VACATION CLASS

ALGORITHM AND DATA STRUCTURE

Week 2

12 September 2025

YEM DARO

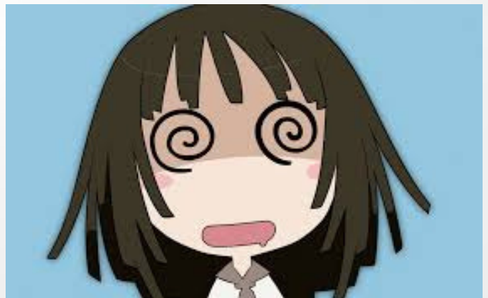
STYLE GUIDE

Rules and best practices for writing code

- How codes should be written
- How codes should be formatted

WHY?

- Improve readability
- Improve teamwork
- NO HEADACHES!!!



STYLE GUIDE

There are two types of people:

```
if (Condition) {
  Statement
  /* ....
   */
}
```

```
if (Condition)
{
  Statement
  /* ....
   */
}
```

STYLE GUIDE

Case Type	Example
Original Variable as String	`some awesome var`
Camel Case	`someAwesomeVar`
Snake Case	`some_awesome_var`
Kebab Case	`some-awesome-var`
Pascal Case	`SomeAwesomeVar`
Upper Case Snake Case	`SOME_AWESOME_VAR`

FUNCTION SCOPE

- GLOBAL SCOPE
- LOCAL SCOPE

*CLOSURE: Inner function uses variables from its outer function

FUNCTION SCOPE

```
a = 10
def outer_function():
    b = 20
    global a
    a = 100
    def inner_function():
        c = 30
        nonlocal b
        b = 200
        print('a value is: ', a)
        print('b value is: ', b)
        print('c value is: ', c)
    inner_function()

outer_function()
```

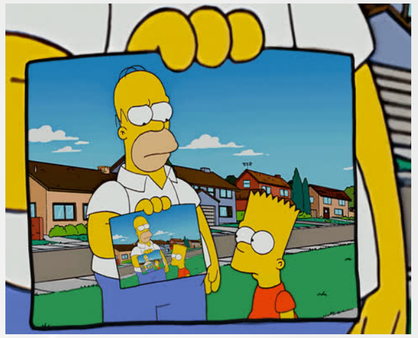
Global Scope

Enclosed Scope

CLOSURE

Local Scope

RECURSION

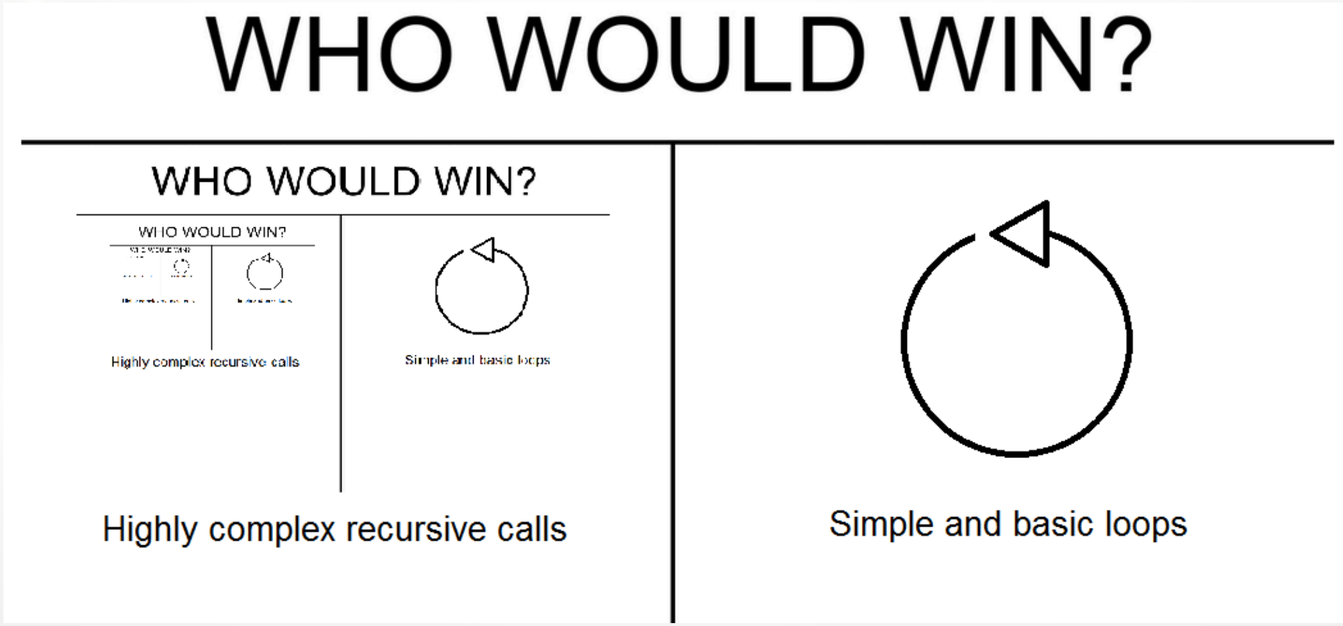


Programming technique
where a function calls itself

- Divide and Conquer
    - reduce a problem to simpler versions of the same problem

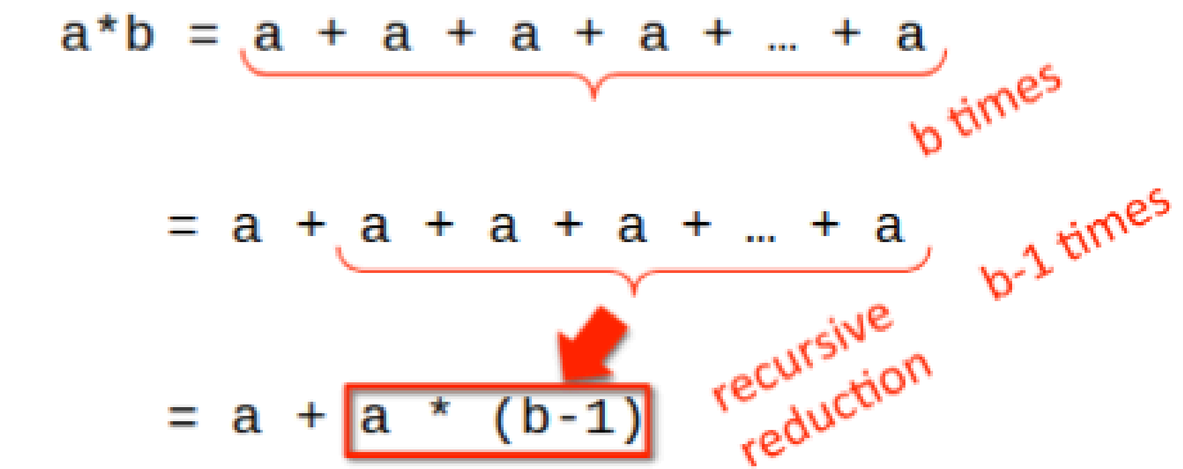
RECURSION

- All loops can be written using recursion
- All recursions can be written using loop



RECURSION

- **recursive step**
  - think how to reduce problem to a **simpler/ smaller version** of same problem
- **base case**
  - keep reducing problem until reach a simple case that can be **solved directly**
  - when b = 1, a*b = a




```
def mult(a, b):
    if b == 1:
        return a
    else:
        return a + mult(a, b-1)
```

base case

recursive step

RECURSION

```
def recursive_sum(n):

    # Base case
    if n == 0:
        return 0

    # Recursive case
    return n + recursive_sum(n - 1)
```

RECURSION

- Some languages don’t have loops like for, while
  - Elixir

Fibonacci (Elixir)

```
defmodule Fib do
  def fib(0), do: 1
  def fib(1), do: 1
  def fib(n), do: fib(n-1) + fib(n-2)
end

IO.puts Fib.fib 6
```

COMPLEXITY

Measure the performance of your code

- **Time complexity:** **Time** taken to run a function given some number of inputs

- **Space complexity:** **Memory** taken to run a function given some number of inputs

COMPLEXITY

SHOULD WE CARE?

COMPLEXITY

YES (maybe)

SHOULD WE CARE?

- Good to know
- Important for competitive programming
    - For scholarships...
- For the love of the game

COMPLEXITY

SIMPLIFICATION EXAMPLES

- drop constants and multiplicative factors
- focus on **dominant terms**

$O(n^2)$: $n^2 + 2n + 2$

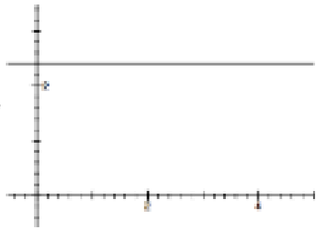
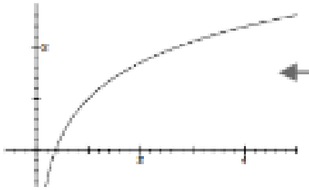
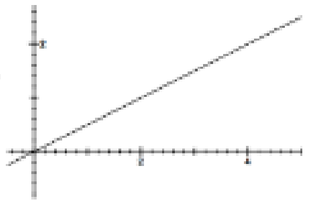
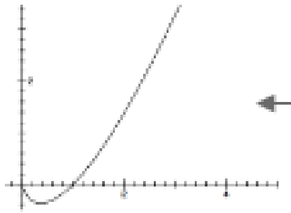
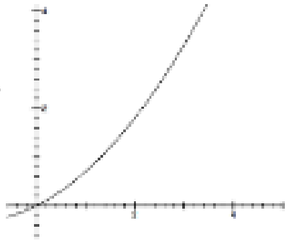
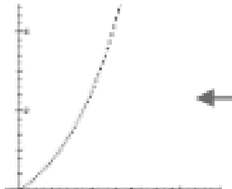
$O(n^2)$: $n^2 + 100000n + 3^{1000}$

$O(n)$: $\log(n) + n + 4$

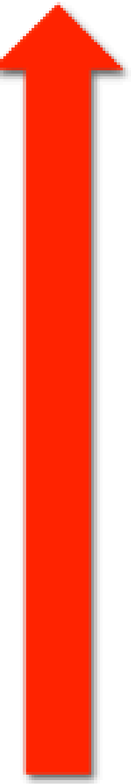
$O(n \log n)$: $0.0001*n*\log(n) + 300n$

$O(3^n)$: $2n^{30} + 3^n$

COMPLEXITY CLASSES ORDERED LOW TO HIGH

$O(1)$	:	constant	
$O(\log n)$	:	 ← logarithmic	
$O(n)$	:	linear	
$O(n \log n)$	:	 ← loglinear	
$O(n^c)$	:	polynomial	
$O(c^n)$	:	 ← exponential	

c is a constant



CLASS	n=10	= 100	= 1000	= 1000000
O(1)	1	1	1	1
O(log n)	1	2	3	6
O(n)	10	100	1000	1000000
O(n log n)	10	200	3000	6000000
O(n^2)	100	10000	1000000	1000000000000
O(2^n)	1024	1267650600228229401496703205376	1071508607186267320948425049060001810561404811705533607443750388370351051124936122493198378815695858127594672917553146825187145285692314043598457757469857480393456777482423098542107460506237114187795418215304647498358194126739876755916554394607706291457119647768654216766042983165262438683720566806937 6	Good luck!!

LEARN MORE

ABSTRACTION AND DECOMPOSITION

https://ocw.mit.edu/courses/...

RECURSION

https://ocw.mit.edu/courses/...

COMPLEXITY

https://ocw.mit.edu/courses/...