

BIG DATA Y PYTHON

MÁSTER EN BIG DATA

201020

GABRIEL MARÍN DÍAZ

hola

Presentación

Yo mismo

Nombre: Gabriel Marín Díaz

A qué me dedico...

- Channel Enablement Manager en Sage
- Profesor Asociado UCM

Perfil de LinkedIn: <https://www.linkedin.com/in/gabrielmarindiaz/>

CONTENIDO

Contenido

Resumen

Tema 1 – Visión General

Tema 2 – Introducción a SQL

Tema 3 – Introducción al Lenguaje Python

Tema 4 – HTML y Python

Tema 5 – Big Data y Python

Tema 6 – Procesamiento Distribuido (Spark)

Prácticas las realizaremos con Python, MongoDB, Apache Spark

Arrancamos?

ÍNDICE

Índice

Contexto

- Trabajando con Datos
- Estructuras de Control
- Funciones
- Ejercicios

CONTEXTO

Arquitectura en capas



1. Capa de Presentación: Interacción entre el usuario y el software. Puede ser tan simple como un menú basado en líneas de comando o tan complejo como una aplicación basada en formas. Su principal función es mostrar información al usuario, interpretar los comandos de este y realizar algunas validaciones simples de los datos ingresados.

2. Capa de Reglas de Negocio (Empresarial): También denominada Lógica de Dominio, esta capa contiene la funcionalidad que implementa la aplicación. Involucra cálculos basados en la información dada por el usuario, datos almacenados y validaciones. Controla la ejecución de la capa de acceso a datos y servicios externos.

3. Capa de Datos: Esta capa contiene la lógica de comunicación con otros sistemas que llevan a cabo tareas por la aplicación. Para el caso de aplicaciones empresariales, está representado por una base de datos, que es responsable del almacenamiento persistente de información. Esta capa debe abstraer completamente a las capas superiores (negocio) del dialecto utilizado para comunicarse con los repositorios de datos (PL/SQL, Transact-SQL, etc.).

Índice

- Contexto
- Trabajando con Datos
- Estructuras de Control
- Funciones
- Ejercicios

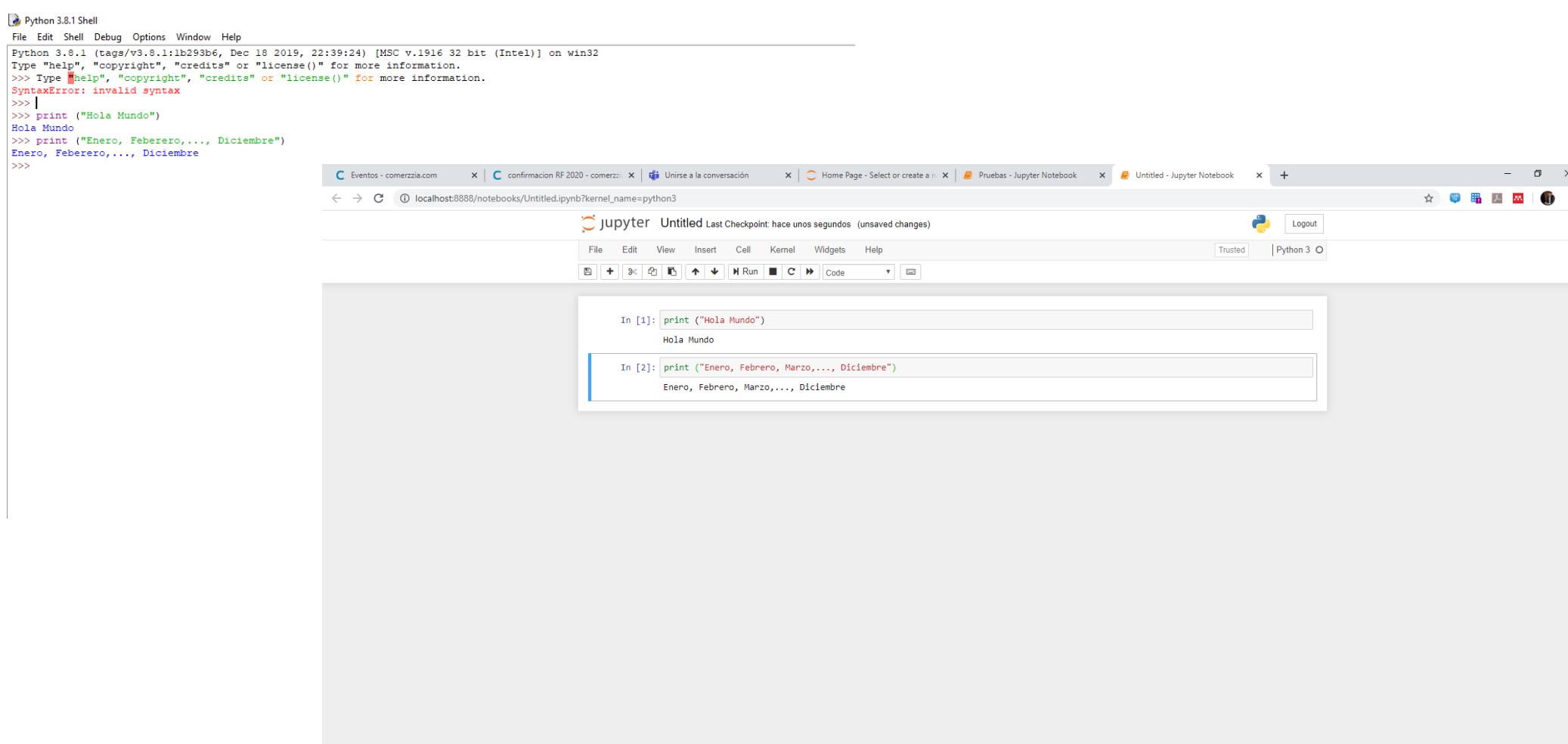
TRABAJANDO CON DATOS EN PYTHON

Comenzando con Python

**ARRANCAMOS EL EDITOR IDLE O
BIEN JUPYTER Y EMPEZAMOS CON
NUESTRO PRIMER PROGRAMA**

Comenzando con Python

COMENZANDO A PROGRAMAR



The screenshot shows a web browser window with multiple tabs open. The active tab is a Jupyter Notebook titled "Untitled" with the URL localhost:8888/notebooks/Untitled.ipynb?kernel_name=python3. The notebook contains two code cells:

```
In [1]: print ("Hola Mundo")
Hola Mundo

In [2]: print ("Enero, Febrero, Marzo,..., Diciembre")
Enero, Febrero, Marzo,..., Diciembre
```

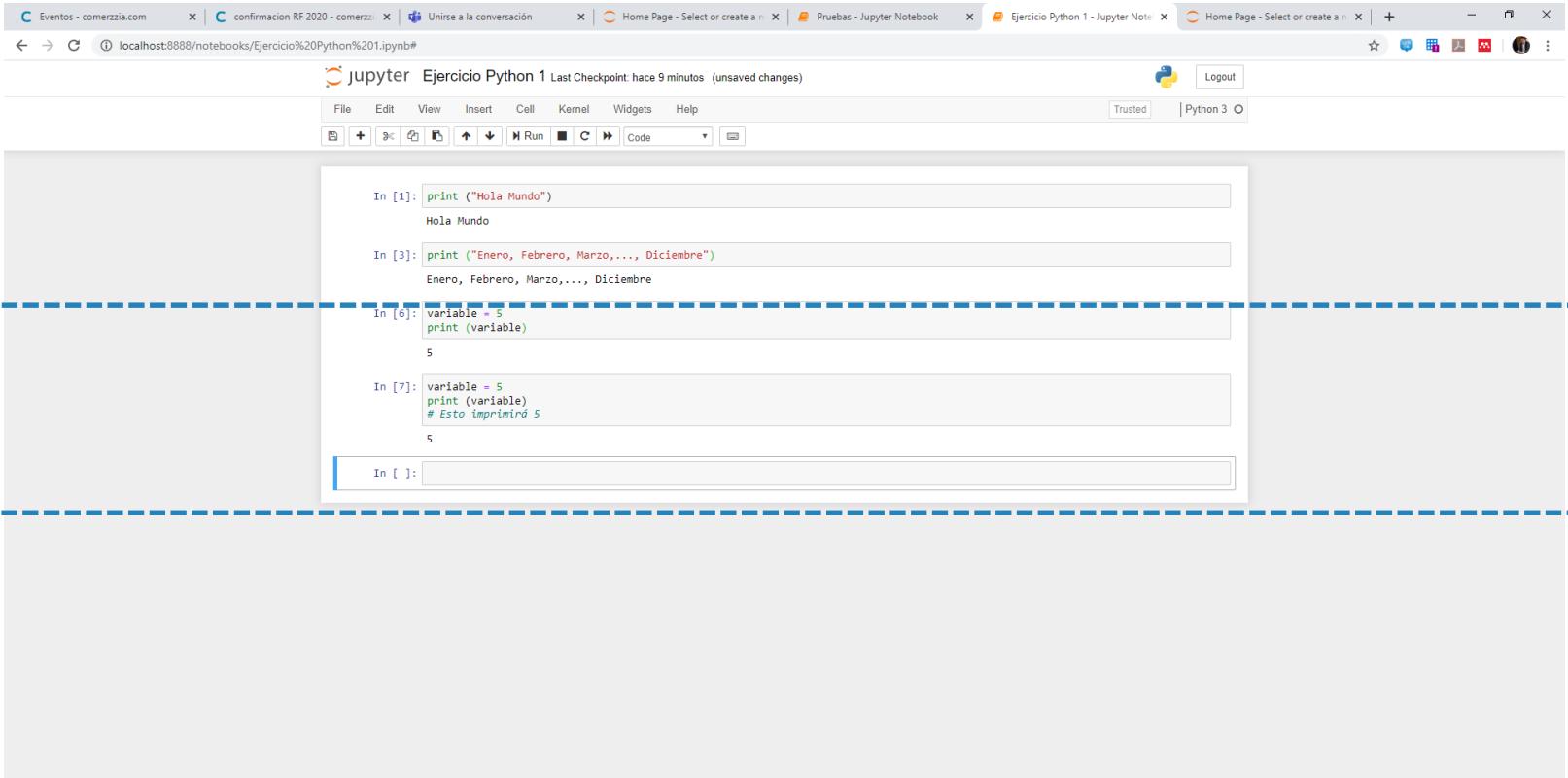
Below the browser is a separate window titled "Python 3.8.1 Shell" showing the Python interpreter's prompt and some basic commands:

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> Type [help", "copyright", "credits" or "license()"]
SyntaxError: invalid syntax
>>> |
>>> print ("Hola Mundo")
Hola Mundo
>>> print ("Enero, Febrero,..., Diciembre")
Enero, Febrero,..., Diciembre
>>>
```

Comenzando con Python

TRABAJANDO CON DATOS

Si solo pudiésemos hacer operaciones con literales, estaríamos bastante limitados. Para conseguir que Python haga por nosotros tareas más complejas necesitamos usar las llamadas variables.



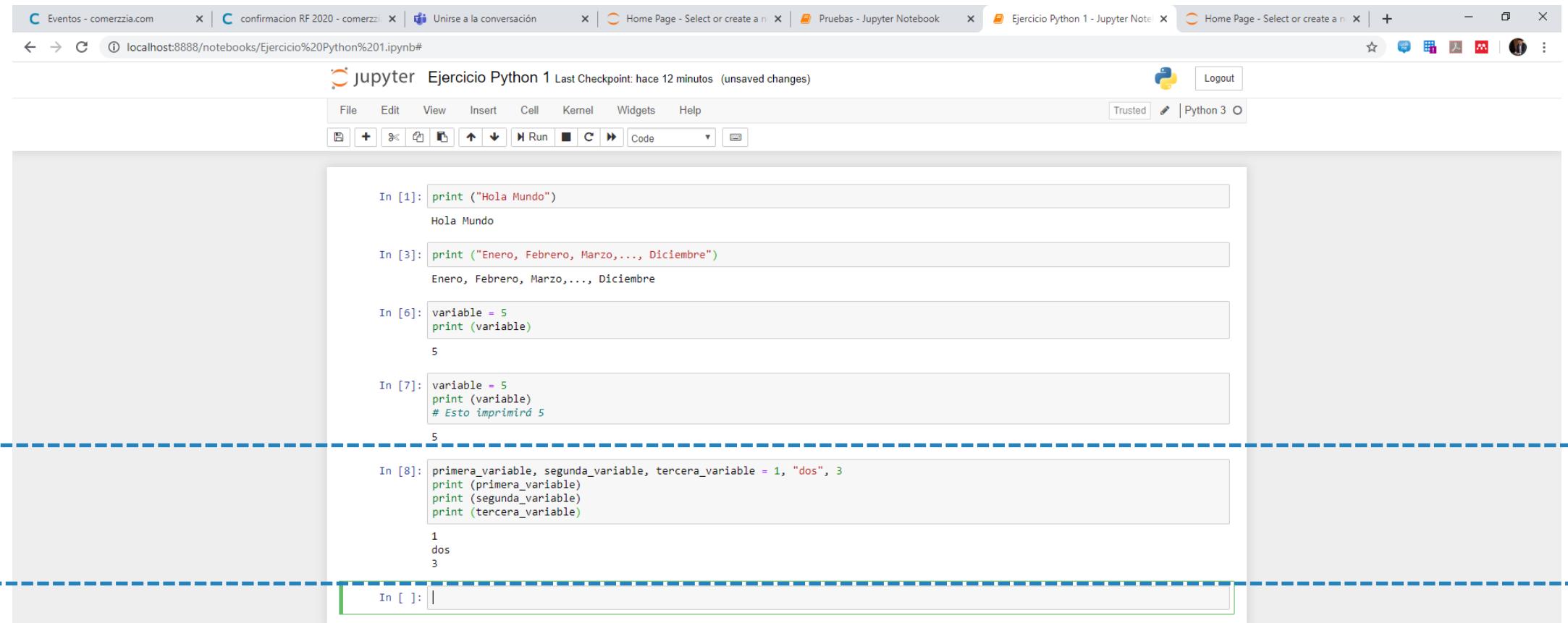
The screenshot shows a Jupyter Notebook interface with several cells of code executed:

- In [1]: `print ("Hola Mundo")`
Output: Hola Mundo
- In [3]: `print ("Enero, Febrero, Marzo,..., Diciembre")`
Output: Enero, Febrero, Marzo,..., Diciembre
- In [6]: `variable = 5
print (variable)`
Output: 5
- In [7]: `variable = 5
print (variable)
Esto imprimirá 5`
Output: 5
- In []:

A large blue dashed rectangle highlights the area containing the code and its output in cells 6 and 7.

Comenzando con Python

TRABAJANDO CON DATOS



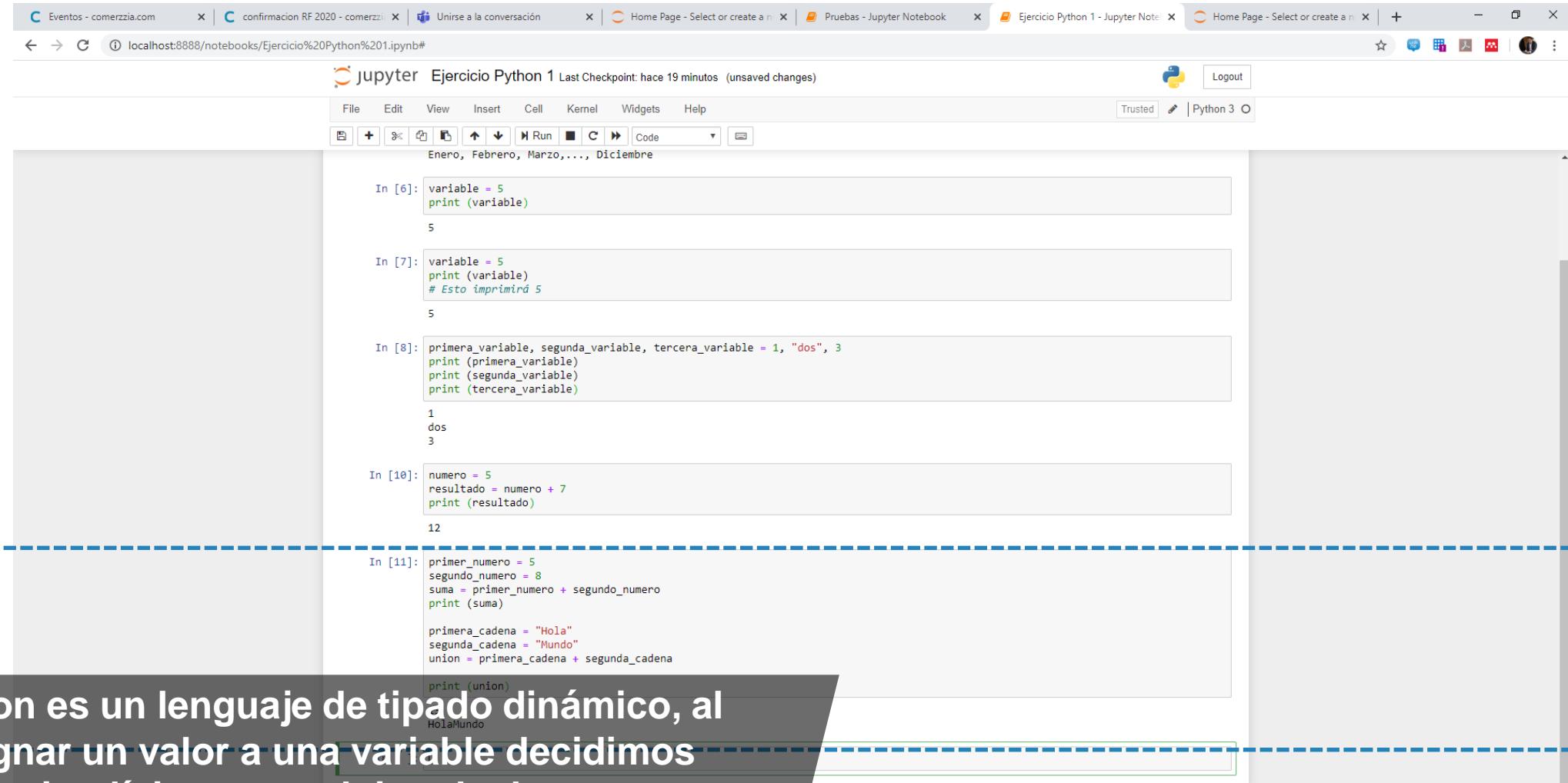
The screenshot shows a Jupyter Notebook interface with several tabs at the top: 'Eventos - comerzzia.com', 'confirmacion RF 2020 - comerzzia...', 'Unirse a la conversación', 'Home Page - Select or create a n...', 'Pruebas - Jupyter Notebook', 'Ejercicio Python 1 - Jupyter Notebo...', and 'Home Page - Select or create a n...'. The active tab is 'Ejercicio Python 1 - Jupyter Notebook'. The notebook contains the following code cells:

- In [1]: `print ("Hola Mundo")`
Output: Hola Mundo
- In [3]: `print ("Enero, Febrero, Marzo,..., Diciembre")`
Output: Enero, Febrero, Marzo,..., Diciembre
- In [6]: `variable = 5
print (variable)`
Output: 5
- In [7]: `variable = 5
print (variable)
Esto imprimirá 5`
Output: 5
- In [8]: `primera_variable, segunda_variable, tercera_variable = 1, "dos", 3
print (primera_variable)
print (segunda_variable)
print (tercera_variable)`
Output: 1
dos
3
- In []: (empty cell)

En Python no es necesario declarar las variables explícitamente antes de usarlas, y adoptan el tipo cuando se les asigna valor.

Comenzando con Python

TRABAJANDO CON DATOS



The screenshot shows a Jupyter Notebook interface with several code cells:

- In [6]:

```
variable = 5
print (variable)
```

Output: 5
- In [7]:

```
variable = 5
print (variable)
# Esto imprimirá 5
```

Output: 5
- In [8]:

```
primera_variable, segunda_variable, tercera_variable = 1, "dos", 3
print (primera_variable)
print (segunda_variable)
print (tercera_variable)
```

Output: 1
 dos
 3
- In [10]:

```
numero = 5
resultado = numero + 7
print (resultado)
```

Output: 12
- In [11]:

```
primer_numero = 5
segundo_numero = 8
suma = primer_numero + segundo_numero
print (suma)

primera_cadena = "Hola"
segunda_cadena = "Mundo"
union = primera_cadena + segunda_cadena

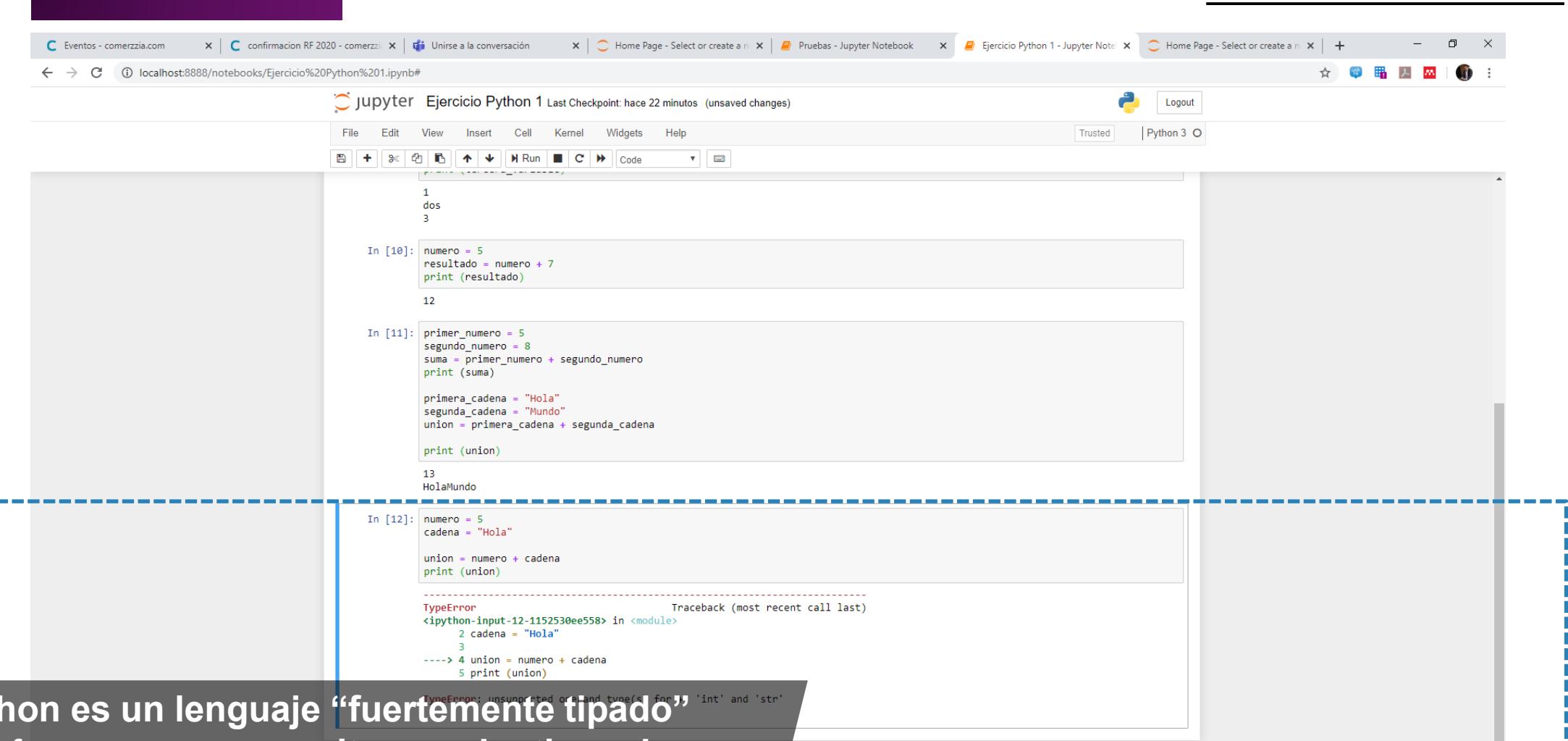
print (union)
```

Output: HolaMundo

Python es un lenguaje de tipado dinámico, al asignar un valor a una variable decidimos implícitamente el tipo de dato.

Comenzando con Python

TRABAJANDO CON DATOS



The screenshot shows a Jupyter Notebook interface with multiple tabs at the top. The active tab is titled "Ejercicio Python 1 - Jupyter Notebook". The notebook contains three code cells:

```
In [10]: numero = 5
resultado = numero + 7
print (resultado)
12

In [11]: primer_numero = 5
segundo_numero = 8
suma = primer_numero + segundo_numero
print (suma)

primera_cadena = "Hola"
segunda_cadena = "Mundo"
union = primera_cadena + segunda_cadena

print (union)
13
HolaMundo

In [12]: numero = 5
Cadena = "Hola"

union = numero + Cadena
print (union)

-----TypeError----- Traceback (most recent call last)
<ipython-input-12-1152530ee558> in <module>
      2 Cadena = "Hola"
      3
----> 4 union = numero + Cadena
      5 print (union)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

A dashed blue rectangle highlights the code in cell In [12] and its resulting error message.

Python es un lenguaje “fuertemente tipado” de forma que no permite mezclar tipos de datos libremente.

Comenzando con Python

TRABAJANDO CON DATOS

Python Tipos de Datos

Tipo	Clase	Notas	Ejemplo
str	Cadena	Inmutable	"Hola"
unicode	Cadena	Versión Unicode de str	u"Hola"
list	Secuencia	Mutable, contiene objetos de diverso tipo	[4, "Hola", 3.14]
tuple	Secuencia	Inmutable, contiene objetos de diverso tipo	(4, "Hola", 3.14)
set	Conjunto	Mutable, sin orden y sin duplicados	set([4, "Hola", 3.14])
frozenset	Conjunto	Inmutable, sin orden, sin duplicados	frozenset([4, "Hola", 3.14])
dict	Diccionario	Pares de clave:valor	{"clave1": 4, "clave2": "Hola"}
int	Entero	Precisión fija, convierte a long si necesario	32
long	Entero	Precisión arbitraria	32L ó 1298918298398923L
float	Decimal	Coma flotante de doble precisión	3.141592
complex	Complejo	Parte real e imaginaria.	(4.5 + 3j)
bool	Booleano	Valores verdadero o falso	True o False

Vamos probando...

Comenzando con Python

TRABAJANDO CON DATOS

ENTEROS (int)

En Python el tamaño máximo que puede tener un **entero** depende de la plataforma pero, como mínimo, será de **32 bits**, lo que permite manejar un rango de números entre el **-2147483647** y el **2147483647**. En ordenadores de **64 bits** el rango sube hasta **-9223372036854775807** y **9223372036854775807**.

ENTEROS LARGOS (long)

La longitud del entero largo es arbitraria, no tiene límite superior e inferior, para especificar que un número se almacene como long en lugar de int, le añadimos la letra “L”.

```
entero_largo = 145L
```

COMA FLOTANTE (float)

Los números en coma flotante (float) representan a los números racionales. El máximo número que permite este tipo es **1.7976931348623157e+308**, y el mínimo **2.2250738585072014e-308**. En Python los decimales se separan con un punto (.)

Comenzando con Python

TRABAJANDO CON DATOS

COMPLEJOS (complex)

Los números complejos son una abstracción matemática, ideada para resolver el problema de las raíces de números negativos. Son aquellos que tienen parte real y parte imaginaria, el formato numérico más exótico de los que hemos visto.

`numero = 34 + 5j`

NOTACIÓN

Para indicar un número, normalmente usamos la notación decimal, en base 10, pero no es la única que Python permite...

`numero = 0b1001` (binario)
`numero = 0xB16A` (hexadecimal)
`numero = 12e5` (científica)

**Probemos con el editor Jupyter a ver qué
salida nos da la variable número...**

Comenzando con Python

TRABAJANDO CON DATOS

BOOLEANO (bool)

El tipo booleano sirve para mostrar valores lógicos (True, False).

CADERAS (str)

Una cadena es un trozo de texto, también puede contener números y todo tipo de caracteres... pero siempre tratados como texto.

Código	Significado
\ \	Barra invertida ()
\ '	Comilla simple ('')
\ "	Comilla doble ("")
\a	Campana (BEL)
\b	Retroceso (BS)
\f	Salto de página (FF)
\n	Nueva línea (LF)
\r	Retorno de carro (CR)
\t	Tabulador (TAB)
\v	Tabulador vertical Tab (VT)

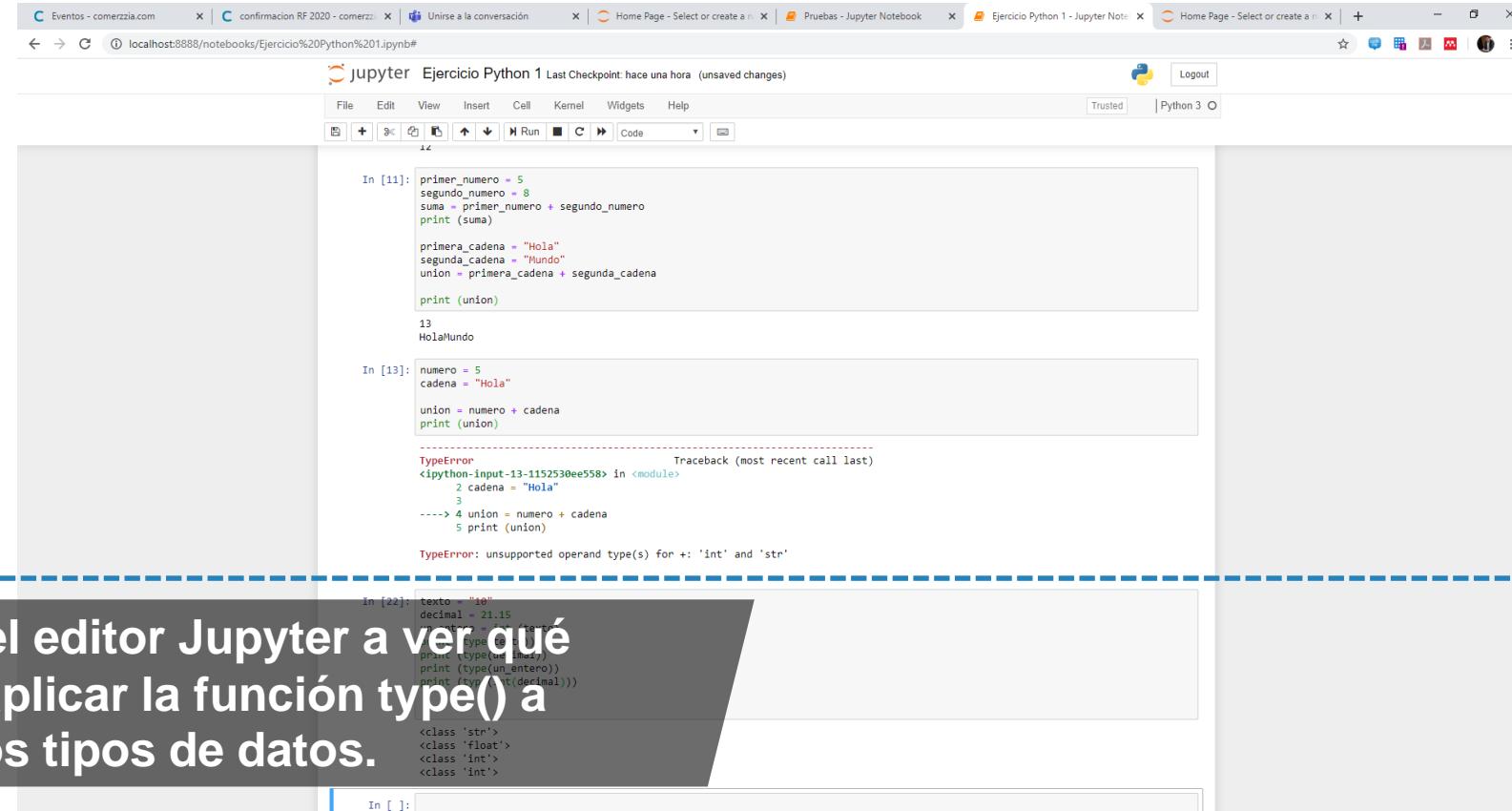
Probemos con el editor Jupyter a ver qué salida nos da aplicar en una cadena estos códigos...

Comenzando con Python

TRABAJANDO CON DATOS

NONE (None)

Indica la ausencia de valor... Además Python pone a nuestra disposición la función ***type()*** para averiguar el tipo de una variable o bien de un literal.



The screenshot shows a Jupyter Notebook interface with several tabs at the top: 'Eventos - comerzio.com', 'confirmacion RF 2020 - comerzio.com', 'Unirse a la conversación', 'Home Page - Select or create a...', 'Pruebas - Jupyter Notebook', 'Ejercicio Python 1 - Jupyter Notebook', and 'Home Page - Select or create a...'. The main area displays two code cells:

```
In [11]: primer_numero = 5
segundo_numero = 8
suma = primer_numero + segundo_numero
print (suma)

primera_cadena = "Hola"
segunda_cadena = "Mundo"
union = primera_cadena + segunda_cadena

print (union)
13
HolaMundo
```

```
In [13]: numero = 5
cadena = "Hola"

union = numero + cadena
print (union)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Below the cells, a traceback is shown:

```
Traceback (most recent call last)
<ipython-input-13-1152530ee558> in <module>
      2     cadena = "Hola"
      3
----> 4 union = numero + cadena
      5 print (union)

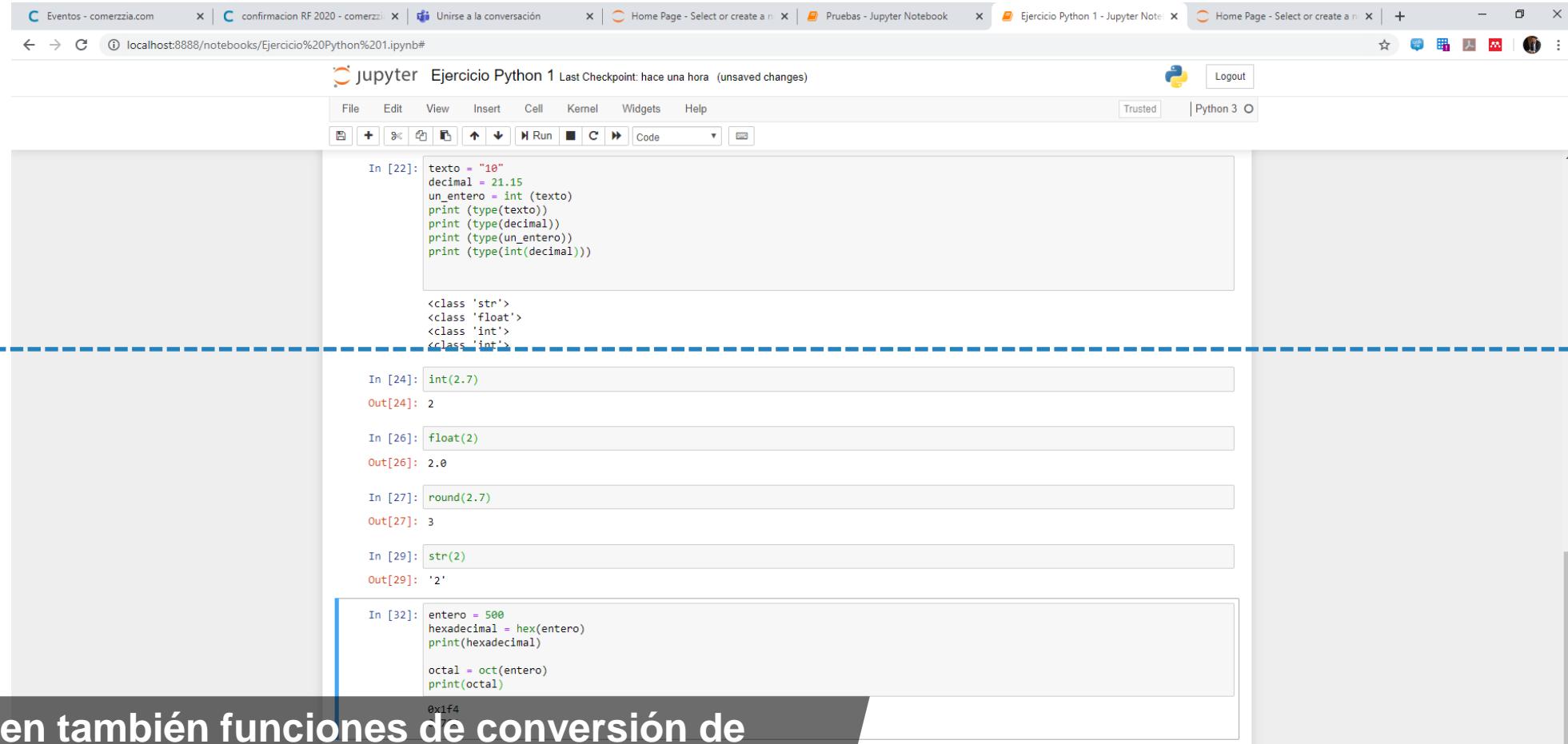
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

A callout box with a dashed blue border highlights the text:

Probemos con el editor Jupyter a ver qué salida nos da aplicar la función ***type()*** a distintos tipos de datos.

Comenzando con Python

TRABAJANDO CON DATOS



The screenshot shows a Jupyter Notebook interface with the title "Ejercicio Python 1". The notebook contains several code cells:

```
In [22]: texto = "10"
decimal = 21.15
un_entero = int(texto)
print(type(texto))
print(type(decimal))
print(type(un_entero))
print(type(int(decimal)))
```

```
In [24]: int(2.7)
Out[24]: 2
```

```
In [26]: float(2)
Out[26]: 2.0
```

```
In [27]: round(2.7)
Out[27]: 3
```

```
In [29]: str(2)
Out[29]: '2'
```

```
In [32]: entero = 500
hexadecimal = hex(entero)
print(hexadecimal)

octal = oct(entero)
print(octal)
```

The output for the last cell is:

```
0x1f4
0774
```

Existen también funciones de conversión de datos... Revisad las que están disponibles en Python.

Comenzando con Python

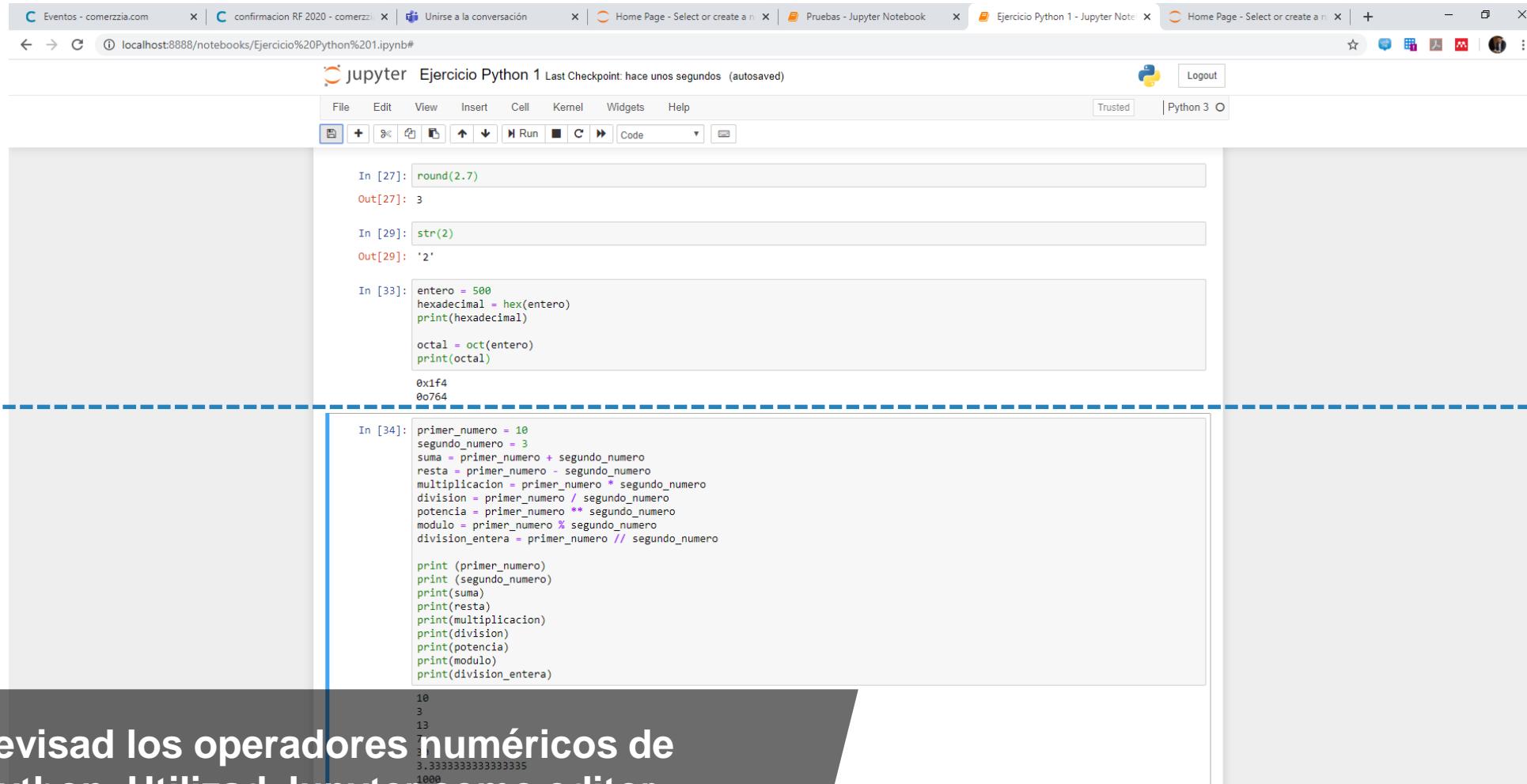
OPERADORES

Los operadores matemáticos se aplican a valores numéricos, consignan las operaciones matemáticas básicas y son los siguientes:

- ▶ + (suma)
- ▶ - (resta)
- ▶ * (multiplicación)
- ▶ / (división)
- ▶ ** (exponente)
- ▶ // (división entera)
- ▶ % (módulo)

Comenzando con Python

OPERADORES



The screenshot shows a Jupyter Notebook interface with several cells of code and their outputs:

- In [27]: `round(2.7)`
Out[27]: 3
- In [29]: `str(2)`
Out[29]: '2'
- In [33]:

```
entero = 500
hexadecimal = hex(entero)
print(hexadecimal)

octal = oct(entero)
print(octal)
```


Out[33]:
0x1f4
0764
- In [34]:

```
primer_numero = 10
segundo_numero = 3
suma = primer_numero + segundo_numero
resta = primer_numero - segundo_numero
multiplicacion = primer_numero * segundo_numero
division = primer_numero / segundo_numero
potencia = primer_numero ** segundo_numero
modulo = primer_numero % segundo_numero
division_entera = primer_numero // segundo_numero

print(primer_numero)
print(segundo_numero)
print(suma)
print(resta)
print(multiplicacion)
print(division)
print(potencia)
print(modulo)
print(division_entera)
```


Out[34]:
10
3
13
7
3.333333333333335
1000
1
3

Revisad los operadores numéricos de Python. Utilizad Jupyter como editor.

Comenzando con Python

OPERADORES

Para todos estos operadores existe una versión con asignación, cuyo símbolo es el mismo pero seguido de un signo igual (=):

- ▶ `+=` (suma con asignación)
- ▶ `-=` (resta con asignación)
- ▶ `*=` (multiplicación con asignación)
- ▶ `/=` (división con asignación)
- ▶ `**=` (exponente con asignación)
- ▶ `//=` (división entera con asignación)
- ▶ `%=` (módulo con asignación)

Revisad los operadores numéricos de asignación en Python. Utilizad Jupyter como editor.

Comenzando con Python

OPERADORES

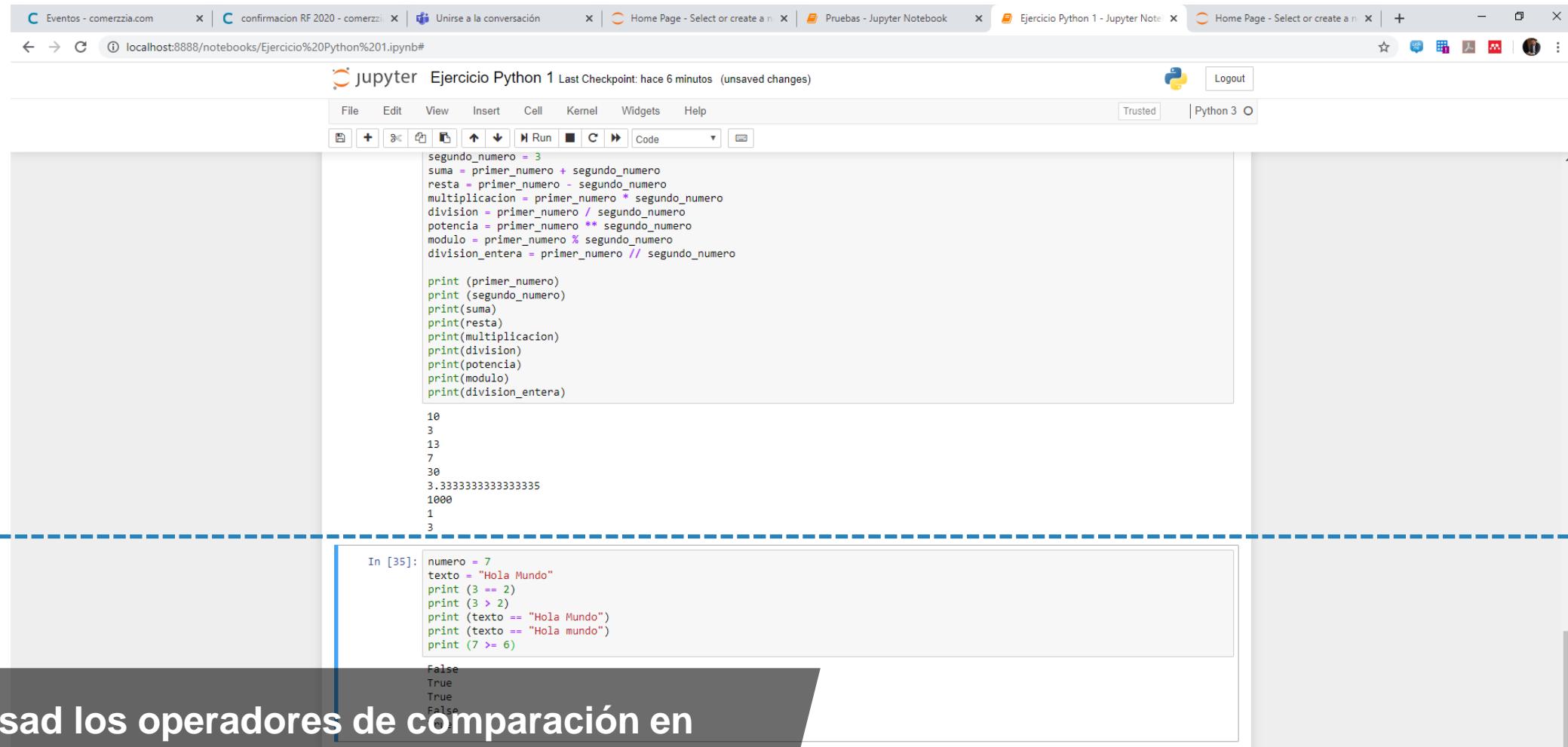
Los operadores de comparación sirven para comparar números, cadenas, objetos y, en general, cualquier cosa en Python. Devuelven siempre un valor lógico (True si se cumple la condición o False si no se cumple) y son los siguientes:

- ▶ == (igualdad)
- ▶ != (desigualdad)
- ▶ <> (desigualdad)
- ▶ > (mayor que)
- ▶ < (menor que)
- ▶ >= (mayor o igual que)
- ▶ <= (menor o igual que)

Revisad los operadores de comparación en Python. Utilizad Jupyter como editor.

Comenzando con Python

OPERADORES



The screenshot shows a Jupyter Notebook interface with several tabs at the top, including "Pruebas - Jupyter Notebook" and "Ejercicio Python 1 - Jupyter Notebook". The main area displays two code cells. The first cell contains code for arithmetic operations:segundo_numero = 3
suma = primer_numero + segundo_numero
resta = primer_numero - segundo_numero
multiplicacion = primer_numero * segundo_numero
division = primer_numero / segundo_numero
potencia = primer_numero ** segundo_numero
modulo = primer_numero % segundo_numero
division_entera = primer_numero // segundo_numero

print(primer_numero)
print(segundo_numero)
print(suma)
print(resta)
print(multiplicacion)
print(division)
print(potencia)
print(modulo)
print(division_entera)

The output of this cell is:

```
10
3
13
7
30
3.333333333333335
1000
1
3
```

The second cell, labeled "In [35]:", contains code for comparison operators:numero = 7
texto = "Hola Mundo"
print(3 == 2)
print(3 > 2)
print(texto == "Hola Mundo")
print(texto == "Hola mundo")
print(7 >= 6)

The output of this cell is:

```
False
True
True
False
True
```

A large blue dashed rectangle highlights the second cell, containing the comparison operator examples.

Revisad los operadores de comparación en Python. Utilizad Jupyter como editor.

Comenzando con Python

OPERADORES

El operador `and` equivale a la conjunción “y lógica”, y retorna un valor `True` solo si ambos operandos son también `True`.

Tabla del operador `and`:

Primer operando	Segundo operando	Resultado
True	True	True
True	False	False
False	True	False
False	False	False

Los operadores booleanos sirven para realizar operaciones con valores lógicos (AND, OR, NOT)

Comenzando con Python

OPERADORES

El operador lógico `or` equivale a la conjunción “o lógica”, y retorna un valor `True` siempre que no sean falsos ambos operandos.

Tabla del operador `or`:

Primer operando	Segundo operando	Resultado
True	True	True
True	False	True
False	True	True
False	False	False

Los operadores booleanos sirven para realizar operaciones con valores lógicos (AND, OR, NOT)

Comenzando con Python

OPERADORES

Al contrario que los dos anteriores, el operador `not` se aplica sobre un solo operando, e invierte el valor lógico de este. Si el operando es `True` devolverá `False`, y viceversa.

Tabla del operador `not`:

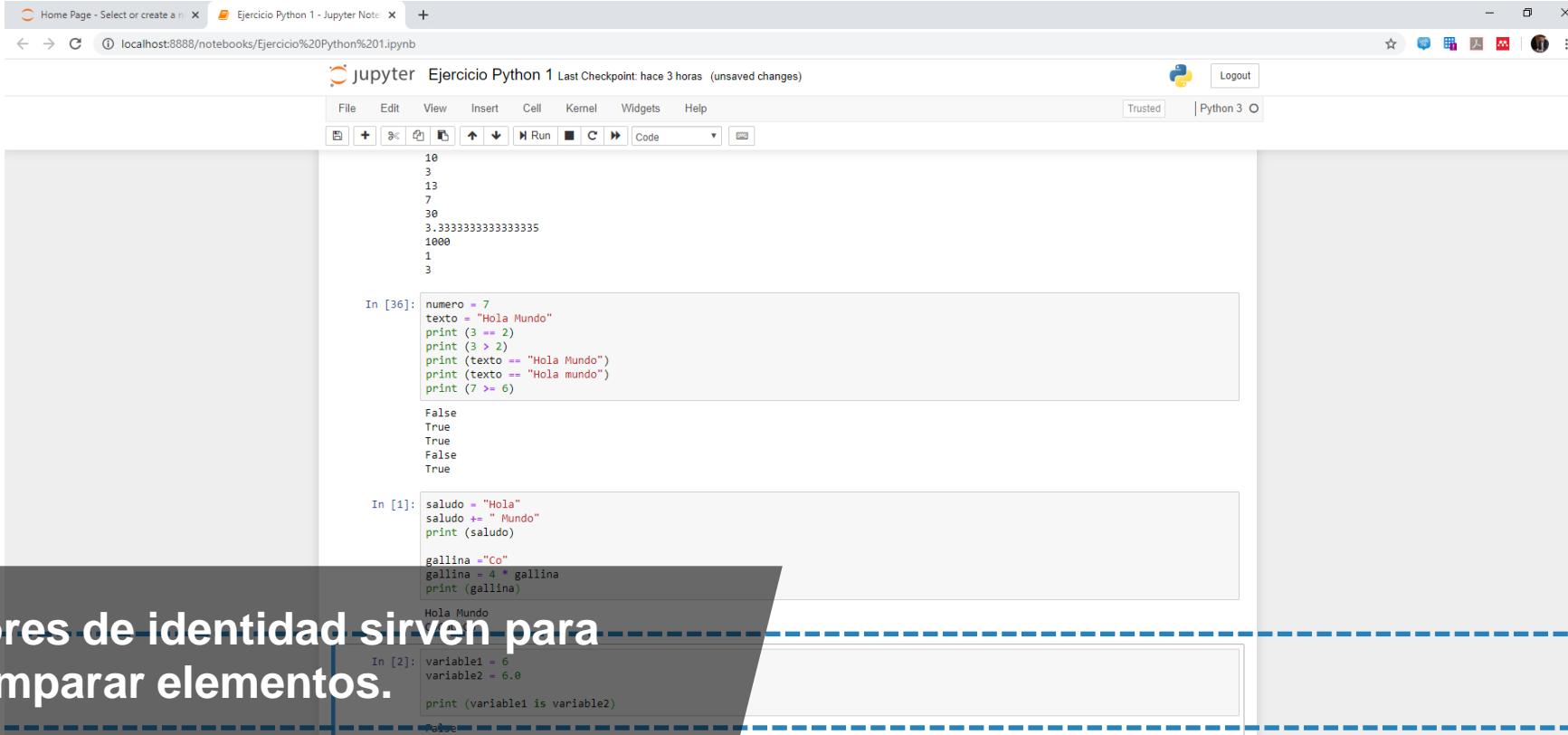
Operando	Resultado
<code>True</code>	<code>False</code>
<code>False</code>	<code>True</code>

Los operadores booleanos sirven para realizar operaciones con valores lógicos (AND, OR, NOT)

Comenzando con Python

OPERADORES

Un caso especial son los **operadores de identidad**, *is* y *is not*. Ambos sirven para comparar elementos. El operador *is* devuelve True si ambos elementos comparados son el mismo objeto (poseen el mismo id), y False en caso contrario. El operador *is not* hace lo opuesto.



The screenshot shows a Jupyter Notebook interface with two code cells. The first cell contains:10
3
13
7
30
3.333333333333335
1000
1
3In [36]: numero = 7
texto = "Hola Mundo"
print (3 == 2)
print (3 > 2)
print (texto == "Hola Mundo")
print (texto == "Hola mundo")
print (7 > 6)Output:
False
True
True
False
TrueThe second cell contains:

```
In [1]: saludo = "Hola"  
saludo += " Mundo"  
print (saludo)  
  
gallina ="Co"  
gallina = 4 * gallina  
print (gallina)
```

Output:
Hola Mundo
CoCoCoCoThe third cell at the bottom is partially visible:

```
In [2]: variable1 = 6  
variable2 = 6.0  
  
print (variable1 is variable2)
```

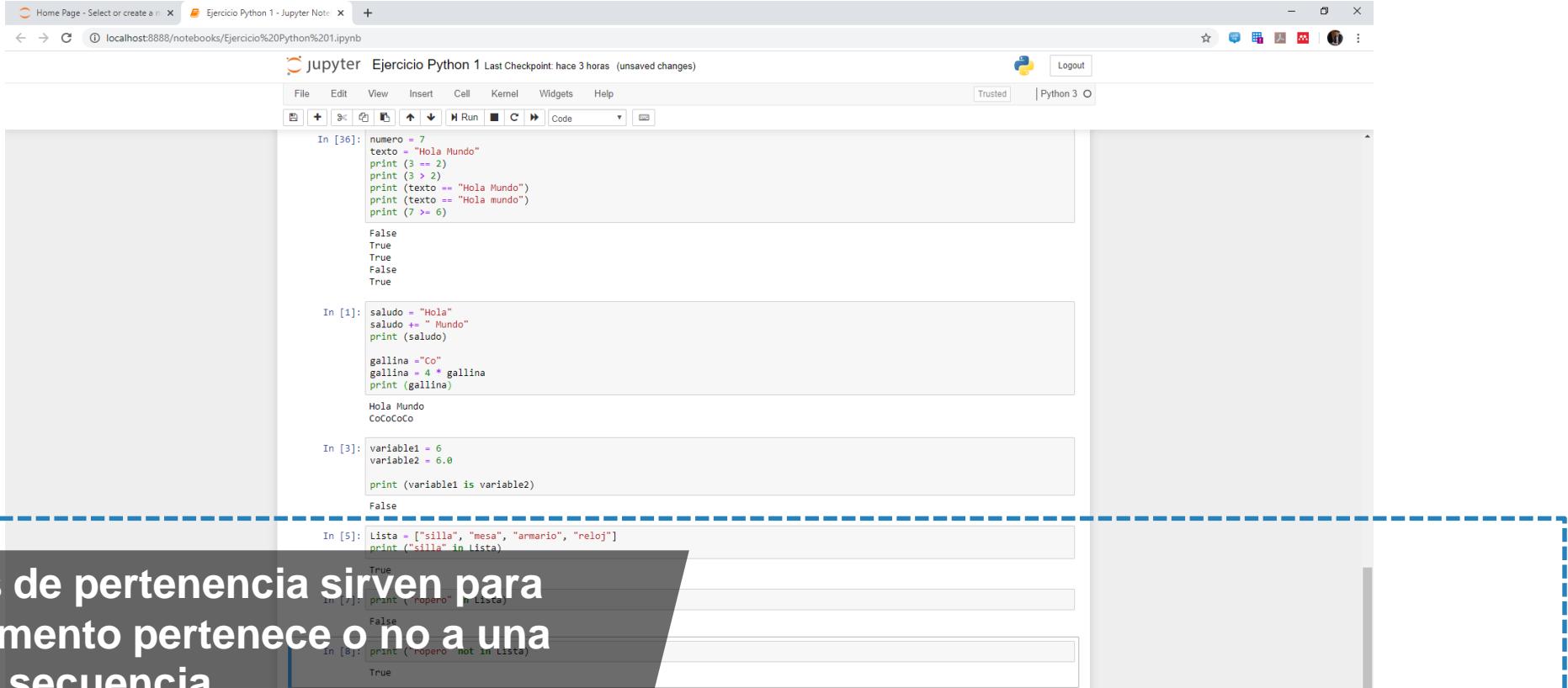
Output:
FalseA callout box with a dashed border points from the text above to the first cell of the notebook, highlighting the code example.

Los operadores de identidad sirven para comparar elementos.

Comenzando con Python

OPERADORES

El operador *in* retorna un valor True si el elemento indicado a la izquierda del operador está en la secuencia indicada a la derecha del operador. El operador *not in* hace lo contrario, y da True si el elemento no está en la secuencia y False si lo está.



The screenshot shows a Jupyter Notebook interface with several code cells and their outputs:

- In [36]:** numero = 7
texto = "Hola Mundo"
print (3 == 2)
print (3 > 2)
print (texto == "Hola Mundo")
print (texto == "Hola mundo")
print (7 >= 6)
Output:
False
True
True
False
True
- In [1]:** saludo = "Hola"
saludo += " Mundo"
print (saludo)

gallina ="Co"
gallina = 4 * gallina
print (gallina)
Output:
Hola Mundo
cocococo
- In [3]:** variable1 = 6
variable2 = 6.0

print (variable1 is variable2)
Output:
False
- In [5]:** Lista = ["silla", "mesa", "armario", "reloj"]
print ("silla" in Lista)

In [7]: print ("silla" not in Lista)
Output:
True
False
- In [8]:** print ("silla" not in Lista)
Output:
True

Los operadores de pertenencia sirven para indicar si un elemento pertenece o no a una secuencia.

Comenzando con Python

OPERADORES

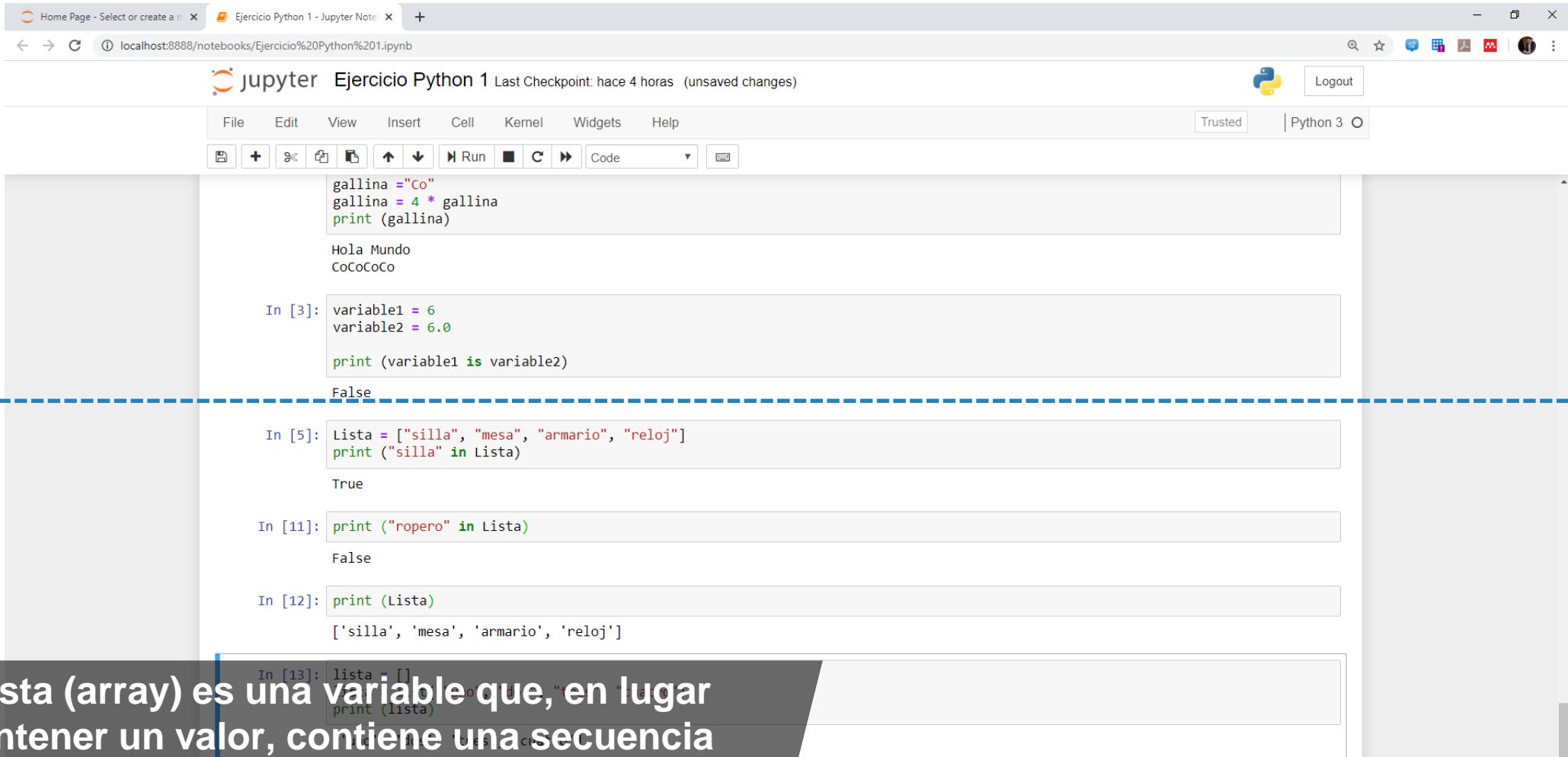
La evaluación de las operaciones se hace de acuerdo a la siguiente lista, siempre se pueden utilizar paréntesis para cambiar el orden:

** (potencia)
~ (complemento a uno)
* (multiplicación), / (división), // (división entera), % (módulo)
+ (suma), - (resta)
<< (desplazamiento lógico a la izquierda), >> (desplazamiento lógico a la derecha)
& (AND binario)
^ (XOR binario)
(OR binario)
> (mayor que), < (menor que), >= (mayor o igual que), <= (menor o igual que), == (igualdad),
!= y != (desigualdad), “is” y “not is” (operadores de identidad)
“in” y “not in” (operadores de pertenencia)
“not” (negación)
“and” (y lógica)
“or” (o lógica)

En Python las expresiones se evalúan de izquierda a derecha, excepto las asignaciones.

Comenzando con Python

SECUENCIAS Y ESTRUCTURAS DE DATOS



The screenshot shows a Jupyter Notebook interface with the title "Ejercicio Python 1 - Jupyter Notebooks". The notebook contains the following code snippets:

```
gallina = "Co"
gallina = 4 * gallina
print (gallina)

Hola Mundo
CoCoCoCo
```

```
In [3]: variable1 = 6
variable2 = 6.0

print (variable1 is variable2)

False
```

```
In [5]: Lista = ["silla", "mesa", "armario", "reloj"]
print ("silla" in Lista)

True
```

```
In [11]: print ("ropero" in Lista)

False
```

```
In [12]: print (Lista)

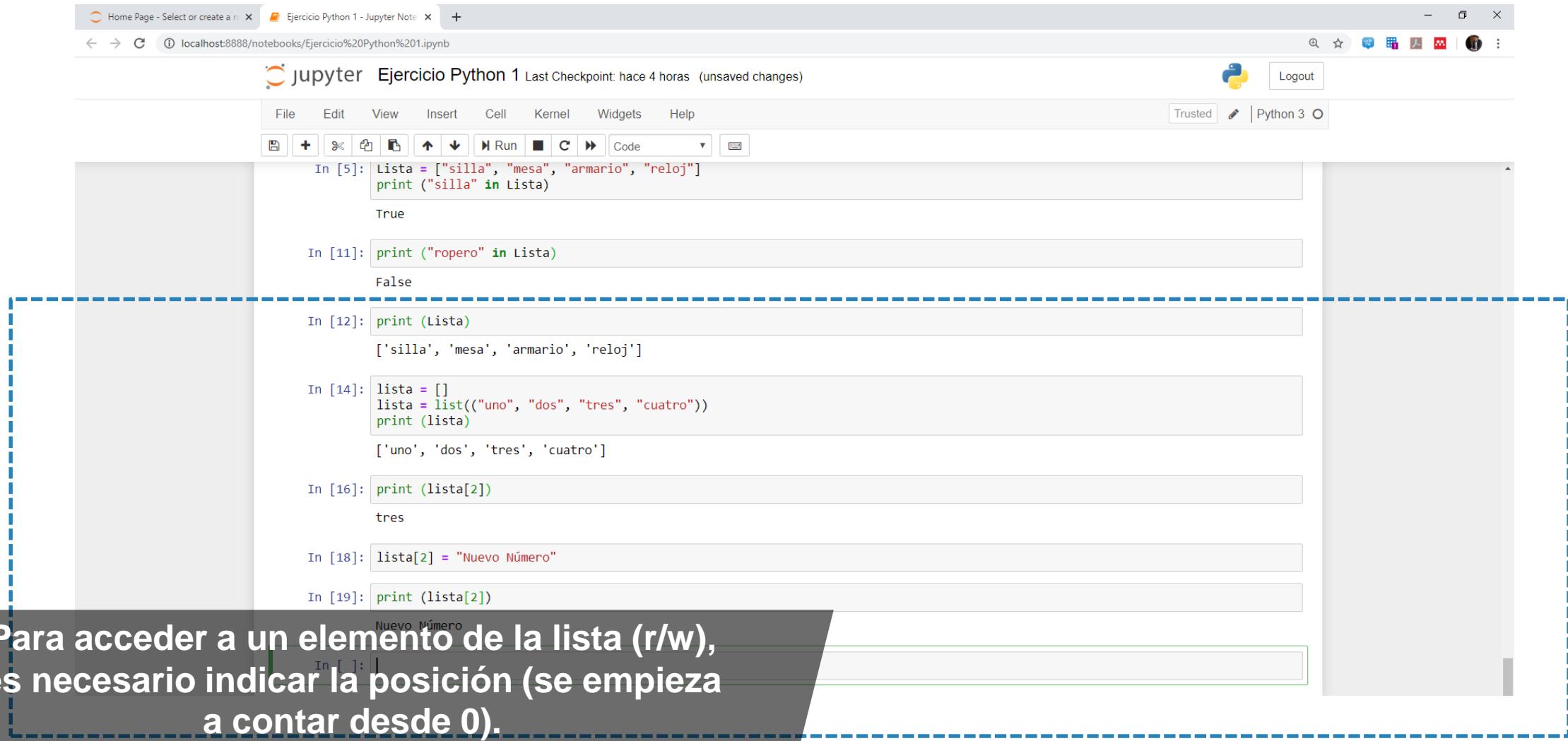
['silla', 'mesa', 'armario', 'reloj']
```

A blue dashed box highlights the last three code cells (In [5], In [11], and In [12]), which demonstrate the use of lists (arrays) in Python.

Una lista (array) es una variable que, en lugar de contener un valor, contiene una secuencia ordenada de estos.

Comenzando con Python

SECUENCIAS Y ESTRUCTURAS DE DATOS



The screenshot shows a Jupyter Notebook interface with the title "Ejercicio Python 1". The notebook contains several code cells:

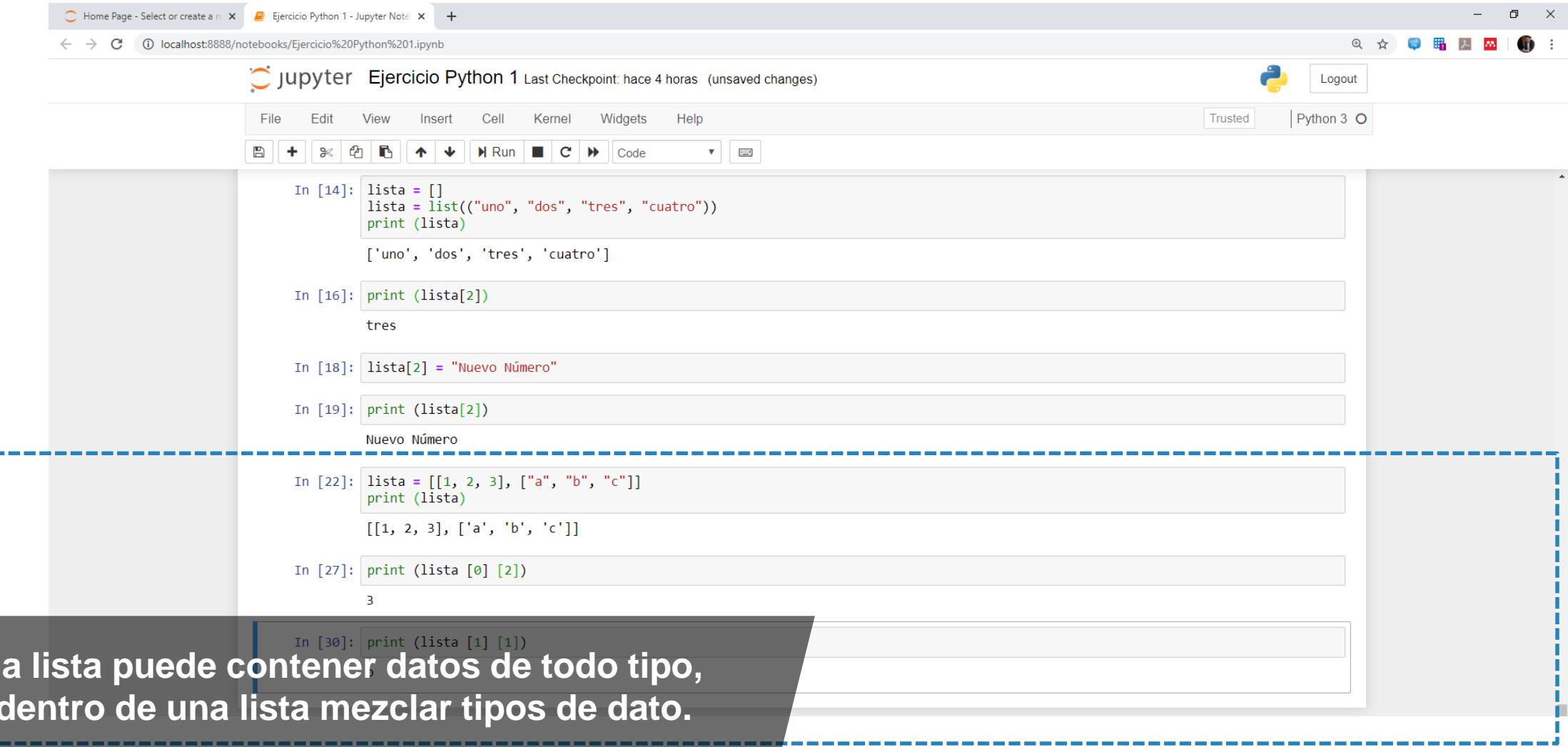
- In [5]: `Lista = ["silla", "mesa", "armario", "reloj"]
print ("silla" in Lista)`
Output: True
- In [11]: `print ("ropero" in Lista)`
Output: False
- In [12]: `print (Lista)`
Output: ['silla', 'mesa', 'armario', 'reloj']
- In [14]: `lista = []
lista = list(("uno", "dos", "tres", "cuatro"))
print (lista)`
Output: ['uno', 'dos', 'tres', 'cuatro']
- In [16]: `print (lista[2])`
Output: tres
- In [18]: `lista[2] = "Nuevo Número"`
- In [19]: `print (lista[2])`
Output: Nuevo Número

A large blue dashed rectangular box highlights the last three code cells (In [18], In [19], and their output). A dark grey overlay box with white text is positioned at the bottom left, containing the following text:

Para acceder a un elemento de la lista (r/w),
es necesario indicar la posición (se empieza
a contar desde 0).

Comenzando con Python

SECUENCIAS Y ESTRUCTURAS DE DATOS



The screenshot shows a Jupyter Notebook interface with the title "Ejercicio Python 1 - Jupyter Notebooks". The notebook contains the following code snippets:

```
In [14]: lista = []
lista = list(("uno", "dos", "tres", "cuatro"))
print (lista)
['uno', 'dos', 'tres', 'cuatro']

In [16]: print (lista[2])
tres

In [18]: lista[2] = "Nuevo Número"

In [19]: print (lista[2])
Nuevo Número

In [22]: lista = [[1, 2, 3], ["a", "b", "c"]]
print (lista)
[[1, 2, 3], ['a', 'b', 'c']]

In [27]: print (lista [0] [2])
3

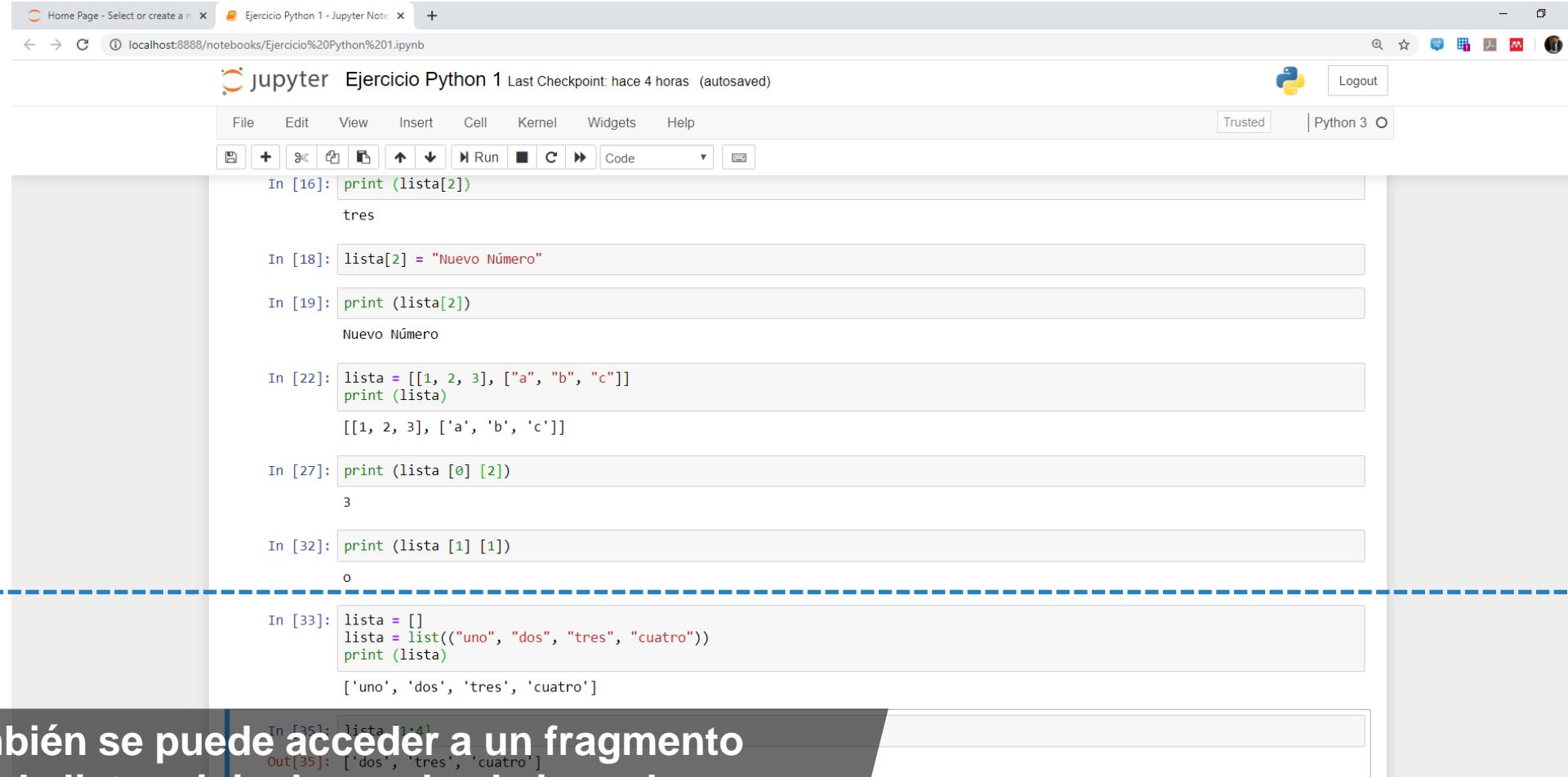
In [30]: print (lista [1] [1])
```

A blue dashed rectangle highlights the last two code cells, which demonstrate nested lists.

**Una lista puede contener datos de todo tipo,
y dentro de una lista mezclar tipos de dato.**

Comenzando con Python

SECUENCIAS Y ESTRUCTURAS DE DATOS



The screenshot shows a Jupyter Notebook interface with the title "Ejercicio Python 1 - Jupyter Notebooks". The notebook contains the following code snippets:

```
In [16]: print (lista[2])
tres

In [18]: lista[2] = "Nuevo Número"

In [19]: print (lista[2])
Nuevo Número

In [22]: lista = [[1, 2, 3], ["a", "b", "c"]]
print (lista)
[[1, 2, 3], ['a', 'b', 'c']]

In [27]: print (lista [0] [2])
3

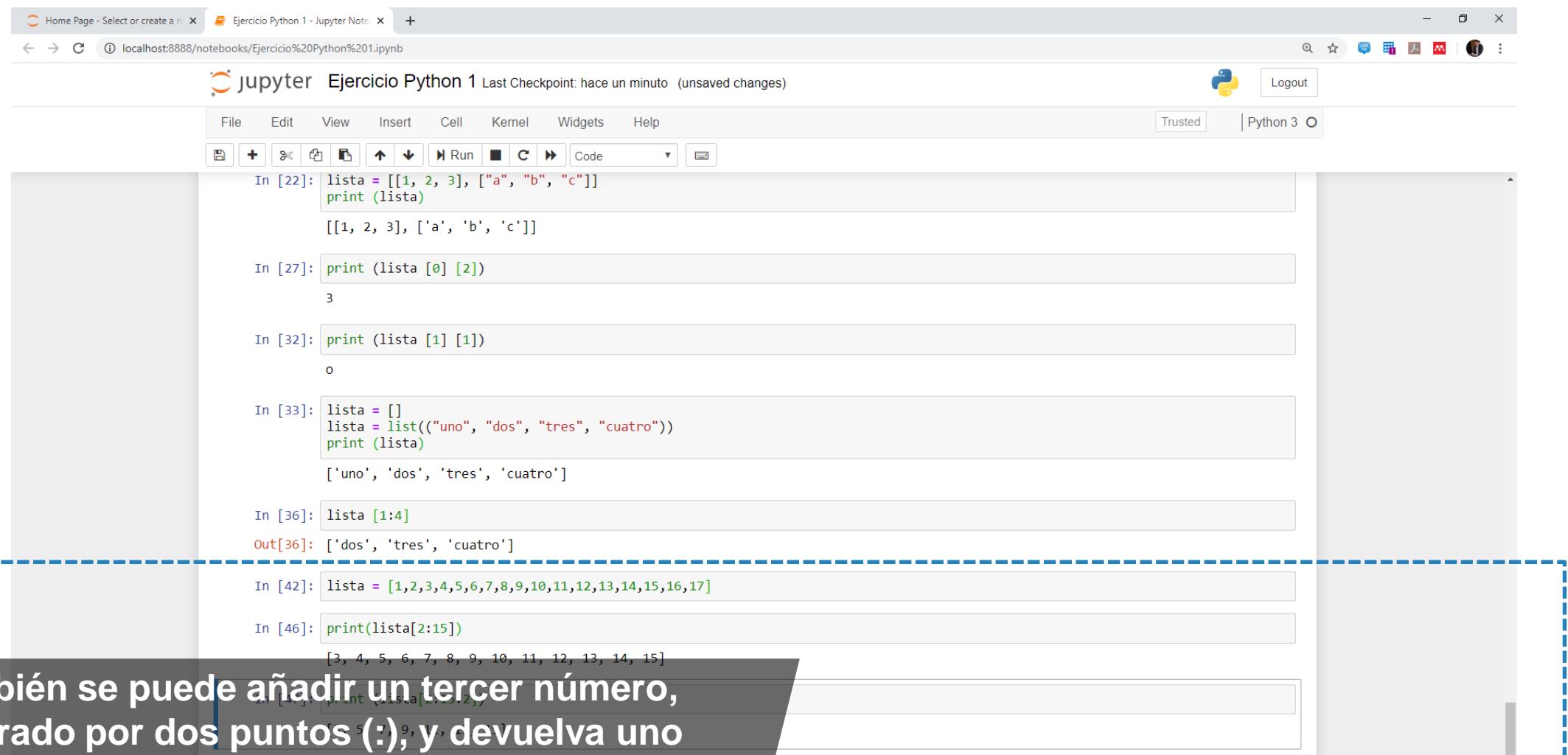
In [32]: print (lista [1] [1])
0

In [33]: lista = []
lista = list(("uno", "dos", "tres", "cuatro"))
print (lista)
['uno', 'dos', 'tres', 'cuatro']
```

También se puede acceder a un fragmento de la lista original usando el signo dos puntos (:).

Comenzando con Python

SECUENCIAS Y ESTRUCTURAS DE DATOS



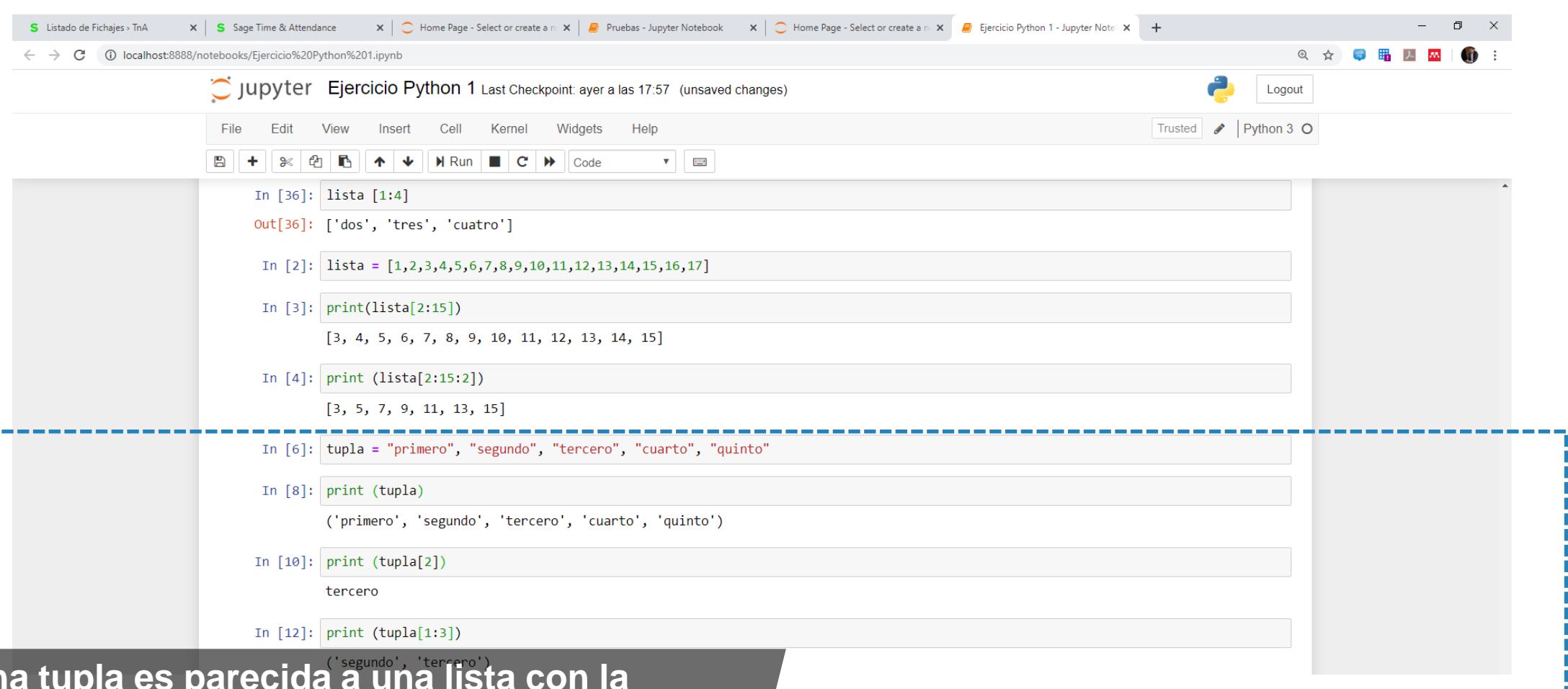
The screenshot shows a Jupyter Notebook interface with the title "Ejercicio Python 1 - Jupyter Notebooks". The notebook contains several code cells:

- In [22]: `lista = [[1, 2, 3], ["a", "b", "c"]]`
Out[22]: `[[1, 2, 3], ['a', 'b', 'c']]`
- In [27]: `print (lista [0] [2])`
Out[27]: `3`
- In [32]: `print (lista [1] [1])`
Out[32]: `o`
- In [33]: `lista = []`
`lista = list(("uno", "dos", "tres", "cuatro"))`
`print (lista)`
Out[33]: `['uno', 'dos', 'tres', 'cuatro']`
- In [36]: `lista [1:4]`
Out[36]: `['dos', 'tres', 'cuatro']`
- In [42]: `lista = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]`
- In [46]: `print(lista[2:15])`
Out[46]: `[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]`

También se puede añadir un tercer número,
separado por dos puntos (:), y devuelva uno
de cada dos...

Comenzando con Python

SECUENCIAS Y ESTRUCTURAS DE DATOS



The screenshot shows a Jupyter Notebook interface with the title "Ejercicio Python 1". The notebook contains the following code:

```
In [36]: lista [1:4]
out[36]: ['dos', 'tres', 'cuatro']

In [2]: lista = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]

In [3]: print(lista[2:15])
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

In [4]: print (lista[2:15:2])
[3, 5, 7, 9, 11, 13, 15]

In [6]: tupla = "primero", "segundo", "tercero", "cuarto", "quinto"
In [8]: print (tupla)
('primero', 'segundo', 'tercero', 'cuarto', 'quinto')

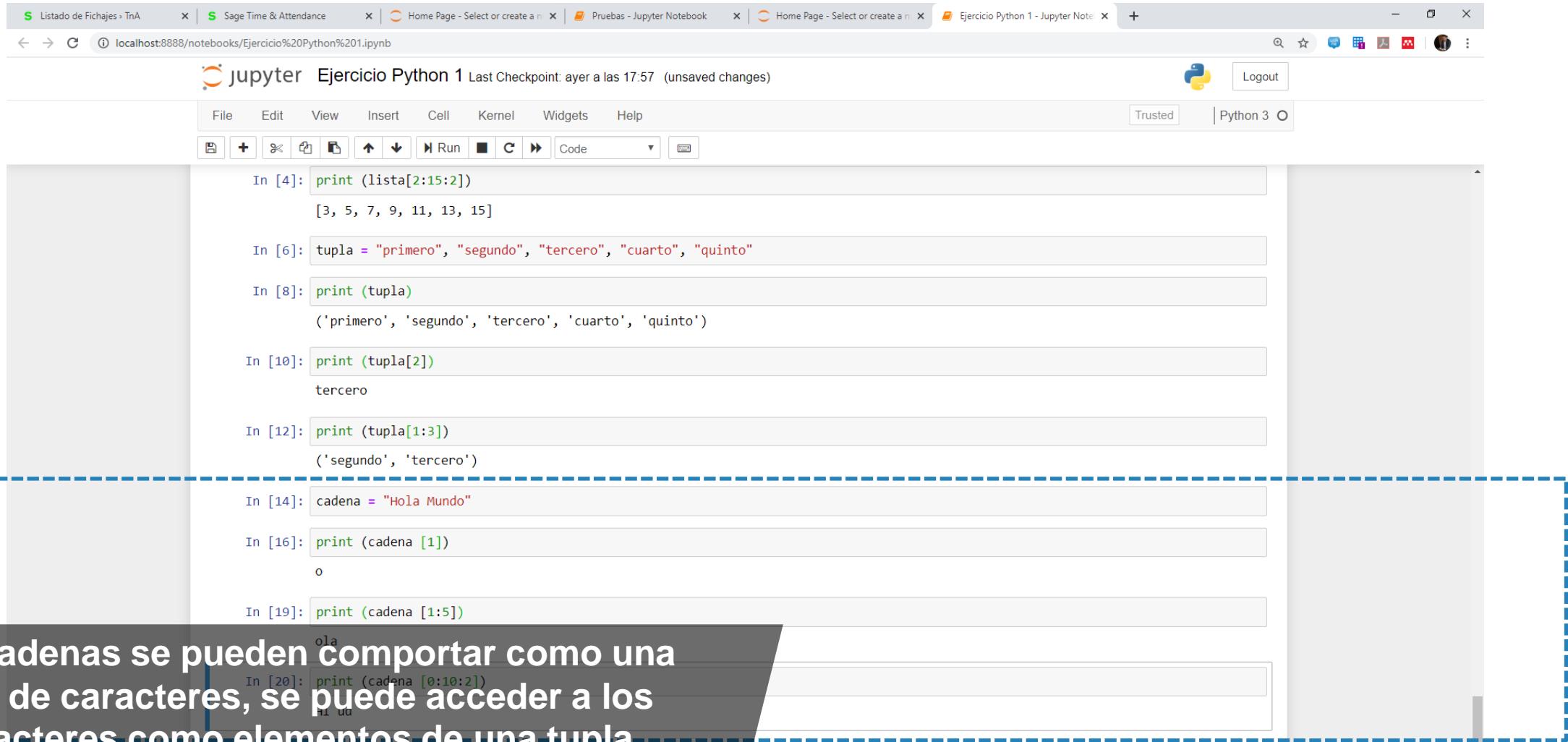
In [10]: print (tupla[2])
tercero

In [12]: print (tupla[1:3])
('segundo', 'tercero')
```

Una tupla es parecida a una lista con la salvedad que no puede modificarse. Las tuplas son objetos inmutables.

Comenzando con Python

SECUENCIAS Y ESTRUCTURAS DE DATOS



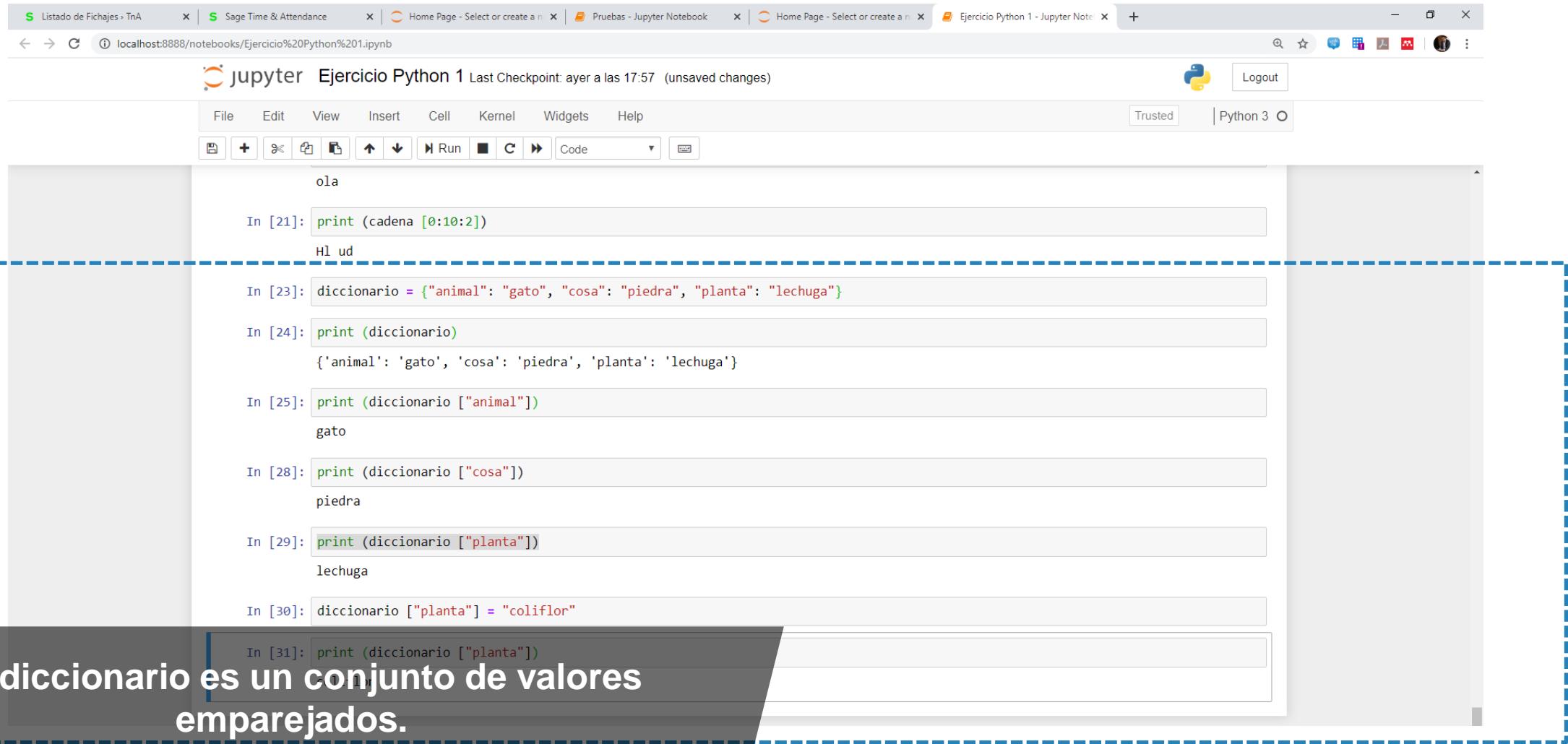
The screenshot shows a Jupyter Notebook interface with several code cells and their outputs:

- In [4]: `print (lista[2:15:2])`
[3, 5, 7, 9, 11, 13, 15]
- In [6]: `tupla = "primero", "segundo", "tercero", "cuarto", "quinto"`
- In [8]: `print (tupla)`
('primero', 'segundo', 'tercero', 'cuarto', 'quinto')
- In [10]: `print (tupla[2])`
tercero
- In [12]: `print (tupla[1:3])`
('segundo', 'tercero')
- In [14]: `cadena = "Hola Mundo"`
- In [16]: `print (cadena [1])`
o
- In [19]: `print (cadena [1:5])`
ola
- In [20]: `print (cadena [0:10:2])`
Hl d

Las cadenas se pueden comportar como una lista de caracteres, se puede acceder a los caracteres como elementos de una tupla.

Comenzando con Python

SECUENCIAS Y ESTRUCTURAS DE DATOS



The screenshot shows a Jupyter Notebook interface with several tabs at the top: 'Listado de Fichajes > TnA', 'Sage Time & Attendance', 'Home Page - Select or create a new notebook', 'Pruebas - Jupyter Notebook', 'Home Page - Select or create a new notebook', and 'Ejercicio Python 1 - Jupyter Notebook'. The active tab is 'Ejercicio Python 1 - Jupyter Notebook'. The notebook content is as follows:

```
In [21]: print (cadena [0:10:2])
Hl ud

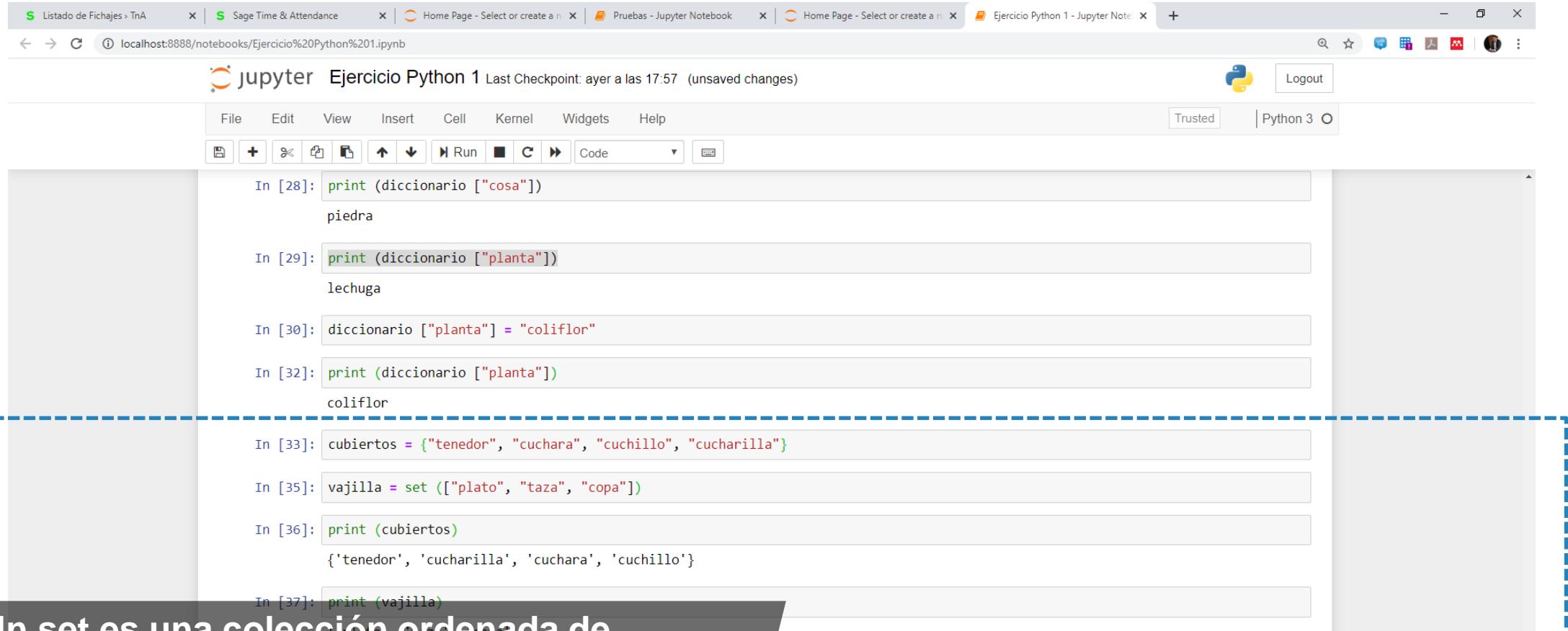
In [23]: diccionario = {"animal": "gato", "cosa": "piedra", "planta": "lechuga"}
In [24]: print (diccionario)
{'animal': 'gato', 'cosa': 'piedra', 'planta': 'lechuga'}
In [25]: print (diccionario ["animal"])
gato
In [28]: print (diccionario ["cosa"])
piedra
In [29]: print (diccionario ["planta"])
lechuga
In [30]: diccionario ["planta"] = "coliflor"
In [31]: print (diccionario ["planta"])
coliflor
```

A blue dashed rectangular box highlights the code from In [23] to In [31]. A dark grey overlay at the bottom left contains the text:

Un diccionario es un conjunto de valores emparejados.

Comenzando con Python

SECUENCIAS Y ESTRUCTURAS DE DATOS



The screenshot shows a Jupyter Notebook interface with the title "Ejercicio Python 1". The notebook contains the following code:

```
In [28]: print (diccionario ["cosa"])
piedra

In [29]: print (diccionario ["planta"])
lechuga

In [30]: diccionario ["planta"] = "coliflor"

In [32]: print (diccionario ["planta"])
coliflor

In [33]: cubiertos = {"tenedor", "cuchara", "cuchillo", "cucharilla"}

In [35]: vajilla = set ("plato", "taza", "copa")

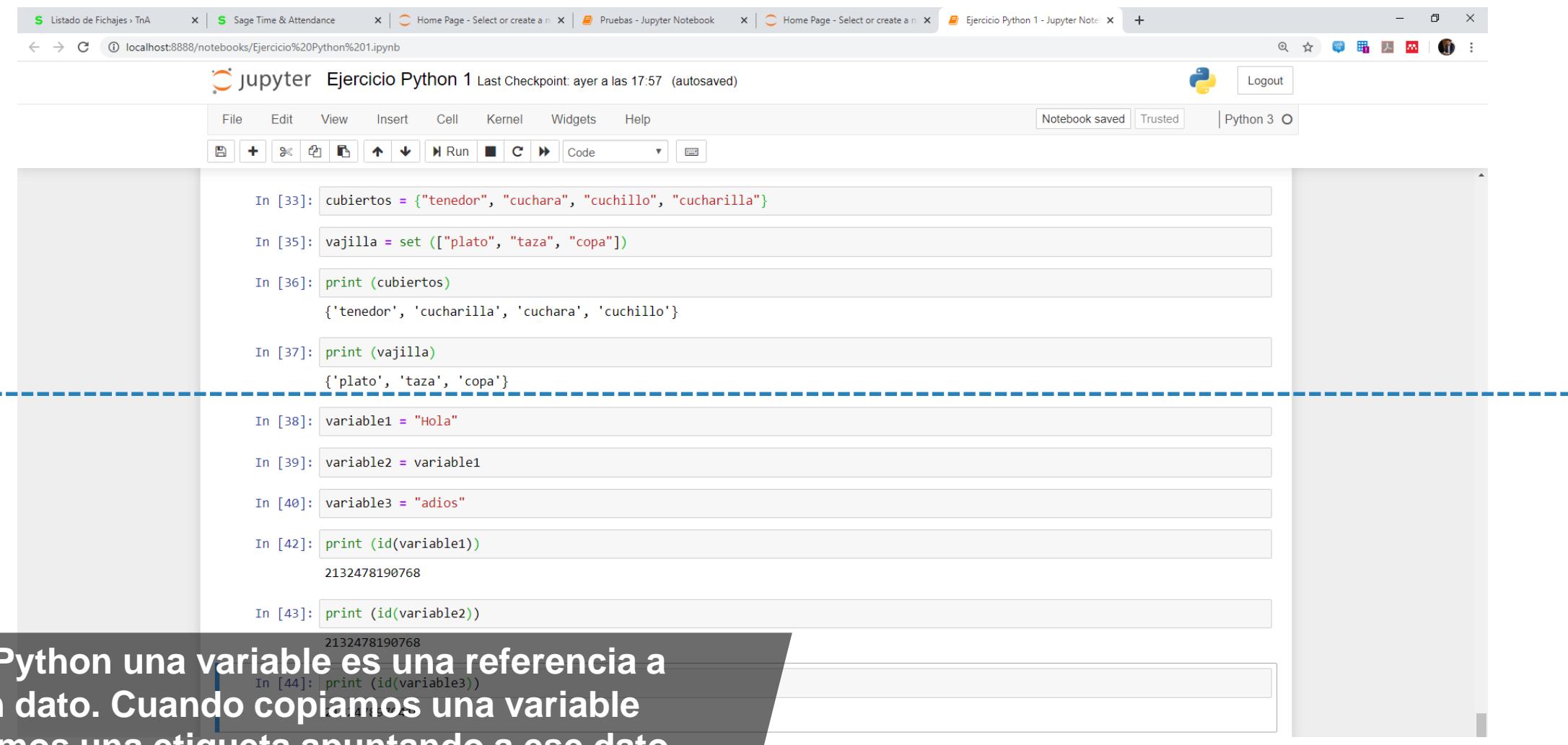
In [36]: print (cubiertos)
{'tenedor', 'cucharilla', 'cuchara', 'cuchillo'}

In [37]: print (vajilla)
```

Un set es una colección ordenada de objetos, no posee índice y no puede contener objetos repetidos.

Comenzando con Python

VARIABLES



The screenshot shows a Jupyter Notebook interface with the title "jupyter Ejercicio Python 1" and the message "Last Checkpoint: ayer a las 17:57 (autosaved)". The notebook has a Python 3 kernel and is marked as "Notebook saved" and "Trusted". The code cells are numbered [33] through [44]. Cells [33] to [37] demonstrate creating sets and printing their elements. Cells [38] to [42] demonstrate variable assignment and printing their memory addresses. Cell [43] shows the result of printing the memory address of variable2. Cell [44] is partially visible at the bottom.

```
In [33]: cubiertos = {"tenedor", "cuchara", "cuchillo", "cucharilla"}  
In [35]: vajilla = set(["plato", "taza", "copa"])  
In [36]: print (cubiertos)  
{'tenedor', 'cucharilla', 'cuchara', 'cuchillo'}  
In [37]: print (vajilla)  
{'plato', 'taza', 'copa'}  
In [38]: variable1 = "Hola"  
In [39]: variable2 = variable1  
In [40]: variable3 = "adios"  
In [42]: print (id(variable1))  
2132478190768  
In [43]: print (id(variable2))  
2132478190768  
In [44]: print (id(variable3))
```

En Python una variable es una referencia a un dato. Cuando copiamos una variable creamos una etiqueta apuntando a ese dato.

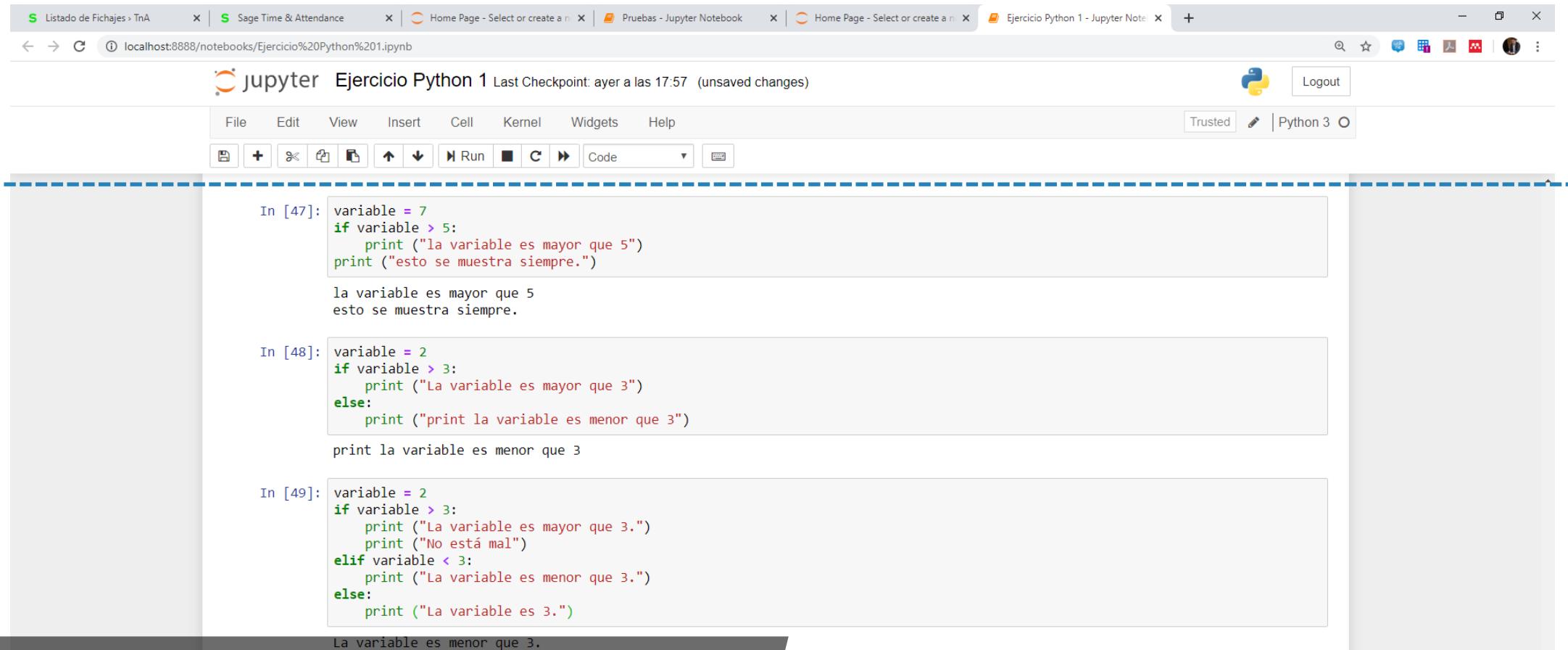
Índice

- Contexto
- Trabajando con Datos
- Estructuras de Control
- Funciones
- Ejercicios

ESTRUCTURAS DE CONTROL EN PYTHON

Comenzando con Python

ESTRUCTURAS DE CONTROL



The screenshot shows a Jupyter Notebook interface with the title "Ejercicio Python 1". The notebook contains three code cells:

```
In [47]: variable = 7
if variable > 5:
    print ("la variable es mayor que 5")
print ("esto se muestra siempre.")

la variable es mayor que 5
esto se muestra siempre.

In [48]: variable = 2
if variable > 3:
    print ("La variable es mayor que 3")
else:
    print ("La variable es menor que 3")

print la variable es menor que 3

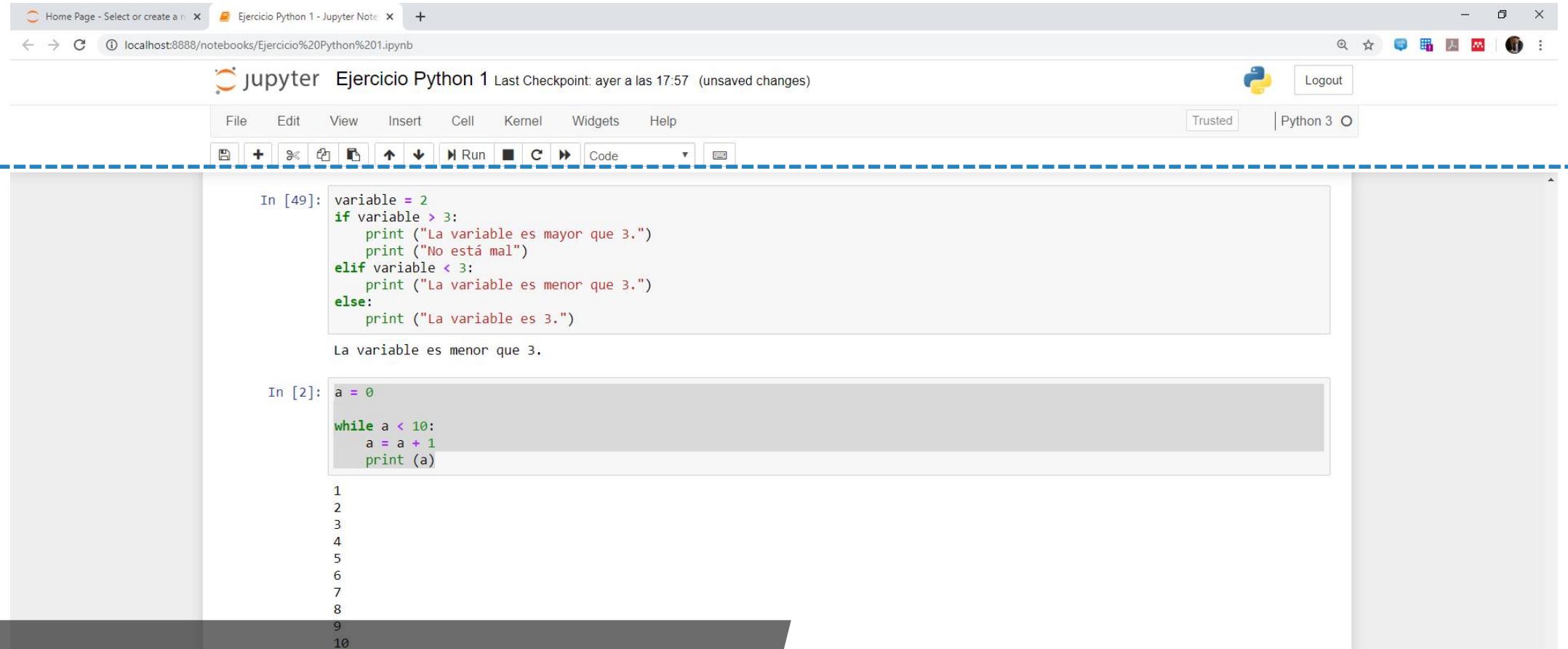
In [49]: variable = 2
if variable > 3:
    print ("La variable es mayor que 3.")
    print ("No está mal")
elif variable < 3:
    print ("La variable es menor que 3.")
else:
    print ("La variable es 3.")

La variable es menor que 3.
```

La estructura de control por excelencia es la sentencia IF.

Comenzando con Python

ESTRUCTURAS DE CONTROL



```
In [49]: variable = 2
if variable > 3:
    print ("La variable es mayor que 3.")
    print ("No está mal")
elif variable < 3:
    print ("La variable es menor que 3.")
else:
    print ("La variable es 3.")

La variable es menor que 3.

In [2]: a = 0

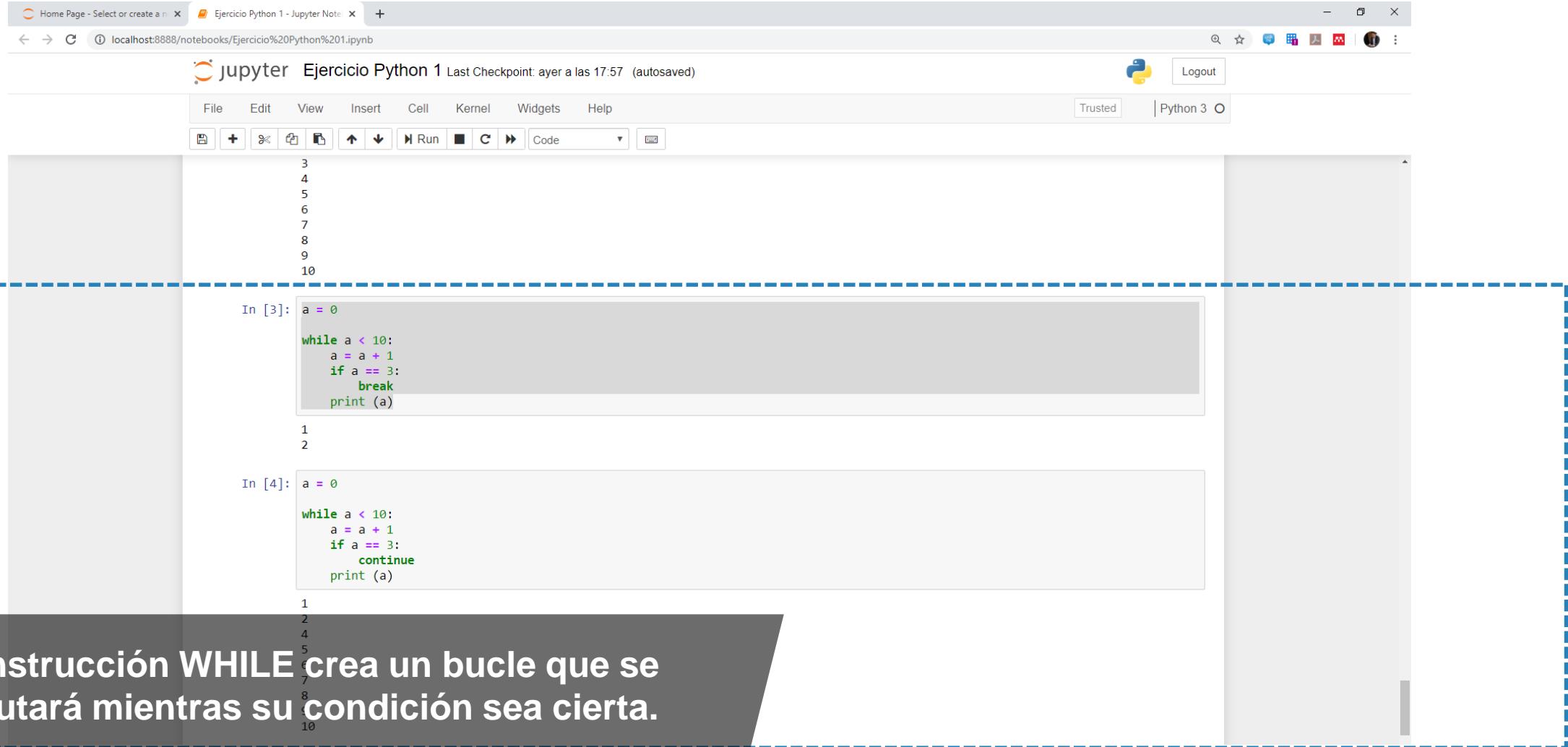
while a < 10:
    a = a + 1
    print (a)

1
2
3
4
5
6
7
8
9
10
```

La instrucción WHILE crea un bucle que se ejecutará mientras su condición sea cierta.

Comenzando con Python

ESTRUCTURAS DE CONTROL



The screenshot shows a Jupyter Notebook interface with two code cells. The first cell (In [3]) contains a while loop that prints integers from 3 to 10. The second cell (In [4]) contains a similar while loop but includes a continue statement for the value 3, so it only prints 1 and 2.

```
3
4
5
6
7
8
9
10

In [3]: a = 0
while a < 10:
    a = a + 1
    if a == 3:
        break
    print (a)

1
2

In [4]: a = 0
while a < 10:
    a = a + 1
    if a == 3:
        continue
    print (a)
```

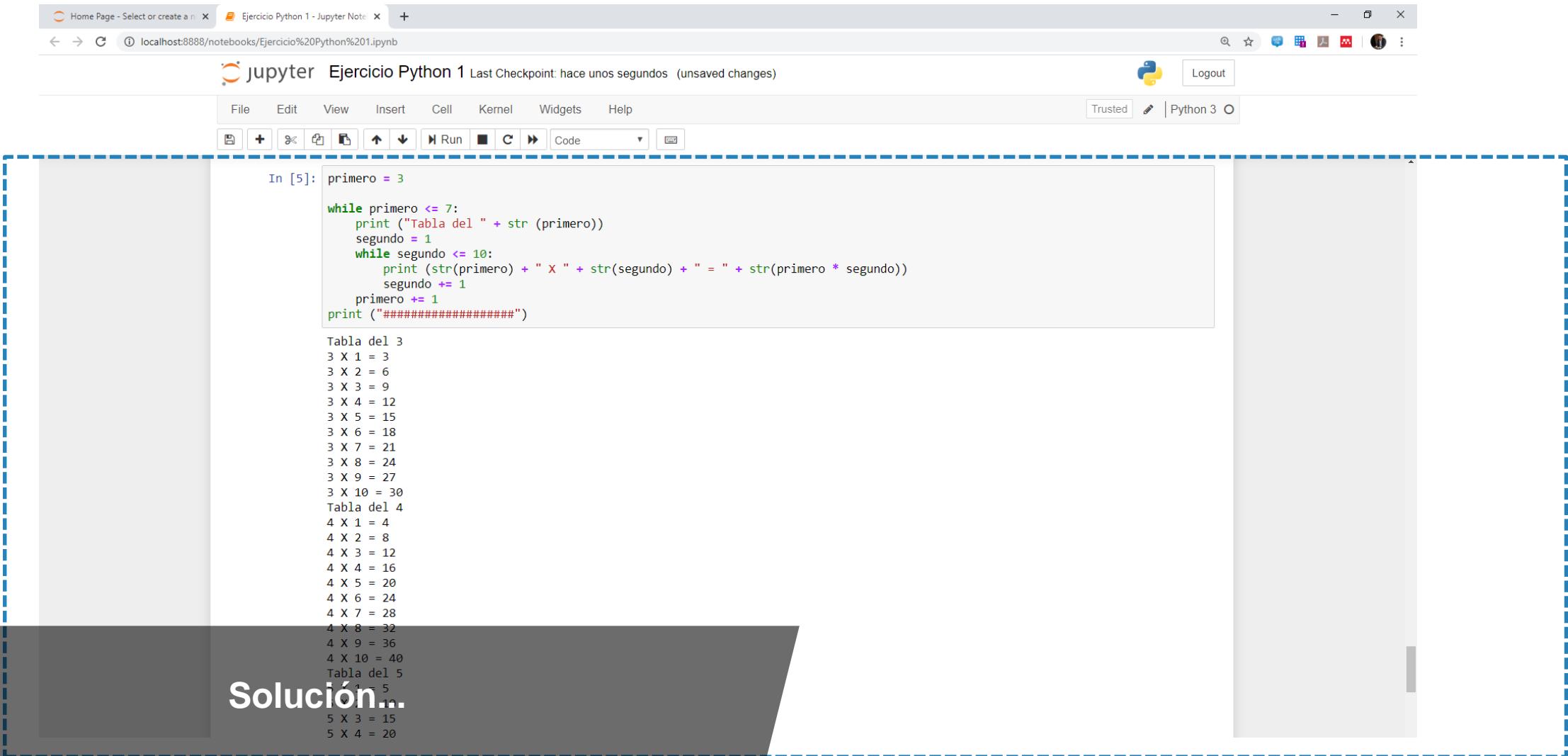
La instrucción WHILE crea un bucle que se ejecutará mientras su condición sea cierta.

Requisitos

ESCRIBIR UN PEQUEÑO PROGRAMA QUE IMPRIMA LAS TABLAS DE MULTIPLICAR DEL TRES AL SIETE

Comenzando con Python

ESTRUCTURAS DE CONTROL



The screenshot shows a Jupyter Notebook interface with the title "Ejercicio Python 1 - Jupyter Notebooks". The notebook contains a single code cell (In [5]) with the following Python code:

```
In [5]: primero = 3

while primero <= 7:
    print ("Tabla del " + str (primero))
    segundo = 1
    while segundo <= 10:
        print (str(primero) + " X " + str(segundo) + " = " + str(primero * segundo))
        segundo += 1
    primero += 1
print ("#####")
```

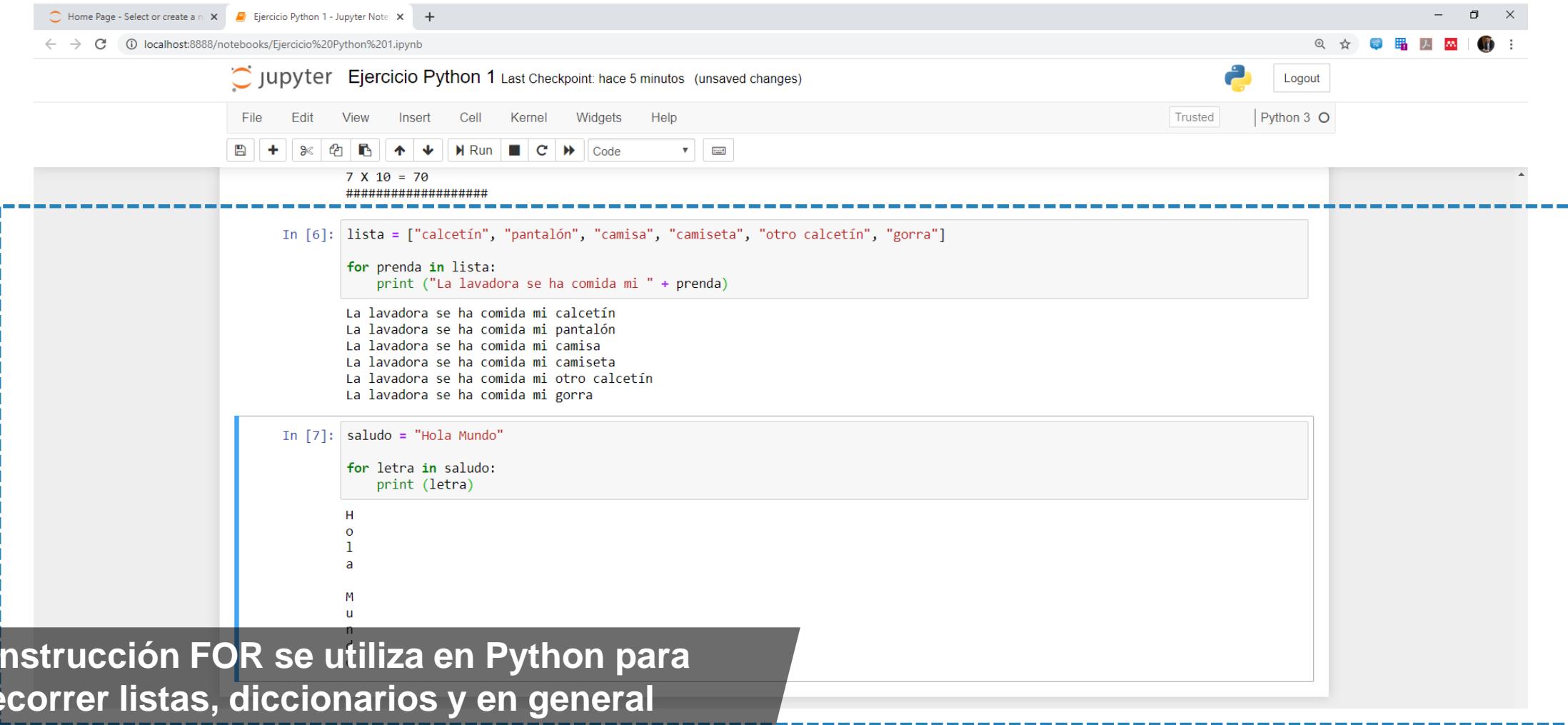
The output of this code is displayed below the cell, showing the multiplication tables for numbers 3, 4, and 5.

Solución...

```
Tabla del 3
3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
3 X 4 = 12
3 X 5 = 15
3 X 6 = 18
3 X 7 = 21
3 X 8 = 24
3 X 9 = 27
3 X 10 = 30
Tabla del 4
4 X 1 = 4
4 X 2 = 8
4 X 3 = 12
4 X 4 = 16
4 X 5 = 20
4 X 6 = 24
4 X 7 = 28
4 X 8 = 32
4 X 9 = 36
4 X 10 = 40
Tabla del 5
5 X 1 = 5
5 X 2 = 10
5 X 3 = 15
5 X 4 = 20
```

Comenzando con Python

ESTRUCTURAS DE CONTROL



```
7 X 10 = 70
#####
In [6]: lista = ["calcetín", "pantalón", "camisa", "camiseta", "otro calcetín", "gorra"]
for prenda in lista:
    print ("La lavadora se ha comida mi " + prenda)

La lavadora se ha comida mi calcetín
La lavadora se ha comida mi pantalón
La lavadora se ha comida mi camisa
La lavadora se ha comida mi camiseta
La lavadora se ha comida mi otro calcetín
La lavadora se ha comida mi gorra

In [7]: saludo = "Hola Mundo"
for letra in saludo:
    print (letra)

H
o
l
a

M
u
n
```

La instrucción FOR se utiliza en Python para recorrer listas, diccionarios y en general iteradores.

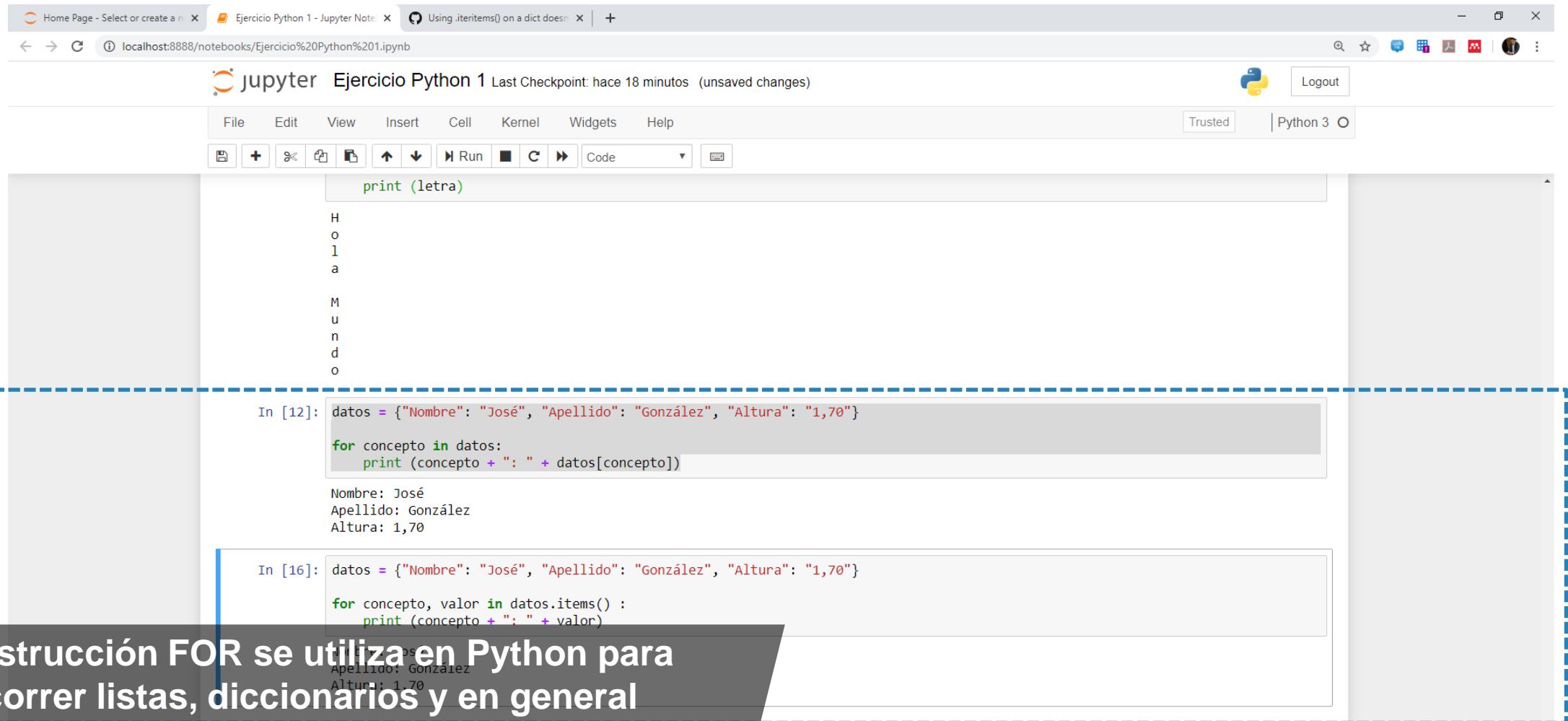
Requisitos

UTILIZAR FOR PARA RECORRER EL SIGUIENTE DICCIONARIO:

```
datos = {"Nombre": "José", "Apellido": "González", "Altura": "1,70"}
```

Comenzando con Python

ESTRUCTURAS DE CONTROL



The screenshot shows a Jupyter Notebook interface with three tabs at the top: "Home Page - Select or create a notebook", "Ejercicio Python 1 - Jupyter Notebook", and "Using iteritems() on a dict does not work". The main area displays the following code and output:

```
print (letra)
H
o
l
a

M
u
n
d
o
```

In [12]:

```
datos = {"Nombre": "José", "Apellido": "González", "Altura": "1,70"}  
for concepto in datos:  
    print (concepto + ": " + datos[concepto])
```

Nombre: José
Apellido: González
Altura: 1,70

In [16]:

```
datos = {"Nombre": "José", "Apellido": "González", "Altura": "1,70"}  
for concepto, valor in datos.items():  
    print (concepto + ": " + valor)
```

Apellido: González
Altura: 1,70

A blue dashed box highlights the code in In [12] and its output. A black box highlights the code in In [16] and its output.

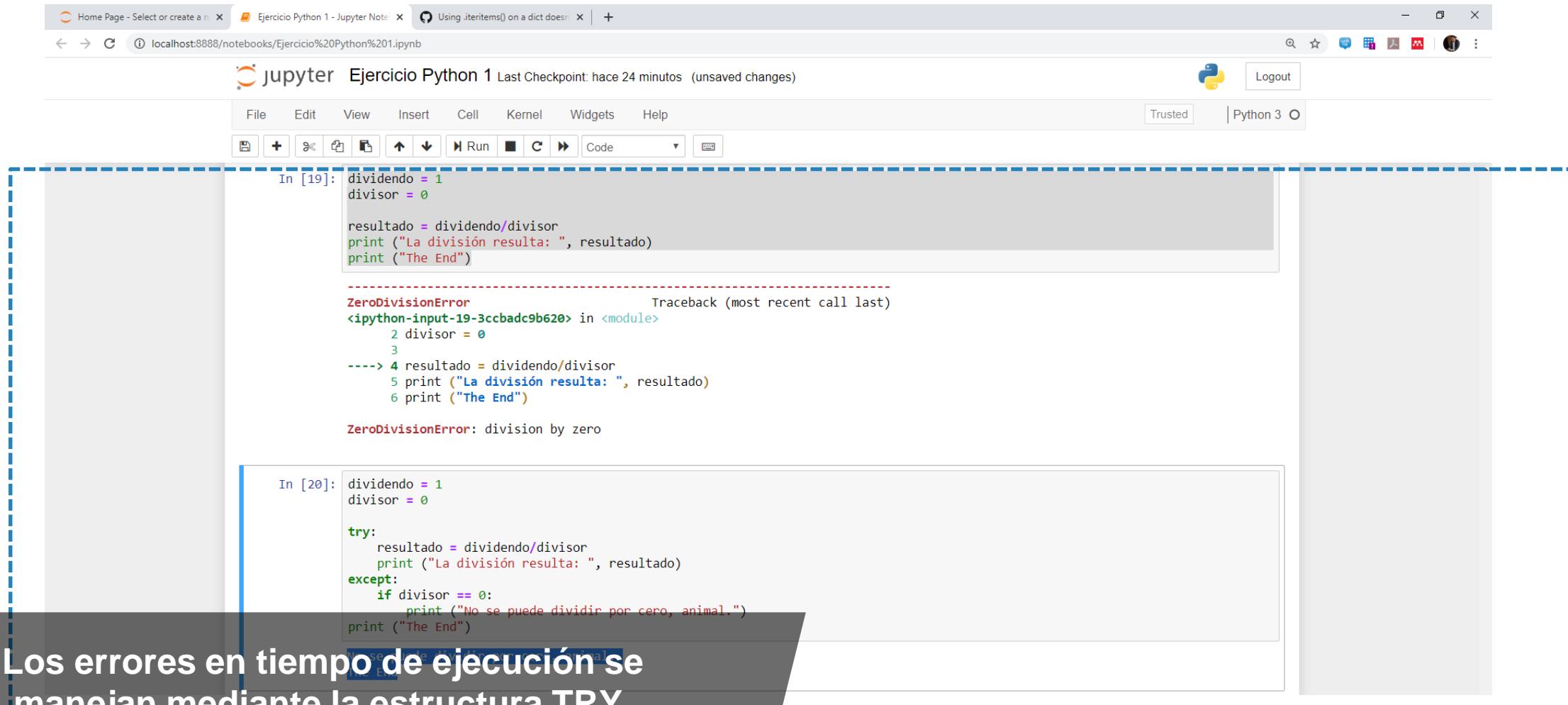
La instrucción FOR se utiliza en Python para recorrer listas, diccionarios y en general iteradores.

Ejercicio

**ESCRIBIR UN PEQUEÑO PROGRAMA QUE
DIVIDA UN NÚMERO POR CERO. VER QUÉ
PASA...**

Comenzando con Python

ESTRUCTURAS DE CONTROL



The screenshot shows a Jupyter Notebook interface with two code cells. Cell [19] contains a simple division operation that results in a ZeroDivisionError. Cell [20] demonstrates exception handling to manage this error.

```
In [19]: dividendo = 1
divisor = 0

resultado = dividendo/divisor
print ("La división resulta: ", resultado)
print ("The End")

-----  

ZeroDivisionError: division by zero
```

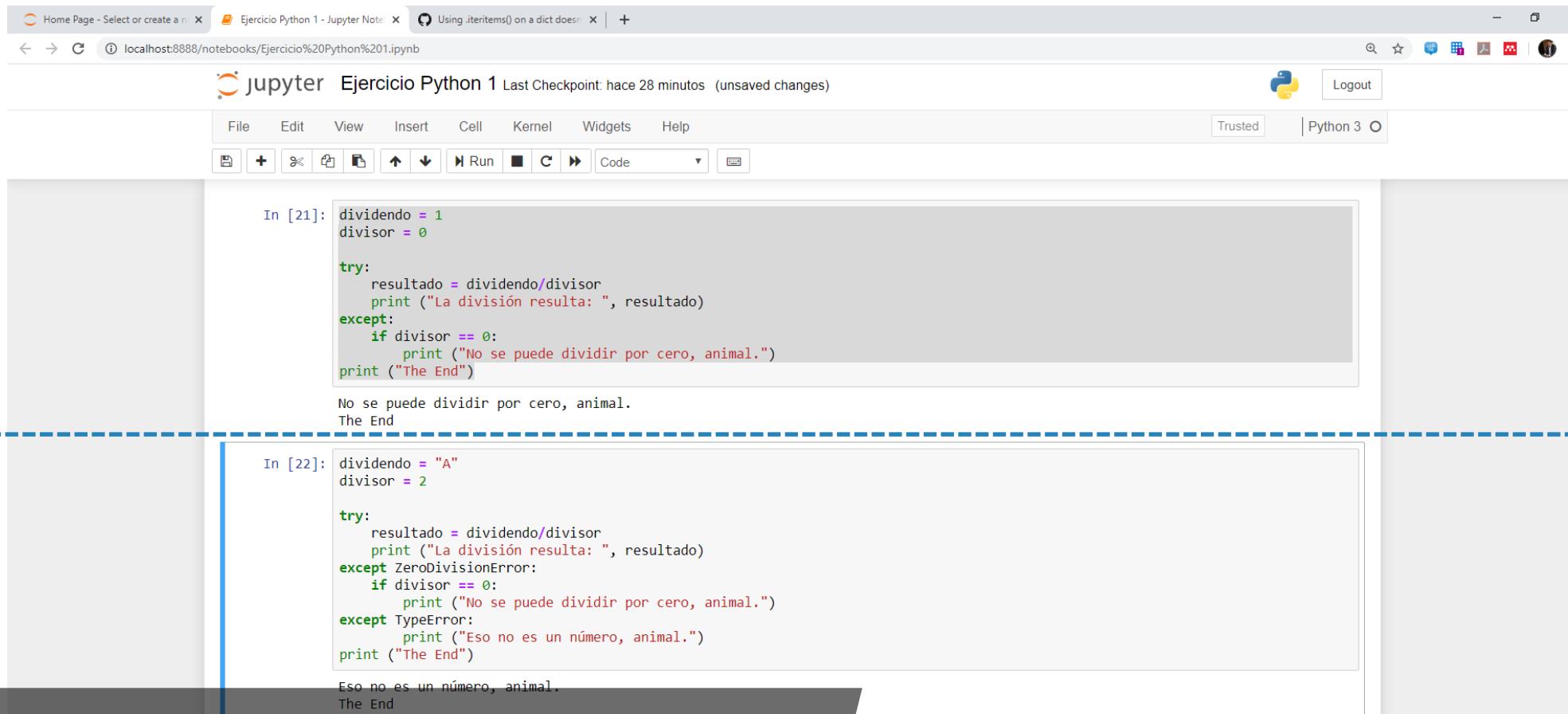
```
In [20]: dividendo = 1
divisor = 0

try:
    resultado = dividendo/divisor
    print ("La división resulta: ", resultado)
except:
    if divisor == 0:
        print ("No se puede dividir por cero, animal.")
print ("The End")
```

Los errores en tiempo de ejecución se manejan mediante la estructura TRY.

Comenzando con Python

ESTRUCTURAS DE CONTROL



```
In [21]: dividendo = 1
divisor = 0

try:
    resultado = dividendo/divisor
    print ("La división resulta: ", resultado)
except:
    if divisor == 0:
        print ("No se puede dividir por cero, animal.")
print ("The End")

No se puede dividir por cero, animal.
The End
```

```
In [22]: dividendo = "A"
divisor = 2

try:
    resultado = dividendo/divisor
    print ("La división resulta: ", resultado)
except ZeroDivisionError:
    if divisor == 0:
        print ("No se puede dividir por cero, animal.")
except TypeError:
    print ("Eso no es un número, animal.")
print ("The End")

Eso no es un número, animal.
The End
```

Los errores en tiempo de ejecución se manejan mediante la estructura TRY.

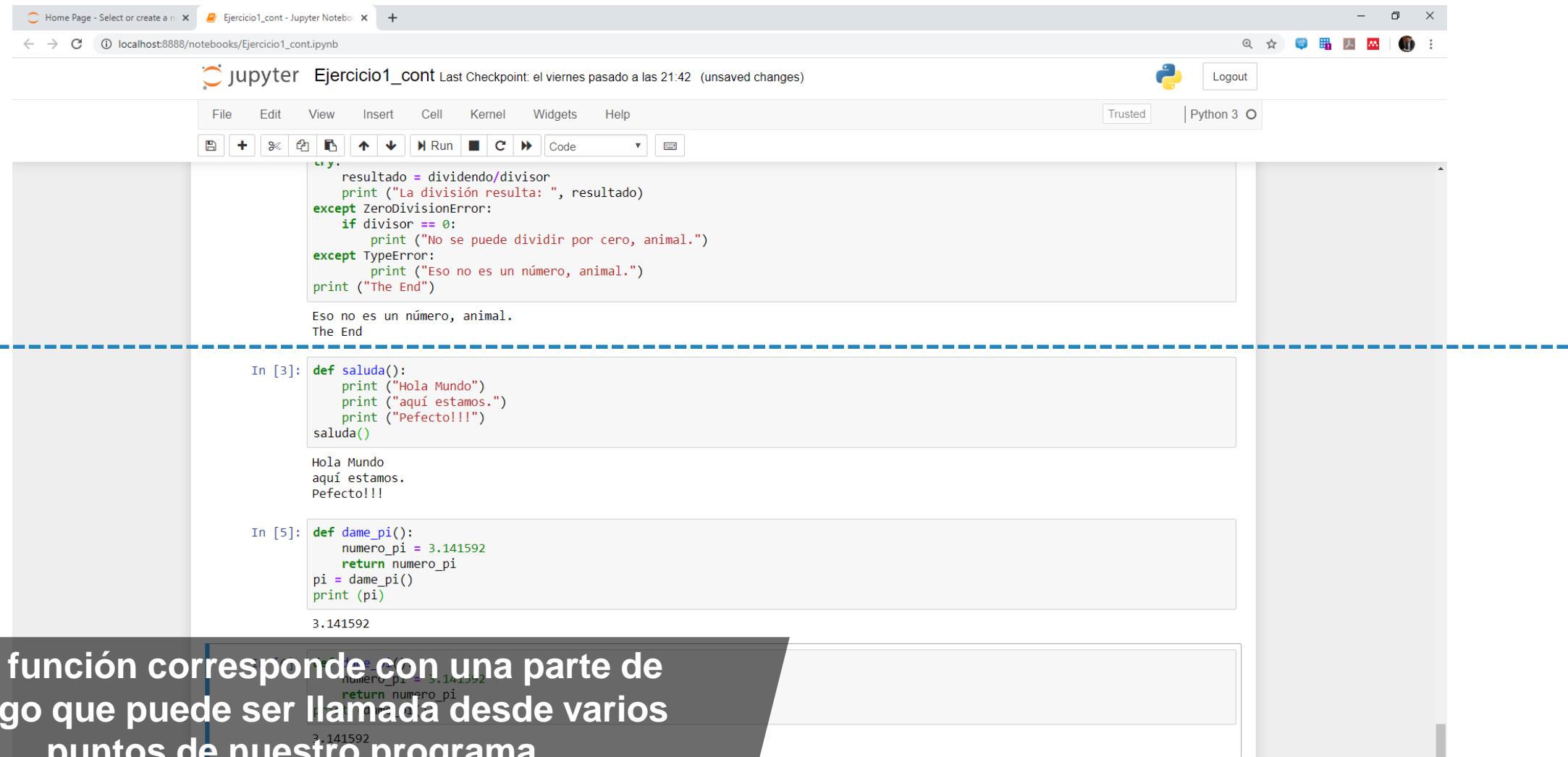
Índice

- Contexto
- Trabajando con Datos
- Estructuras de Control
- Funciones
- Ejercicios

FUNCIONES EN PYTHON

Comenzando con Python

ESTRUCTURAS DE CONTROL



The screenshot shows a Jupyter Notebook interface with the title "Ejercicio1_cont - Jupyter Notebook". The notebook contains several cells of Python code:

```
y
    resultado = dividendo/divisor
    print ("La división resulta: ", resultado)
except ZeroDivisionError:
    if divisor == 0:
        print ("No se puede dividir por cero, animal.")
except TypeError:
    print ("Eso no es un número, animal.")
print ("The End")
```

Eso no es un número, animal.
The End

In [3]:

```
def saluda():
    print ("Hola Mundo")
    print ("aquí estamos.")
    print ("Pefecto!!!")
saluda()
```

Hola Mundo
aquí estamos.
Pefecto!!!

In [5]:

```
def dame_pi():
    numero_pi = 3.141592
    return numero_pi
pi = dame_pi()
print (pi)
```

3.141592

A callout box highlights the first code block with the text: "Una función corresponde con una parte de código que puede ser llamada desde varios puntos de nuestro programa."

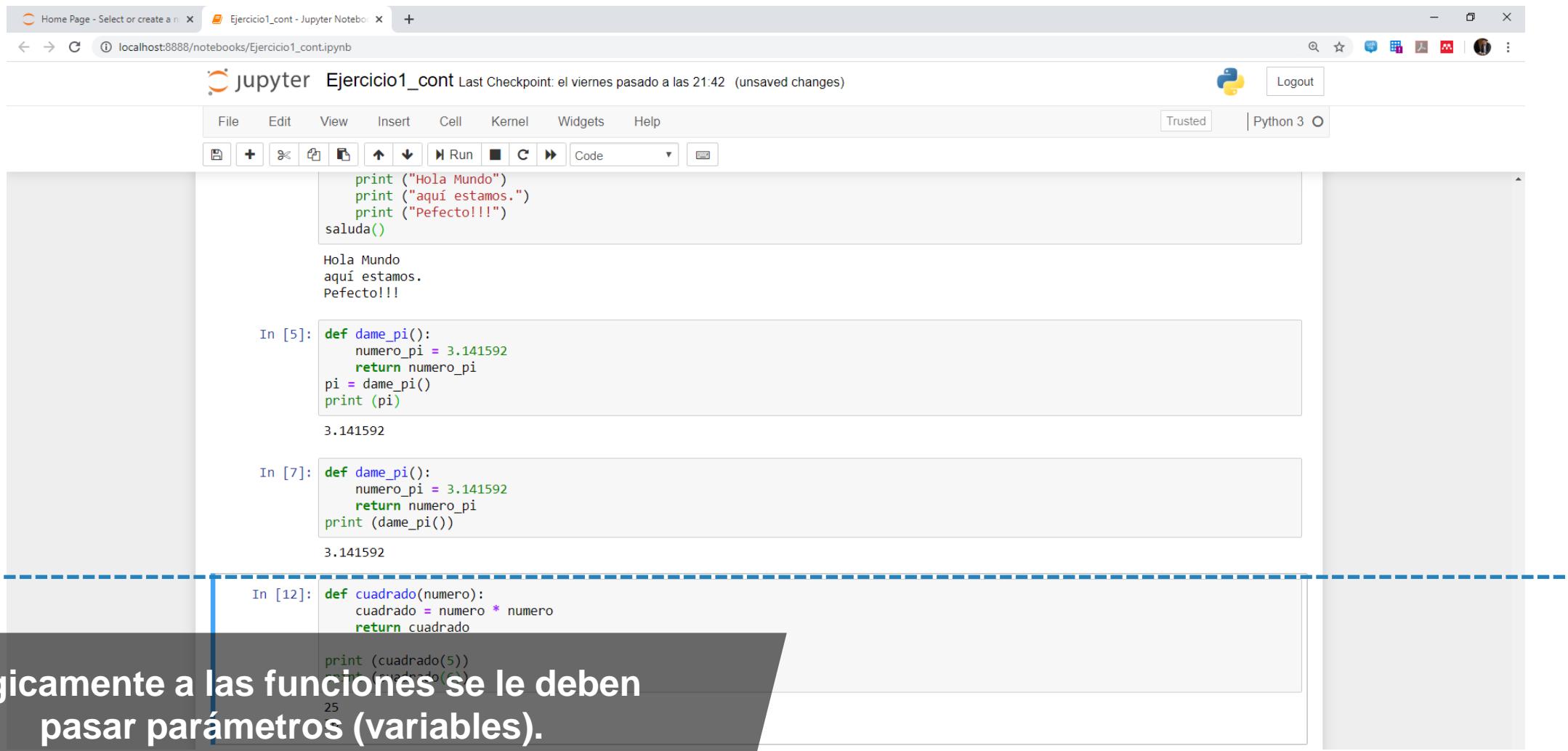
Una función corresponde con una parte de código que puede ser llamada desde varios puntos de nuestro programa.

Ejercicio

**ESCRIBIR UN PEQUEÑO PROGRAMA QUE
DADO UN NÚMERO, DEVUELVA EL
CUADRADO DE DICHO NÚMERO**

Comenzando con Python

ESTRUCTURAS DE CONTROL



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
print ("Hola Mundo")
print ("aqui estamos.")
print ("Pefecto!!!")
saluda()

Hola Mundo
aqui estamos.
Pefecto!!!

In [5]: def dame_pi():
    numero_pi = 3.141592
    return numero_pi
pi = dame_pi()
print (pi)

3.141592

In [7]: def dame_pi():
    numero_pi = 3.141592
    return numero_pi
print (dame_pi())

3.141592

In [12]: def cuadrado(numero):
    cuadrado = numero * numero
    return cuadrado

print (cuadrado(5))
25
```

A callout box with a dashed blue border highlights the last cell (In [12]) and contains the text:

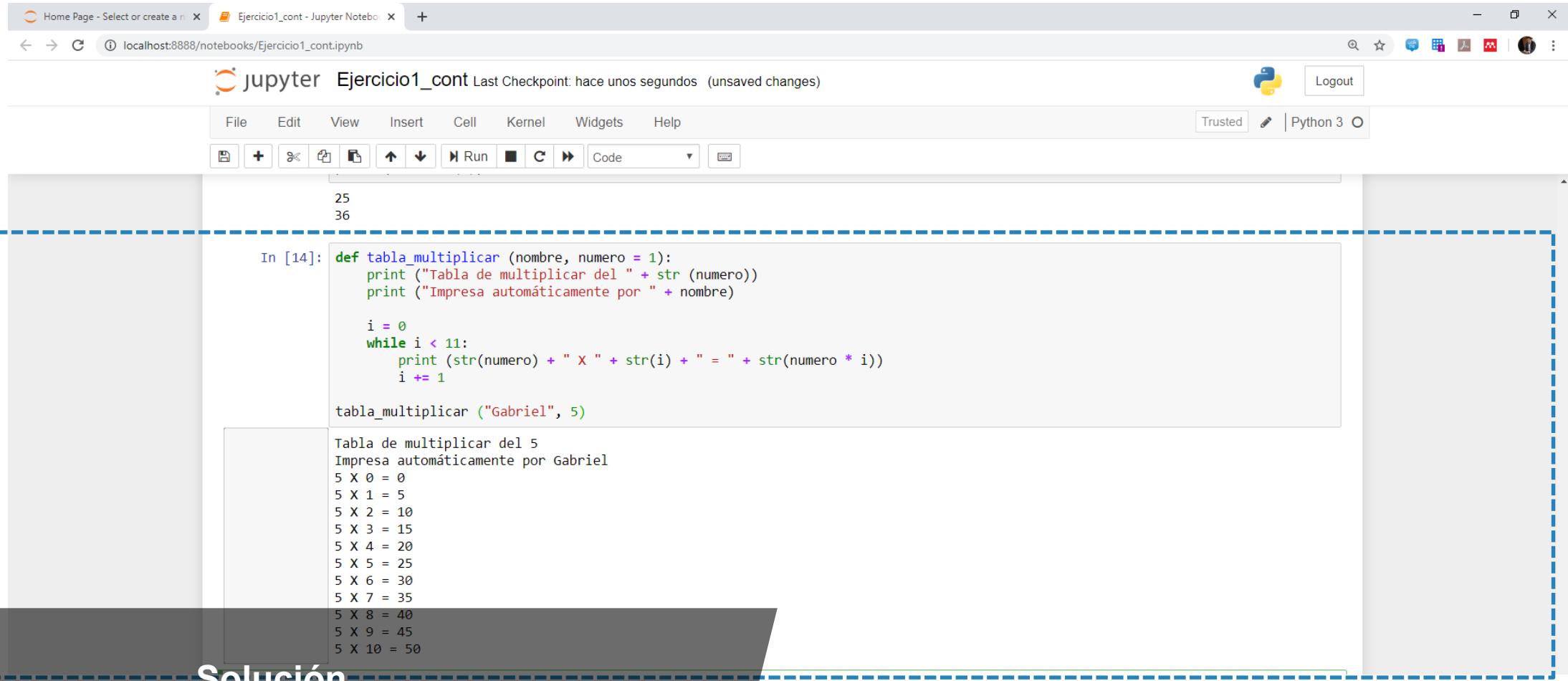
Lógicamente a las funciones se le deben pasar parámetros (variables).

Ejercicio

**ESCRIBIR UN PEQUEÑO PROGRAMA QUE
DADO UN NÚMERO (1 AL 10) DEVUELVA LA
TABLA DE MULTICAR DE DICHO NÚMERO**

Comenzando con Python

ESTRUCTURAS DE CONTROL



The screenshot shows a Jupyter Notebook interface with the title "Ejercicio1_cont". The code in cell [14] defines a function to print a multiplication table for a given number. The output shows the table for the number 5.

```
def tabla_multiplicar (nombre, numero = 1):
    print ("Tabla de multiplicar del " + str (numero))
    print ("Impresa automáticamente por " + nombre)

    i = 0
    while i < 11:
        print (str(numero) + " X " + str(i) + " = " + str(numero * i))
        i += 1

tabla_multiplicar ("Gabriel", 5)
```

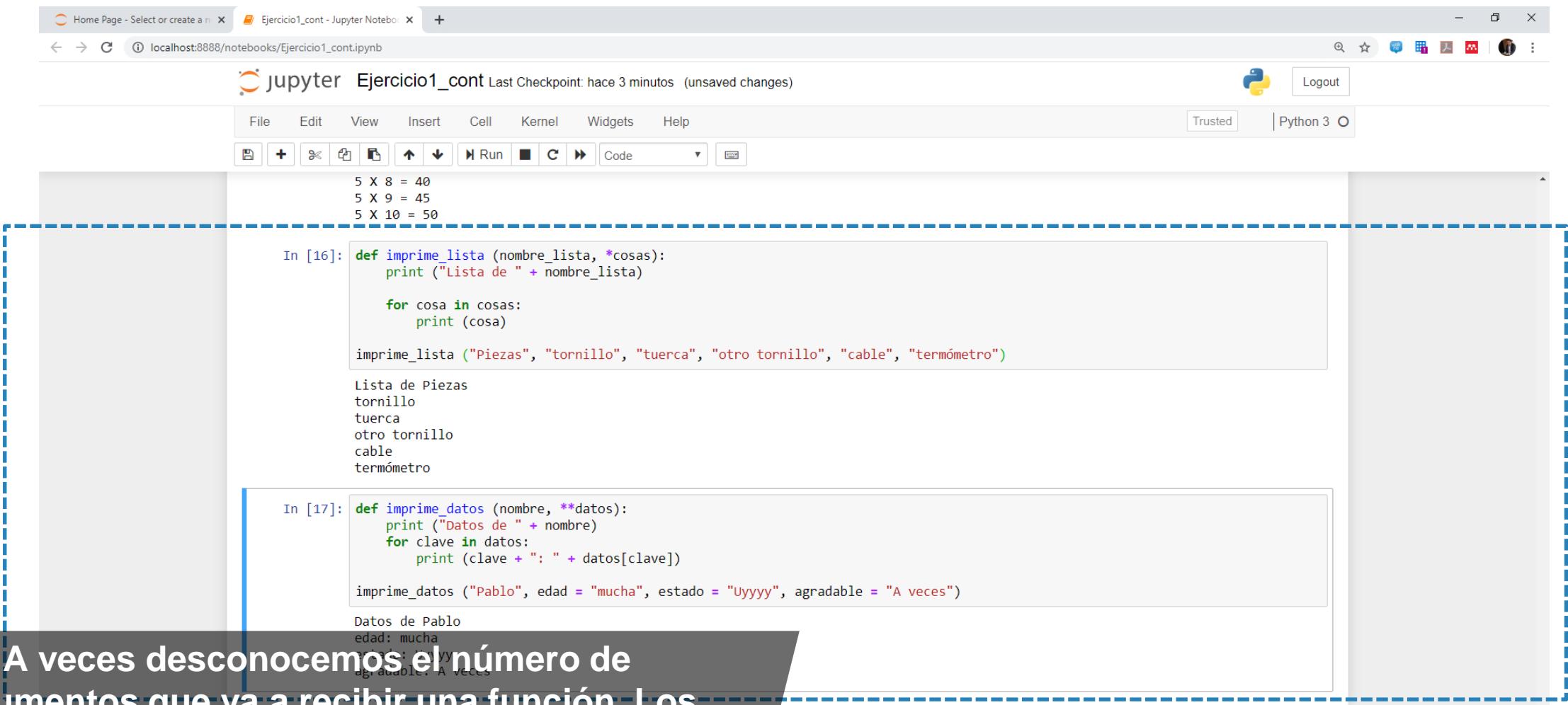
Output:

```
Tabla de multiplicar del 5
Impresa automáticamente por Gabriel
5 X 0 = 0
5 X 1 = 5
5 X 2 = 10
5 X 3 = 15
5 X 4 = 20
5 X 5 = 25
5 X 6 = 30
5 X 7 = 35
5 X 8 = 40
5 X 9 = 45
5 X 10 = 50
```

Solución...

Comenzando con Python

ESTRUCTURAS DE CONTROL



The screenshot shows a Jupyter Notebook interface with two code cells. The first cell (In [16]) defines a function `imprime_lista` that prints a list of items. It demonstrates unpacking a tuple into a list and using a for loop to print each item. The second cell (In [17]) defines a function `imprime_datos` that prints data from a dictionary. It shows how to use double asterisks to receive keyword arguments and access them by key.

```
5 X 8 = 40
5 X 9 = 45
5 X 10 = 50

In [16]: def imprime_lista (nombre_lista, *cosas):
    print ("Lista de " + nombre_lista)

    for cosa in cosas:
        print (cosa)

imprime_lista ("Piezas", "tornillo", "tuerca", "otro tornillo", "cable", "termómetro")

Lista de Piezas
tornillo
tuerca
otro tornillo
cable
termómetro

In [17]: def imprime_datos (nombre, **datos):
    print ("Datos de " + nombre)
    for clave in datos:
        print (clave + ": " + datos[clave])

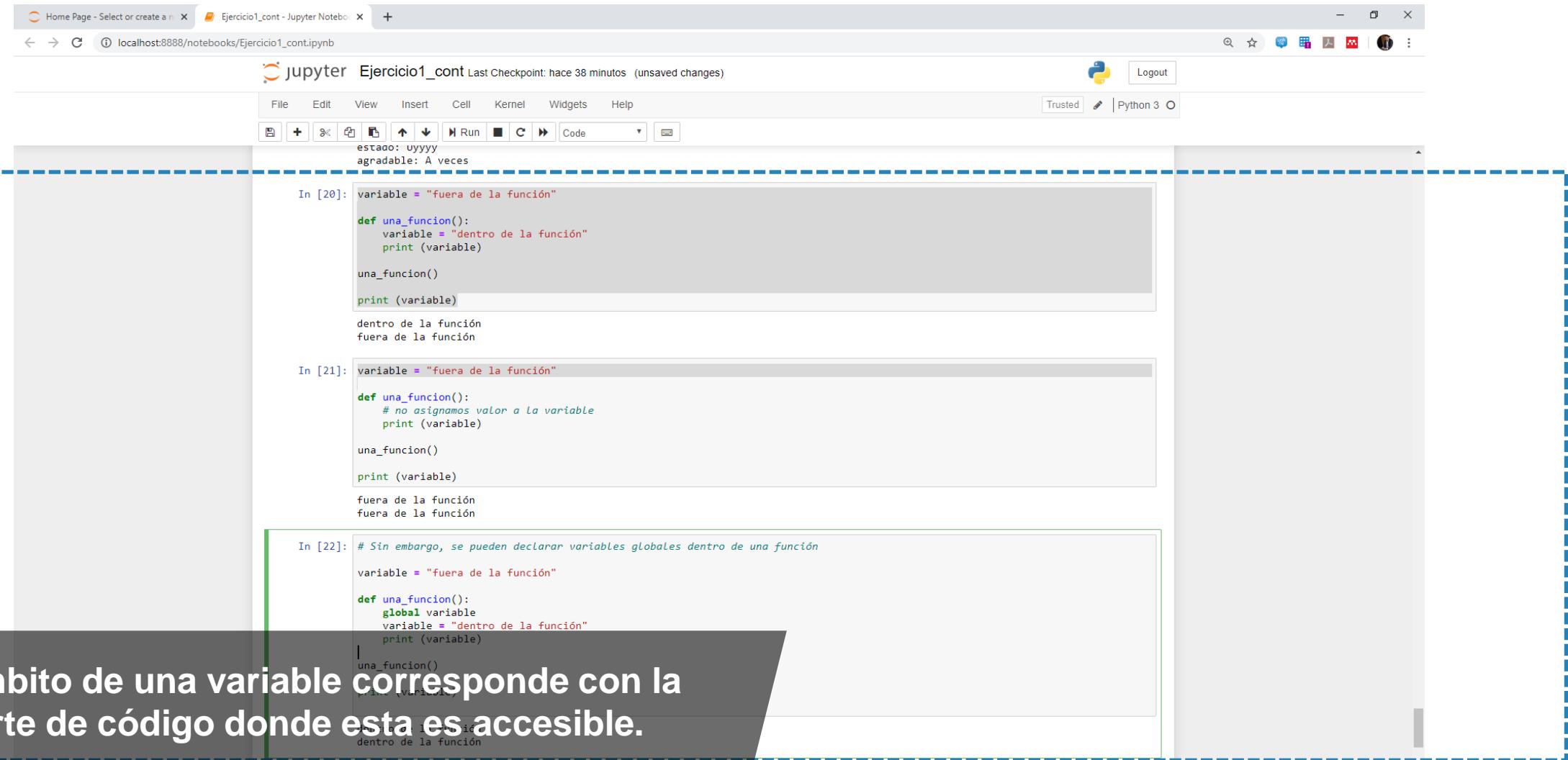
imprime_datos ("Pablo", edad = "mucha", estado = "Uyyyy", agradable = "A veces")

Datos de Pablo
edad: mucha
estado: Uyyyy
agradable: A veces
```

A veces desconocemos el número de argumentos que va a recibir una función. Los argumentos fijos, deben ir al principio.

Comenzando con Python

ESTRUCTURAS DE CONTROL



The screenshot shows a Jupyter Notebook interface with three code cells:

- In [20]:** A function "una_funcion" is defined. It prints the value of the variable "variable" which is assigned inside the function. The output shows the variable is "dentro de la función".
- In [21]:** The same function is defined again, but it does not assign a value to the variable "variable". The output shows the variable is "fuera de la función".
- In [22]:** The function is defined again, but this time it uses the "global" keyword to declare that the variable "variable" refers to the global variable. The output shows the variable is "dentro de la función".

A green box highlights the code in In [22] with the caption: "El ámbito de una variable corresponde con la parte de código donde esta es accesible."

```
estado: uyyyyy
agradable: A veces

In [20]: variable = "fuera de la función"

def una_funcion():
    variable = "dentro de la función"
    print (variable)

una_funcion()
print (variable)

dentro de la función
fuera de la función

In [21]: variable = "fuera de la función"

def una_funcion():
    # no asignamos valor a la variable
    print (variable)

una_funcion()
print (variable)

fuera de la función
fuera de la función

In [22]: # Sin embargo, se pueden declarar variables globales dentro de una función

variable = "fuera de la función"

def una_funcion():
    global variable
    variable = "dentro de la función"
    print (variable)

una_funcion()
print(variable)

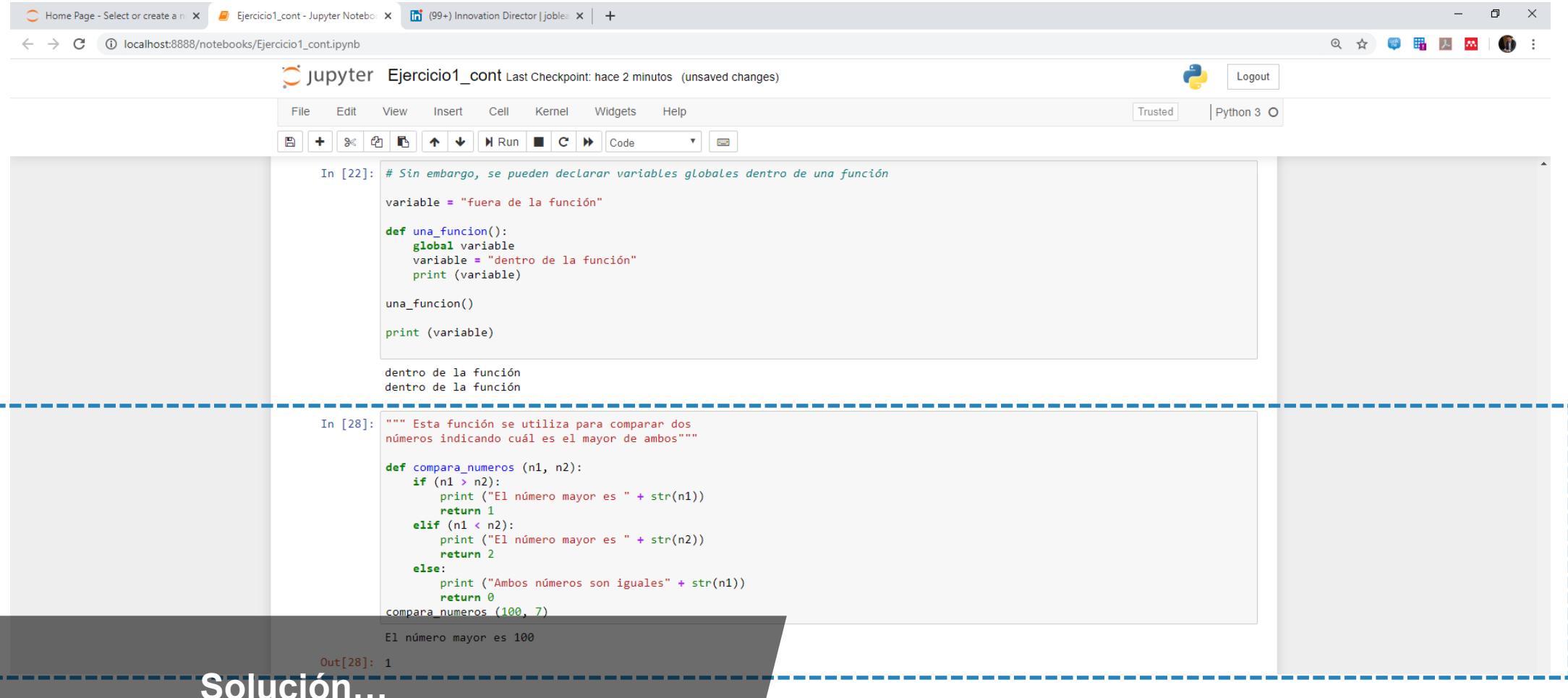
dentro de la función
```

Ejercicio

**ESCRIBIR UN PEQUEÑO PROGRAMA QUE
RECIBA COMO PARÁMETROS DOS
NÚMEROS Y DEVUELVA CUAL ES EL MAYOR**

Comenzando con Python

ESTRUCTURAS DE CONTROL



In [22]: *# Sin embargo, se pueden declarar variables globales dentro de una función*

```
variable = "fuera de la función"

def una_funcion():
    global variable
    variable = "dentro de la función"
    print (variable)

una_funcion()

print (variable)
```

dentro de la función
dentro de la función

In [28]: *"" Esta función se utiliza para comparar dos números indicando cuál es el mayor de ambos""*

```
def compara_numeros (n1, n2):
    if (n1 > n2):
        print ("El número mayor es " + str(n1))
        return 1
    elif (n1 < n2):
        print ("El número mayor es " + str(n2))
        return 2
    else:
        print ("Ambos números son iguales" + str(n1))
        return 0
compara_numeros (100, 7)
```

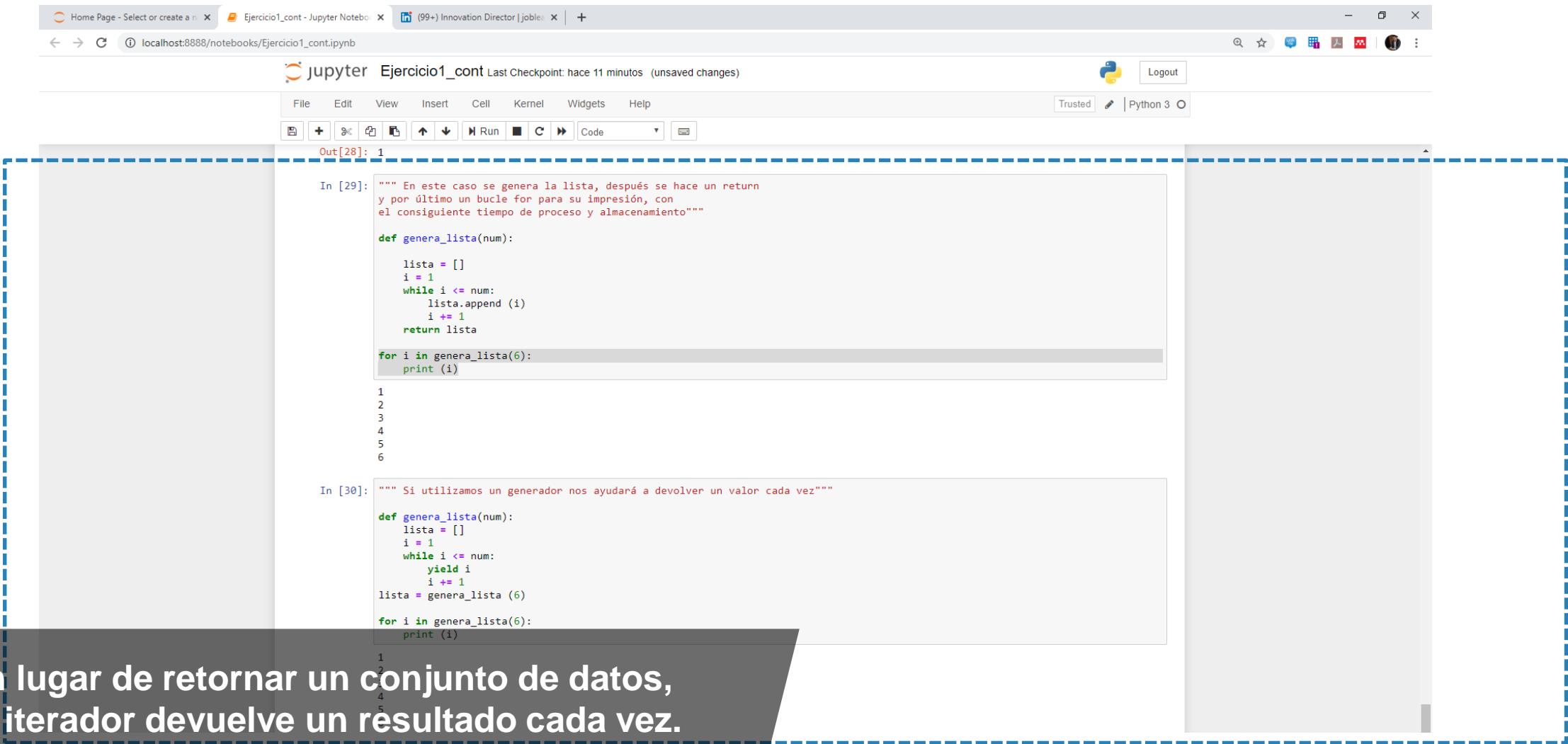
El número mayor es 100

Out[28]: 1

Solución...

Comenzando con Python

ESTRUCTURAS DE CONTROL



The screenshot shows a Jupyter Notebook interface with two code cells. The first cell, In [29], contains a function definition for generating a list from 1 to n using a while loop, followed by a for loop that prints each value. The output, Out [28], shows the numbers 1 through 6. The second cell, In [30], contains a function definition for generating a list using a generator expression, followed by a for loop that prints each value. The output, Out [29], shows the numbers 1 through 6.

```
In [29]: """ En este caso se genera la lista, después se hace un return y por último un bucle for para su impresión, con el consiguiente tiempo de proceso y almacenamiento"""

def genera_lista(num):

    lista = []
    i = 1
    while i <= num:
        lista.append (i)
        i += 1
    return lista

for i in genera_lista(6):
    print (i)

Out[28]: 1
2
3
4
5
6

In [30]: """ Si utilizamos un generador nos ayudará a devolver un valor cada vez"""

def genera_lista(num):
    lista = []
    i = 1
    while i <= num:
        yield i
        i += 1
    lista = genera_lista (6)

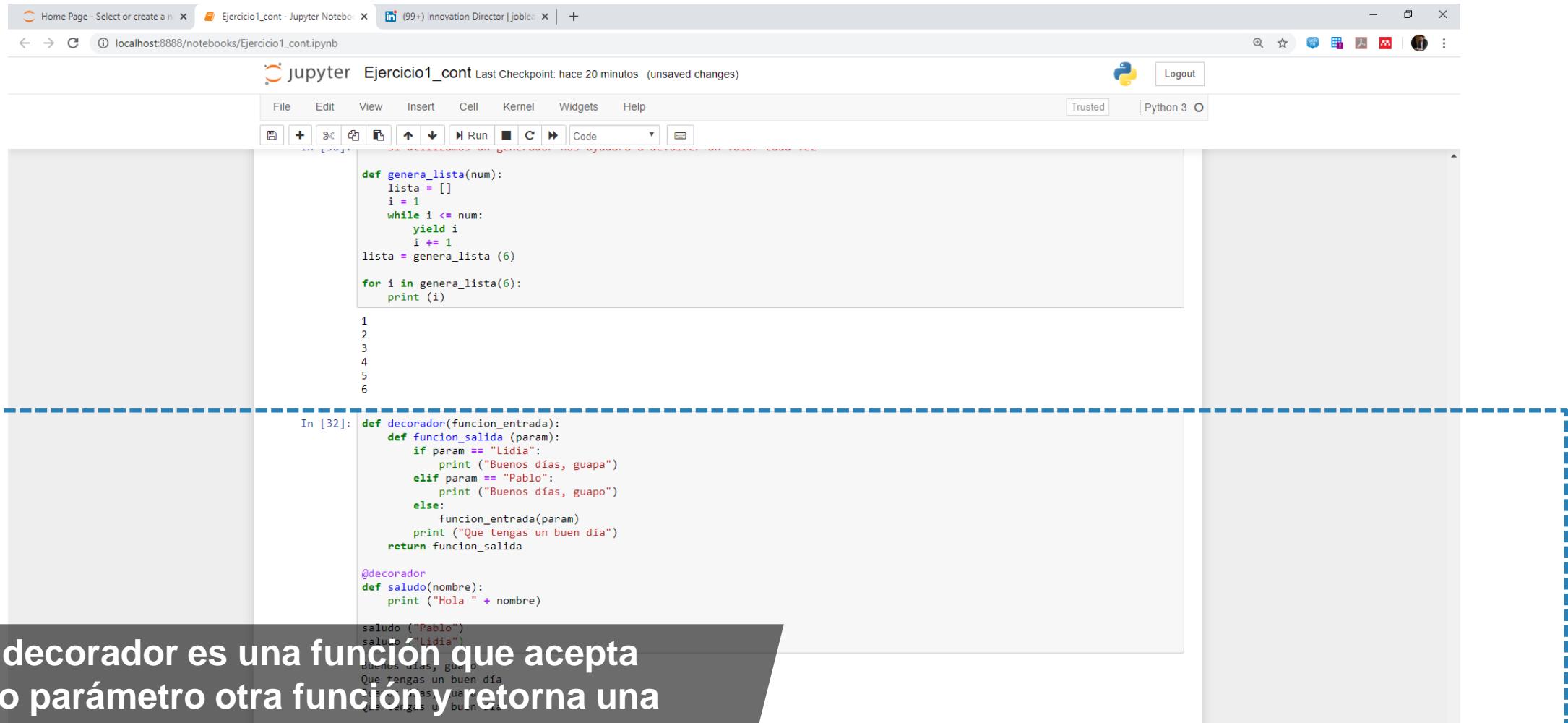
for i in genera_lista(6):
    print (i)

Out[29]: 1
2
3
4
5
6
```

En lugar de retornar un conjunto de datos, un iterador devuelve un resultado cada vez.

Comenzando con Python

ESTRUCTURAS DE CONTROL



```
def genera_lista(num):
    lista = []
    i = 1
    while i <= num:
        yield i
        i += 1
lista = genera_lista(6)

for i in genera_lista(6):
    print(i)

1
2
3
4
5
6
```

```
In [32]: def decorador(funcion_entrada):
    def funcion_salida(param):
        if param == "Lidia":
            print ("Buenos días, guapa")
        elif param == "Pablo":
            print ("Buenos días, guapo")
        else:
            funcion_entrada(param)
            print ("Que tengas un buen día")
    return funcion_salida

@decorador
def saludo(nombre):
    print ("Hola " + nombre)

saludo ("Pablo")
saludo ("Lidia")
```

Que tengas un buen día

Un decorador es una función que acepta como parámetro otra función y retorna una tercera función.

Índice

- Contexto
- Trabajando con Datos
- Estructuras de Control
- Funciones
- Ejercicios

EJERCICIOS

Ejercicios

1. Revisar que son las funciones Lambda y poner un ejemplo.
2. Definir una función `max_de_tres()`, que tome tres números como argumentos y devuelva el mayor de ellos.
3. Escribir una función que tome un carácter y devuelva True si es una vocal, de lo contrario devuelve False.
4. Escribir una función `sum()` y una función `multip()` que sumen y multipliquen respectivamente todos los números de una lista. Por ejemplo: `sum([1,2,3,4])` debería devolver 10 y `multip([1,2,3,4])` debería devolver 24.
5. Escribir un pequeño programa donde:
 - Se ingresa el año en curso.
 - Se ingresa el nombre y el año de nacimiento de tres personas.
 - Se calcula cuántos años cumplirán durante el año en curso.

Para obtener datos por el teclado es necesario utilizar la función `input()`

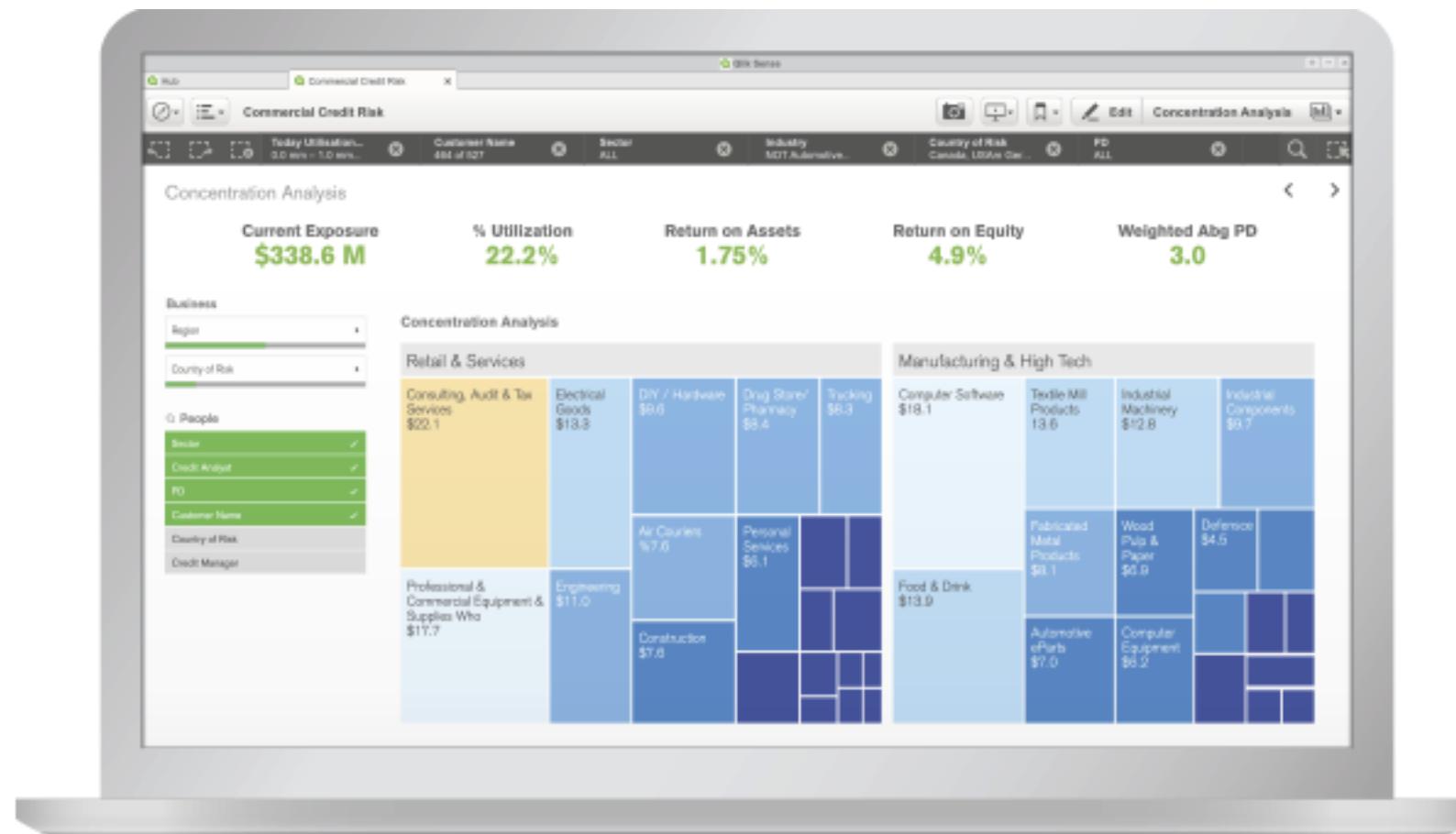
Ejercicios

6. Definir una tupla con 10 edades de personas. Imprimir la cantidad de personas con edades superiores a 20.
Puedes variar el ejercicio para que sea el usuario quien ingrese las edades.
7. Definir una lista con un conjunto de nombres, imprimir la cantidad de comienzan con la letra a.
También se puede hacer elegir al usuario la letra a buscar. (Un poco mas emocionante).
8. Crear una función contar_vocales(), que reciba una palabra y cuente cuantas letras "a" tiene, cuantas letras "e" tiene y así hasta completar todas las vocales. Se puede hacer que el usuario sea quien elija la palabra.
9. Escribir una función es_bisiesto() que determine si un año determinado es un año bisiesto. Un año bisiesto es divisible por 4, pero no por 100. También es divisible por 400.
10. Escribe un programa que pida dos palabras y diga si riman o no. Si coinciden las tres últimas letras tiene que decir que riman. Si coinciden sólo las dos últimas tiene que decir que riman un poco y si no, que no riman.

Para obtener datos por el teclado es necesario utilizar la función input()

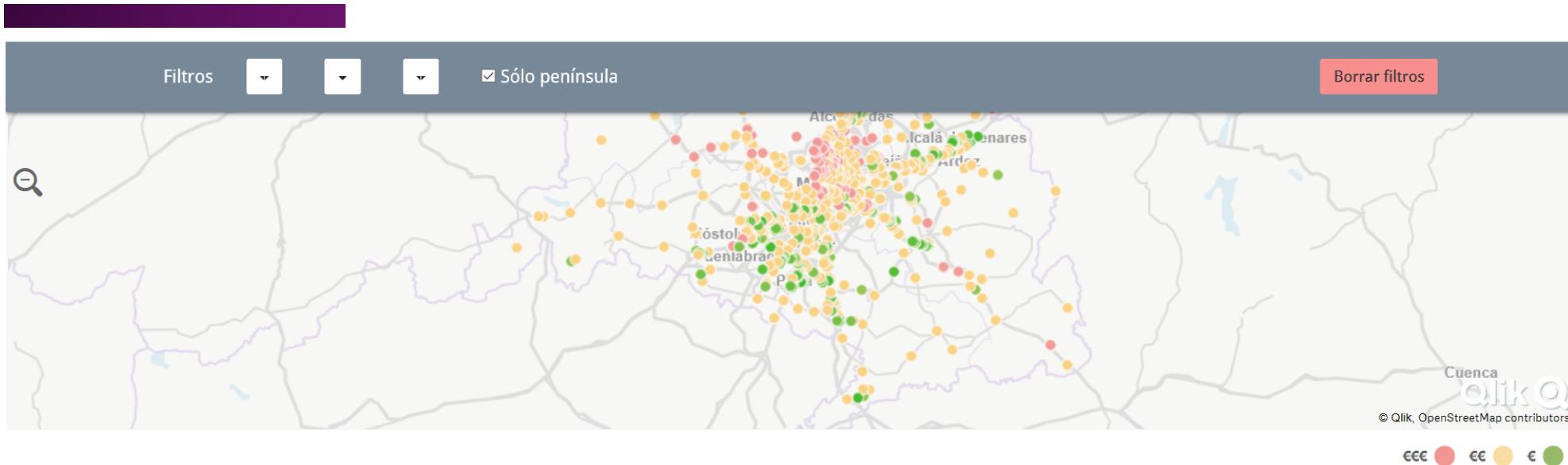
BIG DATA

QlikSense



Click me!

Gasolineras en España



La más cara: Collado Villalba: 1,349 €/l

La más barata: Getafe: 1,097 €/l

Gasolineras: 674



El precio más alto del periodo analizado se alcanzó el 12-10-2018. La Gasolina 95 llegó a costar entonces 1,49 euros por litro, mientras que el precio más bajo se registró el 26-02-2016 cuando la Gasolina 95 costaba 0,932 euros por litro.



Customer Service & Call Center

≡ ▾ 🔍 Customer Service & Call Center

Análisis Historia

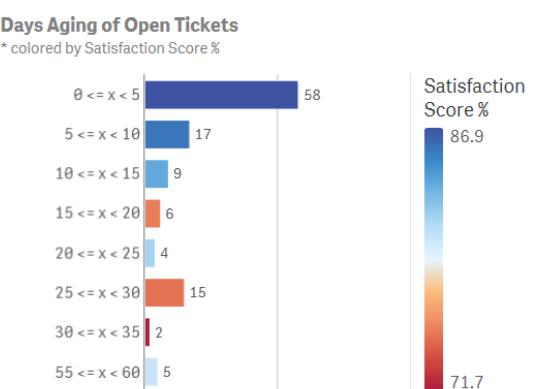
🔖 ▾
Open Tickets
◀ ▶

☰
Selecciones
Conocimientos

Open Tickets

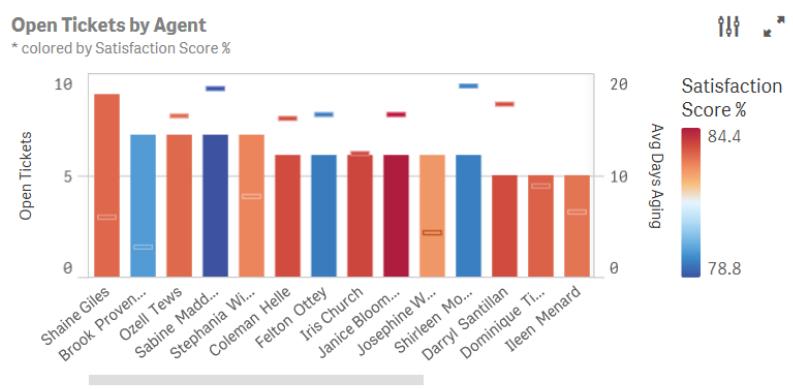
Severity Type
Customer Tier
Service Type
Location
Handle Type
Group

Days Aging of Open Tickets
* colored by Satisfaction Score %



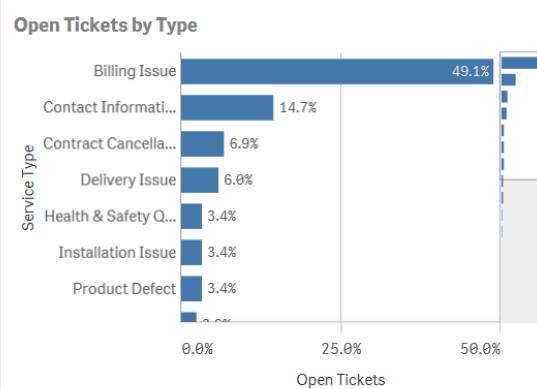
Aging Period	Open Tickets
0 <= x < 5	58
5 <= x < 10	17
10 <= x < 15	9
15 <= x < 20	6
20 <= x < 25	4
25 <= x < 30	15
30 <= x < 35	2
35 <= x < 60	5

Open Tickets by Agent
* colored by Satisfaction Score %



Agent	Open Tickets
Shaine Giles	10
Brook Proven...	8
Ozell Tews	8
Sabine Madd...	7
Stephanie Wi...	6
Coleman Heile	6
Felton Ottey	6
Iris Church	6
Janice Bloom...	6
Josephine W...	5
Shirleen Mo...	5
Darryl Santillan	4
Dominique Ti...	4
Ileen Menard	4

Open Tickets by Type



Service Type	Open Tickets
Billing Issue	49.1%
Contact Informati...	14.7%
Contract Cancella...	6.9%
Delivery Issue	6.0%
Health & Safety Q...	3.4%
Installation Issue	3.4%
Product Defect	3.4%

Agent Details

Agent	Q	Open Tickets	Avg Days Aging	Oldest Ticket
Totales		116	11	59
Shirleen Moman		6	19	59
Sabine Madden		7	19	29
Darryl Santillan		5	17	59
Willow Barry		5	16	58
Felton Ottey		6	16	59
Janice Bloomberg		6	16	58
Ozell Tews		7	16	30

Customer Details

Company	Q	Open Tickets	Avg Days Aging	Oldest Ticket
Totales		116	11	59
Ravenwerks		5	21	58
NBX		7	16	58
Bre-X		9	16	59
Vanstar		4	15	29
LiveWire BBS and Favourite Links		3	14	29
Dayton Malleable Inc.		7	14	30

Service Type Details

Service Type	Q	Open Tickets	Avg Days Aging	Oldest Ticket
Totales		116	11	59
Product Order		3	24	29
Parts Order		3	19	30
Contract Cancellation Request		8	17	58
Installation Issue		4	16	28
Functionality Question		2	16	29
Contact Information Change		17	13	59

Click me!

Salesforce

SALESFORCE

Dashboard Opportunities Accounts SalesPerson

SELECTIONS: None

Countries ▾ All Sales People ▾ All Opp Types ▾ All Products ▾ Today's date: 3/12/2019

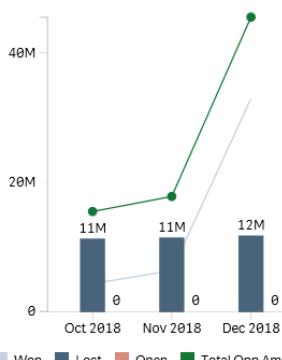
DASHBOARD

OPEN OPPS VALUE: \$19,894,456

OPEN OPPS: 291 NEW CUSTOMERS: 147 EXISTING CUSTOMERS: 144 | WIN RATE: 39% 22% WR THIS TIME LAST YEAR

PIPELINE

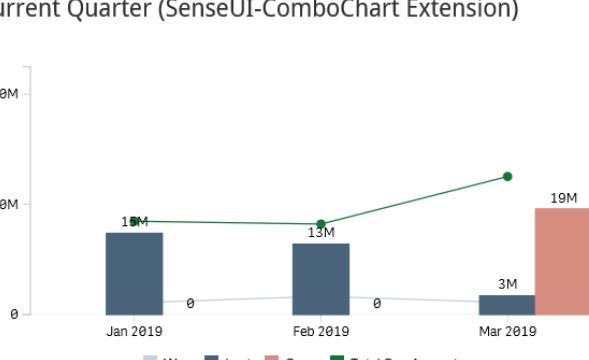
PREVIOUS QUARTER



Month	Total Opp Amount
Oct 2018	11M
Nov 2018	11M
Dec 2018	12M

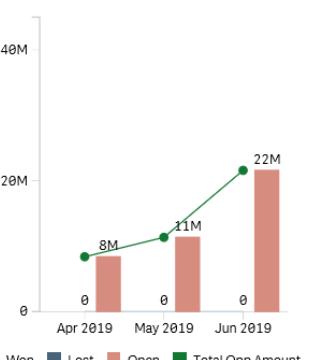
CURRENT QUARTER

Current Quarter (SenseUI-ComboChart Extension)



Month	Total Opp Amount
Jan 2019	18M
Feb 2019	13M
Mar 2019	22M

NEXT QUARTER FORECAST



Month	Total Opp Amount
Apr 2019	8M
May 2019	11M
Jun 2019	22M

TOP 3 CLOSED DEALS

Top 3 Closed Deals

TOP 3 PENDING DEALS

Top 3 Pending Deals

Click me!

Retail OmniChannel Analytics

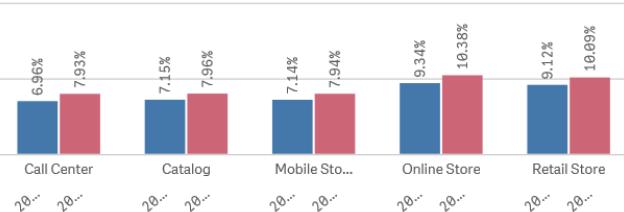
Omni-Channel Dashboard

Compare channel sales for 2014 and 2013

Last reloaded: 12/31/2014

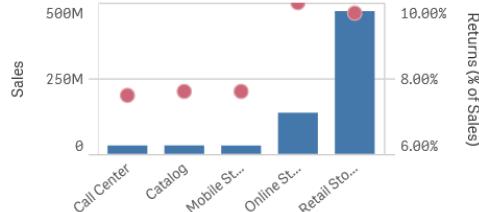
Total All Channels	Retail Store	Online Store	Call Center	Mobile Store	Catalog
2014 Sales: \$349,317,496	2014 Sales \$241,119,434	2014 Sales \$66,143,884	2014 Sales \$14,036,073	2014 Sales \$13,965,292	2014 Sales \$14,052,813
2013 Sales: \$340,966,570	2013 Sales \$230,273,656	2013 Sales \$69,772,627	2013 Sales \$13,611,683	2013 Sales \$13,540,919	2013 Sales \$13,767,684
Variance: 2.4%					
2014 Returns % of Sales: 10.4%	2013 to 2014 Variance \$10,845,778 ▲	2013 to 2014 Variance \$-3,628,743 ▼	2013 to 2014 Variance \$424,390 ▲	2013 to 2014 Variance \$424,373 ▲	2013 to 2014 Variance \$285,129 ▲
2014 Sales	4.7%	-5.2%	3.1%	3.1%	2.1%
2014 Returns as % of Sales 10.1%	2014 Returns as % of Sales 10.4%	2014 Returns as % of Sales 7.9%	2014 Returns as % of Sales 7.9%	2014 Returns as % of Sales 8.0%	2014 Returns as % of Sales 8.0%

Returns as % of Sales



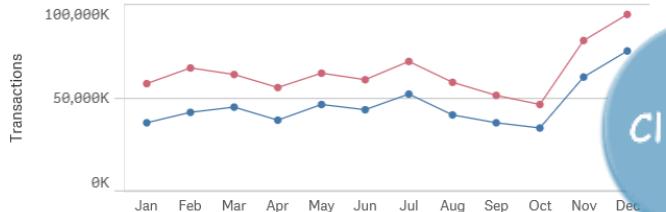
Channel	Returns as % of Sales
Call Center	6.96%
Catalog	7.15%
Mobile Store	7.14%
Online Store	9.34%
Retail Store	9.12%

Returns by Channel



Channel	Sales (M)	Returns (% of Sales)
Call Center	~100M	~7.0%
Catalog	~100M	~7.5%
Mobile Store	~100M	~7.5%
Online Store	~300M	~9.5%
Retail Store	~500M	~9.5%

Transactions by Month



Month	Sales (K)	Returns (K)
Jan	~40K	~60K
Feb	~45K	~65K
Mar	~48K	~62K
Apr	~42K	~58K
May	~45K	~60K
Jun	~43K	~59K
Jul	~50K	~70K
Aug	~45K	~65K
Sep	~40K	~55K
Oct	~35K	~50K
Nov	~55K	~75K
Dec	~60K	~80K

Click me!

Workforce Management

WORKFORCE MANAGEMENT
 ORDER TYPE ▾ DEPOT ▾

DASHBOARD | ●●●

NUMBER OF ORDERS	1,174
OPEN WORK ORDERS	126
TARGET	-30.00% 180
EMERGENCY WORK ORDERS	25
TARGET	-44.44% 45

OPEN ORDERS | ●●

OPEN WORK ORDERS	126
TARGET	-30.00% 180
EMERGENCY WORK ORDERS	25
TARGET	-44.44% 45

EMPLOYEE ANALYSIS

UTILIZATION AVERAGE %	88.9%
UNDER UTILIZED	26
OVER UTILIZED	8

ROUTE OPTIMIZATION

TRAVEL DISTANCE (KM)	26,878
----------------------	--------

SELECTIONS: None

DASHBOARD

NUMBER OF ORDERS
1,174
34 orders | +2.98% over target

TARGET
1140

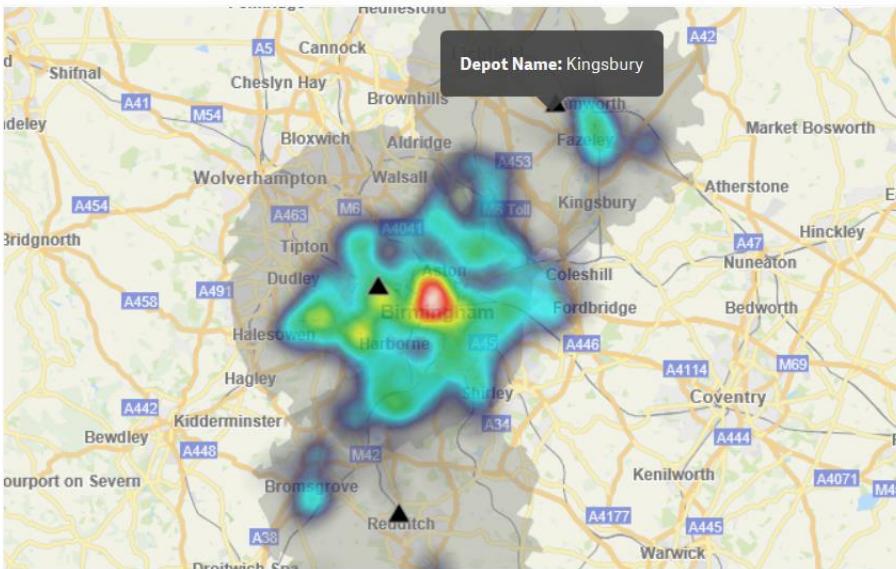
OPEN WORK ORDERS
126
-54 orders | -30.00% below target

TARGET
180

EMERGENCY WORK ORDERS
25
-20 orders | -44.44% below target

TARGET
45

No of Orders by Depot
*Gray area represents 10 mile radius from depot

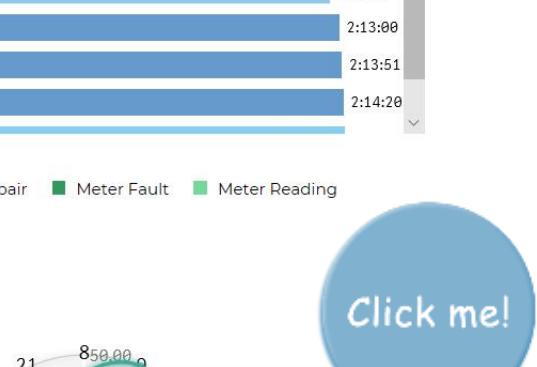


Avg Time to Resolve by Type

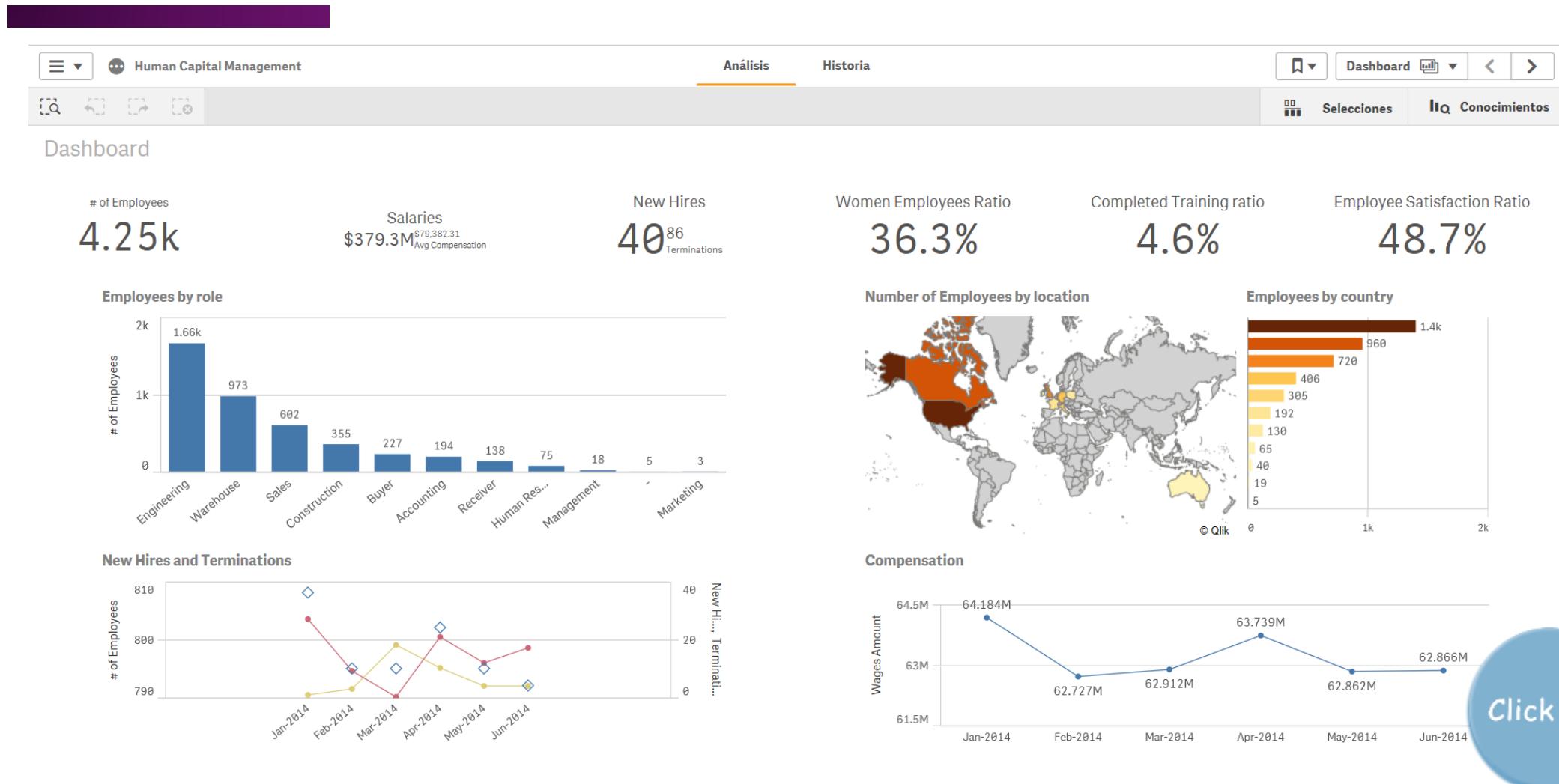
Type	Avg Time
Meter Reading	1:58:23
Priority Call	2:05:48
External Gas Escape	2:05:57
Emergency Gas Safety Check	2:07:05
Unknown Meter Fault	2:08:26
Meter Not Functioning	2:09:44
No Supply External	2:13:00
No Supply Internal	2:13:51
Internal Equipment Fault	2:14:20

Legend: Emergency (Dark Blue), Internal Repair (Light Blue), Meter Fault (Green), Meter Reading (Light Green)

Orders by Time of Day

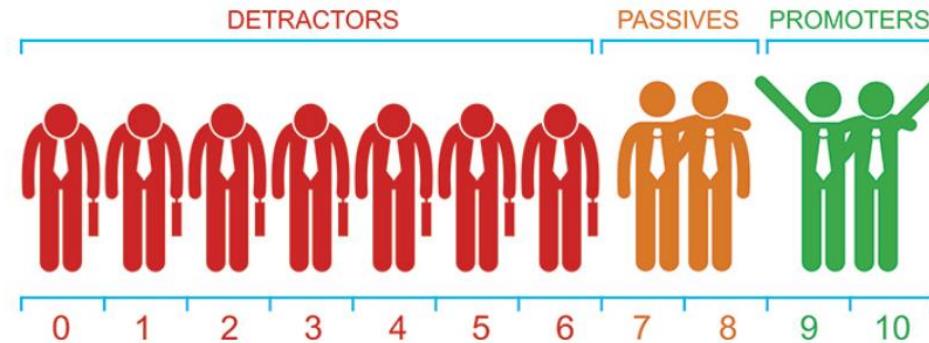


Human Capital Management



Sales, Customer Experience & Churn

Product Planning Marketing Planning & Customer Experience Management



This application analyses Customer Experience using the global Net Promoter Score (NPS) standard. We have a traditional descriptive analytics dashboard showing NPS, Sales & Gross Margin by Products. NPS can be a very challenging KPI when used in isolation, as it has limited diagnostic properties on its own.

To address this, we correlate NPS with a number of other KPI's to identify areas where NPS can be improved:

- Customer Satisfaction (usually measured by survey beside NPS)
- Customer Effort (a great predictor of future renewal / spend)
- Customer Journeys (routes taken through different channels)
- Moments of Truth (which equate to different stages in the customer lifecycle)

Click me!

¡Muchas Gracias!



GABRIEL MARÍN DÍAZ
LCDO. CIENCIAS FÍSICAS UCM

www.linkedin.com/in/gabrielmarindiaz/