

INTRODUCCIÓN A LA PROGRAMACIÓN

CON  python

ASPECTOS BÁSICOS


Literales

Los **literales** nos permiten representar valores. Un **literal es un valor** que se expresa como sí mismo en lugar una variable o el resultado de una expresión, como el número 3 o la cadena "Hello"

Estos valores pueden ser de diferentes tipos, de esta manera tenemos diferentes tipos de literales.

Literales

Literales numéricos

- Para representar números **enteros** utilizamos cifras enteras. Ejemplos: 3, 12, -23.
 - Para los números **reales** utilizamos un punto para separar la parte entera de la decimal. Ejemplos: 2.3, 45.6.
 - Podemos indicar que la parte decimal es 0, por ejemplo 10., o la parte entera es 0, por ejemplo .001.
- 

Literales

Literales cadenas

Nos permiten representar **cadenas** de caracteres.

Para delimitar las cadenas podemos usar el carácter ' o el carácter ". También podemos utilizar la combinación """ cuando la cadena ocupa más de una línea. Ejemplos:

```
'hola que tal!'  
"Muy bien"  
"3"  
'Les dije a mis amigos, "¡Python es mi lenguaje favorito!"'
```

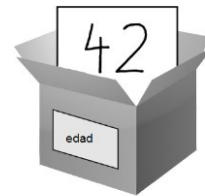
Es decir, cualquier cosa entre comillas es considerada una cadena en Python.



Variables

Una **variable** es como una caja en la memoria del ordenador donde puedes almacenar un valor.

Si quieres usar el resultado de una expresión más adelante en tu programa, lo puedes almacenar en una variable.



Expresiones

```
>>>2+2
```

```
4
```

En Python $2+2$ se denomina **expresión** que es la instrucción más básica en este lenguaje. Las expresiones pueden consistir de **variables**, **literales**, **operadores** y **funciones** que pueden siempre ser evaluadas (ser reducidas) a un simple valor.

Tipos de datos

Un **tipo de dato** es una categoría de valores que puede tomar una variable. Los tipos de datos más comunes en Python son:

- Entero (Integer) -2 -1 0 1 2
- Real (Float) -1.25 -1.9 0.5 1.0
- Cadena (String) "a" "aaa" "Hola@pepito" "11 gatos"
- Booleano (Boolean) True (verdadero) False (falso)

Cualquier número que tenga decimales lo llama float. Este término es utilizado en la mayoría de los lenguajes de programación, y se refiere al hecho de que un punto decimal puede aparecer en cualquier posición en un número.

Operadores

Los operadores que podemos utilizar se clasifican según el tipo de datos con los que trabajen y podemos poner algunos ejemplos:

- Operadores aritméticos: + , - , * , / , // , % , ** .
- Operadores de cadenas: + , *
- Operadores de asignación: =
- Operadores de comparación: == , != , >= , > , <= , <
- Operadores lógicos: and , or , not
- Operadores de pertenencia: in , not in

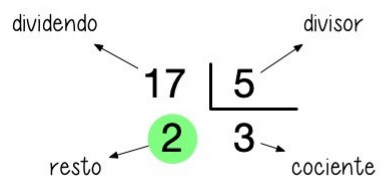
Operadores aritméticos

**	exponente	2**3	8
%	módulo	22%8	6
//	división entera	22//8	2
/	división	22/8	2.75
*	multiplicación	3*5	15
-	resta	5-2	3
+	suma	2+2	4

El orden de las operaciones (precedencia) matemáticas en Python es igual al de las matemáticas

La operación módulo

La operación módulo, cuyo símbolo en Python es %, se define como el resto de dividir dos números. Veamos un ejemplo para entender bien su funcionamiento:



$$\text{dividendo} = \text{divisor} * \text{cociente} + \text{resto}$$

$$\text{resto} = \text{dividendo} \% \text{divisor}$$

¡Pruébalo!

```
>>> 2+3*6
```

```
>>> (2+3)*6
```

```
>>> 2**8
```

```
>>> 2*1.0
```

```
>>> 23/7
```

¡Pruébalo!

```
>>> 8/2
```

Cuando se dividen dos números enteros que resulta en un número sin decimales el resultado es siempre con decimal

```
>>> 23//7
```

```
>>> 23%7
```

```
>>> (5-1)*((7+1)/(3-1))
```

```
>>> 1 + 2.0
```

Si mezclas un entero y un decimal en cualquier operación, obtendrás un decimal

Cadenas

Las cadenas no son más que texto entre comillas simples ('hola') o dobles ("hola"):

- "2"
- "¡Hola a todos!"
- "7 manzanas, 14 naranjas, 3 limones"
- "Nunca es tarde para bien hacer, haz hoy lo que no hiciste ayer."
- "0*&#wY%*&0CfsdY0*&gfc%Y0*&%3yc8r2"

Guiones bajos en números

Cuando escribes números, puedes agrupar dígitos utilizando guiones bajos para hacer que los números grandes sean más legibles:

```
>>> edad_universo = 14_000_000_000
```

Pero, cuando imprimes un número que fue definido utilizando guiones bajos. Python imprime solo los dígitos.

```
>>> print(edad_universo)
```

```
14000000000
```

Concatenación de cadenas

El significado de un operador puede cambiar basado en los tipos de datos de los valores con los que se este utilizando.

Por ejemplo, `+` es el operador suma cuando opera con dos enteros. Sin embargo cuando se una `+` con dos cadenas lo que hace es concatenarlas actuando el `+` como operador de concatenación.

```
>>> "Ana" + "María"  
'AnaMaría'
```

Repetición de cadenas

El operador `*` se usa para efectuar la multiplicación cuando opera con dos enteros o dos números decimales. Pero, cuando `*` es usado con una cadena y un valor entero, se convierte en el operador de repetición de cadenas.

```
>>> "Ana" * 5  
'AnaAnaAnaAnaAna'
```


Valores booleanos

Como decíamos al comienzo del tema una variable de tipo booleano sólo puede tener dos valores: **True** (cierto) y **False** (falso).

Estos valores son especialmente importantes para las expresiones condicionales y los bucles, como veremos más adelante.

Operadores de comparación

Los operadores de comparación comparan dos valores y lo reducen a un solo valor **booleano**:

==	Igual a	¿son iguales a y b?
!=	Distinto	¿son distintos a y b?
<	Menor que	¿es a menor que b?
>	Mayor que	¿es a mayor que b?
<=	Menor o igual que	¿es a menor o igual que b?
>=	Mayor o igual que	¿es a mayor o igual que b?

¡Pruébalo!

```
>>> 42==42
>>> 42==49
>>> 2!=3
>>> 2!=2
>>> "hola"=="hola"
>>> "hola"=="Hola"
>>> "perro" != "gato"

>>> True ==True
>>> True != False
>>> 42=="42"
```

Operadores lógicos

Estos son los distintos tipos de operadores con los que podemos trabajar con valores booleanos y son tres: **and**, **or** y **not**

and ¿se cumple a y b?

or ¿se cumple a o b?

not No a

- El resultado de **and** es verdadero cuando ambos operandos lo son
- El resultado de **or** es verdadero cuando cualquiera de los operandos lo es
- El operador **not** invierte el valor del operando

x	y	x or y
False	False	False
False	True	True
True	False	True
True	True	True

x	y	x and y
False	False	False
False	True	False
True	False	False
True	True	True

x	not x
False	True
True	False

Mezclando operadores lógicos y operadores de comparación

Los valores booleanos son además el resultado de expresiones que utilizan operadores de comparación

$(4 < 5)$ and $(5 < 6)$

$(4 < 5)$ and $(9 < 6)$

$(1 == 2)$ or $(2 == 2)$

not $2 + 2 == 5$

Precedencia de operadores

Operadores	Sentido
()	paréntesis
**	Exponente
+x, -x, ~x	Unario más, Unario menos, Bit a bit NO
*, /, //, %	Multiplicación, División, División de piso, Módulo
+, -	Suma resta
<<, >>	Operadores de desplazamiento bit a bit
&	Y bit a bit
^	XOR bit a bit
	O bit a bit
==, !=, >, >=, <, <=, is, is not, in, not in	Comparaciones, Identidad, Operadores de pertenencia
not	NO lógico
and	Y lógico
or	O lógico

Sentencia de asignación

Almacenas valores en variables con una sentencia de asignación.

Una **sentencia de asignación** consiste del nombre de la variable, el signo = y el valor a ser almacenado:

Ojo! No confundir estos operadores

edad = 42

== (igualdad)

nombre = "Juan"

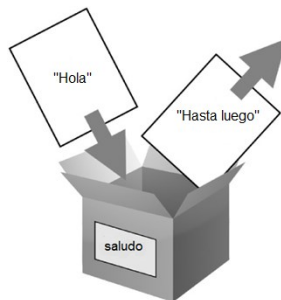
= (asignación)

Sentencia de asignación

Cuando a una variable se le asigna un nuevo valor el valor anterior se pierde.

```
>>> saludo = "Hasta luego"
>>> saludo
"Hasta luego"
```


```
>>> saludo = "Hola"
>>> saludo
"Hola"
```



¡Pruébalo!

```
>>> alumnos = 15
>>> alumnos = alumnos + 5
>>> alumnos

>>> alumnos = alumnos + 3
>>> alumnos
```



Asignación múltiple

Puedes asignar valores a más de una variable usando solo una línea. Esto puede ayudar a acortar sus programas y hacerlos más fáciles de leer; utiliza esta técnica con mayor frecuencia al inicializar un conjunto de números. Por ejemplo, así es como puede inicializar las variables x, y y z a cero:

```
>>> x, y, z = 0, 0, 0
```



Operadores de asignación compuestos


Operador	Ejemplo	Equivalencia
<code>+=</code>	<code>x += 2</code>	<code>x = x + 2</code>
<code>-=</code>	<code>x -= 2</code>	<code>x = x - 2</code>
<code>*=</code>	<code>x *= 2</code>	<code>x = x * 2</code>
<code>/=</code>	<code>x /= 2</code>	<code>x = x / 2</code>
<code>%=</code>	<code>x %= 2</code>	<code>x = x % 2</code>
<code>//=</code>	<code>x //= 2</code>	<code>x = x // 2</code>
<code>**=</code>	<code>x **= 2</code>	<code>x = x ** 2</code>

Nombres de variables


El nombre de una variable debe obedecer las siguientes reglas. No seguirlas causa errores:

- Solo puede ser una **palabra**
- Solo puede contener **letras**, **números** y el carácter **_** (**guión bajo**)
- No puede comenzar con un número. Por ejemplo, puedes llamar a una variable `mensaje_1` pero no `1_mensaje`.

Nombres de variables

- No se permiten espacios en los nombres de variables, pero se pueden usar guiones bajos. Se utiliza para separar palabras en nombres de variables. Por ejemplo, **mensaje_bienvenida** funciona, pero el **mensaje bienvenida** causará errores.
 - Evita usar **palabras clave** de Python y nombres de funciones como nombres de variable nombres; es decir, no uses palabras que Python ha reservado para un propósito programático particular, como la palabra `print`.
- 

Nombres de variables

- Los nombres de las variables deben **ser breves pero descriptivos**. Por ejemplo, **nombre** es mejor que **n**, **nombre_estudiante** es mejor que **n_e** y **longitud_nombre** es mejor que **longitud_de_nombre_de_personas**
 - Las variables de Python que estás utilizando en este punto deben estar en **minúsculas**. No dará error si usas letras mayúsculas, pero letras mayúsculas en nombres de variables tienen significados especiales que discutiremos en más adelante
- 

Nombres de variables

Los programadores generalmente eligen nombres para sus variables que tengan sentido y documenten para qué se usa esa variable.


saldo

saldo_actual

_saldo

SALDO

Los nombres de variables son sensibles a las mayúsculas lo cual significa que saldo, SALDO, Saldo y sALDo son 4 variables distintas.



Constantes

Una constante es como una variable cuyo valor permanece igual durante todo el vida de un programa. Python no tiene tipos constantes incorporados, pero los programadores usan todas las **letras mayúsculas** para indicar que una variable debe ser tratada como una constante y nunca ser cambiada:

MAX_CONNECTIONS = 5000



Comentarios de una línea

A medida que los programas se van volviendo más grandes se vuelven más difíciles de leer. Es una buena idea añadir notas a tus programas. Estas notas reciben el nombre de **comentarios** y en Python los comentarios de una línea comienzan con el símbolo **#**

```
# Calcula el porcentaje de hora transcurrido
```

```
porcentaje = (minuto * 100) / 60
```

```
o
```

```
porcentaje = (minuto * 100) / 60          # porcentaje de una hora
```

Los comentarios son más útiles cuando documentan características del código que no resultan obvias

Comentarios de múltiples líneas

```
""" Este es un comentario multilínea. La
siguiente parte realiza una serie
de cosas muy chulas """
```

The zen of Python



Los programadores experimentados te animarán a que evites la complejidad y que aspire a la sencillez siempre que sea posible. La filosofía de la comunidad de Python se encuentra en “The Zen of Python” de Tim Peters.

Tim Peters, experto en Python desde hace mucho tiempo, canaliza sucintamente los principios rectores de BDFL (Benevolent Dictator For Life, el apodo del creador de Python, Guido van Rossum) para el diseño de Python en 20 aforismos, de los que sólo se han escrito 19.

Puedes acceder a estos principios escribiendo en el interprete:

```
>>> import this
```

```
.
```

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Lo bonito es mejor que lo feo.
 Lo explícito es mejor que lo implícito.
 Lo simple es mejor que lo complejo.
 Lo complejo es mejor que lo complicado.
 Plano es mejor que anidado.
 Esparcido es mejor que denso.
 La legibilidad cuenta.
 Los casos especiales no son tan especiales como para romper las reglas.
 Aunque la practicidad gana a la pureza.
 Los errores nunca deben pasar en silencio.
 A menos que se silencien explícitamente.
 Ante la ambigüedad, rechaza la tentación de adivinar.
 Debe haber una -y preferiblemente sólo una- forma obvia de hacerlo.
 Aunque esa manera puede no ser obvia al principio, a menos que seas holandés.
 Ahora es mejor que nunca.
 Aunque a menudo “nunca” es mejor que “ahora mismo”.
 Si la implementación es difícil de explicar, es una mala idea.
 Si la implementación es fácil de explicar, puede ser una buena idea.
 Los espacios de nombres son una gran idea: ¡hagamos más!

¡Pruébalo!

```
nombre = input("Dime tu nombre: ") #pregunta tu nombre
print("¡Qué bueno conocerte", nombre+"!")
print("La longitud de tu nombre es: ")
print(len(nombre))
edad = input("Dime tu edad: ")      #pregunta tu edad
print("Entonces", nombre, "tienes", edad, "años")
```

Funciones predefinidas utilizadas

`print(x)` imprime x en pantalla

`input(x)`

- muestra la cadena x solicitando al usuario que introduzca algo
- espera hasta que el usuario ingrese algo por teclado
- devuelve lo que ingreso el usuario como **cadena**

`len(x)` devuelve el número de caracteres de x

Funciones

Una **función** es un fragmento de código con un nombre asociado que realiza una serie de tareas.

Las funciones en Python permiten **parámetros** de entrada (lo que está entre paréntesis se llama parámetro). Pueden recibir un parámetro, varios parámetros o ninguno, y **devuelven un valor**.

Más adelante aprenderás como definir y utilizar tus propias funciones.

Conversión de tipos

Las funciones **str()**, **int()** y **float()** convierten a una cadena, entero o decimal respectivamente el valor pasado como parámetro.

```
edad= 29
print ("Tengo " + str(edad) + " años")
print ("Tengo", edad, "años")


numero= int(input("Dime un número: "))

precio = float(input("Ingresa el precio: "))
```

str() es útil cuando tienes un entero o decimal y quieres concatenarlo a una cadena. Las funciones int() y float() puede ser útiles cuando se tiene un número como una cadena y se quiere utilizar en una operación matemática.

¡Pruébalo!


```
>>>str(0)
>>>str(-3.14)
>>>int("42")
>>>int("-99")
>>>int(1.25)
>>>float("3.14")
>>>float(10)
```



Función type

La función **type()** nos devuelve el tipo de dato de un objeto dado.

```
>>> type(5)
<class 'int'>
>>> type(5.5)
<class 'float'>
>>> type("hola")
<class 'str'>
```



Documentación de funciones incorporadas

docs.python.org

¿Tipo char?

En algunos lenguajes de programación existe una relación entre los caracteres y los números enteros.

El lenguaje Python utiliza el conjunto de caracteres **Unicode**, que incluye no solamente el conjunto ASCII sino también caracteres específicos de la mayoría de los alfabetos.

Para simplificar el siguiente ejemplo, podemos tomar la tabla de codificación de caracteres ASCII.

TABLA DE CARACTERES DEL CÓDIGO ASCII

1	0	25	↓	49	1	73	I	97	a	121	y	145	␣	169	~	193	␣	217	␣	241	␣
2	●	26		50	2	74	J	98	b	122	z	146	␣	170	␣	194	␣	218	␣	242	␣
3	♥	27		51	3	75	K	99	c	123	{	147	␣	171	␣	195	␣	219	␣	243	␣
4	♦	28	↖	52	4	76	L	100	d	124		148	␣	172	␣	196	␣	220	␣	244	␣
5	▲	29	↗	53	5	77	M	101	e	125	}	149	␣	173	␣	197	␣	221	␣	245	␣
6	★	30	↖	54	6	78	N	102	f	126	~	150	␣	174	␣	198	␣	222	␣	246	␣
7		31	↗	55	7	79	O	103	g	127	␣	151	␣	175	␣	199	␣	223	␣	247	␣
8		32		56	8	80	P	104	h	128	␣	152	␣	176	␣	200	␣	224	␣	248	␣
9		33	!	57	9	81	Q	105	i	129	␣	153	␣	177	␣	201	␣	225	␣	249	␣
10		34	"	58	:	82	R	106	j	130	␣	154	␣	178	␣	202	␣	226	␣	250	␣
11		35	#	59	;	83	S	107	k	131	␣	155	␣	179	␣	203	␣	227	␣	251	␣
12		36	\$	60	<	84	T	108	l	132	␣	156	␣	180	␣	204	␣	228	␣	252	␣
13		37	%	61	=	85	U	109	m	133	␣	157	␣	181	␣	205	␣	229	␣	253	␣
14		38	>	62	>	86	V	110	n	134	␣	158	␣	182	␣	206	␣	230	␣	254	␣
15		39	?	63	?	87	W	111	o	135	␣	159	␣	183	␣	207	␣	231	␣	255	␣
16	↖	40	(64	␣	88	X	112	p	136	␣	160	␣	184	␣	208	␣	232	␣		␣
17		41)	65	A	89	Y	113	q	137	␣	161	␣	185	␣	209	␣	233	␣		␣
18		42	*	66	B	90	Z	114	r	138	␣	162	␣	186	␣	210	␣	234	␣		␣
19	␣	43	+	67	C	91	[115	s	139	␣	163	␣	187	␣	211	␣	235	␣		␣
20	␣	44	,	68	D	92	\	116	t	140	␣	164	␣	188	␣	212	␣	236	␣		␣
21	␣	45	-	69	E	93]	117	u	141	␣	165	␣	189	␣	213	␣	237	␣		␣
22	␣	46	.	70	F	94	^	118	v	142	␣	166	␣	190	␣	214	␣	238	␣		␣
23	␣	47	/	71	G	95	␣	119	w	143	␣	167	␣	191	␣	215	␣	239	␣		␣
24	␣	48	0	72	H	96	␣	120	x	144	␣	168	␣	192	␣	216	␣	240	␣		␣

¿Tipo char?

En dicha tabla existe una correspondencia entre el **nº de orden del carácter** y la representación literal del **carácter**. Python no tiene un tipo de datos de carácter incorporado. Un carácter es simplemente una cadena de longitud 1. Las funciones integradas **ord()** y **chr()** operan sobre caracteres.

ord(c) devuelve el entero ordinal (Unicode) de una cadena de un carácter

```
>>> ord('A')
```

```
65
```

chr(i) devuelve una cadena de un carácter del Unicode ordinal i

```
>>> chr(65)
```

```
'A'
```



Función round

En Python hay una función **round()** incorporada que redondea un número al número dado de dígitos. La función round() acepta dos argumentos numéricos, n y x dígitos y devuelve el número n después de redondearlo a x dígitos.

```
>>> numero = 1.237437874376
```

```
>>> numero = round(numero, 2)
```

```
>>> numero
```

```
1.24
```



Autoevaluación

1. ¿Cuáles de los siguientes son operadores y cuales valores?
`*` `"Hola"` `-8.8` `-` `/` `+` `5`
2. ¿Cuál es una variable y cual es una cadena?
`edad` `"edad"`
3. ¿Cuál es la diferencia entre el operador `=` y el operador `==`?
4. ¿Qué contiene la variable `manzana` luego de ejecutar el siguiente código?
`manzana = 20`
`manzana=manzana + 1`

Entrada y salida estándar

Función `input`

Nos permite leer por teclado información. Devuelve una cadena de caracteres y puede tener como argumento una cadena que se muestra en pantalla:

```
>>> nombre = input("Nombre:")
>>> edad = int(input("Edad:"))
```



Entrada y salida estándar

Función print

Nos permite escribir en la salida estándar. Podemos indicar varios datos a imprimir, que por defecto serán separado por un espacio. Podemos también imprimir varias cadenas de texto utilizando la concatenación.

```
>>> print("¡Hola mundo!")  
¡Hola mundo!
```

```
>>> print(33)  
33
```




Entrada y salida estándar

Función print

```
>>> print("Tengo","dos","manzanas")  
Tengo dos manzanas
```

```
>>> print("Tengo",2,"manzanas")  
Tengo 2 manzanas
```



Entrada y salida estándar

Más sobre el uso de la **función print**

Se le puede pasar un parámetro que es el separador. Especifica cómo separar los elementos, si hay más de uno. Por defecto es ' '.

Un ejemplo de uso es:

```
>>> día = 23
>>> mes = 12
>>> año = 2023
>>> print(día,mes,año,sep='-')
```

Y la salida que produce es:

```
23-12-2023
```

Autoevaluación

5. ¿Cuál es el resultado de evaluar las siguientes expresiones?
 'perro' + 'perroperro'
 'perro' * 3
6. ¿Por qué **edad** es un nombre de variable válido y **100** no?
7. ¿Por qué la siguiente expresión da error?
 "He comido " + 99 + " flamenquines"

Has aprendido

- la función `print()` para mostrar resultados
- la función `input()` para leer de teclado
- tipos de datos: entero, decimal, cadena y booleano (y “char”)
- a hacer cálculos
- expresiones básicas
- conversiones de tipos usando `str()`, `int()`, `float()` y `type()`
- las funciones `len()` y `round()`
- comentarios
- lo que es una variable
- nombres legales de variables

Actividad

1. Escribe un programa que pregunte el nombre del usuario en la consola y después de que el usuario lo introduzca muestre por pantalla la cadena ¡Hola <nombre>!, donde <nombre> es el nombre que el usuario haya introducido
2. Escribe un programa que pregunte al usuario por el número de horas trabajadas y el coste por hora. Después debe mostrar por pantalla la paga que le corresponde.
3. Escribe código que pueda calcular el **área del círculo** usando las variables **radio** y **pi = 3.14159**. La fórmula, en caso de que no la sepas es: **$A = \pi \times r^2$** . Imprime el resultado de su programa de la siguiente manera: "El área de un círculo con radio ... es ..."
4. Escribe un programa que haga el siguiente cálculo: El precio de un **libro es de \$24.95**, pero las librerías obtienen un **40% de descuento**. El **envío cuesta \$3** por la primera copia y **75 centavos por cada copia adicional**. Calcular el costo **total** al por mayor de **60 copias**.

Solución

```
precio_libro = 24.95
precio_descuento = precio_libro*0.6
print(precio_descuento)
envio = 3 + *59
total = precio_descuento*60 + envio
print(round(total,2))
```

Escribe un programa que haga el siguiente cálculo: **El precio de un libro es de \$24.95, pero las librerías obtienen un 40% de descuento.** El envío cuesta \$3 por la primera copia y **75 centavos por cada copia adicional.** Calcular el costo **total** al por mayor de **60 copias**.

El código que han escrito no resuelve este problema para caso general, solo para el caso específico de 60 libros que cuestan 24,95 por libro.

Si quieres escribir código que resuelva problemas de una manera más general, necesita usar variables para almacenar valores.

INTRODUCCIÓN A LA PROGRAMACIÓN

EN LÍNEA

Los siguientes enlaces sirven para:

- hacer pruebas en el intérprete

<https://www.python.org/shell/>

- escribir programas completos

<https://www.programiz.com/python-programming/online-compiler/>


App para aprender python




Aprende Python

SoloLearn Educación

★★★★★ 39.221 

 PEGI 3

 Esta aplicación es compatible con algunos de tus dispositivos.