# ***Noción de Algoritmos Deterministas y No Deterministas***

En un sentido tradicional, un algoritmo es determinista si al conocer las entradas del algoritmo siempre producirá la misma salida, y la máquina interna pasará por la misma secuencia de estados. Mientras que para un algoritmo no determinista con la misma entrada ofrece diversos resultados y por tanto no ofrece una solución única.

# https://upload.wikimedia.org/wikipedia/commons/thumb/1/16/Difference_between_deterministic_and_Nondeterministic.svg/1920px-Difference_between_deterministic_and_Nondeterministic.svg.png

Esquema gráfico de ambos tipos de algoritmos (Eleschinski, 2000)

En este taller se entrega la implementación de **Autómata\_Unidimensional.py**, el cual contiene tal implementación y corresponde a un algoritmo determinístico. A continuación, se explican las nociones de los Autómatas Unidimensionales.

# ***Nociones de Autómatas Unidimensionales***

Los autómatas celulares (AC) son sistemas computacionales abstractos discretos que han sido útiles tanto como modelos generales de complejidad como representaciones más específicas de dinámicas no lineales. Son aplicados en una gran variedad de campos científicos, como:

* Analizadores sintácticos y semánticos
* Motores de cómputo
* Simuladores de sistemas de cómputo discretos
* Modelos conceptuales para estudiar la formación de patrones y la complejidad
* Como modelos originales de física fundamental.
* Representación Fractal

Lectura Recomendada: <https://plato.stanford.edu/entries/cellular-automata/>

Los AC son habitualmente espaciales y temporalmente discretos. Están compuestos por un conjunto finito o numerable de unidades simples y homogéneas, tradicionalmente llamados átomos, células o celdas. En cada unidad de tiempo, PASO n, las celdas crean una instancia de un conjunto finito de estados.

Los AC Evolucionan en paralelo en pasos de tiempo discretos, siguiendo **las funciones de actualización de estado o reglas de transición dinámicas**. La actualización de un estado de celda se obtiene tomando en cuenta los estados de las celdas en su vecindario local; ¡entonces, no hay acciones a distancia!

Los AC son abstractos, es decir, pueden especificarse en términos puramente matemáticos e implementarse en estructuras físicas. Además, los AC son sistemas computacionales, considerando que pueden calcular funciones y resolver problemas algorítmicos. Los AC con reglas adecuadas puede emular una máquina universal de Turing y procesar, cualquier cosa computable. Para más detalles de Máquinas de Turing consultar: <https://en.wikipedia.org/wiki/Turing_machine>

* 1. **Bases para la implementación de un Autómata Celular UNIDIMENSIONAL.**
* Estado Inicial, también conocido como clase inicial: es una lista que representa el estado inicial del sistema de n celdas (átomos o células) en esta Lista Verdadero es 1 (^) y Falso es 0 (\_). A continuación, se presentan 32 celdas [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,**1**,0,0,0, 0,0,0,0,0,0,0,0,0,0, 0,0]



* La regla de transformación: El estado de la celda actual es Verdadero (^) si y sólo si el estado de la celda izquierda o derecha del renglón anterior es Verdadero (^) nunca ambos, en otro caso el estado de la celda es Falso (\_).

Por ejemplo, en un sistema de 3 células,

- 010 produce 101 🡪 \_^\_ produce ^\_^

- 100 produce 010 🡪 ^\_\_ produce \_^\_

- 001 produce 010 🡪 \_\_^ produce \_^\_

- 011 produce 111 🡪 \_^^ produce ^^^

- 111 produce 101 🡪 ^^^ produce ^\_^



Esta es conocida como la 'Hat Rule', para más detalle

sobre esta regla consultar <https://plato.stanford.edu/entries/cellular-automata/>

Existen otras reglas famosas:

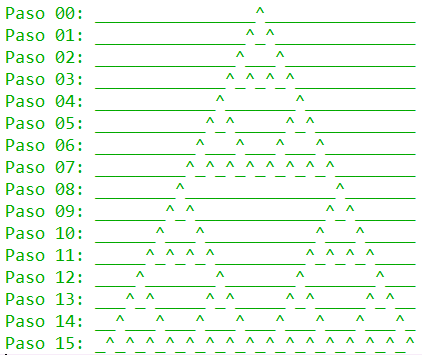
<https://en.wikipedia.org/wiki/Rule_110>

<https://en.wikipedia.org/wiki/Rule_30>

<https://en.wikipedia.org/wiki/Rule_90>

La siguiente es la evolución de la regla en cada paso, con base en el patrón

inicial. Este es el patrón evolutivo resultante de la regla.



# ***Ejercicios***

Los detalles de la implementación los encuentras en el código **Autómata\_Unidimensional.py.** A continuación, los requisitos del ejercicio:

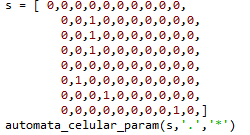
**Parte 1: Algoritmos Determinísticos (Entregar como Automata\_deterministico.py)**

* 1. Analizar, comprender y ejecutar el código del Autómata Unidimensional.
  2. Cree e implemente otro método,

def automata\_celular\_param(SisIni, cFalso, cVerdadero)

donde SisIni es el listado inicial del sistema, es decir, se le puede pasar por parámetro una lista de n posiciones; cVerdadero es el carácter que va a reemplazar cuando en el patrón se encuentre un valor verdadero; mientras que, cFalso el carácter cuando sea falso.

Por ejemplo: una llamada con una lista de 80 células, con un 1 en diferentes posiciones; con punto (.) cuando es falso y asterisco (\*) cuando es verdadero.



Parte de la Impresión:



Obvious Tip: necesitarás n = len(lista) 🡪 obtiene la cantidad de elementos de la lista.

**Parte 2: Algoritmos NO Determinísticos (Entregar como Automata\_no\_deterministico.py)**

* 1. Ahora el objetivo es crear un autómata unidimensional no determinístico (habitualmente se usan para encriptar). Para esto tomando como base el determinístico resuelto de la Parte 1, primero implemente, documente (agregar comentarios explicando) y pruebe dos grupos de reglas adicionales (puede ser de las “famosas” u otras que usted ingenie). Luego, la selección de cada grupo de reglas debe ser aleatoria, para cada iteración. Para cada iteración se debe imprimir cual fue el conjunto de reglas aplicado.

**El taller se puede resolver en pareja, pero cada estudiante carga en la plataforma su correspondiente código.**