

Introducción a la Inteligencia Artificial

Clase 1.1. Conceptos básicos de programación (en Python)

Departamento de Computación, FCEFQyN, Universidad Nacional de Río Cuarto, 2024

Python

- Ha conseguido una larga y activa comunidad de usuarios y desarrolladores
- Es hoy en día uno de los lenguajes más usados para
 - Ciencia de datos
 - Machine learning
 - Desarrollo de software en general
- Cuenta con potentes bibliotecas para diferentes dominios (pandas, matplotlib, scikit-learn, ...)

Modos de ejecución

1. IPython

Podemos ejecutar Python de forma interactiva

Similar a intérpretes de otros lenguajes, e.g., Haskell

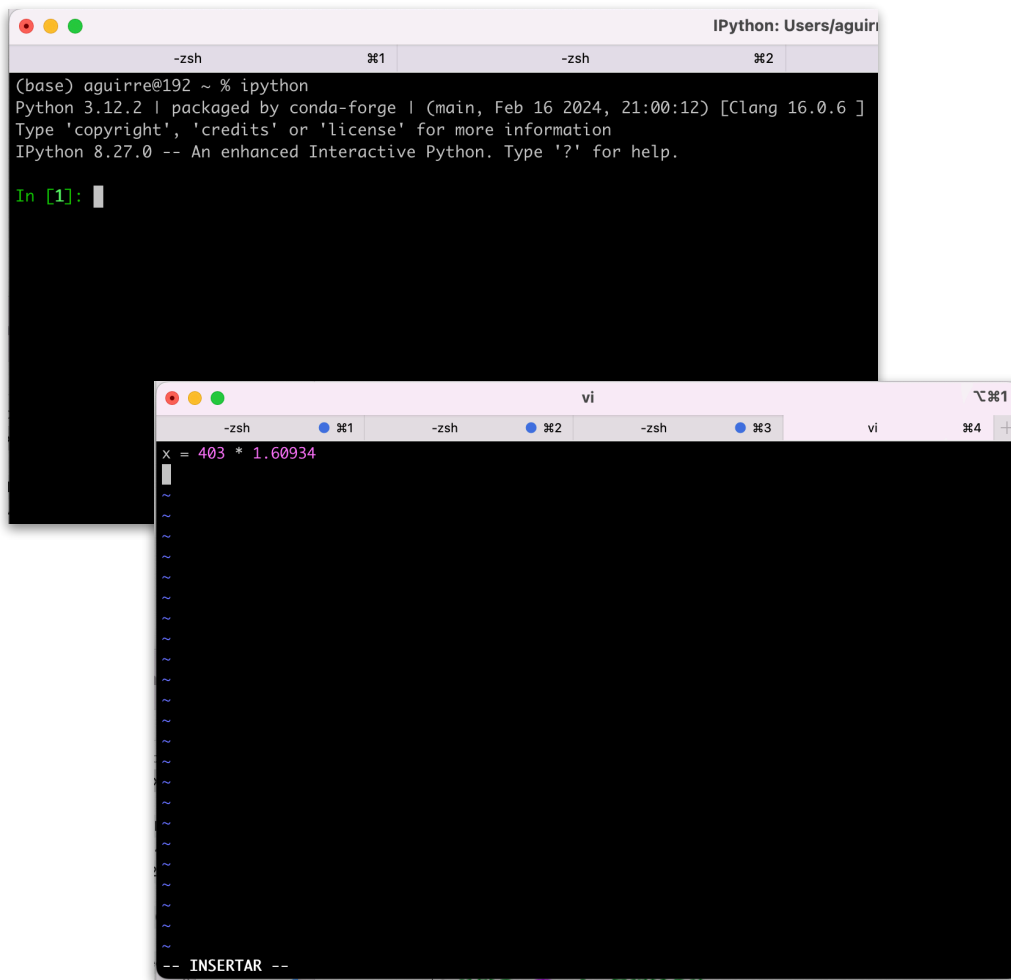
2. Scripts Python (programas)

A partir de una decena de líneas de código, **es la forma más adecuada de usar Python.**

Los scripts Python los almacenamos en archivos de texto

La extensión por convención es .py

Edición y ejecución de programas Python



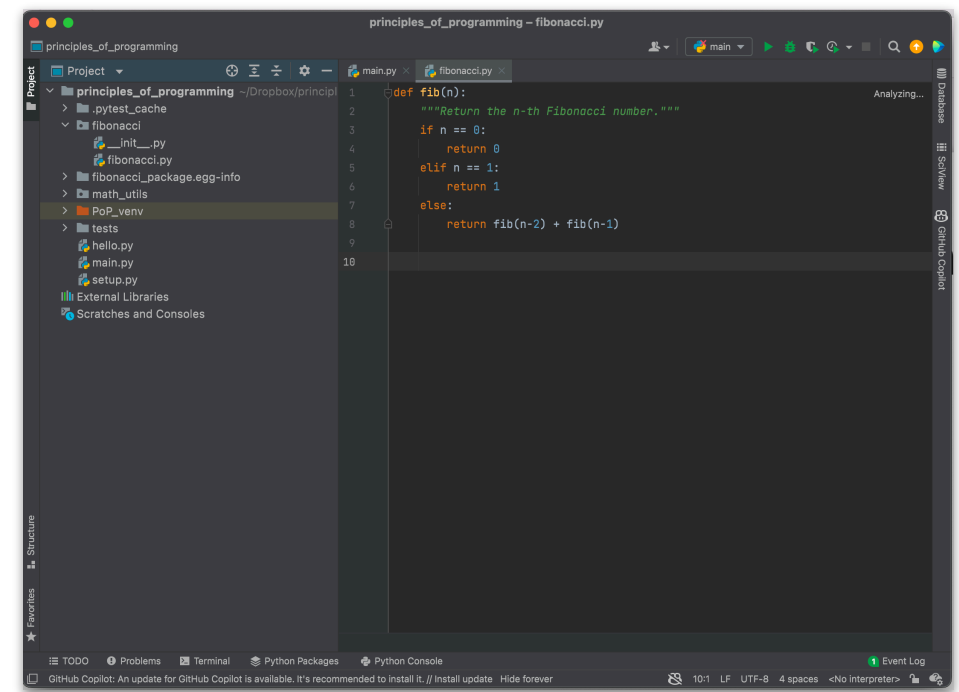
The image shows two terminal windows. The top window is titled 'IPython: Users/aguirre' and displays the IPython prompt. The bottom window is titled 'vi' and shows a text editor with a single line of code.

```
(base) aguirre@192 ~ % ipython
Python 3.12.2 | packaged by conda-forge | (main, Feb 16 2024, 21:00:12) [Clang 16.0.6 ]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.27.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

```
x = 403 * 1.60934
```

-- INSERTAR --



The image shows a code editor window titled 'principles_of_programming - fibonacci.py'. The editor displays a Python function for calculating the nth Fibonacci number. The left sidebar shows a project structure with files like 'fibonacci.py', 'main.py', and 'setup.py'. The bottom status bar indicates the file is 10 lines long, using LF line endings, UTF-8 encoding, and 4 spaces for indentation.

```
def fib(n):
    """Return the n-th Fibonacci number."""
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

Agenda

- Variables
- Tipos
- Operadores aritméticos
- Lógica proposicional (y operadores lógicos)
- Strings (cadenas de caracteres)
- I/O básica (entradas por consola e interactivas, impresión en pantalla)

Python desde el intérprete interactivo

- Calculemos la edad de Bobi (el perro más longevo) en años humanos

```
(base) aguirre@192 ~ % ipython
Python 3.12.2 | packaged by conda-forge | (main, Feb 16 2024, 21:00:12) [Clang 16.0.6]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.27.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: 31.45 * 7
Out[1]: 220.15

In [2]: █
```

Variables

- Python es un lenguaje de **programación imperativa**
 - Se define el **estado de un programa** a través de **variables**
 - Las variables almacenan valores
 - Se puede acceder al valor almacenado en una variable por su nombre
 - Los valores de variables se pueden actualizar
 - Los nombres de variables son “case sensitive”



```
IPython: Users/aguirre 1

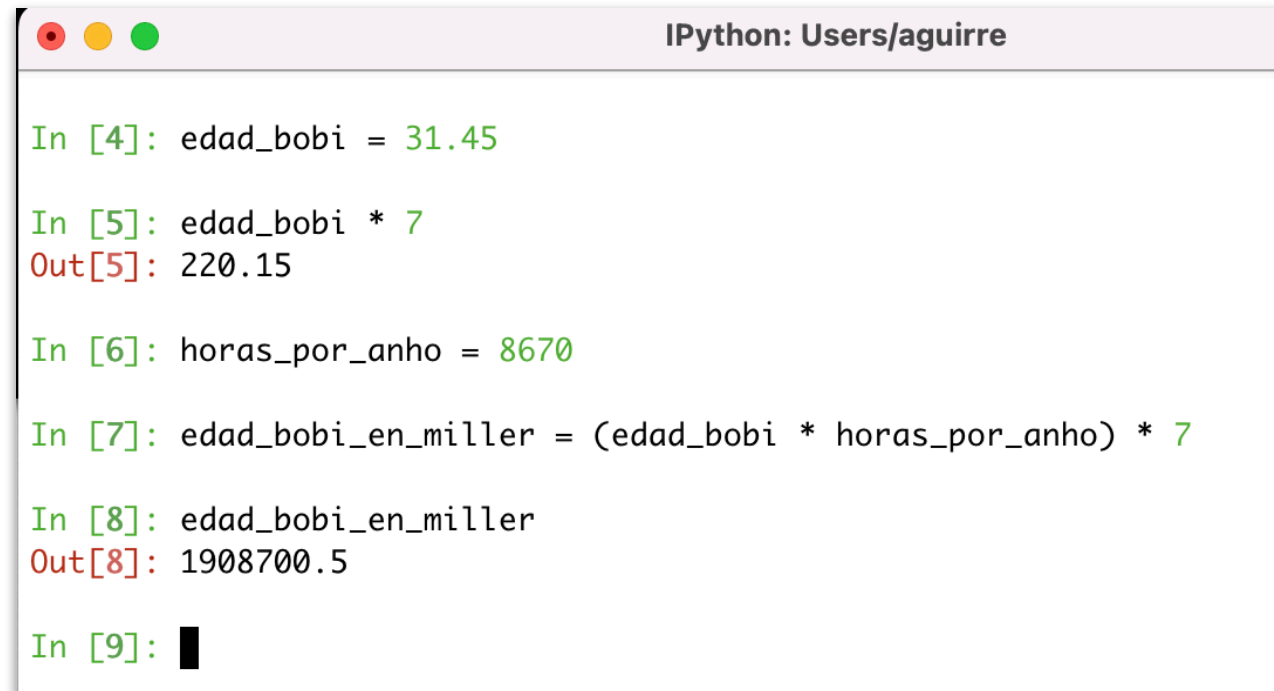
In [4]: edad_bobi = 31.45

In [5]: edad_bobi * 7
Out[5]: 220.15

In [6]:
```

Variables (cont.)

Los valores en las variables persisten hasta que los sobre-escribamos, termine el programa (o se abandone el bloque respectivo)



```
IPython: Users/aguirre

In [4]: edad_bobi = 31.45

In [5]: edad_bobi * 7
Out[5]: 220.15

In [6]: horas_por_anho = 8670

In [7]: edad_bobi_en_miller = (edad_bobi * horas_por_anho) * 7

In [8]: edad_bobi_en_miller
Out[8]: 1908700.5

In [9]: █
```

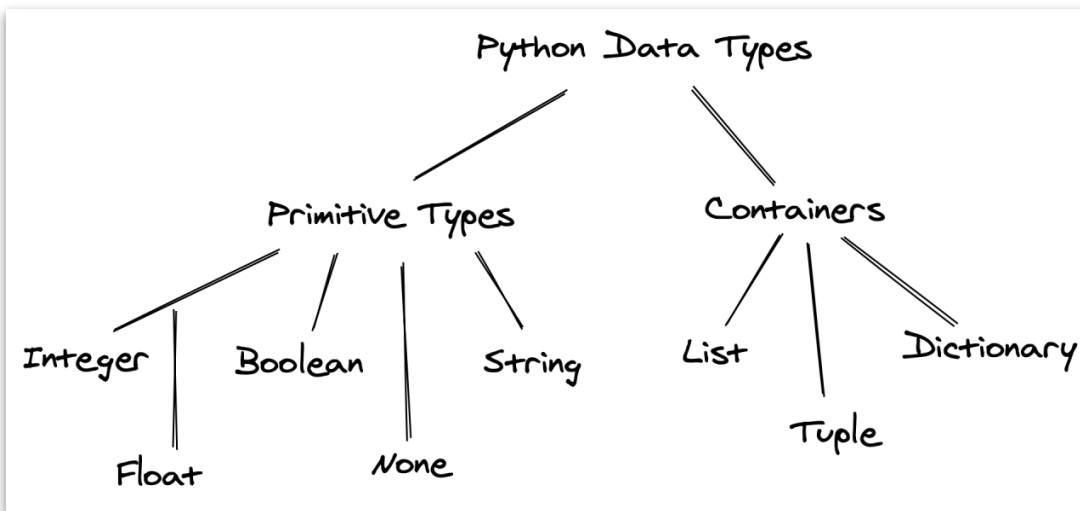

Tipos

Todas las variables en Python tienen un **tipo**

- El tipo define los valores admisibles (posibles) de una variable, y la forma en que éstos se almacenan

Python es un lenguaje con **tipado dinámico**

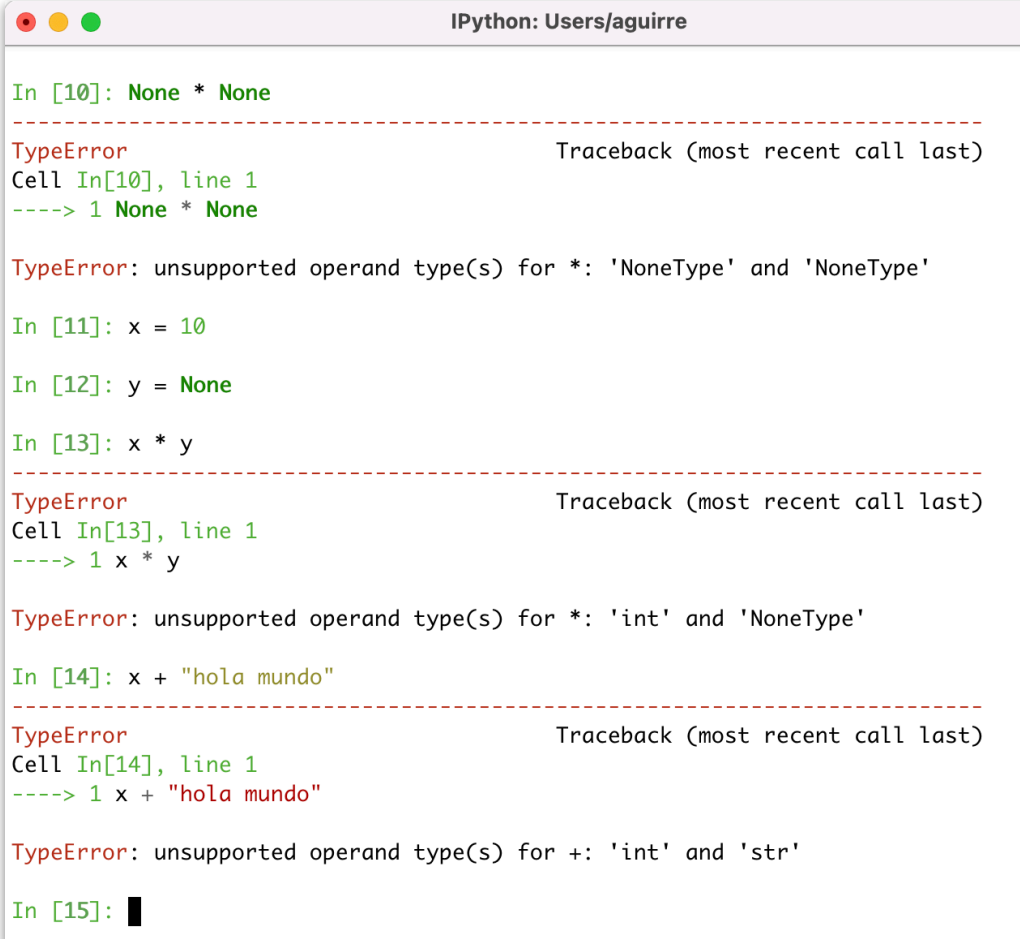
- Se puede cambiar el tipo de una variable en tiempo de ejecución (asignando un valor de otro tipo)



La importancia de los tipos de datos

Es importante usar los tipos de datos consistentemente

- Los tipos de las variables y expresiones limitan la viabilidad de las operaciones
- Es importante **no abusar del tipado dinámico**
- Es importante respetar el tipo de las variables



```
IPython: Users/aguirre

In [10]: None * None
-----
TypeError                                 Traceback (most recent call last)
Cell In[10], line 1
----> 1 None * None

TypeError: unsupported operand type(s) for *: 'NoneType' and 'NoneType'

In [11]: x = 10
In [12]: y = None
In [13]: x * y
-----
TypeError                                 Traceback (most recent call last)
Cell In[13], line 1
----> 1 x * y

TypeError: unsupported operand type(s) for *: 'int' and 'NoneType'

In [14]: x + "hola mundo"
-----
TypeError                                 Traceback (most recent call last)
Cell In[14], line 1
----> 1 x + "hola mundo"

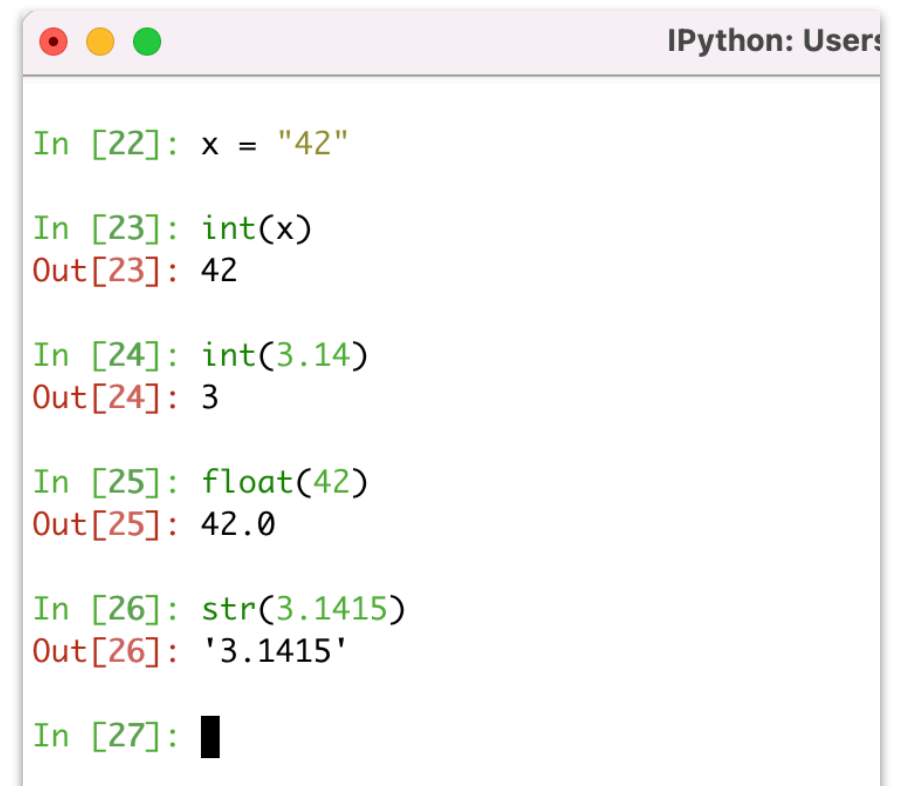
TypeError: unsupported operand type(s) for +: 'int' and 'str'

In [15]: █
```

Conversión de tipos (casts)

Python permite la **conversión de tipos** a través de **casts**

- Los casts reciben una expresión de cierto tipo, y producen valores del tipo que el cast indica
- Los podemos pensar como operaciones de conversión predefinidas

A screenshot of an IPython terminal window. The title bar at the top says "IPython: Users". The terminal shows several input-output pairs for type casting. The inputs are in green, and the outputs are in red. The last line shows an input in green followed by a black cursor.

```
In [22]: x = "42"

In [23]: int(x)
Out[23]: 42

In [24]: int(3.14)
Out[24]: 3

In [25]: float(42)
Out[25]: 42.0

In [26]: str(3.1415)
Out[26]: '3.1415'

In [27]: █
```

Operaciones aritméticas

La sintaxis es similar a la de otros lenguajes de programación (y Matemática)

Los órdenes de precedencia son los clásicos, y ayudan a reducir el número de paréntesis en las expresiones

- Podemos usar paréntesis para alterar la precedencia o ganar claridad

Operators	Meaning	Example	Result
+	Addition	$4 + 2$	6
-	Subtraction	$4 - 2$	2
*	Multiplication	$4 * 2$	8
/	Division	$4 / 2$	2
%	Modulus operator to get remainder in integer division	$5 \% 2$	1
**	Exponent	$5**2 = 5^2$	25
//	Integer Division/ Floor Division	$5//2$ $-5//2$	2 -3

Operadores lógicos

- Python cuenta con el tipo bool para valores booleanos (lógica proposicional)
 - Constantes: True y False
 - Python cuenta con los operadores lógicos clásicos:
 - and (conjunción)
 - or (disyunción)
 - not (negación)
 - == (si y sólo si)
 - != (disyunción exclusiva)

a	not a	a	b	a and b	a or b
False	True	False	False	False	False
True	False	False	True	False	True
		True	False	False	True
		True	True	True	True

Truth-table definitions of bool operations

Operadores de Comparación

Los operadores de comparación (u operadores relacionales) ofrecen una sintaxis similar a la de otros lenguajes de programación

Su tipo es *booleano*

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

Encadenamiento de Operadores de Comparación

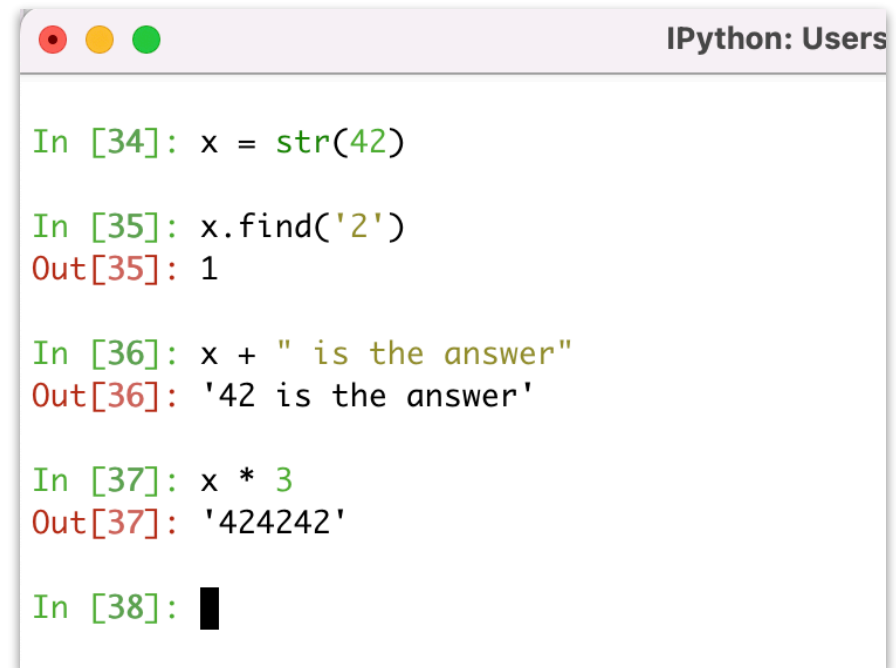
Python también admite el encadenamiento de operadores de comparación

Es una forma concisa y declarativa de expresar muchas comparaciones compuestas

```
x < y < z
```

Strings (cadenas de caracteres)

- Python ofrece una amplia variedad de operadores sobre strings
 - Casts (de y hacia str)
 - Concatenación (+)
 - Multiplicación (*)
 - Métodos (las strings son objetos) como upper(), lower(), find(), ...
- Las cadenas de caracteres son **inmutables** en Python

A screenshot of an IPython terminal window titled "IPython: Users". The window shows a series of Python commands and their outputs. The commands are: 1. x = str(42) (In [34]), 2. x.find('2') (In [35]), 3. x + " is the answer" (In [36]), 4. x * 3 (In [37]), and 5. A blank line (In [38]). The outputs are: 1. (no output shown for In [34]), 2. 1 (Out[35]), 3. '42 is the answer' (Out[36]), 4. '424242' (Out[37]), and 5. (no output shown for In [38]).

```
IPython: Users

In [34]: x = str(42)

In [35]: x.find('2')
Out[35]: 1

In [36]: x + " is the answer"
Out[36]: '42 is the answer'

In [37]: x * 3
Out[37]: '424242'

In [38]:
```


Métodos de String

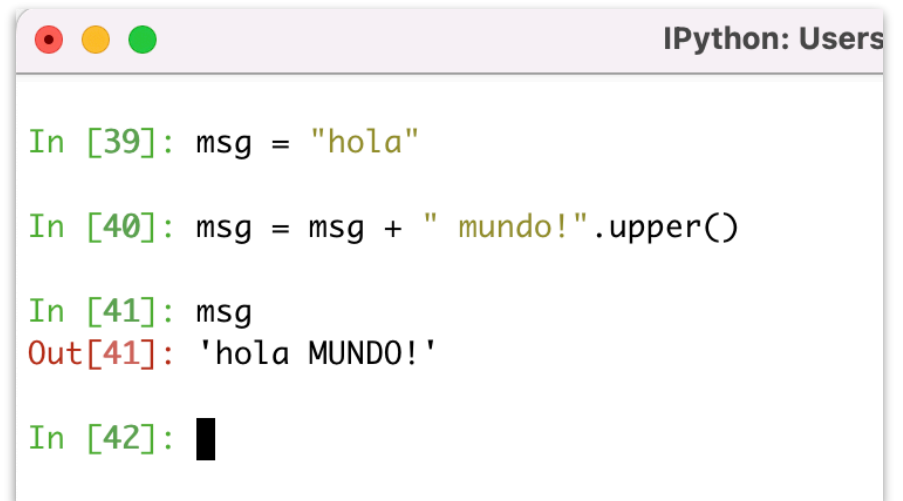
Las cadenas de caracteres son objetos

Ya veremos objetos y clases en detalle más adelante

Admiten la ejecución de métodos

Funciones definidas sobre strings, que se invocan con ‘

El comando **dir('str')** lista los métodos disponibles en strings



```
IPython: Users

In [39]: msg = "hola"

In [40]: msg = msg + " mundo!".upper()

In [41]: msg
Out[41]: 'hola MUNDO!'

In [42]:
```

Combinación de strings con otros valores

Cuando necesitamos combinar strings con otros datos, podemos hacerlo a través de **casts**:

```
In [43]: str(42) + " is the answer"
Out[43]: '42 is the answer'
```

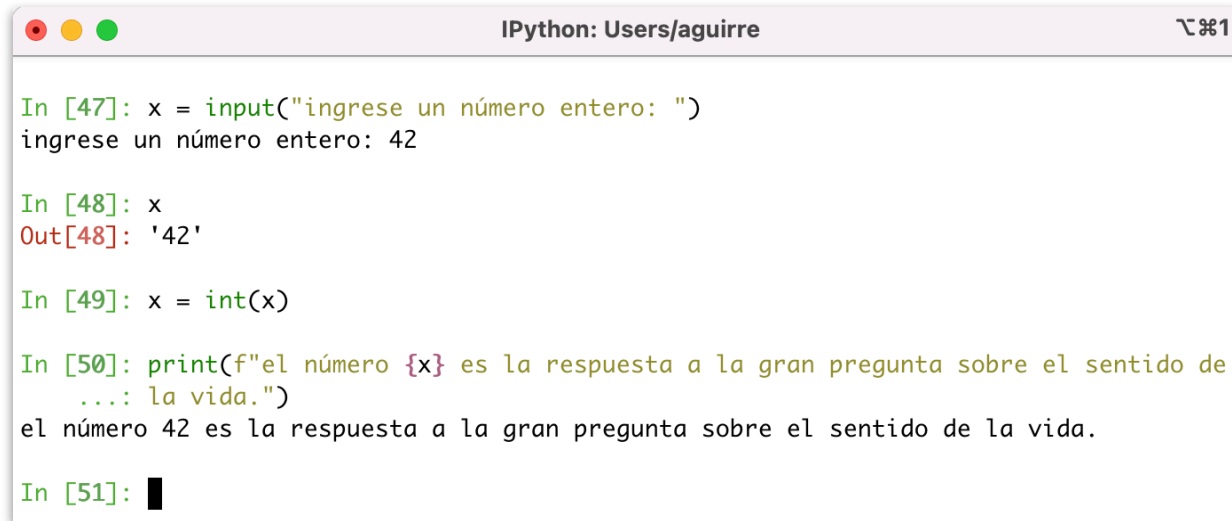
O podemos hacerlo a través de **cadenas formateadas** (formatted strings):

```
In [44]: x = 42

In [45]: f"{x} is the answer"
Out[45]: '42 is the answer'
```

Entrada/Salida básica

- Especialmente cuando usamos scripts, necesitamos interactuar con entrada/salida
 - Solicitar datos al usuario
 - Imprimir valores en la pantalla (consola o terminal)
- Python ofrece dos funciones muy útiles para esto:
 - **input**: espera por un dato y lo retorna (en formato string)
 - **print**: imprime valores en pantalla

A screenshot of an IPython terminal window titled "IPython: Users/aguirre". The window shows a series of Python commands and their outputs. The first command is `x = input("ingrese un número entero: ")`, which prompts the user to enter a number. The user enters "42". The second command is `x`, which returns the string `'42'`. The third command is `x = int(x)`. The fourth command is `print(f"el número {x} es la respuesta a la gran pregunta sobre el sentido de ...: la vida.")`, which prints the message "el número 42 es la respuesta a la gran pregunta sobre el sentido de la vida.". The fifth command is `In [51]:` followed by a black square, indicating the prompt is ready for the next input.

```
IPython: Users/aguirre 1

In [47]: x = input("ingrese un número entero: ")
ingrese un número entero: 42

In [48]: x
Out[48]: '42'

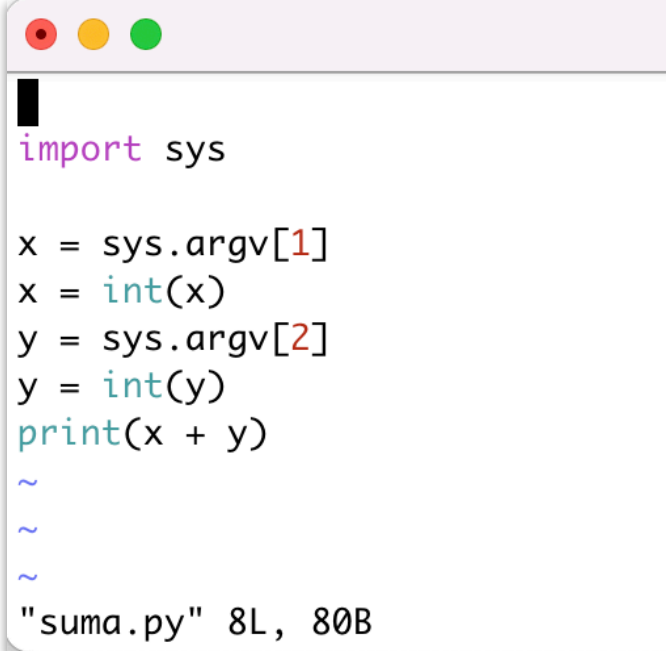
In [49]: x = int(x)

In [50]: print(f"el número {x} es la respuesta a la gran pregunta sobre el sentido de
...: la vida.")
el número 42 es la respuesta a la gran pregunta sobre el sentido de la vida.

In [51]: █
```

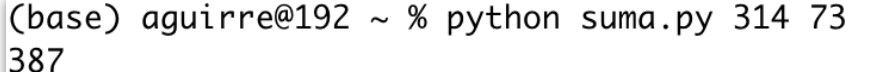
Pasaje de parámetros por línea de comandos

- Cuando escribimos scripts, **el uso de `input()` es en general poco conveniente**
 - Convierte al script en interactivo
 - Dificulta la implementación de infraestructura “independiente”
- En general, **suele ser más conveniente pasar parámetros a scripts Python**
 - **por línea de comandos** (si las entradas son básicas, simples, acotadas)
 - a través de fuentes de datos (e.g., archivos)
- Para acceder a parámetros de línea de comandos, podemos usar **`sys.argv`**



```
import sys

x = sys.argv[1]
x = int(x)
y = sys.argv[2]
y = int(y)
print(x + y)
~
~
~
"suma.py" 8L, 80B
```



```
(base) aguirre@192 ~ % python suma.py 314 73
387
```