MOOC de Criptología Matemática. Alfabetos, Listas y Permutaciones

Leandro Marín

Módulo I. Sesión 1.
Dificultad Baja

1 Alfabetos

2 Listas

3 Listas y Cadenas de Caracteres

4 Permutaciones

Criptografía y Alfabetos

- La criptografía es la ciencia que estudia la forma de escribir mensajes de forma oculta.

Criptografía y Alfabetos

- La criptografía es la ciencia que estudia la forma de escribir mensajes de forma oculta.
- Las razones por las que los mensajes se pretenden ocultar es diversa e históricamente hay ejemplos de todo tipo.

Criptografía y Alfabetos

- La criptografía es la ciencia que estudia la forma de escribir mensajes de forma oculta.
- Las razones por las que los mensajes se pretenden ocultar es diversa e históricamente hay ejemplos de todo tipo.
- Lo que es imprescindible para escribir un mensaje, ya sea de forma oculta o de cualquier otra forma, es un conjunto de símbolos.

Criptografía y Alfabetos

- La criptografía es la ciencia que estudia la forma de escribir mensajes de forma oculta.
- Las razones por las que los mensajes se pretenden ocultar es diversa e históricamente hay ejemplos de todo tipo.
- Lo que es imprescindible para escribir un mensaje, ya sea de forma oculta o de cualquier otra forma, es un conjunto de símbolos.
- Esos símbolos pueden ser letras, números, o cualquier otra pieza básica de nuestros mensajes.

Criptografía y Alfabetos

- La criptografía es la ciencia que estudia la forma de escribir mensajes de forma oculta.
- Las razones por las que los mensajes se pretenden ocultar es diversa e históricamente hay ejemplos de todo tipo.
- Lo que es imprescindible para escribir un mensaje, ya sea de forma oculta o de cualquier otra forma, es un conjunto de símbolos.
- Esos símbolos pueden ser letras, números, o cualquier otra pieza básica de nuestros mensajes.
- Este conjunto de piezas básicas es lo que llamaremos *alfabeto*.

Alfabetos y Números

- Este curso estudiará la criptografía desde el punto de vista matemático.

Alfabetos y Números

- Este curso estudiará la criptografía desde el punto de vista matemático.
- Puesto que nuestros mensajes estarán formados por símbolos del alfabeto, deberemos dotar a estos símbolos de significado matemático.

Alfabetos y Números

- Este curso estudiará la criptografía desde el punto de vista matemático.
- Puesto que nuestros mensajes estarán formados por símbolos del alfabeto, deberemos dotar a estos símbolos de significado matemático.
- Lo mas sencillo es utilizar representaciones numéricas para los símbolos y olvidarnos de la representación gráfica.

Alfabetos y Números

- Este curso estudiará la criptografía desde el punto de vista matemático.
- Puesto que nuestros mensajes estarán formados por símbolos del alfabeto, deberemos dotar a estos símbolos de significado matemático.
- Lo mas sencillo es utilizar representaciones numéricas para los símbolos y olvidarnos de la representación gráfica.
- Desde el punto de vista matemático, la representación gráfica es irrelevante, por lo que suele ser lo más sencillo considerar que el alfabeto está formado por los números $\{0, 1, 2, \cdots, n\}$ hasta el valor de n que sea necesario para representar todos los símbolos.

Alfabetos y Números

- Este curso estudiará la criptografía desde el punto de vista matemático.
- Puesto que nuestros mensajes estarán formados por símbolos del alfabeto, deberemos dotar a estos símbolos de significado matemático.
- Lo mas sencillo es utilizar representaciones numéricas para los símbolos y olvidarnos de la representación gráfica.
- Desde el punto de vista matemático, la representación gráfica es irrelevante, por lo que suele ser lo más sencillo considerar que el alfabeto está formado por los números $\{0, 1, 2, \cdots, n\}$ hasta el valor de n que sea necesario para representar todos los símbolos.
- Para hacer conversiones entre números y símbolos, asi como para manipularlos, pondremos los números en listas.

Listas en sage

- Una lista no es mas que un conjunto de objetos del mismo tipo y ordenados.

Listas en sage

- Una lista no es mas que un conjunto de objetos del mismo tipo y ordenados.
- En sage una lista se puede definir poniendo los objetos entre corchetes y separados por comas.

Listas en sage

- Una lista no es mas que un conjunto de objetos del mismo tipo y ordenados.
- En sage una lista se puede definir poniendo los objetos entre corchetes y separados por comas.
- Por ejemplo `L = [1,4,5,7]` asignará a la variable `L` la lista con los valores $1, 4, 5$ y 7.

Listas en sage

- Una lista no es mas que un conjunto de objetos del mismo tipo y ordenados.
- En sage una lista se puede definir poniendo los objetos entre corchetes y separados por comas.
- Por ejemplo `L = [1,4,5,7]` asignará a la variable `L` la lista con los valores $1, 4, 5$ y 7.
- Los elementos en la lista están numerados desde 0 hasta la longitud de la lista -1.

Listas en sage

- Una lista no es mas que un conjunto de objetos del mismo tipo y ordenados.
- En sage una lista se puede definir poniendo los objetos entre corchetes y separados por comas.
- Por ejemplo `L = [1,4,5,7]` asignará a la variable `L` la lista con los valores $1, 4, 5$ y 7.
- Los elementos en la lista están numerados desde 0 hasta la longitud de la lista -1.
- En el ejemplo anterior, `print L[0]` nos escribiría el valor 1 y `print L[2]` nos escribiría el valor 5.

El comando `range`

- Unas listas que se usan muy a menudo en sage son las que nos dan sucesiones de números del tipo `[0,1,2,3,4,5,6]`.

El comando `range`

- Unas listas que se usan muy a menudo en sage son las que nos dan sucesiones de números del tipo `[0,1,2,3,4,5,6]`.
- Para especificar este tipo de listas, especialmente si son largas, sería muy incómodo tener que escribir todos los números uno a uno.

El comando `range`

- Unas listas que se usan muy a menudo en sage son las que nos dan sucesiones de números del tipo `[0,1,2,3,4,5,6]`.
- Para especificar este tipo de listas, especialmente si son largas, sería muy incómodo tener que escribir todos los números uno a uno.
- El comando `range` nos soluciona el problema, si escribimos `L = range(7)` asignaraá a la variable `L` los números desde el 0 y estrictamente menores que 7, es decir `range(7)` es equivalente a `[0,1,2,3,4,5,6]`.

El comando `range`

- Unas listas que se usan muy a menudo en sage son las que nos dan sucesiones de números del tipo `[0,1,2,3,4,5,6]`.
- Para especificar este tipo de listas, especialmente si son largas, sería muy incómodo tener que escribir todos los números uno a uno.
- El comando `range` nos soluciona el problema, si escribimos `L = range(7)` asignaraá a la variable `L` los números desde el 0 y estrictamente menores que 7, es decir `range(7)` es equivalente a `[0,1,2,3,4,5,6]`.
- Es importante tener presente que si se especifica un único parámetro en `range(n)` los números empezarán siempre en 0 y terminarán en $n-1$, sin llegar a n.

El comando `range`

- Unas listas que se usan muy a menudo en sage son las que nos dan sucesiones de números del tipo `[0,1,2,3,4,5,6]`.
- Para especificar este tipo de listas, especialmente si son largas, sería muy incómodo tener que escribir todos los números uno a uno.
- El comando `range` nos soluciona el problema, si escribimos `L = range(7)` asignaraá a la variable `L` los números desde el 0 y estrictamente menores que 7, es decir `range(7)` es equivalente a `[0,1,2,3,4,5,6]`.
- Es importante tener presente que si se especifica un único parámetro en `range(n)` los números empezarán siempre en 0 y terminarán en $n-1$, sin llegar a n.
- Esto se puede modificar introduciento más parámetros, que no son obligatorios. De momento vamos a utilizarlo de esta forma que es la mas habitual.

Recorriendo listas I

- Aunque en este curso supondremos que se tiene familiaridad con la programación, vamos a recordar algunas estructuras básicas de la programación imperativa en sage.

Recorriendo listas I

- Aunque en este curso supondremos que se tiene familiaridad con la programación, vamos a recordar algunas estructuras básicas de la programación imperativa en sage.
- Una de las más utilizadas es el lazo `for` que nos permite recorrer todos los valores de una lista para hacer operaciones con los elementos de forma individual.

Recorriendo listas I

- Aunque en este curso supondremos que se tiene familiaridad con la programación, vamos a recordar algunas estructuras básicas de la programación imperativa en sage.
- Una de las más utilizadas es el lazo `for` que nos permite recorrer todos los valores de una lista para hacer operaciones con los elementos de forma individual.
- Por ejemplo, el siguiente programa

```
L = [1,8,12,5]
for x in L:
  print x
```

definirá la lista `L` con los valores $1, 8, 12$ y 5. Luego recorrerá los valores de la lista uno por uno y los escribirá en la consola.

Recorriendo lista II

- Notemos por una parte los dos puntos que hay en la línea del `for` que nos permite iniciar un bloque de código.

Recorriendo lista II

- Notemos por una parte los dos puntos que hay en la línea del `for` que nos permite iniciar un bloque de código.
- Y también la obligatoriedad de indentar el contenido del bloque de código que se debe ejecutar en cada ciclo del `for`, en este caso `print x`.

Recorriendo lista II

- Notemos por una parte los dos puntos que hay en la línea del `for` que nos permite iniciar un bloque de código.
- Y también la obligatoriedad de indentar el contenido del bloque de código que se debe ejecutar en cada ciclo del `for`, en este caso `print x`.
- Vamos a observarlos en este segundo ejemplo:

```
for x in [1,8,12,5]:
  y = 3*x-5
  print y
```

- Como se puede ver, volvemos a poner dos puntos y las dos operaciones que se hacen en cada ciclo del lazo for están indentadas a la misma altura.

Recorriendo lista II

- Notemos por una parte los dos puntos que hay en la línea del `for` que nos permite iniciar un bloque de código.
- Y también la obligatoriedad de indentar el contenido del bloque de código que se debe ejecutar en cada ciclo del `for`, en este caso `print x`.
- Vamos a observarlos en este segundo ejemplo:

```
for x in [1,8,12,5]:
  y = 3*x-5
  print y
```

- Como se puede ver, volvemos a poner dos puntos y las dos operaciones que se hacen en cada ciclo del lazo for están indentadas a la misma altura.
- Observemos que no hemos dado ningún nombre a la lista. Esto es válido, siempre que podamos usar el nombre de una lista, podemos usar la lista en sí misma.

Transformaciones de listas I

- Supongamos que queremos la lista de números y queremos calcular la lista formada por sus cuadrados.

Transformaciones de listas I

- Supongamos que queremos la lista de números y queremos calcular la lista formada por sus cuadrados.
- Esto se puede hacer creando una nueva lista vacía `L = [ ]` y añadiendo cada uno de los nuevos elementos con el comando `append`. Por ejemplo:

```
M = [1,8,4,5]
L = []
for x in M:
  L.append(x**2)
print L
```

Transformaciones de listas I

- Supongamos que queremos la lista de números y queremos calcular la lista formada por sus cuadrados.
- Esto se puede hacer creando una nueva lista vacía `L = [ ]` y añadiendo cada uno de los nuevos elementos con el comando `append`. Por ejemplo:

```
M = [1,8,4,5]
L = []
for x in M:
  L.append(x**2)
print L
```

- Observemos que `print L` lo hemos puesto sin la indentación del lazo `for`, esto es debido a que solo se debe ejecutar al final, no en cada ciclo.

Transformaciones de listas II

- En matemáticas existen dos notaciones para definir conjuntos, una es dar todos sus elementos expícitamente $M = \{1, 8, 4, 5\}$ y otra es especificar la fórmula que define el conjunto $L = \{x^2 : x \in M\}$.

Transformaciones de listas II

- En matemáticas existen dos notaciones para definir conjuntos, una es dar todos sus elementos expícitamente $M = \{1, 8, 4, 5\}$ y otra es especificar la fórmula que define el conjunto $L = \{x^2 : x \in M\}$.
- En sage también podemos utilizar esta segunda forma mucho más compacta, lo cual simplifica mucho nuestro código.

Transformaciones de listas II

- En matemáticas existen dos notaciones para definir conjuntos, una es dar todos sus elementos expícitamente $M = \{1, 8, 4, 5\}$ y otra es especificar la fórmula que define el conjunto $L = \{x^2 : x \in M\}$.
- En sage también podemos utilizar esta segunda forma mucho más compacta, lo cual simplifica mucho nuestro código.

```
M = [1,8,4,5]
L = [x**2 for x in M]
```

Transformaciones de listas II

- En matemáticas existen dos notaciones para definir conjuntos, una es dar todos sus elementos expícitamente $M = \{1, 8, 4, 5\}$ y otra es especificar la fórmula que define el conjunto $L = \{x^2 : x \in M\}$.
- En sage también podemos utilizar esta segunda forma mucho más compacta, lo cual simplifica mucho nuestro código.
- ```
  M = [1,8,4,5]
  L = [x**2 for x in M]
  ```
- O simplemente `L = [x**2 for x in [1,8,4,5]]`.
  ```

Cadenas de caracteres

- Un objeto similar a las listas son las cadenas de caracteres.

Cadenas de caracteres

- Un objeto similar a las listas son las cadenas de caracteres.
- Como en muchos otros lenguajes se pueden especificar con el texto entre comillas.

Cadenas de caracteres

- Un objeto similar a las listas son las cadenas de caracteres.
- Como en muchos otros lenguajes se pueden especificar con el texto entre comillas.
- Por ejemplo `print "Hola Mundo"` nos escribirá en pantalla el típico mensaje *Hola Mundo*.

Cadenas de caracteres

- Un objeto similar a las listas son las cadenas de caracteres.
- Como en muchos otros lenguajes se pueden especificar con el texto entre comillas.
- Por ejemplo `print "Hola Mundo"` nos escribirá en pantalla el típico mensaje *Hola Mundo*.
- Aunque no es exactamente una lista de símbolos, se comporta de forma similar, y podemos poner:

```
for x in "Esto es un mensaje":
  print x
```

Cadenas de caracteres

- Un objeto similar a las listas son las cadenas de caracteres.
- Como en muchos otros lenguajes se pueden especificar con el texto entre comillas.
- Por ejemplo `print "Hola Mundo"` nos escribirá en pantalla el típico mensaje *Hola Mundo*.
- Aunque no es exactamente una lista de símbolos, se comporta de forma similar, y podemos poner:

```
for x in "Esto es un mensaje":
  print x
```

- O incluso poner subíndices

```
texto = "Esto es un mensaje"
print texto[0]
print texto[1]
```

Cadenas de Caracteres y Alfabetos

- Podemos utilizar cadenas de caracteres para definir alfabetos y para hacer las transformaciones entre símbolos y números.

Cadenas de Caracteres y Alfabetos

- Podemos utilizar cadenas de caracteres para definir alfabetos y para hacer las transformaciones entre símbolos y números.
- Supongamos por ejemplo que queremos usar como alfabeto las letras mayúsculas y el espacio del alfabeto inglés, podemos hacerlo poniendo:

```
alfabeto = "ABCDEFGHIJKLMNOPQRSTUVWXYZ "
for x in range(27):
  print x,alfabeto[x]
```

Cadenas de Caracteres y Alfabetos

- Podemos utilizar cadenas de caracteres para definir alfabetos y para hacer las transformaciones entre símbolos y números.
- Supongamos por ejemplo que queremos usar como alfabeto las letras mayúsculas y el espacio del alfabeto inglés, podemos hacerlo poniendo:

```
alfabeto = "ABCDEFGHIJKLMNOPQRSTUVWXYZ "
for x in range(27):
  print x,alfabeto[x]
```

- Como vemos en el ejemplo, dado un número `x` entre 0 y la longitud del alfabeto -1, podemos ver el símbolo al que corresponde con `alfabeto[x]` .

Cadenas de Caracteres y Listas

- Aunque las cadenas de caracteres son casi igual que las listas, no lo son del todo. Veamos la diferencia haciendo la siguiente transformación:

```
alf1 = "ABCDE"
alf2 = [x for x in alf1]
print alf1
print alf2
```

Cadenas de Caracteres y Listas

- Aunque las cadenas de caracteres son casi igual que las listas, no lo son del todo. Veamos la diferencia haciendo la siguiente transformación:

```
alf1 = "ABCDE"
alf2 = [x for x in alf1]
print alf1
print alf2
```

- Aunque las variables `alf1` y `alf2` comparten propiedades, la segunda sí es una lista.

Cadenas de Caracteres y Listas

- Aunque las cadenas de caracteres son casi igual que las listas, no lo son del todo. Veamos la diferencia haciendo la siguiente transformación:

```
alf1 = "ABCDE"
alf2 = [x for x in alf1]
print alf1
print alf2
```

- Aunque las variables `alf1` y `alf2` comparten propiedades, la segunda sí es una lista.
- Una de las propiedades que no tienen las cadenas de caracteres y sí las listas es la función `index`.

La función `index`

- La fución `index` nos dice la posición de un determinado elemento en una lista.

La función `index`

- La fución `index` nos dice la posición de un determinado elemento en una lista.
- Esto nos puede servir para calcular el valor numérico de un símbolo de nuestro alfabeto si lo tenemos en forma de lista. Por ejemplo:

```
alf = [x for x in "ABCDEFGHIJKLMNOPQRSTUVWXYZ "]
print alf.index['X']
```

La función `index`

- La fución `index` nos dice la posición de un determinado elemento en una lista.
- Esto nos puede servir para calcular el valor numérico de un símbolo de nuestro alfabeto si lo tenemos en forma de lista. Por ejemplo:

```
alf = [x for x in "ABCDEFGHIJKLMNOPQRSTUVWXYZ "]
print alf.index['X']
```

- Nos dirá que `X` está en la posición número 23 de nuestro alfabeto.

Restaurando Cadenas de Caracteres

- Hemos visto que nos puede interesar tener una cadena de caracteres en forma de lista, pero a veces nos puede interesar también volver atrás y restaurar la cadena de caracteres, especialmente para poder sacarla por la consola.

Restaurando Cadenas de Caracteres

- Hemos visto que nos puede interesar tener una cadena de caracteres en forma de lista, pero a veces nos puede interesar también volver atrás y restaurar la cadena de caracteres, especialmente para poder sacarla por la consola.
- Por ejemplo, si hemos separado las letras de una cadena de caracteres, `L = ['H','o','l','a']` y queremos volver a juntarlas, podemos utiliar `cad = ''.join(L)`.

Restaurando Cadenas de Caracteres

- Hemos visto que nos puede interesar tener una cadena de caracteres en forma de lista, pero a veces nos puede interesar también volver atrás y restaurar la cadena de caracteres, especialmente para poder sacarla por la consola.
- Por ejemplo, si hemos separado las letras de una cadena de caracteres, `L = ['H','o','l','a']` y queremos volver a juntarlas, podemos utiliar `cad = ''.join(L)`.
- Esto nos juntará las letras de `L` en una cadena de caracteres `cad` que podemos imprimir con `print cad`.

El código ASCII

- Una forma habitual de representar los caracteres alfanuméricos en memoria es utilizando un número entre 0 y 255.

El código ASCII

- Una forma habitual de representar los caracteres alfanuméricos en memoria es utilizando un número entre 0 y 255.
- Estos 256 símbolos reciben el nombre de código ASCII y antes de la aparición de sistemas como el UTF-8 era casi el único sistema utilizado.

El código ASCII

- Una forma habitual de representar los caracteres alfanuméricos en memoria es utilizando un número entre 0 y 255.
- Estos 256 símbolos reciben el nombre de código ASCII y antes de la aparición de sistemas como el UTF-8 era casi el único sistema utilizado.
- Si queremos saber el número que corresponde a un símbolo concreto en este código ASCII podemos utilizar `ord`

```
for x in "Hola":
  print x,ord(x)
```

Nos dará como resultado los valores numéricos correspondientes a estos símbolos:

```
H 72
o 111
l 108
a 97
```

Introducción

- Una forma de transformar mensajes para hacerlos ilegibles a un atacante, es proceder a intercambiar los valores del alfabeto por otros.

Introducción

- Una forma de transformar mensajes para hacerlos ilegibles a un atacante, es proceder a intercambiar los valores del alfabeto por otros.
- El emisor del mensaje debe disponer de una tabla de transformación que nos diga cual es el símbolo correspondiente a cada uno de los orginales.

Introducción

- Una forma de transformar mensajes para hacerlos ilegibles a un atacante, es proceder a intercambiar los valores del alfabeto por otros.
- El emisor del mensaje debe disponer de una tabla de transformación que nos diga cual es el símbolo correspondiente a cada uno de los orginales.
- Pero no todas las transformaciones son válidas, por ejemplo, si procedemos a transformar la letra H y la letra Z en la letra A, el receptor del mensaje, al intentar deshacer el cambio, no podría distinguir si la A que ve en el mensaje cifrado corresponde a una H o a una Z.

Introducción

- Una forma de transformar mensajes para hacerlos ilegibles a un atacante, es proceder a intercambiar los valores del alfabeto por otros.
- El emisor del mensaje debe disponer de una tabla de transformación que nos diga cual es el símbolo correspondiente a cada uno de los orginales.
- Pero no todas las transformaciones son válidas, por ejemplo, si procedemos a transformar la letra H y la letra Z en la letra A, el receptor del mensaje, al intentar deshacer el cambio, no podría distinguir si la A que ve en el mensaje cifrado corresponde a una H o a una Z.
- Eso es lo que matemáticamente hablando se denomina que la aplicación debe ser inyectiva. Para conjuntos finitos esta condición es equivalente a que la transformación sea invertible.

Introducción

- Una forma de transformar mensajes para hacerlos ilegibles a un atacante, es proceder a intercambiar los valores del alfabeto por otros.
- El emisor del mensaje debe disponer de una tabla de transformación que nos diga cual es el símbolo correspondiente a cada uno de los orginales.
- Pero no todas las transformaciones son válidas, por ejemplo, si procedemos a transformar la letra H y la letra Z en la letra A, el receptor del mensaje, al intentar deshacer el cambio, no podría distinguir si la A que ve en el mensaje cifrado corresponde a una H o a una Z.
- Eso es lo que matemáticamente hablando se denomina que la aplicación debe ser inyectiva. Para conjuntos finitos esta condición es equivalente a que la transformación sea invertible.
- Vamos a fomalizarlo matemáticamente.

Definición

- Sea X un conjunto finito (por ejemplo un alfabeto). Una permutación de X es una aplicación $\sigma : X \to X$ inyectiva, es decir, que si $x \neq y$ entonces $\sigma(x) \neq \sigma(y)$.

Definición

- Sea X un conjunto finito (por ejemplo un alfabeto). Una permutación de X es una aplicación $\sigma : X \to X$ inyectiva, es decir, que si $x \neq y$ entonces $\sigma(x) \neq \sigma(y)$.
- Para definir σ debemos dar el valor de cada uno de los elementos de X. Por ejemplo, si $X = \{A, B, C, D\}$ escribiremos

$$\sigma = \begin{pmatrix} A & B & C & D \\ D & A & C & B \end{pmatrix}$$

para indicar que A se transforma en D, B en A, C en C y D en B. Es decir, pondremos en la fila superior los valores de X y en la fila inferior los elementos en los cuales son transformados mediante la permutación σ.

Definición

- Sea X un conjunto finito (por ejemplo un alfabeto). Una permutación de X es una aplicación $\sigma : X \to X$ inyectiva, es decir, que si $x \neq y$ entonces $\sigma(x) \neq \sigma(y)$.
- Para definir σ debemos dar el valor de cada uno de los elementos de X. Por ejemplo, si $X = \{A, B, C, D\}$ escribiremos

$$\sigma = \begin{pmatrix} A & B & C & D \\ D & A & C & B \end{pmatrix}$$

para indicar que A se transforma en D, B en A, C en C y D en B. Es decir, pondremos en la fila superior los valores de X y en la fila inferior los elementos en los cuales son transformados mediante la permutación σ.
- El número de permutaciones posibles de n elementos es el producto $1 \cdot 2 \cdot 3 \cdots n$. Dicho número se denomina n factorial y se escribe $n!$.

Permutaciones y Listas

- Una forma de representar una permutación en sage es mediante listas.

Permutaciones y Listas

- Una forma de representar una permutación en sage es mediante listas.
- Si queremos escribir una reordenación de los números $\{0, 1, 2, 3, 4, 5\}$ podemos escribir `S = [5,2,1,0,3,4]`

Permutaciones y Listas

- Una forma de representar una permutación en sage es mediante listas.
- Si queremos escribir una reordenación de los números $\{0, 1, 2, 3, 4, 5\}$ podemos escribir `S = [5,2,1,0,3,4]`
- Para calcular el valor asignado a 3 ponemos `S[3]`

Permutaciones y Listas

- Una forma de representar una permutación en sage es mediante listas.
- Si queremos escribir una reordenación de los números $\{0, 1, 2, 3, 4, 5\}$ podemos escribir `S = [5,2,1,0,3,4]`
- Para calcular el valor asignado a 3 ponemos `S[3]`
- Si queremos calcular la transformación inversa podemos usar `S.index(5)`, que nos dará la posición del 5 dentro de la lista. Puesto que hemos dado los valores ordenados, eso corresponderá al 0, que es el valor que nos proporciona `S[0]`.

Permutaciones y Listas

- Una forma de representar una permutación en sage es mediante listas.
- Si queremos escribir una reordenación de los números $\{0,1,2,3,4,5\}$ podemos escribir `S = [5,2,1,0,3,4]`
- Para calcular el valor asignado a 3 ponemos `S[3]`
- Si queremos calcular la transformación inversa podemos usar `S.index(5)`, que nos dará la posición del 5 dentro de la lista. Puesto que hemos dado los valores ordenados, eso corresponderá al 0, que es el valor que nos proporciona `S[0]`.
- Esta será una forma muy habitual de darnos permutaciones en sage puesto que en muchos casos tendremos permutaciones de los números entre 0 y $n-1$.

Listas Aletorias

- Supongamos que tenemos una lista `L` y queremos obtener una reordenación aleatoria de ella, entonces podemos aplicar `shuffle(L)` y `L` será modificada de forma aletoria.

Listas Aletorias

- Supongamos que tenemos una lista `L` y queremos obtener una reordenación aleatoria de ella, entonces podemos aplicar `shuffle(L)` y `L` será modificada de forma aletoria.
- Veámoslo con un ejemplo:

```
L = range(10)
shuffle(L)
print L
```

Listas Aletorias

- Supongamos que tenemos una lista `L` y queremos obtener una reordenación aleatoria de ella, entonces podemos aplicar `shuffle(L)` y `L` será modificada de forma aletoria.
- Veámoslo con un ejemplo:

```
L = range(10)
shuffle(L)
print L
```

- Al tratarse de un resultado aletorio, cada vez que ejecutemos este código nos puede dar un resultado distinto de entre los $10! = 10 \cdot 9 \cdots 2 \cdot 1 = 3628800$ valores posibles.

Cifrados Monoalfabéticos

- Si tenemos una permutación de un alfabeto, podemos cifrar mensajes sustituyendo cada letra del mensaje original mediante dicha permutación. Este tipo de cifrado se conoce como cifrado monoalfabético.

Cifrados Monoalfabéticos

- Si tenemos una permutación de un alfabeto, podemos cifrar mensajes sustituyendo cada letra del mensaje original mediante dicha permutación. Este tipo de cifrado se conoce como cifrado monoalfabético.
- Veámos cómo se podrá implementar:

```
P = [x for x in "ABCDEFGHIJKLMNOPQRSTUVWXYZ "]
C = [x for x in "AFGLIOJZHR EYCXUPSQTNKVMDWB"]
claro = "HOLA MUNDO"
cifrado = ''.join([C[P.index(x)] for x in claro])
```

Cifrados Monoalfabéticos

- Si tenemos una permutación de un alfabeto, podemos cifrar mensajes sustituyendo cada letra del mensaje original mediante dicha permutación. Este tipo de cifrado se conoce como cifrado monoalfabético.
- Veámos cómo se podrá implementar:

```
P = [x for x in "ABCDEFGHIJKLMNOPQRSTUVWXYZ "]
C = [x for x in "AFGLIOJZHR EYCXUPSQTNKVMDWB"]
claro = "HOLA MUNDO"
cifrado = ''.join([C[P.index(x)] for x in claro])
```

- Como podemos ver, para cada símbolo del mensaje obtenemos su posición el el alfabeto con `P.index(x)` y luego calculamos su nuevo valor aplicando `C[P.index(x)]`. El resultado lo reagrupamos en una cadena de caracteres.

La Permutación Inversa

- Dada una permutación σ, podemos definir la permutación inversa σ^{-1} que es la única que cumple que $\sigma^{-1}(\sigma(x)) = x$ para todo valor de x.

La Permutación Inversa

- Dada una permutación σ, podemos definir la permutación inversa σ^{-1} que es la única que cumple que $\sigma^{-1}(\sigma(x)) = x$ para todo valor de x.
- Dicho de otro modo $\sigma^{-1}(x) = y$ precisamente si $\sigma(y) = x$.

La Permutación Inversa

- Dada una permutación σ, podemos definir la permutación inversa σ^{-1} que es la única que cumple que $\sigma^{-1}(\sigma(x)) = x$ para todo valor de x.
- Dicho de otro modo $\sigma^{-1}(x) = y$ precisamente si $\sigma(y) = x$.
- Por ejemplo, dada la permuación:

$$\sigma = \begin{pmatrix} A & B & C & D \\ D & A & C & B \end{pmatrix}$$

su inversa sería

$$\sigma^{-1} = \begin{pmatrix} A & B & C & D \\ B & D & C & A \end{pmatrix}$$

La Permutación Inversa

- Dada una permutación σ, podemos definir la permutación inversa σ^{-1} que es la única que cumple que $\sigma^{-1}(\sigma(x)) = x$ para todo valor de x.
- Dicho de otro modo $\sigma^{-1}(x) = y$ precisamente si $\sigma(y) = x$.
- Por ejemplo, dada la permuación:

$$\sigma = \begin{pmatrix} A & B & C & D \\ D & A & C & B \end{pmatrix}$$

su inversa sería

$$\sigma^{-1} = \begin{pmatrix} A & B & C & D \\ B & D & C & A \end{pmatrix}$$

- Lo único que hay que hacer para calcularla es leer la permutación de abajo a arriba en lugar de en el sentido habitual.

Permutación Inversa y Descifrado

- Las permutaciones inversas sirven para deshacer el camino hecho por una permutación.

Permutación Inversa y Descifrado

- Las permutaciones inversas sirven para deshacer el camino hecho por una permutación.
- Si tenemos un cifrado monoalfabético dado por una permutación σ, para descifrar haremos la misma operación pero con la permutación inversa σ^{-1}.

Permutación Inversa y Descifrado

- Las permutaciones inversas sirven para deshacer el camino hecho por una permutación.
- Si tenemos un cifrado monoalfabético dado por una permutación σ, para descifrar haremos la misma operación pero con la permutación inversa σ^{-1}.
- Así el emisor y el receptor deben disponer de permutaciones inversas la una de la otra.

S-Boxes

- Los cifrados monoalfabéticos con alfabetos pequeños son inseguros.

S-Boxes

- Los cifrados monoalfabéticos con alfabetos pequeños son inseguros.
- Para hacer un cifrado monoalfabético seguro mediante tablas necesitaríamos alfabetos enormes y tablas extremadamente grandes, tanto que son materialmente imposibles.

S-Boxes

- Los cifrados monoalfabéticos con alfabetos pequeños son inseguros.
- Para hacer un cifrado monoalfabético seguro mediante tablas necesitaríamos alfabetos enormes y tablas extremadamente grandes, tanto que son materialmente imposibles.
- Sin embargo, las permutaciones son la base de muchos sistemas criptográficos, que se definen como una serie de operaciones matemáticas basadas en fórmulas aritméticas combinadas con otras basadas en permutaciones dadas mediante tablas.

S-Boxes

- Los cifrados monoalfabéticos con alfabetos pequeños son inseguros.
- Para hacer un cifrado monoalfabético seguro mediante tablas necesitaríamos alfabetos enormes y tablas extremadamente grandes, tanto que son materialmente imposibles.
- Sin embargo, las permutaciones son la base de muchos sistemas criptográficos, que se definen como una serie de operaciones matemáticas basadas en fórmulas aritméticas combinadas con otras basadas en permutaciones dadas mediante tablas.
- Estas partes que se definen mediante tablas de pequeño tamaño se suelen denominar *Selection Boxes* o *S-Boxes*.

S-Boxes

- Los cifrados monoalfabéticos con alfabetos pequeños son inseguros.
- Para hacer un cifrado monoalfabético seguro mediante tablas necesitaríamos alfabetos enormes y tablas extremadamente grandes, tanto que son materialmente imposibles.
- Sin embargo, las permutaciones son la base de muchos sistemas criptográficos, que se definen como una serie de operaciones matemáticas basadas en fórmulas aritméticas combinadas con otras basadas en permutaciones dadas mediante tablas.
- Estas partes que se definen mediante tablas de pequeño tamaño se suelen denominar *Selection Boxes* o *S-Boxes*.
- Las S-Boxes son parte de la definición del sistema y permanecen constantes en todas las implementaciones.

S-Boxes

- Los cifrados monoalfabéticos con alfabetos pequeños son inseguros.
- Para hacer un cifrado monoalfabético seguro mediante tablas necesitaríamos alfabetos enormes y tablas extremadamente grandes, tanto que son materialmente imposibles.
- Sin embargo, las permutaciones son la base de muchos sistemas criptográficos, que se definen como una serie de operaciones matemáticas basadas en fórmulas aritméticas combinadas con otras basadas en permutaciones dadas mediante tablas.
- Estas partes que se definen mediante tablas de pequeño tamaño se suelen denominar *Selection Boxes* o *S-Boxes*.
- Las S-Boxes son parte de la definición del sistema y permanecen constantes en todas las implementaciones.
- Al ser siempre las mismas para el algoritmo deben elegirse con mucha atención. Sus propiedades son fundamentales para la seguridad del sistema.