Institución Universitaria
Reacreditada en Alta Calidad

TEMA 01
Principios y Técnicas de Conteo

190304003-1

MATEMÁTICAS PARA INFORMÁTICA AVANZADA

ROBERTO RAHAMUT SUTEU
DOCENTE

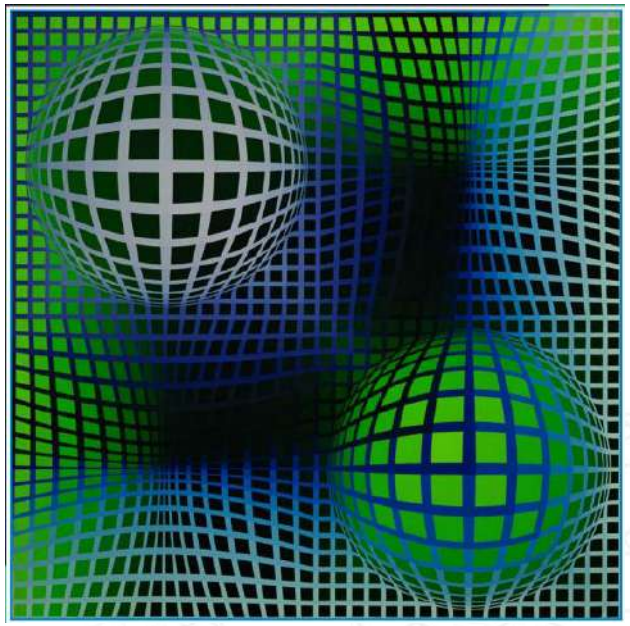
Somos Innovación Tecnológica con Sentido Humano

Alcaldía de Medellín

Institución Universitaria
Acreditada en Alta Calidad

Referencias

- Discrete Mathematics and its applications, Seventh Edition. Kenneth Rosen. McGraw-Hill, 2012.
  - Chapter 2-Sequences and Summations, Chapter 6-Basic Counting, Chapter 8-Advanced Counting Techniques;
- Discrete Mathematics for Computer Scientists, First Edition. Cliff L. Stein et al. Addison-Wesley, 2010
  - Chapter 1-Counting; Chapter 5-Probability;
- Matemáticas Discretas, Sexta Edición. Richard Johnsonbaugh. Prentice Hall, 2013.
  - Capítulo 6-Métodos de conteo;
- https://docs.python.org/3/library/math.html



Institución Universitaria
Acreditada en Alta Calidad

Al final de este tema, ustedes ...

- Emplear métodos de conteo para establecer agrupaciones y enumeraciones de objetos aplicando permutaciones y combinaciones.
- Por medio de técnicas de conteo determinar y contar el número de objetos que tienen ciertas propiedades específicas.
- Aprovechar técnicas de conteo para evaluar la complejidad computacional de un algoritmo
- Diferenciar entre las técnicas de conteo y sus posibles casos de aplicación.
- Codificar un programa computacional con las diferentes técnicas de conteo.

Institución Universitaria
Acreditada en Alta Calidad

Objetivos

- Principios básicos de Conteo
  - Reglas del producto o de la multiplicación
  - Regla de la suma o de la adición
- Técnicas avanzadas de Conteo
  - Principios de inclusión y exclusión
  - Progresiones aritméticas y geométricas
- Permutaciones y combinaciones
- Relaciones de recurrencia

Institución Universitaria
Acreditada en Alta Calidad

"¡Hay 10 tipos de personas en el mundo!
Las que entienden binario y las que no."

TEORÍA:
Principios básicos y técnicas avanzadas de Conteo

"A veces, es la gente de la que nadie imagina nada,
la que hace cosas que nadie puede imaginar."

Alan Turing

Definiciones sobre conteo

- El **conteo en matemáticas** se refiere a la determinación del número de elementos en un conjunto finito de objetos.
- En informática, este concepto es clave en el análisis de algoritmos, la probabilidad y la teoría de la computación.
- Según Richard Johnsonbaugh:
  - **Reglas del producto o la multiplicación**. **Product Rule**

    Si una actividad se puede construir en t pasos sucesivos y el paso 1 se puede hacer de n_1 maneras, el paso 2 se puede realizar de n_2 maneras, . . . , y el paso t de n_t maneras, entonces el número de actividades posibles diferentes.

$$n_1 \cdot n_2 \cdots nt = \prod_{i=1}^{t} n_i$$

  - **Reglas de la suma o de la adición**. **Sum Rule**

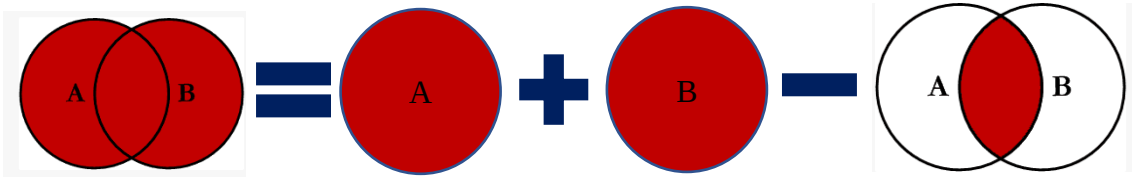
    El tamaño de una unión de una familia de conjuntos mutuamente disjuntos es la suma de los tamaños de los conjuntos. En otras palabras, si S1, S2, ...,Sn son conjuntos disjuntos, entonces.

$$|S_1 \cup S_2 \cup \cdots \cup S_n| = |S_1| + |S_2| + \cdots + |S_n| \;\gg\; \left|\bigcup_{i=1}^{n} S_i\right| = \sum_{i=1}^{n} |S_i|$$

Principios de inclusión y exclusión

- El principio de inclusión y exclusión es un método fundamental en combinatoria utilizado para contar el número de elementos en la unión de conjuntos superpuestos sin contar duplicados. Este principio se utiliza en teoría de conjuntos, probabilidad y análisis de algoritmos.
- Formalmente, para dos conjuntos A y B, el número de elementos en su unión se expresa como:

$$|A \cup B| = |A| + |B| - |A \cap B|$$



- Para tres conjuntos: A, B, C,

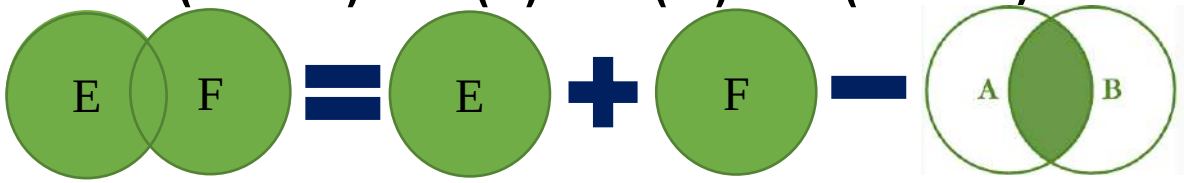
$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

Principios de inclusión y exclusión para Probabilidad

The Principle of Inclusion–Exclusion for probability

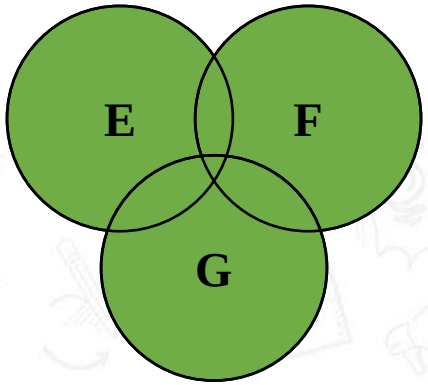
- Probabilidad de una unión de dos eventos.

$$P(E \cup F) = P(E) + P(F) - P(E \cap F)$$



- Probabilidad de una unión de tres eventos.

$$P(E \cup F \cup G) = P(E) + P(F) + P(G) - P(E \cap F) - P(E \cap G) - P(F \cap G) + P(E \cap F \cap G)$$



Institución Universitaria
Acreditada en Alta Calidad

Progresiones Aritméticas y Geométricas

Las progresiones o secuencias, tanto las geométricas como las aritméticas, son patrones matemáticos fundamentales que tienen múltiples aplicaciones prácticas.

Progresión Aritmética

Arithmetic Progression

Donde cada término se suma una diferencia constante:

- Progresión de salarios o subscripciones
- Numeración de asientos o lockers
- Escalas de temperatura
- Distancias en movimiento uniforme
- Calendarios y fechas
- Patrones en programación (índices de arrays)

Progresión Geométrica

Geometric Progression

Donde cada término se multiplica por una razón constante:

- Crecimiento poblacional
- Interés compuesto en finanzas
- Crecimiento exponencial (como en propagación de virus)
- Decaimiento radioactivo
- Escalas musicales
- Fractales y patrones en la naturaleza

An *arithmetic progression* is a sequence of the form

$$a, a+d, a+2d, \ldots, a+nd, \ldots$$

where the *initial term* a and the *common difference* d are real numbers.

A *sequence* is a function from a subset of the set of integers (usually either the set $\{0, 1, 2, \ldots\}$ or the set $\{1, 2, 3, \ldots\}$) to a set S. We use the notation a_n to denote the image of the integer n. We call a_n a *term* of the sequence.

A *geometric progression* is a sequence of the form

$$a, ar, ar^2, \ldots, ar^n, \ldots$$

where the *initial term* a and the *common ratio* r are real numbers.

Institución Universitaria
Acreditada en Alta Calidad

Progresiones Aritméticas y Geométricas

Progresión Aritmética

Una sucesión de números en la que cada término se obtiene sumando una constante al anterior: $a_n = a_1 + d \cdot (n - 1)$

diferencia

Ejemplo en Python:

Arithmetic Progression

```
def suma_progresion_aritmetica(n, a1, d):
    return (n / 2) * (2 * a1 + (n - 1) * d)

print(suma_progresion_aritmetica(10, 2, 3))
```

```
C:/Python/python.exe progresion_aritmetica.py
155.0
```

$$a_1$$
$$a_2 = a_1 + d$$
$$a_3 = a_2 + d$$
$$a_4 = a_3 + d$$
$$a_5 = a_4 + d$$
$$a_6 = a_5 + d$$
$$\dots$$
$$a_{n+1} = a_n + d$$

An *arithmetic progression* is a sequence of the form

$$a, a + d, a + 2d, \dots, a + nd, \dots$$

where the *initial term* a and the *common difference* d are real numbers.

A *sequence* is a function from a subset of the set of integers (usually either the set $\{0, 1, 2, \dots\}$ or the set $\{1, 2, 3, \dots\}$) to a set S. We use the notation a_n to denote the image of the integer n. We call a_n a *term* of the sequence.

Progresión Geométrica

Una sucesión en la que cada término se obtiene multiplicando el anterior por una constante: $a_n = a_1 \cdot r^{n-1}$

razón

Ejemplo en Python:

Geometric Progression

```
def suma_progresion_geometrica(n, a1, r):
    return a1 + r ** (n - 1)

print(suma_progresion_geometrica(5, 2, 3))
```

```
C:/Python/python.exe progresion_geometrica.py
83
```

$$a_1$$
$$a_2 = a_1 \cdot r$$
$$a_3 = a_2 \cdot r$$
$$a_4 = a_3 \cdot r$$
$$\dots$$

A *geometric progression* is a sequence of the form

$$a, ar, ar^2, \dots, ar^n, \dots$$

where the *initial term* a and the *common ratio* r are real numbers.

Notación Factorial
n!

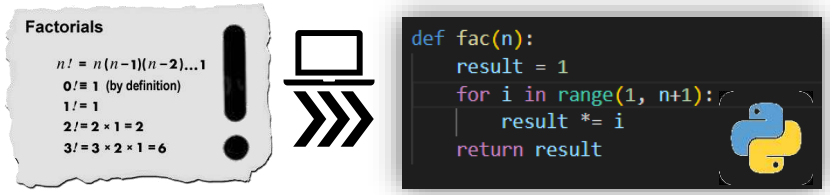
- Se usa la notación **n!** para denotar el producto de los enteros positivos desde **1** hasta **n**.

n! = 1 x 2 x 3 x 4 x… x n

0! = 1

1! = 1

n! = (n – 1)! x n



- La definición de **n!** (ene factorial) es el producto desde la **unidad** hasta el valor que tiene **n**.
- Ejemplos:

n! = 1 x 2 x 3 x 4 x… x n	n! = (n – 1)! x n
2! = 1 x 2 = 2	2! = (2 – 1)! x 2 = 1! x 2 = 1 x 2 = 2
3! = 1 x 2 x 3 = 6	3! = (3 – 1)! x 3 = 2! x 3 = 2 x 3 = 6
5! = 1 x 2 x 3 x 4 x 5 = 120	5! = (5 – 1)! x 5 = 4! x 5 = 24 x 5 = 120
6! = 1 x 2 x 3 x 4 x….x 6 = 720	6! = (6 – 1)! x 6 = 5! X 6 = 120 x 6 = 720
8! = 1 x 2 x 3 x 4 x……..x 8 = 40320	8! = (8 – 1)! x 10 = 7! x 8 = 5040 x 8 = 40320
10! = 1 x 2 x 3 x 4 x……..x 10 = 3628800	10! = (10 – 1)! x 10 = 9! x 10 = 362880 x 10 = 3628800

Institución Universitaria
Acreditada en Alta Calidad

Permutaciones y Combinaciones

Muchos problemas de conteo se pueden resolver hallando la cantidad de maneras de ordenar una cantidad específica de elementos distintos de un conjunto de un tamaño particular, donde el orden de estos elementos es importante. Muchos otros problemas de conteo se pueden resolver hallando la cantidad de maneras de seleccionar un particular número de elementos de un conjunto de un tamaño particular, donde el orden de los elementos seleccionados no es importante.

Permutaciones

Permutations

Según K. Rosen, una permutación de un conjunto de objetos distintos es una disposición ordenada de estos objetos. También nos interesan las disposiciones ordenadas de algunos de los elementos de un conjunto. Una disposición ordenada de r elementos de un conjunto se denomina r-permutación.

Una permutación es un arreglo ordenado de elementos. Se calcula como:

$$P(n,r) = \frac{n!}{(n-r)!}$$

Ejemplo: ¿Cuántas formas hay de ordenar 3 personas en una fila, si hay 5 personas disponibles?

$$P(5,3) = \frac{5!}{(5-3)!} = \frac{5!}{2!} = 60$$

Combinaciones

Combinations

Según K. Rosen, una combinación de elementos de un conjunto es una selección desordenada de elementos del conjunto (r-combinación, r-elementos).

Las combinaciones son selecciones de elementos sin importar el orden:

$$C(n,r) = \frac{n!}{r!\,(n-r)!}$$

Ejemplo: Si hay 10 candidatos y debemos elegir 3 sin importar el orden:

$$C(10,3) = \frac{10!}{3!(10-3)!} = 120$$

Permutaciones y Combinaciones

Permutaciones

Permutations

Una permutación es un arreglo ordenado de elementos. Se calcula como:

$$P(n,r) = \frac{n!}{(n-r)!}$$

Ejemplo: ¿Cuántas formas hay de ordenar 3 personas en una fila, si hay 10 personas disponibles?

$$P(10,3) = \frac{10!}{(10-7)!} = \frac{10!}{3!} = 720$$

permutaciones.py > ...

```python
from math import factorial

def permutaciones(n, r):
    return factorial(n) // factorial(n - r)
print(permutaciones(5, 3))
```

C:/Python/python.exe permutaciones.py
60

Combinaciones

Combinations

Una selección de objetos que no toma en cuenta el orden se llama combinación:

$$C(n,r) = \frac{n!}{r!\,(n-r)!}$$

Ejemplo: Si hay 10 candidatos y debemos elegir 3 sin importar el orden:

$$C(10,3) = \frac{10!}{3!(10-3)!} = 120$$

combinaciones.py

```python
from math import comb
print(comb(10, 3))
```

C:/Python/python.exe combinaciones.py
120

Relaciones de Recurrencia

Recurrence Relations

- Las relaciones de recurrencia son útiles en ciertos problemas de conteo.
- Una relación de recurrencia relaciona el n-ésimo elemento de una sucesión con sus predecesores. Como las relaciones de recurrencia tienen una relación cercana con los algoritmos recursivos, surgen de manera natural en el análisis de la recursividad estructuras de datos y análisis de la complejidad computacional.
- Ejemplo:

  Considere las siguientes instrucciones para generar una sucesión:

  1. Iniciar con 5.
  2. Dado cualquier término, sume 3 para obtener el siguiente.

  Si se listan los términos de la sucesión, se obtiene 5, 8, 11, 14, 17, ...

  El primer término es 5 por la instrucción 1. El segundo término es 8 porque la instrucción 2 dice que se sume 3 a 5 para obtener el siguiente término. El tercer término es 11 porque la instrucción 2 dice que se sume 3 a 8 para obtener el siguiente término. Si se siguen las instrucciones 1 y 2, se puede calcular cualquier término de la sucesión. Las instrucciones 1 y 2 no dan una fórmula explícita para el n-ésimo término de la sucesión en el sentido de proporcionar una fórmula en la que se pueda “sustituir n” para obtener el valor del n-ésimo término, sino que al calcular término por término en algún momento se podrá obtener cualquier término de la sucesión. se denota por $a_1, a_2, \ldots$, se puede enunciar de nuevo la instrucción 1 como $a_1 = 5$ y la instrucción 2 se puede establecer como $a_n = a_{n-1} + 3$, $n \geq 2$, luego haciendo $n = 2$ se obtiene $a_2 = a_1 + 3 = 8$.

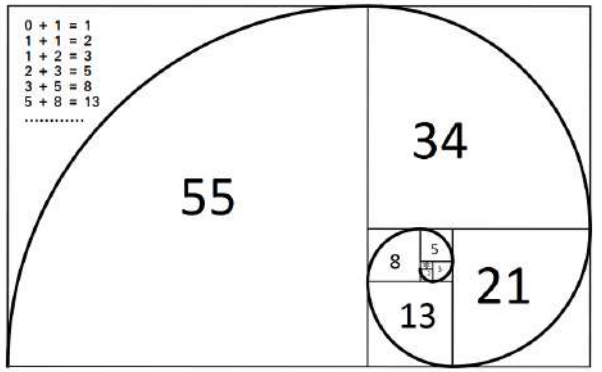
FIBONACCI

0, 1, 1, 2, 3, 5,

8, 13, 21, 34, 55,

89, 144 …

Secuencia de Fibonacci



- La secuencia de Fibonacci es definida por la relación de recurrencia:

$$fn = fn-1 + fn-2 \text{ , n} \geq 3$$

y las condiciones iniciales

$$f_1 = 1, \qquad f_2 = 1.$$

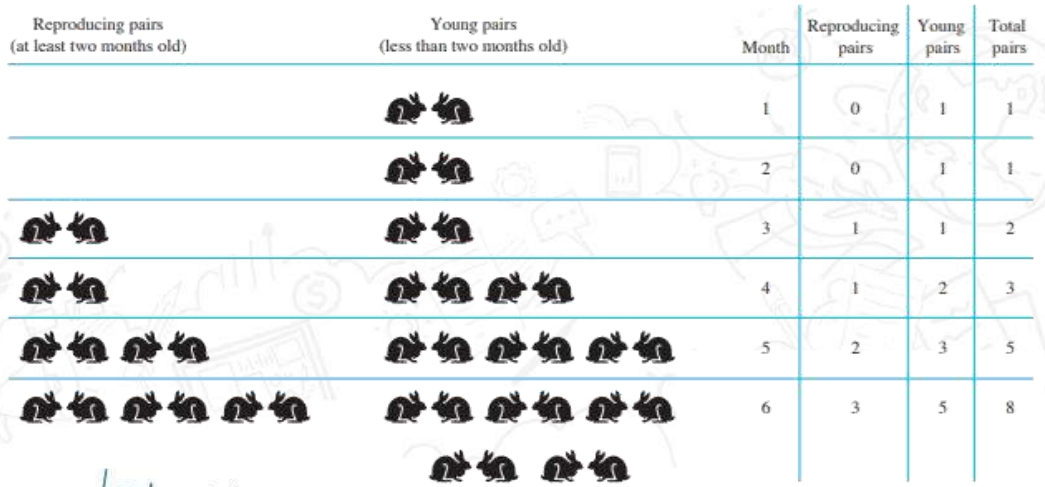
- Ejemplo:

Supongamos que una pareja de conejos recién nacidos tarda un mes en madurar y, a partir del segundo mes, cada pareja produce otra pareja cada mes. Si todos los conejos sobreviven, ¿cuántos conejos habrá después de n meses? Esta situación se modela perfectamente con la secuencia de Fibonacci.

python examples > fibonacci.py > ...

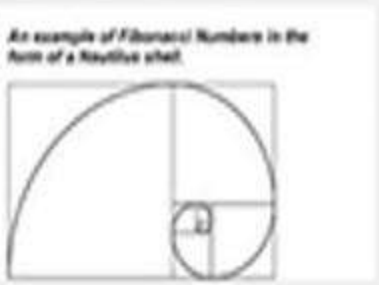
```python
# Implementación recursiva de Fibonacci
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

print(fibonacci(10))  # Calcula el término 10 de Fibonacci
```

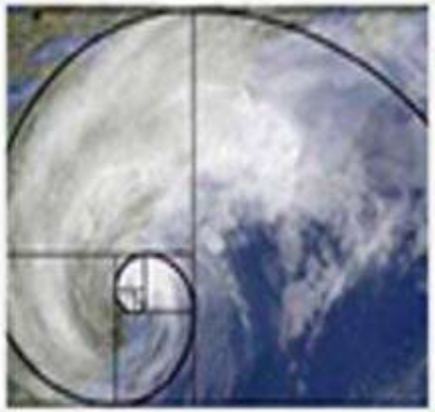
C:/Python/python.exe fibonacci.py
55

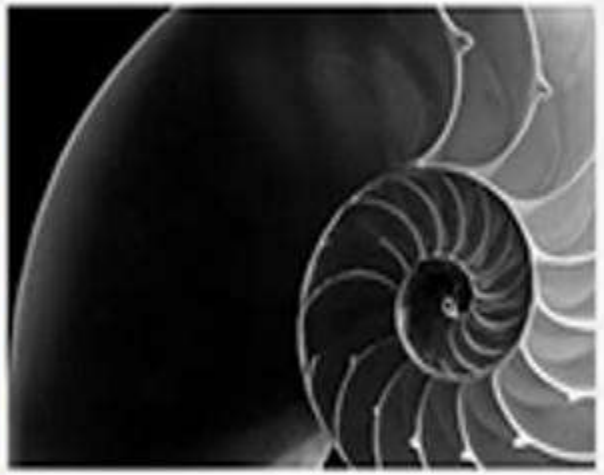


Reproducing pairs (at least two months old)	Young pairs (less than two months old)	Month	Reproducing pairs	Young pairs	Total pairs
	🐇🐇	1	0	1	1
	🐇🐇	2	0	1	1
🐇🐇	🐇🐇	3	1	1	2
🐇🐇	🐇🐇 🐇🐇	4	1	2	3
🐇🐇 🐇🐇	🐇🐇 🐇🐇 🐇🐇	5	2	3	5
🐇🐇 🐇🐇 🐇🐇	🐇🐇 🐇🐇 🐇🐇 🐇🐇	6	3	5	8

Rabbits on an Island.


An example of Fibonacci Numbers in the form of a Nautilus shell.





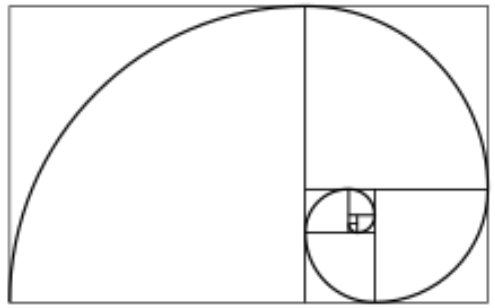
Institución Universitaria
Acreditada en Alta Calidad

Leonardo de Pisa Fibonacci

De manera explícita, tendríamos que es: 1, 1, 2, 3, 5, 8, 13, 21, 34… La descubrió, en el siglo XIII, el matemático italiano Leonardo de Pisa, más conocido como Fibonacci. Su aprendizaje se produjo gracias a los viajes que hacía junto a su padre, que era comerciante.

Número áureo

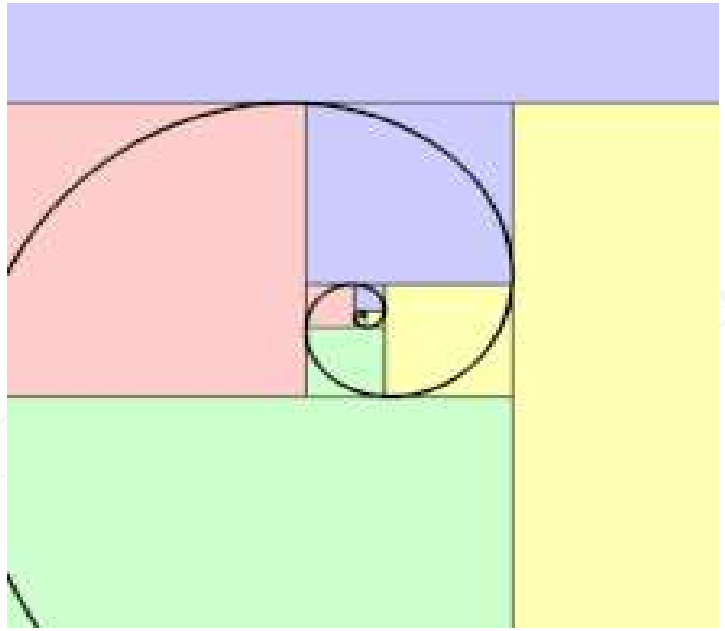
Si divides cualquier número en la secuencia de Fibonacci por el anterior, por ejemplo, 55/34, o 21/13, y la respuesta siempre es cercana a 1.61803. Y es por eso que la secuencia de Fibonacci también es conocida como la secuencia dorada, pues ese 1,61803 es lo que se conoce como el número áureo.



Una espiral de Fibonacci se aproxima a la espiral dorada; cuando se inscribe en cuadrados cuyos lados responden a la sucesión de Fibonacci:
1, 1, 2, 3, 5, 8, 13, 21 y 34.

Aproximaciones a la espiral dorada

Existen aproximaciones a la espiral dorada, que no son iguales. Este tipo de espirales, a menudo se confunden con la espiral dorada. Un ejemplo es la espiral de Fibonacci que resulta ser una aproximación a la espiral dorada..



Espiral dorada

La Espiral dorada (denominada también espiral áurea) es una espiral logarítmica asociada a las propiedades geométricas del rectángulo dorado. La razón de crecimiento es Φ, es decir la razón dorada o número áureo. Esta espiral aparece representada en diversas figuras de la naturaleza (planta, galaxias espirales), así como en el arte.

Institución Universitaria
Acreditada en Alta Calidad

– *"¿Por qué los matemáticos nunca confunden combinaciones con permutaciones?"*
"Porque el orden sí importa... cuando están esperando en la fila de la cafetería."

EJEMPLOS PRÁCTICOS

"Si una máquina puede responder preguntas de manera indistinguible de un humano, ¿cómo sabrías si es consciente?" Alan Turing

Ejemplo #1 de caso de conteo

Regla del Producto o Multiplicación

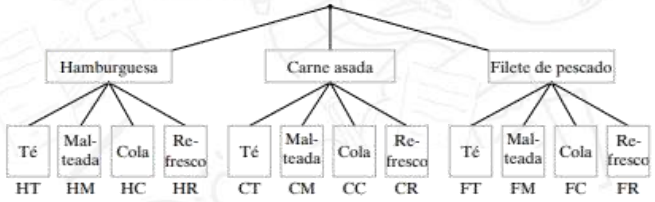
$$\prod_{i=1}^{t} n_i$$

Product Rule

En un restaurante, el menú de comida rápida, contiene dos entremeses, tres platos fuertes y cuatro bebidas. ¿Qué comidas diferentes se pueden formar por un plato fuerte y una bebida? ¿Cuántas comidas tendrían usando los tres platos?

- La comida que consiste en un plato fuerte cuya primera letra es X y una bebida cuya primera letra es Y se denota por XY. Por ejemplo, CR se refiere a una comida que consiste en carne asada y refresco de sabor.

1. Observe que se tienen tres platos fuertes y cuatro bebidas, X=3 y Y=4 lo que da un resultado de : $3 \cdot 4 = 12$.
   - Si se listan todas las comidas posibles que consisten en un plato fuerte y una bebida HT, HM, HC, HR, CT, CM, CC, CR, FT, FM, FC, FR, se ve que existen 12 comidas diferentes.
2. En este ejemplo se encuentra que el número total de comidas es igual al producto de los números de cada categoría: $2 \cdot 3 \cdot 4 = 24$.



Institución Universitaria
Acreditada en Alta Calidad

Código Ejemplo #1
Regla de la Multiplicación

$$\prod_{i=1}^{t} n_i$$

Product Rule

```python
# Definir las opciones del menú
entremes = ["N", "S"]  # Nachos, Salami
platos_fuertes = ["H", "C", "F"]  # Hamburgesa, Carne, Filete
bebidas = ["T", "M", "C", "R"]   # Té, Malteada, Cerveza, Refresco

#1 Generar todas las comidas posibles de un plato fuerte con una bebida
comidas_1 = [pf + b for pf in platos_fuertes for b in bebidas]

#2 Generar todas las comidas posibles con los tres platos
comidas_2 = [e + pf + b for e in entremes for pf in platos_fuertes for b in bebidas]

# Mostrar resultados de comidas diferentes y la cantidad total posible
print("Comidas posibles con plato fuerte y bebidas:", comidas_1)
print("Total de comidas con los tres platos:", len(comidas_2))
```

C:/Python/python.exe ejemplo #1.py

Comidas posibles con plato fuerte y bebidas: ['HT', 'HM', 'HC', 'HR', 'CT', 'CM', 'CC', 'CR', 'FT', 'FM', 'FC', 'FR']

Total de comidas con los tres platos: 24

ENTREMÉS

Nachos	2.15
Salami especial	1.90

PLATO FUERTE

Hamburguesa	3.25
Carne asada	3.65
Filete de pescado	3.15

BEBIDAS

Té	.70
Malteada	.85
Cola	.75
Refresco de sabor	.75

Existen 24 comidas posibles de un entremés, un plato fuerte y una bebida: NHT, NHM, NHC, NHR, NCT, NCM, NCC, NCR, NFT, NFM, NFC, NFR, SHT, SHM, SHC, SHR, SCT, SCM, SCC, SCR, SFT, SFM, SFC, SFR.

Playgrounds:
www.online-python.com

Ejemplo #2 de caso de conteo

Regla de la Multiplicación

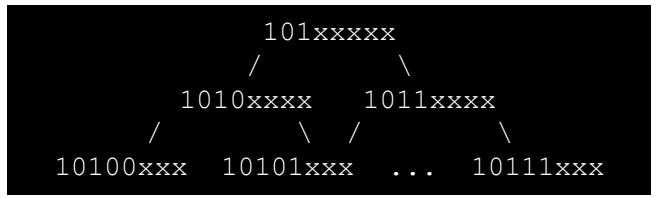
$$\prod_{i=1}^{t} n_i$$

Product Rule

- La regla del producto para demostrar que un conjunto $\{x_1, \ldots, x_n\}$ de n elementos tiene 2^n subconjuntos.
- Un subconjunto se construye en n pasos sucesivos: se elige o no se elige x_1 ; se elige o no x_2 ; . . . ; se elige o no x_n . Cada paso se puede realizar de dos maneras.
- Entonces, el número de subconjuntos posibles es:

$$\underbrace{2 \cdot 2 \cdots 2}_{\text{n factores}} = 2^n.$$

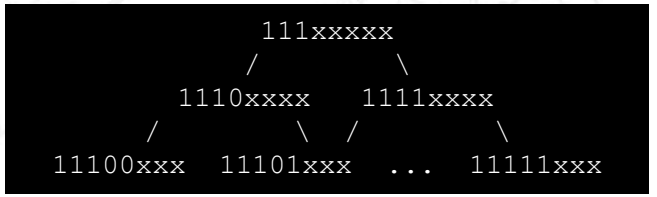
Estructura del árbol para las cadenas que comienzan con 101



- ¿Cuántas cadenas de ocho bits comienzan con 101 o con 111?
  - Una cadena de ocho bits que comienza con 101 se puede construir en cinco pasos sucesivos: se selecciona el cuarto bit; se selecciona el quinto bit; . . . ; se selecciona el octavo bit.
  - Como cada uno de los cinco bits se puede seleccionar de dos maneras, por el principio de la multiplicación o regla del producto :

$$2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 2^5 = 32$$

Estructura del árbol para las cadenas que comienzan con 111



Ejemplo #2 de caso de conteo

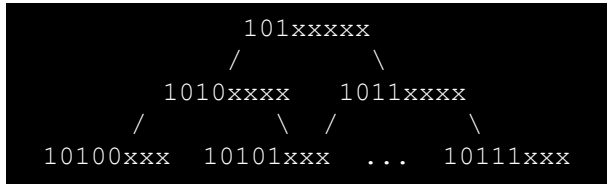
Regla de la Multiplicación

$$\prod_{i=1}^{t} n_i$$

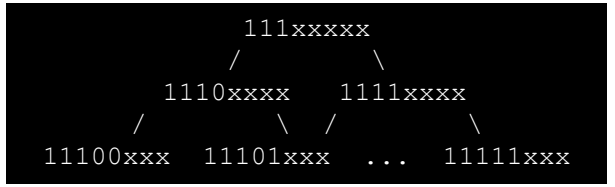
Product Rule

- Contando las diferentes cadenas posibles, se tiene:
  - Total de cadenas que comienzan con 101 ⟶ 32
  - Total de cadenas que comienzan con 111 ⟶ 32

Estructura del árbol para las cadenas que comienzan con 101



Estructura del árbol para las cadenas que comienzan con 111



- Existen 32 cadenas de ocho bits que comienzan con 101. El mismo argumento se utiliza para mostrar que existen 32 cadenas de ocho bits que comienzan con 111.
- Como hay 32 cadenas de 8 bits que comienzan con 101 y 32 cadenas de 8 bits que comienzan con 111, existen 32 + 32 = 64 cadenas de 8 bits que comienzan con 101 o 111.
- Se sumaron los números de cadenas de 8 bits (32 y 32) de cada tipo para determinar el resultado final. Con esto se da paso a la regla o principio de la suma que dice cuándo sumar para calcular el número total de posibilidades.

Institución Universitaria
Acreditada en Alta Calidad

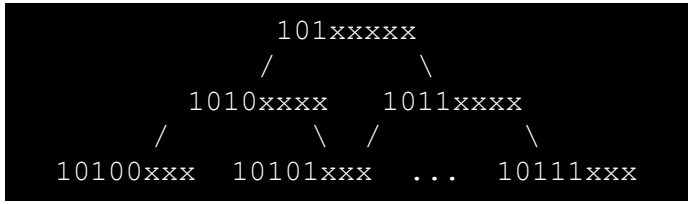
Código Ejemplo #2
Regla de la Multiplicación

$$\prod_{i=1}^{t} n_i$$

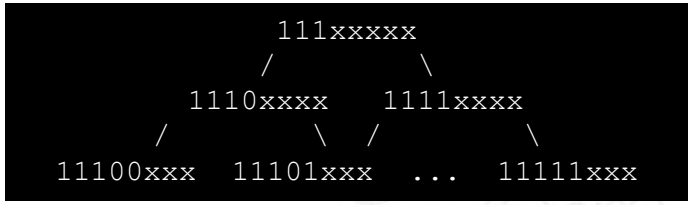
Product Rule

```python
def contar_cadenas():
    # Definir la cantidad de bits libres después del prefijo
    bits_libres = 5  # 8 bits totales - 3 bits fijos

    # Calcular las diferentes cadenas posibles para cada prefijo
    cadenas_101 = 2 ** bits_libres
    cadenas_111 = 2 ** bits_libres

    # Sumar ambos casos (no hay intersección)
    total_cadenas_101y111 = cadenas_101 + cadenas_111

    return total_cadenas_101y111

# Ejecutar la función e imprimir el resultado
print("Total de cadenas de ocho bits que comienzan con 101 o 111:", contar_cadenas())
```

Estructura del árbol para las cadenas que comienzan con 101



Estructura del árbol para las cadenas que comienzan con 111



C:/Python/python.exe ejemplo #2.py

Total de cadenas de ocho bits que comienzan con 101 o 111: 64

Kenneth Rosen

THE PRODUCT RULE Suppose that a procedure can be broken down into a sequence of two tasks. If there are n_1 ways to do the first task and for each of these ways of doing the first task, there are n_2 ways to do the second task, then there are $n_1 n_2$ ways to do the procedure.

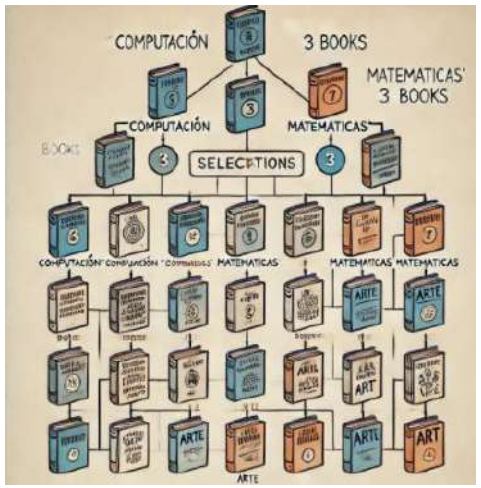
Playgrounds:
www.online-python.com

Ejemplo #3 de caso de conteo

Regla de la Multiplicación y de la Suma

Sum Rule

Product Rule



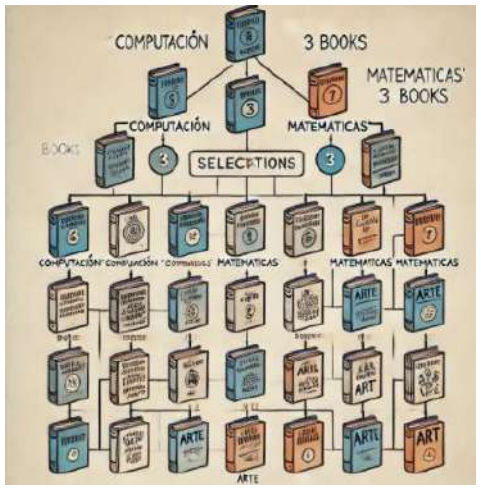
Recapitulando, según Cliff L. Stein, el tamaño de una unión de una familia de conjuntos finitos mutuamente disjuntos es la suma de los tamaños de los conjuntos.

En una librería, ¿De cuántas maneras se pueden seleccionar dos libros de temas diferentes entre cinco libros de computación distintos, tres libros de matemáticas diferentes y dos libros de arte distintos?

- Mediante el principio de la multiplicación, se encuentra que podemos seleccionar dos libros, uno de computación y uno de matemáticas, de $5 \cdot 3 = 15$ maneras.
- De forma similar, podemos seleccionar dos libros, uno de computación y uno de arte, de $5 \cdot 2 = 10$ maneras, y podemos seleccionar dos libros, uno de matemáticas y uno de arte, de $3 \cdot 2 = 6$ maneras.
- Como estos conjuntos de selecciones son ajenas por pares, podemos usar el principio de la suma para concluir que existen $15 + 10 + 6 = 31$ maneras de seleccionar dos libros con temas diferentes entre los libros de computación, matemáticas y arte.

Código Ejemplo #3
Regla de la Suma


```python
def contar_selecciones():
    # Cantidad de libros en cada categoría
    computacion = 5
    matematicas = 3
    arte = 2

    # Calcular diferentes libros posibles
    libros_cm = computacion * matematicas # Computación y Matemáticas
    libros_ca = computacion * arte        # Computación y Arte
    libros_ma = matematicas * arte        # Matemáticas y Arte

    # Sumar todas las posibildades de diferentes libros
    total_libros = libros_cm + libros_ca + libros_ma

    return total_libros

# Ejecutar la función e imprimir el resultado
print("Total de maneras de seleccionar dos libros de temas diferentes:", contar_selecciones())
```



$$\sum_{i=1}^{n} |X_i|$$

C:/Python/python.exe ejemplo #3.py

Total de maneras de seleccionar dos libros de temas diferentes: 31

$$\prod_{i=1}^{t} n_i$$

Playgrounds:
www.online-python.com

Ejemplo #4 de caso de conteo

Regla de la Suma

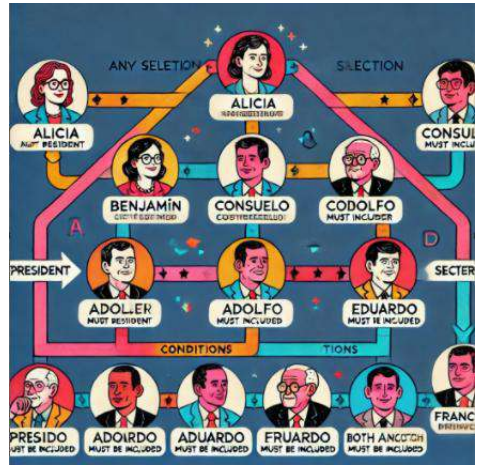
Un comité de seis personas, compuesto por Alicia, Benjamín, Consuelo, Adolfo, Eduardo y Francisco, debe seleccionar un presidente, secretario y tesorero.

a) ¿De cuántas maneras pueden hacer esto?

b) ¿De cuántas maneras pueden hacerlo si Alicia o Benjamín debe ser el presidente?

c) ¿De cuántas maneras pueden hacerlo si Eduardo debe ocupar uno de los puestos?

d) ¿De cuántas maneras pueden hacerlo si tanto Adolfo como Francisco deben ocupar un puesto?



- A) Se usa el principio de la multiplicación. Se selecciona a los directivos en tres pasos sucesivos: se elige el presidente, se elige el secretario, se elige el tesorero. El presidente se puede elegir de seis maneras. Una vez elegido, el secretario se puede elegir de cinco maneras. Después de elegir al presidente y el secretario, el tesorero se puede seleccionar de cuatro maneras. Por lo tanto, el número total de posibilidades es $6 \cdot 5 \cdot 4 = 120$.
- B) Con un argumento como el del inciso a), si Alicia es presidente, se tienen $5 \cdot 4 = 20$ maneras de seleccionar los puestos restantes. De igual manera, si Benjamín es presidente, existen 20 maneras de seleccionar los puestos restantes. Como estos casos son ajenos, por el principio de la suma, existen $20 + 20 = 40$ posibilidades.

$$\sum_{i=1}^{n} |X_i|$$

$$\prod_{i=1}^{t} n_i$$

Ejemplo #4 de caso de conteo

Regla de la Suma

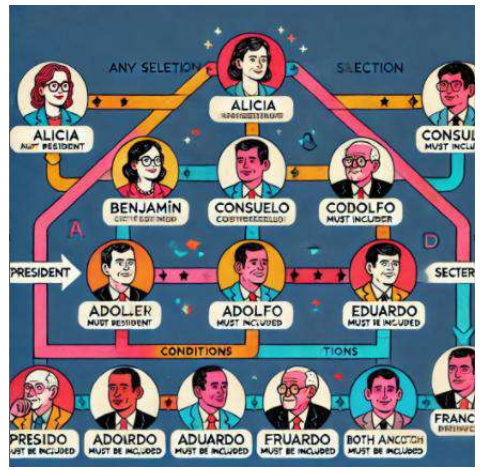
Un comité de seis personas, compuesto por Alicia, Benjamín, Consuelo, Adolfo, Eduardo y Francisco, debe seleccionar un presidente, secretario y tesorero.

a) ¿De cuántas maneras pueden hacer esto?

b) ¿De cuántas maneras pueden hacerlo si Alicia o Benjamín debe ser el presidente?

c) ¿De cuántas maneras pueden hacerlo si Eduardo debe ocupar uno de los puestos?

d) ¿De cuántas maneras pueden hacerlo si tanto Adolfo como Francisco deben ocupar un puesto?



- C) Considere que la actividad de asignar a Eduardo y otros dos para los puestos está compuesta por tres pasos sucesivos: asignar a Eduardo a un puesto, asignar el puesto más alto que queda, asignar el último puesto. Existen tres maneras de asignar a Eduardo a un puesto. Una vez asignado, existen cinco maneras de asignar el puesto más alto que queda. Una vez asignados estos dos puestos, hay cuatro maneras de asignar el último puesto. Por el principio de la multiplicación, existen $3 \cdot 5 \cdot 4 = 60$ posibilidades.
- D) Considere que la actividad de asignar a Adolfo, Francisco y otra persona a los puestos se compone de tres pasos sucesivos: asignar a Adolfo, a Francisco y el puesto que queda. Existen tres maneras de asignar a Adolfo. Una vez asignado, hay dos maneras de asignar a Francisco. Una vez asignados Adolfo y Francisco, hay cuatro maneras de asignar el último puesto. Por el principio de la multiplicación, existen $3 \cdot 2 \cdot 4 = 24$ posibilidades.

$$\sum_{i=1}^{n} |X_i|$$

$$\prod_{i=1}^{t} n_i$$

Institución Universitaria
Acreditada en Alta Calidad

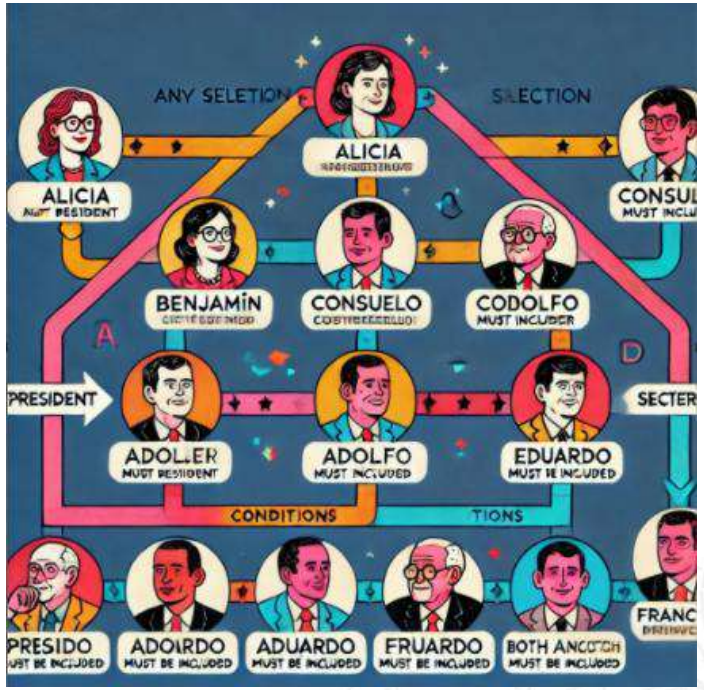
Código Ejemplo #4

Regla de la Suma


```python
from itertools import permutations

# Lista de personas
personas = ["Alicia", "Benjamín", "Consuelo", "Adolfo", "Eduardo", "Francisco"]

# a) Número total de maneras de seleccionar presidente, secretario y tesorero
def parte_a():
    # Generar todas las permutaciones de 3 personas
    todas_las_permutaciones = list(permutations(personas, 3))
    # El número de permutaciones es la respuesta
    return len(todas_las_permutaciones)

# b) Número de maneras si Alicia o Benjamín debe ser el presidente
def parte_b():
    contador = 0
    # Generar todas las permutaciones de 3 personas
    for perm in permutations(personas, 3):
        # Verificar si el presidente es Alicia o Benjamín
        if perm[0] in ["Alicia", "Benjamín"]:
            contador += 1
    return contador

# c) Número de maneras si Eduardo debe ocupar uno de los puestos
def parte_c():
    contador = 0
    # Generar todas las permutaciones de 3 personas
    for perm in permutations(personas, 3):
        # Verificar si Eduardo está en la permutación
        if "Eduardo" in perm:
            contador += 1
    return contador

# d) Número de maneras si Adolfo y Francisco deben ocupar un puesto
def parte_d():
    contador = 0
    # Generar todas las permutaciones de 3 personas
    for perm in permutations(personas, 3):
        # Verificar si Adolfo y Francisco están en la permutación
        if "Adolfo" in perm and "Francisco" in perm:
            contador += 1
    return contador

# Mostrar resultados
print("Personas del comité: ", personas)
print("a) Número total de maneras:", parte_a())
print("b) Maneras si Alicia o Benjamín es presidente:", parte_b())
print("c) Maneras si Eduardo ocupa un puesto:", parte_c())
print("d) Maneras si Adolfo y Francisco ocupan un puesto:", parte_d())
```

C:/Python/python.exe ejemplo #4.py

Personas del comité: ['Alicia', 'Benjamín', 'Consuelo', 'Adolfo', 'Eduardo', 'Francisco']
a) Número total de maneras: 120
b) Maneras si Alicia o Benjamín es presidente: 40
c) Maneras si Eduardo ocupa un puesto: 60



$$\sum_{i=1}^{n} |X_i|$$

$$\prod_{i=1}^{t} n_i$$

THE SUM RULE If a task can be done either in one of n_1 ways or in one of n_2 ways, where none of the set of n_1 ways is the same as any of the set of n_2 ways, then there are $n_1 + n_2$ ways to do the task.

Kenneth Rosen

Ejemplo #4 de caso de conteo

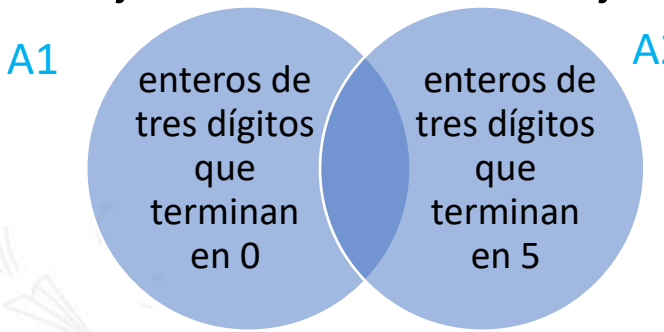
Regla de la Suma

¿Cuntos enteros de tres dígitos (enteros de 100 a 999 inclusive) son divisibles por 5?

- Enteros que son divisibles por 5 y terminan en 5 o en 0.
- El conjunto de todos los enteros de tres dígitos que son divisibles por 5 puede dividirse en dos subconjuntos mutuamente disjuntos A1 y A2:

$$\sum_{i=1}^{n} |X_i|$$

$$\prod_{i=1}^{t} n_i$$



$A_1 \cup A_2 =$ el conjunto de todos los enteros de tres dígitos que son divisibles por 5

$A_1 \cap A_2 = \emptyset$

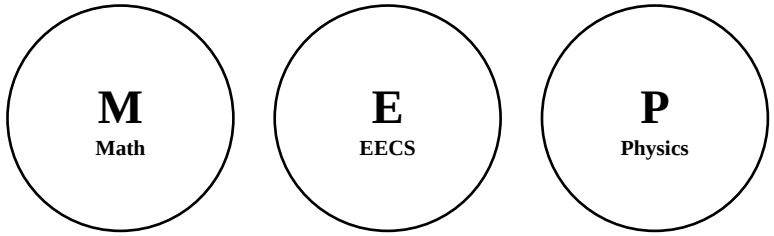
- Ahora hay tantos enteros de tres dígitos que terminan en 0 como opciones posibles para los dígitos de en medio y del extremo izquierdo (ya que el dígito de la derecha debe ser 0).
- Como se muestra a continuación, hay nueve opciones para el dígito del extremo izquierdo (dígitos del 1 al 9) y diez opciones para el dígito de en medio (dígitos de 0 a 9).
- Por tanto N(A2) = 9 x 10 = 90 y N(A2) = 9 x 10 = 90

Ejemplo #5 de caso de conteo avanzado

Principios de inclusión y exclusión

The Principle of Inclusion–Exclusion for counting

- How many students are there in the Math (Exact Sciences in Mathematics), EECS (Electrical Engineering and Computer Science), and Physics (Sciences of Physics) departments? In other words, what is **| M ∪ E ∪ P |** if:
  - **| M |** = 60
  - **| E |** = 200
  - **| P |** = 40



- The size of a union of three sets is given by a more complicated Inclusion-Exclusion formula:

$$|S_1 \cup S_2 \cup S_3| = |S_1| + |S_2| + |S_3| - |S_1 \cap S_2| - |S_1 \cap S_3| - |S_2 \cap S_3| + |S_1 \cap S_2 \cap S_3|$$

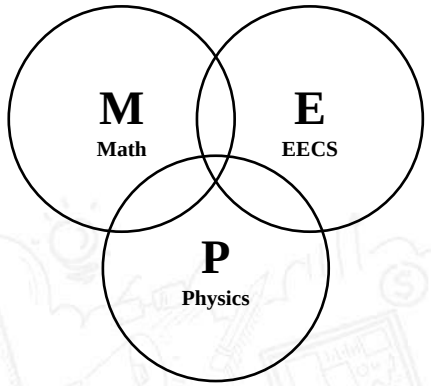
- Remarkably, the expression on the right accounts for each element in the union of S_1, S_2, and S_3 exactly once.

Ejemplo #5 de caso de conteo avanzado

Principios de inclusión y exclusión

The Principle of Inclusion–Exclusion for counting

- For example, suppose that x is an element of all three sets. Then x is counted three times (by the $|S_1|$, $|S_2|$, and $|S_3|$ terms), subtracted off three times (by the $|S_1 \cap S_2|$, $|S_1 \cap S_3|$, and $|S_2 \cap S_3|$ terms), and then counted once more (by the $|S_1 \cap S_2 \cap S_3|$ term). The net effect is that x is counted just once.
- If x is in two sets (say, S_1 and S_2), then x is counted twice (by the $|S_1|$ and $|S_2|$ terms) and subtracted once (by the $|S_1 \cap S_2|$ term). In this case, x does not contribute to any of the other terms, since $x \notin S_3$.
- So we can't answer the original question without knowing the sizes of the various intersections.
- Let's suppose that there are:
  - 4 Math-EECS Double Majors
  - 3 Math-Physics Double Majors
  - 11 EECS-Physics Double Majors
  - 2 Triple Majors



Ejemplo #5 de caso de conteo avanzado

Principios de inclusión y exclusión

The Principle of Inclusion–Exclusion for counting

- Then,
  - $|M \cap E| = 4+2$, $|M \cap P| = 3+2$, $|E \cap P| = 11+2$, and $|M \cap E \cap P| = 2$
- Plugging all this into the formula gives:
  - $|M \cup E \cup P| = |M| + |E| + |P| - |M \cap E| - |M \cap P| - |E \cap P| + |M \cap E \cap P|$

$= 60 + 200 + 40 - 6 - 5 - 13 + 2$

$= 278$

The total number of students in the departments of Mathematics, EECS and Physics is 278. 🎯

Código Ejemplo #5

Principios de inclusión y exclusión

The Principle of Inclusion–Exclusion for counting

```python
def inclusion_exclusion(M, E, P, ME, MP, EP, MEP):
    # Aplicamos la fórmula de inclusión y exclusión
    total_students = M + E + P - ME - MP - EP + MEP

    # Mostramos los cálculos paso a paso
    print(f"|M| = {M}, |E| = {E}, |P| = {P}")
    print(f"|M ∩ E| = {ME}, |M ∩ P| = {MP}, |E ∩ P| = {EP}")
    print(f"|M ∩ E ∩ P| = {MEP}")
    print("\nAplicando la fórmula de Inclusión y Exclusión:")
    print(f"|M ∪ E ∪ P| = {M} + {E} + {P} - {ME} - {MP} - {EP} + {MEP}")
    print(f"Resultado: {total_students}")

    return total_students

# Datos del problema
M = 60   # Estudiantes de Matemáticas
E = 200  # Estudiantes de EECS
P = 40   # Estudiantes de Física
ME = 4 + 2   # Doble especialización en Matemáticas y EECS
MP = 3 + 2   # Doble especialización en Matemáticas y Física
EP = 11 + 2  # Doble especialización en EECS y Física
MEP = 2      # Triple especialización

# Llamamos a la función
total = inclusion_exclusion(M, E, P, ME, MP, EP, MEP)

# Output final
print(f"\nNúmero total de estudiantes en los departamentos: {total}")
```

```
C:/Python/python.exe Tema-01_ejemplo #5.py
|M| = 60, |E| = 200, |P| = 40
|M ∩ E| = 6, |M ∩ P| = 5, |E ∩ P| = 13
|M ∩ E ∩ P| = 2

Aplicando la fórmula de Inclusión y Exclusión:
|M ∪ E ∪ P| = 60 + 200 + 40 - 6 - 5 - 13 + 2
Resultado: 278
```

|M| = 60
|E| = 200
|P| = 40

Total = 278

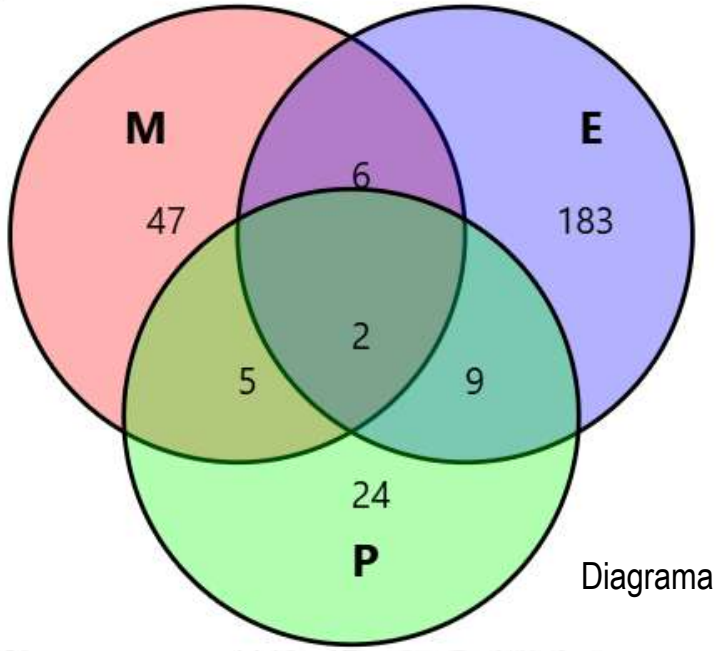


Diagrama de Venn de PIE

THE PRINCIPLE OF INCLUSION–EXCLUSION Let $A_1, A_2, \ldots, A_n$ be finite sets. Then

$$|A_1 \cup A_2 \cup \cdots \cup A_n| = \sum_{1 \le i \le n} |A_i| - \sum_{1 \le i < j \le n} |A_i \cap A_j| + \sum_{1 \le i < j < k \le n} |A_i \cap A_j \cap A_k| - \cdots + (-1)^{n+1} |A_1 \cap A_2 \cap \cdots \cap A_n|.$$

Kenneth Rosen

Playgrounds:
www.online-python.com

Ejemplo #6 de caso de conteo avanzado
Principios de inclusión y exclusión en Probabilidad

Solución #1

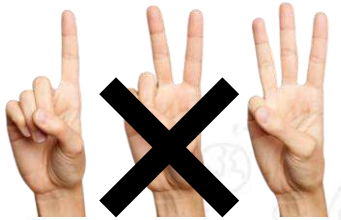
- Calcular la probabilidad de que en tres lanzamientos de una moneda, la moneda salga cara en el primer lanzamiento o en el último lanzamiento.

1.Identifiquemos lo que buscamos:
- Queremos P(cara en el 1er lanzamiento O cara en el último lanzamiento)

2.Podemos usar la notación:
- Sea C = cara, S = sello
- El primer lanzamiento es posición 1
- El último lanzamiento es posición 3

3.Los casos favorables son cuando:
- C aparece en posición 1 O
- C aparece en posición 3 O
- C aparece en ambas posiciones (1 y 3)

4. Listemos todos los casos favorables:
- CSS (C en 1)
- CSC (C en 1 y 3)
- CCS (C en 1)
- CCC (C en 1 y 3)
- SCS (C en medio, no cuenta)
- SCC (C en 3)
- SSC (C en 3)
- SSS (ninguna C, no cuenta)



5.Contando casos favorables:
- Total de casos favorables = 6 casos (CSS, CSC, CCS, CCC, SCC, SSC)
- Total de casos posibles = 8 (todas las combinaciones posibles en 3 lanzamientos)

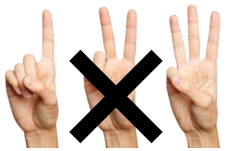
6.Por lo tanto: Probabilidad = Casos favorables / Casos posibles = 6/8 = 3/4 = 0.75

- La probabilidad es ¾ o 75% de que salga cara en el primer lanzamiento o en el último lanzamiento

Ejemplo #6 de caso de conteo avanzado
Principios de inclusión y exclusión en Probabilidad

Solución #2

- Calcular la probabilidad de que en tres lanzamientos de una moneda, la moneda salga cara en el primer lanzamiento o en el último lanzamiento.
- Vamos a resolverlo usando el principio de inclusión-exclusión en probabilidad, donde:
  - A = Evento "cara en el primer lanzamiento“
  - B = Evento "cara en el último lanzamiento“
- Por el principio de inclusión-exclusión: $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
- Calculemos cada término:



1. P(A) = Probabilidad de cara en el primer lanzamiento
   - P(A) = ½
2. P(B) = Probabilidad de cara en el último lanzamiento
   - P(B) = ½
3. $P(A \cap B)$ = Probabilidad de cara en el primer y último lanzamiento
   - Para que esto ocurra, necesitamos cara en posición 1 y 3
   - $P(A \cap B) = 1/2 \times 1/2 = 1/4$
   - (El lanzamiento del medio no importa)

Ahora aplicamos la fórmula:
$P(A \cup B) = P(A) + P(B) - P(A \cap B)$
$= 1/2 + 1/2 - 1/4$
$= 4/8 + 4/8 - 2/8$
$= 6/8$
$= ¾$

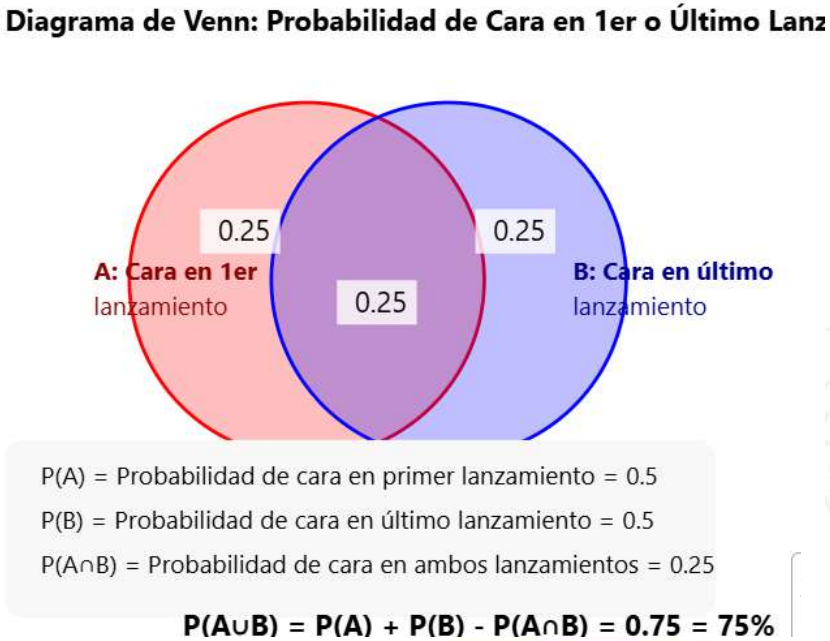
Código Ejemplo #6

Principios de inclusión y exclusión en Probabilidad

Este código:

- Define una función principal que calcula todas las probabilidades
- Calcula P(A) y P(B) individualmente
- Calcula la intersección P(A∩B)
- Aplica la fórmula de inclusión-exclusión
- Muestra todos los pasos del cálculo
- Retorna el resultado final

Esta solución usando inclusión-exclusión es más elegante ya que:
1.No necesitamos enumerar todos los casos
2.Solo necesitamos calcular las probabilidades individuales y su intersección
3.La fórmula nos da directamente el resultado




```python
def calcular_probabilidad_moneda():
    # Probabilidad de cara en el primer lanzamiento
    P_A = 1/2
    print(f"P(A) = Probabilidad de cara en primer lanzamiento = {P_A}")

    # Probabilidad de cara en el último lanzamiento
    P_B = 1/2
    print(f"P(B) = Probabilidad de cara en último lanzamiento = {P_B}")

    # Probabilidad de cara en ambos lanzamientos (intersección)
    P_AB = 1/2 * 1/2  # Eventos independientes
    print(f"P(A∩B) = Probabilidad de cara en ambos lanzamientos = {P_AB}")

    # Aplicando principio de inclusión-exclusión
    P_AUB = P_A + P_B - P_AB
    print(f"\nP(A∪B) = P(A) + P(B) - P(A∩B)")
    print(f"P(A∪B) = {P_A} + {P_B} - {P_AB}")
    print(f"P(A∪B) = {P_AUB}")
    print(f"\nLa probabilidad es {P_AUB:.2%}")

    return P_AUB
# Ejecutar el cálculo
if __name__ == "__main__":
    print("Calculando la probabilidad de obtener cara en el primer O último lanzamiento...")
    print("-" * 70)
    resultado = calcular_probabilidad_moneda()
```

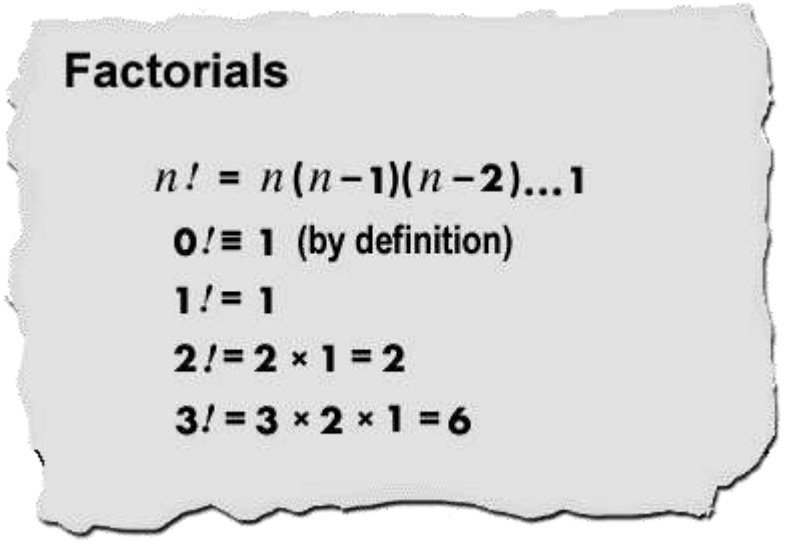
```
C:/Python/python.exe Tema-01_ejemplo #6.py
Calculando la probabilidad de obtener cara en el primer O último lanzamiento...
----------------------------------------------------------------------
P(A) = Probabilidad de cara en primer lanzamiento = 0.5
P(B) = Probabilidad de cara en último lanzamiento = 0.5
P(A∩B) = Probabilidad de cara en ambos lanzamientos = 0.25

P(A∪B) = P(A) + P(B) - P(A∩B)
P(A∪B) = 0.5 + 0.5 - 0.25
P(A∪B) = 0.75

La probabilidad es 75.00%
```

Institución Universitaria
Acreditada en Alta Calidad

Código Ejemplo #7

Factorial

```python
def fac(n):
    """Calcula el factorial de un número."""
    result = 1
    for i in range(1, n+1):
        result *= i
    return result

while True:
    n = input('Ingresa un número (o Q/q/X/x para salir): ')
    if n.upper() in ['Q', 'X']:
        break
    try:
        n = int(n)
        print(f'El factorial de {n}! es {fac(n)}')
    except ValueError:
        print("Entrada inválida. Por favor, ingresa un número o Q/q/X/x para salir.")
```



C:/Python/python.exe factorial+.py

Ingresa un número (o Q/q/X/x para salir): 4

El factorial de 4! es 24

Ingresa un número (o Q/q/X/x para salir): 5

El factorial de 5! es 120

Ingresa un número (o Q/q/X/x para salir): 6

El factorial de 6! es 720

Ingresa un número (o Q/q/X/x para salir): 7

El factorial de 7! es 5040

Ingresa un número (o Q/q/X/x para salir): 9

El factorial de 9! es 362880

Ingresa un número (o Q/q/X/x para salir): Q

Entrada inválida. Por favor, ingresa un número o Q/q/X/x para salir.

```python
def fac(n):
    result = 1
    for i in range(1, n+1):
        result *= i
    return result
```

Ejemplo #7 - Caso de conteo avanzado
Progresiones Aritmética

Arithmetic Progression

Análisis de Secuencias Aritméticas

- Secuencia $\{s_n\}$:
  - Fórmula: $s_n = -1 + 4n$
  - Término inicial (s_0): -1
  - Diferencia común: 4
  - Primeros 6 términos: -1, 3, 7, 11, 15, 19, ...
- Secuencia $\{t_n\}$:
  - Fórmula: $t_n = 7 - 3n$
  - Término inicial (t_0): 7
  - Diferencia común: -3
  - Primeros 6 términos: 7, 4, 1, -2, -5, -8, ...
- Ejemplos de Strings en Informática
  - String vacío (λ):
  - Longitud del string vacío: 0
  - Ejemplo de bit string de longitud 8: 10110101
  - Longitud del bit string: 8

```python
def generar_secuencia_sn(n_terminos):
    """
    Genera la secuencia sn = -1 + 4n
    Parámetros:
    n_terminos: número de términos a generar
    """
    return [-1 + 4*n for n in range(n_terminos)]

def generar_secuencia_tn(n_terminos):
    """
    Genera la secuencia tn = 7 - 3n
    Parámetros:
    n_terminos: número de términos a generar
    """
    return [7 - 3*n for n in range(n_terminos)]

def mostrar_secuencia(nombre, secuencia, formula, termino_inicial, diferencia):
    """
    Muestra la información detallada de una secuencia
    """
    print(f"\nSecuencia {nombre}:")
    print(f"Fórmula: {formula}")
    print(f"Término inicial: {termino_inicial}")
    print(f"Diferencia común: {diferencia}")
    print(f"Primeros {len(secuencia)} términos: {', '.join(map(str, secuencia))}")

def generar_string_binario(longitud):
    """
    Ejemplo de generación de string binario
    """
    from random import choice
    return ''.join(choice(['0', '1']) for _ in range(longitud))

def main():
    # Número de términos a generar
    n_terminos = 6

    print("Análisis de Secuencias Aritméticas")
    print("=" * 50)

    # Generar y mostrar secuencia sn
    secuencia_sn = generar_secuencia_sn(n_terminos)
    mostrar_secuencia(
        "sn",
        secuencia_sn,
        "sn = -1 + 4n",
        termino_inicial=-1,
        diferencia=4
    )

    # Generar y mostrar secuencia tn
    secuencia_tn = generar_secuencia_tn(n_terminos)
    mostrar_secuencia(
        "tn",
        secuencia_tn,
        "tn = 7 - 3n",
        termino_inicial=7,
        diferencia=-3
    )

    # Ejemplos de strings
    print("\nEjemplos de Strings en Informática")
    print("=" * 50)
    print("String vacío (λ):", "")
    print("Longitud del string vacío:", 0)

    # Ejemplo con string binario
    bit_string = generar_string_binario(8)
    print(f"\nEjemplo de bit string de longitud 8: {bit_string}")
    print(f"Longitud del bit string: {len(bit_string)}")

if __name__ == "__main__":
    main()
```

```
C:/Python/python.exe Tema-01_ejemplo #7.py"
Análisis de Secuencias Aritméticas
==================================================

Secuencia sn:
Fórmula: sn = -1 + 4n
Término inicial: -1
Diferencia común: 4
Primeros 6 términos: -1, 3, 7, 11, 15, 19

Secuencia tn:
Fórmula: tn = 7 - 3n
Término inicial: 7
Diferencia común: -3
Primeros 6 términos: 7, 4, 1, -2, -5, -8

Ejemplos de Strings en Informática
==================================================
String vacío (λ):
Longitud del string vacío: 0

Ejemplo de bit string de longitud 8: 00101111
Longitud del bit string: 8
Información de las secuencias:
--------------------------------------------------
Secuencia bn: término inicial = 1, razón = -1
Secuencia cn: término inicial = 2, razón = 5
Secuencia dn: término inicial = 6, razón = 1/3
```

An *arithmetic progression* is a sequence of the form

$$a, a+d, a+2d, \ldots, a+nd, \ldots$$

where the *initial term* a and the *common difference* d are real numbers.

Institución Universitaria
Acreditada en Alta Calidad

Ejemplo #8 de caso de conteo avanzado
Progresiones Geométricas

Geometric Progression

Análisis de Secuencias Geométricas

- Las secuencias $\{b_n\}$ con $b_n = (-1)^n$, $\{c_n\}$ con $c_n = 2 \cdot 5^n$, y $\{d_n\}$ con $d_n = 6 \cdot (1/3)^n$ son progresiones geométricas con término inicial y razón común iguales a 1 y -1; 2 y 5; y 6 y 1/3, respectivamente, si comenzamos en n = 0.
- La lista de términos $b_0, b_1, b_2, b_3, b_4, \ldots$ comienza con 1, -1, 1, -1, 1, ...;
- la lista de términos $c_0, c_1, c_2, c_3, c_4, \ldots$ comienza con 2, 10, 50, 250, 1250, ...;
- y la lista de términos $d_0, d_1, d_2, d_3, d_4, \ldots$ comienza con 6, 2, 2/3, 2/9, 2/27, ...

```python
def generar_secuencia_bn(n_terminos):
    """Genera la secuencia bn = (-1)^n"""
    return [(-1)**n for n in range(n_terminos)]

def generar_secuencia_cn(n_terminos):
    """Genera la secuencia cn = 2*5^n"""
    return [2 * (5**n) for n in range(n_terminos)]

def generar_secuencia_dn(n_terminos):
    """Genera la secuencia dn = 6*(1/3)^n"""
    return [6 * (1/3)**n for n in range(n_terminos)]

def mostrar_secuencia(nombre, secuencia, formato=""):
    """Muestra la secuencia con formato específico"""
    print(f"\nSecuencia {nombre}:")
    if formato == "fraccion":
        # Para la secuencia dn, mostrar como fraccio-
nes
        terminos = []
        for num in secuencia:
            if num >= 1:
                terminos.append(str(num))
            else:
                # Convertir decimales a fracciones
simples
                denominador = 3 **
(len(str(num).split('.')[-1]))
                numerador = int(num * denominador)
                terminos.append(f"{numerador}/{denom-
inador}")
    else:
        terminos = [str(num) for num in secuencia]

    print(", ".join(terminos))

def main():
    # Número de términos a generar
    n_terminos = 5

    print("Generando las tres secuencias geométri-
cas:")
    print("-" * 50)
```

```python
    # Generar y mostrar secuencia bn
    secuencia_bn = generar_secuencia_bn(n_terminos)
    mostrar_secuencia("bn = (-1)^n", secuencia_bn)

    # Generar y mostrar secuencia cn
    secuencia_cn = generar_secuencia_cn(n_terminos)
    mostrar_secuencia("cn = 2*5^n", secuencia_cn)

    # Generar y mostrar secuencia dn
    secuencia_dn = generar_secuencia_dn(n_terminos)
    mostrar_secuencia
    # Mostrar información adicional
    print("\nInformación de las secuencias:")
    print("-" * 50)
    print("Secuencia bn: término inicial = 1, razón =
-1")
    print("Secuencia cn: término inicial = 2, razón =
5")
    print("Secuencia dn: término inicial = 6, razón =
1/3")

if __name__ == "__main__":
    main()
```

```
C:/Python/python.exe Tema-01_ejemplo #7.py"
Generando las tres secuencias geométricas:
--------------------------------------------------

Secuencia bn = (-1)^n:
1, -1, 1, -1, 1

Secuencia cn = 2*5^n:
2, 10, 50, 250, 1250

Secuencia dn = 6*(1/3)^n:
6.0, 2.0, 28697814/43046721, 28697813/129140163,
9565937/129140163

Información de las secuencias:
--------------------------------------------------
Secuencia bn: término inicial = 1, razón = -1
Secuencia cn: término inicial = 2, razón = 5
Secuencia dn: término inicial = 6, razón = 1/3
```

A *geometric progression* is a sequence of the form

$$a, ar, ar^2, \ldots, ar^n, \ldots$$

where the *initial term* a and the *common ratio* r are real numbers.

Ejemplo #9 de caso de permutaciones

Permutations

Podio de Ganadores

- Supongamos que hay 5 personas y necesitas elegir 3 para sentarlas en una fila numerada (por ejemplo, en un podio de ganadores: 1er lugar, 2do lugar, 3er lugar).Si eliges a Ana, Juan y Pedro, esto es diferente a elegir Pedro, Juan y Ana, porque el orden en que los sientas cambia el resultado.
- Imagina que hay **5 finalistas** en una competencia y se entregan premios de **oro, plata y bronce** (1er, 2do y 3er lugar).
  Si eliges **3 personas**, hay muchas maneras de asignarles los premios:
  1. Ana, Juan, Pedro ⟶ Ana gana oro, Juan gana plata, Pedro gana bronce.
  2. Ana, Pedro, Juan ⟶ Ana gana oro, Pedro gana plata, Juan gana bronce.
  3. Pedro, Juan, Ana ⟶ Pedro gana oro, Juan gana plata, Ana gana bronce
  4. …. (y así hasta llegar a 60 combinaciones diferentes).

Cada elección de 3 personas en un **orden diferente** cuenta como una posibilidad distinta, por eso usamos la fórmula de **permutaciones**.

Permutaciones (P) ⟶ El orden importa (ej. premios de oro, plata y bronce).

```python
from itertools import permutations, combinations
from math import factorial, comb

personas = ["Ana", "Juan", "Pedro", "Luis", "María"]

def permutaciones(n, r):
    return factorial(n) // factorial(n - r)

def permutar(r):
    for p in permutations(personas, r):
        print(p)
    return

# Ejecutar el cálculo
if __name__ == "__main__":
    # Permutaciones de 3 lugares de 5 personas posibles (orden importa)
    print("Permutaciones (el orden importa).")
    print(f'Se tienen {permutaciones(5, 3)} posibles permutaciones para 3 lugares de las 5 personas posibles:')
    permutar(3)
```

```
C:/Python/python.exe  permuta-combina.py"
Permutaciones (el orden importa).
Se tienen 60 posibles permutaciones para 3 lugares de las 5 personas posibles:
('Ana', 'Juan', 'Pedro')
('Ana', 'Juan', 'Luis')
('Ana', 'Juan', 'María')
('Ana', 'Pedro', 'Juan')
('Ana', 'Pedro', 'Luis')
('Ana', 'Pedro', 'María')
('Ana', 'Luis', 'Juan')
('Ana', 'Luis', 'Pedro')
('Ana', 'Luis', 'María')
('Ana', 'María', 'Juan')
('Ana', 'María', 'Pedro')
('Ana', 'María', 'Luis')
('Juan', 'Ana', 'Pedro')
('Juan', 'Ana', 'Luis')
('Juan', 'Ana', 'María')
('Juan', 'Pedro', 'Ana')
('Juan', 'Pedro', 'Luis')
('Juan', 'Pedro', 'María')
('Juan', 'Luis', 'Ana')
('Juan', 'Luis', 'Pedro')
('Juan', 'Luis', 'María')
('Juan', 'María', 'Ana')
('Juan', 'María', 'Pedro')
('Juan', 'María', 'Luis')
('Pedro', 'Ana', 'Juan')
('Pedro', 'Ana', 'Luis')
('Pedro', 'Ana', 'María')
('Pedro', 'Juan', 'Ana')
('Pedro', 'Juan', 'Luis')
('Pedro', 'Juan', 'María')
('Pedro', 'Luis', 'Ana')
('Pedro', 'Luis', 'Juan')
('Pedro', 'Luis', 'María')
('Pedro', 'María', 'Ana')
('Pedro', 'María', 'Juan')
('Pedro', 'María', 'Luis')
('Luis', 'Ana', 'Juan')
('Luis', 'Ana', 'Pedro')
('Luis', 'Ana', 'María')
('Luis', 'Juan', 'Ana')
('Luis', 'Juan', 'Pedro')
('Luis', 'Juan', 'María')
('Luis', 'Pedro', 'Ana')
('Luis', 'Pedro', 'Juan')
('Luis', 'Pedro', 'María')
('Luis', 'María', 'Ana')
('Luis', 'María', 'Juan')
('Luis', 'María', 'Pedro')
('María', 'Ana', 'Juan')
('María', 'Ana', 'Pedro')
('María', 'Ana', 'Luis')
('María', 'Juan', 'Ana')
('María', 'Juan', 'Pedro')
('María', 'Juan', 'Luis')
('María', 'Pedro', 'Ana')
('María', 'Pedro', 'Juan')
('María', 'Pedro', 'Luis')
('María', 'Luis', 'Ana')
```

If n is a positive integer and r is an integer with $1 \le r \le n$, then there are

$$P(n, r) = n(n-1)(n-2)\cdots(n-r+1)$$

r-permutations of a set with n distinct elements.

If n and r are integers with $0 \le r \le n$, then $P(n, r) = \dfrac{n!}{(n-r)!}$

Ejemplo #9 de caso de combinaciones

Combinations

Grupo de trabajo

- Ahora, imagina que hay 5 candidatos postulados para ser parte de un grupo de trabajo y necesitas elegir 3 personas.
- Aquí no importa en qué orden elijas a los miembros, porque solo te interesa quiénes entran al grupo, no quién fue elegido primero o después.
- Si eliges a María, Luis y Sofía para el grupo, es lo mismo que elegir Sofía, María y Luis. El grupo sigue siendo el mismo, sin importar el orden de selección.
- Por eso, hay menos combinaciones posibles que permutaciones (solo 10 formas en lugar de muchas más).

Cada elección de 3 personas sin importar el **orden**, cuenta como una posibilidad, por eso usamos la fórmula de **combinaciones**.

Combinaciones (C) ⟶ El orden NO importa (ej. formar un grupo de trabajo).

```python
from itertools import permutations, combinations
from math import factorial, comb

personas = ["Ana", "Juan", "Pedro", "Luis", "María"]

def combinaciones(n, r):
    return {comb(5, 3)}

def combinar(r):
    for c in combinations(personas, 3):
        print(c)
    return

# Ejecutar el cálculo
if __name__ == "__main__":
    # Combinaciones de 3 lugares de 5 personas posibles (el orden NO importa)
    print("Combinaciones (el orden NO importa).")
    print(f'Se tienen {combinaciones(5, 3)} posibles combinaciones para 3 lugares de las 5 personas posibles:')
    combinar(3)
```

```
C:/Python/python.exe  permuta-combina.py

Combinaciones (el orden NO importa).
Se tienen {10} posibles permutaciones para 3 lugares de las 5 personas posibles:
('Ana', 'Juan', 'Pedro')
('Ana', 'Juan', 'Luis')
('Ana', 'Juan', 'María')
('Ana', 'Pedro', 'Luis')
('Ana', 'Pedro', 'María')
('Ana', 'Luis', 'María')
('Juan', 'Pedro', 'Luis')
('Juan', 'Pedro', 'María')
('Juan', 'Luis', 'María')
('Pedro', 'Luis', 'María')
```

The number of r-combinations of a set with n elements, where n is a nonnegative integer and r is an integer with $0 \le r \le n$, equals

$$C(n,r) = \frac{n!}{r!\,(n-r)!}.$$

Let n and r be nonnegative integers with $r \le n$. Then $C(n,r) = C(n,n-r)$

Ejemplo #10 de caso de relaciones de recurrencia

Números de Catalán

- Encuentra una relación de recurrencia para C_n, por medio de la cantidad de formas de poner entre paréntesis el producto de n + 1 números, $x_0 \cdot x_1 \cdot x_2 \cdot \cdots \cdot x_n$, para especificar el orden de multiplicación.
- Para desarrollar una relación de recurrencia para C_n, observamos que independientemente de cómo insertemos paréntesis en el producto $x_0 \cdot x_1 \cdot x_2 \cdot \cdots \cdot x_n$, un operador "·" permanece fuera de todos los paréntesis, es decir, el operador para la multiplicación final que se realizará.
- El objetivo es determinar el número de formas distintas de insertar paréntesis en el producto de n+1 números $x_0 \cdot x_1 \cdot x_2 \cdots x_n$ para especificar el orden de multiplicación. Por ejemplo:

  C_0=1: Solo hay una forma de poner entre paréntesis un solo número (no se necesita multiplicación).

  C_1=1: Solo hay una forma de poner entre paréntesis $x_0 \cdot x_1$.

  C_2=2: Hay dos formas de parentizar $x_0 \cdot x_1 \cdot x_2$:

  1. $(x_0 \cdot x_1) \cdot x_2$
  2. $x_0 \cdot (x_1 \cdot x_2)$

  C_3=5: Como se da en el problema, hay cinco formas de parentizar $x_0 \cdot x_1 \cdot x_2 \cdot x_3$ para determinar el orden de multiplicación.

  1. $((x_0 \cdot x_1) \cdot x_2) \cdot x_3$ 2. $(x_0 \cdot (x_1 \cdot x_2)) \cdot x_3$ 3. $(x_0 \cdot x_1) \cdot (x_2 \cdot x_3)$ 4. $x_0 \cdot ((x_1 \cdot x_2) \cdot x_3)$ 5. $x_0 \cdot (x_1 \cdot (x_2 \cdot x_3))$

- La sucesión $\{C_n\}$ es la sucesión de números de Catalán, llamada así en honor a Eugène Charles Catalán. Esta sucesión aparece como solución a muchos problemas de conteo diferentes.

Ejemplo #10 de caso de relaciones de recurrencia

Recurrence Relations

Derivar la relación de recurrencia

Para derivar una relación de recurrencia para C_n, consideremos lo siguiente:

- Supongamos que tenemos n+1 números: $x_0, x_1, x_2, \ldots, x_n$.
- Podemos dividir el producto en dos partes en cualquier posición k, donde $0 \leq k \leq n-1$. La primera parte contiene k+1 números $(x_0 \cdot x_1 \cdots x_k)$, y la segunda parte contiene n−k números $(x_{k+1} \cdot x_{k+2} \cdots x_n)$.
- El número de formas de poner entre paréntesis la primera parte es C_k, y el número de formas de parentizar la segunda parte es C_{n-k-1}.
- Como la división puede ocurrir en cualquier posición k, sumamos sobre todas las k posibles.
- Así, la relación de recurrencia es: $C_n = \sum_{k=0}^{n-1} C_k \cdot Cn_{-k-1}$ para n≥1, con el caso base C_0=1

- El término Ck representa el número de formas de parentizar los primeros k+1 números.
- El término $Cn-k-1$ representa el número de formas de parentizar los restantes $n-k$ números.
- El producto $Ck \cdot Cn-k-1$ da el número total de formas de parentizar el producto completo cuando la división ocurre en la posición k.
- Esta es la recurrencia de los números de Catalan, y C_n es el n-ésimo número de Catalan.
- Al sumar sobre todas las k posibles, obtenemos el número total de formas de parentizar el producto de n+1 números.
  Para n=3:
  $C_3 = C_0 \cdot C_2 + C_1 \cdot C_1 + C_2 \cdot C_0$
- Substituyendo los valores conocidos:
  $C_3 = 1 \cdot 2 + 1 \cdot 1 + 2 \cdot 1 = 2+1+2 = 5$

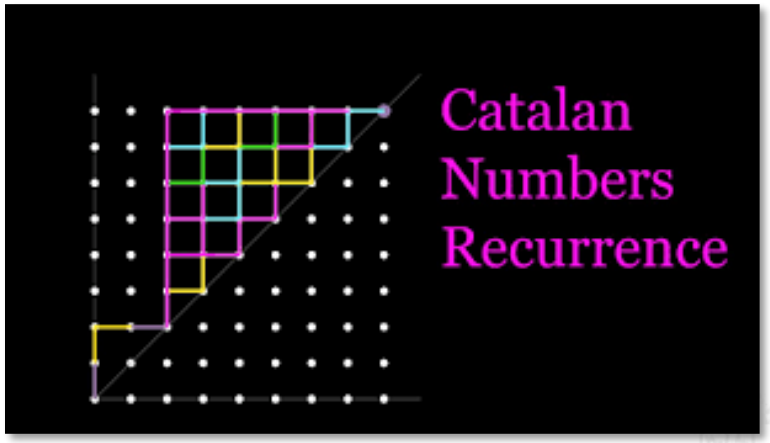
Ejemplo #10 de caso de relaciones de recurrencia

Recurrence Relations

Código

Este código calcula el número de formas de colocar los paréntesis utilizando la fórmula cerrada de los números de Catalán.

```python
import math

def catalan_number(n):
    return math.comb(2 * n, n) // (n + 1)

def generate_parenthesizations(n):
    if n == 0:
        return [""]
    result = []
    for i in range(n):
        for left in generate_parenthesizations(i):
            for right in generate_parenthesizations(n - 1 - i):
                result.append(f"({left}){right}")
    return result

# Solicitar al usuario el valor de n
n = int(input("Ingrese el valor de n: "))

# Calcular y mostrar el número de maneras de parentizar
print(f"Número de formas de parentizar {n+1} números: {catalan_number(n)}")
print("Formas posibles:")
for p in generate_parenthesizations(n):
    print(p)
```




```
C:/Python/python.exe Tema-01_ejemplo #10.py
Ingrese el valor de n: 3
Número de formas de parentizar 4 números: 5
Formas posibles:
()()()
()(())
(())()
(()())
((()))
```

Institución Universitaria
Acreditada en Alta Calidad

– "Oye me contaron un chiste sobre la sucesión de Fibonacci"
>> Qué?

> Qué?
>> Qué de qué?
> Qué de qué de qué?
>> Qué de qué de qué de qué de qué…

EJERCICIOS PROPUESTOS

"Daría todo lo que sé por la mitad de lo que ignoro."
René Descartes

Resumen

Regla del Producto y de la Suma

- Según Richard Johnsonbaugh:
  - La clave para resolver problemas en esta sección es determinar cuándo usar el principio de la multiplicación y cuándo el principio de la suma.
  - Utilice el principio de la multiplicación cuando tenga un proceso paso por paso para construir una actividad.
    - Por ejemplo, en el caso del restaurante, para diseñar una comida del menú de comida rápida que consista en un entremés, un plato fuerte y una bebida, se requiere un proceso de tres pasos: 1. Elegir un entremés; 2. Elegir un plato fuerte; 3. Elegir una bebida.
    - El número de actividades diferentes posibles es el producto del número de maneras en que se puede realizar cada paso. En este caso, el entremés se puede seleccionar de 2 maneras, un plato fuerte de 3 maneras y una bebida de 4 maneras. Entonces, el número de comidas es $2 \cdot 3 \cdot 4 = 24$.
  - Utilice el principio de la suma cuando desee contar el número de elementos en un conjunto y pueda dividir el conjunto en subconjuntos que no se traslapan.
    - Por ejemplo, suponga que se desea contar el número total de artículos disponibles en el menú de comidas rápidas. Como hay 2 entremeses, 3 platos fuertes y 4 bebidas, y ningún artículo pertenece a dos categorías, el número total de artículos disponibles es $2 + 3 + 4 = 9$
- Estas reglas son fundamentales para resolver problemas de combinatoria y se aplican en teoría de la probabilidad, análisis de algoritmos y optimización.

Institución Universitaria
Acreditada en Alta Calidad

Resumen

Técnicas avanzadas de Conteo

- Los Principios de Inclusión y Exclusión (PIE) son fundamentales tanto para conteo como para probabilidad, algunas aplicaciones son:
  - Conteo:
    - Problemas de selección múltiple
    - Organización de eventos
    - Análisis de datos
    - Teoría de conjuntos
  - Probabilidad:
    - Análisis de riesgo
    - Control de calidad
    - Probabilidad de eventos compuestos
    - Diagnósticos médicos
    - Confiabilidad de sistemas
- En el contexto de ciencias de la computación, respecto a las progresiones o secuencias:
  - Las secuencias son fundamentales para entender patrones y algoritmos
  - Los strings (mencionados en el segundo ejemplo) son secuencias de caracteres esenciales en programación
  - Ayudan a entender estructuras de datos y algoritmos de búsqueda
  - Son útiles para análisis de complejidad y crecimiento de funciones

Institución Universitaria
Acreditada en Alta Calidad

Resumen

Permutaciones, Combinaciones y Relaciones de Recurrencia

Permutations

Combinations

Recurrence Relations

- Permutations (P)
  - **Order matters** (e.g. gold, silver and bronze awards).
  - It is any arrangement of elements where we are interested in the place or position that each one occupies of the elements that constitute such an arrangement.
  - Counting permutations when repetition of elements is allowed can easily be done using the product rule.
- Combinations (C)
  - **Order DOES NOT matter** (e.g. forming a working group).
  - A combination is an arrangement of elements where we are not interested in the place or position that they occupy within the arrangement. In a combination we are interested in forming groups and their content.
- Recurrence Relations
  - It is an equation that defines a sequence based on its previous terms.
  - Generally, this type of situation is perfectly modeled with the Fibonacci sequence.
  - They are fundamental in **recursive algorithms**, data structures, and computational complexity analysis.

Institución Universitaria
Acreditada en Alta Calidad

Ejercicios propuestos

Regla de la Multiplicación

Product Rule

1. En un restaurante, el menú de comida rápida, contiene dos entremeses, tres platos fuertes y cuatro bebidas. Si el dueño del restaurante decide ajustar el menú de comida rápida agregando una nueva bebida.
   - ¿Cuántas comidas de un plato fuerte y una bebida estarían disponibles para los comensales?
   - ¿Cuántas comidas diferentes se pueden tener si están formadas por un entremés, un plato fuerte y una bebida?
2. A new company with just two employees, Sanchez and Patel, rents a floor of a building with 12 offices. How many ways are there to assign different offices to these two employees?
3. The chairs of an auditorium are to be labeled with an uppercase English letter followed by a positive integer not exceeding 100. What is the largest number of chairs that can be labeled differently?

Ejercicios propuestos

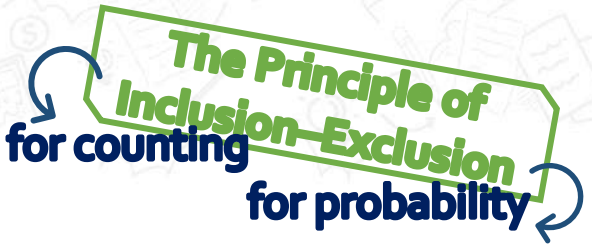
Regla de la Suma

Sum Rule

1. Se lanzan dos dados, uno azul y otro rojo.
   - ¿Cuántos resultados posibles hay?
   - ¿Cuantos resultados son dobles? (Un doble ocurre cuando los dos dados muestran el mismo número).
   - ¿Cuántos resultados suman 4?
   - ¿Cuántos resultados suman 7 u 11?
   - ¿En cuántos resultados el dado azul muestra 2?
   - ¿Cuántos resultados tienen al menos un dado que muestra 2?
   - ¿En cuántos resultados ningún dado muestra 2?
   - ¿Cuántos resultados dan una suma par?
2. A student can choose a computer project from one of three lists. The three lists contain 23, 15, and 19 possible projects, respectively. No project is on more than one list. How many possible projects are there to choose from?
3. In how many ways can you draw a first card and then a second card from a deck of 52 cards? In how many ways can you draw two cards from a deck of 52 cards? In how many ways can you draw a first, second, and third card from a deck of 52 cards?

Ejercicios propuestos

Principios de inclusión y exclusión para Conteo y para Probabilidad



1. Se lanzan dos dados. ¿Cuál es la probabilidad de que salga un dado con seis puntos en la parte superior?
2. En una universidad hay 345 estudiantes que han tomado un curso de cálculo, 212 que han tomado un curso de matemáticas discretas y 188 que han tomado cursos tanto de cálculo como de matemáticas discretas. ¿Cuántos estudiantes han tomado un curso de cálculo o de matemáticas discretas?
3. Suppose that there are 1807 freshmen at your school. Of these, 453 are taking a course in computer science, 567 are taking a course in mathematics, and 299 are taking courses in both computer science and mathematics. How many are not taking a course either in computer science or in mathematics?
4. How many positive integers not exceeding 1000 are divisible by 7 or 11?

Institución Universitaria
Acreditada en Alta Calidad

Ejercicios propuestos

Progresiones Aritméticas y Geométricas

1. Encuentra estos términos de la sucesión $\{a_n\}$, donde $a_n = 2 \cdot (-3)^n + 5^n$.
   Para a) a_0 b) a_1 c) a_4 d) a_5 .
2. Un empleado se incorporó a una empresa en 2009 con un salario inicial de 5.000.000 pesos. Cada año, este empleado recibe un aumento de 100.000 dólares más el 5% del salario del año anterior.
3. Assume that the population of the world in 2010 was 6.9 billion and is growing at the rate of 1.1% a year.
   - a) Set up a recurrence relation for the population of the world n years after 2010.
   - b) Find an explicit formula for the population of the world n years after 2010.
   - c) What will the population of the world be in 2030?

Institución Universitaria
Acreditada en Alta Calidad

Ejercicios propuestos
Permutaciones y Combinaciones

Permutations | Combinations

1. Considere 3 libros: de computación, física e historia. Suponga que la biblioteca tienen al menos 6 copias de cada uno. ¿De cuántas maneras se pueden seleccionar 6 libros?
2. Entre las historietas cómicas Batman, Superman, Spiderman, X-Men, Capitana Marvel y Archie. ¿Cuántas maneras hay para seleccionar 6 historietas? ¿Cuántas maneras hay para seleccionar 10 historietas? ¿Cuántas maneras hay para seleccionar 10 historietas si elegimos al menos una de cada título?
3. How many permutations of the letters ABCDEFGH contain the string ABC ?
4. Suppose that there are eight runners in a race. The winner receives a gold medal, the second place finisher receives a silver medal, and the third-place finisher receives a bronze medal. How many different ways are there to award these medals, if all possible outcomes of the race can occur and there are no ties?

Ejercicios propuestos
Relaciones de Recurrencia

Recurrence Relations

1. Supongamos que una pareja de conejos recién nacidos tarda un mes en madurar y, a partir del segundo mes, cada pareja produce otra pareja cada mes. Si todos los conejos sobreviven, ¿cuántos conejos habrá después de n meses?
2. Al pagar un préstamo con un monto inicial A y un pago mensual M a una tasa de interés del p por ciento, el monto total T(n) del préstamo después de n meses se calcula sumando p/12 por ciento al monto adeudado después de n − 1 meses y luego restando el pago mensual M. Convierta esta descripción en una recurrencia para el monto adeudado después de n meses.
3. Enumerate of string of decimal digits. A computer system considers a string of decimal digits a valid codeword if it contains an even number of 0 digits. For instance, 1230407869 is valid, whereas 120987045608 is not valid. Let an be the number of valid n-digit codewords. Find a recurrence relation for a_n.


ITM
Institución Universitaria
Acreditada en Alta Calidad
Extras
Somos Innovación Tecnológica con Sentido Humano
Alcaldía de Medellín

Institución Universitaria
Acreditada en Alta Calidad

LISTA DE SÍMBOLOS

Notación de conjuntos

$\{x_1,\ldots,x_n\}$	conjunto que consta de los elementos $x_1,\ldots,x_n$;
$\{x \mid p(x)\}$	conjunto de los elementos x que satisfacen la propiedad $p(x)$;
$x \in X$	x es un elemento de X;
$x \notin X$	x no es un elemento de X;
$X = Y$	igualdad de conjuntos (X y Y tienen los mismos elementos);
$\lvert X\rvert$	número de elementos en X;
$\emptyset$	conjunto vacío;
$X \subseteq Y$	X es un subconjunto de Y;
$X \subset Y$	X es un subconjunto propio de Y;
$\mathcal{P}(X)$	conjunto potencia de X (todos los subconjuntos de X);
$X \cup Y$	X unión Y (todos los elementos en X o Y);
$\bigcup_{i=1}^{n} X_i$	unión de $X_1,\ldots,X_n$ (todos los elementos que pertenecen al menos a un conjunto de $X_1,\ldots,X_n$);
$\bigcup_{i=1}^{\infty} X_i$	unión de $X_1, X_2,\ldots$ (todos los elementos que pertenecen al menos a uno de $X_1, X_2,\ldots$);
$\cup\mathcal{S}$	unión de $\mathcal{S}$ (todos los elementos que pertenecen al menos a un conjunto en $\mathcal{S}$);
$X \cap Y$	X intersección Y (todos los elementos en X y Y);
$\bigcap_{i=1}^{n} X_i$	intersección de $X_1,\ldots,X_n$ (todos los elementos que pertenecen a todos los conjuntos $X_1, X_2,\ldots, X_n$);
$\bigcap_{i=1}^{\infty} X_i$	intersección de $X_1, X_2,\ldots$ (todos los elementos que pertenecen a todos los conjuntos $X_1, X_2,\ldots$);
$\cap\mathcal{S}$	intersección de $\mathcal{S}$ (todos los elementos que pertenecen a todos los conjuntos de $\mathcal{S}$);
$X - Y$	diferencia de conjuntos (todos los elementos en X pero no en Y);
$\overline{X}$	complemento de X (todos los elementos que no están en X);
(x, y)	par ordenado;
$(x_1,\ldots,x_n)$	n-eada;
$X \times Y$	producto cartesiano de X y Y [pares (x, y) con x en X y y en Y];

Lógica

$p \vee q$	p o q;
$p \wedge q$	p y q;
$\neg p$	no p;
$p \rightarrow q$	si p, entonces q;
$p \leftrightarrow q$	p si y sólo si q;
$P \equiv Q$	P y Q son lógicamente equivalentes;
$\forall$	para todo;
$\exists$	existe;
$\backslash$	por lo tanto;

Relaciones

$x\,R\,y$	(x, y) está en R (x está relacionada con y mediante la relación R);
$[x]$	clase de equivalencia que contiene a x;
R^{-1}	relación inversa [todo (x, y) que está en R];
$R_2 \circ R_1$	composición de relaciones;
$x \preceq y$	$x\,R\,y$;

Funciones

$f(x)$	valor asignado a x;
$f: X \rightarrow Y$	función de X a Y;
$f \circ g$	composición de f y g;
f^{-1}	función inversa [todo (y, x) con (x, y) que está en f];
$f(n) = O(g(n))$	$\lvert f(n)\rvert \leq C\lvert g(n)\rvert$ para n suficientemente grande;
$f(n) = \Omega(g(n))$	$c\lvert g(n)\rvert \leq \lvert f(n)\rvert$ para n suficientemente grande;
$f(n) = \Theta(g(n))$	$c\lvert g(n)\rvert \leq \lvert f(n)\rvert \leq C\lvert g(n)\rvert$ para n suficientemente grande;

Conteo

$C(n, r)$	número de combinaciones r de un conjunto de n elementos $(n!/[(n-r)!r!])$;
$P(n, r)$	número de permutaciones r de un conjunto de n elementos $[n(n-1)\cdots(n-r+1)]$;

Gráficas

$G = (V, E)$	gráfica G con conjunto de vértices V y conjunto de aristas E;
(v, w)	arista;
$\delta(v)$	grado del vértice v;
$(v_1,\ldots,v_n)$	trayectoria de v_1 a v_n;
$(v_1,\ldots,v_n),\ v_1 = v_n$	ciclo;
K_n	gráfica completa en n vértices;
$K_{m,n}$	gráfica completa bipartita en m y n vértices;
$w(i, j)$	peso de la arista (i, j);
F_{ij}	flujo en la arista (i, j);
C_{ij}	capacidad de la arista (i, j);
$(P, \overline{P})$	cortadura en una red;

Probabilidad

$P(x)$	probabilidad del resultado x;
$P(E)$	probabilidad del evento E;
$P(E \mid F)$	probabilidad condicional de E dado F $[P(E \cap F)/P(F)]$;

Institución Universitaria
Acreditada en Alta Calidad

python™

Quick Start Guide for Python in VS Code

The Python extension makes Visual Studio Code an excellent Python editor, works on any operating system, and is usable with a variety of Python interpreters.

Get started by installing:

- VS Code
- A Python Interpreter (any actively supported Python version)
- Python extension from the VS Code Marketplace

https://www.python.org/downloads/

https://code.visualstudio.com/

Playgrounds:
https://www.online-python.com/
https://www.w3schools.com/python/python_compiler.asp

datacamp

Python Basics

Python Cheat Sheet for Beginners

Learn Python online at www.DataCamp.com

How to use this cheat sheet

Python is the most popular programming language in data science. It is easy to learn and comes with a wide array of powerful libraries for data analysis. This cheat sheet provides beginners and intermediate users a guide to starting using python. Use it to jump-start your journey with python. If you want more detailed Python cheat sheets, check out the following cheat sheets below:

Accessing help and getting object types

```
1 + 1 # Everything after the hash symbol is ignored by Python
help(max) # Display the documentation for the max function
type('a') # Get the type of an object - this returns str
```

Importing packages

Python packages are a collection of useful tools developed by the open-source community. They extend the capabilities of the python language. To install a new package (for example, pandas), you can go to your command prompt and type in `pip install pandas`. Once a package is installed, you can import it as follows:

```
import pandas # Import a package without an alias
import pandas as pd # Import a package with an alias
from pandas import DataFrame # Import an object from a package
```

The working directory

The working directory is the default file path that python reads or saves files into. An example of the working directory is "C://file/path". The os library is needed to set and get the working directory.

```
import os # Import the operating system package
os.getcwd() # Get the current directory
os.setcwd("new/working/directory") # Set the working directory to a new file path
```

Operators

Arithmetic operators

```
102 + 37 # Add two numbers with +
102 - 37 # Subtract a number with -
4 * 6 # Multiply two numbers with *
22 / 7 # Divide a number by another with /
22 // 7 # Integer divide a number with //
3 ** 4 # Raise to the power with **
22 % 7 # Returns 1 # Get the remainder after division with %
```

Assignment operators

```
a = 5 # Assign a value to a
x[0] = 1 # Change the value of an item in a list
```

Numeric comparison operators

```
3 == 3 # Test for equality with ==
3 != 3 # Test for inequality with !=
3 > 1  # Test greater than with >
3 >= 3 # Test greater than or equal to with >=
3 < 4  # Test less than with <
3 <= 4 # Test less than or equal to with <=
```

Logical operators

```
~(2 == 2)          # Logical NOT with ~
(1 == 1) & (1 < 1) # Logical AND with &
(1 >= 1) | (1 < 1) # Logical OR with |
(1 != 1) ^ (1 < 1) # Logical XOR with ^
```

Getting started with lists

A list is an ordered and changeable sequence of elements. It can hold integers, characters, floats, strings, and even objects.

Creating lists

```
# Create lists with [], elements separated by commas
x = [1, 3, 2]
```

List functions and methods

```
x.sorted(x) # Return a sorted copy of the list e.g., [1,2,3]
x.sort() # Sorts the list in-place (replaces x)
reversed(x) # Reverse the order of elements in x e.g., [2,3,1]
x.reversed() # Reverse the list in-place
x.count(2) # Count the number of element 2 in the list
```

Selecting list elements

Python lists are zero-indexed (the first element has index 0). For ranges, the first element is included but the last is not.

```
# Define the list
x = ['a', 'b', 'c', 'd', 'e']
x[0] # Select the 0th element in the list
x[-1] # Select the last element in the list
x[1:3] # Select 1st (inclusive) to 3rd (exclusive)
x[2:] # Select the 2nd to the end
x[:3] # Select 0th to 3rd (exclusive)
```

Concatenating lists

```
# Define the x and y lists
x = [1, 3, 6]
y = [10, 15, 21]
x + y # Returns [1, 3, 6, 10, 15, 21]
3 * x # Returns [1, 3, 6, 1, 3, 6, 1, 3, 6]
```

Getting started with dictionaries

A dictionary stores data values in key-value pairs. That is, unlike lists which are indexed by position, dictionaries are indexed by their keys, the names of which must be unique.

Creating dictionaries

```
# Create a dictionary with {}
{'a': 1, 'b': 4, 'c': 9}
```

Dictionary functions and methods

```
x = {'a': 1, 'b': 2, 'c': 3} # Define the x dictionary
x.keys() # Get the keys of a dictionary, returns dict_keys(['a', 'b', 'c'])
x.values() # Get the values of a dictionary, returns dict_values([1, 2, 3])
```

Selecting dictionary elements

```
x['a'] # 1 # Get a value from a dictionary by specifying the key
```

NumPy arrays

NumPy is a python package for scientific computing. It provides multidimensional array objects and efficient operations on them. To import NumPy, you can run this Python code `import numpy as np`

Creating arrays

```
# Convert a python list to a NumPy array
np.array([1, 2, 3]) # Returns array([1, 2, 3])
# Return a sequence from start (inclusive) to end (exclusive)
np.arange(1,5) # Returns array([1, 2, 3, 4])
# Return a stepped sequence from start (inclusive) to end (exclusive)
np.arange(1,5,2) # Returns array([1, 3])
# Repeat values n times
np.repeat([1, 3, 6], 3) # Returns array([1, 1, 1, 3, 3, 3, 6, 6, 6])
# Repeat values n times
np.tile([1, 3, 6], 3) # Returns array([1, 3, 6, 1, 3, 6, 1, 3, 6])
```

Math functions and methods

All functions take an array as the input.

```
np.log(x) # Calculate logarithm
np.exp(x) # Calculate exponential
np.max(x) # Get maximum value
np.min(x) # Get minimum value
np.sum(x) # Calculate sum
np.mean(x) # Calculate mean
np.quantile(x, q) # Calculate q-th quantile
np.round(x, n) # Round to n decimal places
np.var(x) # Calculate variance
np.std(x) # Calculate standard deviation
```

Getting started with characters and strings

```
# Create a string with double or single quotes
"DataCamp"

# Embed a quote in string with the escape character \
"He said, \"DataCamp\""

# Create multi-line strings with triple quotes
"""
A Frame of Data
Tidy, Mine, Analyze It
Now You Have Meaning
Citation: https://mdsr-book.github.io/haikus.html
"""

str[0] # Get the character at a specific position
str[0:2] # Get a substring from starting to ending index (exclusive)
```

Combining and splitting strings

```
"Data" + "Framed" # Concatenate strings with +, this returns 'DataFramed'
3 * "data " # Repeat strings with *, this returns 'data data data '
'beekeepers'.split('e') # Split a string on a delimiter, returns ['b', '', 'k', '', 'p', 'rs']
```

Mutate strings

```
str = "Jack and Jill" # Define str
str.upper() # Convert a string to uppercase, returns 'JACK AND JILL'
str.lower() # Convert a string to lowercase, returns 'jack and jill'
str.title() # Convert a string to title case, returns 'Jack And Jill'
str.replace("J", "P") # Replaces matches of a substring with another, returns 'Pack and Pill'
```

Getting started with DataFrames

Pandas is a fast and powerful package for data analysis and manipulation in python. To import the package, you can use `import pandas as pd`. A pandas DataFrame is a structure that contains two-dimensional data stored as rows and columns. A pandas series is a structure that contains one-dimensional data.

Creating DataFrames

```
# Create a dataframe from a dictionary
pd.DataFrame({
 'a': [1, 2, 3],
 'b': np.array([4, 4, 6]),
 'c': ['x', 'x', 'y']
})

# Create a dataframe from a list of dictionaries
pd.DataFrame([
 {'a': 1, 'b': 4, 'c': 'x'},
 {'a': 1, 'b': 4, 'c': 'x'},
 {'a': 3, 'b': 6, 'c': 'y'}
])
```

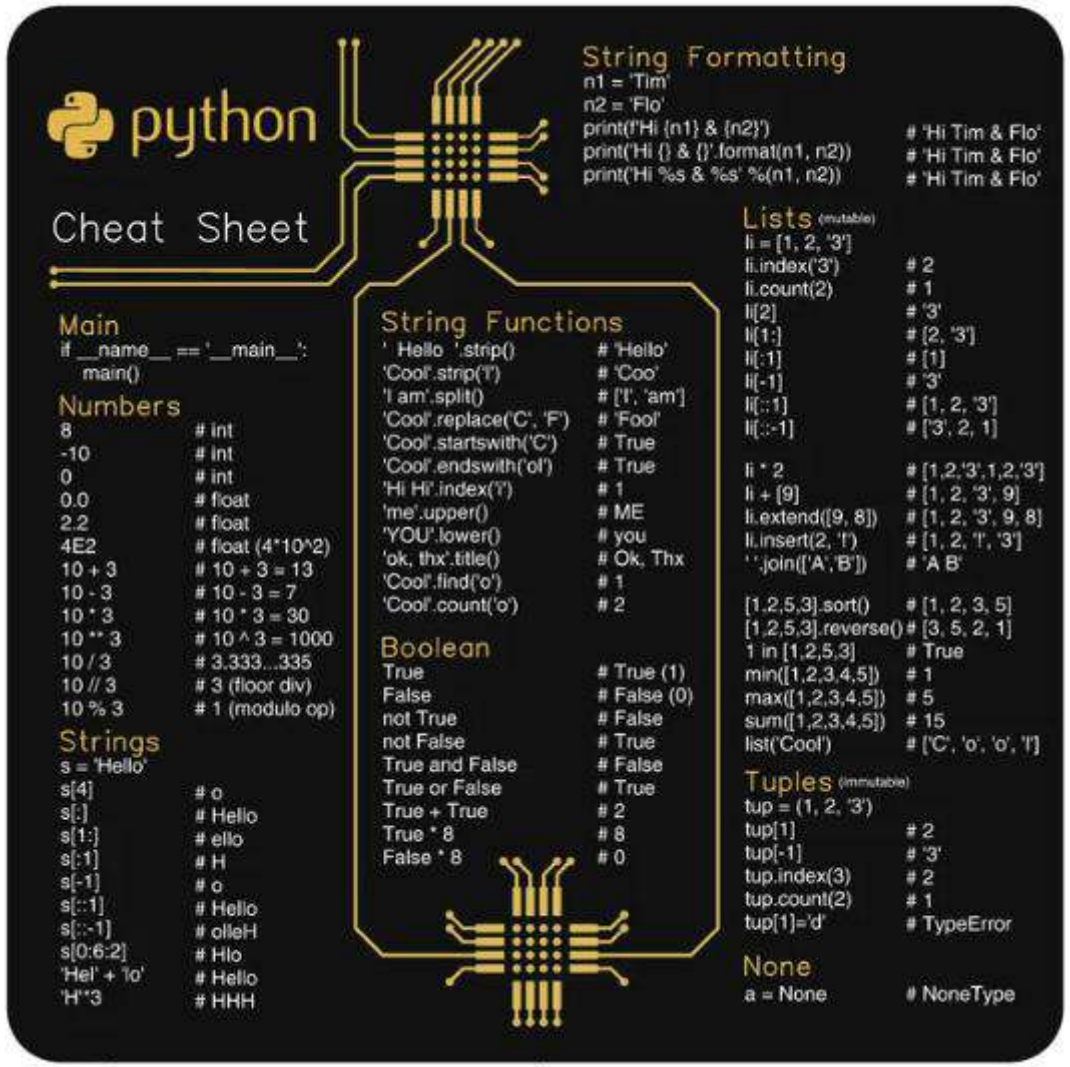
Selecting DataFrame Elements

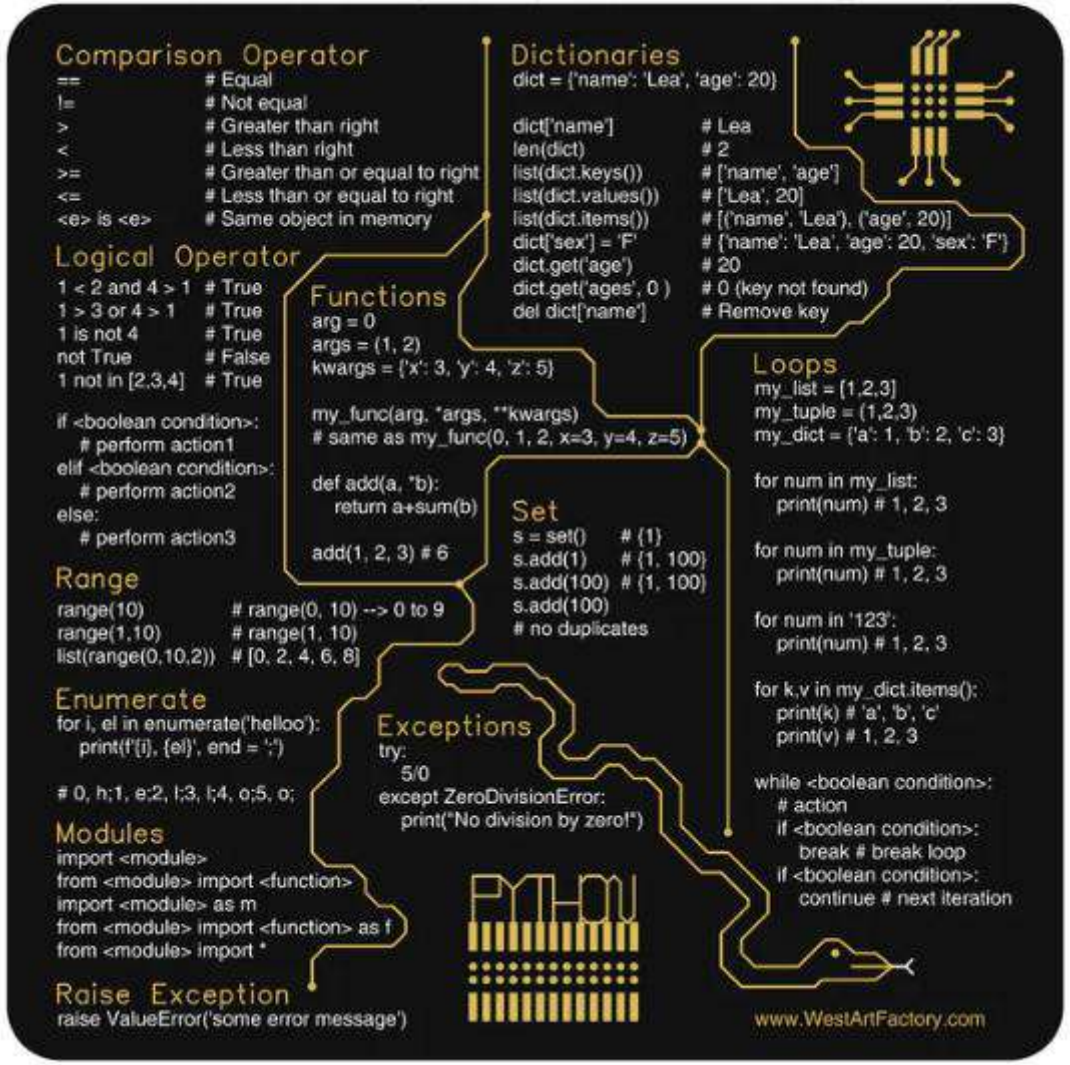
Select a row, column or element from a dataframe. Remember: all positions are counted from zero, not one.

```
# Select the 3rd row
df.iloc[3]
# Select one column by name
df['col']
# Select multiple columns by names
df[['col1', 'col2']]
# Select 2nd column
df.iloc[:, 2]
# Select the element in the 3rd row, 2nd column
df.iloc[3, 2]
```

Manipulating DataFrames

```
# Concatenate DataFrames vertically
pd.concat([df, df])
# Concatenate DataFrames horizontally
pd.concat([df,df],axis="columns")
# Get rows matching a condition
df.query('logical_condition')
# Drop columns by name
df.drop(columns=['col_name'])
# Rename columns
df.rename(columns={"oldname": "newname"})
# Add a new column
df.assign(temp_f=9 / 5 * df['temp_c'] + 32)

# Calculate the mean of each column
df.mean()
# Get summary statistics by column
df.agg(aggregation_function)
# Get unique rows
df.drop_duplicates()
# Sort by values in a column
df.sort_values(by='col_name')
# Get rows with largest values in a column
df.nlargest(n, 'col_name')
```





Institución Universitaria
Acreditada en Alta Calidad

TEMA 01

Principios y Técnicas de Conteo

190304003-1

MATEMÁTICAS PARA INFORMÁTICA AVANZADA

ROBERTO RAHAMUT SUTEU
DOCENTE

Institución Universitaria

Acreditada en Alta Calidad

¡Gracias!

Somos Innovación Tecnológica con Sentido Humano