Práctica 1

Ejercicio 1

Un `Symbol` es un identificador inmutable y único en Ruby

- **Con literales**: Los símbolos se crean utilizando el prefijo `:` seguido del nombre del símbolo
- **Con el constructor** `new`: No es posible crear símbolos con `new`, ya que los símbolos son inmutables y se crean internamente de manera única

Las `String` son secuencias de caracteres.

- **Con literales**: Se pueden definir usando comillas simples o dobles
- **Con el constructor** `new`: También puedes crear una cadena usando `new`.

Un `Array` es una colección ordenada de elementos.

- **Con literales**: Los arrays se pueden crear usando corchetes `[]`
- **Con el constructor** `new`: También puedes crearlos utilizando el constructor `Array.new`.

Un `Hash` es una colección de pares clave-valor.

- **Con literales**: Se pueden crear hashes usando llaves `{}`.
- **Con el constructor** `new`: Puedes crear hashes con `Hash.new`.

Ejercicio 2

2. ¿De qué forma(s) se puede convertir un objeto (cualquiera fuere su tipo o clase) en String?

- Utilizando el método `to_s`
- Usando el método `String()`

Ejercicio 3

3. ¿De qué forma(s) se puede convertir un objeto String en un símbolo?

- Utilizando el método `to_sym`
- Utilizando el método `intern` : es un alias de `to_sym`. Ambos hacen exactamente lo mismo.

Ejercicio 4

4. ¿Qué devuelve la siguiente comparación? ¿Por qué?

```
'TTPS Ruby'.object_id == 'TTPS Ruby'.object_id
```

En Ruby, cada vez que escribes una cadena literal, como `'TTPS Ruby'`, se crea un **nuevo objeto** de tipo `String`, incluso si las cadenas tienen el mismo contenido. Al escribir `'TTPS Ruby'` dos veces, estás creando **dos objetos diferentes**, cada uno con su propio `object_id`

En Ruby, el método `object_id` devuelve el identificador único del objeto en la memoria. Este ID es diferente para cada instancia de un objeto, incluso si el contenido del objeto es el mismo

---

Ejercicio 5

5. Escribí una función llamada `reemplazar` que, dado un `String` que recibe como parámetro, busque y reemplace en éste cualquier ocurrencia de `"{"` por `"do\n"` y cualquier ocurrencia de `"}"` por `"\nend"`, de modo que convierta los bloques escritos con llaves por bloques multilínea con **do** y `end`. Por ejemplo:

```
reemplazar("3.times { |i| puts i }")
# => "3.times do\n |i| puts i \nend"
```

Método `gsub` (global substitution) : El método más común para reemplazar caracteres o secuencias de caracteres en una cadena es `gsub`. Este método permite reemplazar todas las ocurrencias de un patrón por otro

- Sintaxis : gsub(patrón, reemplazo)
- El patrón puede ser una cadena o una expresión regular (regex)

```
cadena = "Hola Mundo"
cadena_modificada = cadena.gsub('o', '0')   # Reemplaza todas
puts cadena_modificada  # "H0la Mund0"
```

- Usa `gsub` para reemplazar todas las ocurrencias de un carácter o patrón.

- Usa `sub` si solo necesitas reemplazar la **primera** ocurrencia.
- Usa `tr` cuando necesitas reemplazar caracteres **individuales** de forma eficiente.
- Si quieres modificar la cadena original directamente, utiliza las versiones con `!`.

Ejercicio 6

6. Escribí una función que exprese en palabras la hora que recibe como parámetro, según las siguientes reglas:
    - Si el minuto está entre 0 y 10, debe decir “en punto”,
    - si el minuto está entre 11 y 20, debe decir “y cuarto”,
    - si el minuto está entre 21 y 34, debe decir “y media”,
    - si el minuto está entre 35 y 44, debe decir “menos veinticinco” con la hora siguiente,
    - si el minuto está entre 45 y 55, debe decir “menos cuarto” con la hora siguiente,
    - y si el minuto está entre 56 y 59, debe decir “Casi son las” con la hora siguiente

Sintaxis de función Time.new :

```
Time.new(year, month, day, hour, minute, second, utc_offset)
```

Ejercicio 7

7. Escribí una función llamada `contar` que reciba como parámetro dos `String` y que retorne la cantidad de veces que aparece el segundo `String` en el primero, en una búsqueda *case-insensitive* (sin distinguir mayúsculas o minúsculas). Por ejemplo:

```
contar("La casa de la esquina tiene la puerta roja y la ventana blanca
    .", "la")
# => 5
```

Duda : afecta en algo si la segunda cadena es case sensitive o insensitive?

- `downcase`: Convierte todas las letras a minúsculas para que la búsqueda no sea sensible a mayúsculas/minúsculas.
- `scan(subcadena)`: Devuelve un array con todas las ocurrencias de la subcadena en la cadena original.
- `count`: Cuenta el número de elementos en ese array, que corresponde al número de coincidencias.

---

8. Modificá la función anterior para que sólo considere como aparición del segundo `String` cuando se trate de palabras completas. Por ejemplo:

```
contar_palabras("La casa de la esquina tiene la puerta roja y la
    ventana blanca.", "la")
# => 4
```

Explicación de la sentencia `/\b#{Regexp.quote(subcadena)}\b/i`

1. **Delimitadores de la expresión regular (`/ ... /`)**:
   - En Ruby, las expresiones regulares se escriben entre barras (`/`). Todo lo que está entre ellas se considera parte del patrón de búsqueda.
2. `\b` **(Delimitador de palabra)**:

- `\b` es un **metacaracter** de expresiones regulares que indica un **límite de palabra**.
- Un límite de palabra es una posición entre un carácter alfanumérico (letra o número) y un carácter no alfanumérico (como un espacio, puntuación, o el inicio/final de la cadena).
- En este contexto, `\b` asegura que la subcadena que estamos buscando se trata como una **palabra completa**. Es decir, no será parte de otra palabra más grande.

3. `#{Regexp.quote(subcadena)}`
   - `#{}` es una forma de interpolar una variable o expresión en una cadena o expresión regular en Ruby.
   - `Regexp.quote(subcadena)` es una función que toma el valor de `subcadena` y **escapa** cualquier carácter especial que pudiera tener. En Ruby (y otros lenguajes), algunos caracteres tienen un significado especial en las expresiones regulares (por ejemplo, `.` para cualquier carácter o `*` para repetición). Si queremos buscar esos caracteres como texto literal, debemos escaparlos.
   - **Ejemplo**: Si `subcadena = "?"`, la expresión regular sin `Regexp.quote` podría interpretarlo como un carácter especial. Con `Regexp.quote`, el `?` será tratado como un carácter literal.
4. `\b` **(Delimitador de palabra)**:
   - Esta es la segunda aparición de `\b`, que asegura que el final de la subcadena también está en un límite de palabra. De esta manera, la subcadena completa está delimitada por límites de palabra al inicio y al final, garantizando que es una palabra independiente.
5. `i` **(Modificador case-insensitive)**:
   - El `i` que aparece después de la segunda barra `/` es un **modificador** de la expresión regular que hace que la búsqueda sea **insensible a mayúsculas y minúsculas** (case insensitive).
   - Esto significa que la expresión regular buscará coincidencias de la subcadena sin importar si las letras están en mayúsculas o minúsculas.
   - Ejemplo: Si `subcadena = "la"`, tanto `"La"`, `"la"`, o `"LA"` coincidirán.

Ejercicio 9

9. Dada una cadena cualquiera, y utilizando los métodos que provee la clase `String`, realizá las siguientes operaciones sobre dicha cadena, implementando métodos que funcionen de la siguiente forma:

   - `string_reverso`: retorna el string con los caracteres en orden inverso.
   - `string_sin_espacio`: elimina los espacios en blanco que contenga.
   - `string_a_arreglo_ascii`: retorna un arreglo con cada uno de los caracteres convertidos a su correspondiente valor ASCII.
   - `string_remplaza_vocal`: cambia las vocales por números:
     - `"a"` o `"A"` por `"4"`,
     - `"e"` o `"E"` por `"3"`,
     - `"i"` o `"I"` por `"1"`,
     - `"o"` u `"O"` por `"0"`,
     - y `"u"` o `"U"` por `"6"`.

`tr`: El método `tr` en Ruby realiza un reemplazo de caracteres. En este caso, estamos reemplazando todas las vocales (`aeiouAEIOU`) por los números que corresponden según tu especificación (`4310643106`).

- `"a" o "A"` → `"4"`
- `"e" o "E"` → `"3"`
- `"i" o "I"` → `"1"`
- `"o" o "O"` → `"0"`
- `"u" o "U"` → `"6"`

Ejercicio 10

10. ¿Cuál es el valor de retorno del siguiente código?

```ruby
[:upcase, :downcase, :capitalize, :swapcase].map do |meth|
  "TTPS Ruby".send(meth)
end
```

El código retorna lo siguiente :

```
["TTPS RUBY", "ttps ruby", "Ttps ruby", "ttps rUBY"]
```

El código está utilizando un arreglo de **símbolos** (`:upcase`, `:downcase`, `:capitalize`, `:swapcase`) y está aplicando cada uno de estos métodos sobre la cadena `"TTPS Ruby"`

- En el arreglo de **símbolos**, hay métodos que se pueden invocar sobre un string en Ruby.
- `map` es un método que recorre cada elemento del arreglo, en este caso, cada símbolo. Por cada simbolo del arreglo, se ejecuta el bloque de código dentro de `do...end`.
- El método `send` en Ruby invoca dinámicamente un método de un objeto.
- En este caso, `"TTPS Ruby".send(meth)` toma la cadena `"TTPS Ruby"` y le aplica el método que está representado por el símbolo `meth`.
- Por ejemplo, cuando `meth` es `:upcase`, el código ejecuta `"TTPS Ruby".upcase`.

Ejercicio 11 → Revisar

11. Tomando el ejercicio anterior como referencia, ¿en qué situaciones usarías los métodos `send` y `public_send` definidos en la clase `Object`? ¿Cuál es la principal diferencia entre esos dos métodos?

Los métodos `send` y `public_send` se utilizan en Ruby para invocar métodos de un objeto dinámicamente, lo que permite llamar métodos cuyos nombres se determinan en tiempo de ejecución

Método `send`

- El método `send` permite llamar a cualquier método de un objeto, **incluyendo métodos privados**.
- Se usa cuando necesitas invocar métodos de un objeto y no te importa si el método es público, protegido, o privado.

Método `public_send`:

- El método `public_send` solo permite invocar métodos **públicos** de un objeto. Si intentas llamar un método protegido o privado con `public_send`, lanzará un error (`NoMethodError`).
- Se usa cuando solo deseas invocar métodos públicos de manera segura y quieres respetar la encapsulación de un objeto.

Ejercicio 12

12. Escribí una función `longitud` que dado un arreglo que contenga varios `String` cualesquiera, retorne un nuevo arreglo donde cada elemento es la longitud del `String` que se encuentra en la misma posición del arreglo recibido como parámetro. Por ejemplo:

```
longitud(['TTPS', 'Opción', 'Ruby', 'Cursada 2024'])
# => [4, 6, 4, 12]
```

Ejercicio 13

13. Escribí una función llamada `listar` que reciba un `Hash` y retorne un `String` con los pares de clave/valor formateados en una lista ordenada en texto plano. Por ejemplo:

```
listar({ perros: 2, gatos: 2, peces: 0, aves: 0 })
# => "1. perros: 2\n2. gatos: 2\n3. peces: 0\n4. aves: 0"
```

- La función `each_with_index` en un hash de Ruby permite iterar sobre cada par de clave-valor, proporcionando tanto el par (clave-valor) como el índice de la iteración
- A diferencia de `each`, que solo pasa los elementos del hash, `each_with_index` también pasa un índice numérico que comienza desde 0.

```
hash.each_with_index do |(clave, valor), indice|
  # bloque de código
end
```

Ejercicio 14

14. Mejorar la función anterior en una nueva llamada `listar_mejorada` para que además reciba opcionalmente un parámetro llamado `pegamento` (su valor por defecto debe ser `": "`) que sea el que utilice para unir los pares de clave/valor. Por ejemplo:

```
listar_mejorada({ perros: 2, gatos: 2, peces: 0, aves: 0 }, " -> ")
# => "1. perros -> 2\n2. gatos -> 2\n3. peces -> 0\n4. aves -> 0"
```

Ejercicio 15

15. Escribí una función llamada `rot13` que *encripte* un `String` recibido como parámetro utilizando el algoritmo `ROT13`. Por ejemplo:

```
rot13("¡Bienvenidos a la cursada de TTPS Opción Ruby!")
# => "¡Ovrairavqbf n yn phefnqn qr GGCF Bcpvóa Ehol!"
```

- En Ruby puedes implementar el algoritmo **ROT13** fácilmente utilizando el método `tr` (translate), que permite realizar reemplazos de caracteres en una cadena
- El algoritmo **ROT13** es un cifrado simple que rota las letras del alfabeto 13 posiciones.

La cadena "N-ZA-Mn-za-m" utilizada en el método tr es un rango de caracteres que define cómo se deben reemplazar las letras del alfabeto en el cifrado ROT13

¿Cómo funciona la cadena `"N-ZA-Mn-za-m"`?

- `A-Z`: Son las letras mayúsculas de la 'A' a la 'Z'.
- `a-z`: Son las letras minúsculas de la 'a' a la 'z'.
- `N-ZA-M`: Las letras mayúsculas entre la 'N' y la 'Z' se mapean a las letras de la 'A' a la 'M', y viceversa.
- `n-za-m`: Lo mismo ocurre para las minúsculas, donde las letras entre la 'n' y la 'z' se mapean a las letras entre la 'a' y la 'm', y viceversa.

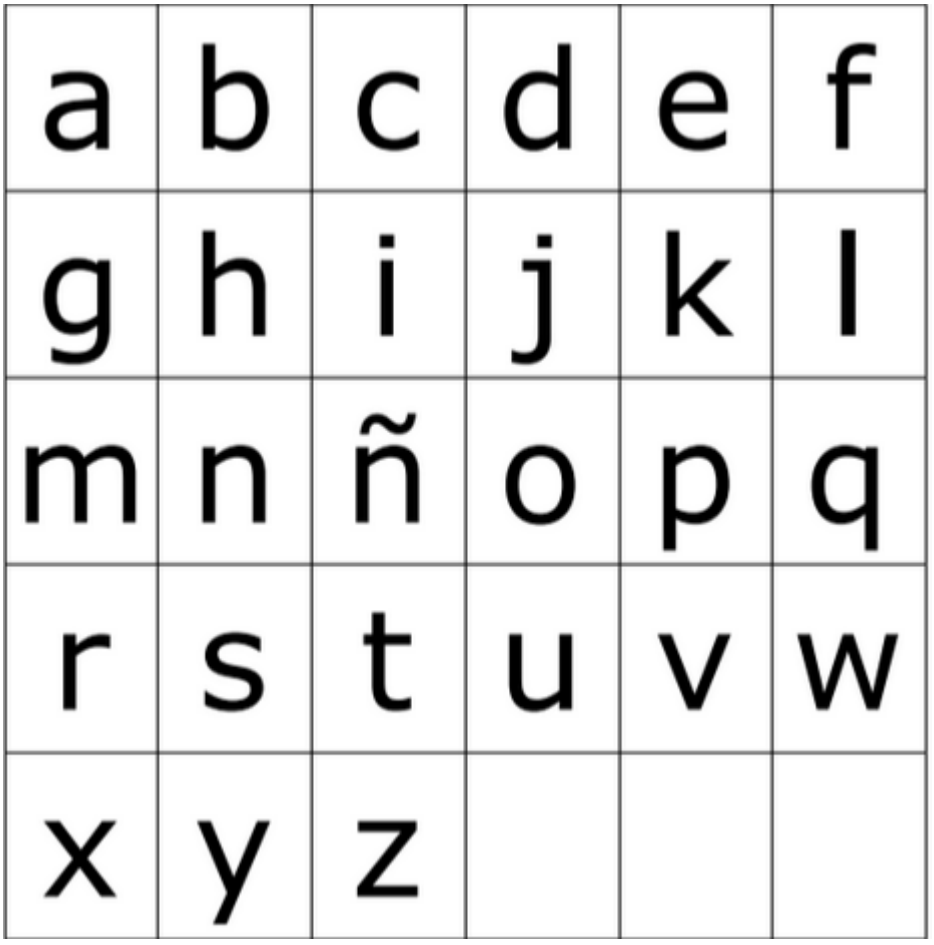
Descomposición de la cadena:

1. `N-Z`: Representa todas las letras mayúsculas entre `N` y `Z` (es decir, N, O, P, Q, R, S, T, U, V, W, X, Y, Z).
2. `A-M`: Representa todas las letras mayúsculas entre `A` y `M` (es decir, A, B, C, D, E, F, G, H, I, J, K, L, M).
3. `n-z`: Representa todas las letras minúsculas entre `n` y `z` (es decir, n, o, p, q, r, s, t, u, v, w, x, y, z).
4. `a-m`: Representa todas las letras minúsculas entre `a` y `m` (es decir, a, b, c, d, e, f, g, h, i, j, k, l, m).

Ejemplo con letra P

- Está en el rango `A-Z` (mayúsculas).
- El método `tr` mapea letras en este rango de la siguiente manera: `'A'..'M'` se mapean a `'N'..'Z'`, y `'N'..'Z'` se mapean a `'A'..'M'`.
- La letra **P** está en el rango `'A-Z'`, y se encuentra en la posición 16 (contando desde 1, es la posición 16 en el alfabeto).
- Aplicando ROT13, la **P** se convierte en la **C** (13 posiciones más adelante).

(No contar la letra Ñ)



Ejercicio 16 → Sin terminar

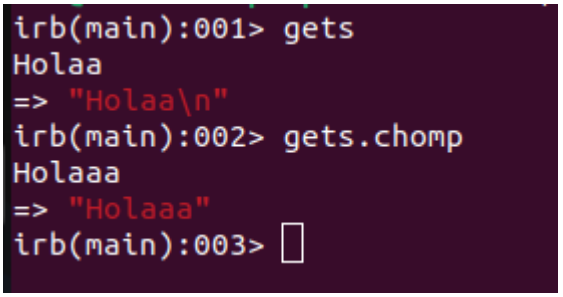
16. Escribí una función más genérica, parecida a la del ejercicio anterior, que reciba como parámetro un `String` y un número `n`, y que realice una rotación de `n` lugares de las letras del `String` y retorne el resultado. Por ejemplo:

```
rot("¡Bienvenidos a la cursada 2024 de TTPS Opción Ruby!", 13)
# => "¡Ovrairavqbf n yn phefnqn 2024 qr GGCF Bcpvóa Ehol!"
```

Ejercicio 17

17. Escribí un script en Ruby que le pida a quien lo ejecute su nombre, el cual ingresará por entrada estándar (el teclado), y que lo utilice para saludarl@ imprimiendo en pantalla `¡Hola, <nombre>!`. Por ejemplo:

```
$ ruby script.rb
Por favor, ingresá tu nombre:
R2-D2
¡Hola, R2-D2!
```



Ejercicio 18

18. Escribí un nuevo script, que de manera similar al implementado en el punto anterior haga el saludo usando un nombre que se provea, pero que en lugar de solicitar que el nombre se ingrese por entrada estándar, éste se reciba como argumento del script. Por ejemplo:

```
$ ruby script.rb C-3PO
¡Hola, C-3PO!
```

> Tip: investigá cómo se puede trabajar con los argumentos que recibió el script Ruby en su ejecución.

En Ruby, puedes hacer que un script reciba argumentos de la línea de comandos utilizando el array especial `ARGV`. Este array contiene todos los argumentos que se pasan al script cuando es ejecutado.Como puedo manejar

Para poder resolver el ejercicio, tuve que preguntar si la variable nombre era nil en lugar de preguntar si estaba vacia

---

Ejercicio 19

19. Implementá las funciones necesarias para que, dado un color expresado en notación RGB, se pueda obtener su representación en las notaciones entera y hexadecimal. La notación entera se define como `red + green * 256 + blue * 256 * 256` y la hexadecimal como el resultado de expresar en base 16 el valor de cada color y concatenarlos en orden. Por ejemplo:

```
notacion_hexadecimal([0, 128, 255])
# => '#0080FF'
notacion_entera([0, 128, 255])
# => 16744448
```

- Para llevar un numero a base 16 tuve que aplicar el método to_s con el argumento 16
- Para concatenar los elementos de un arreglo y pasarlos a un string use el metodo join

---

Ejercicio 20

20. Investigá qué métodos provee Ruby para:

    - Obtener la lista de ancestros (superclases) de una clase.
    - Conocer la lista de métodos de una clase.
    - Conocer la lista de métodos de instancia de una clase.
    - Conocer las variables de instancia de una clase.
    - Obtener el valor de una variable de instancia de un objeto (sin utilizar un método generado con `attr_reader` o similar) accediéndolo desde fuera de éste.
    - Establecer el valor de una variable de instancia de un objeto (sin utilizar un método generado con `attr_writer` o similar) desde fuera de éste.

- Obtener la lista de ancestros (superclases) de una clase :
  - Se utiliza el método `ancestors`
- Conocer la lista de métodos de una clase
  - `methods`: Lista de métodos tanto de la clase como de sus ancestros.
- Conocer la lista de métodos de instancia de una clase
  - `instance_methods`: Métodos de instancia disponibles para objetos de esa clase, incluyendo métodos heredados.
- Conocer las variables de instancia de una clase
  - Método `instance_variables`
- Obtener el valor de una variable de instancia de un objeto (sin utilizar un método generado con attr_reader o similar) accediéndolo desde fuera de éste
  - Método `instance_variable_get`
- Establecer el valor de una variable de instancia de un objeto (sin utilizar un método gene-
  rado con attr_writer o similar) desde fuera de éste
  - Método `instance_variable_set`

Ejercicio 21

21. Escribí una función que encuentre la suma de todos los números naturales múltiplos de 3 y 5 (ambos) que sean menores que un número `tope` que reciba como parámetro. Por ejemplo:

```
multiplos_de_3_y_5(100)
# => 315
```

Ejercicio 22

22. Creá otra función, similar a la anterior, que busque los números naturales múltiplos de *N* números enteros que recibirá como parámetro en un arreglo. Por ejemplo:

```
multiplos_de([3, 5], 100)
# => 315
multiplos_de([3, 5, 17, 28, 65], 100_000)
# => 92820
```

- En Ruby, puedes calcular el **mínimo común múltiplo (MCM)** entre varios números utilizando el método `lcm`, que está disponible en la clase `Integer`
- Si deseas calcular el MCM de más de dos números, puedes combinarlo usando `reduce` o `inject` para aplicarlo de manera acumulativa

```
a = 12
b = 15
minimo_comun_multiplo = a.lcm(b)
puts mcm  # Resultado: 60
```

```
numeros = [12, 15, 20]
mcm = numeros.reduce(:lcm)
puts mcm  # Resultado: 60
```

- `lcm`: Método que devuelve el mínimo común múltiplo de dos números.
- `reduce(:lcm)`: Aplica el método `lcm` sucesivamente a todos los elementos del array para calcular el MCM de todos los números.

- Si el valor de `mcm` es mayor que `tope`, el rango no incluye ningún valor, y como resultado, la suma permanece en `0`