Cátedra 18 -- PRÁCTICA PARA LA SOLEMNE

Problema 1 -- No es tipo solemne

Conteste en el espacio indicado a qué término se refiere cada definición.

Operación de *pegar* dos strings: _____

Tipo de valor al que corresponden **True** y **False**: _____

Sangría desde el lado izquierdo de las líneas: _____

La instrucción **elif** es la contracción de: _____

Tipo de valor al que corresponden **0**, **1**, **2**, etc: _____

En castellano decimos *subcadena*, en inglés es: _____

La instrucción **while** se lee en castellano como: _____

Método para contar subcadenas dentro de un string: _____

Método que cuenta cuántos caracteres tiene un string: _____

Método que verifica si un string consiste sólo de dígitos: _____

Problema 2 -- Tipo solemne

Escriba un programa que reciba una cantidad indefinida de números y que los multiplique. El programa deberá dejar de recibir inputs una vez que reciba el input **FIN**.

Caso 1. Inputs:	**Caso 2. Inputs**:	**Caso 3. Inputs**:	**Caso 4. Inputs**:
10 50 FIN **Output**: 500	7 FIN **Output**: 7	25 5 5 FIN **Output**: 625	11 11 FIN **Output**: 121

Problema 3 -- No es tipo solemne

```
x = input()
num_previo = None
while x != 'fin':
      este_num = float(x)
      if num_previo == None:
            num_previo = este_num
      media = (este_num + num_previo)/2
      print( media )
      num_previo = este_num
      x = input()
```

A continuación escriba qué imprime este programa ante los inputs **7**, **1**, **4**, **4**, **2**, **fin**.

_____

_____

_____

_____

_____

_____

Problema 4 -- Tipo solemne

Escriba un programa que reciba un string. Este string consistirá en números separados por espacios. Su programa deberá imprimir cada número en líneas separadas.

Caso 1. Input: 123 24 55 **Output**: 123 24 55	**Caso 2. Input**: 15 **Output**: 15	**Caso 3. Input**: 1 2 3 4 5 **Output**: 1 2 3 4 5	**Caso 4. Input**: 30512 9652 391904 **Output**: 30512 9652 391904

RESPUESTAS

Aquí van las soluciones con comentarios.

Problema 1 -- No es tipo solemne
Conteste en el espacio indicado a qué término se refiere cada definición.

Operación de *pegar* dos strings: CONCATENACIÓN

Concatenar y *concatenación* son palabras que hemos usado recurrentemente en el curso. Como recordaremos, concatenar significa *conectar dos cadenas*, ya que, etimológicamente, *catena* quiere decir cadena. De hecho, castellano, concatenar es sinónimo de encadenar.

Tipo de valor al que corresponden **True** y **False**: bool (o booleano)

Llamamos *booleanos* a los valores de verdad, en honor al matemático-logicista George Boole.

Sangría desde el lado izquierdo de las líneas: INDENTACIÓN

El término *indentación* es de uso típico del mundo editorial. Se refiere a la distancia entre el texto y el lado izquierdo de la hoja.

La instrucción **elif** es la contracción de: else + if

Tipo de valor al que corresponden **0**, **1**, **2**, etc: int (entero)

En Python, el nombre exacto es **int**, que es abreviatura de integer o integral, que significa entero.

En castellano decimos *subcadena*, en inglés es: SUBSTRING

Recordemos que *string* se traduce como *cadena* en castellano, cuando hablamos de computación.

La instrucción **while** se lee en castellano como: MIENTRAS

Método para contar subcadenas dentro de un string: count

Este método se usa así: para contar las ocurrencias del string **x** dentro del string **y**, escribimos: **y.count(x)**. Aquí, **x** e **y** pueden ser variables o directamente los strings. Por ejemplo, podemos escribir **"Solemne 1".count("e")** para recibir el valor 2.

Método que cuenta cuántos caracteres tiene un string: len

La función **len** es esencial para contar el largo de *cosas* en Python. Así, **len("Solemne 1")** nos devuelve **9** (7 letras en Solemne, 1 dígito, 1 espacio).

Método que verifica si un string consiste sólo de dígitos: isdigit

Por ejemplo, **"1234".isdigit()** nos devuelve True, mientras que **"Solemne".isdigit()** nos devuelve False.

Problema 2 -- Tipo solemne

Escriba un programa que reciba una cantidad indefinida de números y que los multiplique. El programa deberá dejar de recibir inputs una vez que reciba el input **FIN**.

Caso 1. Inputs:	**Caso 2. Inputs**:	**Caso 3. Inputs**:	**Caso 4. Inputs**:
10 50 FIN **Output**: 500	7 FIN **Output**: 7	25 5 5 FIN **Output**: 625	11 11 FIN **Output**: 121

Solución

Vayamos escribiendo el código paso a paso. El código recién agregado irá resaltado.

¿Qué queremos? Calcular una multiplicación de *varios* números. Por lo tanto, **debemos crear una variable para calcular (construir) nuestra respuesta**, y esta variable **debe iniciar valiendo 1**.

```
r = 1
```

Ojo, si esta variable iniciara con el valor 0, nos arruinaría todo, pues cualquier cosa multiplicada por cero nos da cero. Esto no ocurre si iniciamos con el valor 1.

Como recibimos una cantidad **indefinida** de números (que son **inputs**), entonces necesitamos un **while**. Y vamos a repetir ***mientras*** no recibamos el string **"FIN"**. Actualizamos el código:

```
r = 1
while x != "FIN":
```

Hemos usado **x** en la *guarda* del **while**, ¡pero no hemos definido esta variable! Como se refiere a los **inputs**, corregimos:

```
r = 1
x = input()
while x != "FIN":
```

Ya, veamos que el **while** es básicamente un **if**, pero que se vuelve a evaluar hasta que la guarda se hace False. Por eso, si **x! = "FIN"**, entonces **x debe ser un número entero** (per enunciado). Convertimos **x** en entero y lo guardamos en la variable **n**:

```
r = 1
x = input()
while x != "FIN":
    n = int(x)
```

Pero ese número entero **n** (tipo **int**) no nos sirve ahí. Nosotros queremos actualizar **r**, que es nuestra variable de respuesta; debemos multiplicarla por **n**. Por ende:

```
r = 1
x = input()
while x != "FIN":
     n = int(x)
     r *= n
```

Ahora vemos que no estamos pidiendo **input**s otra vez. Sólo hemos pedido un **input** al principio. Para volver a pedir un **input**, lo agregamos al final del bloque indentado del **while**. Como usamos para variable **x** para los **input**s:

```
r = 1
x = input()
while x != "FIN":
     n = int(x)
     r *= n
     x = input()
```

Presto! Ahora nos queda presentar la respuesta que hemos construido. Esto lo hacemos al final de nuestro código:

```
r = 1
x = input()
while x != "FIN":
     n = int(x)
     r *= n
     x = input()
print(r)
```

En resumen, el código queda como:

```
r = 1
x = input()
while x != "FIN":
     n = int(x)
     r *= n
     x = input()
print(r)
```

Este código ha sido completado.

Problema 3 -- No es tipo solemne

```
x = input()
num_previo = None
while x != 'fin':
      este_num = float(x)
      if num_previo == None:
            num_previo = este_num
      media = (este_num + num_previo)/2
      print( media )
      num_previo = este_num
      x = input()
```

A continuación escriba qué imprime este programa ante los inputs **7**, **1**, **4**, **4**, **2**, **fin**.

_____

_____

_____

_____

_____

_____

Solución

Es **muy importante ser capaces de interpretar código por nuestra propia cuenta**. Esto quiere decir que debemos ser capaces de ejecutar el código línea por línea y tener un registro de cómo se actualizan sus variables.

La solución en este caso es:

7.0
4.0
2.5
4.0
3.0

No hay más outputs (**print**s) ejecutados por el programa.

Siendo muy detallados, la siguiente tabla reproduce, línea por línea, qué ocurre con el programa.

Instrucción	Descripción	**x**	**num_previo**	**este_num**	**media**
`x = input()`	recibe '7' y lo guarda en x	**'7'**	--	--	--
`num_previo = None`	define num_previo con el valor None	'7'	**None**	--	--
`while x != 'fin':`	mientras x no es 'fin' (así es)	'7'	None	--	--
	(inicia el bloque indentado)	'7'	None	--	--
`este_num = float(x)`	convierte x en flotante	'7'	None	**7.0**	--
`if num_previo == None:`	si num_previo es None, y lo es	'7'	None	7.0	--
`num_previo = este_num`	asigna a num_previo el valor de este_num	'7'	**7.0**	7.0	--
`media = (este_num + num_previo)/2`	asigna a media el valor 7.0	'7'	7.0	7.0	**7.0**
`print( media )`	imprime: 7.0	'7'	7.0	7.0	7.0
`num_previo = este_num`	asigna a num_previo el valor 7.0	'7'	**7.0**	7.0	7.0
`x = input()`	recibe el valor '1' y lo guarda en x	**'1'**	7.0	7.0	7.0
	(termina el bloque indentado)	'1'	7.0	7.0	7.0
`while x != 'fin':`	mientras x no es 'fin' (así es)	'1'	7.0	7.0	7.0

Instrucción	Descripción	**x**	**num_previo**	**este_num**	**media**
	(inicia el bloque indentado)	'1'	7.0	7.0	7.0
este_num = float(x)	convierte x en flotante	'1'	7.0	**1.0**	7.0
if num_previo == None:	si num_previo es None, pero no lo es	'1'	7.0	1.0	7.0
media = (este_num + num_previo)/2	asigna a media el valor 4.0	'1'	7.0	1.0	**4.0**
print(media)	imprime: 4.0	'1'	7.0	1.0	4.0
num_previo = este_num	asigna a num_previo el valor 1.0	'1'	**1.0**	1.0	4.0
x = input()	recibe el valor '4' y lo guarda en x	**'4'**	1.0	1.0	4.0
	(termina el bloque indentado)	'4'	1.0	1.0	4.0
while x != 'fin':	mientras x no es 'fin' (así es)	'4'	1.0	1.0	4.0
	(inicia el bloque indentado)	'4'	1.0	1.0	4.0
este_num = float(x)	convierte x en flotante	'4'	1.0	**4.0**	4.0
if num_previo == None:	si num_previo es None, pero no lo es	'4'	1.0	4.0	4.0
media = (este_num + num_previo)/2	asigna a media el valor 2.5	'4'	1.0	4.0	**2.5**
print(media)	imprime: 2.5	'4'	1.0	4.0	2.5
num_previo = este_num	asigna a num_previo el valor 4.0	'4'	**4.0**	4.0	2.5
x = input()	recibe el valor '4' y lo guarda en x	**'4'**	4.0	4.0	2.5
	(termina el bloque indentado)	'4'	4.0	4.0	2.5
while x != 'fin':	mientras x no es 'fin' (así es)	'4'	4.0	4.0	2.5
	(inicia el bloque indentado)	'4'	4.0	4.0	2.5
este_num = float(x)	convierte x en flotante	'4'	4.0	**4.0**	2.5
if num_previo == None:	si num_previo es None, pero no lo es	'4'	4.0	4.0	2.5
media = (este_num + num_previo)/2	asigna a media el valor 4.0	'4'	4.0	4.0	**4.0**
print(media)	imprime: 4.0	'4'	4.0	4.0	4.0
num_previo = este_num	asigna a num_previo el valor 4.0	'4'	**4.0**	4.0	4.0
x = input()	recibe el valor '2' y lo guarda en x	**'2'**	4.0	4.0	4.0
	(termina el bloque indentado)	'2'	4.0	4.0	4.0
while x != 'fin':	mientras x no es 'fin' (así es)	'2'	4.0	4.0	4.0
	(inicia el bloque indentado)	'2'	4.0	4.0	4.0
este_num = float(x)	convierte x en flotante	'2'	4.0	**2.0**	4.0
if num_previo == None:	si num_previo es None, pero no lo es	'2'	4.0	2.0	4.0
media = (este_num + num_previo)/2	asigna a media el valor 3.0	'2'	4.0	2.0	**3.0**
print(media)	imprime: 3.0	'2'	4.0	2.0	3.0
num_previo = este_num	asigna a num_previo el valor 2.0	'2'	**2.0**	2.0	3.0
x = input()	recibe el valor 'fin' y lo guarda en x	**'fin'**	2.0	2.0	3.0
	(termina el bloque indentado)	'fin'	2.0	2.0	3.0
while x != 'fin':	mientras x no es 'fin', pero es 'fin'	'fin'	2.0	2.0	3.0
	(termina el programa)	'fin'	2.0	2.0	3.0

Problema 4 -- Tipo solemne

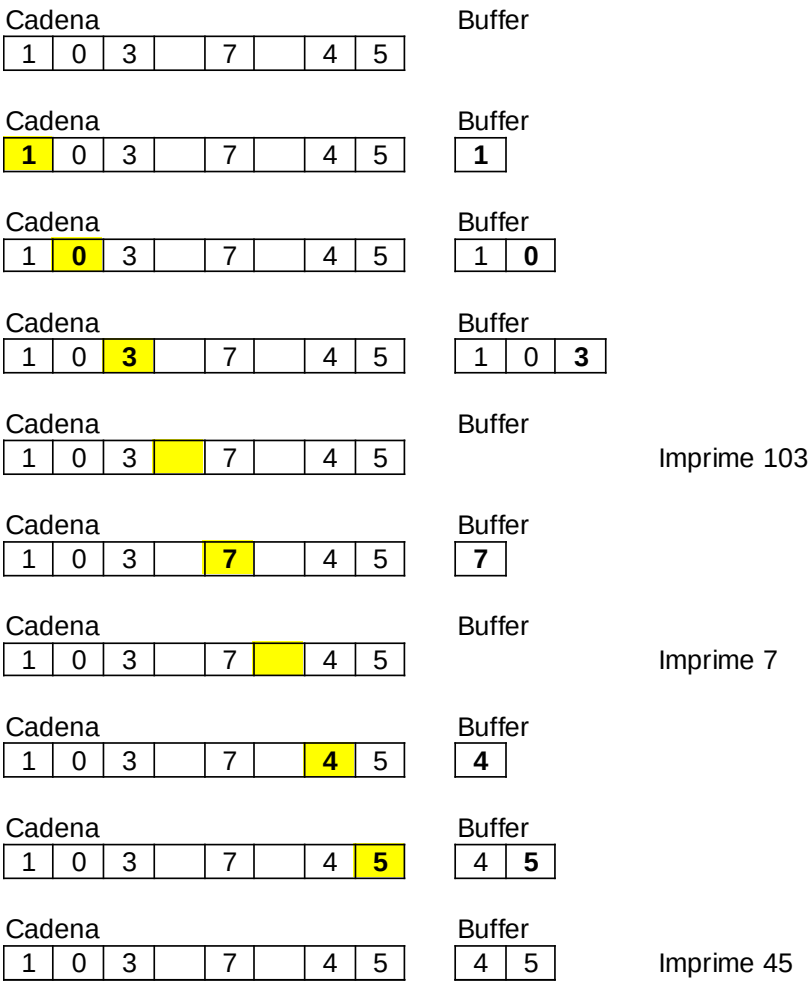
Escriba un programa que reciba un string. Este string consistirá en números separados por espacios. Su programa deberá imprimir cada número en líneas separadas.

Caso 1. Input: 123 24 55 **Output**: 123 24 55	**Caso 2. Input**: 15 **Output**: 15	**Caso 3. Input**: 1 2 3 4 5 **Output**: 1 2 3 4 5	**Caso 4. Input**: 30512 9652 391904 **Output**: 30512 9652 391904

Solución

Algoritmo. Vamos a recorrer el string recibido de izquierda a derecha, como si moviésemos un cabezal sobre una cinta. Vamos a hacer esto: si estamos ante un dígito, agregamos este dígito a nuestro *buffer* (necesitaremos una variable para esto); si no es dígito, imprimiremos el buffer en pantalla (asumimos que contiene dígitos) y lo vaciamos.

La figura de abajo muestra esto para el input `103 7 45`; el cabezal se marca en amarillo.



Como nuestro algoritmo revisa cada *caracter* del string de entrada, yendo de izquierda a derecha, usaremos **for**. La variable que usaremos para construir la respuesta, *buffer*, la inicializaremos con el string vacío. Comenzamos escribiendo:

```
entrada = input()
buffer = ''
for c in entrada:
```

En el bloque indentado del **for**, escribimos: si **c** es un dígito, lo agregamos al buffer. O sea:

```
entrada = input()
buffer = ''
for c in entrada:
     if c.isdigit():
          buffer = buffer + c
```

Pero si **c** no es un dígito (en este caso, sería un espacio), entonces imprimimos el buffer y luego lo vaciamos:

```
entrada = input()
buffer = ''
for c in entrada:
     if c.isdigit():
          buffer = buffer + c
     else:
          print(buffer)
          buffer = ''
```

Finalmente, quedará algo en el buffer al terminar el **for**. Por eso, volvemos a imprimir el buffer luego de concluída la repetición:

```
entrada = input()
buffer = ''
for c in entrada:
     if c.isdigit():
          buffer = buffer + c
     else:
          print(buffer)
          buffer = ''
print(buffer)
```

Ese es nuestro código ya terminado.