Inclusive Design and Logical Operations

In this lecture, we will cover

- Defining:
  - Accessible Design
  - Universal Design
  - Inclusive Design
- Cambridge Inclusive (accessible) Design Toolkit
- Microsoft Inclusive Design Principles

> Third Law of Technology Technology comes in packages, big and small.
> Further Thinking: All technology, big and small, needs to be designed with the client in mind.

Opening Question

How are you feeling about the coding side of things?
0 - I am not coding yet
3 - I am comfortable and still learning
5 - We need to pick up the speed/do more

Your future in CS

I used to include this on my slides, but since these slides have changed - going to just leave it up here for every notebook. I get a lot of questions about more programming courses, the concentrations, and minors in computer science. Here is a brief reminder.

CS 164 – Next Course In Sequence, also consider CS 220 (math and stats especially)

- CO Jobs Report 2021 – 77% of *all* new jobs in Colorado require programming
- 60% of all STEM jobs requires *advanced* (200-300 level)
- 31% of all Bachelor of Arts degree titled jobs also required coding skills
- 2016 Report found on average jobs that require coding skills paid $22,000 more

- Concentrations in CS:

  - Computer science has a number of concentrations. General concentration is the most flexible, and even allows students to double major or minor pretty easily. The others are specialized paths in CS. Software Engineering, Computing Systems, Human Centered Computing (Psychology+CS), Networking and Security, Artificial Intelligence, and Computer Science Education.
  - Minors: We have three minors. Minor in computer science (choose your own adventure minor), Minor in Machine Learning (popular with stats/math), and Minor in

Bioinformatics (Biology+Computer Science)

Defining Accessibility

Accessible Design

By default, the bare minimum is accessible design, and it is (loosely) protected by law

- Meets Compliance
- Often requires special tools or accommodations

Example of accessible, but not universal or inclusive?

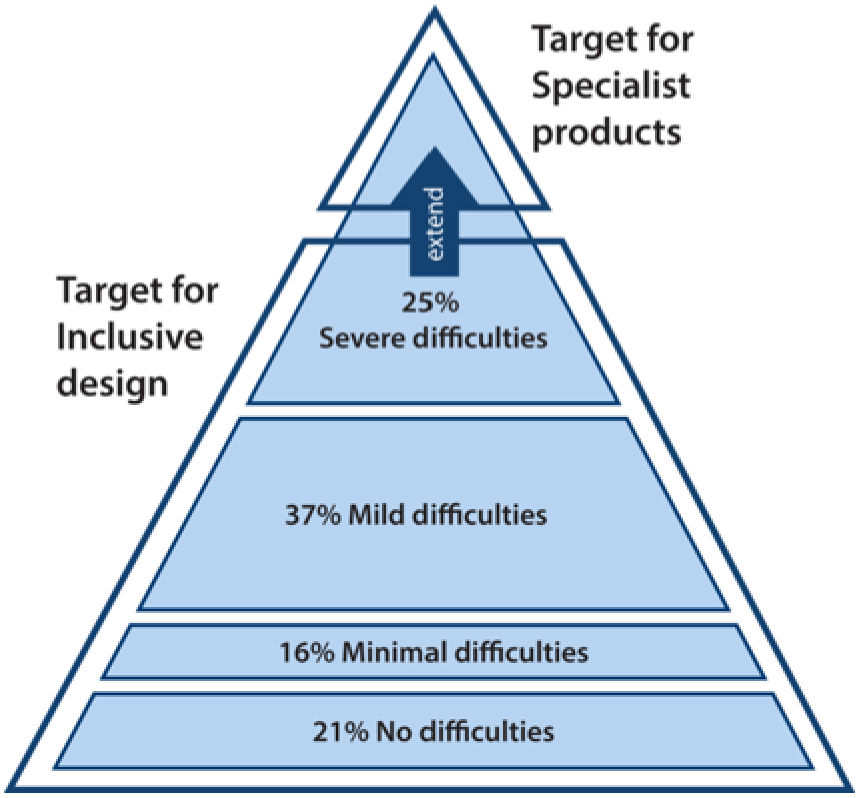
- To access your content, you have a special application designed for different ability levels.
    - A gray scale version of the site, accessible by a special url.
- A more obvious real world example?
    - The accessible access entrance to the building is in the alley / back entrance
        - Even with a sign pointing towards it, it meets compliance standards but not every welcoming

Universal Design

Designing in such a way, that accessibility is already built into the design.

- Makes compliance default / built into design
- Website example:
    - Uses color pallet that is accessible for all ranges of colorblindness
    - Looks good for all people
- BBC Guidelines Reference
    - put people first
    - add value for disabled people
    - prioritize familiarity and consistency
    - give control over content
    - provide different ways to interact with elements of the user interface.
- These changes benefit everyone!

- University of Cambridge Inclusive (Universal) Design Toolkit



NOTICE: Only 21% of your audience have no-difficulties.

Microsoft Spectrum

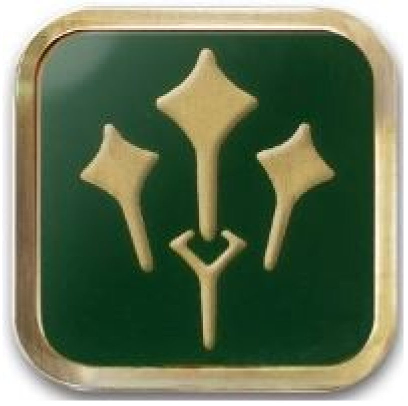
- Difficulties are on a spectrum of:
  - Situational
    - You are holding tea in your hand, so making difficult to text your friend.
  - Temporary
    - You broke your hand, so will have difficulty texting for a few months.
  - Permanent
    - You got in a car wreck from texting and driving and are paralyzed from the neck down. (also, don't text and drive!)

Inclusive Design

Accessible by different ability levels, gender identification, backgrounds, and cultural identities without alienation

- Goes beyond being "usable"
- Takes into account how to best:
  - Make all audiences feel welcome
  - Feel like a valued client
  - Takes into account various backgrounds and perspectives
    - in the design process
    - the implementation phase
    - in the final product

Recent Example:



- Final Fantasy XIV:
    - Changed "Sage" class icon due to original model triggering Trypophobia

        > The design concept is unchanged, with the icon being based on the four nouliths which form the sage's armament. The holes in the original design were added for detail, but they ended up appearing as a cluster. To address the problem, the new icon reduces the holes while accentuating the design concept. ~ Naoki Yoshida

    - This change happened during the testing phase of the game, before it was fully open to the general public.
    - They took client feedback
    - Made a change more welcoming of everyone. (yes, there are other things that are not)

> Depending on the location, Universal and Inclusive are used interchangeably. We will keep the definitions separate for this course.

Creating Inclusive Applications

The challenge

- It is *impossible* to represent all cultures and backgrounds on your design team!
    - Though we need to have more diverse backgrounds, than what the industry has
    - Including diverse backgrounds has been shown to improve your final product
- So how do we design to include all audiences?

Microsoft Inclusive Design Principles

- Recognize exclusion
- Solve for one, extend to many
- Learn from diversity

Modern Design and Software Engineering

- A iterative process
    - prototype
    - test
    - get feedback
    - prototype again
- A process that involves
    - communication with a **wide** variety of audiences
    - input from a team of people
    - individuals who specialize in making products more accessible
        - Human Centered Computing
        - Requires: psychology, statistics, and/or design thinking!
        - Why, because they are at the forefront of design!

> Discussion
> To start brainstorming for your research paper. Start talking about the applications you use. How inclusive are they? What can they do to be improved upon?

Logical Operators

Switching gears back to programming, we will introduce logical operators!

- You are already using the Conditional Operators:
    - `<` - is my left **less than** my right
    - `<=` - is my left **less than or equal to** my right
    - `>` - is my left **greater than** my right
    - `>=` - is my left **greater than or equal to** my right
    - `==` - is my left **equal to** my right
    - `!=` - is my left **not equal to** my right

These all evaluate to `True` or `False`

Now, what if we want to compare truth statements?

- Example:
    - I like both apples and pineapples
    - I want either cake or ice cream?

Logical Operators

Operator	Description	Example
`and`	True if a and b are True	a `and` b
`or`	True if a or b are True	a `or` b

Operator	Description	Example
`not`	Flips the bit (True becomes False)	`not` a

Reading Check-in

Given the following output, what operator should I use for this code?

```
def reading_check(left, right):
    return left `?` right
```

and the output is:

```
True
False
False
False
```

A - and

B - or

C - not

In [ ]:
```
def reading_check(left, right):
    return left ? right

print(reading_check(True, True))
print(reading_check(True, False))
print(reading_check(False, True))
print(reading_check(False, False))
```

```
True
False
False
False
```

What if the output is?

```
True
True
True
False
```

In Class Activity:

Take the following function. Given left and right, I want to have it return

```
False
False
False
True
```

How can I structure my statement?

In [ ]:
```
def inclass(left, right):
    return False ## Change this line of code!

print(inclass(True, True))
print(inclass(True, False))
print(inclass(False, True))
print(inclass(False, False))
```

```
False
False
False
True
```

Expanding upon if statements or other conditions!

Of course, logical operators are only useful if we want to build on each other!

So let's take the following:

In [ ]:
```
import random


def rps(throw1, throw2):  ## inclass activity 2
    if throw1 == "rock" and throw2 == "scissors":
        return True
    return False

def computer_play():
    play= random.randint(1,3) #gets a random number between 1-3
    if play == 1:
        return "rock"
    elif play == 2:
        return "scissors"
    return "paper"

def check_winner(player1, player2) :
    if rps(player1, player2):
        return "Player 1"
    elif rps(player2, player1):
        return "Computer"
    return "No one"

player1 = input("Rock, Paper, or Scissors").lower() # takes the string and makes it lo
player2 = computer_play()
print(f"Player 1 plays {player1}.")
print(f"The computer plays {player2}.")

winner = check_winner(player1, player2)

print(f"{winner} wins!")
```

```
Player 1 plays rock.
The computer plays rock.
No one wins!
```

In class Activity 2

Modify `rps` to work for all Rock, Paper, Scissors options!

Thinking further: is an if/else statement actually needed here?

Next Class

We will focus on coding, and working on Lists! A key structure we need for more advanced programming.