   }
   /* move backward a specified distance, drawing a line as the turtle
   moves */
   public void back(int distance) {
      // some code here
   }
   /* move turtle to a specified position without
   drawing a line */
   public void setLocation(Point2D loc) {
      // some code here
   }
   /* return turtle's location */
   public Point2D getLocation() {
      // some code here
   }
   /* return the Polygon (the triangle) contained in Turtle class so
   that we can graphically add it in the P.O.*/
   public Shape getShape() {
      // some code here
   }
}
```

6/72

6

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

7/72

7

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 `getShape()`

```
public class PaneOrganizer {
    // draws each pattern
    private Turtle turtle;

    private Pane root;

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

`getShape()` 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 a top-level logic class like we do in CS15 projects

8/72

8

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

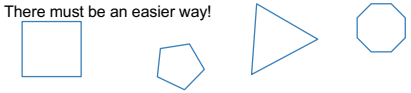
```
public void drawSquare(int sideLen) {
    this.turtle.forward(sideLen);
    this.turtle.right(90);
    this.turtle.forward(sideLen);
    this.turtle.right(90);
    this.turtle.forward(sideLen);
    this.turtle.right(90);
    this.turtle.forward(sideLen);
    this.turtle.right(90);
}
```

9/72

9

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 determine in advance 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!



10/72

10

Outline

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

11/72

11

Looping (1/2)

- Execute a section of code repeatedly
  - uses `boolean`s (`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**

12/72

12

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

13/72

13

Outline

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

14/72

14

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

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

15/72

15

The `while` loop (2/2)

- Examples of loop conditions:

```
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** (**!**))

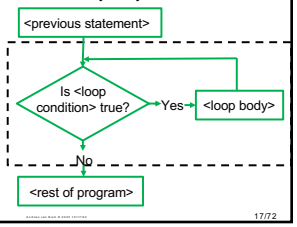
```
(numClasses >= 3) && (numClasses <=5)
(peopleStanding <= maxPeople) || (maxPeople < 50)
```

16/72

16

`while` loop Flowchart (1/2)

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

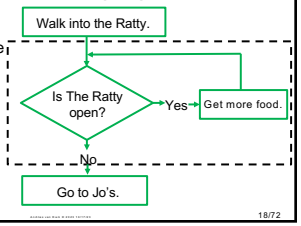


17/72

17

`while` loop Flowchart (2/2)

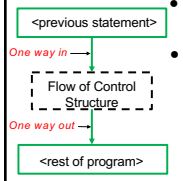
- `while` loops continue **while** the loop condition is `true`
- `<loop condition>` can be any Boolean expression



18/72

18

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



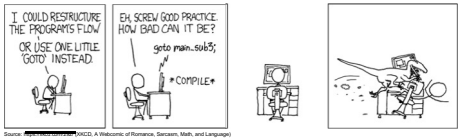
- Benefits of **predictable** flow of control:
  - much easier debugging
  - compiler can optimize much better
- Different from "spaghetti" code (unorganized and difficult to maintain code) with `goto` methods to allow program to jump to another line of code

  - *Go To Statement Considered Harmful* letter by Edsger Dijkstra, CACM, 1968
  - `if-else`, etc., are "structured flow-of-control"

19/72

19

So, just how bad is `goto`?



Source: [redacted] (XKCD, A Webcomic of Romance, Sarcasm, Math, and Language)

20/72

20

Syntax: Random Walk Using `while`

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

```
public void randomWalk() {
// while this.turtle's position is inside its pane, move this.turtle randomly
// this.turtle's initial location set to (0,0)
    while (this.root.contains(this.turtle.getLocation())) {
        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

21/72

21

TopHat Question 1

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

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

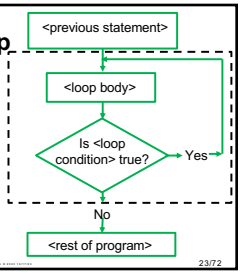
A. 10
B. 9
C. 12
D. The loop will never terminate

22/72

22

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



23/72

23

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

```
public void centeredRandomWalk() {
    // moves turtle to pane's center
    this.turtle.home();

    // moves 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

24/72

24

do while vs. while (1/2)

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

25/72

25

do while vs. while (2/2)

- `do while`
  - body always executes at least once
  - loop condition tested at bottom of loop body
- `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. `while (ref != null)`)

26/72

26

TopHat Question 2

What's the difference between these two loops?

Loop 1:

```
while(andyIsAway()) {
    this.tas.takeADayOff();
}
```

Loop 2:

```
do {
    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

27/72

27

Outline

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

28

`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
  - `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>
}
```


29

`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., `int i = 0`); also called loop index
  - executed at **start** of loop code, only once, not for each time through the loop

30

`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 (e.g. `i < 5`)
  - checked at **start** of every loop (including the first)

31/72

31

`for` loops (4/4)

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

- `<update>`
  - expression that modifies loop counter
  - executed at **end** of every `<loop body>`, just before returning to the top of the loop
  - (e.g. `i++`) this would increase the loop counter by 1 each loop

32/72

32

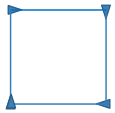
`drawSquare` Revisited

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

```
public void drawSquare(int sideLen) {

    /* start with integer i initialized to 0;
    execute as long as i < 4; each execution
    increments i by 1 at the bottom of the loop */

    for (int i = 0; i < 4; i++) {
        this.turtle.forward(sideLen);
        this.turtle.right(90);
    }
}
```

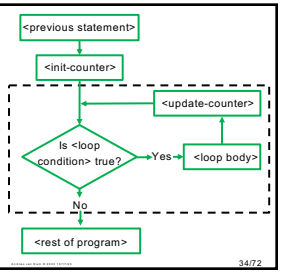


33/72

33

for Flowchart

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



34/72

34

for Flowchart

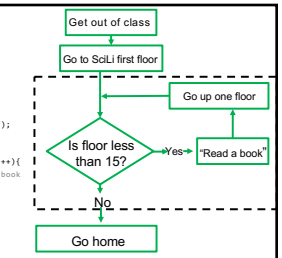
- We can use an example of a student reading books on different floors of the SciLi.

```
Student student = new Student("Sarah");
student.goToSciLi();

for (int floor = 1; floor < 15; floor++){
    student.readBook(); //read a new book
}

student.goHome();
```

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



35/72

35

Outline

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

36/72

36

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 and sequentially looping through elements of an array (Thursday's Lecture)

37/72

37

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.*

38/72

38

TopHat Question 3

What is the value of sum at the end of the following loop?

```
sum = 0;
for (int i = 0; i <= 10; i+=2) {
    sum++;
}
```

A. 10 **B.** 11 **C.** 5 **D.** 6

39/72

39

Syntax: Nested Loops

- Loops, just like `if` statements, can be **nested**!
- Example: `drawFilledSquare`

```
public void drawFilledSquare(int sideLen) {
        // fill in concentric squares
    for (int i = 0; i < (sideLen/2); i++) {
        for (int j = 0; j < 4; j++) {
            this.turtle.forward(sideLen - (2*i));
            this.turtle.right(90);
        }
        /* note we can use loop counter R/O (read-only)
        in body but never reset it there! */
        // position turtle for next iteration
        this.turtle.right(90);
        this.turtle.forward(1);
        this.turtle.left(90);
        this.turtle.forward(1);
    }
}
```

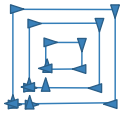
- What does this do?
  - decrementing `sideLen` by 2 each iteration to guarantee that each "inner square" drawn in the inner loop is exactly one unit away on either side from square immediately "outside" of it (hence, one + one = two)

40/72

40

Syntax for Nested Loops Explained

- Turtle is represented by ▲
- What is the outer loop doing?
  - first draws outer square



▲ Turtle starts upright!
► Rotate 90 degrees right!
► Move forward 1 unit!
▲ Rotate 90 degrees left!
▲ Move forward 1 unit!
▲ Draw inner square

`drawFilledSquare` draws concentric squares; each individual square is drawn using the nested loop

Note: Diagram is misleading in that lines should be a pixel unit wide so the filled square will look solid

41/72

41

Looping to Make a Filled-in Design(1/2)

- 3D Printing Food!!

42/72

42

Looping to Make a Filled-in Design (2/2)

- 3D Printed House

43/72

43

Decrementing Counter

- We can count backwards in our loop too
  - just change the counter update expression
  - 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);
    }
}
```

Output:
54321

- `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!*

44/72

44

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…

45/72

45

`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

46/72

46

`continue` Example

```
/* We oversee letting kids on a rollercoaster ride if they are
 tall enough */

for (int i = 0; i < 20; i++) {
    if(!ride.kidIsTallEnough(i)) {
    // if the kid at i is not tall enough
    // skip to the next iteration (the next kid in line)
        continue;
    }
    // only do this if the kid is tall enough
    this.rideRollercoaster(ride.getKid(i)); // let kid onto ride
}
// more code here
```

47/72

47

Boolean Predicates and Flags

- A **Boolean predicate** is a method that returns a `boolean` (e.g., `isLeft()`, `isAvailable()`, `kidIsTallEnough(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`):

```
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.

48/72

48

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;
    }
}
```

49/72

49

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

50/72

50

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!

51/72

51

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;
}
```

Produces incorrect result if `number` is assigned an even value. Values from 2 to `number`-2 will be added (i.e., `number` is excluded)

  - should be:

```
while (count <= number) {
        //loop body elided
}
```

Now, value of `number` is included in summation

52/72

52

Syntax: Other Loop Errors (1/2)

- Make sure test variables have proper values before loop is entered

```
int product = 0;
while (product < 100) {
    product *= 2;
}
/* What will happen here? */
```

- Make sure tests check proper conditions

```
for (int i = 1; i != 100; i += 2) {
    // do something here
}
/* Will we ever get here? */
```

53/72

53

TopHat Question 5

Given the following code:

```
int num = 2023;
do {
    num--;
} while (num < 2023);
```

What do you expect will happen?

A. Loop will never end
B. Loop will run 2023 times (until `num` is 0), then end
C. Loop will run only once

54/72

54

Syntax: Other Loop Errors (2/2)

- **ALWAYS HAND SIMULATE** first, last, and typical cases in 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!

55/72

55

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

56/72

56

Announcements

- Collaboration Policy Phase 2 & Quiz
- Cartoon Deadlines
  - Early due **Thursday 10/19**
  - On-time due **Saturday 10/21**
  - Late due **Monday 10/23**
- Lab 5 – GitHub and Debugging this week
- Don't forget to finish Cartoon Mini-Assignment
- Tshirt form is out now and open until **Thursday 10/19**

57/72

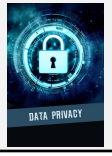
57

58

Topics in Socially Responsible Computing

Privacy and Surveillance I: A Brief History

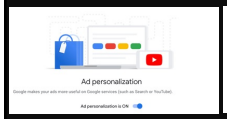
CS15 Fall 2023

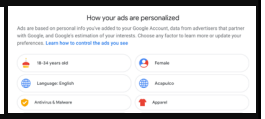


59

Past Week's SRC Lab!

Google Personal Data Insights



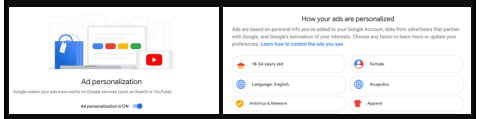


Source: adssettings.google.com

60

Past Week's SRC Lab!

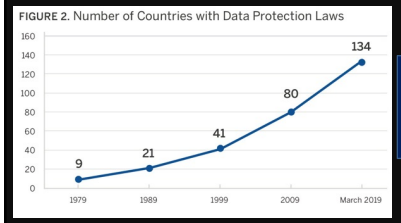
Google Personal Data Insights



Source: adssettings.google.com

61

Privacy: A Hot Topic Today! –

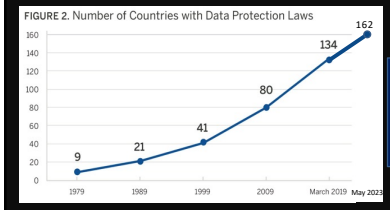


From 2019 Study

Source: Global Data Privacy Laws (Greenleaf 2019)

62

Privacy: A Hot Topic Today! –



From 2023 Update to Study

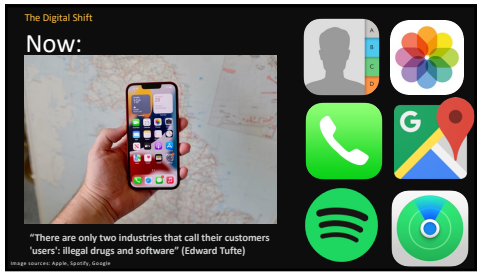
Source: Global Data Privacy Laws (Greenleaf 2023)

Note: Having a data privacy law is not the same as enforcing it properly!

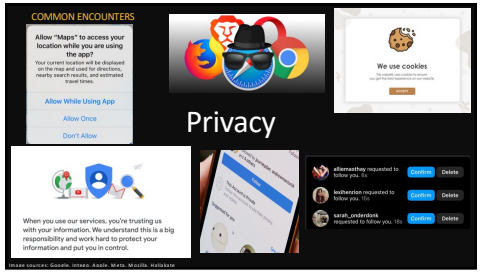
63


A BRIEF HISTORY: Analog Communication
Before...
VHS tapes
Rolodex
Answering Machine
Telephone
Portable CD player/Walkman
FAX machines

64


The Digital Shift
Now:
"There are only two industries that call their customers 'users': illegal drugs and software" (Edward Tufte)

65


COMMON ENCOUNTERS
Allow "Maps" to access your location while you are using the app?
Allow While Using App
Allow Once
Don't Allow
Privacy
We use cookies
When you use our services, you're trusting us with your information. We understand this is a big responsibility and work hard to protect your information and put you in control.

66

A BRIEF HISTORY

Early American Surveillance

J. Edgar Hoover testifying before the House on Un-American Activities Committee (1947)

- FBI mission: "Protect American people and uphold the Constitution of the United States"
- Investigate criminal enterprise and domestic terrorism, subversion of the state etc.

Photo Credit: biography.com

67

A BRIEF HISTORY

CIA: foreign intelligence to policymakers for national security decisions, prohibited from collecting info on 'US Persons'

1908 — 1947 — 1952

FBI: domestic cases, investigate crimes, work w/ Dept of Justice

NSA: counterintellience against adversaries and foreign intelligence

Photo credits: Wikimedia Commons

68

1978: Foreign Intelligence Surveillance Act

- US Federal Law: physical + electronic surveillance
- FISC: oversee requests for surveillance warrants by federal law enforcement + intelligence agencies

2007: Protect America Act

- Electronic communications monitored by NSA (previously restricted to non-Americans)
- No more 'absolute' privacy
- Without court order/ oversight

Image credits: Wikimedia commons, Tech Yahoo

69

A BRIEF HISTORY: Whistleblowers

Edward Snowden/ NSA + CIA

- Employee of CIA & later NSA subcontractor

PRISM: Top-secret NSA program leaked by Snowden

- NSA can 'reach directly into servers' from Facebook, Apple, Microsoft, Google, Youtube and other platforms
- Stored communications + real time collection (chat, videos, photos, file transfers)
- Critique: did 'grave damage' to American intelligence capacities
- Violated 'Espionage Act of 1917' + theft of government property
- US Passport revoked
- Recent news on citizenship:

Europe

3 minute read · September 25, 2022 11:32 PM EDT · Last Updated 7 days ago

Putin grants Russian citizenship to U.S. whistleblower Snowden

Reuters

Photo credits: New York Times, Electronic Frontier Foundation, Reuters

70

A BRIEF HISTORY

WikiLeaks

- Non-profit organization that publishes news leaks/ classified media by anonymous sources
- Founder: Julian Assange, Australian editor
- **2010**: Chelsea Manning leaks documents about the Afghan + Iraq war
- **2016**: Presidential election campaign
  - Released sensitive documents from the DNC
  - Published 20,000+ pages of emails from John Podesta, Clinton's campaign chair
- Criticized for violating the personal privacy of individuals numbers
- **2019-2023**: Julian Assange in U.K. jail and faces potential extradition to the U.S.

WikiLeaks Founder Julian Assange

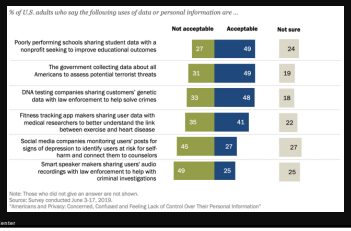
Photo credits: The Washington Post, WikiLeaks, Reuters, The Intercept

71

Personal Perspectives

"People are more accepting of personal data collection for *specific* purposes."

% of U.S. adults who say the following uses of data or personal information are ...



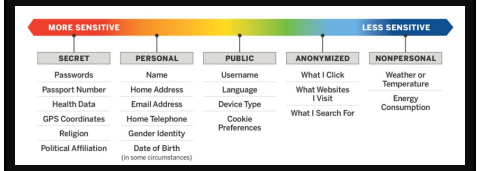
Note: Those who did not give an answer are not shown.
Source: Survey conducted June 3-17, 2019.
"Americans and Privacy: Concerned, Confused and Feeling Lack of Control Over Their Personal Information"

Source: Pew Research Center

72

Personal Perspectives

Privacy as a Spectrum



Source: Center for International Media Assistance

73