Recitation 1

(cause the first one was zero (duh))

02120 Fall 2025

Style Guidelines 

02120 Fall 2025

✨Variables✨

- **Appropriate Scope:** Variables should only persist as long as necessary-don't keep them around longer than needed.
- **Descriptive Names:** Use clear and descriptive names for variables
- **Single Letters for Temporary Variables:** Use single letters like i, j, k for temporary integers.
- **snake case for Compound Names:** Use snake case for variable names that combine multiple words.

Good

```
nucleotide
```

```
num_cells
```

```
for i in range (0,length)
```

Bad

```
x
```

```
thing
```

```
numcells
```

```
for iterator in range (0,length)
```

Functions / Modularity

- **Reasonable Partitioning:** Your program should be divided into functions that each accomplish a single task.
- **Function beginning:** The function name and parameters should serve as a "topic sentence," clearly conveying the function's purpose.
- **Reusability:** Functions should be clearly reusable in different contexts, where applicable.
- **Appropriate Names:** Function names should clearly reflect their purpose.
- **Logical Input/Output:** Functions should only take in and return what's necessary.
- **Type hints:** Function parameters and outputs should have type hints.

Good

```python
def modulate(dividend : int ,divisor : int) -> int:
    remainder = dividend % divisor
    return remainder
```

Bad

```python
def f(a,b,c):
    return a % b
```

Comments

- **File Header:** Include your name and the date at the top of each file. (Also indicate if there are any other students with whom you had high-level discussions.)
- **Function Documentation:** Each function should have a docstring at the beginning explaining its purpose.
- **Internal Comments:** Provide comments within functions to clarify complex sections of code

```
//Name: Phillip Compeau
//Date: 12/07/2004
//Collaborators: Mia Sanborn


def find_sound(animal):
    """
    prints the sound an animal would make (ex. dog -> woof)
    Parameters:  animal (str): string representation for the
animal, must be within the known dataset
    Returns:  sound (str): the sound the animal input would make
              returns the empty string if animal not in
              dataset
    """
    if (animal == "dog"): return("woof")
    elif (animal == "lion"): return("roar")
    elif (animal == "fish"): return("bloop") #is bloop a sound?
    elif (animal == "dolphin"): return ("eeeeeee")
    else: return("") #if animal is not in the dataset
```

good!

📈 Efficiency 🏃 💨

- **Efficiency:** Your program should provide a reasonably efficient solution to the problem at hand.
- **Code Cleanup:** Make sure that your program doesn't include any old or unnecessary code

dirty

```
def unused_song():
    print("Old MacDonald had a farm...
E-I-E-I-O!")


def find_unicorn(animals):
    for i in range(1000000):
        for a in animals:
            if a == "unicorn":
                return "Found the unicorn!"
    return "No unicorn here..."
```

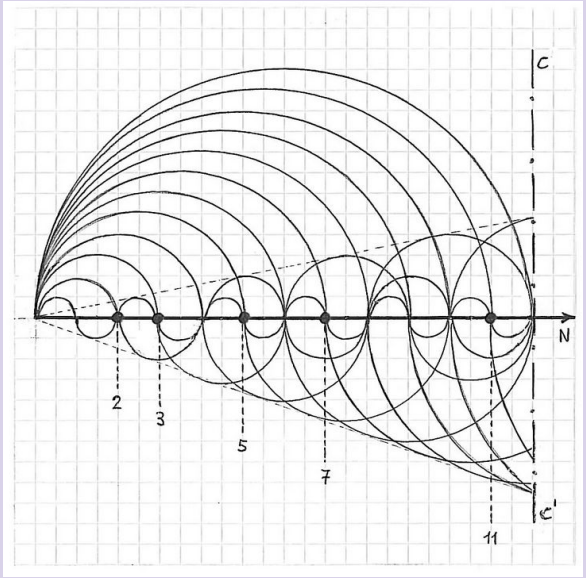
bad!

inefficient

Now to Joelle for some really cool visualizations

Why do we care about prime numbers?

- Simple but complex
- Intricate patterns
- Cryptography! 🔒



Tips for Success

02120 Fall 2025

Get Running Programs First, Then Focus on Formatting

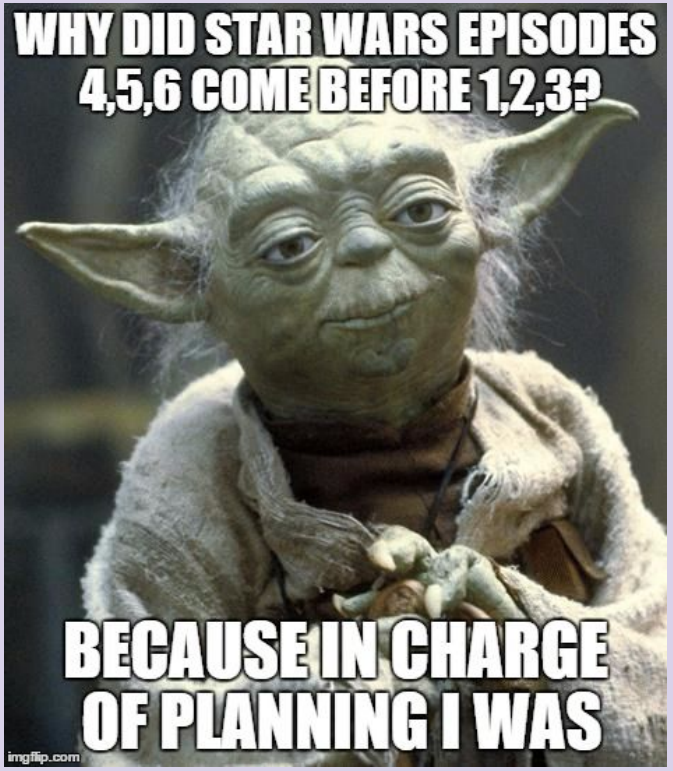
The primary focus should be on getting your programs to run correctly. Once the core functionality is in place, you can refine the formatting to improve readability and maintainability

Comment Your Code More Often Than You Think You Should

Comments are essential for making your code understandable to others (and to your future self). It's better to over-explain than to leave others guessing about your logic.

Plan before you code

Avoid jumping straight into writing code. Start by planning your approach in English, then draft pseudocode, and finally, translate it into actual code. This process is akin to writing a short story: clear planning leads to a coherent final produce

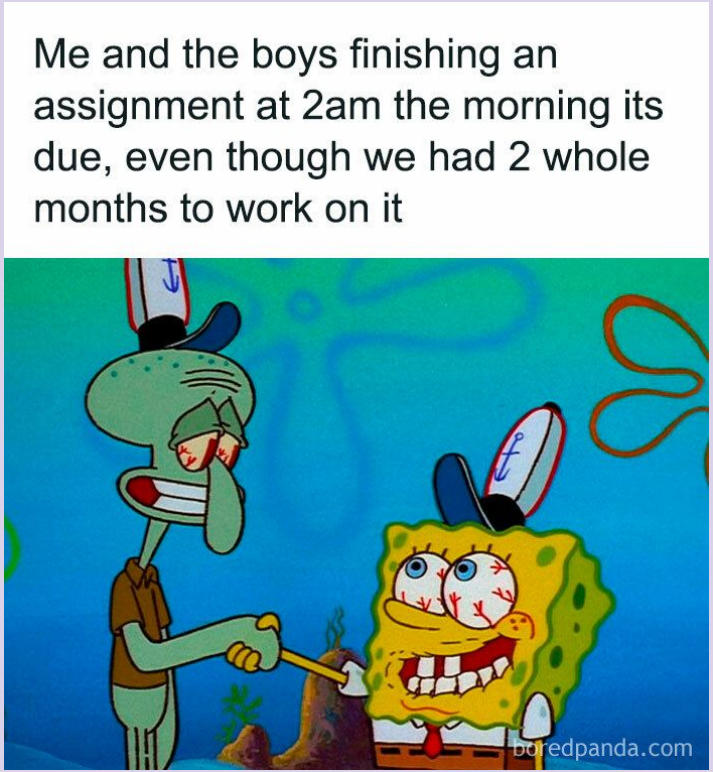


Code the pieces you know first

Begin by coding the parts of the program that you are most confident about. This approach helps build momentum and clarifies the remaining challenges.

Code a Little Bit Every Day

Consistent, daily coding is more effective than cramming all your work into a single session right before a deadline. Regular practice helps reinforce concepts and reduces stress.



Don’t Get Stuck on a Single Issue for Too Long

If you find yourself stuck on a particular problem for more than 15-20 minutes, take a break or move on to another part of the program. This strategy helps maintain productivity and prevents frustration.

Check Your Code Frequently

Testing your code regularly allows you to catch errors early, making debugging easier and more manageable.

Prioritize Readability with Negative Space

Just as in writing, the presentation of your code matters. Proper use of negative space (i.e., spacing, indentation) makes your code easier to read and understand.


US WHEN YOU COME TO RECITATION
imgflip.com