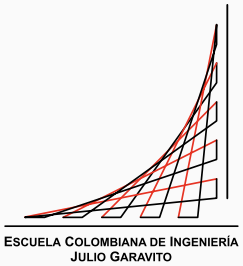

Introducción

Alejandro Anzola Ávila

2024-2

Algoritmos y Estructuras de Datos - Grupo 4

Conozcámonos

- ¿Que esperan de la materia?
- ¿Que carrera estudian?
- ¿Que esperan de su carrera?

Reglas de juego

- ChatGPT

Reglas de juego

- ChatGPT
- Quizzes

Reglas de juego

- ChatGPT
- Quizzes
- Arenas: Virtual Judge

Reglas de juego

- ChatGPT
- Quizzes
- Arenas: Virtual Judge
- Asistencia

Refuerzo

Entrada estándar i

```
1 from sys import stdin
```

Entrada estándar ii

ex.py

```
3 from sys import stdin
2
1 line = stdin.readline().strip()
4 print(">>> {} <<<".format(line))
```

input.txt

```
1 Hola Mundo
```

```
[-] term:///private/var/folders/mw/6v3jq6ws51ndwl_h3hwxh2gm0000gn/T/tmp.Gg7oWH5dFg//9738:/bin/zsh

/var/folders/mw/6v3jq6ws51ndwl_h3hwxh2gm0000gn/T/tmp.Gg7oWH5dFg
❯ python3 ex.py < input.txt
>>> Hola Mundo <<<

/var/folders/mw/6v3jq6ws51ndwl_h3hwxh2gm0000gn/T/tmp.Gg7oWH5dFg
❯ echo "Hola Mundo 2" | python3 ex.py
>>> Hola Mundo 2 <<<
```

Resolvamos un ejercicio ...

Entrada estándar iii

```
1  from sys import stdin
2
3  def main():
4      line = stdin.readline().strip()
5      while len(line) != 0:
6          print(line[:5])
7          line = stdin.readline().strip()
8
9  main()
```

Entrada estándar iv

```
1  from sys import stdin
2
3  def main():
4      nums = stdin.readline().strip().split()
5      for n in nums:
6          n = int(n)
7          print(n*n)
8
9  main()
```

Entrada estándar v

```
1  from sys import stdin
2
3  def main():
4      nums = [int(n) for n in stdin.readline().strip().split()]
5      for n in nums:
6          print(n*n)
7
8  main()
```

Aserciones i

```
def myfunc(n):
    assert isinstance(n, int) # precondicion
    assert n < 10
    assert n > 10, "el numero no cumple la condicion"
    return n*n
```

```
def process_list(n):
    assert isinstance(n, list) and \
        all([isinstance(i, int) for i in n])
    return [i*i for i in n]
```

Aserciones ii

Pueden estar tentados a pensar que esto es suficiente

```
def myfunc(n: int) -> int:
    return n*n
```

```
def process_list(n: list[int]) -> list[int]:
    return [i*i for i in n]
```

pero Python **no** realiza verificaciones de esto en *tiempo de ejecución*.

Aunque si es deseable por **legibilidad**.

Código Limpio (Clean Code)

¿Esto es legible? ¿Se entiende que hace? ¡

```
1  float Q_rsqrt(float number)
2  {
3    long i;
4    float x2, y;
5    const float threehalfs = 1.5F;
6
7    x2 = number * 0.5F;
8    y  = number;
9    i  = * ( long * ) &y;                       // evil floating point bit level h
10   i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
11   y  = * ( float * ) &i;
12   y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
13   // y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be r
14
15   return y;
16 }
```

Figura 1: Código de Quake III Arena (1999).

¿Esto es legible? ¿Se entiende que hace? ii

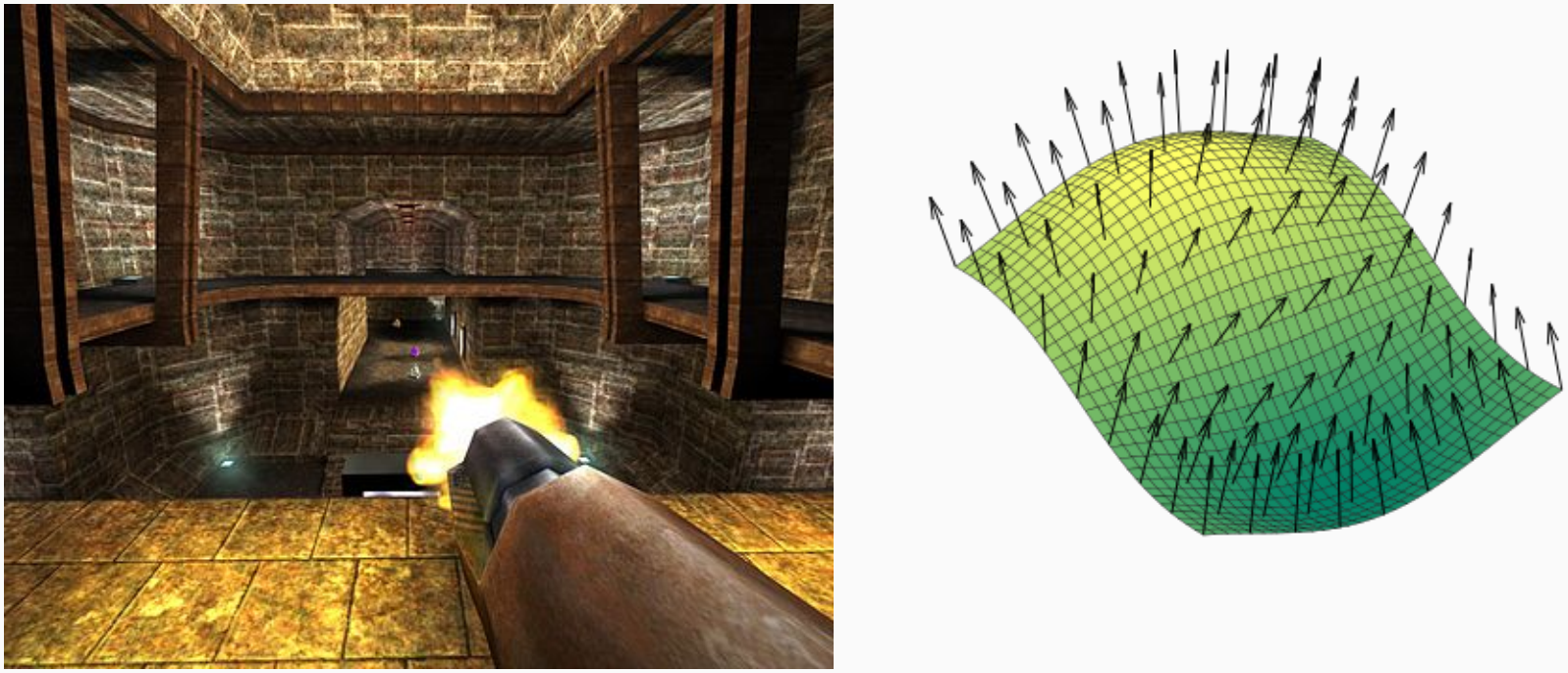

Figura 2: Quake III Arena y normales de superficie.

$$f(x) = \frac{1}{\sqrt{x}}$$

Recomendaciones i

La iteración debería ser limpia

¡No!

```
n, i = [v for v in range(100)], 0
while True:
    i += 1
    if i >= len(n):
        break
```

¡Mejor!

```
n, i = [v for v in range(100)], 0
while i < len(n):
    i += 1
```

Recomendaciones ii

Nombres de variables, funciones, clases o modulos deberían ser descriptivos

¡No!

```
def f(a, i, j, k):
    return a[max(min(i, k), j)]
```

¡Mejor!

```
def get_bounded_element(array, idx, min_idx, max_idx):
    return array[max(min(idx, max_idx), min_idx)]
```

Como regla general, entre mas lejos este la declaración de una variable de su uso, mas se necesita de un buen nombre.

Recomendaciones iii

¡No!

```
def process_data(data):
    if data:
        if isinstance(data, list):
            for item in data:
                if item > 0:
                    print(item)
                else:
                    print("Negative item found")
        else:
            print("Data is not a list")
    else:
        print("No data provided")
```

Recomendaciones iv

¡Mejor!

```python
def process_data(data):
    if not data:
        print("No data provided")
        return

    if not isinstance(data, list):
        print("Data is not a list")
        return

    for item in data:
        if item > 0:
            print(item)
        else:
            print("Negative item found")
```

Recursión

¿Qué es?

Es un método de solución de problemas computacionales, donde la solución depende de soluciones mas pequeñas del mismo problema.

¿Qué es?

Es un método de solución de problemas computacionales, donde la solución depende de soluciones mas pequeñas del mismo problema.

Este necesita de:

- Uno o mas casos bases.

¿Qué es?

Es un método de solución de problemas computacionales, donde la solución depende de soluciones mas pequeñas del mismo problema.

Este necesita de:

- Uno o mas casos bases.
- Uno o mas casos recursivos.

¿Qué es?

Es un método de solución de problemas computacionales, donde la solución depende de soluciones mas pequeñas del mismo problema.

Este necesita de:

- Uno o mas casos bases.
- Uno o mas casos recursivos.
- Cada caso debe ser *mutuamente exclusivo*.

Ejemplos triviales

Asúmase que $n \in \mathbb{N}$

- Fibonacci: $f(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ f(n-1) + f(n-2) & \text{si } n > 1 \end{cases}$
- Factorial: $n! = \begin{cases} 1 & \text{si } n = 0 \\ n(n-1)! & \text{si } n > 0 \end{cases}$
- Potencias: $a^n = \begin{cases} 1 & \text{si } n = 0 \\ a \cdot a^{n-1} & \text{si } n > 0 \end{cases}$

Ejemplos triviales

Asúmase que $n \in \mathbb{N}$

- Fibonacci: $f(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ f(n-1) + f(n-2) & \text{si } n > 1 \end{cases}$
- Factorial: $n! = \begin{cases} 1 & \text{si } n = 0 \\ n(n-1)! & \text{si } n > 0 \end{cases}$
- Potencias: $a^n = \begin{cases} 1 & \text{si } n = 0 \\ a \cdot a^{n-1} & \text{si } n > 0 \end{cases}$

En general, la cantidad de casos base debe al menos coincidir con la cantidad de pasos atrás que se estén dando en los casos recursivos.

Ejercicio

Dado un numero $n \in \mathbb{N}$, diseñe una función f de recurrencia que calcule cuantas veces n se puede dividir entre 2.

Solución

$$f(n) = \begin{cases} 0 & \text{si } n < 2 \vee n \text{ mód } 2 \neq 0 \\ 1 + f(n/2) & \text{si } n \text{ mód } 2 = 0 \end{cases}$$