# Day 4

# How to be a Good Programmer!

# Programmer Tips

**Optional, but these can improve your productivity!**

# Keyboard Shortcuts
and
# Text Editors

# Keyboard Shortcuts

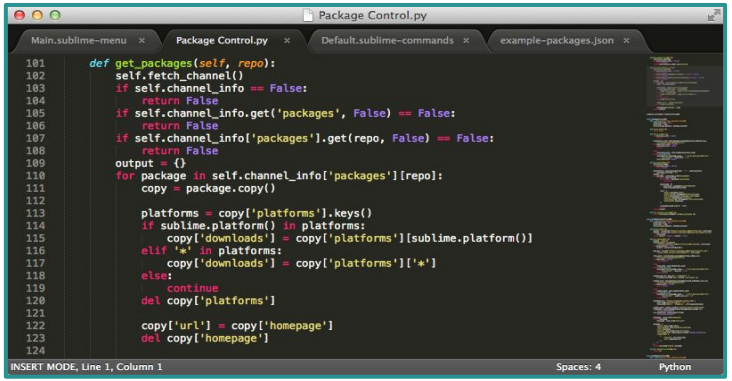
- **CTRL + F**
- **CTRL + LEFT/RIGHT**
  - Skips to the beginning of the word
  - Useful for deleting whole word or skipping through words quickly
- **CTRL + SHIFT + LEFT/RIGHT**
- **CTRL + SHIFT + UP/DOWN**
- **CTRL + BACKSPACE**
- **CTRL + A**
  - Select all words in the file
- **WINDOWS + E**
  - Open Windows file explorer

# Sublime Keyboard Shortcuts

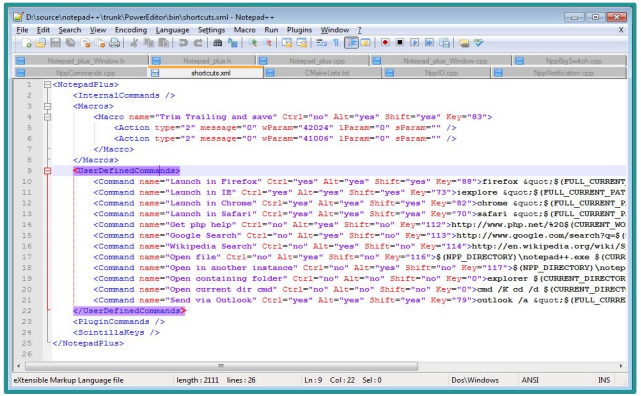
- **CTRL + B**
- **CTRL + X**
  - Cut line
- **CTRL + L**
  - Select entire line
  - Repeat to select multiple lines
- **CTRL + D**
  - Select word and display all other occurrences of word
- **CTRL + K**
  - Delete line
- **CTRL + ALT + UP/DOWN**
  - Move line up or down
- **ALT + SHIFT + 2**

# Text Editors

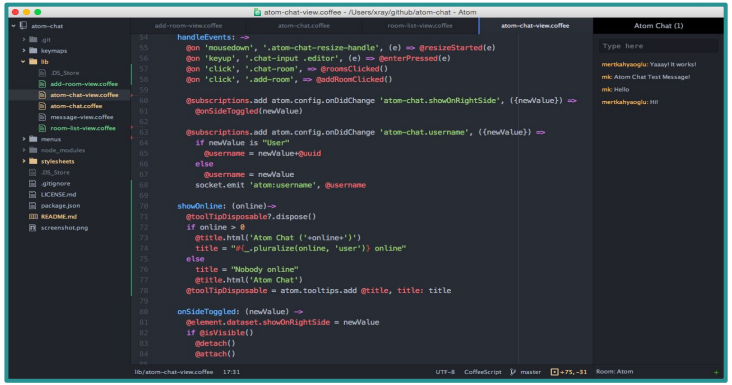
- **Notepad++**
  - Difficulty: low
  - Extensibility: low




- **Atom**
  - Difficulty: low
  - Extensibility: medium

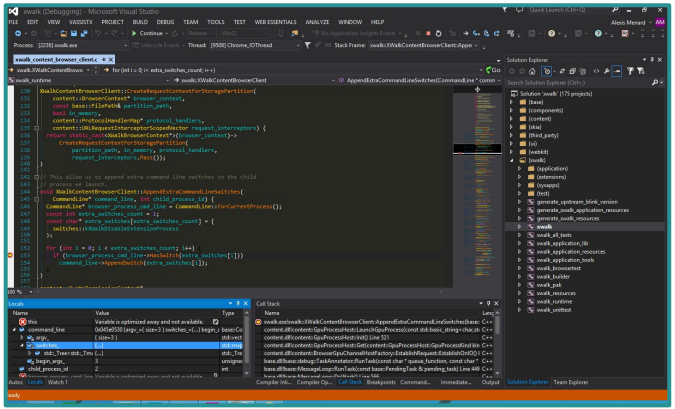



- **Sublime Text**
  - Difficulty: low
  - Extensibility: low
  - Beauty: high

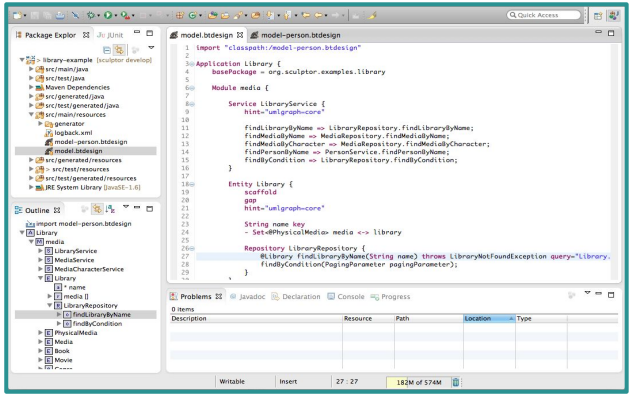


# Text Editors

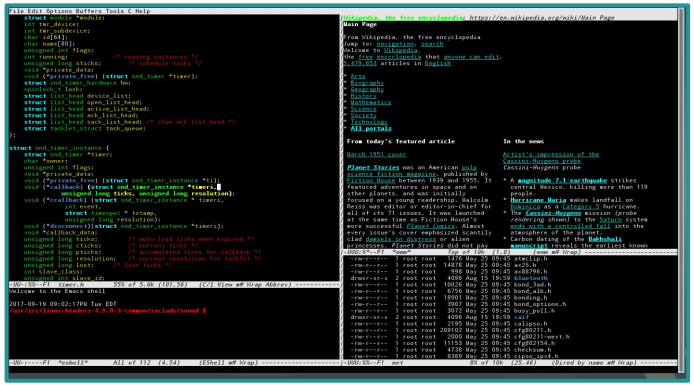
- **Eclipse**
  - Specialty: Java
  - Extensibility: medium



- **Vim or Emacs**
  - Difficulty: high
  - Extensibility: extremely high
  - Flexibility: extremely high

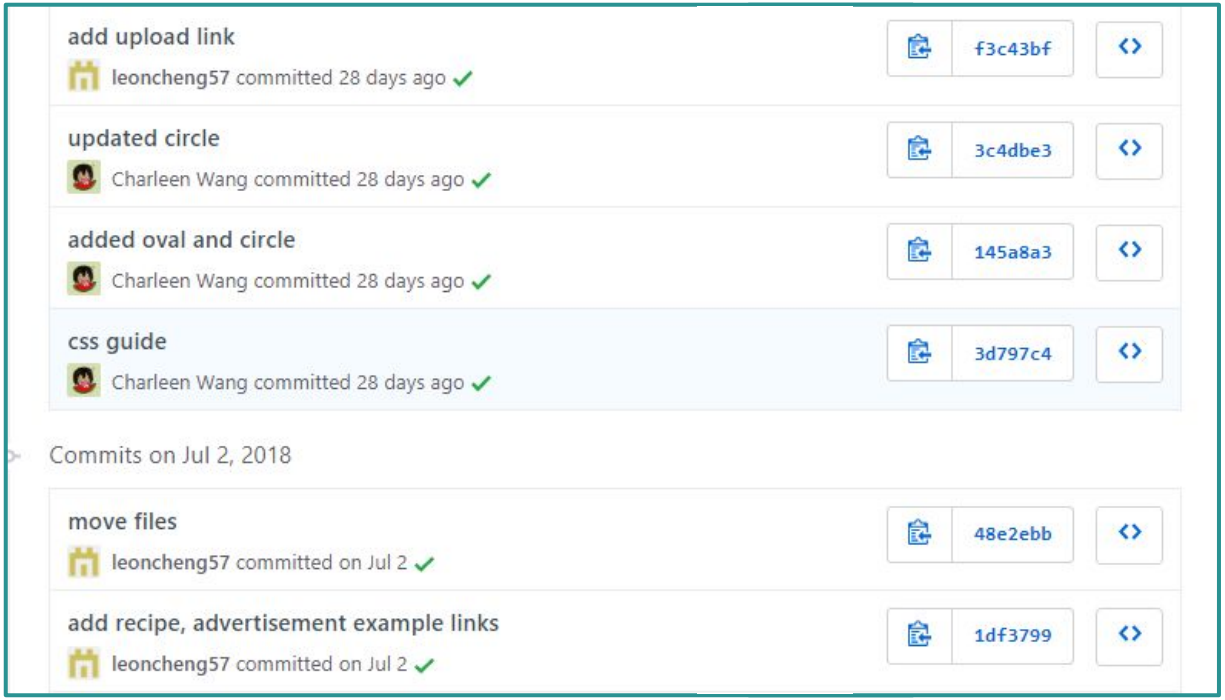


- **Visual Studio**
  - Difficulty: medium
  - Extensibility: medium
  - Specialty: Large Projects



# Saving Backups
and
# Comments
and
# Refactoring
and
# Pseudocode

# Saving backups / versions




- Saves your butt if your program crashes
- Try to use descriptive names

# Writing Comments

```python
class Sprite:
    name = "default_name"
    hometown = "default_hometown"

    # Make sure to use 'self' as a parameter
    # And use it when calling class variables
    def get_description(self):
        result = ""
        result += "Name: " + self.name + "\n"
        result += "Hometown: " + self.hometown + "\n"
        return result

    # Different from 'self' class vars and input parameter
    def set_name(self, name):
        self.name = name

    def set_hometown(self, hometown):
        if (hometown == "NYC"):
            hometown = "NYC (da best)"
        self.hometown = hometown

    # Without 'self' parameter, it is a Static class function
    def generic_greeting():
        return "Hi, how are you?"

```

- Easier to understand for teammates
- Easier to understand for future you

# Refactoring / Don't Repeat Code (DRY)

```
def drive_left():

    if (gas not in engine):
        return "Error"
    start_engine()
    if (people_in_front == True):
        return "Error"
    step_on_gas()

    rotate_wheel_to_left()
    continue_driving()
    return "Success"


def drive_right():

    if (gas not in engine):
        return "Error"
    start_engine()
    if (people_in_front == True):
        return "Error"
    step_on_gas()

    rotate_wheel_to_right()
    continue_driving()
    return "Success"
```

```
def ready_to_drive():
    if (gas not in engine):
        return "Error"
    start_engine()
    if (people_in_front == True):
        return "Error"
    step_on_gas()


def drive_left():
    ready_to_drive()

    rotate_wheel_to_left()
    continue_driving()
    return "Success"


def drive_right():
    ready_to_drive()

    rotate_wheel_to_right()
    continue_driving()
    return "Success"
```

- Shorter, more readable code
- Easier to write future code
- Fewer mistakes in the future if changes made

# Pseudocode

algorithm/code but in words

Power of 2:

```
Read n.
Initialize power to 1.
Repeat n times:
    Double power.
return power
```

```python
def power_of_two(n):
    # Initialize power to 1.
    power = 1
    # Repeat n times:
    for i in range(n):
        # Double power
        power *= 2
    # return power
    return power
```

# Naming
and
# Documentation
and
# Testing

# Naming

```
class JuniorRegister:

    student_list = ['Charleen', 'Leon', 'Chris']
    MAX_CLASS_SIZE = 200

    def __init__(self, x, y):
        self._x = x
        self._y = y

    def this_is_my_function(self):
```

- Variables, functions, methods, packages, modules
  Lower_case_with_underscores
- Classes and Exceptions
  CamelCase
- Private methods
  __double_leading_underscore(self, ...)
- Constants
  ALL_CAPS_WITH_UNDERSCORES

# General Naming Guidelines

- Use intention-revealing names
  - functions, variables, classes and everything
  - For example:
    - number instead of nu, num, n
    - Product instead of pro, p etc

```
class Algebra:

    number =
    operation =

    def my_product(numbers):
        product =
```

# General Naming Guidelines

- Use intention-revealing names
- Avoid one-letter variables (esp. l, O, I)
    - will be confused with 1 or 0
    - makes your life difficult

# General Naming Guidelines

- Use intention-revealing names
- Avoid one-letter variables (esp. l, O, I)
- Avoid redundant labeling

```
import audio

core = audio.Core()
controller = audio.Controller()
```

YES

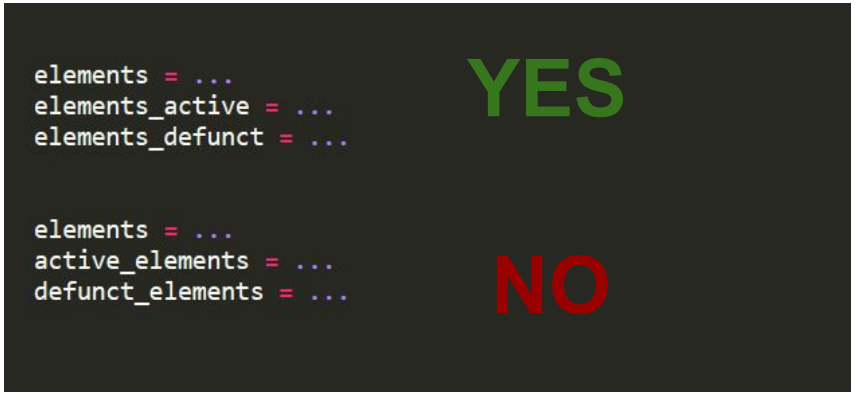
```
import audio

core = audio.AudioCore()
controller = audio.AudioController()
```

NO

# General Naming Guidelines

- Use intention-revealing names
- Avoid one-letter variables (esp. l, O, I)
- Avoid redundant labeling
- Prefer "reverse notation"



# Documentation

```
def my_product(numbers):
    """ Compute the product of a sequence of numbers.

    Parameters
    ----------
    numbers : sequence
        list of numbers to multiply

    Returns
    -------
    product : number
        the final product

    Raises
    ------
    TypeError
        if argument is not a sequence or sequence contains
        types that can't be multiplied
    """
```

# Documentation

- Not only for others, but also for yourself !
- Minimum requirement: at least a docstring
- Document arguments and return objects, including types
- For complex algorithms, document every line
- For big programs/projects provide an how-to or quick-start your website

# Testing & Debugging

- Testing increases the confidence that your code works correctly, not only for yourself but also for your reviewers
- Tests are the only way to trust your code!
- It might take you a while to get used to writing them
- Unit case: The smallest testable piece of code

# Unit Tests

- **Unit Test:** The smallest testable piece of code
- Why?
    i. Easier to test the whole program, if the units work
    ii. Can modify parts, and be sure the rest still works
    iii. Provides examples of how to use code

# Unit Tests Example

```
import unittest

def my_product ( numbers ):
    """ Compute the product of a sequence of numbers . """
    total = 1
    for item in numbers :
        total *= item
    return total

class TestUM(unittest.TestCase):

    def test_numbers_3_4(self):
        self.assertEqual( my_product([3,4]), 12)

    def test_strings_a_3(self):
        self.assertEqual( my_product([2,3,1]), 6)

if __name__ == '__main__':
    unittest.main()
```

# Unit Tests Example

```
77  import unittest
78
79  def my_product ( numbers ):
80      """ Compute the product of a sequence of numbers . """
81      total = 1
82      for item in numbers :
83          total *= item
84      return total
85
86  class TestUM(unittest.TestCase):
87
88      def test_numbers_3_4(self):
89          self.assertEqual( my_product([3,4]), 12)
90
91      def test_strings_a_3(self):
92          self.assertEqual( my_product([2,3,1]), 6)
```

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
[Finished in 0.2s]
```

# Important . . .

- Testing all functions and adherence to specifications
- Finding all edge cases
- Testing false arguments e.g. passing in a string when should only integers allowed

```
def test_numbers_3_4(self):
    self.assertEqual( my_product([3,"aa"]), "Input type error!")
```

# Resource for Python Conventions

## PEP 8 -- Style Guide for Python Code

| | |
|---|---|
| PEP: | 8 |
| Title: | Style Guide for Python Code |
| Author: | Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Nick Coghlan <ncoghlan at gmail.com> |
| Status: | Active |
| Type: | Process |
| Created: | 05-Jul-2001 |
| Post-History: | 05-Jul-2001, 01-Aug-2013 |

# Programmer Tips

- Use **terminal** for running and managing files
- Use **GitHub** for teamwork and version control (saving backups)
- **Keyboard shortcuts** for easier coding
- Customize your **text editor** for more comfortable coding
- Write out **pseudocode** to extract your ideas before starting to code
- Write **comments** to make your teammates and future you understand
- **Refactoring** or **DRY** so that code is reused
- **Name** variables, **document** methods, and **test** your code

# Effective Presentation Tips and Tricks

# Why do we care about presentation skills?

# Story Time :)

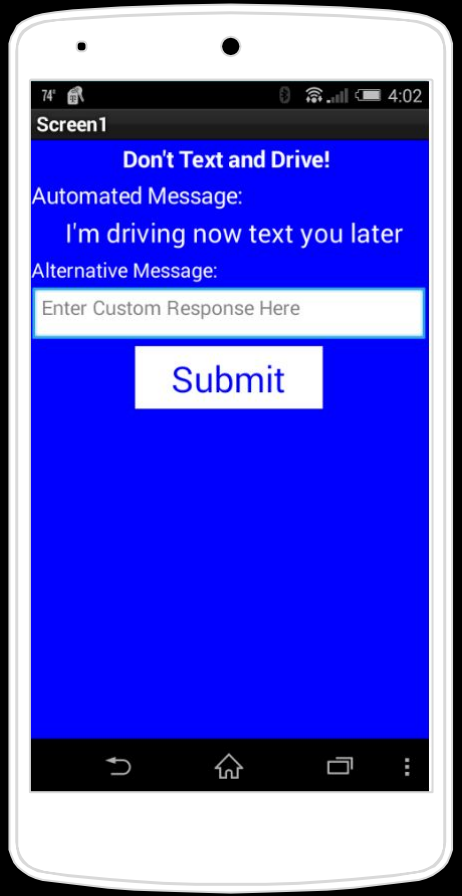
# TextBuddy

1. read out loud the received text
2. send an automated message back
3. customize automated message

Ugly

Minimal function

**Has purpose**



relevance

key stats

49% of drivers under the age of 35 text and drive

60% of all drivers use their phones while they drive

**every year, 1.6 million car accidents are cell phone related**

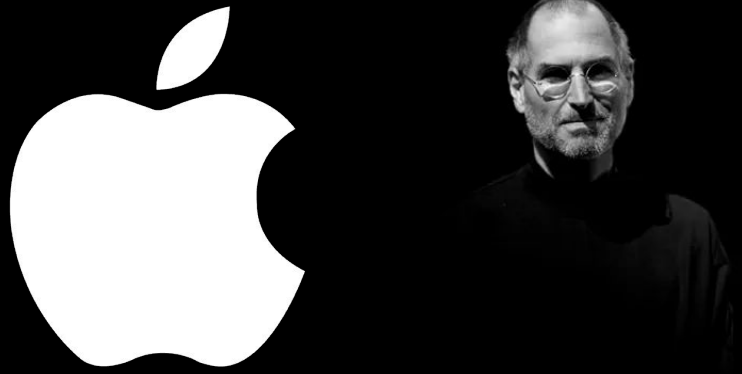

These accidents cause about a half million injuries and take 6,000 lives

**people who text and drive are driving outside of their lanes 10% of the time**

**they are also 3x more likely to crash**

Images and diagrams

# Persuasion



# Customer Loyalty

A bad mission statement:

> To sell the best computers and make lots of money!

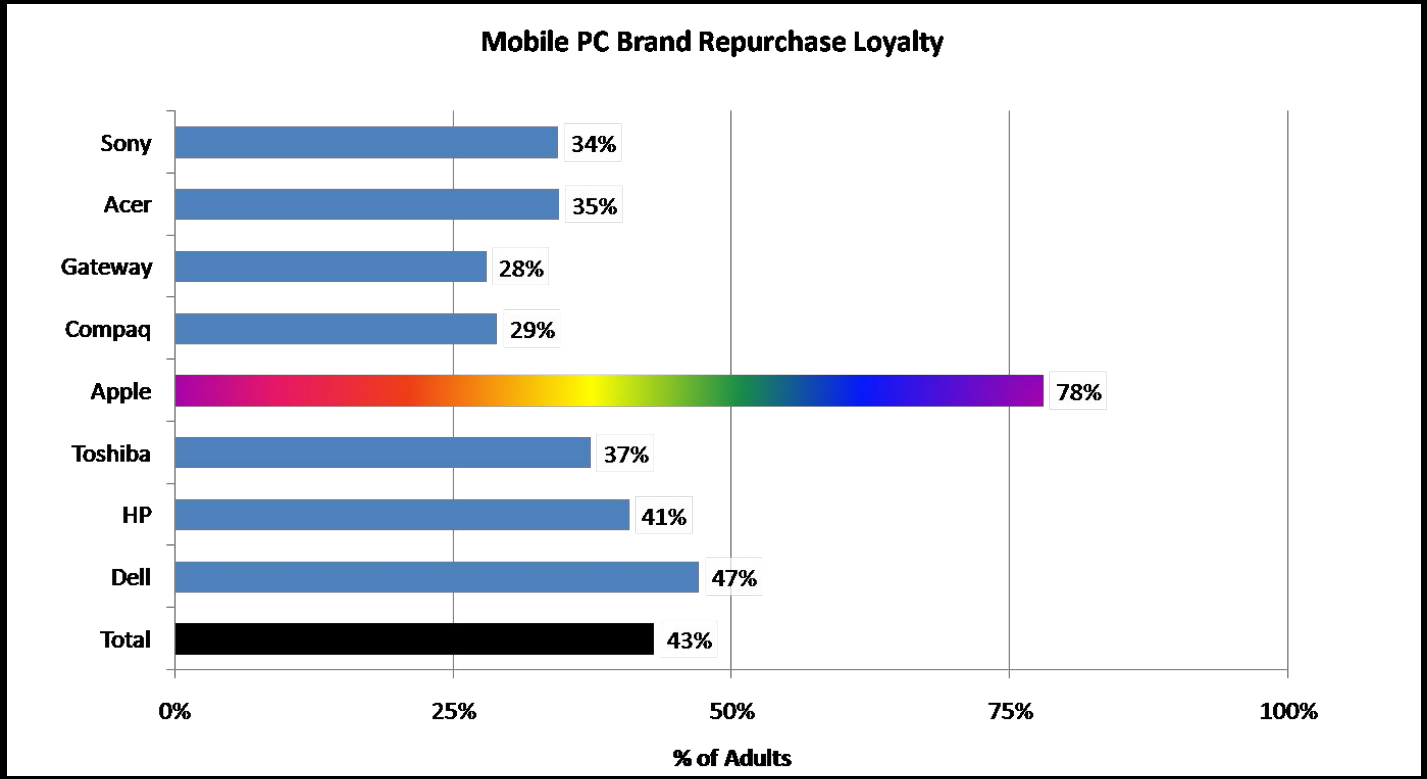
Steve Jobs:

> "To make a contribution to the world by making tools for the mind that advance humankind."

Dell:

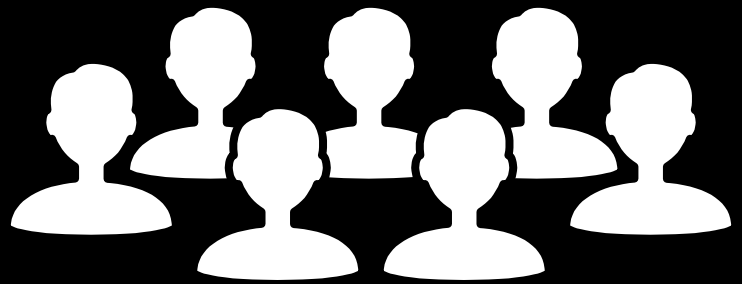
> none...

# Apple Brand Retention

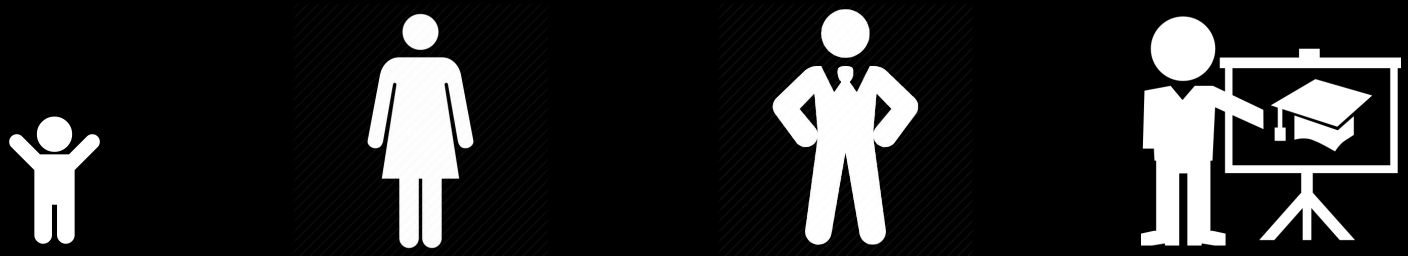


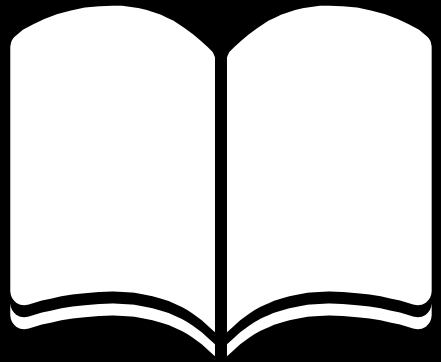
# PURPOSE

WHY are you presenting?
WHY am I listening to you?
WHY is this important?



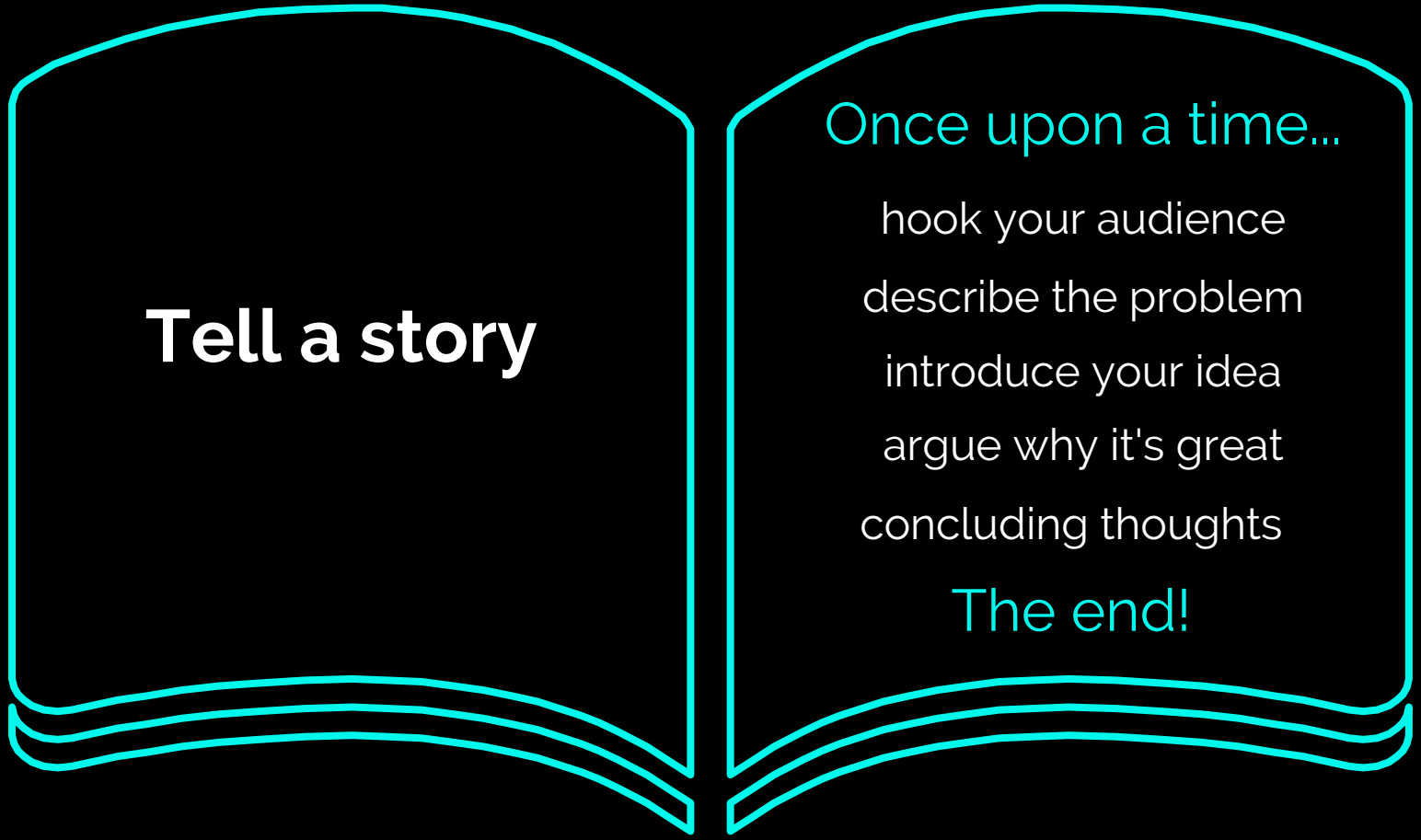
# AUDIENCE





education

entertainment

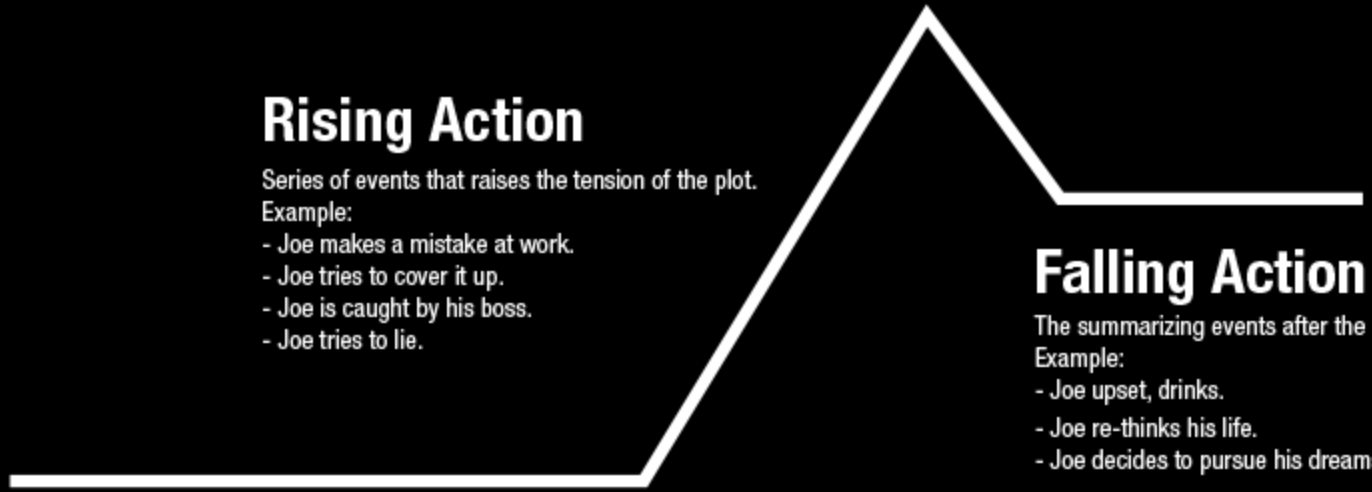

Tell a story
Once upon a time...
hook your audience
describe the problem
introduce your idea
argue why it's great
concluding thoughts
The end!

## Climax

The peak of the story.
Example:
- Joe gets fired from his job.

## Rising Action

Series of events that raises the tension of the plot.
Example:
- Joe makes a mistake at work.
- Joe tries to cover it up.
- Joe is caught by his boss.
- Joe tries to lie.



## Falling Action

The summarizing events after the peak.
Example:
- Joe upset, drinks.
- Joe re-thinks his life.
- Joe decides to pursue his dreams.

## Beginning

The Initial events that set the story in motion.
Example:
- Joe goes into work today, like any other day.

# Visuals

# Color Usage

Colors: use the same colors consistently

Images: choose images that are similar in style

Fonts: use three or less fonts, use them the same way (Headings/body text)

# Color Usage

Colors: use the same colors consistently

Images: choose images that are similar in style

Fonts: use three or less fonts, use them the same way (Headings/body text)

# Color Usage

Colors: use the same colors consistently

Images: choose images that are similar in style

Fonts: use three or less fonts, use them the same way (Headings/body text)

# Color Usage

Colors: use the same colors consistently

Images: choose images that are similar in style

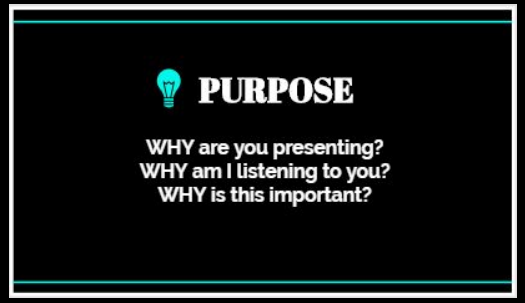
Fonts: use three or less fonts, use them the same way (Headings/body text)

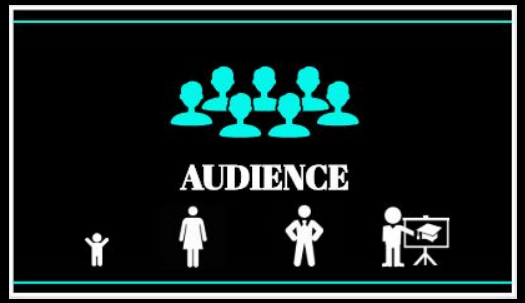
# Consistency/Coherence

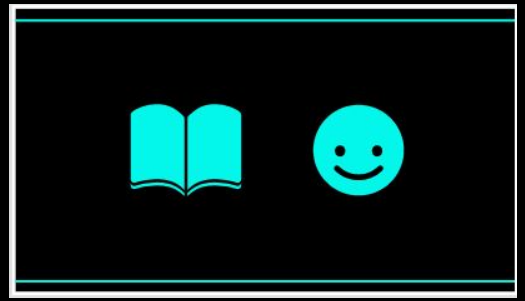
Colors: use the same colors consistently

Images: choose images that are similar in style

Fonts: use three or less fonts, use them the same way (Headings/body text)



# Consistency/Coherence

Colors: use the same colors consistently

Images: choose images that are similar in style

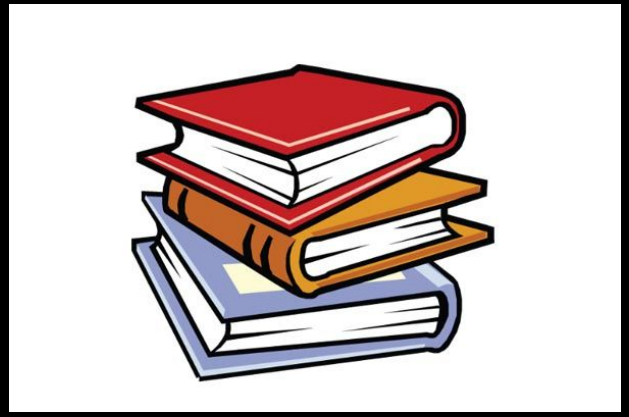
Fonts: use three or less fonts, use them the same way (Headings/body text)

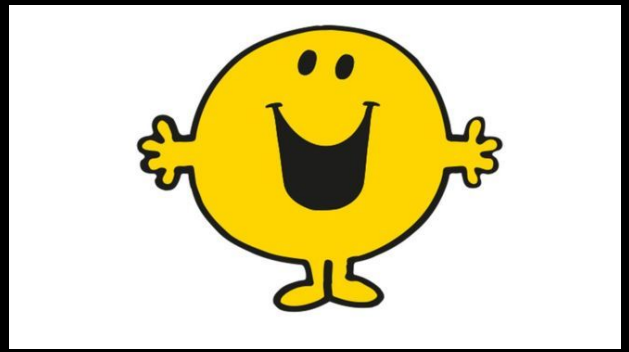
# Consistency/Coherence

Colors: use the same colors consistently

Images: choose images that are similar in style

Fonts: use three or less fonts, use them the same way (Headings/body text)



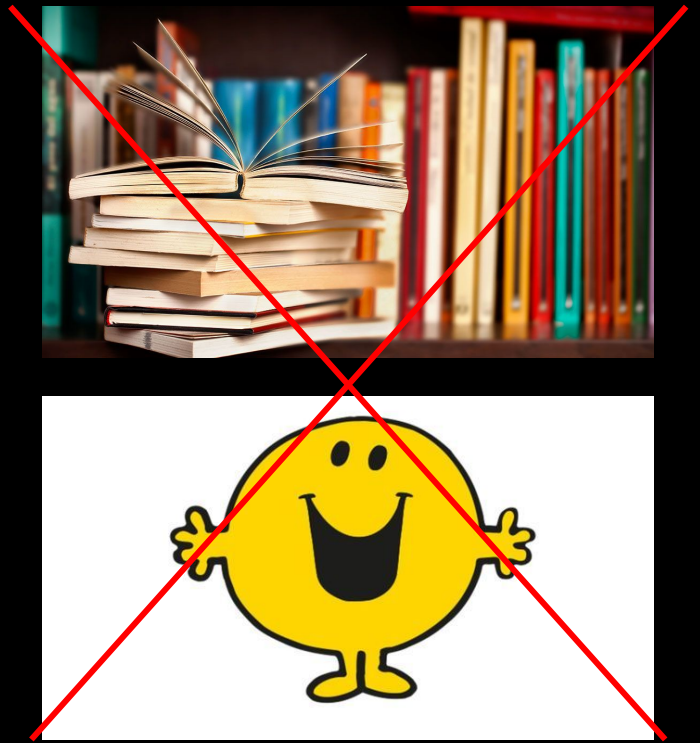


# Consistency/Coherence

Colors: use the same colors consistently

Images: choose images that are similar in style

Fonts: use three or less fonts, use them the same way (Headings/body text)

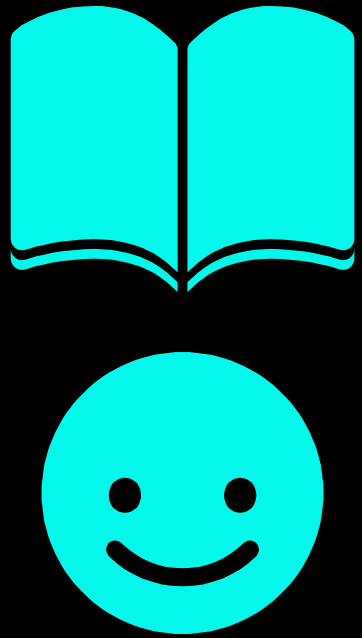


# Consistency/Coherence

Colors: use the same colors consistently

Images: choose images that are similar in style

Fonts: use three or less fonts, use them the same way (Headings/body text)

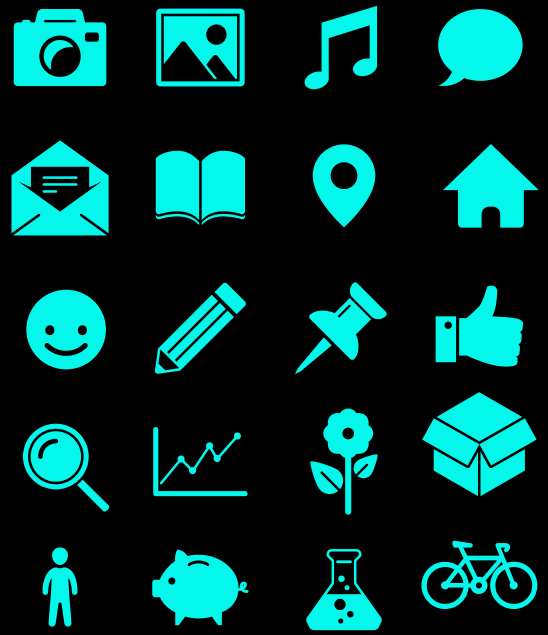


# Consistency/Coherence

Colors: use the same colors consistently

Images: choose images that are similar in style

Fonts: use three or less fonts, use them the same way (Headings/body text)

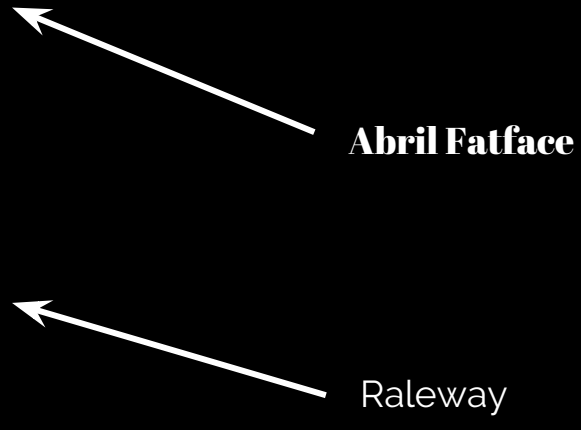


# Consistency/Coherence

Colors: use the same colors consistently

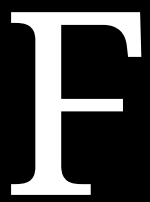
Images: choose images that are similar in style

Fonts: use three or less fonts, use them the same way (Headings/body text)



# Fonts

Serif



Sans Serif

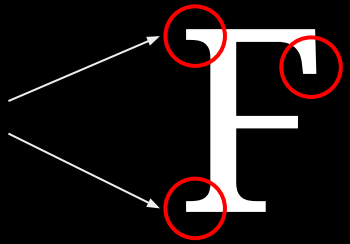
# Fonts

Serif

Sans Serif

"Sans" means "without" in French

These things are called "serifs"



F

# Fonts

| Serif | Sans Serif |
| --- | --- |
|  |  |
| For larger blocks of text | Modern, Simple, Clean, Friendly |
| Easier to read | More eye catching |
| Formal | Informal |

# Limit Font Usage

1. Choose complementary or contrasting:
   A. "PERSONALITY OF FONTS"
   **B. CHUNKY AND BOLD VERSUS THIN AND LIGHT**
2. MIX SERIF AND SANS SERIF
3. Be aware of sizing
   a. TRY ALL CAPS FOR SMALLER FONTS
4. *Match content*
5. Define a hierarchy:
   a. Use the same font for the same type of thing
   B. SEPARATE TO HEADINGS/TEXT

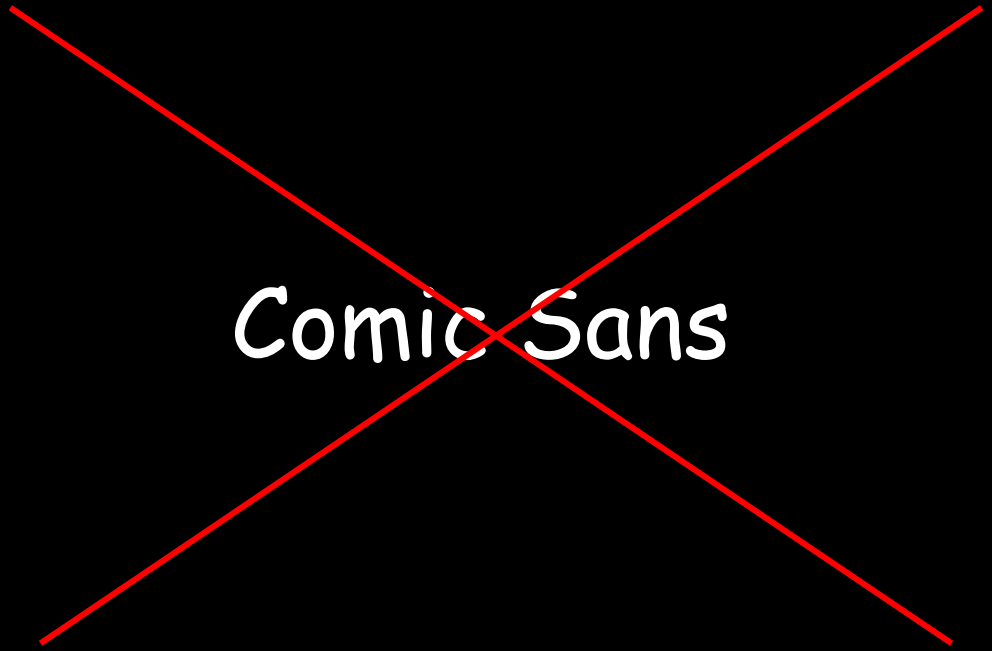
This is too hard to read!!

# Mixing Fonts

1. Choose complementary or contrasting:
    a. "personality of fonts"
    b. Chunky and bold versus thin and light
2. Mix Serif and Sans Serif
3. Be aware of sizing
    a. Try all caps for smaller fonts
4. Match content
5. Define a hierarchy:
    a. Use the same font for the same type of thing
    b. Separate to headings/text

# Font Pairing

| | |
|---|---|
| Similar Font Personalities | Special Elite<br>Consolas |
| Chunky bold, thin and light | **Acme**<br>Raleway |
| Serif and Sans Serif | **Sniglet**<br>Garamond |
| Fancy Themed | *Parisienne*<br>Lora |
| Same Font Family | **Caveat Brush**<br>*Caveat* |


Comic Sans

Google Google

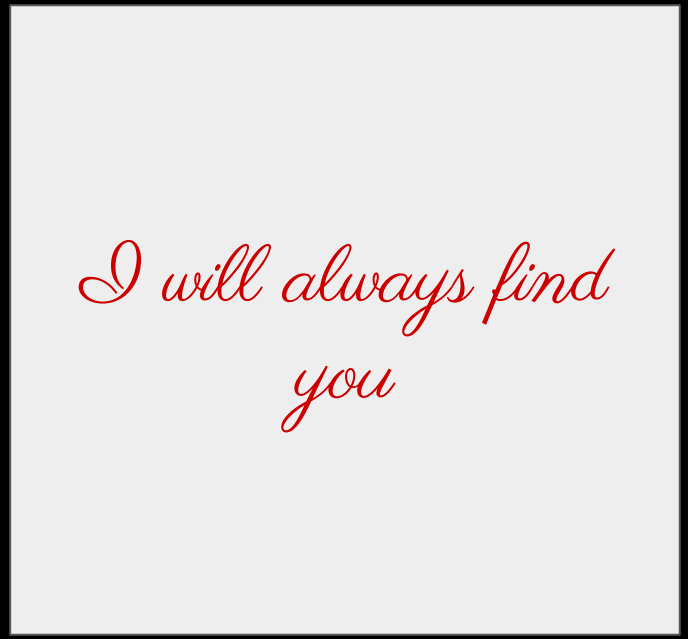

Microsoft

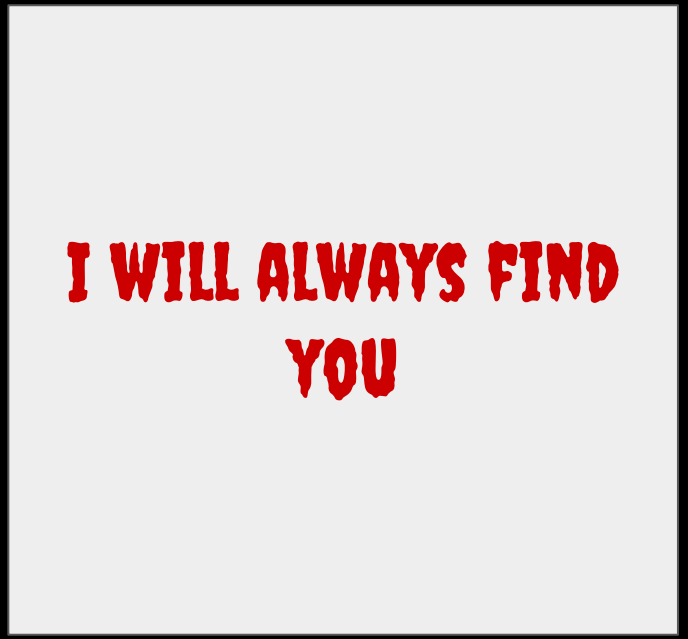

Microsoft


CHANEL


CHANEL

# Love Letters



# Content

When you write your slides, less is more

With content that you put on it, it is important that you don't write too much stuff. Your audience can either listen to you or read the slide, but not both

Long sentences are difficult to follow and clutter the slide. Use shorter sentences to make what you are saying more clear.

Make sure the slide is not too cluttered so that your audience doesn't know where to look

# Content

Less is more

Don't write too much

Audience can either listen or read

Use short sentences/phrases

Don't clutter your slides

Hello everyone!

Can you see me?

Yoo hooo~

Look at me!

Do I have your attention?

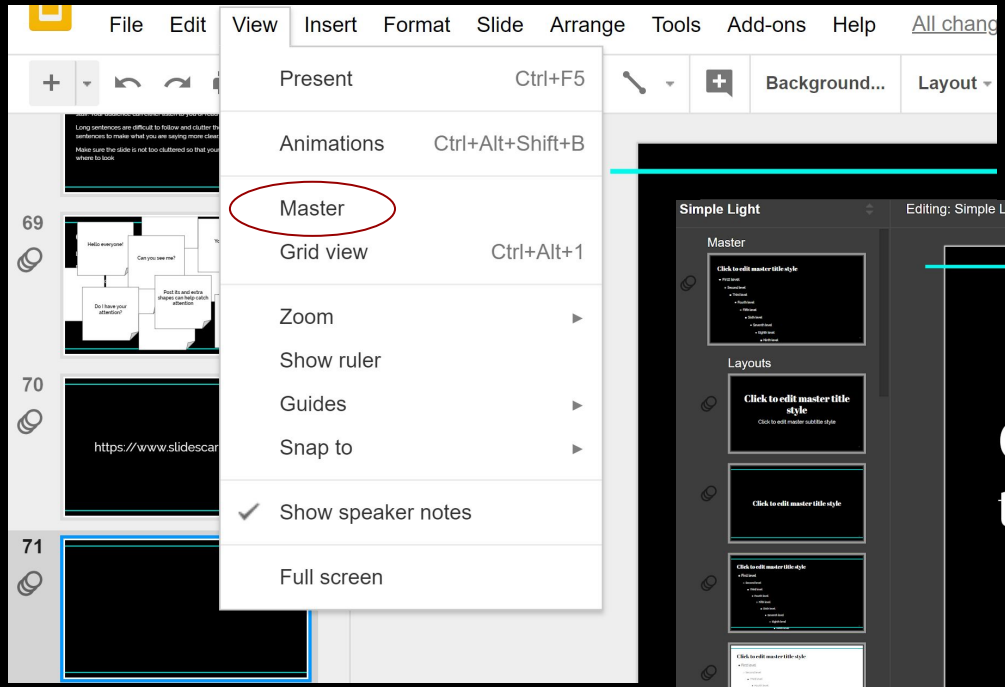
Post its and extra shapes can help catch attention

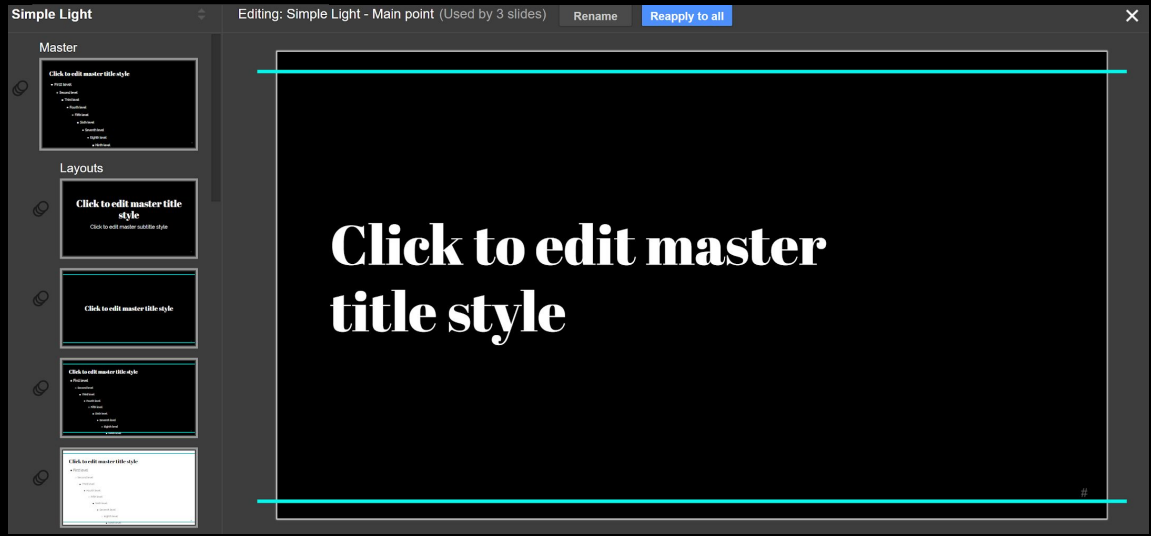
Notice how this slide is getting too cluttered

https://www.slidescarnival.com/

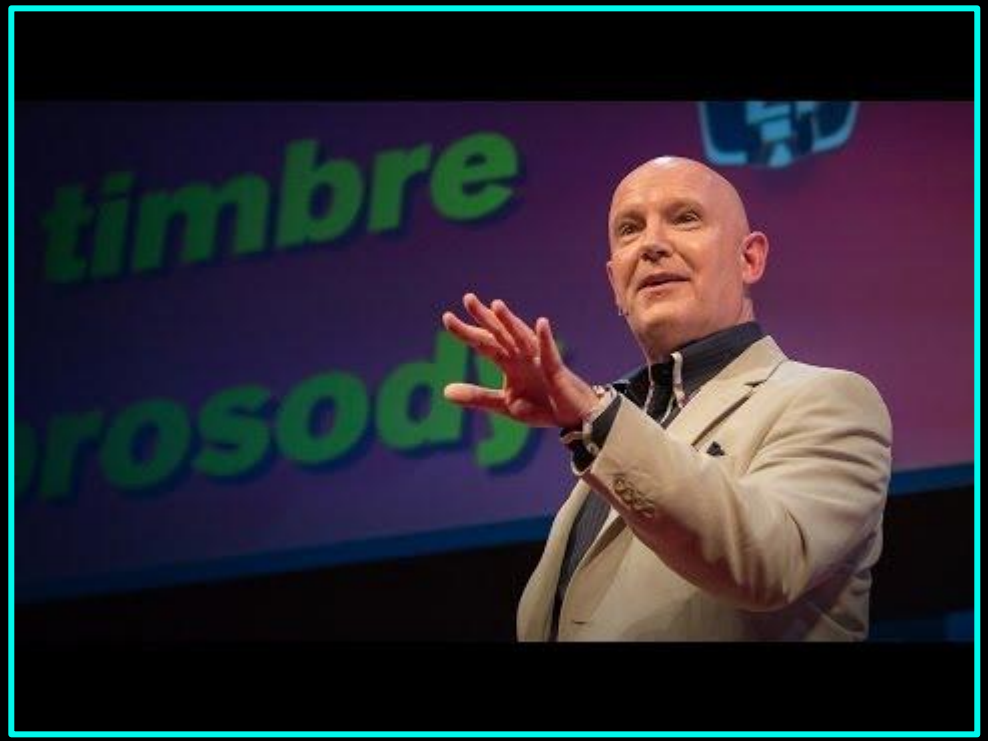
https://slidesgala.com/

# Make your own themes in Google Slides!



How to sound smart in your TEDx Talk | Will Stephen | TEDxNewYork
https://youtu.be/8S0FDjFBj8o




How to speak so that people want to listen | Julian Treasure
https://youtu.be/eIho2S0ZahI