2.2 Literales Python

Literales - los datos en sí mismos

Un literal son datos cuyos valores están determinados por el propio literal.

Eche un vistazo al siguiente conjunto de dígitos: 123

¿Puedes adivinar qué valor representa?

Pero ¿qué pasa con esto?: c

- **123 es un literal, c no lo es.**

Literales - los datos en sí mismos

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

En este ejemplo, dos tipos diferentes de literales:

- Una cadena (**string)**
- Y un número entero (**integer)**

Trabajando con números en Python

Los números que manejan las computadoras modernas son de dos tipos

Enteros (integers), es decir, aquellos que carecen de la parte fraccionaria;

y números de **coma flotante** (o simplemente **flotantes, floats**), que contienen (o pueden contener) la parte fraccionaria.

Mira estos dos números:

- 4
- 4.0

Enteros (Integers)

Estamos familiarizados con las representaciones de este número

11,111,111

11.111.111

11 111 111

Python no acepta cosas como estas. Está prohibido. Puede escribir este número en Python de esta manera:

11111111

O así

11_111_111

Ejemplos

- `print(0o123)`
- `print(0x123)`
- `print(0.000000000000000000000001)`

- Veamos el ejemplo en la IDLE de Python.

Strings

- Las cadenas se utilizan cuando necesita procesar texto (como nombres de todo tipo, direcciones, novelas, etc.) y siempre necesitan comillas.
- Ejemplo

Soy un string.

```
print(“Soy un string.”)
```

Trate de mostrar en pantalla (imprimir) → Me gusta “Python”

Strings

1) Podemos usar el carácter de escape

```
print("I like \"Monty Python\"")
```

2) Python puede usar un apóstrofe en lugar de una cita

```
print('I like "Monty Python"')
```

Boolean

- Los literales booleanos representan solo dos valores: verdadero o falso.
- En Python:
  - El valor de 1 se asume como verdadero
  - El valor de 0 se asume como Falso.

Debe tomar estos símbolos como están, incluida la distinción entre mayúsculas y minúsculas

Laboratorio 2.1.2.11

Escenario

Escriba un fragmento de código de una línea, utilizando la función `print()`, así como los caracteres de nueva línea y escape, para que coincida con el resultado esperado en tres líneas.

Salida esperada

```
Estoy
aprendiendo
"Python"
```

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

Operadores

Operadores básicos

Un operador es un símbolo del lenguaje de programación, que puede operar sobre los valores.

Algunos operadores comunes son

+	//
-	%
*	**
/	

Operadores aritméticos: exponenciación

Un signo ** (doble asterisco) es un operador de potenciación (potencia). Su argumento de la izquierda es la base, su derecho, el exponente.

```
print(2 ** 3)      8
print(2 ** 3.)     8.
print(2. ** 3)     8.
print(2. ** 3.)    8.
```

Operadores aritméticos: multiplicación y división

- Un signo * (asterisco) es un operador de multiplicación.
- El signo / (barra) es un operador divisional.

```
print(2 * 3)     6        print(6 / 3)     2.0
print(2 * 3.)    6.0      print(6 / 3.)    2.0
print(2. * 3)    6.0      print(6. / 4)    1.5
print(2. * 3.)   6.0      print(6. / 4.)   1.5
```

Operadores aritméticos: división de enteros

Un signo // (doble barra) es un operador divisional de números enteros. Se diferencia del estándar / operador en dos detalles:

- su resultado carece de la parte fraccionaria - está ausente (para enteros), o siempre es igual a cero (para flotantes); esto significa que **los resultados siempre se redondean**;
- se ajusta a la regla de números enteros frente a flotantes.

```
print(6 // 3)      2
print(6 // 3.)     2.0
print(6. // 3)     2.0
```

Operadores aritméticos: división de enteros

```
print(6 // 4)   1        print(6 / 4)   1.5
print(6. // 4)  1.0      print(6. / 4)  1.5
```

- ¿Puedes predecir?

```
print(-6 // 4)= ?
print(6. // -4) = ?
```

- **El redondeo siempre va al valor entero menor**

Operadores: resto (módulo)

Su representación gráfica en Python es el signo% (porcentaje) y el resultado del operador es un **resto** que queda **después** de **la división entera**.

```
print(14 % 4)
```
⇨ 2

¿Puede predecir `print(12 % 4.5)`?

Operadores: suma y resta

El operador de suma es el signo + (más)

- `print(-4 + 4)` `0`
- `print(-4. + 8)` `4.0`

El operador de resta es obviamente el signo - (menos)

- `print(-4 - 4)` `-8`
- `print(4. - 8)` `-4.0`

Operadores unarios y binarios

El signo - (menos) también tiene otro significado:

Puede cambiar el signo de un número.

```
print(-1.1)
```

Unario

```
print(-4 - 4)
print(-4 + 4)
print(6 * 3)
```

Binario

Operadores y sus prioridades

Considere la siguiente expresión:

$$2 + 3 * 5$$

Probablemente recuerdes de la escuela que **las multiplicaciones preceden a las sumas**.

```
print(9 % 6 % 2)
```

La mayoría de los operadores de Python tienen enlace del lado izquierdo

Operadores y sus enlaces: exponenciación

La vinculación del operador determina el orden de los cálculos realizados por algunos operadores con la misma prioridad, colocados uno al lado del otro en una expresión.

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

El operador de exponenciación utiliza la vinculación del lado derecho.

Lista de prioridades

Priority	Operator	
1	+, -	Unario
2	**	
3	*, /, %, //	
4	+, -	Binario

- `print(2 * 8 % 5)` ⇨ `1`

- `print(8 % 5 * 2)` ⇨ `6`

Operadores y paréntesis

- De acuerdo con las reglas aritméticas, **las subexpresiones entre paréntesis siempre se calculan primero**.
- Puede utilizar tantos paréntesis como necesite, y a menudo se utilizan para mejorar la **legibilidad de una expresión**, incluso si **no cambian** el orden de las operaciones.

```
print((5 * ((25 % 13) + 100) / (2 * 13)) // 2)
```

Operadores y paréntesis

```
print((5 * ((25 % 13) + 100) / (2 * 13)) // 2)
      ((5 * ((12) + 100) / (26)) // 2)
   ➔  ((5 * (12 + 100) / 26) // 2)
         ((5 * (112) / 26) // 2)
      ➔  ((5 * 112 / 26) // 2)
            ((560 / 26) // 2)
         ➔  (21.53856 // 2)
```

10

Preguntas de interacción

Ejercicio 1
¿Cuál es el resultado del siguiente fragmento?

- `print((2 ** 4), (2 * 4.), (2 * 4))`

Ejercicio 2
¿Cuál es el resultado del siguiente fragmento?

- `print((-2 / 4), (2 / 4), (2 // 4), (-2 // 4))`

Ejercicio 3
¿Cuál es el resultado del siguiente fragmento?

- `print((2 % -4), (2 % 4), (2 ** 3 ** 2))`

Variables

¿Qué son las variables?

Las variables se utilizan para almacenar números o para almacenar los resultados de las operaciones, con el fin de utilizarlos en otras operaciones.

Cada variable de Python debe tener:

- un nombre
- un valor (el contenido del recipiente)

Reglas variables

- el nombre de la variable debe estar compuesto por letras mayúsculas o minúsculas, dígitos y el carácter _ (guión bajo)
- el nombre de la variable debe comenzar con una letra
- el carácter de subrayado es una letra
- las letras mayúsculas y minúsculas se tratan como diferentes
- el nombre de la variable no debe ser ninguna de las palabras reservadas de Python (las palabras clave - explicaremos más sobre esto pronto).

Correct and incorrect variable names

Correcto:

- MyVariable
- i
- t34
- Exchange_Rate
- Counter
- DaysToChristmas
- _
- TheNameIsSoLongThatYouWillMakeMistakesWithIt

Incorrecto:

- 10t
- Exchange Rate

Palabras clave y variables

'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‘

Case-sensitive (Distingue mayúsculas y minúsculas**)**

print

Print

Creando variables

- Una variable surge como resultado de asignarle un valor. A diferencia de otros idiomas, no es necesario declararlo de ninguna manera especial.
- Si asigna algún valor a una variable inexistente, la variable se creará automáticamente. No necesitas hacer nada más.

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

Usando Variables

```
var = 1
account_balance = 1000.0
clientName = 'John Doe'
print(var, account_balance, clientName)
print(var)
```

No se le permite usar una variable que no existe, en otras palabras, una variable a la que no se le dio un valor.

```
var = "3.7.1"
print("Python version: " + var)
```

⇨ Python version: 3.7.1

Asignar un nuevo valor a una variable existente

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

Laboratorio 2.1.4.7

Scenario

- Haga 3 variables: john, mary, and adam;
- Asigne valores numéricos a las variables.
- Una vez almacenados lo valores las variables, imprima las variables en una línea y sepárelas con una coma;
- Ahora haga una variable llamada “total” que sea la suma de las variables anteriores
- Imprima el valor de “total”

Operadores de atajos

```
x = x * 2        ──────────►  x *= 2
oveja = oveja + 1  ────────►  oveja += 1
```

```
variable = variable op expresión

Variable op= expresión
```

Operadores de atajos - 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
```

Priority	Operator	
1	+, -	Unario
2	**	
3	*, /, %, //	
4	+, -	Binario

Laboratorio 2.1.4.9 - Variables: un conversor simple

Escenario

- Cree un convertidor de millas a kilómetros
- Complete el programa en el editor para que convierta:
  - millas a kilómetros;
  - kilómetros a millas.
- No cambie nada en el código existente. Escriba su código en los lugares indicados por ###.

Pruebe su programa con los datos que le proporcionamos en el código fuente.

1 milla = 1.61 km

```
kilometers = 12.25
miles = 7.38

miles_to_kilometers = miles * 1.61
kilometers_to_miles = kilometers / 1.61

print(miles, "miles is", round(miles_to_kilometers, 2), "kilometers")
print(kilometers, "kilometers is", round(kilometers_to_miles, 2), "miles")
```

7.38 miles is 11.88 kilometers
12.25 kilometers is 7.61 miles

Lab 2.1.4.10 - Operators and expressions

Scenario

- Create a code in order to evaluate the following expression:

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

- Create a variable **x**
- Create the code
- The result should be assign to **y**
- Print **y**

Sample input

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

Expected Output

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

Comentarios

Comentarios

Un comentario insertado en el programa, que se omite en tiempo de ejecución.

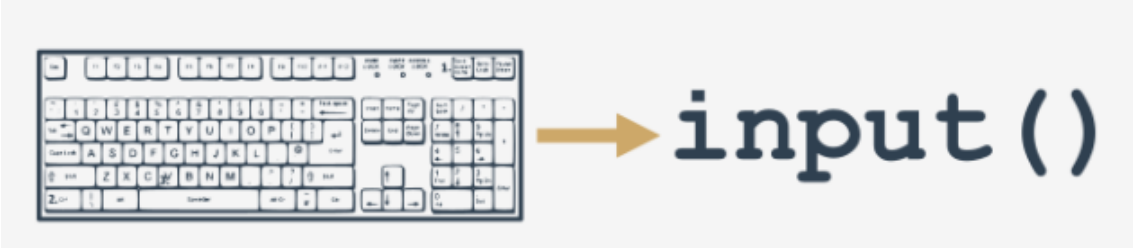
```
# Este programa evalúa la hipotenusa (c)
# a y b son las longitudes de los catetos
a = 3.0
b = 4.0
c = (a ** 2 + b ** 2) ** 0.5 # usamos ** en lugar de raíz cuadrada
print("c =", c)
```

Cómo hablar con una computadora

La función `input()`

- La función `input ()` puede leer los datos ingresados por el usuario y devolver los mismos datos al programa en ejecución.

```
print("Dime cualquier cosa...")
algo = input()
print("Me acabas de decir", algo, "!")
```



La función `input()` con un argumento

```
algo = input(“Dime cualquier cosa...")
print(“Me acabas de decir”, algo, “!”)
```

- La función `input ()` se invoca con un argumento: es una cadena que contiene un mensaje;
- El mensaje se mostrará en la consola antes de que el usuario tenga la oportunidad de ingresar algo.

El resultado de la función `input ()`

El resultado es siempre una cadena (**string)**

```
anything = input("Enter a number: ")
something = anything ** 2.0
print(anything, " elevado a 2 es ", something)
```

```
Enter a number: 6
Traceback (most recent call last):
  File "C:/Users/jleiton/AppData/Local/Programs/Python/Python37-32/ttt.py", line
 4, in <module>
    something = anything ** 2.0
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'float'
```

conversión de *Tipo*

Python ofrece dos funciones simples para especificar un tipo de datos y resolver este problema:

- `int()` toma un argumento e intenta convertirlo en un entero
- `float()` toma un argumento e intenta convertirlo en flotante

En ambos casos, si falla, todo el programa fallará también.

```
anything = float(input("Enter a number: "))
something = anything ** 2.0
print(anything, "to the power of 2 is", something)
```

Operadores de cadena - concatenación

- El signo + (más), cuando se aplica a dos cadenas, se convierte en un operador de concatenación

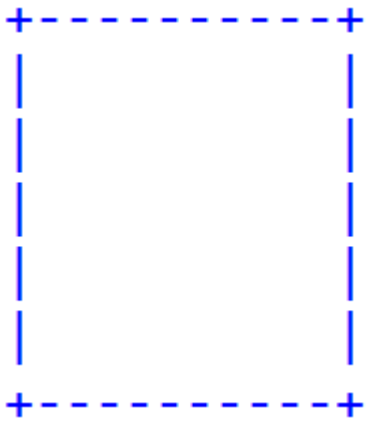
```
string + string
```

```
fnam = input("¿Podría darme su nombre, por favor?")
lnam = input("¿Podría darme su apellido, por favor?")
print("Gracias.\n\n")
print("Tu nombre es " + fnam + " " + lnam + ".")
```

Operadores de cadena - replicación

El signo * (asterisco), cuando se aplica a una cadena y un número (o un número y una cadena, ya que permanece conmutativo en esta posición) se convierte en un operador de replicación.

Por ejemplo:



```
"James" * 3 gives "JamesJamesJames"

3 * "an" gives "ananan"

5 * "2" (or "2" * 5) gives "22222" (not 10!)
```

Conversión de tipo: `str()`

También puede convertir un número en una cadena. Una función capaz de hacer eso se llama `str()`:

```
str(number)
```

```
str(2 ** 3)
```

Laboratorio 2.1.6.9 - Entrada y salida simples

Escenario

- Complete un código que calcule el resultado de las cuatro operaciones aritméticas básicas
- Imprima los resultados.

```
# ingrese un valor flotante
# para la variable a.

# ingrese un valor flotante
# para la variable b.

# muestre el resultado de la suma
# muestre el resultado de la resta
# muestre el resultado de la multiplicación
# muestre el resultado de la división
```

Laboratorio 2.1.6.10 – Operadores y expresiones

Escenario

- Ingrese un valor **x**, realice el cálculo y asigne el valor a **y** e imprima.

$$\cfrac{1}{x+\cfrac{1}{x+\cfrac{1}{x+\cfrac{1}{x}}}}$$

Ejemplo de entrada: 1
Salida esperada
`y = 0.6000000000000001`

Ejemplo de entrada: 10
Salida esperada
`y = 0.09901951266867294`

Ejemplo de entrada: -5
Salida esperada
`y = -0.19258202567760344`

Laboratorio 2.1.6.11 – Operadores y expresiones

Escenario

- Debe crear un código que permita calcular la hora final de un periodo de tiempo, dados el número de minutos (puede ser cualquier cantidad de minutos). El inicio de da como horas (0..23) y minutos (0..59). El resultado se debe imprimir.

Ejemplo de entrada

```
hour = 12
mins = 17
dur = 59
```

Salida esperada

```
Termina a las 13:16
```

Ejemplo de entrada

```
hour = 23
mins = 58
dur = 642
```

Salida esperada

```
Termina a las 10:40
```

Resumen

- Los métodos básicos de formato y salida de datos que ofrece Python, junto con los principales tipos de datos y operadores numéricos, sus relaciones mutuas y enlaces.
- El concepto de variables y la forma correcta de denominarlas.
- El operador de asignación, las reglas que gobiernan la construcción de expresiones.
- Entrada y conversión de datos.

Cuestionario

- ¿Qué tipo de literales son los siguientes cuatro ejemplos?

  "1.5", 2.0, 528, False

- Respuesta:

  El primero es una cadena, el segundo es numérico (flotante), el tercero es numérico (entero) y el cuarto es booleano.

Cuestionario

- ¿Cuál es la salida del siguiente fragmento de código?

```
print((2**4), (2*4.), (2*4))
```

- Respuesta:

```
16   8.0   8
```

Cuestionario

- ¿Cuál es el resultado del siguiente fragmento de código?

```
a = '1'
b = "1"
print(a + b)
```

- Respuesta:

  11

Cuestionario

- ¿Cuál es el resultado del siguiente código si se introduce un 3 y un 6 respectivamente?

```
x = input()
y = int(input())
print(x * y)
```

- Respuesta:

  333333