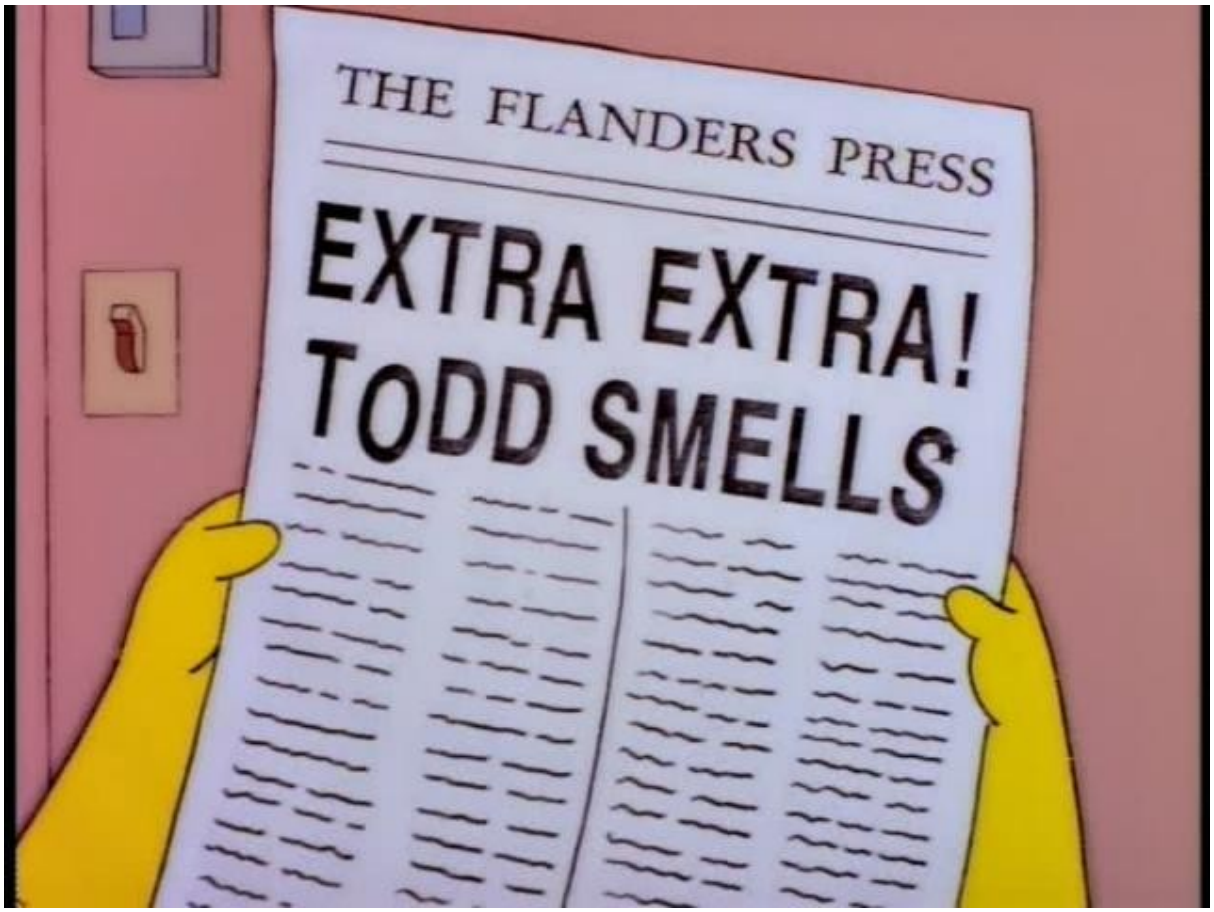

EXTRA, EXTRA!

Ejercicios que no están en la guía y otras cosas que, ponele, son interesantes.

"Convertir una cadena a mayúsculas, a nivel de caracter. Opción 1."

| toyStory wendyMalibú stacyMalibú |

toyStory:= 'Hay una serpiente en mi bota.'.
stacyMalibú:= ''.

1 to: toyStory size do: [:i| wendyMalibú:= toyStory at:i. stacyMalibú:= stacyMalibú, wendyMalibú asString asUppercase].

^stacyMalibú.

~~~~~~

"Opción 2."

| toyStory stacyMalibú |

toyStory:= 'Hay una serpiente en mi bota.'.
stacyMalibú:= ''.

1 to: toyStory size do: [:i| stacyMalibú:= stacyMalibú, ((toyStory at:i) asString asUppercase)].

^stacyMalibú.

~~~~~~

“Opción 3.”
| cadena caracter |

cadena:= 'a esta cadena la vas a estar viendo en mayúsculas, aunque se escribió en minúsculas.'.

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

^cadena.

~~~~~~

“Opción 4.”

| cadena |

cadena:= 'a esta cadena la vas a estar viendo en mayúsculas, aunque se escribió en minúsculas.'.

1 to: (cadena size) do: [:i| cadena at:i put: ((cadena at:i) asUppercase)].

^cadena.

~~~~~~

Hacela bien y aprendé Pharo Smalltalk con Stef, papá!!

https://amber-lang.net/learn.html

~~~~~~

Usá el Transcript como un máster:

Transcript clear. Transcript open.

(3 > 10) ifTrue: [Transcript show: 'maybe there''s a bug ....'] ifFalse: [Transcript show: 'No : 3 is less than 10'].

~~~~~~

"Loops are high-level collection iterators, implemented as regular methods."

"Basic loops:
to:do:

to:by:do"

Transcript clear. Transcript open.
1 to: 100 do: [:i | Transcript show: i asString; cr].

1 to: 100 by: 3 do: [:i | Transcript show: i asString; cr].

100 to: 0 by: -2 do: [:i | Transcript show: i asString; cr].

~~~~~~

Mensajes que están buenísimos:

#(11 38 3 -2 10) collect: [:each | each negated].
"#(-11 -38 -3 2 -10)"

#(11 38 3 -2 10) collect: [:each | each odd].
"#(true false true false false)"

#(11 38 3 -2 10) select: [:each | each odd].
"#(11 3)"

#(11 38 3 -2 10) collect: [:each | each > 10].
"#(true true false false false)"

#(11 38 3 -2 10) select: [:each | each > 10].
"#(11 38)"

#(11 38 3 -2 10) reject: [:each | each > 10].
"#(3 -2 10)"

~~~~~~

Ejemplo pedorro con el mensaje 'include:' :

| b m w |

```
b:= 'h,o.rse'.

m:= #($, $.).

w:= ((b at:2) = ($,)).

w:= m includes: (b at:2).

^(w not).
```

~~~~~~

Ejemplo del uso de bloques y Transcrip para mostrar mensajes de forma muy cool:

```
| varAfueraDelBlock1 varAfueraDelBlock2 coflerBlockSuma coflerBlockProd resultSuma resultProd |

Transcript clear open.

coflerBlockSuma:= [:x :y| x + y ].
coflerBlockProd:= [:a :b| a * b ].

varAfueraDelBlock1:= 4. "Es una valor definido fuera del bloque que utilizaremos."
varAfueraDelBlock2:= 8. "Ídem a la línea anterior."

resultSuma:= coflerBlockSuma value: varAfueraDelBlock1 value: varAfueraDelBlock2.

resultProd:= coflerBlockProd value: varAfueraDelBlock1 value: varAfueraDelBlock2.

^Transcript show: resultSuma asString, String cr, resultProd asString.
```

~~~~~~

Ejemplos sobre el uso de los bucles.

“Esto no funciona en Pharo 11 ya que en esa versión no se puede modificar el string sobre sí mismo.”
| toyStory toyStory2 toyStory3 toyStory4 indie holanda |

Transcript clear open.

toyStory:= 'Hay una serpiente en mi bota.'.
toyStory2:= 'Hay una serpiente en mi bota.'.
toyStory3:= 'Hay una serpiente en mi bota'.
toyStory4:= 'Hay una serpiente en mi bota.' substrings.
holanda:= 'Hay una serpiente en mi bota.'.

"Este código toma cada caracter del string y lo reemplaza por sí mismo pero en mayúscula."
1 to: toyStory size do: [:i| toyStory at:i put: (toyStory at:i) asUppercase].

Transcript show: 'Esto es toyStory después de ejecutar el bucle >> ', toyStory, String cr.

"Este código también toma cada caracter del string toyStory2. Lo malo es que no tenemos un índice para trabajar, por lo que tenemos que definir una cosa más."
indie:= 0.
toyStory2 do: [:each| indie:= indie + 1. toyStory2 at: indie put: each asUppercase].

Transcript show: 'Esto es toyStory2 después de ejecutar el bucle >> ', toyStory2, String cr.

toyStory3 substrings do: [:each| Transcript show: each, String cr.].

Transcript show: 'Esto es toyStory3 después de ejecutar el bucle >> ', toyStory3, String cr.

toyStory4 do: [:each| Transcript show: each, String cr.].

Transcript show: 'Esto es toyStory4 después de ejecutar el bucle >> ', toyStory4 asString, String cr.

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

Cálculo de PI con la fórmula de Leibniz:

“Opción Agustinosa: “

```
|pi|

pi:=0.
0 to: 99 do: [:n| pi:=(pi + (((-1) ** n)/((2 * n) + 1)))].
pi
```

~~~~~~

“Opción Juanchéscoli: “

```
| pi numerador denominador |

numerador:= [:n| (1 negated) raisedTo: n ].
denominador:= [:n| (2 * n) + 1 ].

pi:= 0.
0 to: 99 do: [:i| pi:= pi + ( (numerador value: i) / (denominador value: i) ) ].

^pi.
```

~~~~~~

Contar las palabras que empiezen con prefijo algún dado:

```
| texto prefijo str cont |

texto:= 'Quizás tenga flores en su ombligo... y además, en sus dedos que se vuelven
pan.' asLowercase.
prefijo:= 'ten'.

cont:= 0.
```

```
texto substrings do: [:each| str:= ''. (prefijo size <= each size) ifTrue: [ 1 to: (prefijo size) do: [:i| str:= str, (each at:i) asString ]. (prefijo = str) ifTrue: [ cont:= cont + 1 ] ] ].

^cont.
 "1"
```

~~~~~~

1° Parcial Práctico.

1. Realizar la división entre dos números enteros utilizando restas hasta el primer dígito decimal, si es necesario. Tener en cuenta que tanto el dividendo como el divisor pueder ser positivos o negativos. Ingresar la información a través del UIManager.
   Devolver un string con el formato ‘eee, d’, siendo ‘e’ la parte entera y ‘d’ los decimales.

```
| n1 n2 aux1 aux2 intResult decResult |

n1:= UIManager default request: 'Ingrese el numerador: '.
(n1 = nil) ifTrue: [ ^nil ].
n1:= n1 asNumber.

n2:= UIManager default request: 'Ingrese el denominador: '.
(n2 = nil) ifTrue: [ ^nil ].
n2:= n2 asNumber.

aux1:= n1 abs.
aux2:= n2 abs.
intResult:= 0.

[ aux1 >= aux2 ] whileTrue: [ intResult:= intResult + 1. aux1:= (aux1 - aux2) ].

(((n1 >= 0) & (n2 < 0)) | ((n1 <= 0) & (n2 > 0))) ifTrue: [ intResult:= intResult negated ].

decResult:= 0.
```
~~~~~~

(aux1 ~= 0) ifTrue: [aux1:= aux1 * 10. [aux1 >= aux2] whileTrue: [decResult:= decResult + 1. aux1:= (aux1 - aux2)]].

^intResult asString, ',', decResult asString.

2. A partir de un texto de entrada arbitrario realizar el cifrado César, consistente en desplazar una determinada cantidad de caracteres (también arbitrarios). Ingresar el texto y el desplazamiento a través del UIManager.
   Por ejemplo, con un desplazamiento de 3 la letra ‘A’ sería sustituída por la ‘D’.
   Aclaración: El profesor indicó que si ocurre un desbordamiento, por ejemplo, que se tenga que cifrar la letra ‘Z’ con un desplazamiento de 3, no se haga nada. Por lo que no es necesario en el parcial que sea “cíclico” el cifrado.

```
“Opción 1, lo que me salió en el examen: “
| texto abc des x |

texto:= UIManager default request: 'Ingrese el texto que desee cifrar: '.
(texto = nil) ifTrue: [ ^nil ].
texto:= texto asLowercase.

des:= UIManager default request: 'Ingrese el desplazamiento para el cifrado: '.
(des = nil) ifTrue: [ ^nil ].
des:= des asInteger.

abc:= #($a $b $c $d $e $f $g $h $i $j $k $l $m $n $ñ $o $p $q $r $s $t $u $v $w $x $y
$z).

1 to: texto size do: [:i|
   x:= 1.
   [ ((texto at:i) ~= (abc at:x)) & ((x+des) < (abc size)) ] whileTrue: [ x:= x + 1 ].
   ((texto at:i) = (abc at:x)) ifTrue: [ texto at:i put: (abc at: (x + des)) ]
   ].

^texto.
```

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

“La opción 1 puede no ser la mejor. Analizar el siguiente código antes de revisar la opción 2: “

```
| var |

Transcript clear open.

var:= $A.

Transcript show: var asciiValue asString, String cr, (var asciiValue + 1) asString, String cr, ((var asciiValue) asCharacter) asString, String cr, ((var asciiValue + 1) asCharacter) asString.
```

~~~~~~

“Opción 2: “

```
| texto des |

texto:= UIManager default request: 'Ingrese el texto que desee cifrar: '.
(texto = nil) ifTrue: [ ^nil ].
texto:= texto asLowercase.

des:= UIManager default request: 'Ingrese el desplazamiento con el que desee realizar el cifrado: '.
(des = nil) ifTrue: [ ^nil ].
des:= des asInteger.

1 to: (texto size) do: [:i| texto at: i put: (((texto at: i) asciiValue + des) asCharacter) ].

^texto.
```

~~~~~~