

Introducción a la Programación

Principio básico de programación

Analizo → Resuelvo → Programo

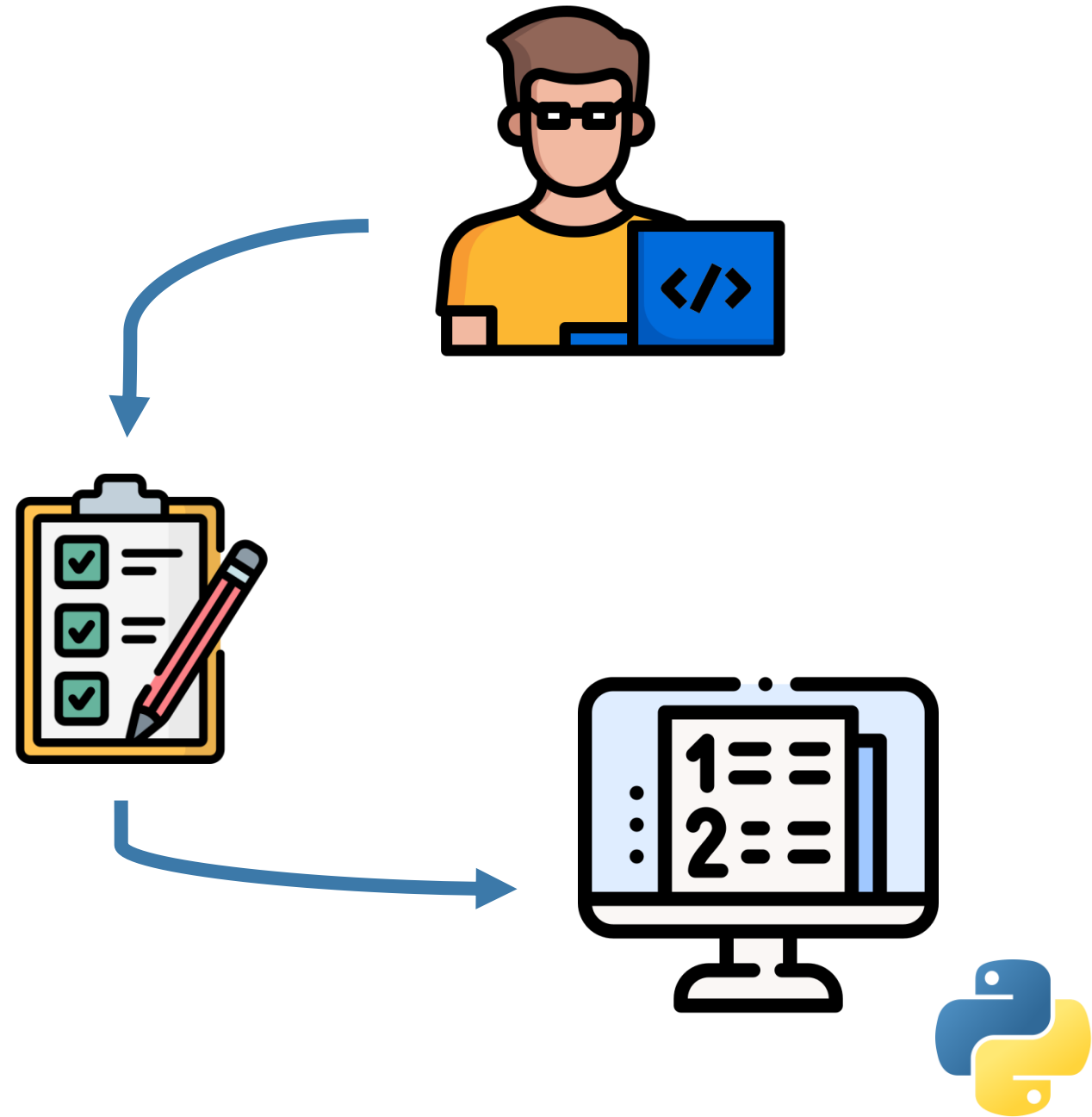
- Primero analizo
- Luego resuelvo
- Al final programo



Ejemplo

Lista de Compras:

- ☐ 1 Kg. de arroz
- ☐ 3 diente de ajo
- ☐ 2 Kg. de tomate
- ☐ 1 Lt. de leche
- ☐ 1 Coca Cola



Conceptos de Programación

Algoritmo. - conjunto ordenado de procesos que llevan a cabo una serie de instrucciones que ofrecen respuestas a problemas determinados.

Las partes de un algoritmo son:

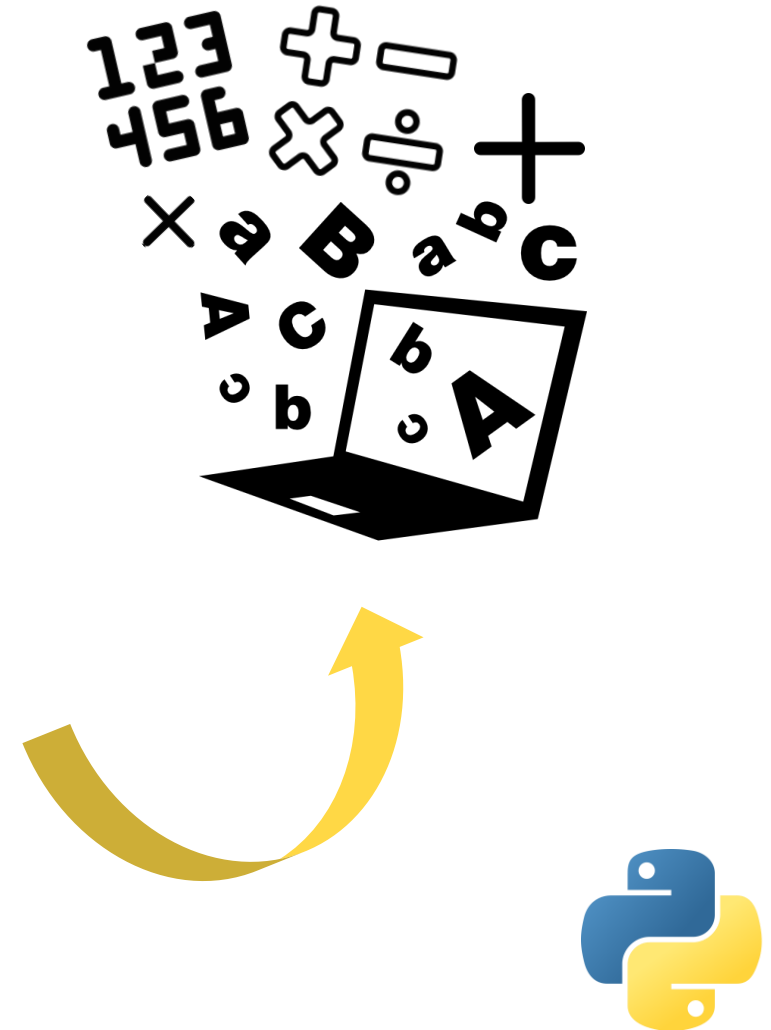
1. Datos de entrada
2. Procesamiento de datos
3. Resultado



Conceptos de Programación

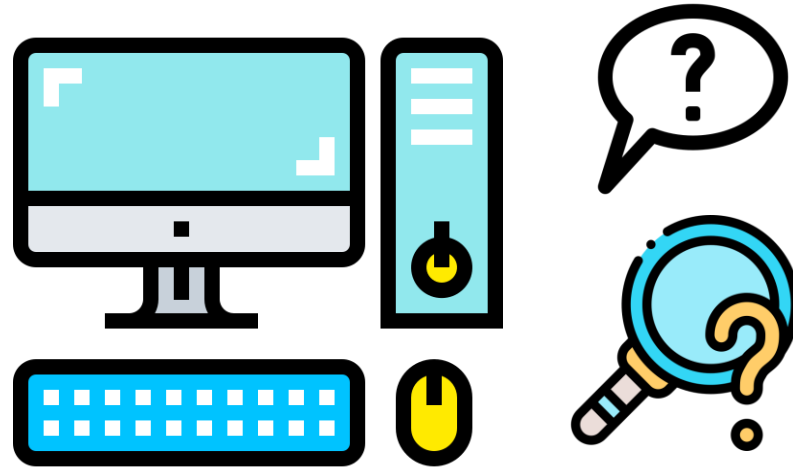
Lenguaje de programación. – es un lenguaje formal que proporciona a una persona, en este caso el programador, la capacidad y habilidad de escribir (o programar) una serie de instrucciones o secuencias de órdenes en forma de algoritmos.

- Los lenguajes de programación permiten crear programas que ejecutan algoritmos, mismos que controlan el comportamiento de una máquina.



Conceptos de Programación

¿Que es programar?



A través de los lenguajes de programación, un programador escribe un programa o algoritmo para indicarle a una maquina o un computador que hacer, como hacerlo, cuando hacerlo, etc.

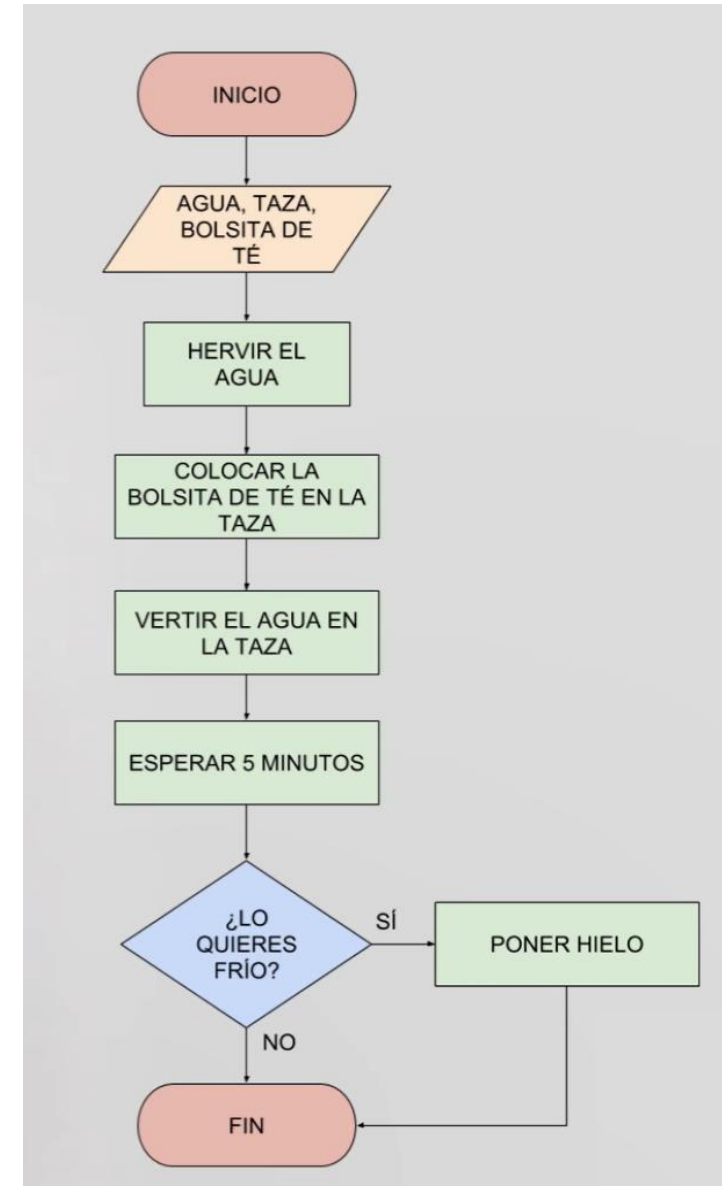
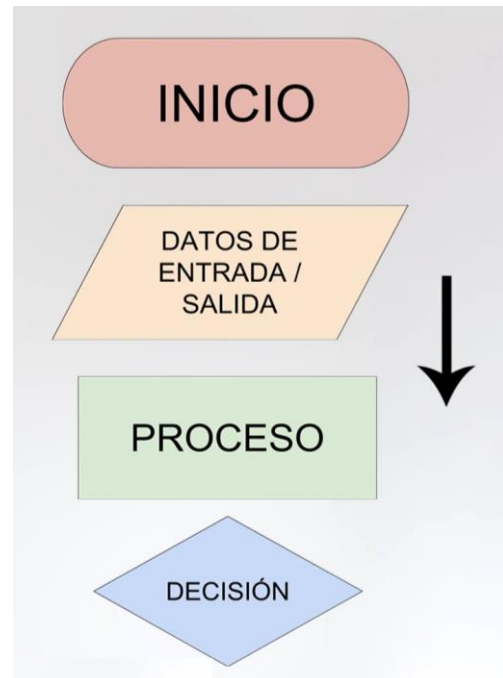


Conceptos de Programación

Diagrama de Flujo

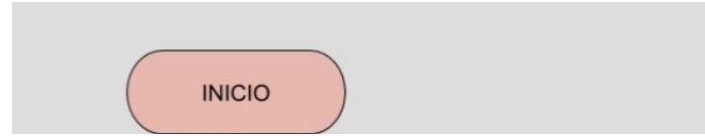
Es una forma visual de representar un algoritmo.

Un diagrama de flujo puede tener las siguientes partes:



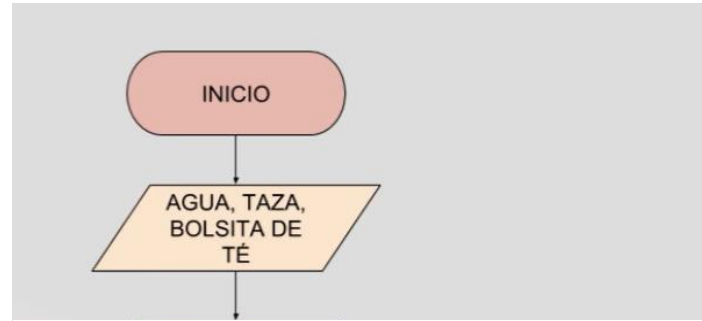
Conceptos de Programación

Diagrama de Flujo



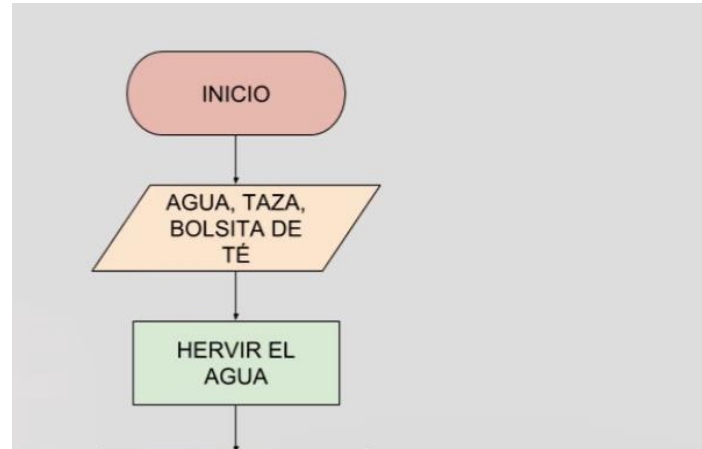
Conceptos de Programación

Diagrama de Flujo



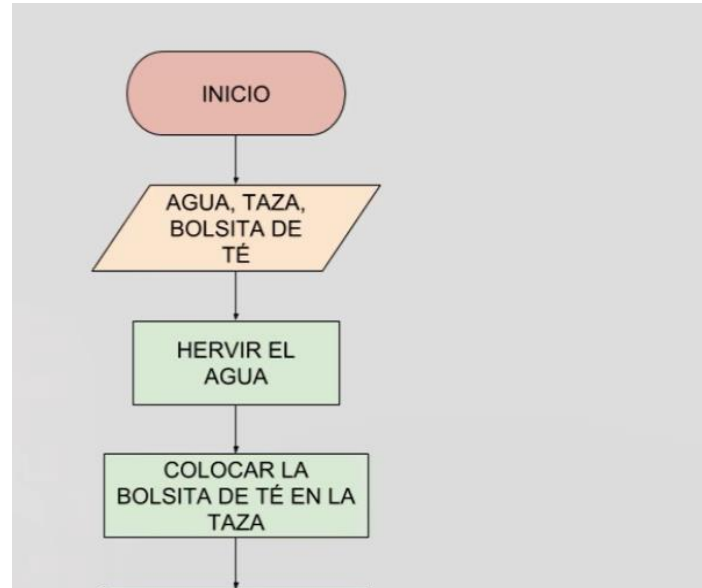
Conceptos de Programación

Diagrama de Flujo



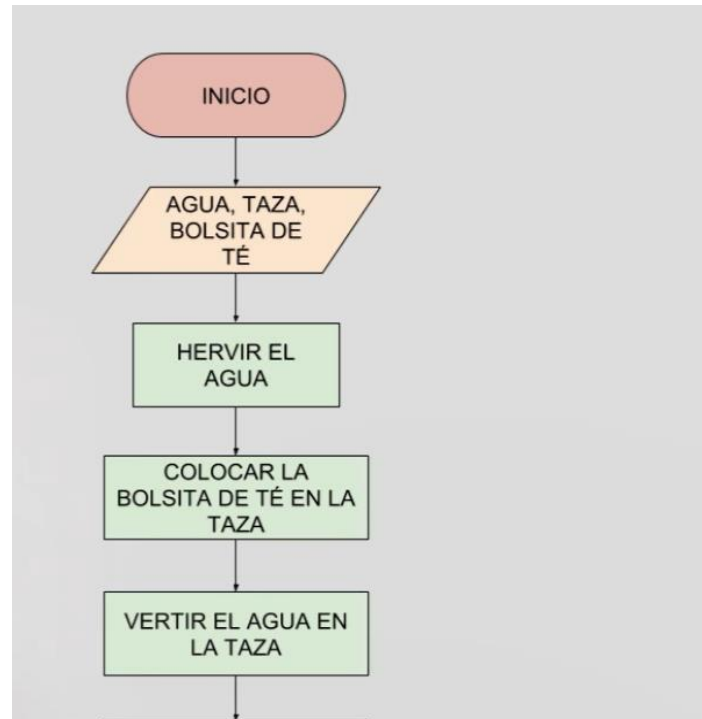
Conceptos de Programación

Diagrama de Flujo



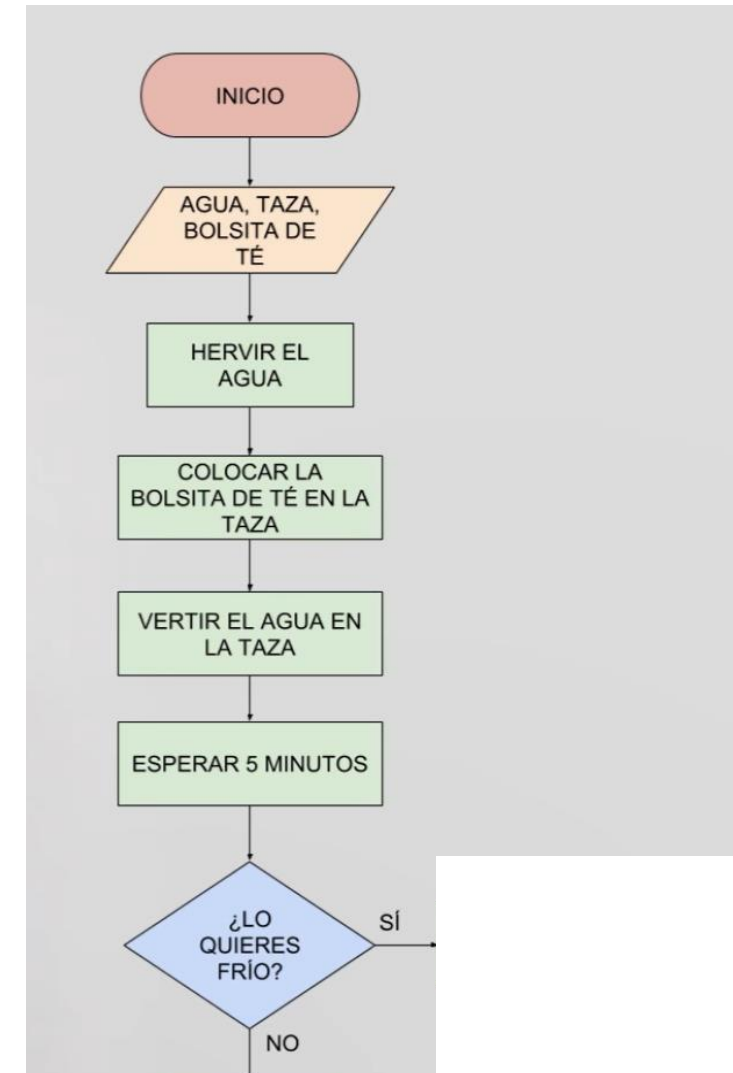
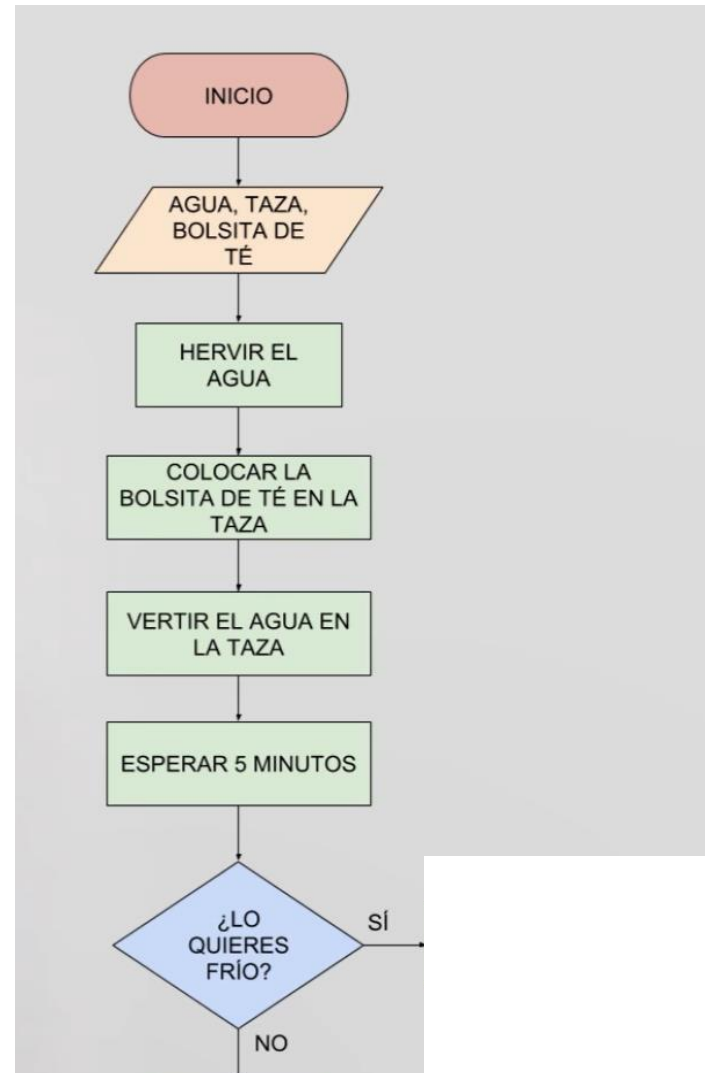
Conceptos de Programación

Diagrama de Flujo



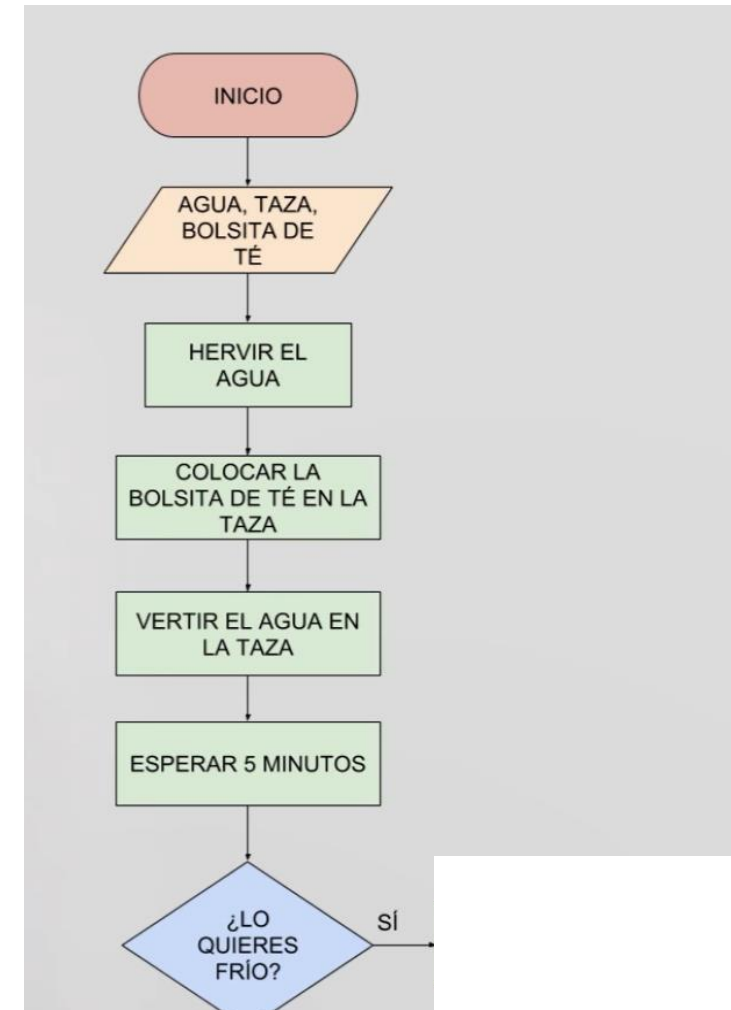
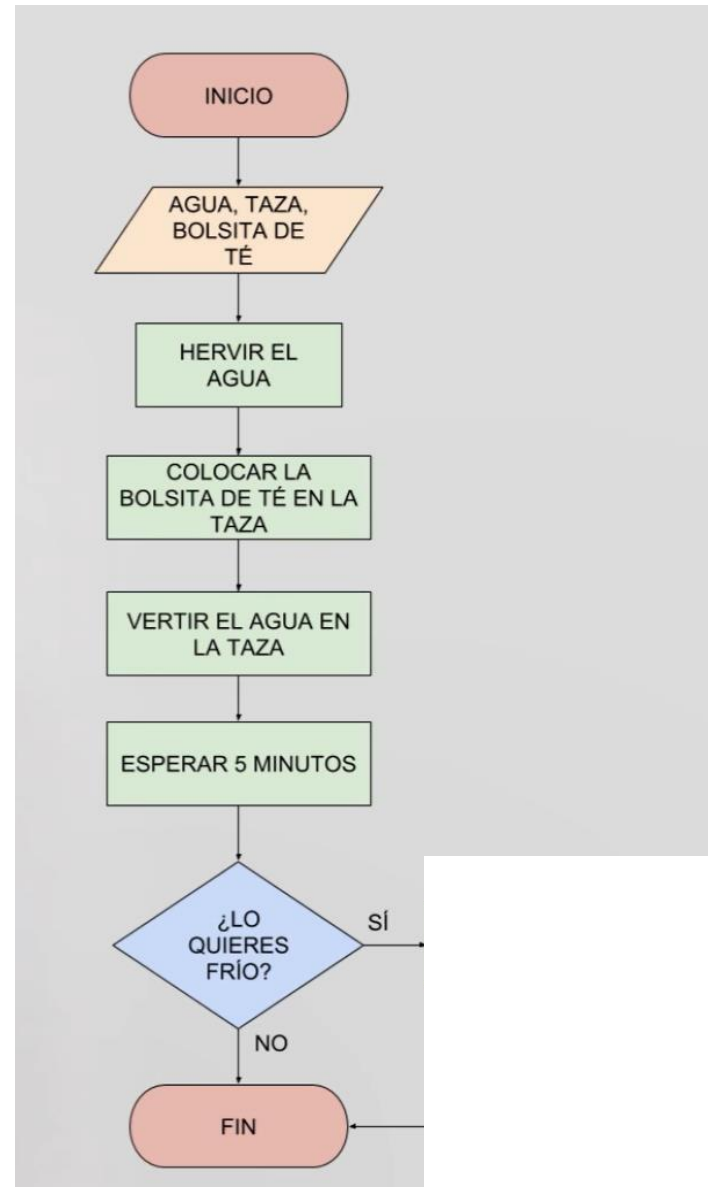
Conceptos de Programación

Diagrama de Flujo



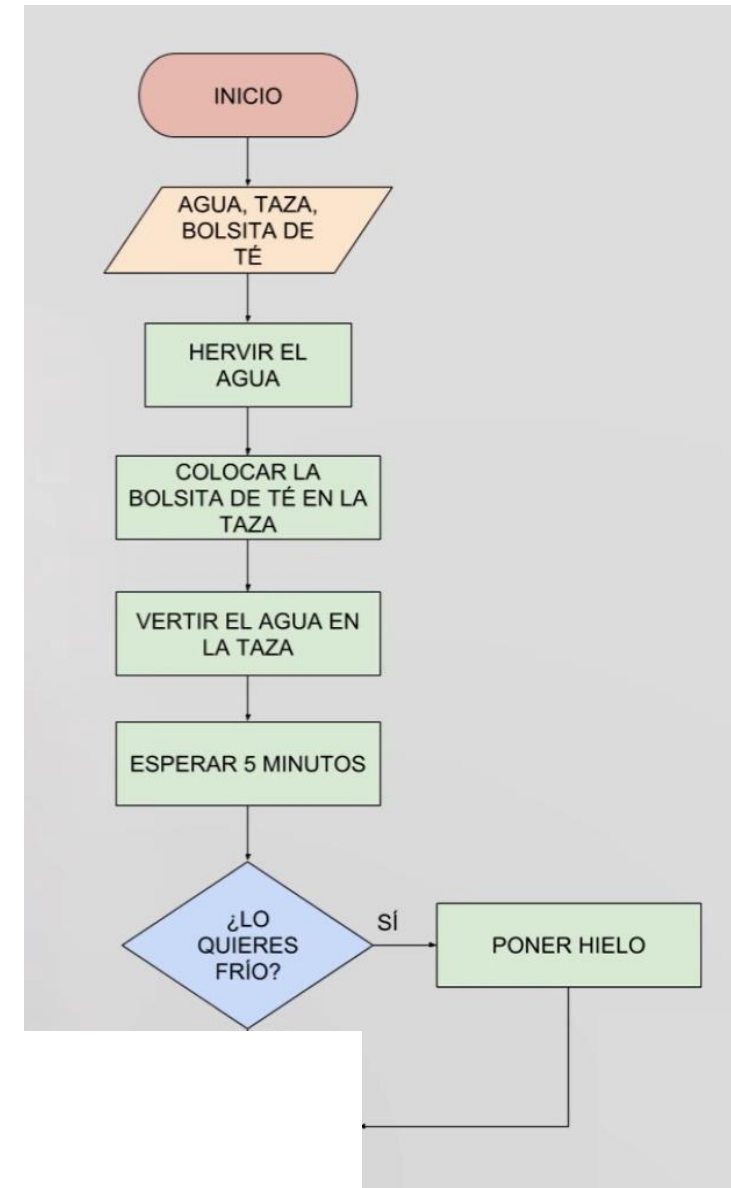
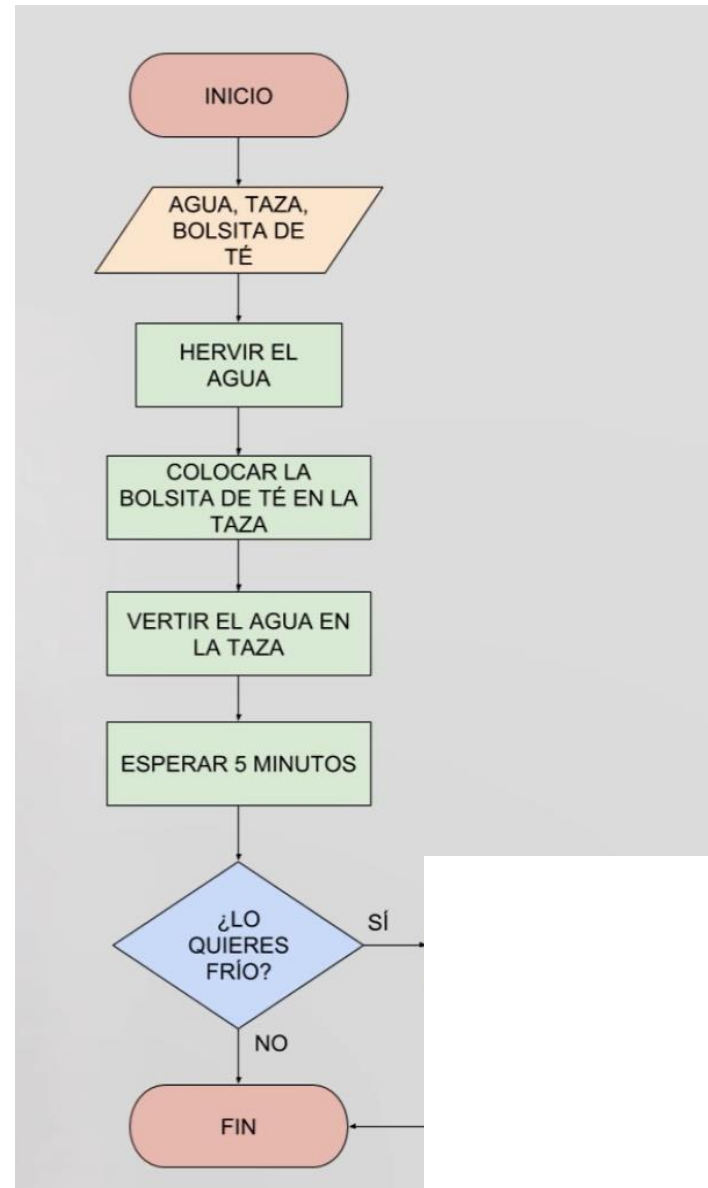
Conceptos de Programación

Diagrama de Flujo



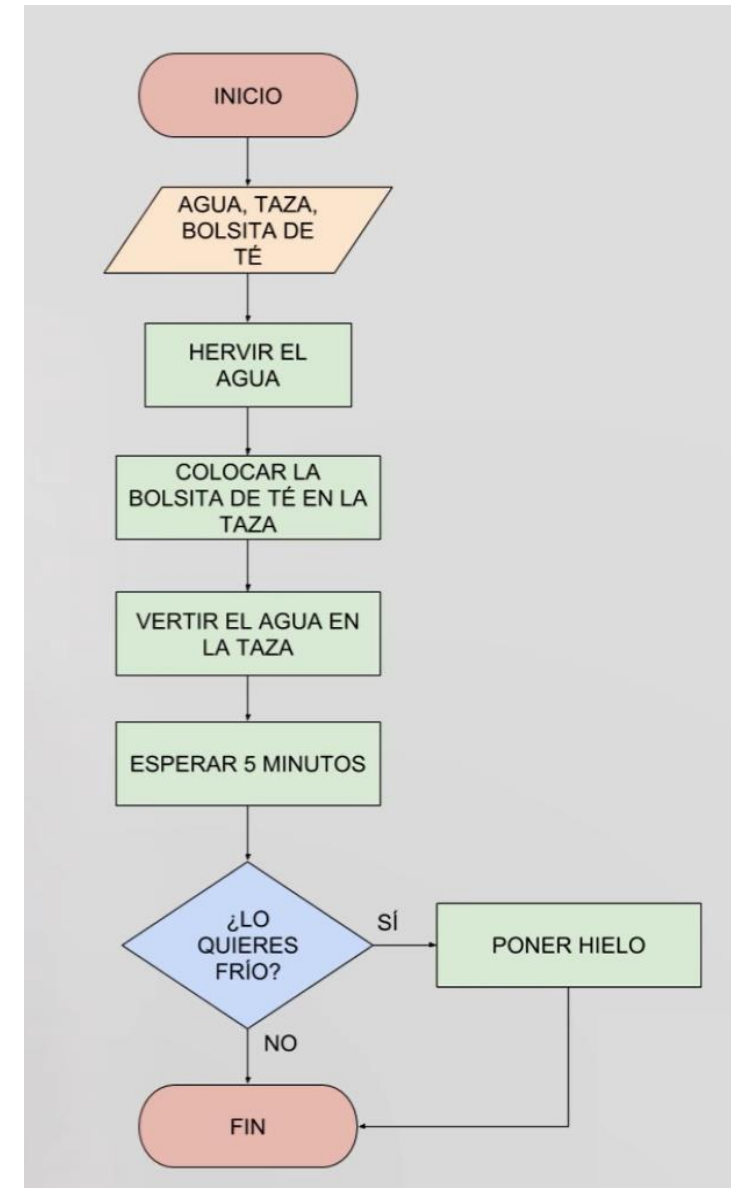
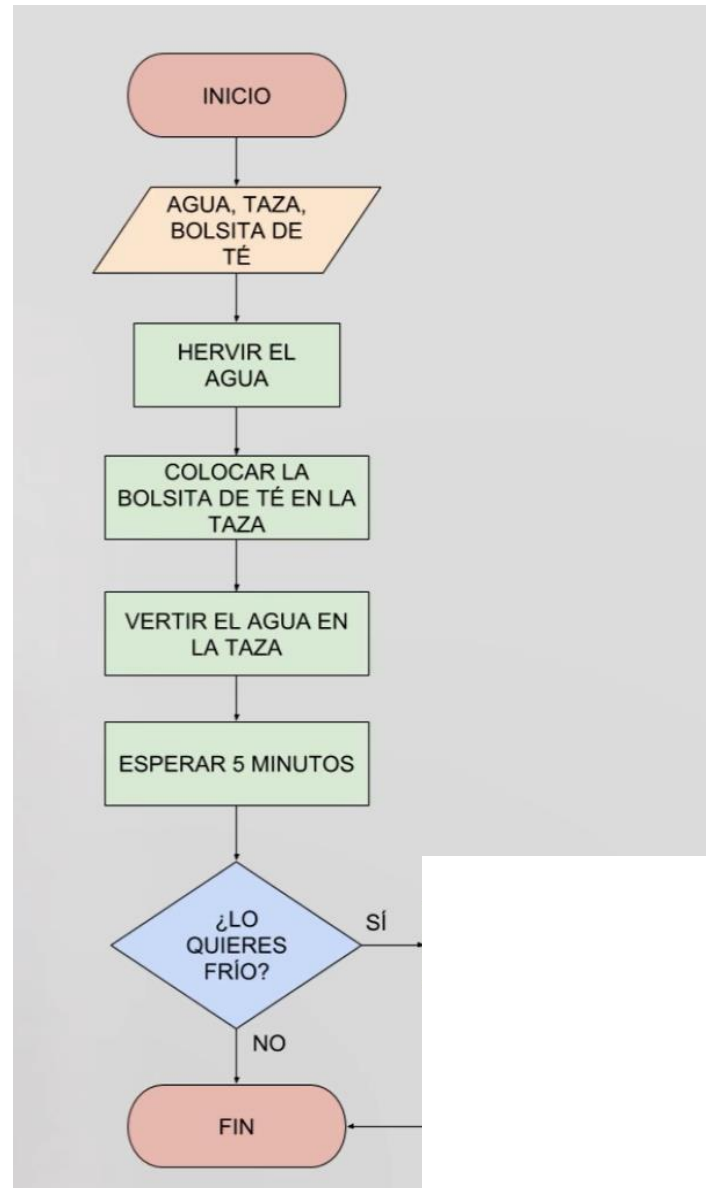
Conceptos de Programación

Diagrama de Flujo



Conceptos de Programación

Diagrama de Flujo



Conceptos de Programación

Variables.- Es un nombre para representar el valor de un dato.

Ej:

x = 1; y = 2;



Conceptos de Programación

Variables.- Es un nombre para representar el valor de un dato.

Ej:

x = 1; y = 2; resultado;



Conceptos de Programación

Variables.- Es un nombre para representar el valor de un dato.

Ej:

x = 1; y = 2; resultado; **resultado = x + y**



Conceptos de Programación

Variables.- Es un nombre para representar el valor de un dato.

Ej:

x = 1; y = 2; resultado; resultado = x + y

Existen distintos tipos de variables, algunas son:

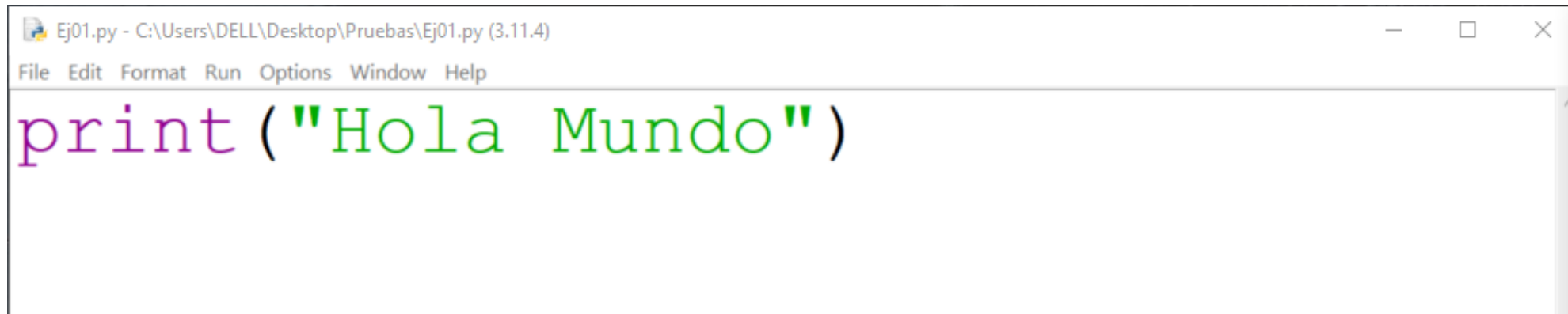
- Entero -> 1 ; 2 ; -8 ; 16 ; 2023 , etc.
- Flotante -> 1,0 ; 5,5 ; - 9,23 , etc.
- Booleanos -> Verdadero y Falso
- Cadenas o String -> "hola" ; "Juan"



Primer programa con Python

La función `print()`

Cada uno de los elementos anteriores juega un papel muy importante en el código.

A screenshot of a Python IDE window. The title bar reads 'Ej01.py - C:\Users\DELL\Desktop\Pruebas\Ej01.py (3.11.4)'. The menu bar includes 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The code editor contains a single line of Python code: `print("Hola Mundo")`. The word `print` is highlighted in purple, and the string `"Hola Mundo"` is highlighted in green.

```
print("Hola Mundo")
```



Primer programa con Python

La función `print()`

La palabra `print` que puedes ver aquí es el nombre de una **función**.

Una función (en este contexto) es una parte separada del código de computadora el cual es capaz de:

- **Causar algún efecto**
- **Evaluar un valor**

El nombre de la función debe ser significativo (el nombre de la función `print` es evidente), imprime en la terminal.

A screenshot of a Python IDE window. The title bar reads "Ej01.py - C:\Users\DELL\Desktop\Pruebas\Ej01.py (3.11.4)". The menu bar includes "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The code editor displays the line `print("Hola Mundo")` with syntax highlighting: `print` is in purple, the opening parenthesis is in green, and the string `"Hola Mundo"` is in green. The closing parenthesis is in purple.

```
Ej01.py - C:\Users\DELL\Desktop\Pruebas\Ej01.py (3.11.4)
File Edit Format Run Options Window Help
print("Hola Mundo")
```



Primer programa con Python

La función `print()`

```
Ej01.py - C:\Users\DELL\Desktop\Pruebas\Ej01.py (3.11.4)
File Edit Format Run Options Window Help
print("Hola Mundo")
```

Una función puede tener:

- Un efecto.
- Un resultado.

Las funciones matemáticas usualmente toman un argumento, por ejemplo, $\sin(x)$ toma una x , que es la medida de un ángulo.

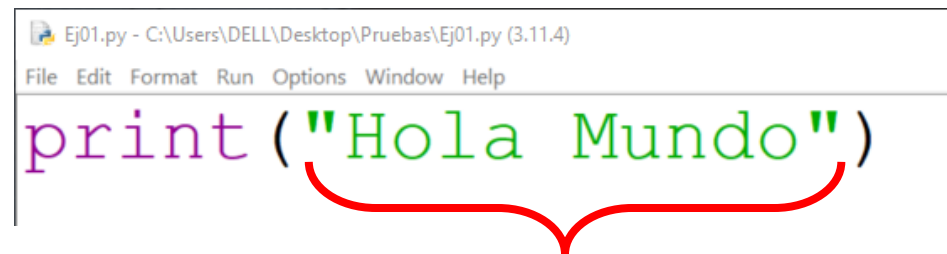
¿La función `print()` en nuestro ejemplo tiene algún argumento?



Primer programa con Python

La función `print()`

El único argumento entregado a la función `print()` en este ejemplo es una cadena:



```
Ej01.py - C:\Users\DELL\Desktop\Pruebas\Ej01.py (3.11.4)
File Edit Format Run Options Window Help
print("Hola Mundo")
```

The screenshot shows a Python IDE window titled 'Ej01.py - C:\Users\DELL\Desktop\Pruebas\Ej01.py (3.11.4)'. The menu bar includes 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The code editor displays the line `print("Hola Mundo")`. The word `print` is in purple, the opening parenthesis is in red, the string `"Hola Mundo"` is in green, and the closing parenthesis is in red. A red curly bracket is drawn under the string `"Hola Mundo"` to highlight it as the argument.

La cadena está delimitada por comillas, las comillas forman la cadena, recortan una parte del código y le asignan un significado diferente.



Primer programa con Python

La función `print()`

El nombre de la función (`print` en este caso) junto con los paréntesis y los argumentos, forman la invocación de la función.



```
Ej01.py - C:\Users\DELL\Desktop\Pruebas\Ej01.py (3.11.4)
File Edit Format Run Options Window Help
print("Hola Mundo")
```



Primer programa con Python

La función `print()`

¿Qué sucede cuando Python encuentra una invocación como la que está a continuación?

```
nombre_función(argumento)
```

- Primero, Python comprueba si el nombre especificado es legal.
- En segundo lugar, Python comprueba si los requisitos de la función para el número de argumentos le permiten invocar la función.
- Tercero, Python deja el código por un momento.
- Cuarto, la función ejecuta el código.
- Finalmente, Python regresa al código



Primer programa con Python

Ejercicio

Objetivos

- Familiarizarse con la función `print()` y sus capacidades de formato.
- Experimentar con el código de Python.

Escenario

El comando `print()`, el cual es una de las directivas más sencillas de Python, simplemente imprime una línea de texto en la pantalla.

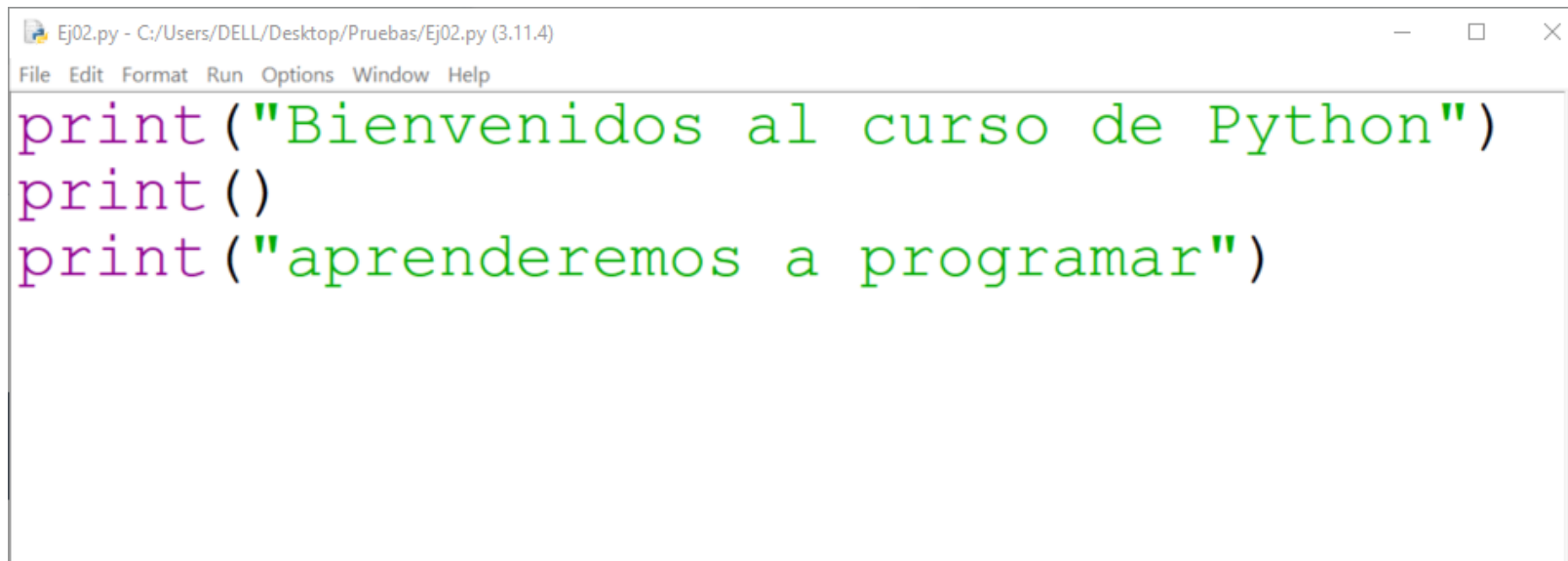
En tu primer laboratorio:

- Utiliza la función `print()` para imprimir la línea `"¡Hola, Mundo!"` en la pantalla.
- Una vez hecho esto, utiliza la función `print()` nuevamente, pero esta vez imprime tu nombre.
- Elimina las comillas dobles y ejecuta el código. Observa la reacción de Python. ¿Qué tipo de error se produce?
- Luego, elimina los paréntesis, vuelve a poner las comillas dobles y vuelve a ejecutar el código. ¿Qué tipo de error se produce esta vez?
- Experimenta tanto como puedas. Cambia las comillas dobles a comillas simples, utiliza múltiples funciones `print()` en la misma línea y luego en líneas diferentes. Observa que es lo que ocurre.



Primer programa con Python

La función **print()** - invocación vacía



```
Ej02.py - C:/Users/DELL/Desktop/Pruebas/Ej02.py (3.11.4)
File Edit Format Run Options Window Help
print("Bienvenidos al curso de Python")
print()
print("aprenderemos a programar")
```



Primer programa con Python

La función `print()` - utilizando argumentos múltiples

- Una función `print()` invocada con más de un argumento genera la salida en una sola línea.
- La función `print()` coloca un espacio entre los argumentos emitidos por iniciativa propia.

Ej02.py - C:/Users/DELL/Desktop/Pruebas/Ej02.py (3.11.4)

File Edit Format Run Options Window Help

```
print("Bienvenido", "al curso de Python", "aprenderemos a programar")
```

IDLE Shell 3.11.4

File Edit Shell Debug Options Window Help

```
>>>
```

```
= RESTART: C:/Users/DELL/Desktop/Pruebas/Ej02.py
```

```
Bienvenido al curso de Python aprenderemos a programar
```

```
>>>
```



Primer programa con Python

La función `print()` - los argumentos de palabra clave

- Una función `print()` invocada con más de un argumento genera la salida en una sola línea.
- La función `print()` coloca un espacio entre los argumentos emitidos por iniciativa propia.

```
*Ej02.py - C:/Users/DELL/Desktop/Pruebas/Ej02.py (3.11.4)*
File Edit Format Run Options Window Help
print("Bienvenido", "al curso de Python", end = " |")
print("aprenderemos a programar")
```

```
IDLE Shell 3.11.4
File Edit Shell Debug Options Window Help
>>>
===== RESTART: C:/Users/DELL/Desktop/Pruebas/Ej02.py =====
Bienvenido al curso de Pythonaprenderemos a programar
>>>
```

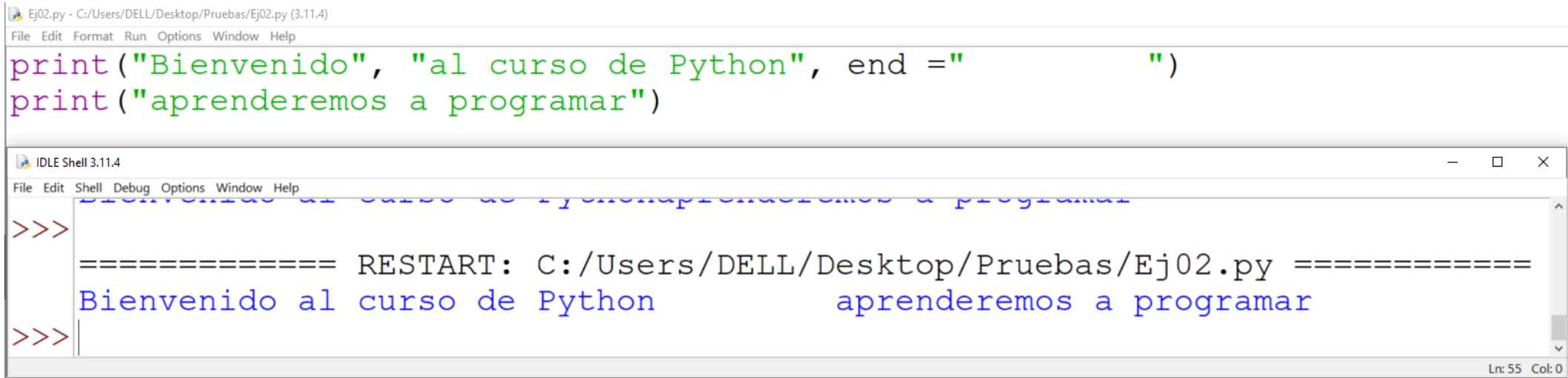


Primer programa con Python

La función `print()` - los argumentos de palabra clave

Observando vemos que se ha utilizado el argumento ***end***, pero su cadena asignada está vacía (no contiene ningún carácter).

Ya que al argumento ***end*** se le ha asignado a nada, la función `print()` tampoco genera nada, una vez que se hayan agotado los argumentos posicionales.



The screenshot shows the Python IDLE environment. The top window, titled 'Ej02.py - C:/Users/DELL/Desktop/Pruebas/Ej02.py (3.11.4)', contains the following code:

```
print("Bienvenido", "al curso de Python", end = " ")
print("aprenderemos a programar")
```

The bottom window, titled 'IDLE Shell 3.11.4', shows the output of the script after execution. It displays a 'RESTART' message followed by the printed text on two lines:

```
>>>
===== RESTART: C:/Users/DELL/Desktop/Pruebas/Ej02.py =====
Bienvenido al curso de Python      aprenderemos a programar
>>>
```

The status bar at the bottom right of the shell window indicates 'Ln: 55 Col: 0'.

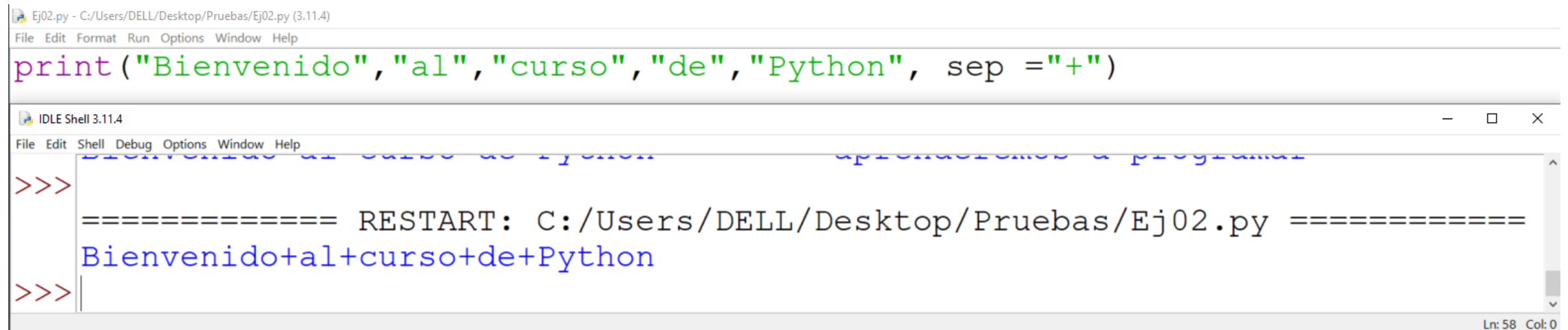


Primer programa con Python

La función `print()` - los argumentos de palabra clave

Se estableció anteriormente que la función `print()` separa los argumentos generados con espacios. Este comportamiento también puede ser cambiado.

El argumento de palabra clave que puede hacer esto se denomina ***sep*** (separador).



```
Ej02.py - C:/Users/DELL/Desktop/Pruebas/Ej02.py (3.11.4)
File Edit Format Run Options Window Help

print("Bienvenido", "al", "curso", "de", "Python", sep="+")

IDLE Shell 3.11.4
File Edit Shell Debug Options Window Help
>>> Bienvenido+al+curso+de+Python
>>>

===== RESTART: C:/Users/DELL/Desktop/Pruebas/Ej02.py =====
Bienvenido+al+curso+de+Python
>>>

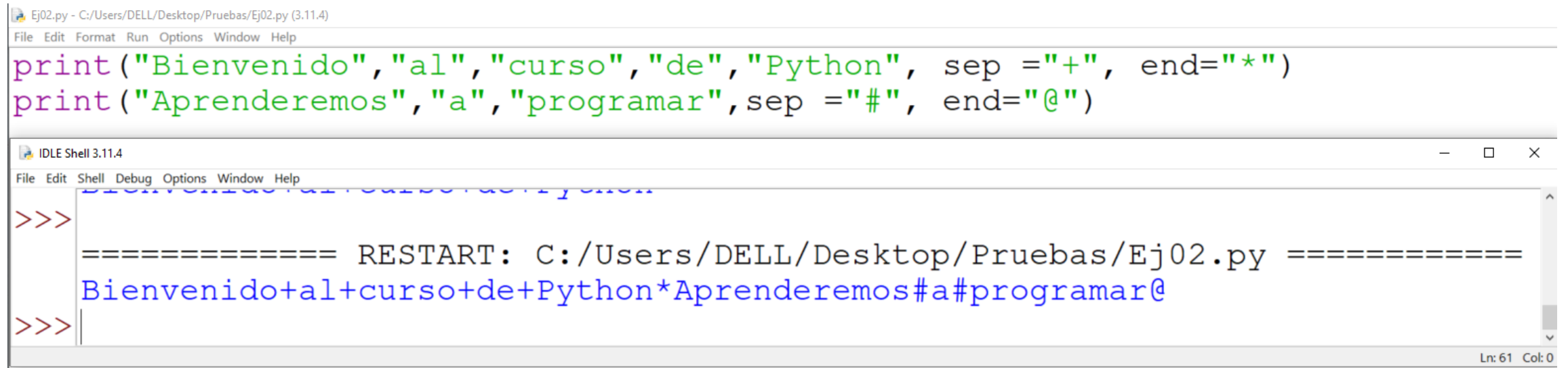
Ln: 58 Col: 0
```



Primer programa con Python

La función `print()` - los argumentos de palabra clave

Combinando *end* y *sep* en una misma invocación



The screenshot displays a Python IDE with two windows. The top window, titled 'Ej02.py', contains the following Python code:

```
print("Bienvenido", "al", "curso", "de", "Python", sep="+", end="*")
print("Aprenderemos", "a", "programar", sep="#", end="@")
```

The bottom window, titled 'IDLE Shell 3.11.4', shows the execution of this code. It displays a 'RESTART' message followed by the output of the two print statements:

```
>>>
===== RESTART: C:/Users/DELL/Desktop/Pruebas/Ej02.py =====
Bienvenido+al+curso+de+Python*Aprenderemos#a#programar@
>>>
```

The status bar at the bottom right of the shell window indicates 'Ln: 61 Col: 0'.



Primer programa con Python

Literales en Python

Un literal se refiere a datos cuyos valores están determinados por el literal mismo.

Debido a que es un concepto un poco difícil de entender, un buen ejemplo puede ser muy útil.

123

c

m

Se utilizan literales para codificar datos y ponerlos dentro del código.

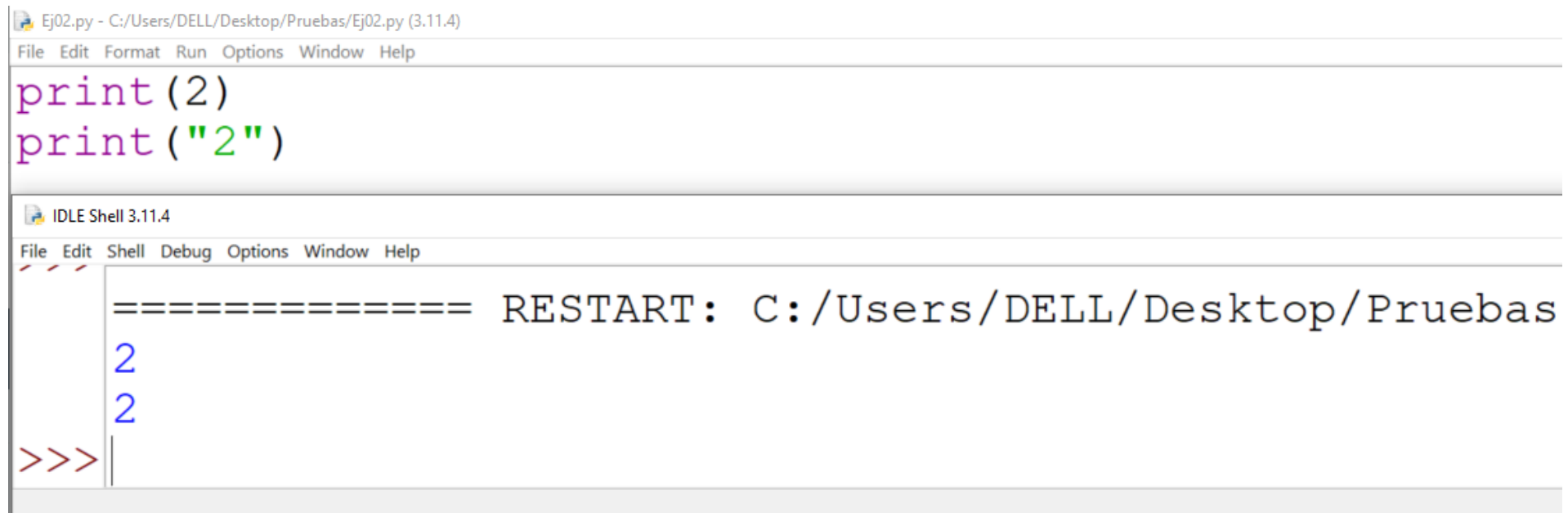


Primer programa con Python

Literales en Python

A través de este ejemplo, encuentras dos tipos diferentes de literales:

- Una **cadena**, la cual ya conoces.
- Y un **número entero**, algo completamente nuevo.



The screenshot displays the Python IDLE Shell interface. The top window, titled 'Ej02.py - C:/Users/DELL/Desktop/Pruebas/Ej02.py (3.11.4)', contains the following code:

```
print(2)
print("2")
```

The bottom window, titled 'IDLE Shell 3.11.4', shows the output of the program after a restart. The prompt '==== RESTART: C:/Users/DELL/Desktop/Pruebas' is followed by two lines of output, both displaying the number '2' in blue. The prompt '>>>' is visible at the bottom left of the shell window.



Primer programa con Python

Enteros

Los números manejados por las computadoras modernas son de dos tipos:

- Enteros, es decir, aquellos que no tienen una parte fraccionaria.
- Y números punto-flotantes, los cuales contienen una parte fraccionaria.



Primer programa con Python

Enteros

El número once millones ciento once mil ciento once. Si tomarías un lápiz en tu mano, escribirías el siguiente número: **11,111,111** , o así: **11.111.111** , incluso de esta manera: **11 111 111**.

Es claro que la separación hace que sea más fácil de leer, especialmente cuando el número tiene demasiados dígitos. Sin embargo, Python no acepta estas cosas. Está prohibido. ¿Qué es lo que Python permite? El uso de guion bajo en los literales numéricos.

Por lo tanto, el número se puede escribir ya sea así: **11111111**, o como sigue: **11_111_111**



Primer programa con Python

Flotantes

Son números que tienen (o pueden tener) una parte fraccionaria después del punto decimal.

Cuando se usan términos como dos y medio o menos cero punto cuatro, pensamos en números que la computadora considera como números punto-flotante:

2.5

-0.4

No se deben usar comas

-.4



-0.4

12



12.0



Primer programa con Python

Enteros frente a Flotantes

2.0

2

Se puede pensar que son idénticos, pero Python los ve de una manera completamente distinta.

- **2** es un número entero, mientras que **2.0** es un número punto-flotante.

La velocidad de la luz, expresada en metros por segundo. Escrita directamente se vería de la siguiente manera: **3000000000**.

Para evitar escribir tantos ceros, los libros de texto emplean la forma abreviada, la cual probablemente hayas visto: **3 x 10⁸**.

En Python, el mismo efecto puede ser logrado de una manera similar, observa lo siguiente:

3E8



Primer programa con Python

Cadenas

Las cadenas se emplean cuando se requiere procesar texto (como nombres de cualquier tipo, direcciones, novelas, etc.), no números.

“Hola como estas?”

Cual seria la instrucción para que la respuesta de Python sea:

Estamos aprendiendo “Python”



Primer programa con Python

Valores Booleanos

Para comprender bien el funcionamiento de los booleanos se podría partir de la premisa que cada vez que se le pregunta a Python si un número es más grande que otro, el resultado es la creación de un tipo de dato muy específico - un **valor booleano**.

- Álgebra Booleana - una parte del álgebra que hace uso de dos valores: **Verdadero** y **Falso**, denotados como **1** y **0**.

Un programador escribe un programa, y el programa hace preguntas. Python ejecuta el programa, y provee las respuestas. El programa debe ser capaz de reaccionar acorde a las respuestas recibidas.

True

False



Primer programa con Python

Objetivos

- Familiarizarse con la función `print()` y sus capacidades de formato.
- Practicar el codificar cadenas.
- Experimentar con el código de Python.

Escenario

Escribe una sola línea de código, utilizando la función `print()`, así como los caracteres de nueva línea y escape, para obtener la salida esperada de tres líneas.

Salida Esperada

```
"Estoy"  
""aprendiendo""  
"""Python"""
```

salida



Operadores: herramientas para la manipulación de datos

Python como una calculadora

Python es capaz de ejecutar los operadores mas básicos y/o conocidos, sobre todo aquellas que están asociados con las operaciones aritméticas más conocidas:



Cuando los datos y operadores se unen, forman juntos expresiones. La expresión más sencilla es el **literal**.

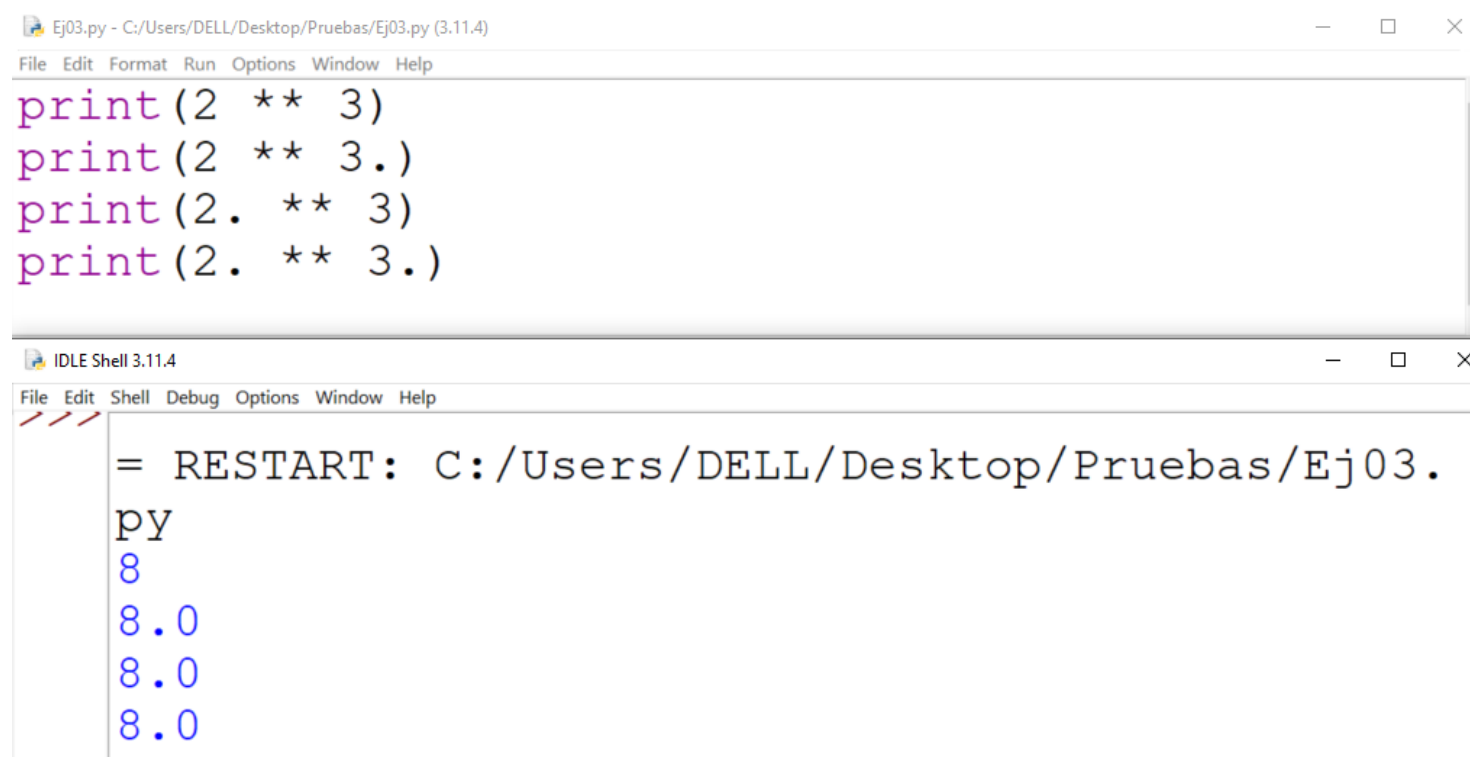


Operadores: herramientas para la manipulación de datos

Operadores Aritméticos: exponenciación

Un signo de `**` (doble asterisco) es un operador de exponenciación (potencia). El argumento a la izquierda es la base, el de la derecha, el exponente.

$$2^3 \longrightarrow 2^{**}3$$



The screenshot displays the Python IDLE 3.11.4 interface. The top window, titled 'Ej03.py', contains the following Python code:

```
print(2 ** 3)
print(2 ** 3.)
print(2. ** 3)
print(2. ** 3.)
```

The bottom window, titled 'IDLE Shell 3.11.4', shows the output of the script after execution. It begins with a restart message and then displays the results of the four print statements:

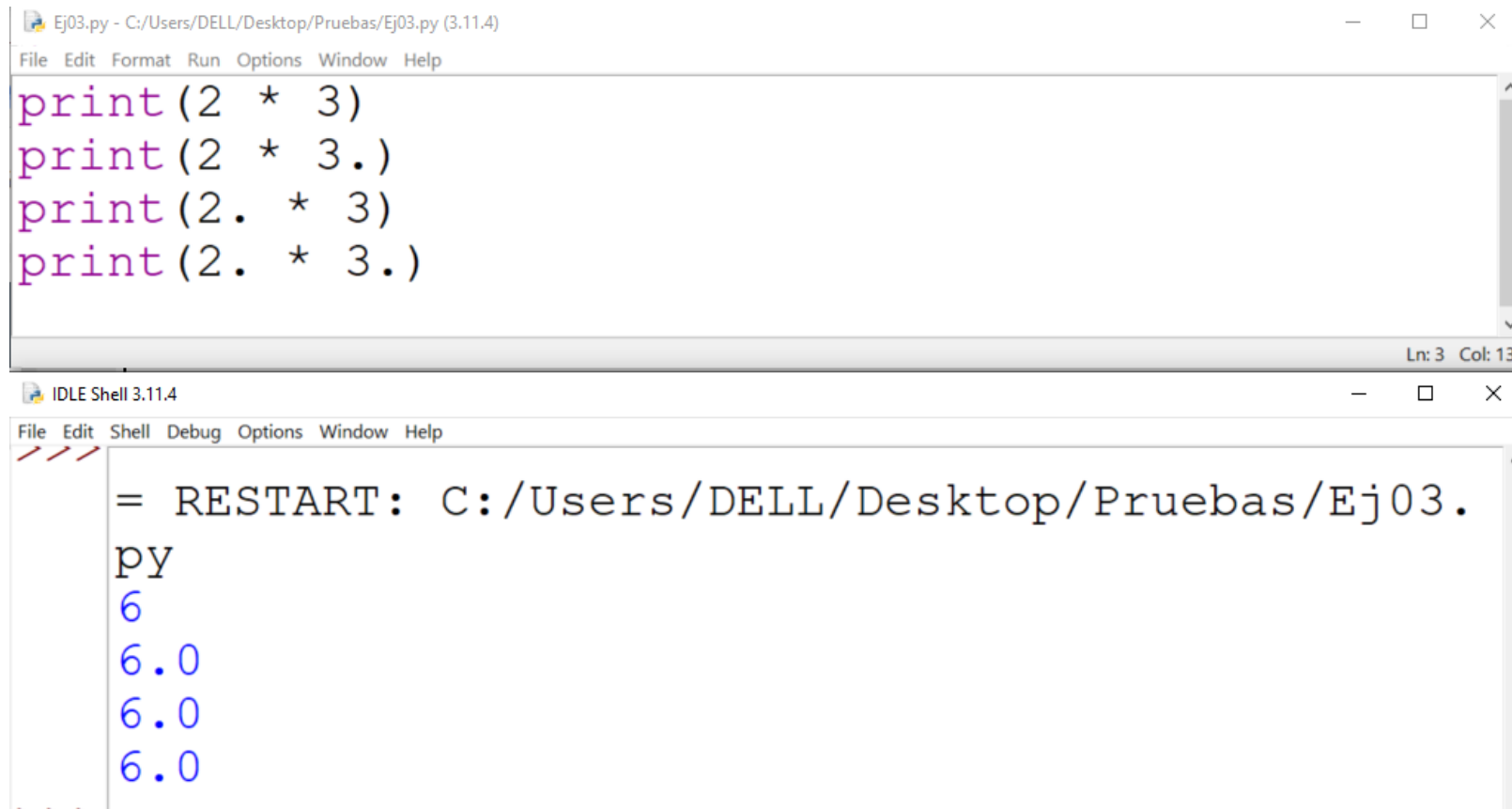
```
= RESTART: C:/Users/DELL/Desktop/Pruebas/Ej03.py
8
8.0
8.0
8.0
```



Operadores: herramientas para la manipulación de datos

Operadores Aritméticos: multiplicación

En símbolo de * (asterisco) es un operador de multiplicación.



The screenshot displays the Python IDLE environment. The top window, titled 'Ej03.py - C:/Users/DELL/Desktop/Pruebas/Ej03.py (3.11.4)', contains the following Python code:

```
print(2 * 3)
print(2 * 3.)
print(2. * 3)
print(2. * 3.)
```

The bottom window, titled 'IDLE Shell 3.11.4', shows the output of the code after execution. It begins with a restart message and then displays the results of the four print statements:

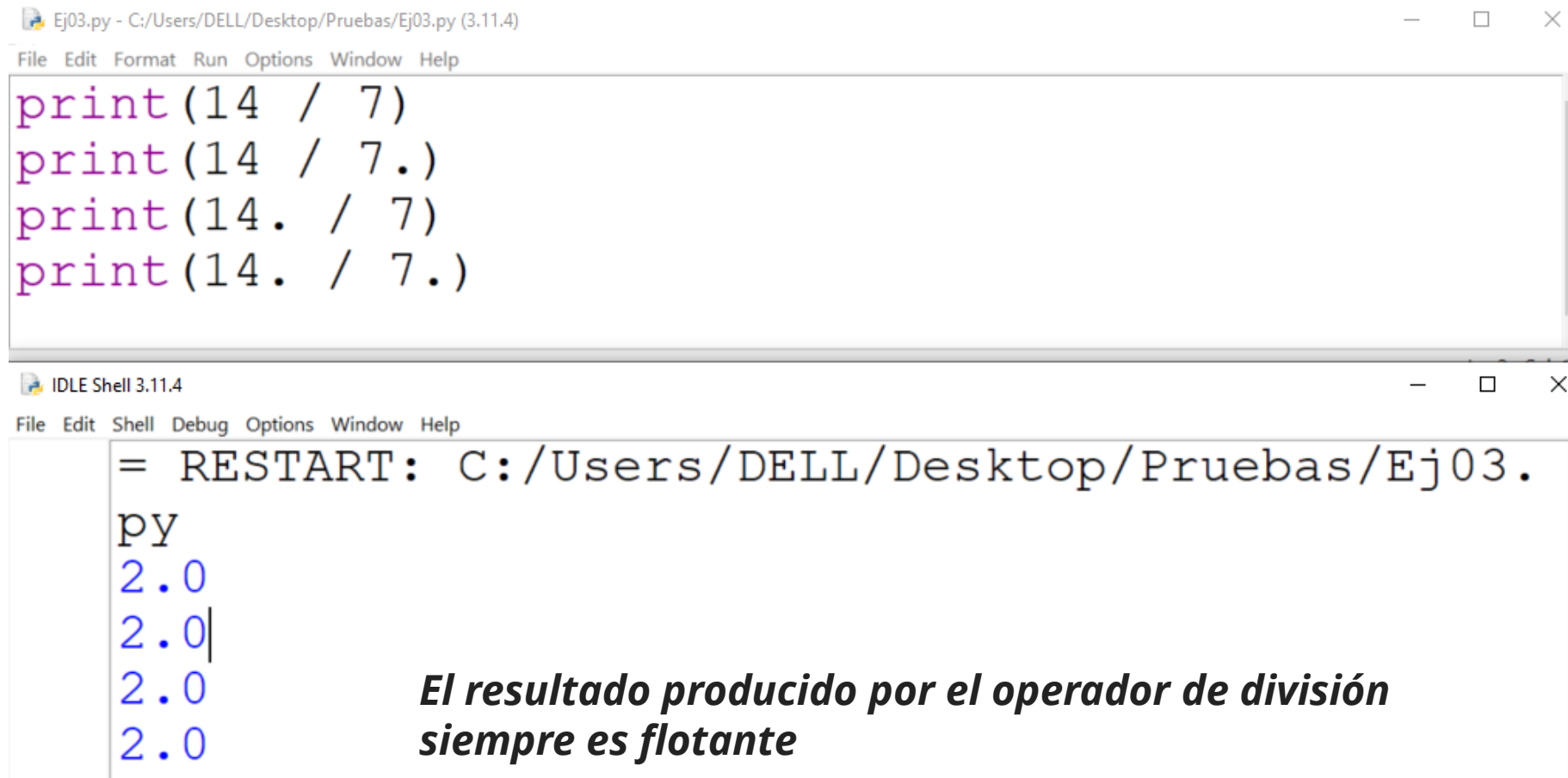
```
= RESTART: C:/Users/DELL/Desktop/Pruebas/Ej03.py
6
6.0
6.0
6.0
```



Operadores: herramientas para la manipulación de datos

Operadores Aritméticos: división

Un símbolo de / (diagonal) es un operador de división.



The screenshot displays two windows from the Python IDLE 3.11.4 application. The top window, titled 'Ej03.py - C:/Users/DELL/Desktop/Pruebas/Ej03.py (3.11.4)', contains the following Python code:

```
print(14 / 7)
print(14 / 7.)
print(14. / 7)
print(14. / 7.)
```

The bottom window, titled 'IDLE Shell 3.11.4', shows the output of the script after execution. It begins with a restart message and then displays the results of the four print statements, all of which are the float value 2.0.

```
= RESTART: C:/Users/DELL/Desktop/Pruebas/Ej03.py
2.0
2.0
2.0
2.0
```

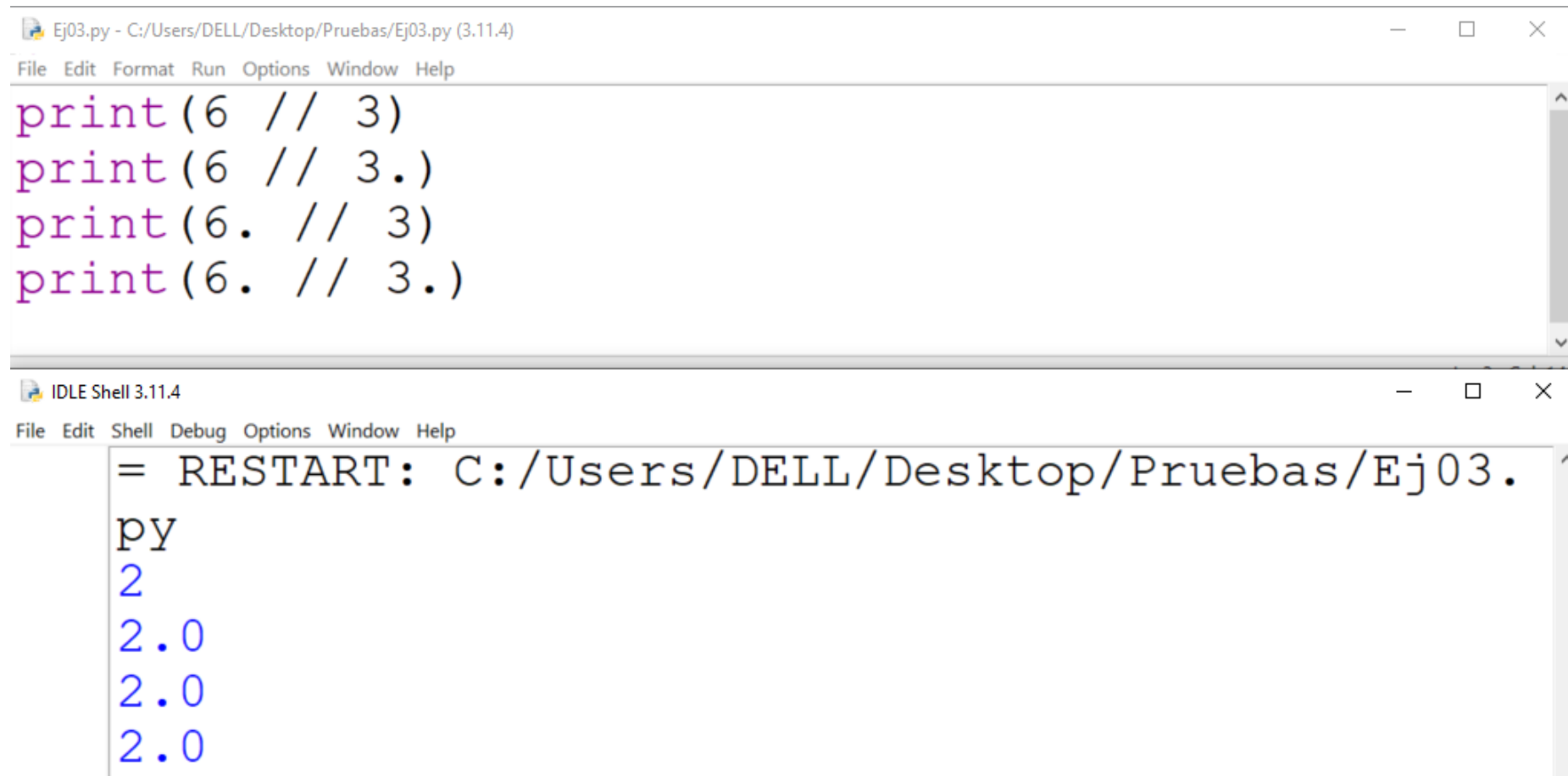
El resultado producido por el operador de división siempre es flotante



Operadores: herramientas para la manipulación de datos

Operadores Aritméticos: división entera

Un símbolo de // (doble diagonal) es un operador de división entera.



```
Ej03.py - C:/Users/DELL/Desktop/Pruebas/Ej03.py (3.11.4)
File Edit Format Run Options Window Help
print(6 // 3)
print(6 // 3.)
print(6. // 3)
print(6. // 3.)

IDLE Shell 3.11.4
File Edit Shell Debug Options Window Help
= RESTART: C:/Users/DELL/Desktop/Pruebas/Ej03.py
2
2.0
2.0
2.0
```



Operadores: herramientas para la manipulación de datos

Operadores: residuo (módulo)

Su representación gráfica en Python es el símbolo de % (porcentaje), lo cual puede ser un poco confuso. El resultado que genera no es nada mas que el **residuo** de una división.

$$\begin{array}{r|l} 15 & 6 \\ \hline 3 & 2 \end{array}$$

A yellow arrow points from the number 15 to the number 3.



The screenshot shows a Python IDLE Shell window. The title bar reads "Ej03.py - C:/Users/DELL/Desktop/Pruebas/Ej03.py (3.11.4)". The menu bar includes "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The code editor contains the line `print(15%6)`. Below the editor is the "IDLE Shell 3.11.4" window, which has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The shell prompt is `>>>`. The output shows a restart message: `= RESTART: C:/Users/DELL/Desktop/Pruebas/Ej03.py`, followed by the result `3`.



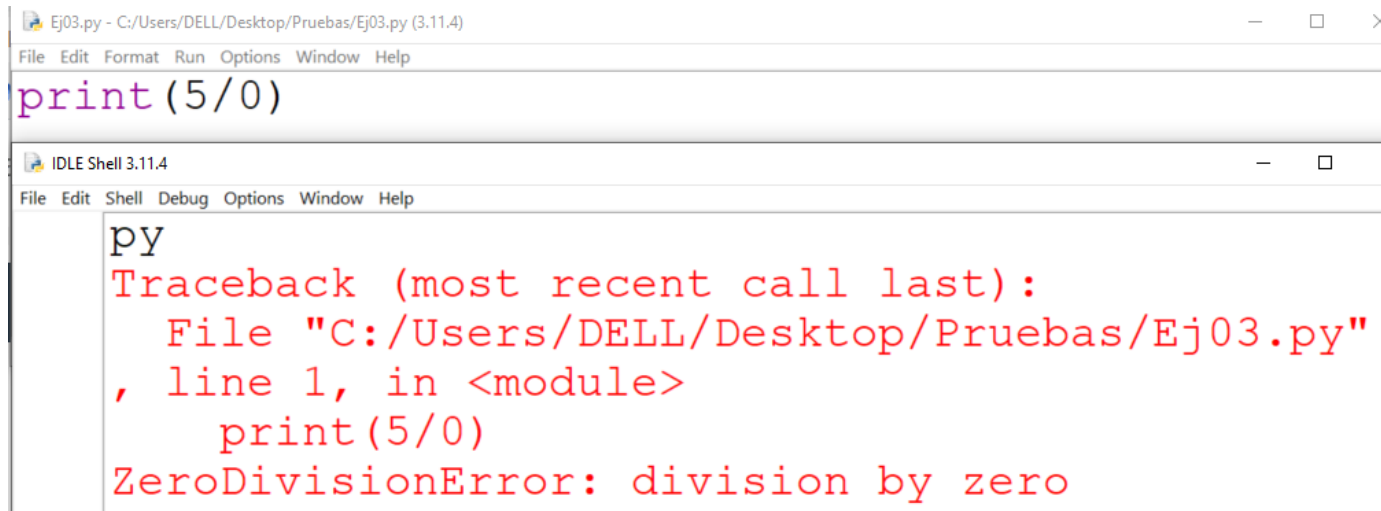
Operadores: herramientas para la manipulación de datos

Operadores Aritméticos: como no dividir

La división entre cero no funciona, no se debe intentar:

- Dividir entre cero.
- Realizar una división entera entre cero.
- Encontrar el residuo de una división entre cero.

El intentarlo derivara, no en un resultado erróneo, si no en un error a nivel del código.



The screenshot shows a Python IDLE Shell window. The top window is titled 'Ej03.py - C:/Users/DELL/Desktop/Pruebas/Ej03.py (3.11.4)' and contains the code `print(5/0)`. The bottom window is titled 'IDLE Shell 3.11.4' and displays a traceback error message in red text:

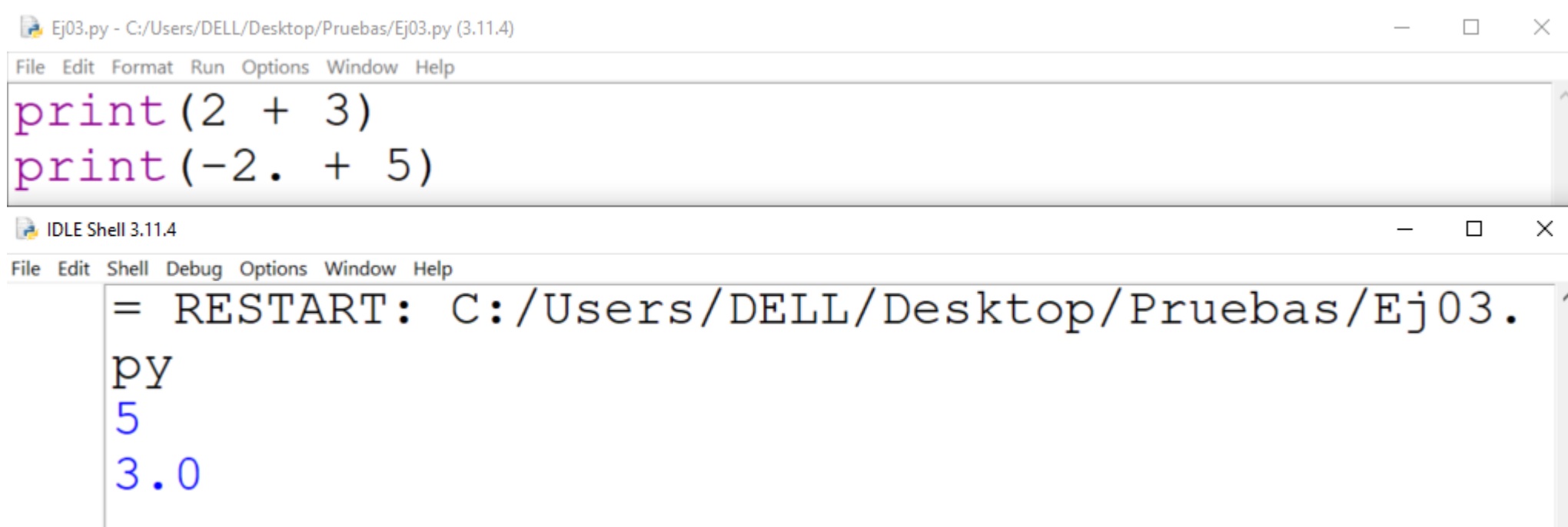
```
py
Traceback (most recent call last):
  File "C:/Users/DELL/Desktop/Pruebas/Ej03.py",
    line 1, in <module>
      print(5/0)
ZeroDivisionError: division by zero
```



Operadores: herramientas para la manipulación de datos

Operadores: suma

El símbolo del operador de suma es el + (signo de más), el cual esta completamente alineado a los estándares matemáticos.



```
Ej03.py - C:/Users/DELL/Desktop/Pruebas/Ej03.py (3.11.4)
File Edit Format Run Options Window Help
print(2 + 3)
print(-2. + 5)

IDLE Shell 3.11.4
File Edit Shell Debug Options Window Help
= RESTART: C:/Users/DELL/Desktop/Pruebas/Ej03.py
5
3.0
```



Operadores: herramientas para la manipulación de datos

El operador de resta, operadores unarios y binarios

El símbolo del operador de resta es obviamente - (el signo de menos), sin embargo debes notar que este operador tiene otra función - **puede cambiar el signo de un número.**



The screenshot displays the Python IDLE 3.11.4 environment. The top window, titled 'Ej03.py - C:/Users/DELL/Desktop/Pruebas/Ej03.py (3.11.4)', contains the following Python code:

```
print(-4 - 4)
print(4. - 8)
print(-1.1)
```

The bottom window, titled 'IDLE Shell 3.11.4', shows the output of the script after execution. It begins with a restart message and then displays the results of the three print statements:

```
= RESTART: C:/Users/DELL/Desktop/Pruebas/Ej03.py
-8
-4.0
-1.1
```



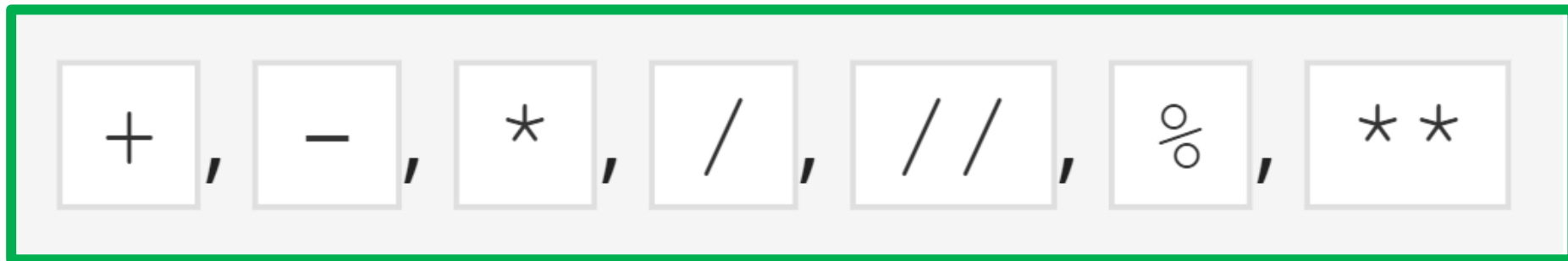
Operadores: herramientas para la manipulación de datos

Operadores y sus prioridades

Consideremos la siguiente expresión:

$$2 + 3 * 5$$

El fenómeno que causa que algunos operadores actúen antes que otros es conocido como la **jerarquía de prioridades**.



Operadores: herramientas para la manipulación de datos

Operadores y paréntesis

Se permite hacer uso de paréntesis, lo cual cambiará el orden natural del cálculo de la operación.

De acuerdo con las reglas aritméticas, las sub-expresiones dentro de los paréntesis siempre se calculan primero.

```
print ((5 * ((25 % 13) + 100) / (2 * 13)) // 2)
print ((5 * (( 12 ) + 100) / ( 26 )) // 2)
print ((5 * ( 12 + 100) / 26 ) // 2)
print ((5 * ( 112 ) / 26 ) // 2)
print ( 560 / 26 ) // 2)
print ( 21.5384615384615 ) // 2)
print ( 10.0 )
```



Variables: cajas en forma de datos

¿Qué son las Variables?

Python permite codificar literales, las cuales contengan valores numéricos y cadenas, permite operaciones aritméticas con estos números: sumar, restar, etc.

La pregunta es: cómo es que se pueden **almacenar los resultados** de estas operaciones, para poder emplearlos en otras operaciones, y así sucesivamente.



Variables: cajas en forma de datos

¿Qué son las Variables?

Los componentes o elementos de una variable:

- Un nombre.
- Un valor (el contenido del contenedor).

Las variables no aparecen en un programa automáticamente. Como desarrollador, tu debes decidir cuantas variables deseas utilizar en tu programa., se las debes de nombrar.



Variables: cajas en forma de datos

¿Qué son las Variables?

Si se desea nombrar una variable, se deben seguir las siguientes reglas:

- El nombre de la variable debe de estar compuesto por MAYÚSCULAS, minúsculas, dígitos, y el carácter _ (guion bajo).
- El nombre de la variable debe comenzar con una letra.
- El carácter guion bajo es considerado una letra.
- Las mayúsculas y minúsculas se tratan de forma distinta (un poco diferente que en el mundo real - Alicia y ALICIA son el mismo nombre, pero en Python son dos nombres de variable distintos, subsecuentemente, son dos variables diferentes).
- El nombre de las variables no pueden ser igual a alguna de las palabras reservadas de Python (se explicará más de esto pronto).

Alicia ≠ alicia ≠ ALICIA



Variables: cajas en forma de datos

Nombres correctos e incorrectos de variables

Python no impone restricciones en la longitud de los nombres de las variables, pero eso no significa que un nombre de variable largo sea mejor que uno corto. El nombre de la variable debe estar compuesto por MAYÚSCULAS, minúsculas, dígitos, y el carácter _ (guion bajo).

MiVariable

i

T34

Tasa_Cambio

Contador

dias_para_navidad

ElNombreEsTanLargoQueSeCometeranErroresConEl

Mi Variable

34C



Variables: cajas en forma de datos

Palabras Clave

Son llamadas palabras clave o (mejor dicho) palabras reservadas. Son reservadas porque no se deben utilizar como nombres: ni para variables, ni para funciones, ni para cualquier otra cosa que se desee crear.

```
['False', 'None', 'True', 'and', 'as',  
'assert', 'break', 'class', 'continue',  
'def', 'del', 'elif', 'else', 'except',  
'finally', 'for', 'from', 'global', 'if',  
'import', 'in', 'is', 'lambda', 'nonlocal',  
'not', 'or', 'pass', 'raise', 'return',  
'try', 'while', 'with', 'yield']
```



Variables: cajas en forma de datos

Creando variables

Se puede utilizar una variable para almacenar cualquier tipo de los valores que ya se han mencionado, y muchos mas de los cuales aun no se han explicado.

Una variable se crea cuando se le asigna un valor. A diferencia de otros lenguajes de programación, no es necesario declararla.

Puedes colocar
CUALQUIER
COSA
dentro de la variable



Variables: cajas en forma de datos

Creando variables

La creación es muy simple: solo utiliza el nombre de la variable deseada, después el signo de igual (=) y el valor que se desea colocar dentro de la variable.

Consiste de dos simples instrucciones:

- La primera crea una variable llamada var, y le asigna un literal con un valor entero de 1.
- La segunda imprime el valor de la variable recientemente creada en la consola.

```
var = 1  
print(var)
```



Variables: cajas en forma de datos

Utilizando variables

Se tiene permitido utilizar cuantas declaraciones de variables sean necesarias para lograr el objetivo del programa, por ejemplo:

```
moneda = 'Bs'  
Cuenta_bancaria = 1000.0  
Nombre_cliente = 'Pedro Perez'  
print(Nombre_cliente, Cuenta_bancaria, moneda)  
print(Cuenta_bancaria)
```



Variables: cajas en forma de datos

Utilizando variables

Sin embargo, no se permite utilizar una variable que no exista, (en otras palabras, una variable a la cual no se le ha dado un valor).

Este ejemplo ocasionará un error:

```
Nombre_cliente = 'Pedro Perez'  
print(Nombre_Cliente)
```

Se ha tratado de utilizar la variable llamada **Nombre_Cliente**, la cual no tiene ningún valor.

Nota: Nombre_Cliente y Nombre_cliente son entidades diferentes, y no tienen nada en común dentro de Python).



Variables: cajas en forma de datos

Utilizando variables

Para combinar variables con cadenas es posible concatenar utilizando el operador `+` o también una coma.

```
moneda = 'Bs'
Cuenta_bancaria = 1000.0
Nombre_cliente = 'Pedro Perez'
print(Nombre_cliente , 'tiene' , Cuenta_bancaria, moneda)
```



Variables: cajas en forma de datos

Asignar un valor nuevo a una variable ya existente

El signo de igual es de hecho un operador de asignación. Aunque esto suene un poco extraño, el operador tiene una sintaxis simple y una interpretación clara y precisa.

```
var = 1
print(var)
var = var + 1
print(var)
```

```
var = 100
var = 200 + 300
print(var)
```



Variables: cajas en forma de datos

Resolviendo problemas matemáticos simples

Es posible resolver problemas matemáticos sencillos como el Teorema de Pitágoras:

El cuadrado de la hipotenusa es igual a la suma de los cuadrados de los dos catetos.

$$\sqrt{x} = x^{(1/2)}$$

y

$$c = \sqrt{a^2 + b^2}$$



Variables: cajas en forma de datos

Resolviendo problemas matemáticos simples

Es posible resolver problemas matemáticos sencillos como el Teorema de Pitágoras:

El cuadrado de la hipotenusa es igual a la suma de los cuadrados de los dos catetos.

El siguiente código evalúa la longitud de la hipotenusa (es decir, el lado más largo de un triángulo rectángulo, el opuesto al ángulo recto) utilizando el Teorema de Pitágoras:

```
a = 3.0
b = 4.0
c = (a ** 2 + b ** 2) ** 0.5
print("c =", c)
```



Variables: cajas en forma de datos

Escenario

A continuación una historia:

Érase una vez en la Tierra de las Manzanas, Juan tenía tres manzanas, María tenía cinco manzanas, y Adán tenía seis manzanas. Todos eran muy felices y vivieron por muchísimo tiempo. Fin de la Historia.

Tu tarea es:

- Crear las variables: `juan`, `maria`, y `adan`.
- Asignar valores a las variables. El valor debe de ser igual al número de manzanas que cada quien tenía.
- Una vez almacenados los números en las variables, imprimir las variables en una línea, y separar cada una de ellas con una coma.
- Después se debe crear una nueva variable llamada `total_manzanas` y se debe igualar a la suma de las tres variables anteriores.
- Imprime el valor almacenado en `total_manzanas` en la consola.
- **Experimenta con tu código:** crea nuevas variables, asigna diferentes valores a ellas, y realiza varias operaciones aritméticas con ellas (por ejemplo, `+`, `-`, `*`, `/`, `//`, etc.). Intenta poner una cadena con un entero juntos en la misma línea, por ejemplo, `"Número Total de Manzanas:"` y `total_manzanas`.

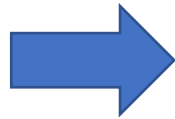
Variables: cajas en forma de datos

Operadores Abreviados

Muy seguido, se desea utilizar la misma variable al lado derecho y al lado izquierdo del operador =.

```
x = x * 2
```

```
hola = hola + 7
```



```
x *= 2
```

```
hola += 7
```

Observemos los siguientes ejemplos.

```
i = i + 2 * j ⇒ i += 2 * j
```

```
var = var / 2 ⇒ var /= 2
```

```
rem = rem % 10 ⇒ rem %= 10
```

```
j = j - (i + var + rem) ⇒ j -= (i + var + rem)
```

```
x = x ** 2 ⇒ x **= 2
```



Variables: cajas en forma de datos

Escenario

Millas y kilómetros son unidades de longitud o distancia.

Teniendo en mente que 1 milla equivale aproximadamente a 1.61 kilómetros, complementa el programa en el editor para que convierta de:

- Millas a kilómetros.
- Kilómetros a millas.

No se debe cambiar el código existente. Escribe tu código en los lugares indicados con `###`. Prueba tu programa con los datos que han sido provistos en el código fuente.

Pon mucha atención a lo que esta ocurriendo dentro de la función `print()`. Analiza como es que se proveen múltiples argumentos para la función, y como es que se muestra el resultado.

Nota que algunos de los argumentos dentro de la función `print()` son cadenas (por ejemplo `"millas son"`), y otros son variables (por ejemplo `miles`).

```
*Ej03.py - C:/Users/DELL/Desktop/Pruebas/Ej03.py (3.11.4)
File Edit Format Run Options Window Help
kilometros = 12.25
millas = 7.38

millas_a_kilometros = ###
kilometros_a_millas = ###

print(millas, "millas son", round(millas_a_kilometros, 2), "kilómetros")
print(kilometros, "kilómetros son", round(kilometros_a_millas, 2), "millas")
```

Resultado Esperado

```
7.38 millas son 11.88 kilómetros
12.25 kilómetros son 7.61 millas
```

salida



Variables: cajas en forma de datos

Escenario

Observa el código en el editor: lee un valor `flotante`, lo coloca en una variable llamada `x`, e imprime el valor de la variable llamada `y`. Tu tarea es completar el código para evaluar la siguiente expresión:

$$3x^3 - 2x^2 + 3x - 1$$

El resultado debe ser asignado a `y`.

Recuerda que la notación algebraica clásica muy seguido omite el operador de multiplicación, aquí se debe de incluir de manera explícita. Nota como se cambia el tipo de dato para asegurarnos de que `x` es del tipo `flotante`.

Mantén tu código limpio y legible, y pruébalo utilizando los datos que han sido proporcionados. No te desanimes por no lograrlo en el primer intento. Se persistente y curioso.

Datos de Prueba

Entrada de Muestra

```
x = 0  
x = 1  
x = -1
```

Salida Esperada

```
y = -1.0  
y = 3.0  
y = -9.0
```

salida



Comentarios

Poner comentarios en el código

Un texto insertado en el programa el cual es, omitido en la ejecución, es denominado un comentario.

Cuando Python se encuentra con un comentario en el programa, el comentario es completamente transparente, desde el punto de vista de Python, el comentario es solo un espacio vacío, sin importar que tan largo sea.

En Python, un comentario es un texto que comienza con el símbolo `#` y se extiende hasta el final de la línea.

```
# Esta programa calcula la hipotenusa (c)
# a y b son las longitudes de los catetos
a = 3.0
b = 4.0
c = (a ** 2 + b ** 2) ** 0.5 # se utiliza ** en lugar de la raíz cuadrada.
print("c =", c) # En esta línea estoy imprimiendo el resultado
```



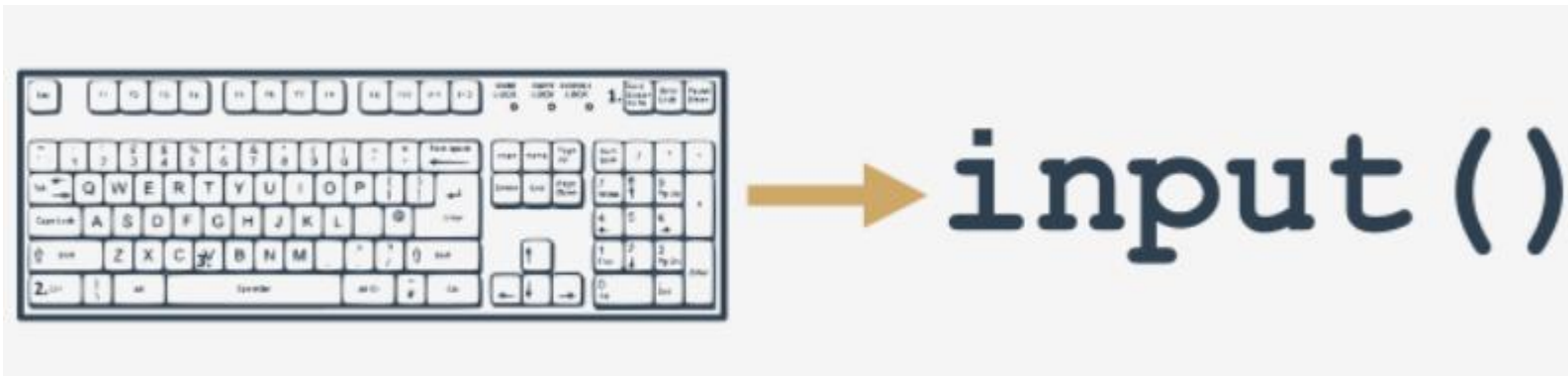
Cómo hablar con una computadora

La función `input()`

Esta nueva función, pareciese un reflejo de la función `print()`. ¿Por qué? Bueno, `print()` envía datos a la consola. Esta nueva función obtiene datos de ella.

`print()` no tiene un resultado utilizable. La importancia de esta nueva función es que **regresa un valor muy utilizable**.

La función **`input()`** es capaz de leer datos que fueron introducidos por el usuario y pasar esos datos al programa en ejecución.



Cómo hablar con una computadora

La función `input()`

```
print("Dime algo...")  
mensaje = input()  
print("Mmm...", mensaje, "...¿en serio?")
```



Cómo hablar con una computadora

La función `input()` con un argumento

- La función `input()` al ser invocada con un argumento, contiene una cadena con un mensaje.
- El mensaje será mostrado en consola antes de que el usuario tenga oportunidad de escribir algo.
- Después de esto `input()` hará su trabajo.

```
mensaje = input("Dime algo...")  
print("Mmm...", mensaje, "...¿En serio?")
```



Cómo hablar con una computadora

El resultado de la función input()

Se ha dicho antes, pero hay que decirlo sin ambigüedades una vez más: el resultado de la función input() es una cadena..

```
anything = input("Inserta un número: ")  
something = anything ** 2.0  
print(anything, "al cuadrado es", something)
```



Cómo hablar con una computadora

Función `input()` y tipos de conversión

Python ofrece dos simples funciones para especificar un tipo de dato y resolver este problema, aquí están: `int()` y `float()`.

Sus nombres indican cual es su función:

- La función **`int()`** toma un argumento
- La función **`float()`** toma un argumento

```
x = float(input("Inserta un número: "))  
resultado = x ** 2.0  
print(x, "al cuadrado es", resultado)
```



Cómo hablar con una computadora

Operadores de cadenas - introducción

Concatenación

El signo de + (más), al ser aplicado a dos cadenas, se convierte en un operador de concatenación:

`string + string`

```
fnam = input("¿Me puedes dar tu nombre por favor? ")
lnam = input("¿Me puedes dar tu apellido por favor? ")
print("Gracias.")
print("\nTu nombre es " + fnam + " " + lnam + ".")
```

