Guía de Trabajos Prácticos N°I

1. Evalúe las siguientes expresiones en el entorno y anote los resultados, entendiendo en cada caso su funcionamiento.

5 + 6
"11"

20 factorial
"2432902008176640000"

'Esto es una prueba' size
"18"

#(1 3 5 7) at: 2
"3"

'Paradigmas' isArray
"false"

5 * 7
"35"

5 // 2
"2"

4 \\ 3
"1"

2 / 6
"(1/3)"

1.5 + 6.3e2
"631.5"

Array new

"#()"

Date today
"13 September 2022"

Time now
"4:18:36.965558 pm"

Antes de seguir con los ejercicios, es recomendable el siguiente material:

Tipos de mensajes en Smalltalk

Primero arranquemos por lo básico. Todo envío de mensaje sigue la siguiente regla: OBJETO MENSAJE PARAMETRO Lo primero que aparece es el objeto receptor, luego viene el mensaje que se le envía a dicho objeto el cual puede o no tener parámetros.

Smalltalk es bastante particular en aspectos sintácticos, por eso es importante detenerse a entender cómo se interpreta. Hay 3 tipos de mensajes en Smalltalk.

Los **mensajes unarios** son aquellos que no reciben parámetros, o sea que el receptor solito puede resolver lo pedido. Un ejemplo de esto sería la negación de un booleano:

true not → retorna false

También existen los **mensajes binarios** que son los que reciben sólo un parámetro y su nombre está compuesto por símbolos (no alfanumérico), por ejemplo la suma entre dos números es un mensaje que recibe un parámetro solo (el número a sumar), el otro número es el receptor del mensaje.

2 + 5 → retorna 7

Por último están los **mensajes de palabra clave** que pueden recibir tantos parámetros como sean necesarios. Estos mensajes se caracterizan por tener una o más partes alfanuméricas terminadas por el caracter “:” luego de los cuales se pasa cada parámetro (o sea, los parámetros van intercalados) como se muestra en los siguientes ejemplos:

5 raisedTo: 2 → es un mensaje que recibe el objeto 5 con un único parámetro, el 2, y retorna 25

3 between: 10 and: 25 → es un mensaje que recibe el objeto 3 con un 2 parámetros, el 10 es el primer parámetro y el 25 es el segundo, y retorna false

Lo que tiene de simpático los parámetros intercalados es la expresividad, pero hay que ser cuidadosos de no cometer errores. Por ejemplo si quisiéramos saber si el 3 está entre 10 y el 5 elevado al cuadrado debemos escribirlo de esta forma:

3 between: 10 and: (5 raisedTo: 2) → el segundo parámetro del mensaje between:and: es el resultado de mandarle raisedTo: a 5 con el parámetro 2, lógicamente también retorna false

Esos paréntesis son inportantes para que se interprete como queremos y no como un único envío de mensajes, donde el mensaje sería between:and:raisedTo: de 3 parámetros, que los números no entienden!

La Precedencia de Mensajes en Smalltalk se basa en estos 3 tipos de mensajes, sólo nos va a interesar esta diferenciación por ese motivo.

2. Evalúe los siguientes mensajes unarios.

```
#( 'arreglo' 'de' 'strings' ) size
 "3"

'Hoy es Jueves' asUppercase
 "'HOY ES JUEVES'"

'hola aquí estoy' reversed
 "'yotse íuqa aloh'"

#( 4 'cinco' 6 7 ) reversed
 "#(7 6 'cinco' 4)"

$A asciiValue
 "65"
```

65 asCharacter
 "$A"

'cuál es la longitud de esta oración?' size
 "36"

3. Evalúe los siguietes mensajes binarios.

'hola', ' aquí estoy'
 "'hola aquí estoy'"

#(1 2 3), #(4 5 6)
 "#(1 2 3 4 5 6)"

4 = 5
 "false"

#(1 2 $a), #($b 'cd')
 "#(1 2 $a $b 'cd')"

4. Evalúe los siguientes mesnajes de palabra clave.

'Esto es una prueba' at: 3
 "$t"

'Hola' includes: $o
 "true"

'hola' at: 1 put: $H
 "$H"

'Paradigmas de Programación' copyFrom: 4 to: 9
 "'adigma'"

#(9 8 7 6 5) at: 3

"7"

#(1 (2 3) 4 5) includes: #(2 3)
"true"

#(9 8 7 6 5) copyFrom: 1 to: 2
"#(9 8)"

Array new: 10
"#(nil nil nil nil nil nil nil nil nil nil)"

5. Mensajes anidados. Evaluarlos y de ser necesario, colocar paréntesis para mostrar de que manera ST evalúa las expresiones.

'hola' size + 4.
"8"

'ahora' size + #(1 2 3 4) size.
"9"

#(1 12 24 36) includes: 4 factorial.
"true"

3 + 4 * 2.
"14"

3 + (4 * 2).
"11"

4 factorial between: 3 + 4 and: 'hola' size * 7.
"true"

'hola' at: (#(5 3 1) at: 2).
"$l"

(6 + 9) asString.
"'15'"

Array with: 1 with: 'hola' with: (1/3).
"{1. 'hola'. (1/3)}"

6. Variables Globales, siempre empiezan con mayúscula. Evaluarlas una a una verificando los resultados.

Display.
"DisplayScreen(860x848x32)"

Transcript show: 'hola'.
“Comando que muestra por pantalla en ‘tools/Transcrypt’ la palabra hola”

Debug.
"nil"

Disk.
"nil"

Distancia.
"nil"

Distancia:= 15.
"15"

Distancia.
"15"

7. Comparando Objetos. Evaluar c/u, verificar resultados y en caso de error indicar la expresión correcta.

3 < 4.
"true"

#(1 2 3 4) = #(1 2 3 4).
"true"

'hola' <= 'adios'.
 "false"

5 = 2 + 3.
"No funciona porque primero evalúa el valor de verdad y luego intenta sumarle 3."

5 = (2 + 3).
 "true"

('hola' size <= 'adios' size).
 "true"

8. Bloques de código. (Recordar que un bloque puede tener argumentos).

[$a isVowel] value.
 "true"

[$b isVowel] value.
 "false"

[3+4. 'hola' asUppercase] value.
 "'HOLA'"

['hola' asUppercase. 3+4] value.
 "7"

| bloque |
bloque:= ['Hola ', ' como estás ?'].
^bloque value.
 "'Hola como estás ?'"

[:c | c isVowel] value: $a.
 "true"

[:c | c isVowel] value: $b.
 "false"

```
| bloque resp |
bloque:= [ :a :b | a , b ].
resp:= bloque value: 'Hola, ' value: '¿como estás ?'.
^resp.
 "'Hola, ¿como estás ?'"


| bloque |
bloque:= [ :a :b | a , b ] value: 'Hola, ' value: '¿como estás ?'.
^bloque.
 "'Hola, ¿como estás ?'"
```

9. Expresiones Booleanas.

```
5 < 2 or: [ $a isVowel ].
 "true"

5 < 2.
 "false"

(5 < 2) not.
 "true"

5 < 2 and: [ $a isVowel ].
 "false"

(5 < 2) not and: [ $a isVowel ].
 "true"

(5 < 2) not or: [ 'hola' size < 2 and: [ $a isVowel ] ].
 "true"
```

10. ¿Cómo sabemos a qué clase pertenece cada objeto? Evalúa las expresiones:

```
#(Francesca Jackie Marisa Bree) class.
 "Array"

{'Francesca'. 'Jackie'. 'Marisa Bree'} class.
```

"Array"

'Rakesh Vijay Charles Daniel Tyler' class.
"ByteString"

5 class.
"SmallInteger"

(1/2) class.
"Fraction"

5.2 class.
"SmallFloat64"

11. Mensajes en cascada, llamamos así al conjunto de mensajes que enviamos a un mismo objeto. Verifica estas dos expresiones y explica la diferencia.

3 factorial factorial.
"720"

12. ST tiene una ventana llamada “Inspector” que permite ver y cambiar las variables de instancia de un objeto. Evalúa este código y di que observas.

```
| a |
a := #( 1 2 sam 'joe' ( 4 5 ) ).
a at: 2 put: 3/4.
a inspect.
```

13. Condicionales. Verifica estas expresiones:

3 < 4 ifTrue: ['el bloque verdadero'] ifFalse: ['el bloque falso'].
"'el bloque verdadero'"

(5 > 2) ifTrue:[^'5 es MAYOR que 2'] ifFalse:[^'5 es menor que 2'].
"'5 es MAYOR que 2'"

$b isVowel ifTrue:[^'es una vocal'] ifFalse:[^'es una consonante'].

"'es una consonante'"

Guía de Trabajos Prácticos N°II

1. Crear variables temporales y referencias de estas a distintos objetos (enteros, reales, caracteres, flotantes, cadena de caracteres, etc.)

```
| entero nReal caracter flotante cadenaDeCaracteres |

entero:= 4.

nReal:= 4.56.

caracter:= $W.

flotante:= 1.02e6.

cadenaDeCaracteres:= 'Hola, ¿cómo andás?.'.
```

2. Realizar las operaciones F/G y F*G. Utilizando sumas y restas sucesivas.

```
|var1 var2 result|

var1:= UIManager default request: 'Ingrese un número.'.
(var1 = nil) ifTrue:[^nil].
var1:= var1 asInteger.

var2:= UIManager default request: 'Ingrese otro número.'.
(var2 = nil) ifTrue:[^nil].
var2:= var2 asInteger.

result:= 0.

1 to: var2 do: [ :i| result:= var1 + result. ].
```

```
^result.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

|var1 var2 result|

var1:= UIManager default request: 'Ingrese el dividendo.'.
(var1 = nil) ifTrue: [^nil].
var1:= var1 asInteger.

var2:= UIManager default request: 'Ahora, ingrese el divisor.'.
(var2 = nil) ifTrue: [^nil].
var2:= var2 asInteger.

(( var1 < var2 ) or: (var2 = 0)) ifTrue: [^nil].

result:= 0.
[ var1 >= var2 ] whileTrue: [ var1:= var1 - var2. result:= 1 + result.].

^result.
```

3. Realizar x^y.

```
"Base (-) exp. (+) --> result más pequeño"
"Base (+) exp. (-) --> result más pequeño"

|var1 var2 result|

var1:= UIManager default request: 'Ingrese la base.'.
(var1 = nil) ifTrue: [^nil].
var1:= var1 asInteger.

var2:= UIManager default request: 'Ahora, ingrese la potencia.'.
(var2 = nil) ifTrue: [^nil].
var2:= var2 asInteger.

result:= var1 raisedTo: var2.
```

```
^result.
```

~~~~~~

“Ahora, el mismo ejercicio pero utilizando multiplicaciones sucesivas.”

```
"Base (-) exp. (+) --> result más pequeño"
"Base (+) exp. (-) --> result más pequeño"

|var1 var2 result aux|

var1:= UIManager default request: 'Ingrese la base.'.
(var1 = nil) ifTrue: [^nil].
var1:= var1 asNumber.

var2:= UIManager default request: 'Ahora, ingrese la potencia.'.
(var2 = nil) ifTrue: [^nil].
var2:= var2 asNumber.

result:= 1.

(var2 < 0) ifFalse: [ 1 to: var2 do: [:i| result:= var1 * result ].]

   ifTrue: [ aux:= 1/var1. 1 to: var2*(-1) do: [:i| result:= aux * result ]].

^result.
```

4. Solicitar al usuario que ingrese los coeficientes de una ecuación cuadrática, calcular sus raíces y mostrarlas.

```
| a b c x1 x2 |

a:= UIManager default request: 'Coeficiente de grado 2: '.
(a = nil) ifTrue: [^nil].
a:= a asNumber.
```
~~~~~~

```
b:= UIManager default request: 'Coeficiente de grado 1: '.
(b = nil) ifTrue: [b:= 0].
b:= b asNumber.

c:= UIManager default request: 'Término independiente: '.
(c = nil) ifTrue: [c:= 0].
c:= c asNumber.

x1:= ( (b*(-1) ) + ( (b raisedTo: 2) - (4*a*c) ) sqrt ) / (2*a).
x2:= ( (b*(-1)) - ( (b raisedTo: 2) - (4*a*c) ) sqrt )  / (2*a).
```

5. Solicitar el ingreso de un número y verificar si este es o no primo.

“¡Recordar!: El símbolo ‘\\’ devuelve el módulo.”

```
| var result |

var:= UIManager default request: 'Ingrese un número para evaluar si es primo.'.
(var = nil) ifTrue: [^nil].
var:= var asInteger.

result:= 0.

1 to: var do: [ :i| (var\\i = 0) ifTrue: [ result:= result + 1. ] ].

(result = 2) ifTrue: [ ^'El número ingresado es primo.' ] ifFalse: [ ^'El número ingresado no es primo.' ].
```

6. Idem anterior, decir si es par o impar.

```
| var |

var:= UIManager default request: 'Ingrese un número para evaluar si es par o impar.'.
(var = nil) ifTrue: [^nil].
var:= var asInteger.
```

(var \\ 2 = 0) ifTrue: [^'El número ingresado es par.'] ifFalse: [^'El número ingresado es impar.'].

7. Dado un número determinar sus múltiplos.

“Opción 1, con un arreglo: “

```
| var múltiplos index |

var:= UIManager default request: 'Ingrese un número para obtener sus múltiplos.'.
(var = nil) ifTrue: [^nil].
var:= var asInteger.

index:= 0.
múltiplos:= Array new:(var + 1).

1 to: var do: [:i| (var \\ i = 0) ifTrue: [ múltiplos at: (index:= index + 1) put:i ] ].

^múltiplos.
```

~~~~~~

“Opción 2, utilizando un ‘OrderedCollection’: “

```
| var múltiplos |

var:= UIManager default request: 'Ingrese un número para obtener sus múltiplos.'.
(var = nil) ifTrue: [^nil].
var:= var asInteger.

múltiplos:= OrderedCollection new.

1 to: var do: [:i| (var \\ i = 0) ifTrue: [ múltiplos add: i ] ].

^múltiplos.
```
~~~~~~

8. Escribir un programa que ingrese un listado de números e informe la cantidad de múltiplos de 2, 3, 5 y 7.

```
| var lista múltiplos |

lista:= OrderedCollection new.

var:= 0.

[ var ~= nil ] whileTrue: [ var:= UIManager default request: 'Añada un número a la lista. Para salir presione "Cancel" '. (var ~= nil) ifTrue: [ lista add: var asNumber. ].].

múltiplos:= Array new: 4.

1 to: múltiplos size do: [:i| múltiplos at: i put: 0 ].

1 to: lista size do: [:i|
   ((lista at: i)  \\ 2 = 0) ifTrue: [ múltiplos at:1 put: (múltiplos at:1) + 1 ].
   ((lista at: i)  \\ 3 = 0) ifTrue: [ múltiplos at:2 put: (múltiplos at:2) + 1 ].
   ((lista at: i)  \\ 5 = 0) ifTrue: [ múltiplos at:3 put: (múltiplos at:3) + 1 ].
   ((lista at: i)  \\ 7 = 0) ifTrue: [ múltiplos at:4 put: (múltiplos at:4) + 1 ].
   ].

"^lista."
^múltiplos.
```

9. Un número entero positivo se dice perfecto si es igual a la suma de todos sus divisores, excepto el mismo. Ejemplo: los números 6 (1+2+3), 28 (1+2+4+7+14) y 496 (1+2+4+8+16+31+62+124 +248) son perfectos. Escriba un método booleano que permita diferenciar si un número (único parámetro) es perfecto.

```
| var acuml |

var:= UIManager default request: 'Ingrese un número para evaluar si es "perfecto". '.
(var = nil) ifTrue: [ ^nil ].
var:= var asNumber.
```

```
acuml:= 0.
1 to: var-1 do: [:i| ((var \\ i) = 0) ifTrue: [ acuml:= acuml + i ] ].

(var = acuml) ifTrue: [ ^'El número ', var asString, ' es perfecto.' ] ifFalse: [ ^'El número ', var asString, ' no es perfecto.' ].
```

10. Dos números se dicen amigos cuando uno de ellos es igual a la suma de todos los divisores del otro excepto el mismo. Ejemplo: los números 220 (1+2+4+5+10+11+20+22+44+55+110=284) y 284 (1+2+4+71+142=220) son amigos. Escriba un método booleano que permita discernir si dos números (parámetros) son amigos.

```
| var1 var2 div1 div2 |

var1:= UIManager default request: 'Ingrese un número: '.
(var1 = nil) ifTrue: [ ^nil ].
var1:= var1 asInteger.

var2:= UIManager default request: 'Ingrese otro número: '.
(var2 = nil) ifTrue: [ ^nil ].
var2:= var2 asInteger.

div1:= 0.
div2:= 0.

1 to: var1 do: [:i| ((var1 \\ i) = 0) ifTrue: [ div1:= div1 + i ] ].
1 to: var2 do: [:i| ((var2 \\ i) = 0) ifTrue: [ div2:= div2 + i ] ].

((div1 - var1) = var2 & (div2 - var2) = var1)
   ifTrue: [ 'Los números ', var1 asString, ' y ', var2 asString, ' son amigos.' ]
   ifFalse: ['Los números ingresados no son amigos.'].
```

~~~~~~

```
| var var2 div div2 |

var:= UIManager default request: 'Ingrese un número'.
(var = nil) ifTrue: [^nil].
```
~~~~~~

```
var:= var asNumber.

var2:= UIManager default request: 'Ingrese un número'.
(var2 = nil) ifTrue: [^nil].
var2:= var2 asNumber.

div:= 0.
div2:= 0.

1 to: (var-1) do:[:i| ((var\\i )= 0) ifTrue: [div:= div+i].].
1 to: (var2-1) do:[:i| ((var2\\i) = 0) ifTrue: [div2:= div2+i].].

((var = div2) and: (var2 = div)) ifTrue:[^true].

^false.
```

11. Ingresar 2 polinomios y realizar la suma y el producto de ambos.

~~~~~~

Antes de realizar la consigna, está bueno que veas este código para “mostrar por pantalla” diferentes valores:

```
| oc |
oc := OrderedCollection new.
oc add: 2.
oc add: #(4 9).
Transcript show: 'Show the collection:'; cr.
Transcript show: oc; cr.
Transcript show: 'Show each element:'; cr.
oc do: [ :element | Transcript show: element; cr ].
```

- *PD: No me salió como quería y me enojé.*

12. Se tiene un arreglo de n números naturales que se quiere ordenar por frecuencia, y en caso de igual frecuencia, por su valor. Por ejemplo, a partir del arreglo [1, 3, 1, 7, 2, 7, 1, 7, 3] se quiere obtener [1, 1, 1, 7, 7, 7, 3, 3, 2].
~~~~~~

- *Ni empedo lo hago.*

13. Realizar un algoritmo que lea una serie de números reales y verifique si están ordenados ascendentemente o no, informando por pantalla.

```
| input lista bandera |

lista:= OrderedCollection new.
input:= 0.

[ input ~= nil ] whileTrue: [ input:= UIManager default request: 'Ingrese un número. Presione "Cancel" o "Esc" para salir.'. (input ~= nil) ifTrue: [ lista add: input asNumber ]. ].

1 to: (lista size)-1 do: [:i| ( (lista at:i) <= (lista at:(i+1)))  ifTrue: [ ^'El arreglo está ordenado.' ] ifFalse: [ ^'El arreglo no está ordenado.' ] ].
```

~~~~~~

```
| res input |

res:= UIManager default request: 'Ingrese un número. Presione "Cancel" o "Esc" para salir.'.
(res = nil) ifTrue: [ ^self error: 'Se ha cancelado la operación antes de ingresar un dato.' ].
res:= res asNumber.

input:= 0.
[ input ~= nil ] whileTrue: [ input:= UIManager default request: 'Ingrese un número. Presione "Cancel" o "Esc" para salir.'.
   (input ~= nil) ifTrue: [ input:= input asNumber. (res <= input) ifTrue: [ res:= input ] ifFalse: [ ^false ] ]
   ].

^true.
```
~~~~~~

Guía de Trabajos Prácticos N°III

1. Convertir una cadena a mayúsculas y minúsculas.

“Opción 1, con mensajes: “

| cadena1 cadena2 |

cadena1:= 'a esta cadena la vas a estar viendo en mayúsculas, aunque se escribió en minúsculas.'.
cadena2:= 'A ESTA CADENA LA VAS A ESTAR VIENDO EN MINÚSCULAS, AUNQUE SE ESCRIBIÓ EN MAYÚSCULAS.'.

cadena1 asUppercase, ' ', cadena2 asLowercase.

^cadena1 asUppercase, String cr, cadena2 asLowercase.

~~~~~~

“Opción 2, a nivel de caracter. Forma 1: ”
| cadena1 cadena2 caracter |

cadena1:= 'a esta cadena la vas a estar viendo en mayúsculas, aunque se escribió en minúsculas.'.
cadena2:= 'A ESTA CADENA LA VAS A ESTAR VIENDO EN MINÚSCULAS, AUNQUE SE ESCRIBIÓ EN MAYÚSCULAS.'.

1 to: (cadena1 size) do: [:i| caracter:= cadena1 at:i. cadena1 at:i put: (caracter asUppercase)].
1 to: (cadena2 size) do: [:i| caracter:= cadena2 at:i. cadena2 at:i put: (caracter asLowercase)].

^cadena1, String cr, cadena2.
~~~~~~

~~~~~~

“Opción 2, a nivel de caracter. Forma 2: “

```
| cadena1 cadena2 aux1 aux2 |.
cadena1:= 'a esta cadena la vas a estar viendo en mayúsculas, aunque se escribió en minúsculas.'.
cadena2:= 'A ESTA CADENA LA VAS A ESTAR VIENDO EN MINÚSCULAS, AUNQUE SE ESCRIBIÓ EN MAYÚSCULAS.'.

aux1:= ''.
aux2:= ''.

cadena1 do: [:caracter| aux1:= aux1, caracter asUppercase asString ].
cadena2 do: [:caracter| aux2:= aux2, caracter asLowercase asString ].

^aux1, String cr, aux2.
```

~~~~~~

“Opción 2, a nivel de caracter. La mejor de todas... forma 3: “

```
| cadena1 cadena2 |

cadena1:= 'a esta cadena la vas a estar viendo en mayúsculas, aunque se escribió en minúsculas.'.
cadena2:= 'A ESTA CADENA LA VAS A ESTAR VIENDO EN MINÚSCULAS, AUNQUE SE ESCRIBIÓ EN MAYÚSCULAS.'.

1 to: (cadena1 size) do: [:i| cadena1 at:i put: ((cadena1 at:i) asUppercase) ].
1 to: (cadena2 size) do: [:i| cadena2 at:i put: ((cadena2 at:i) asLowercase) ].

^cadena1, String cr, cadena2.
```

2. Dada una cadena de entrada, devolver otra en la que los caracteres en mayúsculas hayan sido cambiados por caracteres en minúsculas y viceversa.

"Recordar: $d = ($d asLowercase). --> 'true'"

```
| cadena caracter|

cadena:= 'hoLA, ¿cÓmo ESTÁs?'.

1 to: (cadena size) do: [:i| caracter:= cadena at:i. ( (caracter) = (caracter asUppercase) ) ifTrue: [ cadena at:i put: caracter asLowercase ] ifFalse: [ cadena at:i put: caracter asUppercase ] ].

^cadena.
```

3. Verificar si una frase es un palíndromo o no.

```
"Opción 1, utilizando los mensajes to: do:"
| johnMcLane glados |

johnMcLane:= 'Lavan esa base naval'.
glados:= ''.

1 to: (johnMcLane size) do: [:i| ((johnMcLane at:i) ~= ($ )) ifTrue: [ glados:= glados, ((johnMcLane at:i) asString asLowercase) ] ].

^glados = glados reversed.
```

~~~~~~

```
"Opción 2, utilizando el mensaje do:"
| string sally |

string:= 'Lavan esa base naval'.
sally:= ''.

"Recordar!: El 'do:' agarra cada elemento del objeto.
En este caso, está agarrando cada caracter del string."

string do: [:i| ((i) ~= ($ )) ifTrue: [ sally:= sally, (i asString asLowercase) ] ].
```
~~~~~~

```
^sally = sally reversed.
```

~~~~~~

“Opción 3, ignorando los espacios, comas y puntos.”

```
| string sally |

string:= 'Lavan,,,,,esa base naval.'.
sally:= ''.

"Recordar!: El 'do:' agarra cada elemento del objeto.
En este caso, está agarrando cada caracter del string."

string do: [:i| ( (i ~= ($ )) & (i ~= ($,)) & (i ~= ($.)) ) ifTrue: [ sally:= sally, (i asString asLowercase) ] ].

^sally  = sally reversed.
```

4. Contar la cantidad de vocales de una frase.

“Opción 1, con el mensaje ‘do:’:”

```
| input cont |
input:= UIManager default request: 'Ingrese una frase para analizar la cantidad de vocales que posee: '.
(input = nil) ifTrue: [ ^nil ].

cont:= 0.
input do: [:each| (each isVowel) ifTrue: [ cont:= cont + 1 ] ].

^cont.
```

~~~~~~

“Opción 2, con los mensajes ‘to: do:’:”

```
| input cont |
```

```
input:= UIManager default request: 'Ingrese una frase para analizar la cantidad de vocales que posee: '.
(input = nil) ifTrue: [ ^nil ].

cont:= 0.
1 to: (input size) do: [:i| ((input at:i) isVowel) ifTrue: [ cont:= cont + 1 ] ].

^cont.
```

5. Ingresar dos cadenas y devolver una tercera que contenga los elementos de las dos anteriores pero intercalados.

```
“Opción 1, a nivel de string terminando la operación si una frase termina. Forma 1: ”

| frase1 frase2 sally iteración |

frase1:= 'La Universidad Tecnológica Nacional' substrings.
frase2:= 'Segundo año' substrings.

iteración:= ((frase1 size) - (frase2 size)) abs.
sally:= ''.

1 to: iteración do: [:i| sally:= sally, (frase1 at:i), ' ', (frase2 at:i), ' ' ].

^sally.
"'La Segundo Universidad año '"

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

“Opción 1, a nivel de string terminando la operación si una frase termina. Forma 2: ”
| frase1 frase2 sally men |

frase1:= 'La Universidad Tecnológica Nacional' substrings.
frase2:= 'Segundo año' substrings.

sally:= ''.
(frase1 size <= frase2 size) ifTrue: [ men:= frase1 size ] ifFalse: [ men:= frase2 size ].
```

1 to: men do: [:i| sally:= sally, (frase1 at:i), ' ', (frase2 at:i), ' '].

^sally.
"'La Segundo Universidad año '"

~~~~~~

“Opción 2, a nivel de string siguiendo la operación por más que una frase haya terminado: “

| frase1 frase2 fraseMayor fraseMenor sally |

frase1:= 'La Universidad Tecnológica Nacional' substrings.
frase2:= 'Segundo año' substrings.
sally:= ''.

(frase1 size >= frase2 size) ifTrue: [fraseMayor:= frase1. fraseMenor:= frase2] ifFalse: [fraseMayor:= frase2. fraseMenor:= frase1].

1 to: (fraseMenor size) do: [:i| sally:= sally, (fraseMayor at:i), ' ', (fraseMenor at:i), ' '].

(fraseMayor size > fraseMenor size) ifTrue: [((fraseMenor size) + 1) to: (fraseMayor size) do: [:i| sally:= sally, (fraseMayor at:i), ' ']].

^sally.
"'La Segundo Universidad año Tecnológica Nacional '"

~~~~~~

“Opción 3, a nivel de caracter terminando la operación si una frase termina: ”

| frase1 frase2 sally fraseMayor fraseMenor |

frase1:= 'La Universidad Tecnológica Nacional'.
frase2:= 'Segundo año'.
sally:= ''.

(frase1 size >= frase2 size) ifTrue: [fraseMayor:= frase1. fraseMenor:= frase2] ifFalse: [fraseMayor:= frase2. fraseMenor:= frase1].

1 to: (fraseMenor size) do: [:i| sally:= sally, ((fraseMayor at:i) asString), ((fraseMenor at:i) asString)].

^sally.
"'LSae gUunnidvoe rasñio'"

~~~~~~

“Opción 4, a nivel de caracter siguiendo la operación por más que una frase haya terminado: “

| frase1 frase2 sally fraseMayor fraseMenor |

frase1:= 'La Universidad Tecnológica Nacional'.
frase2:= 'Segundo año'.
sally:= ''.

(frase1 size >= frase2 size) ifTrue: [fraseMayor:= frase1. fraseMenor:= frase2] ifFalse: [fraseMayor:= frase2. fraseMenor:= frase1].

1 to: (fraseMenor size) do: [:i| sally:= sally, ((fraseMayor at:i) asString), ((fraseMenor at:i) asString)].

(fraseMayor size > fraseMenor size) ifTrue: [((fraseMenor size) + 1) to: (fraseMayor size) do: [:i| sally:= sally, ((fraseMayor at:i) asString)]].

^sally.
"'LSae gUunnidvoe rasñiodad Tecnológica Nacional'"

6. Dada una cadena de entrada, devolver otra en la cual las palabras estén en formato ‘Tipo Título’.

| input |
~~~~~~

input:= 'sGT pEPPERS lONELY hearts cLub band'.

input at:1 put: ((input at:1) asUppercase).

2 to: (input size) do: [:i| ((input at:(i-1)) = ($)) ifTrue: [input at:(i) put: ((input at:(i)) asUppercase)] ifFalse: [input at:i put: ((input at:i) asLowercase)]].

^input.
"'Sgt Peppers Lonely Hearts Club Band'"

7. Convertir un número en el sistema decimal al sistema binario.

- Y bueno... no se puede saber todo.

8. Dada una frase contar la cantidad de palabras en mayúsculas.

| input cont |
input:= 'sgt PEPPERS LONELY hearts CLUB band' substrings.

cont:= 0.
input do: [:each| (each = each asUppercase) ifTrue: [cont:= cont + 1]].

^cont.
"3"

9. A partir de una frase ingresada por el usuario contar la cantidad de palabras que empiezan con una determinada letra (también ingresada por el usuario).

"Probar con la frase: all your dreams are made, when you're chained to the mirror and the razorblade
Caracter a ingresar: $a
Resultado: 3.
"

| inputText inputChar count |

```
inputText:= UIManager default request: 'Ingrese una oración.'.
(inputText = nil) ifTrue: [ ^self error: 'Se ha abortado la operación.' ].
inputText:= inputText asLowercase.

inputChar:= UIManager default request: 'Ingrese una letra.'.
(inputChar = nil) ifTrue: [ ^self error: 'Se ha abortado la operación.' ].
inputChar:= (inputChar at:1) asLowercase.

count:= 0.

((inputText at:1) = inputChar) ifTrue: [ count:= count + 1 ].
2 to: (inputText size) do: [:i| ( ((inputText at:(i-1)) = ($ )) & ((inputText at:i) = inputChar) )  ifTrue: [ count:= count + 1 ] ].

^count.
```

10. Ídem anterior, pero contar la cantidad de palabras que terminan con una letra determinada.

“Ya estaba algo cansado así que lo único que hice fue agarrar el ejercicio anterior, invertir las entradas y hacer que exceptúe los puntos y las comas, además de los espacios.”

```
"Probar con la frase: all your dreams are made, when you are chained to the mirror and the razorblade.
Caracter a ingresar: $e
Resultado: 6.
"

| inputText inputChar count |

inputText:= UIManager default request: 'Ingrese una oración.'.
(inputText = nil) ifTrue: [ ^self error: 'Se ha abortado la operación.' ].
inputText:= inputText asLowercase reversed.

inputChar:= UIManager default request: 'Ingrese una letra.'.
(inputChar = nil) ifTrue: [ ^self error: 'Se ha abortado la operación.' ].
inputChar:= (inputChar at:1) asLowercase.
```

```
count:= 0.

((inputText at:1) = inputChar) ifTrue: [ count:= count + 1 ].
2 to: (inputText size) do: [:i| ( ( ((inputText at:(i-1)) = ($ )) | ((inputText at:(i-1)) = ($,)) | ((inputText at:(i-1)) = ($.)) ) & ((inputText at:i) = inputChar) )  ifTrue: [ count:= count + 1 ] ].

^count.
```

11. Dado un texto se pide:
    - La posición de la palabra más larga.
    - La longitud del texto.
    - Cuántas palabras con una longitud entre 8 y 16 caracteres poseen más de tres veces la vocal “a”.

```
| input text hippies indie comparison longPos count coflerBlock acuml |

Transcript clear open.

input:= 'Cansado de pelear con Cris, mi mente esta colgadisima como un arbol. Alamedangarada fue una desesperada.'.

hippies:= #($, $.).
text:= ''.
1 to: input size do: [:i| (hippies includes: (input at:i)) ifFalse: [ text:= text, (input at:i) asString asLowercase ] ].

indie:= 0.
comparison:= 0.
text substrings do: [:each| indie:= indie + 1. (each size > comparison) ifTrue: [ longPos:= indie. comparison:= each size ] ].

coflerBlock:= [:blockazo| (blockazo >= 8) & (blockazo <= 16) ].
indie:= 0.
count:= 0.
acuml:= 0.
```

text substrings do: [:each| (coflerBlock value: (each size)) ifTrue: [1 to: (each size) do: [:i| ((each at:i) = ($a)) ifTrue: [count:= count + 1.]]. (count > 3) ifTrue: [acuml:= acuml + 1. count:= 0]]].

Transcript show: 'Posición de la palabra más larga: ', longPos asString, String cr, 'Longitud del texto: ', input size asString, ' caracteres.', String cr, 'Palabras con una longitud entre 8 y 16 caracteres que poseen más de tres veces la vocal “a”: ', acuml asString.

12. Escribir un subprograma que dado n, lea n caracteres que forman un número romano y que devuelva un string que represente a dicho número romano y un número que represente el equivalente decimal.

- *Mmmmm...*

13. Dado un texto terminado en punto, determinar cuál es la vocal que aparece con mayor frecuencia.

"Dado un texto terminado en punto, determinar cuál es la vocal que aparece con mayor frecuencia."

```
| text coflerBlock vowels |

text:= 'Pobre Lina espera que este mundo sea mejor.'.

vowels:= Array new: 5.
1 to: vowels size do: [:i| vowels at:i put: 0 ].

coflerBlock:= [:x|
   (x = ($a)) ifTrue: [ vowels at:1 put: ((vowels at:1) + 1) ].
   (x = ($e)) ifTrue: [ vowels at:2 put: ((vowels at:2) + 1) ].
   (x = ($i)) ifTrue: [ vowels at:3 put: ((vowels at:3) + 1) ].
   (x = ($o)) ifTrue: [ vowels at:4 put: ((vowels at:4) + 1) ].
   (x = ($u)) ifTrue: [ vowels at:5 put: ((vowels at:5) + 1) ].
].

text do: [:each| coflerBlock value: each ].
```

((vowels at:1) = (vowels max)) ifTrue: [^($a) asString, ' Apariciones >> ', vowels max asString].
((vowels at:2) = (vowels max)) ifTrue: [^($e) asString, ' Apariciones >> ', vowels max asString].
((vowels at:3) = (vowels max)) ifTrue: [^($i) asString, ' Apariciones >> ', vowels max asString].
((vowels at:4) = (vowels max)) ifTrue: [^($o) asString, ' Apariciones >> ', vowels max asString].
((vowels at:5) = (vowels max)) ifTrue: [^($u) asString, ' Apariciones >> ', vowels max asString].

14. Dado un texto terminado en ‘/’ se pide determinar cuántas veces aparece determinada letra, leída de teclado.

“Opción 1, como lo haría si no me dijeran que termina con un slash: ”
"Dado un texto terminado en ‘/’ se pide determinar cuántas veces aparece determinada letra, leída de teclado.
Si se ingresa $o la respuesta es 7."

| text input count |

text:= 'Hola qué onda como estás we que cansados estamos con Agus/' asLowercase.
input:= UIManager default request: 'Ingrese un caracter.'.
(input = nil) ifTrue: [^nil].
input:= (input at:1) asLowercase.

count:= 0.
1 to: (text size) do: [:i| ((text at:i) = input) ifTrue: [count:= count + 1]].

^count.

~~~~~~

“Opción 2, teniendo en cuenta el slash: “

| text input count index |
~~~~~~

```
text:= 'Hola qué onda como estás we que cansados estamos con Agus/' asLowercase.
input:= UIManager default request: 'Ingrese un caracter.'.
(input = nil) ifTrue: [ ^nil ].
input:= (input at:1) asLowercase.

count:= 0.
index:= 1.

[ (text at:index) ~= ($/) ] whileTrue: [ ((text at:index) = input) ifTrue: [ count:= count +
1 ]. index:= index + 1 ].

^count.
```

15. Dado un texto terminado en '/' determinar cuántas veces tres palabras seguidas comienzan con la misma letra.

```
"No voy a tener en cuenta el símbolo del final."

"Antes de plantear el ejercicio, puede ser conveniente examinar el siguiente código: "

| input char |

Transcript clear open.

input:= 'Hola, esta elaboración elemental tiene confinamiento crítico.' asLowercase
substrings.
char:= $e.

input do: [:each| Transcript show: (each at:1); cr. ].

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

| input compChar threeCount acuml |

input:= 'Hola, esta elaboración elemental tiene confinamiento crítico.' asLowercase
substrings.
compChar:= $e.
```

```
threeCount:= 0.
acuml:= 0.

input do: [:each| ((each at:1) = compChar) ifTrue: [ threeCount:= threeCount + 1. (threeCount = 3) ifTrue: [ acuml:= acuml + 1. threeCount:= 0 ] ] ifFalse: [ threeCount:= 0 ] ].

^acuml.
```

16. Leer dos letras de teclado y luego un texto terminado en ‘/’. Se pide determinar la cantidad de veces que la primera letra precede a la segunda en el texto.

```
| fChar sChar text count |

text:= 'Por eso me gusta comer albóndigas de espinaca con espárragos, los cuales, por supuesto, se sirven en la mesa.'.
fChar:= $e.
sChar:= $s.

count:= 0.

2 to: (text size) do: [:i|  ( ((text at:i) = sChar) & ((text at:(i-1)) = fChar) ) ifTrue: [ count:= count + 1 ]. ].

^count.
```