**Bienvenido a Fundamentos de Programación en Python - Parte 1**

**Módulo 1**

Introducción a Python y a la programación.

**Módulo 2**

Tipos de datos, variables, operaciones básicas de entrada y salida, operadores básicos.

**Módulo 3**

Valores booleanos, ejecución condicional, bucles, listas y procesamiento de listas, operaciones lógicas y bit a bit.

**Módulo 4**

Funciones, tuplas, diccionarios y procesamiento de datos.

**Fundamentos de Programación en Python: Módulo 2**

**En este módulo, aprenderás sobre:**

* Los tipos de datos y los métodos básicos de formateo, conversión de datos, entrada y salida de datos.
* Operadores.
* Variables.

**¡Hola, Mundo!**

Es hora de comenzar a escribir **código real y funcional en Python**. Por el momento será muy sencillo.

Como se muestran algunos conceptos y términos fundamentales, estos fragmentos de código no serán complejos ni difíciles.

print("¡Hola, Mundo!")

Ejecuta el código en la ventana del editor a la derecha. Si todo sale bien, veras la **línea de texto** en la ventana de consola.

Como alternativa, inicia IDLE, crea un nuevo archivo fuente de Python, coloca este código, nombra el archivo y guárdalo. Ahora ejecútalo. Si todo sale bien, verás una línea en la ventana de la consola IDLE. El código que has ejecutado debería parecerte familiar. Viste algo muy similar cuando te guiamos a través de la configuración del entorno IDLE.

Ahora dedicaremos un poco de tiempo para mostrarte y explicarte lo que estás viendo y porque se ve así.

Como puedes ver, el primer programa consta de las siguientes partes:

* La palabra print.
* Un paréntesis de apertura.
* Una comilla.
* Una línea de texto: ¡Hola, Mundo!.
* Otra comilla.
* Un paréntesis de cierre.

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

**La función print()**

Mira la línea de código a continuación:

print("¡Hola, Mundo!")

La palabra **print** que puedes ver aquí es el **nombre de una función**. Eso no significa que dondequiera que aparezca esta palabra, será siempre el nombre de una función. El significado de la palabra proviene del contexto en el cual se haya utilizado la palabra.

Probablemente hayas encontrado el término función muchas veces antes, durante las clases de matemáticas. Probablemente también puedes recordar varios nombres de funciones matemáticas, como seno o logaritmo.

Las funciones de Python, sin embargo, son más flexibles y pueden contener más contenido que sus parientes matemáticos.

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

* **Causar algún efecto** (por ejemplo, enviar texto a la terminal, crear un archivo, dibujar una imagen, reproducir un sonido, etc.); esto es algo completamente inaudito en el mundo de las matemáticas.
* **Evaluar un valor o algunos valores** (por ejemplo, la raíz cuadrada de un valor o la longitud de un texto dado); esto es lo que hace que las funciones de Python sean parientes de los conceptos matemáticos.

Además, muchas de las funciones de Python pueden hacer las dos cosas anteriores juntas.

¿De dónde provienen las funciones?

* Pueden venir **de Python mismo**. La función print es una de este tipo; dicha función es un valor agregado de Python junto con su entorno (está **integrada**); no tienes que hacer nada especial (por ejemplo, pedirle a alguien algo) si quieres usarla.
* Pueden provenir de uno o varios de los **módulos** de Python llamados complementos; algunos de los módulos vienen con Python, otros pueden requerir una instalación por separado, cual sea el caso, todos deben estar conectados explícitamente con el código (te mostraremos cómo hacer esto pronto).
* Puedes **escribirlas tú mismo**, colocando tantas funciones como desees y necesites dentro de tu programa para hacerlo más simple, claro y elegante.

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

Si vas a utilizar alguna función ya existente, no podrás modificar su nombre, pero cuando comiences a escribir tus propias funciones, debes considerar cuidadosamente la elección de nombres.

**La función print()**

Como se dijo anteriormente, una función puede tener:

* Un **efecto**.
* Un **resultado**.

También hay un tercer componente de la función, muy importante, el o los **argumento**(s).

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

Las funciones de Python, por otro lado, son más versátiles. Dependiendo de las necesidades individuales, pueden aceptar cualquier número de argumentos, tantos como sea necesario para realizar sus tareas.

**Nota**: algunas funciones de Python no necesitan ningún argumento.

print("¡Hola, Mundo!")

A pesar del número de argumentos necesarios o proporcionados, las funciones de Python demandan fuertemente la presencia de **un par de paréntesis** - el de apertura y de cierre, respectivamente.

Si deseas entregar uno o más argumentos a una función, colócalos **dentro de los paréntesis**. Si vas a utilizar una función que no tiene ningún argumento, aún tiene que tener los paréntesis.

**Nota**: para distinguir las palabras comunes de los nombres de funciones, coloca un **par de paréntesis vacíos** después de sus nombres, incluso si la función correspondiente requiere uno o más argumentos. Esta es una medida estándar.

La función de la que estamos hablando aquí es print().

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

Por supuesto que sí, pero ¿Qué son los argumentos?

**La función print()**

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

print("¡Hola, Mundo!")

Como se puede ver, la **cadena está delimitada por comillas** - de hecho, las comillas forman la cadena, recortan una parte del código y le asignan un significado diferente.

Podemos imaginar que las comillas significan algo así: el texto entre nosotros no es un código. No está diseñado para ser ejecutado, y se debe tomar tal como está.

Casi cualquier cosa que ponga dentro de las comillas se tomará de manera literal, no como código, sino como **datos**. Intenta jugar con esta cadena en particular - puedes modificarla. Ingresa contenido nuevo o borra parte del contenido existente.

Existe más de una forma de como especificar una cadena dentro del código de Python, pero por ahora, esta será suficiente.

Hasta ahora, has aprendido acerca de dos partes importantes del código- la función y la cadena. Hemos hablado de ellos en términos de sintaxis, pero ahora es el momento de discutirlos en términos de semántica.

**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**.

Discutiremos esto en mayor profundidad más adelante, pero por lo pronto, arrojaremos un poco más de luz al asunto.

print("¡Hola, Mundo!")

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

nombreFunción(argumento)

Veamos:

* Primero, Python comprueba si el nombre especificado es **legal** (explora sus datos internos para encontrar una función existente del nombre; si esta búsqueda falla, Python cancela el código).
* En segundo lugar, Python comprueba si los requisitos de la función para el número de argumentos **le permiten invocar** la función de esta manera (por ejemplo, si una función específica exige exactamente dos argumentos, cualquier invocación que entregue solo un argumento se considerará errónea y abortará la ejecución del código).
* Tercero, Python **deja el código por un momento** y salta dentro de la función que se desea invocar; por lo tanto, también toma los argumentos y los pasa a la función.
* Cuarto, la función **ejecuta el código**, provoca el efecto deseado (si lo hubiera), evalúa el (los) resultado(s) deseado(s) y termina la tarea.
* Finalmente, Python **regresa al código** (al lugar inmediatamente después de la invocación) y reanuda su ejecución.

**LABORATORIO**

**Tiempo Estimado**

5 minutos

**Nivel de dificultad**

Muy fácil

**Objetivos**

* Familiarizarse con la función print() y sus capacidades de formateo.
* 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.

**La función print()**

Tres preguntas importantes deben ser respondidas antes de continuar:

**1. ¿Cuál es el efecto que causa la función** print()**?**

El efecto es muy útil y espectacular. La función toma los argumentos (puede aceptar más de un argumento y también puede aceptar menos de un argumento) los convierte en un formato legible para el ser humano si es necesario (como puedes sospechar, las cadenas no requieren esta acción, ya que la cadena ya está legible) y **envía los datos resultantes al dispositivo de salida** (generalmente la consola); en otras palabras, cualquier cosa que se ponga en la función de print() aparecerá en la pantalla.

No es de extrañar entonces, que, de ahora en adelante, utilizarás print() muy intensamente para ver los resultados de tus operaciones y evaluaciones.

**2. ¿Qué argumentos espera** print()**?**

Cualquiera. Te mostraremos pronto que print() puede operar con prácticamente todos los tipos de datos ofrecidos por Python. Cadenas, números, caracteres, valores lógicos, objetos: cualquiera de estos se puede pasar con éxito a print().

**3. ¿Qué valor evalúa la función** print()**?**

Ninguno. Su efecto es suficiente - print() no evalúa nada.

**La función print() - instrucciones**

A estas alturas ya sabes que este programa contiene una invocación de función. A su vez, la invocación de función es uno de los posibles tipos de **instrucciones** de Python. Por lo tanto, este programa consiste de una sola instrucción.

Por supuesto, cualquier programa complejo generalmente contiene muchas más instrucciones que una. La pregunta es, ¿Cómo se acopla más de una instrucción en el código de Python?

La sintaxis de Python es bastante específica en esta área. A diferencia de la mayoría de los lenguajes de programación, Python requiere que **no haya más de una instrucción por una línea**.

Una línea puede estar vacía (por ejemplo, puede no contener ninguna instrucción) pero no debe contener dos, tres o más instrucciones. Esto está estrictamente prohibido.

**Nota**: Python hace una excepción a esta regla: permite que una instrucción se extienda por más de una línea (lo que puede ser útil cuando el código contiene construcciones complejas).

Vamos a expandir el código un poco, puedes verlo en el editor. Ejecútalo y nota lo que ves en la consola.

print("La Witsi Witsi Araña subió a su telaraña.")

print("Vino la lluvia y se la llevó.")

Tu consola Python ahora debería verse así:

La Witsi Witsi Araña subió a su telaraña.

Vino la lluvia y se la llevó.

Esta es una buena oportunidad para hacer algunas observaciones:

* El programa **invoca la función** print() **dos veces**, como puedes ver hay dos líneas separadas en la consola: esto significa que print() comienza su salida desde una nueva línea cada vez que comienza su ejecución. Puedes cambiar este comportamiento, pero también puedes usarlo a tu favor.
* Cada invocación de print() contiene una cadena diferente, como su argumento y el contenido de la consola lo refleja- esto significa que **las instrucciones en el código se ejecutan en el mismo orden** en que se colocaron en el archivo de origen; no se ejecuta la siguiente instrucción hasta que se complete la anterior (hay algunas excepciones a esta regla, pero puedes ignorarlas por ahora).

**La función print() - instrucciones**

Hemos cambiado un poco el ejemplo: hemos agregado una invocación **vacía** de la función print(). La llamamos vacía porque no hemos agregado ningún argumento a la función.

Lo puedes ver en la ventana del editor. Ejecuta el código.

print("La Witsi Witsi Araña subió a su telaraña.")

print()

print("Vino la lluvia y se la llevó.")

¿Qué ocurre?

Si todo sale bien, deberías ver algo como esto:

La Witsi Witsi Araña subió a su telaraña.

Vino la lluvia y se la llevó.

Como puedes ver, la invocación de print() vacía no esta tan vacía como se esperaba - genera una línea vacía (esta interpretación también es correcta) su salida es solo una nueva línea.

Esta no es la única forma de producir **una nueva línea** en la consola de salida. Enseguida mostraremos otra manera.

**La función print() - los caracteres de escape y nueva línea**

Hemos modificado el código de nuevo. Obsérvalo con cuidado.

print("La Witsi Witsi Araña\nsubió a su telaraña.\n")

print()

print("Vino la lluvia\ny se la llevó.")

Hay dos cambios muy sutiles: hemos insertado un par extraño de caracteres dentro del texto. Se ven así: \n.

Curiosamente, mientras **tú ves dos caracteres, Python ve solo uno**.

La barra invertida (\) tiene un significado muy especial cuando se usa dentro de las cadenas, es llamado **el carácter de escape**.

La palabra escape debe entenderse claramente- significa que la serie de caracteres en la cadena se escapa (detiene) por un momento (un momento muy corto) para introducir una inclusión especial.

En otras palabras, la barra invertida no significa nada, sino que es solo un tipo de anuncio, de que el siguiente carácter después de la barra invertida también tiene un significado diferente.

La letra n colocada después de la barra invertida proviene de la palabra newline (nueva línea).

Tanto la barra diagonal inversa como la n forman un símbolo especial denominado **carácter de nueva línea** (newline character), que incita a la consola a iniciar una **nueva línea de salida**.

Ejecuta el código. La consola ahora debería verse así:

La Witsi Witsi Araña

subió a su telaraña.

Vino la lluvia

y se la llevó.

Como se puede observar, aparecen dos nuevas líneas en la canción infantil, en los lugares donde se ha utilizado \n.

**La función print() los caracteres de escape y nueva línea**

El utilizar la diagonal invertida tiene dos características importantes:

1. Si deseas colocar solo una barra invertida dentro de una cadena, no olvides su naturaleza de escape: tienes que duplicarla, por ejemplo, la siguiente invocación causará un error:

print("\")

Mientras que esta no lo hará:

print("\\")

2. No todos los pares de escape (la barra invertida junto con otro carácter) significan algo.

Experimenta con el código en el editor, ejecútalo y observa lo que sucede.

print("La Witsi Witsi Araña\nsubió a su telaraña.")

print()

print("Vino la lluvia\ny se la llevó.")

**La función print() utilizando argumentos múltiples**

Hasta ahora se ha probado el comportamiento de la función print() sin argumentos y con un argumento. También vale la pena intentar alimentar la función print() con más de un argumento.

Mira la ventana del editor. Esto es lo que vamos a probar ahora:

print("Witsi witsi araña" , "subió" , "su telaraña.")

Hay una invocación de la función print() pero contiene **tres argumentos**. Todos ellos son cadenas.

Los argumentos están **separados por comas**. Se han rodeado de espacios para hacerlos más visibles, pero no es realmente necesario y no se hará más.

En este caso, las comas que separan los argumentos desempeñan un papel completamente diferente a la coma dentro de la cadena. El primero es una parte de la sintaxis de Python, el segundo está destinado a mostrarse en la consola.

Si vuelves a mirar el código, verás que no hay espacios dentro de las cadenas.

print("La Witsi Witsi Araña" , "subió" , "a su telaraña.")

Ejecuta el código y observa lo que pasa.

La consola ahora debería mostrar el siguiente texto:

La Witsi witsi araña subió su telaraña.

Los espacios, removidos de las cadenas, han vuelto a aparecer. ¿Puedes explicar por qué?

Dos conclusiones surgen de este ejemplo:

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

**La función print() - La manera posicional de pasar los argumentos**

Ahora que sabes un poco acerca de la función print() y como personalizarla, te mostraremos como cambiarla.

Deberías de poder predecir la salida sin ejecutar el código en el editor.

La forma en que pasamos los argumentos a la función print() es la más común en Python, y se denomina **manera posicional** (este nombre proviene del hecho de que el significado del argumento está dictado por su posición, por ejemplo, el segundo argumento se emitirá después del primero, y no al revés).

print("Mi nombre es", "Python.")

print("Monty Python.")

Ejecuta el código y verifica si la salida coincide con tus predicciones.

**La función print() - los argumentos de palabras clave**

Python ofrece otro mecanismo para transmitir o pasar los argumentos, que puede ser útil cuando se desea convencer a la función print() de que cambie su comportamiento un poco.

No se va a explicar en profundidad ahora. Se planea hacer esto cuando se trate el tema de funciones. Por ahora, simplemente queremos mostrarte como funciona. Siéntete libre de utilizarlo en tus propios programas.

El mecanismo se llama **argumentos de palabras clave**. El nombre se deriva del hecho de que el significado de estos argumentos no se toma de su ubicación (posición) sino de la palabra especial (palabra clave) utilizada para identificarlos.

La función print() tiene dos argumentos de palabras clave que se pueden utilizar para estos propósitos. El primero de ellos se llama end.

En la ventana del editor se puede ver un ejemplo muy simple de como utilizar un argumento de palabra clave.

Para utilizarlo es necesario conocer algunas reglas:

* Un argumento de palabra clave consta de tres elementos: una palabra clave que identifica el argumento (end -termina aquí); un signo de igual (=); y un valor asignado a ese argumento.
* Cualquier argumento de palabra clave debe ponerse después del último argumento posicional (esto es muy importante).

En nuestro ejemplo, hemos utilizado el argumento de palabra clave end y lo hemos igualado a una cadena que contiene un espacio.

print("Mi nombre es", "Python.", end=" ")

print("Monty Python.")

Ejecuta el código para ver como funciona.

La consola ahora debería mostrar el siguiente texto:

Mi nombre es Python. Monty Python.

Como puedes ver, el argumento de palabra clave end determina los caracteres que la función print() envía a la salida una vez que llega al final de sus argumentos posicionales.

El comportamiento predeterminado refleja la situación en la que el argumento de la palabra clave end se usa **implícitamente** de la siguiente manera: end="\n".

**La función print() - los argumentos de palabras clave**

Y ahora, es el momento de intentar algo más difícil.

Si observas detenidamente, verás que hemos utilizado el argumento end, pero su cadena asignada está vacía (no contiene ningún carácter).

print("Mi nombre es ", end="")

print("Monty Python.")

¿Qué pasará ahora? Ejecuta el programa en el editor para averiguarlo.

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.

La consola ahora debería mostrar el siguiente texto:

Mi nombre es Monty Python.

**Nota**: **No se han enviado nuevas líneas a la salida.**

La cadena asignada al argumento de la palabra clave end puede ser de cualquier longitud. Experimenta con ello si gustas.

**La función print() - los argumentos de palabras 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 (como separador).

print("Mi", "nombre", "es", "Monty", "Python.", sep="-")

Mira el código en el editor y ejecútalo.

El argumento sep entrega el siguiente resultado:

Mi-nombre-es-Monty-Python.

La función print() ahora utiliza un guión, en lugar de un espacio, para separar los argumentos generados.

**Nota**: el valor del argumento sep también puede ser una cadena vacía. Pruébalo tú mismo.

**La función print() - los argumentos de palabras clave**

Ambos argumentos de palabras clave pueden **mezclarse en una invocación**, como aquí en la ventana del editor.

El ejemplo no tiene mucho sentido, pero representa visiblemente las interacciones entre end y sep.

¿Puedes predecir la salida?

print("Mi", "nombre", "es", sep="\_", end="\*")

print("Monty", "Python.", sep="\*", end="\*\n")

Ejecuta el código y ve si coincide con tus predicciones.

Ahora que comprendes la función print() , estás listo para considerar aprender cómo almacenar y procesar datos en Python.

Sin print(), no se podría ver ningún resultado.

**LABORATORIO**

**Tiempo Estimado**

5 minutos

**Nivel de dificultad**

Muy fácil

**Objetivos**

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

**Escenario**

print("Fundamentos","Programación","en")

print("Python")

Modifica la primera línea de código en el editor, utilizando las palabras clave sep y end, para que coincida con el resultado esperado. Recuerda, utilizar dos funciones print().

No cambies nada en la segunda invocación de print().

**Resultado Esperado**

Fundamentos\*\*\*Programación\*\*\*en...Python

**LABORATORIO**

**Tiempo Estimado**

5-10 minutos

**Nivel de dificultad**

Fácil

**Objetivos**

* Experimentar con el código Python existente.
* Descubrir y solucionar errores básicos de sintaxis.
* Familiarizarse con la función print() y sus capacidades de formato.

**Escenario**

print(" \*")

print(" \* \*")

print(" \* \*")

print(" \* \*")

print("\*\*\* \*\*\*")

print(" \* \*")

print(" \* \*")

print(" \*\*\*\*\*")

Recomendamos que **juegues con el código** que hemos escrito para ti y que realices algunas correcciones (quizás incluso destructivas). Siéntete libre de modificar cualquier parte del código, pero hay una condición: aprende de tus errores y saca tus propias conclusiones.

Intenta:

* Minimizar el número de invocaciones de la función print() insertando la secuencia \n en las cadenas.
* Hacer la flecha dos veces más grande (pero mantener las proporciones).
* Duplicar la flecha, colocando ambas flechas lado a lado; nota: una cadena se puede multiplicar usando el siguiente truco: "string" \* 2 producirá "stringstring" (te contaremos más sobre ello pronto).
* Elimina cualquiera de las comillas y observa detenidamente la respuesta de Python; presta atención a donde Python ve un error: ¿es el lugar en donde realmente existe el error?
* Haz lo mismo con algunos de los paréntesis.
* Cambia cualquiera de las palabras print en otra cosa (por ejemplo, de minúscula a mayúscula, Print) - ¿Qué sucede ahora?
* Reemplaza algunas de las comillas por apóstrofes; observa lo que pasa detenidamente.

**Literales - los datos en sí mismos**

Ahora que tienes un poco de conocimiento acerca de algunas de las poderosas características que ofrece la función print(), es tiempo de aprender sobre cuestiones nuevas, y un nuevo término - **el literal**.

**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.

Observa los siguientes dígitos:

123

¿Puedes adivinar qué valor representa? claro que puedes - es ciento veintitrés.

Que tal este:

c

¿Representa algún valor? Tal vez. Puede ser el símbolo de la velocidad de la luz, por ejemplo. También puede representar la constante de integración. Incluso la longitud de una hipotenusa en el Teorema de Pitágoras. Existen muchas posibilidades.

No se puede elegir el valor correcto sin algo de conocimiento adicional.

Y esta es la pista: 123 es un literal, y c no lo es.

Se utilizan literales **para codificar datos y ponerlos dentro del código**. Ahora mostraremos algunas convenciones que se deben seguir al utilizar Python.

**Literales - los datos en sí mismos**

Comencemos con un sencillo experimento, observa el fragmento de código en el editor.

La primera línea luce familiar. La segunda parece ser errónea debido a la falta visible de comillas.

print("2")

print(2)

Intenta ejecutarlo.

Si todo salió bien, ahora deberías de ver dos líneas idénticas.

¿Qué paso? ¿Qué significa?

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

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

La función print() los muestra exactamente de la misma manera. Sin embargo, internamente, la memoria de la computadora los almacena de dos maneras completamente diferentes. La cadena existe como eso, solo una cadena, una serie de letras.

El número es convertido a una representación maquina (una serie de bits). La función print() es capaz de mostrar ambos en una forma legible para humanos.

Vamos a tomar algo de tiempo para discutir literales numéricas y su vida interna.

**Enteros**

Quizá ya sepas un poco acerca de como las computadoras hacen cálculos con números. Tal vez has escuchado del **sistema binario**, y como es que ese es el sistema que las computadoras utilizan para almacenar números y como es que pueden realizar cualquier tipo de operaciones con ellos.

No exploraremos las complejidades de los sistemas numéricos posicionales, pero se puede afirmar que todos 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** (o simplemente **flotantes**), los cuales contienen (o son capaces de contener) una parte fraccionaría.

Esta definición no es tan precisa, pero es suficiente por ahora. La distinción es muy importante, y la frontera entre estos dos tipos de números es muy estricta. Ambos tipos difieren significativamente en como son almacenados en una computadora y en el rango de valores que aceptan.

La característica del valor numérico que determina el tipo, rango y aplicación se denomina el **tipo**.

Si se codifica un literal y se coloca dentro del código de Python, la forma del literal determina la representación (tipo) que Python utilizará para **almacenarlo en la memoria**.

Por ahora, dejemos los números flotantes a un lado (regresaremos a ellos pronto) y analicemos como es que Python reconoce un número entero.

El proceso es casi como usar lápiz y papel, es simplemente una cadena de dígitos que conforman el número, pero hay una condición, no se deben insertar caracteres que no sean dígitos dentro del número.

Tomemos, por ejemplo, el número once millones ciento once mil ciento once. Si tomaras ahorita 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.

**NOTA** \*Python 3.6 ha introducido el guion bajo en los literales numéricos, permitiendo colocar un guion bajo entre dígitos y después de especificadores de base para mejorar la legibilidad. Esta característica no está disponible en versiones anteriores de Python.

¿Cómo se codifican los números negativos en Python? Como normalmente se hace, agregando un signo de **menos**. Se puede escribir: -11111111, o -11\_111\_111.

Los números positivos no requieren un signo positivo antepuesto, pero es permitido, si se desea hacer. Las siguientes líneas describen el mismo número: +11111111 y 11111111.

**Enteros: números octales y hexadecimales**

Existen dos convenciones adicionales en Python que no son conocidas en el mundo de las matemáticas. El primero nos permite utilizar un número en su representación **octal.**

Si un número entero esta precedido por un código 0O o 0o (cero-o), el número será tratado como un valor octal. Esto significa que el número debe contener dígitos en el rango del [0..7] únicamente.

0o123 es un número **octal** con un valor (decimal) igual a 83.

La función print() realiza la conversión automáticamente. Intenta esto:

print(0o123)

La segunda convención nos permite utilizar números en **hexadecimal**. Dichos números deben ser precedidos por el prefijo 0x o 0X (cero-x).

0x123 es un número **hexadecimal** con un valor (decimal) igual a 291. La función print() puede manejar estos valores también. Intenta esto:

print(0x123)

**Flotantes**

Ahora es tiempo de hablar acerca de otro tipo, el cual esta designado para representar y almacenar los números que (como lo diría un matemático) tienen una **parte decimal no vacía**.

Son números que tienen (o pueden tener) una parte fraccionaria después del punto decimal, y aunque esta definición es muy pobre, es suficiente para lo que se desea discutir.

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

**Nota**: dos punto cinco se ve normal cuando se escribe en un programa, sin embargo, si tu idioma nativo prefiere el uso de una coma en lugar de un punto, se debe asegurar que **el número no contenga más comas**.

Python no lo aceptará, o (en casos poco probables) puede malinterpretar el número, debido a que la coma tiene su propio significado en Python.

Si se quiere utilizar solo el valor de dos punto cinco, se debe escribir como se mostró anteriormente. Nota que hay un punto entre el 2 y el 5 - no una coma.

Como puedes imaginar, el valor de **cero punto cuatro** puede ser escrito en Python como:

0.4

Pero no hay que olvidar esta sencilla regla, se puede omitir el cero cuando es el único dígito antes del punto decimal.

En esencia, el valor 0.4 se puede escribir como:

.4

Por ejemplo: el valor de 4.0 puede ser escrito como:

4.

Esto no cambiará su tipo ni su valor.

**Enteros vs. Flotantes**

El punto decimal es esencialmente importante para reconocer números punto-flotantes en Python.

Observa estos dos números:

4

4.0

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

4 es un número **entero**, mientras que 4.0 es un número **punto-flotante**.

El punto decimal es lo que determina si es flotante.

Por otro lado, no solo el punto hace que un número sea flotante. Se puede utilizar la letra e.

Cuando se desea utilizar números que son muy pequeños o muy grandes, se puede implementar la **notación científica**.

Por ejemplo, la velocidad de la luz, expresada en metros por segundo. Escrita directamente se vería de la siguiente manera: 300000000.

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

Se lee de la siguiente manera: tres por diez elevado a la octava potencia

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

3E8

La letra E (también se puede utilizar la letra minúscula e - proviene de la palabra **exponente**) la cual significa por diez a la n potencia.

**Nota**:

* El **exponente** (el valor después de la E) debe ser un valor entero.
* La **base** (el valor antes de la E) puede o no ser un valor entero.

**Codificando Flotantes**

Veamos ahora como almacenar números que son muy pequeños (en el sentido de que están muy cerca del cero).

Una constante de física denominada "La Constante de Planck" (denotada como h), de acuerdo con los libros de texto, tiene un valor de: **6.62607 x 10-34**.

Si se quisiera utilizar en un programa, se debería escribir de la siguiente manera:

6.62607E-34

**Nota**: el hecho de que se haya escogido una de las posibles formas de codificación de un valor flotante no significa que Python lo presentará de la misma manera.

Python podría en ocasiones elegir una **notación diferente**.

Por ejemplo, supongamos que se ha elegido utilizar la siguiente notación:

0.0000000000000000000001

Cuando se corre en Python:

print(0.0000000000000000000001)

Este es el resultado:

1e-22

Python siempre elige **la presentación más corta del número**, y esto se debe de tomar en consideración al crear literales.

**Cadenas**

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

Ya conoces un poco acerca de ellos, por ejemplo, que **las cadenas requieren comillas**, así como los flotantes necesitan punto decimal.

Este es un ejemplo de una cadena: "Yo soy una cadena."

Sin embargo, hay una cuestión. ¿Cómo se puede codificar una comilla dentro de una cadena que ya está delimitada por comillas?

Supongamos que se desea mostrar un muy sencillo mensaje:

Me gusta "Monty Python"

¿Cómo se puede hacer esto sin generar un error? Existen dos posibles soluciones.

La primera se basa en el concepto ya conocido del **carácter de escape**, el cual recordarás se utiliza empleando la **diagonal invertida**. La diagonal invertida puede también escapar de la comilla. Una comilla precedida por una diagonal invertida cambia su significado, no es un limitador, simplemente es una comilla. Lo siguiente funcionará como se desea:

print("Me gusta \"Monty Python\"")

**Nota**: ¿Existen dos comillas con escape en la cadena, puedes observar ambas?

La segunda solución puede ser un poco sorprendente. Python puede utilizar **una apóstrofe en lugar de una comilla**. Cualquiera de estos dos caracteres puede delimitar una cadena, pero para ello se debe ser **consistente**.

Si se delimita una cadena con una comilla, se debe cerrar con una comilla.

Si se inicia una cadena con un apóstrofe, se debe terminar con un apóstrofe.

Este ejemplo funcionará también:

print('Me gusta "Monty Python"')

**Nota**: en este ejemplo no se requiere nada de escapes.

**Codificando cadenas**

Ahora, la siguiente pregunta es: ¿Cómo se puede insertar un apóstrofe en una cadena la cual está limitada por dos apóstrofes?

A estas alturas ya se debería tener una posible respuesta o dos.

Intenta imprimir una cadena que contenga el siguiente mensaje:

I'm Monty Python.

¿Sabes cómo hacerlo? Haz clic en Revisar para saber si estas en lo cierto:

print('I\'m Monty Python.') o print("I'm Monty Python.")

Como se puede observar, la diagonal invertida es una herramienta muy poderosa, puede escapar no solo comillas, sino también apóstrofes.

Ya se ha mostrado, pero se desea hacer énfasis en este fenómeno una vez más - **una cadena puede estar vacía** - puede no contener carácter alguno.

Una cadena vacía sigue siendo una cadena:

''

""

**Valores Booleanos**

Para concluir con los literales de Python, existen dos más.

No son tan obvios como los anteriores y se emplean para representar un valor muy abstracto - **la veracidad**.

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**.

El nombre proviene de George Boole (1815-1864), el autor de Las Leyes del Pensamiento, las cuales definen el **Algebra Booleana** - una parte del algebra 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.

Afortunadamente, las computadoras solo conocen dos tipos de respuestas:

* Si, esto es verdad.
* No, esto es falso.

Nunca habrá una respuesta como: No lo sé o probablemente sí, pero no estoy seguro.

Python, es entonces, un reptil **binario**.

Estos dos valores booleanos tienen denotaciones estrictas en Python:

True

False

No se pueden cambiar, se deben tomar estos símbolos como son, incluso respetando las **mayúsculas y minúsculas**.

Reto: ¿Cuál será el resultado del siguiente fragmento de código?

print(True > False)

print(True < False)

Ejecuta el código en la terminal. ¿Puedes explicar el resultado?

**LABORATORIO**

**Tiempo Estimado**

5 minutos

**Nivel de dificultad**

Fácil

**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"""

**Puntos Clave**

1. **Literales** son notaciones para representar valores fijos en el código. Python tiene varios tipos de literales, es decir, un literal puede ser un número, por ejemplo, 123, o una cadena (por ejemplo, "Yo soy un literal.").

2. El **Sistema Binario** es un sistema numérico que emplea 2 como su base. Por lo tanto, un número binario está compuesto por 0s y 1s únicamente, por ejemplo, 1010 es 10 en decimal.

Los sistemas de numeración Octales y Hexadecimales son similares pues emplean 8 y 16 como sus bases respectivamente. El sistema hexadecimal utiliza los números decimales más seis letras adicionales.

3. **Los Enteros** (o simplemente **int**) son uno de los tipos numéricos que soporta Python. Son números que no tienen una parte fraccionaria, por ejemplo, 256, o -1 (enteros negativos).

4. Los números **Punto-Flotante** (o simplemente **flotantes**) son otro tipo numérico que soporta Python. Son números que contienen (o son capaces de contener) una parte fraccionaria, por ejemplo, 1.27.

5. Para codificar un apóstrofe o una comilla dentro de una cadena se puede utilizar el carácter de escape, por ejemplo, 'I\'m happy.', o abrir y cerrar la cadena utilizando un conjunto de símbolos distintos al símbolo que se desea codificar, por ejemplo, "I'm happy." para codificar un apóstrofe, y 'Él dijo "Python", no "typhoon"' para codificar comillas.

6. Los **Valores Booleanos** son dos objetos constantes Verdadero y Falso empleados para representar valores de verdad (en contextos numéricos 1 es True, mientras que 0 es False).

**EXTRA**

Existe un literal especial más utilizado en Python: el literal None. Este literal es llamado un objeto de NonType (ningún tipo), y puede ser utilizado para representar **la ausencia de un valor**. Pronto se hablará más acerca de ello.

**Ejercicio 1**

¿Qué tipos de literales son los siguientes dos ejemplos?

"Hola", "007"

Ambos son cadenas.

**Ejercicio 2**

¿Qué tipo de literales son los siguientes cuatro ejemplos?

"1.5", 2.0, 528, False

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

**Ejercicio 3**

¿Cuál es el valor en decimal del siguiente número en binario?

1011

Es 11, porque (2\*\*0) + (2\*\*1) + (2\*\*3) = 11

**Python como una calculadora**

Ahora, se va a mostrar un nuevo lado de la función print(). Ya se sabe que la función es capaz de mostrar los valores de los literales que le son pasados por los argumentos.

De hecho, puede hacer algo más. Observa el siguiente fragmento de código:

print(2+2)

Reescribe el código en el editor y ejecútalo. ¿Puedes adivinar la salida?

Deberías de ver el número cuatro. Tómate la libertad de experimentar con otros operadores.

Sin tomar esto con mucha seriedad, has descubierto que Python puede ser utilizado como una calculadora. No una muy útil, y definitivamente no una de bolsillo, pero una calculadora sin duda alguna.

Tomando esto más seriamente, nos estamos adentrado en el terreno de los **operadores y expresiones**.

**Los Operadores Básicos**

Un **operado**r es un símbolo del lenguaje de programación, el cual es capaz de realizar operaciones con los valores.

Por ejemplo, como en la aritmética, el signo de + (mas) es un operador el cual es capaz de **sumar** dos números, dando el resultado de la suma.

Sin embargo, no todos los operadores de Python son tan simples como el signo de más, veamos algunos de los operadores disponibles en Python, las reglas que se deben seguir para emplearlos, y como interpretar las reglas que realizan.

Se comenzará con los operadores que están asociados con las operaciones aritméticas más conocidas:

+, -, \*, /, //, %, \*\*

El orden en el que aparecen no es por casualidad. Hablaremos más de ello cuando se hayan visto todos.

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

**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**.

Las matemáticas clásicas prefieren una notación con superíndices, como el siguiente: 23. Los editores de texto puros no aceptan esa notación, por lo tanto, Python utiliza \*\* en lugar de la notación matemática, por ejemplo, 2 \*\* 3.

Observa los ejemplos en la ventana del editor.

**Nota**: En los ejemplos, los dobles asteriscos están rodeados de espacios, no es obligatorio hacerlo, pero hace que el código sea más **legible**.

Los ejemplos muestran una característica importante de los **operadores numéricos** de Python.

print(2 \*\* 3)

print(2 \*\* 3.)

print(2. \*\* 3)

print(2. \*\* 3.)

Ejecuta el código y observa cuidadosamente los resultados que arroja. ¿Puedes observar algo?

**Recuerda**: Es posible formular las siguientes reglas con base en los resultados:

* Cuando **ambos** \*\* argumentos son enteros, el resultado es entero también.
* Cuando **al menos un** \*\* argumento es flotante, el resultado también es flotante.

Esta es una distinción importante que se debe recordar.

**Operadores aritméticos: multiplicación**

Un símbolo de \* (asterisco) es un operador de **multiplicación**.

Ejecuta el código y revisa si la regla de entero vs flotante aún funciona.

print(2 \* 3)

print(2 \* 3.)

print(2. \* 3)

print(2. \* 3.)

**Operadores aritméticos: división**

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

El valor después de la diagonal es el **divisor**, el valor antes de la diagonal es el **dividendo**.

Ejecuta el código y analiza los resultados.

print(6 / 3)

print(6 / 3.)

print(6. / 3)

print(6. / 3.)

Deberías de poder observar que hay una excepción a la regla.

**El resultado producido por el operador de división siempre es flotante**, sin importar si a primera vista el resultado es flotante: 1 / 2, o si parece ser completamente entero: 2 / 1.

¿Esto ocasiona un problema? Sí, en ocasiones se podrá necesitar que el resultado de una división sea entero, no flotante.

Afortunadamente, Python puede ayudar con eso.

**Operadores aritméticos: división entera**

Un símbolo de // (doble diagonal) es un operador de **división entera**. Difiere del operador estándar / en dos detalles:

* El resultado carece de la parte fraccionaria, está ausente (para los enteros), o siempre es igual a cero (para los flotantes); esto significa que **los resultados siempre son redondeados**.
* Se ajusta a la regla *entero vs flotante*.

Ejecuta el ejemplo debajo y observa los resultados:

print(6 // 3)

print(6 // 3.)

print(6. // 3)

print(6. // 3.)

Como se puede observar, una división de entero entre entero da un **resultado entero**. Todos los demás casos producen flotantes.

Hagamos algunas pruebas más avanzadas.

Observa el siguiente fragmento de código:

print(6 // 4)

print(6. // 4)

Imagina que se utilizó / en lugar de // - ¿Podrías predecir los resultados?

Si, sería 1.5 en ambos casos. Eso está claro.

Pero, ¿Qué resultado se debería esperar con una división //?

Ejecuta el código y observa por ti mismo.

Lo que se obtiene son dos unos, uno entero y uno flotante.

El resultado de la división entera siempre se redondea al valor entero inferior más cercano del resultado de la división no redondeada.

Esto es muy importante: **el redondeo siempre va hacia abajo**.

Observa el código e intenta predecir el resultado nuevamente:

print(-6 // 4)

print(6. // -4)

**Nota**: Algunos de los valores son negativos. Esto obviamente afectara el resultado. ¿Pero cómo?

El resultado es un par de dos negativos. El resultado real (no redondeado) es -1.5 en ambo casos. Sin embargo, los resultados se redondean. El redondeo se hace hacia el valor inferior entero, dicho valor es -2, por lo tanto, los resultados son: -2 y -2.0.

**NOTA**

La división entera también se le suele llamar en inglés **floor division**. Más adelante te cruzarás con este término.

**Operadores: residuo (módulo)**

El siguiente operador es uno muy peculiar, porque no tiene un equivalente dentro de los operadores aritméticos tradicionales.

Su representación gráfica en Python es el símbolo de % (porcentaje), lo cual puede ser un poco confuso.

Piensa en el como una diagonal (operador de división) acompañado por dos pequeños círculos.

El resultado de la operación es el **residuo que queda de la división entera**.

En otras palabras, es el valor que sobra después de dividir un valor entre otro para producir un resultado entero.

**Nota**: el operador en ocasiones también es denominado **módulo** en otros lenguajes de programación.

Observa el fragmento de código e intenta predecir el resultado y después ejecútalo:

print(14 % 4)

Como puedes observar, el resultado es dos. Esta es la razón:

* 14 // 4 da como resultado un 3 → esta es la parte entera, es decir el **cociente**.
* 3 \* 4 da como resultado 12 → como resultado de **la multiplicación entre el cociente y el divisor**.
* 14 - 12 da como resultado 2 → este es el **residuo**.

El siguiente ejemplo es un poco más complicado:

print(12 % 4.5)

¿Cuál es el resultado?

3.0 - no 3 pero 3.0 (la regla aun funciona: 12 // 4.5 da 2.0; 2.0 \* 4.5 da 9.0; 12 - 9.0 da 3.0)

**Operadores: como no dividir**

Como probablemente sabes, la **división entre cero no funciona**.

No intentes:

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

**Operadores: suma**

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

De nuevo, observa el siguiente fragmento de código:

print(-4 + 4)

print(-4. + 8)

El resultado no debe de sorprenderte. Ejecuta el código y revisa los resultados.

**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**.

Esta es una gran oportunidad para mencionar una distinción muy importante entre operadores **unarios y binarios**.

En aplicaciones de resta, el **operador de resta espera dos argumentos**: el izquierdo (un **minuendo** en términos aritméticos) y el derecho (un **sustraendo**).

Por esta razón, el operador de resta es considerado uno de los operadores binarios, así como los demás operadores de suma, multiplicación y división.

Pero el operador negativo puede ser utilizado de una forma diferente, observa la última línea de código del siguiente fragmento:

print(-4 - 4)

print(4. - 8)

print(-1.1)

Por cierto: también hay un operador + unario. Se puede utilizar de la siguiente manera:

print(+2)

El operador conserva el signo de su único argumento, el de la derecha.

Aunque dicha construcción es sintácticamente correcta, utilizarla no tiene mucho sentido, y sería difícil encontrar una buena razón para hacerlo.

Observa el fragmento de código que está arriba - ¿Puedes adivinar el resultado o salida?

**Operadores y sus prioridades**

Hasta ahora, se ha tratado cada operador como si no tuviera relación con los otros. Obviamente, dicha situación tan simple e ideal es muy rara en la programación real.

También, muy seguido encontrarás más de un operador en una expresión, y entonces esta presunción ya no es tan obvia.

Considera la siguiente expresión:

2 + 3 \* 5

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

Seguramente recordaras que primero se debe multiplicar 3 por 5, mantener el 15 en tu memoria y después sumar el 2, dando como resultado el 17.

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

Python define la jerarquía de todos los operadores, y asume que los operadores de mayor jerarquía deben realizar sus operaciones antes que los de menor jerarquía.

Entonces, si se sabe que la \* tiene una mayor prioridad que la +, el resultado final debe de ser obvio.

**Operadores y sus enlaces**

El **enlace** de un operador determina el orden en que se computan las operaciones de los operadores con la misma prioridad, los cuales se encuentran dentro de una misma expresión.

La mayoría de los operadores de Python tienen un enlazado hacia la izquierda, lo que significa que el cálculo de la expresión es realizado de izquierda a derecha.

Este simple ejemplo te mostrará como funciona. Observa:

print(9 % 6 % 2)

Existen dos posibles maneras de evaluar la expresión:

* De izquierda a derecha: primero 9 % 6 da como resultado 3, y entonces 3 % 2 da como resultado 1.
* De derecha a izquierda: primero 6 % 2 da como resultado 0, y entonces 9 % 0 causa **un error fatal**.

Ejecuta el ejemplo y observa lo que se obtiene.

El resultado debe ser 1. El operador tiene un **enlazado hacia la izquierda**. Pero hay una excepción interesante.

**Operadores y sus enlaces: exponenciación**

Repite el experimento, pero ahora con exponentes.

Utiliza este fragmento de código:

print(2 \*\* 2 \*\* 3)

Los dos posibles resultados son:

2 \*\* 2 → 4; 4 \*\* 3 → 64

2 \*\* 3 → 8; 2 \*\* 8 → 256

Ejecuta el código, ¿Qué es lo que observas?

El resultado muestra claramente que **el operador de exponenciación utiliza enlazado hacia la derecha**.

**Lista de prioridades**

Como eres nuevo a los operadores de Python, no se presenta por ahora una lista completa de las prioridades de los operadores.

En lugar de ello, se mostrarán solo algunos, y se irán expandiendo conforme se vayan introduciendo operadores nuevos.

Observa la siguiente tabla:

|  |  |  |
| --- | --- | --- |
| Prioridad | Operador |  |
| 1 | +, - | unario |
| 2 | \*\* |  |
| 3 | \*, /, % |  |
| 4 | +, - | binario |

**Nota**: se han enumerado los operadores en orden **de la más alta (1) a la más baja (4) prioridad**.

Intenta solucionar la siguiente expresión:

print(2 \* 3 % 5)

Ambos operadores (\* y %) tienen la misma prioridad, el resultado solo se puede obtener conociendo el sentido del enlazado. ¿Cuál será el resultado?

1

**Operadores y paréntesis**

Por supuesto, 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**.

Se pueden emplear tantos paréntesis como se necesiten, y seguido son utilizados para **mejorar la legibilidad** de una expresión, aun si no cambian el orden de las operaciones.

Un ejemplo de una expresión con múltiples paréntesis es la siguiente:

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

Intenta calcular el valor que se calculará en la consola. ¿Cuál es el resultado de la función print()?

10.0

**Puntos Clave**

1. Una **expresión** es una combinación de valores (o variables, operadores, llamadas a funciones, aprenderás de ello pronto) las cuales son evaluadas y dan como resultado un valor, por ejemplo, 1+2.

2. Los **operadores** son símbolos especiales o palabras clave que son capaces de operar en los valores y realizar operaciones matemáticas, por ejemplo, el \* multiplica dos valores: x\*y.

3. Los operadores aritméticos en Python: + (suma), - (resta), \* (multiplicación), / (división clásica: regresan un flotante si uno de los valores es de este tipo), % (módulo: divide el operando izquierdo entre el operando derecho y regresa el residuo de la operación, por ejemplo, 5%2=1), \*\* (exponenciación: el operando izquierdo se eleva a la potencia del operando derecho, por ejemplo, 2\*\*3=2\*2\*2=8), // (división entera: retorna el número resultado de la división, pero redondeado al número entero inferior más cercano, por ejemplo, 3//2.0=1.0).

4. Un operador **unario** es un operador con solo un operando, por ejemplo, -1, o +3.

5. Un operador **binario** es un operador con dos operados, por ejemplo, 4+5, o 12%5.

6. Algunos operadores actúan antes que otros, a esto se le llama - **jerarquía de prioridades**:

* Unario + y - tienen la prioridad más alta.
* Después: \*\*, después: \*, /, y %, y después la prioridad más baja: binaria + y -.

7. Las sub-expresiones dentro de **paréntesis** siempre se calculan primero, por ejemplo, 15-1\*(5\*(1+2))=0.

8. Los operadores de **exponenciación** utilizan **enlazado hacia la derecha**, por ejemplo, 2\*\*2\*\*3=256.

**Ejercicio 1**

¿Cuál es la salida del siguiente fragmento de código?

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

16 8.0 8

**Ejercicio 2**

¿Cuál es la salida del siguiente fragmento de código?

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

-0.5 0.5 0 -1

**Ejercicio 3**

¿Cuál es la salida del siguiente fragmento de código?

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

-2 2 512

**¿Qué son las Variables?**

Es justo que Python nos permita codificar literales, las cuales contengan valores numéricos y cadenas.

Ya hemos visto que se pueden hacer operaciones aritméticas con estos números: sumar, restar, etc. Esto se hará una infinidad de veces en un programa.

Pero es normal preguntar como es que se pueden **almacenar los resultados** de estas operaciones, para poder emplearlos en otras operaciones, y así sucesivamente.

¿Cómo almacenar los resultados intermedios, y después utilizarlos de nuevo para producir resultados subsecuentes?

Python ayudará con ello. Python ofrece "cajas" (contenedores) especiales para este propósito, estas cajas son llamadas **variables** - el nombre mismo sugiere que el contenido de estos contenedores puede variar en casi cualquier forma.

¿Cuáles son los componentes o elementos de una variable en Python?

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

Comencemos con lo relacionado al nombre de la variable.

Las variables no aparecen en un programa automáticamente. Como desarrollador, tú debes decidir cuantas variables deseas utilizar en tu programa.

También las debes de nombrar.

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

* El nombre de la variable debe de estar compuesto por MAYUSCULAS, 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 puede ser igual a alguna de las palabras reservadas de Python (se explicará más de esto pronto).



**Nombres correctos e incorrectos de variables**

Nota que la misma restricción aplica a los nombres de funciones.

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.

Aquí se muestran algunos nombres de variable que son correctos, pero que no siempre son convenientes:

MiVariable, i, t34, Tasa\_Cambio, contador, DiasParaNavidad, ElNombreEsTanLargoQueSeCometeranErroresConEl, \_.

Además, Python permite utilizar no solo las letras latinas, sino caracteres específicos de otros idiomas que utilizan otros alfabetos.

Estos nombres de variables también son correctos:

Adiós\_Señora, sûr\_la\_mer, Einbahnstraße, переменная.

Ahora veamos algunos **nombres incorrectos**:

10t (no comienza con una letra), Tasa Cambio (contiene un espacio).

**Palabras Clave**

Observa las palabras que juegan un papel muy importante en cada programa de Python.

['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']

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.

El significado de la palabra reservada está **predefinido**, y no debe cambiar.

Afortunadamente, debido al hecho de que Python es sensible a mayúsculas y minúsculas, cualquiera de estas palabras se puede modificar cambiando una o varias letras de mayúsculas a minúsculas o viceversa, creando una nueva palabra, la cual no está reservada.

Por ejemplo - **no se puede nombrar** a la variable así:

import

No se puede tener una variable con ese nombre, está prohibido, pero se puede hacer lo siguiente:

Import

Estas palabras podrían parecer un misterio ahorita, pero pronto se aprenderá acerca de su significado.

**Creando variables**

¿Qué se puede poner dentro de una variable?

Cualquier cosa.

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

El valor de la variable en lo que se ha puesto dentro de ella. Puede variar tanto como se necesite o requiera. El valor puede ser entero, después flotante, y eventualmente ser una cadena.

Hablemos de dos cosas importantes - **como son creadas las variables**, y **como poner valores dentro de ellas** (o, mejor dicho, como dar o **pasarles valores**).

**RECUERDA**

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

Si se le asigna cualquier valor a una variable no existente, la variable será **automáticamente creada**. No se necesita hacer algo más.

La creación (o su sintaxis) 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**.

Observa el siguiente fragmento de código:

var = 1

print(var)

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.

**Nota**: print() tiene una función más y puede manejar variables también. ¿Puedes predecir cual será la salida (resultado) del código?

1

**Utilizando variables**

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

var = 1

balance\_cuenta = 1000.0

nombreCliente = 'John Doe'

print(var, balance\_cuenta, nombreCliente)

print(var)

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 **ocasionara un error**:

var = 1

print(Var)

Se ha tratado de utilizar la variable llamada Var, la cual no tiene ningún valor (nota: var y Var son entidades diferentes, y no tienen nada en común dentro de Python).

**RECUERDA**

Se puede utilizar print() para combinar texto con variables utilizando el operador + para mostrar cadenas con variables, por ejemplo:

var = "3.7.1"

print("Versión de Python: " + var)

¿Puedes predecir la salida del fragmento de código?

Versión de Python: 3.7.1

**Asignar un valor nuevo a una variable ya existente**

¿Cómo se le asigna un valor nuevo a una variable que ya ha sido creada? De la misma manera. Solo se necesita el signo de igual.

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.

Asigna el valor del argumento de la derecha al de la izquierda, aún cuando el argumento de la derecha sea una expresión arbitraria compleja que involucre literales, operadores y variables definidas anteriormente.

Observa el siguiente código:

var = 1

print(var)

var = var + 1

print(var)

El código envía dos líneas a la consola:

1

2

La primera línea del código **crea una nueva variable** llamada var y le asigna el valor de 1.

La declaración se lee de la siguiente manera: asigna el valor de 1 a una variable llamada var.

De manera más corta: asigna 1 a var.

Algunos prefieren leer el código así: var se convierte en 1.

La tercera **línea le asigna a la misma variable un nuevo valor** tomado de la variable misma, sumándole 1. Al ver algo así, un matemático probablemente protestaría, ningún valor puede ser igualado a si mismo más uno. Esto es una contradicción. Pero Python trata el signo = no como igual a, sino como asigna un valor.

Entonces, ¿Cómo se lee esto en un programa?

Toma el valor actual de la variable var, súmale 1 y guárdalo en la variable var.

En efecto, el valor de la variable var ha sido **incrementado** por uno, lo cual no está relacionado con comparar la variable con otro valor.

¿Puedes predecir cuál será el resultado del siguiente fragmento de código?

var = 100

var = 200 + 300

print(var)

500 - ¿Por qué? Bueno, primero, la variable var es creada y se le asigna el valor de 100. Después, a la misma variable se le asigna un nuevo valor: el resultado de sumarle 200 a 300, lo cual es 500.

**Resolviendo problemas matemáticos simples**

Ahora deberías de ser capaz de construir un corto programa el cual resuelva 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)

**Nota**: se necesita hacer uso del operador \*\* para evaluar la raíz cuadrada:

√ (x) = x(½)

y

c = √ a2 + b2

¿Puedes predecir la salida del código?

a = 3.0

b = 4.0

c = (a \*\* 2 + b \*\* 2) \*\* 0.5

print("c =", c)

Revisa abajo y ejecuta el código en el editor para confirmar tus predicciones.

c = 5.0

**LABORATORIO**

**Tiempo Estimado**

10 minutos

**Nivel de dificultad**

Fácil

**Objetivos**

* Familiarizarse con el concepto de almacenar y trabajar con diferentes tipos de datos en Python.
* Experimentar con el código en Python.

**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 totalManzanas y se debe igualar a la suma de las tres variables anteriores.
* Imprime el valor almacenado en totalManzanas 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 totalManzanas.

**Operadores Abreviados**

Es tiempo de explicar el siguiente conjunto de operadores que harán la vida del programador/desarrollador más fácil.

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

Por ejemplo, si se necesita calcular una serie de valores sucesivos de la potencia de 2, se puede usar el siguiente código:

x = x \* 2

También, puedes utilizar una expresión como la siguiente si no puedes dormir y estas tratando de resolverlo con alguno de los métodos tradicionales:

oveja = oveja + 1

Python ofrece una manera más corta de escribir operaciones como estas, lo cual se puede codificar de la siguiente manera:

x \*= 2

oveja += 1

A continuación, se intenta presentar una descripción general para este tipo de operaciones.

Si op es un operador de dos argumentos (esta es una condición muy importante) y el operador es utilizado en el siguiente contexto:

variable = variable op expresión

Puede ser simplificado de la siguiente manera:

variable op= expresión

Observa los siguientes ejemplos. Asegúrate de entenderlos todos.

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

**LABORATORIO**

**Tiempo estimado**

10 minutos

**Nivel de dificultad**

Fácil

**Objetivos**

* Familiarizarse con el concepto de variables y trabajar con ellas.
* Realizar operaciones básicas y conversiones.
* Experimentar con el código de Python.

**Escenario**

kilometros = 12.25

millas = 7.38

millas\_a\_kilometros = millas \* 1.61

kilometros\_a\_millas = kilometros / 1.61

print(millas, " millas son ", round(millas\_a\_kilometros, 2), " kilómetros ")

print(kilómetros, " kilómetros son ", round(kilometros\_a\_millas, 2), " millas ")

Millas y kilómetros son unidades de longitud o distancia.

Teniendo en mente que 1 equivale aproximadamente a 1.61 kilómetros, complemente 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 está 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, millas).

**CONSEJO**

Hay una cosa interesante más que está ocurriendo. ¿Puedes ver otra función dentro de la función print() ? Es la función round(). Su trabajo es redondear la salida del resultado al número de decimales especificados en el paréntesis, y regresar un valor flotante (dentro de la función round() se puede encontrar el nombre de la variable, el nombre, una coma, y el número de decimales que se desean mostrar). Se hablará más de esta función muy pronto, no te preocupes si no todo queda muy claro. Solo se quiere impulsar tu curiosidad.

Después de completar el laboratorio, abre Sandbox (el arenero), y experimenta más. Intenta escribir diferentes convertidores, por ejemplo, un convertidor de USD a EUR, un convertidor de temperatura, etc. Y ¡deja que tu imaginación vuele! Intenta mostrar los resultados combinando cadenas y variables. Intenta utilizar y experimentar con la función round() para redondear tus resultados a uno, dos o tres decimales. Revisa que es lo que sucede si no se provee un dígito al redondear. Recuerda probar tus programas.

Experimenta, saca tus propias conclusiones, y aprende. Se curioso.

**Resultado Esperado**

7.38 millas son 11.88 kilómetros

12.25 kilómetros son 7.61 millas

**LABORATORIO**

**Tiempo Estimado**

10-15 minutos

**Nivel de Dificultad**

Fácil

**Objetivos**

* Familiarizarse con los conceptos de números, operadores y operaciones aritméticas en Python.
* Realizar cálculos básicos.

**Escenario**

# codifica aquí tus datos de prueba.

x = "0"

x = float(x)

# escribe tu código aquí.

y = 3 \* x \*\* 3 - 2 \* x \*\* 2 + 3 \* x - 1

print("y =", y)

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.

**Prueba de Datos**

Datos de Muestra

x = 0

x = 1

x = -1

Salida Esperada

y = -1.0

y = 3.0

y = -9.0

**Puntos Clave**

1. Una **variable** es una ubicación nombrada reservada para almacenar valores en la memoria. Una variable es creada o inicializada automáticamente cuando se le asigna un valor por primera vez.

2. Cada variable debe de tener un nombre único - un **identificador**. Un nombre válido debe ser aquel que no contiene espacios, debe comenzar con un guion bajo (\_), o una letra, y no puede ser una palabra reservada de Python. El primer carácter puede estar seguido de guiones bajos, letras, y dígitos. Las variables en Python son sensibles a mayúsculas y minúsculas.

3. Python es un lenguaje **de tipo dinámico**, lo que significa que no se necesita declarar variables en él. Para asignar valores a las variables, se utiliza simplemente el operador de asignación, es decir el signo de igual (=) por ejemplo, var = 1.

4. También es posible utilizar **operadores de asignación compuesta** (operadores abreviados) para modificar los valores asignados a las variables, por ejemplo, var += 1, o var /= 5 \* 2.

5. Se les puede asignar valores nuevos a variables ya existentes utilizando el operador de asignación o un operador abreviado:

var = 2

print(var)

var = 3

print(var)

var += 1

print(var)

6. Se puede combinar texto con variables empleando el operador +, y utilizando la función print() para mostrar o imprimir los resultados, por ejemplo:

var = "007"

print("Agente " + var)

**Ejercicio 1**

¿Cuál es el resultado del siguiente fragmento de código?

var = 2

var = 3

print(var)

3

**Ejercicio 2**

¿Cuáles de los siguientes nombres de variables son ilegales en Python?

my\_var

m

101

averylongvariablename

m101

m 101

Del

del

my\_var

m

101 # incorrecto (comienza con un digito)

averylongvariablename

m101

m 101 # incorrecto (contiene un espacio)

Del

del # incorrecto (es una palabra clave)

**Ejercicio 3**

¿Cuál es el resultado del siguiente fragmento de código?

a = '1'

b = "1"

print(a + b)

11

**Ejercicio 4**

¿Cuál es el resultado del siguiente fragmento de código?

a = 6

b = 3

a /= 2 \* b

print(a)

1.0

2 \* b = 6

a = 6 → 6 / 6 = 1.0

**Poner comentarios en el código: ¿por qué, ¿cuándo y dónde?**

Quizá en algún momento será necesario poner algunas palabras en el código dirigidas no a Python, sino a las personas quienes estén leyendo el código con el fin de explicarles como es que funciona, o tal vez especificar el significado de las variables, también para documentar quien es el autor del programa y en que fecha fue escrito.

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

¿Cómo se colocan este tipo de comentarios en el código fuente? Tiene que ser hecho de cierta manera para que Python no intente interpretarlo como parte del código.

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.

Si se desea colocar un comentario que abarca varias líneas, se debe colocar este símbolo en cada línea.

Justo como el siguiente código:

# Este 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)

Los desarrolladores buenos y responsables **describen cada pieza importante de código**, por ejemplo, el explicar el rol de una variable; aunque la mejor manera de comentar una variable es dándole un nombre que no sea ambiguo.

Por ejemplo, si una variable determinada está diseñada para almacenar el área de un cuadrado, el nombre areaCuadrado será muchísimo mejor que tiaJuana.

El primer nombre dado a la variable se puede definir como **auto-comentable**.

Los comentarios pueden ser útiles en otro aspecto, se pueden utilizar para **marcar un fragmento de código que actualmente no se necesita**, cual sea la razón. Observa el siguiente ejemplo, sí se descomenta la línea resaltada, esto afectara la salida o resultado del código:

# Este es un programa de prueba

x = 1

y = 2

# y = y + x

print(x + y)

Esto es frecuentemente realizado cuando se está probando un programa, con el fin de aislar un fragmento de código donde posiblemente se encuentra un error.

**LABORATORIO**

**Tiempo Estimado**

5 minutos

**Nivel de Dificultad**

Muy Fácil

**Objetivos**

* Familiarizarse con el concepto de comentarios en Python.
* Utilizar y no utilizar los comentarios.
* Reemplazar los comentarios con código.
* Experimentar con el código de Python.

**Escenario**

# este programa calcula los segundos en cierto número de horas determinadas

# este programa fue escrito hace dos días

a = 2 # número de horas

segundos = 3600 # número de segundos en una hora

print("Horas: ", a) # imprime el número de horas

# print("Segundos en horas: ", a \* segundos) # se imprime el número de segundos en determinado número de horas

# aquí también se debe de imprimir un "Adiós", pero el programador no tuvo tiempo de escribirlo

# este él es fin del programa que calcula el número de segundos en 2 horas

El código en el editor contiene comentarios. Intenta mejorarlo: agrega o quita comentarios donde consideres que sea apropiado (en ocasiones el remover un comentario lo hace más legible), además, cambia el nombre de las variables donde consideres que esto mejorará la comprensión del código.

**NOTA**

Los comentarios son muy importantes. No solo hacen que el programa sea **más fácil de entender**, también sirven para **deshabilitar aquellas partes de código que no son necesarias** (por ejemplo, cuando se necesita probar cierta parte del código, e ignorar el resto). Los buenos programadores **describen** cada parte importante del código, y dan **nombres significativos** a variables, debido a que en ocasiones es mucho más sencillo dejar el comentario dentro del código mismo.

Es bueno utilizar nombres de variables **legibles**, y en ocasiones es mejor **dividir el código** en partes con nombres (por ejemplo, en funciones). En algunas situaciones, es una buena idea escribir los pasos de como se realizaron los cálculos de una forma sencilla y clara.

Una cosa más: puede ocurrir que un comentario contenga una pieza de información incorrecta o errónea, nunca se debe de hacer eso a propósito.

**Puntos Clave**

1. Los comentarios pueden ser utilizados para colocar información adicional en el código. Son omitidos al momento de la ejecución. Dicha información es para los lectores que están manipulando el código. En Python, un comentario es un fragmento de texto que comienza con un #. El comentario se extiende hasta el final de la línea.

2. Si deseas colocar un comentario que abarque varias líneas, es necesario colocar un # al inicio de cada línea. Además, se puede utilizar un comentario para marcar un fragmento de código que no es necesario en el momento y no se desea ejecutar. (observa la última línea de código del siguiente fragmento), por ejemplo:

# Este programa imprime

# un saludo en pantalla

print("Hola!") # Se invoca la función print()

# print("Soy Python.")

3. Cuando sea posible, se deben **auto comentar** **los nombres** de las variables, por ejemplo, si se están utilizando dos variables para almacenar la altura y longitud de algo, los nombres altura y longitud son una mejor elección que mivar1 y mivar2.

4. Es importante utilizar los comentarios para que los programas sean más fáciles de entender, además de emplear variables legibles y significativas en el código. Sin embargo, es igualmente importante **no utilizar** nombres de variables que sean confusos, o dejar comentarios que contengan información incorrecta.

5. Los comentarios pueden ser muy útiles cuando tu estas leyendo tu propio código después de un tiempo (es común que los desarrolladores olviden lo que su propio código hace), y cuando otros están leyendo tu código (les puede ayudar a comprender que es lo que hacen tus programas y como es que lo hacen).

**Ejercicio 1**

¿Cuál es la salida del siguiente fragmento de código?

# print("Cadena #1")

print("Cadena #2")

Cadena #2

**Ejercicio 2**

¿Qué ocurrirá cuando se ejecute el siguiente código?

# Esto es

un comentario

en varias líneas #

print("Hola!")

SyntaxError: invalid syntax

**La función input()**

Ahora se introducirá una nueva función, la cual pareciese ser 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 se llama input(). El nombre de la función lo dice todo.

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

El programa entonces puede manipular los datos, haciendo que el código sea verdaderamente interactivo.

Todos los programas **leen y procesan datos**. Un programa que no obtiene datos de entrada del usuario es un **programa sordo**.

Observa el ejemplo:

print("Dime algo...")

algo = input()

print("Mmm...", algo, "...¿en serio?")

Se muestra un ejemplo muy sencillo de como utilizar la función input().

**Nota**:

* El programa **solicita al usuario que inserte algún dato** desde la consola (seguramente utilizando el teclado, aunque también es posible introducir datos utilizando la voz o alguna imagen).
* La función input() es invocada sin argumentos (es la manera más sencilla de utilizar la función); la función **pondrá la consola en modo de entrada**; aparecerá un cursor que parpadea, y podrás introducir datos con el teclado, al terminar presiona la tecla Enter; todos los datos introducidos serán **enviados al programa** a través del resultado de la función.
* Nota: el resultado debe ser asignado a una variable; esto es crucial, si no se hace los datos introducidos se perderán.
* Después se utiliza la función print() para mostrar los datos que se obtuvieron, con algunas observaciones adicionales.

Intenta ejecutar el código y permite que la función te muestre lo que puede hacer.

**La función input() con un argumento**

La función input() puede hacer algo más: puede mostrar un mensaje al usuario sin la ayuda de la función print().

Se ha modificado el ejemplo un poco, observa el código:

algo = input("Dime algo...")

print("Mmm...", algo, "...¿En serio?")

**Nota**:

* 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.

Esta variante de la invocación de la función input() simplifica el código y lo hace más claro.

**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**.

Una cadena que contiene todos los caracteres que el usuario introduce desde el teclado. No es un entero ni un flotante.

Esto significa que **no se debe utilizar como un argumento para operaciones matemáticas**, por ejemplo, no se pueden utilizar estos datos para elevarlos al cuadrado, para dividirlos entre algo o por algo.

cualquierNumero = input("Inserta un número: ")

algo = cualquierNumero \*\* 2.0

print(cualquierNumero, "al cuadrado es", algo)

**La función input() - operaciones prohibidas**

Observa el código en el editor. Ejecútalo, inserta cualquier número, y oprime Enter.

# Probando mensajes de error

algo = input("Inserta un número: ")

resultado = algo \*\* 2.0

print(algo, "al cuadrado es ", resultado)

¿Qué es lo que ocurre?

Python debió haberte dado la siguiente salida:

Traceback (most recent call last):

File ".main.py", line 4, in <module>

resultado = algo \*\* 2.0

TypeError: unsupported operand type(s) for \*\* or pow(): 'str' and 'float'

La última línea lo explica todo, se intentó aplicar el operador \*\* a 'str' (una cadena) acompañado por un 'float' (valor flotante).

Esto está prohibido.

Esto debe de ser obvio ¿Puedes predecir el valor de "ser o no ser" elevado a la 2 potencia?

No podemos. Python tampoco puede.

¿Habremos llegado a un punto muerto? ¿Existirá alguna solución? Claro que la hay.

**Conversión de datos o casting**

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

Sus nombres indican cuál es su función:

* La función int() **toma un argumento** (por ejemplo, una cadena: int(string)) e intenta convertirlo a un valor entero; si llegase a fallar, el programa entero fallará también (existe una manera de solucionar esto, se explicará más adelante).
* La función float() toma un argumento (por ejemplo, una cadena: float(string)) e intenta convertirlo a flotante (el resto es lo mismo).

Esto es muy simple y muy efectivo. Sin embargo, estas funciones se pueden invocar directamente pasando el resultado de la función input() directamente. No hay necesidad de emplear variables como almacenamiento intermedio.

Se ha implementado esta idea en el editor, observa el código.

¿Puedes imaginar como la cadena introducida por el usuario fluye desde la función input() hacía la función print()?

algo = float(input("Inserta un número: "))

resultado = algo \*\* 2.0

print(algo, "al cuadrado es", resultado)

Intenta ejecutar el código modificado. No olvides introducir un **número valido**.

Prueba con diferentes valores, pequeños, grandes, negativos y positivos. El cero también es un buen valor a introducir.

**Más acerca de la función input() y tipos de conversión**

El tener un equipo compuesto por input()-int()-float() abre muchas nuevas posibilidades.

Eventualmente serás capaz de escribir programas completos, los cuales acepten datos en forma de números, los cuales serán procesados y se mostrarán los resultados.

Por supuesto, estos programas serán muy primitivos y no muy utilizables, debido a que no pueden tomar decisiones, y consecuentemente no son capaces de reaccionar acorde a cada situación.

Sin embargo, esto no es un problema; se explicará como solucionarlo pronto.

El siguiente ejemplo hace referencia al programa anterior que calcula la longitud de la hipotenusa. Vamos a reescribirlo, para que pueda leer las longitudes de los catetos desde la consola.

Revisa la ventana del editor, así es como se ve ahora.

Este programa le preguntó al usuario los dos catetos, calcula la hipotenusa e imprime el resultado.

cateto\_a = float(input("Inserta la longitud del primer cateto: "))

cateto\_b = float(input("Inserta la longitud del segundo cateto: "))

hipo = (cateto\_a\*\*2 + cateto\_b\*\*2) \*\* .5

print("La longitud de la hipotenusa es: ", hipo)

Ejecútalo de nuevo e intenta introducir valores negativos.

El programa desafortunadamente, no reacciona correctamente a este error.

Vamos a ignorar esto por ahora. Regresaremos a ello pronto.

Debido a que la función print() acepta una expresión como argumento, se puede **quitar la variable** del código.

Como se muestra en el siguiente código:

cateto\_a = float(input("Inserta la longitud del primer cateto: "))

cateto\_b = float(input("Inserta la longitud del segundo cateto "))

print("La longitud de la hipotenusa es: ", (cateto\_a\*\*2 + cateto\_b\*\*2) \*\* .5)

**Operadores de cadenas - introducción**

Es tiempo de regresar a estos dos operadores aritméticos: + y \*.

Ambos tienen una función secundaría. Son capaces de hacer algo más que **sumar y multiplicar**.

Los hemos visto en acción cuando sus argumentos son (flotantes o enteros).

Ahora veremos que son capaces también de manejar o manipular cadenas, aunque, en una manera muy específica.

**Concatenación**

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

string + string

Simplemente **concatena** (junta) dos cadenas en una. Además, puede ser utilizado más de una vez en una misma expresión.

En contraste con el operador aritmético, el operador de concatenación no es **conmutativo**, por ejemplo, "ab" + "ba" no es lo mismo que "ba" + "ab".

No olvides, si se desea que el signo + sea un **concatenador**, no un sumador, solo se debe asegurar que **ambos argumentos sean cadenas**.

No se pueden mezclar los tipos de datos aquí.

Este es un programa sencillo que muestra como funciona el signo + como concatenador:

nom = input("¿Me puedes dar tu nombre por favor? ")

ape = input("¿Me puedes dar tu apellido por favor? ")

print("Gracias.")

print("\nTu nombre es " + nom + " " + ape + ".")

**Nota**: El utilizar + para concatenar cadenas te permite construir la salida de una manera más precisa, en comparación de utilizar únicamente la función print(), aun cuando se enriquezca con los argumentos end= y sep=.

Ejecuta el código y comprueba si la salida es igual a tus predicciones.

**Replicación**

El signo de \* (asterisco), cuando es aplicado a una cadena y a un número (o a un número y cadena) se convierte en un **operador de replicación**.

cadena \* número

número \* cadena

Replica la cadena el número de veces indicado por el número.

Por ejemplo:

* "James" \* 3 nos da "JamesJamesJames".
* 3 \* "an" nos da "ananan".
* 5 \* "2" (o "2" \* 5) da como resultado "22222" (no 10).

**RECUERDA**

Un número menor o igual que cero produce una **cadena vacía**.

Este sencillo programa "dibuja" un rectángulo, haciendo uso del operador (+), pero en un nuevo rol:

print("+" + 10 \* "-" + "+")

print(("|" + " " \* 10 + "|\n") \* 5, end="")

print("+" + 10 \* "-" + "+")

Nota como se ha utilizado el paréntesis en la segunda línea de código.

¡Intenta practicar para crear otras figuras o tus propias obras de arte!

**Conversión de tipos de datos: str()**

A estas alturas ya sabes como emplear las funciones int() y float() para convertir una cadena a un número.

Este tipo de conversión no es en un solo sentido. También se puede **convertir un número a una cadena**, lo cual es más fácil y rápido, esta operación es posible hacerla siempre.

Una función capaz de hacer esto se llama str():

str(número)

Sinceramente, puede hacer mucho más que transformar números en cadenas, eso lo veremos después.

**El "triángulo rectángulo" de nuevo**

Este es el programa del "triángulo rectángulo" visto anteriormente:

cateto\_a = float(input("Ingresa la longitud del primer cateto: "))

cateto\_b = float(input("Ingresa la longitud del segundo cateto: "))

print("La longitud de la hipotenusa es: " + str((cateto\_a\*\*2 + cateto\_b\*\*2) \*\* .5))

Se ha modificado un poco para mostrar cómo es que la función str() trabaja. Gracias a esto, podemos **pasar el resultado entero a la función** print() **como una sola cadena**, sin utilizar las comas.

Has hecho algunos pasos importantes en tu camino hacia la programación de Python.

Ya conoces los tipos de datos básicos y un conjunto de operadores fundamentales. Sabes cómo organizar la salida y cómo obtener datos del usuario. Estos son fundamentos muy sólidos para el Módulo 3. Pero antes de pasar al siguiente módulo, hagamos unos cuantos laboratorios y resumamos todo lo que has aprendido en esta sección.

**LABORATORIO**

**Tiempo Estimado**

5-10 minutos

**Nivel de Dificultad**

Fácil

**Objetivos**

* Familiarizarse con la entrada y salida de datos en Python.
* Evaluar expresiones simples.

**Escenario**

# ingresa un valor flotante para la variable a aquí

a = 3.0

# ingresa un valor flotante para la variable b aquí

b = 2.5

# muestra el resultado de la suma aquí

print("El resultado de la suma es: " + str(a + b))

# muestra el resultado de la resta aquí

print("El resultado de la resta es: " + str(a - b))

# muestra el resultado de la multiplicación aquí

print("El resultado de la multiplicación es: " + str(a + b))

# muestra el resultado de la división aquí

print("El resultado de la división es: ", a / b)

print("\n¡Eso es todo, amigos!")

La tarea es completar el código para evaluar y mostrar el resultado de cuatro operaciones aritméticas básicas.

El resultado debe ser mostrado en consola.

Quizá no podrás proteger el código de un usuario que intente dividir entre cero. Por ahora, no hay que preocuparse por ello.

Prueba tu código - ¿Produce los resultados esperados?

**LABORATORIO**

**Tiempo estimado**

20 minutos

**Nivel de dificultad**

Intermedio

**Objetivos**

* Familiarizarse con los conceptos de números, operadores y expresiones aritméticas en Python.
* Comprender la precedencia y asociatividad de los operadores de Python, así como el correcto uso de los paréntesis.

**Escenario**

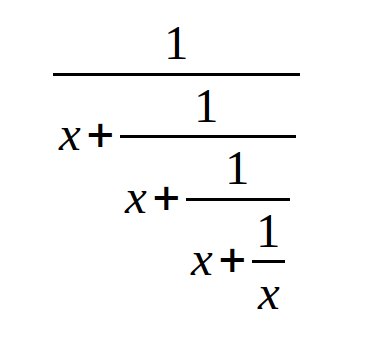
x = float(input("Ingresa el valor para x: "))

# coloca tu código aquí

Rpta. y = 1 / ( x + 1 / (x + 1 / (x + 1 / x)) )

print("y =", y)

La tarea es completar el código para poder evaluar la siguiente expresión:



El resultado debe de ser asignado a y. Se cauteloso, observa los operadores y priorízalos. Utiliza cuantos paréntesis sean necesarios.

Puedes utilizar variables adicionales para acortar la expresión (sin embargo, no es muy necesario). Prueba tu código cuidadosamente.

Datos de Prueba

Entrada de muestra: 1

Salida esperada:

y = 0.6000000000000001

Entrada de muestra: 10

Salida esperada:

y = 0.09901951266867294

Entrada de muestra: 100

Salida esperada:

y = 0.009999000199950014

Entrada de muestra: -5

Salida esperada:

y = -0.19258202567760344

**LABORATORIO**

**Tiempo estimado**

15-20 minutos

**Nivel de dificultad**

Fácil

**Objetivos**

* Mejorar la habilidad de implementar números, operadores y operaciones aritméticas en Python.
* Utilizar la función print() y sus capacidades de formateo.
* Aprender a expresar fenómenos del día a día en términos de un lenguaje de programación.

**Escenario**

hora = int(input("Hora de inicio (horas): "))

min = int(input("Minuto de inicio (minutos): "))

dura = int(input("Duración del evento (minutos): "))

# coloca tu código aquí

hora = (((hora \* 60) + min + dura) // 60 ) % 24

min = (min + dura) % 60

print(hora, min, sep=":")

La tarea es preparar un código simple para evaluar o encontrar el tiempo final de un periodo de tiempo dado, expresándolo en horas y minutos. Las horas van de 0 a 23 y los minutos de 0 a 59. El resultado debe ser mostrado en la consola.

Por ejemplo, si el evento comienza a las 12:17 y dura 59 minutos, terminará a las 13:16.

No te preocupes si tu código no es perfecto, está bien si acepta una hora inválida, lo más importante es que el código produzca una salida correcta acorde a la entrada dada.

Prueba el código cuidadosamente. Pista: utilizar el operador % puede ser clave para el éxito.

Datos de Prueba

Entrada de muestra:

12

17

59

Salida esperada: 13:16

Entrada de muestra:

23

58

642

Salida esperada: 10:40

Entrada de muestra:

0

1

2939

Salida esperada: 1:0

**Puntos Clave**

1. La función print() **envía datos a la consola**, mientras que la función input() **obtiene datos de la consola**.

2. La función input() viene con un parámetro inicial: **un mensaje de tipo cadena para el usuario**. Permite escribir un mensaje antes de la entrada del usuario, por ejemplo:

nombre = input("Ingresa tu nombre: ")

print("Hola, " + nombre + ". ¡Un gusto conocerte!")

3. Cuando la función input() es llamada o invocada, el flujo del programa se detiene, el símbolo del cursor se mantiene parpadeando (le está indicando al usuario que tome acción ya que la consola está en modo de entrada) hasta que el usuario haya ingresado un dato y/o haya presionado la tecla Enter.

**NOTA**

Puedes probar la funcionalidad completa de la función input() localmente en tu máquina. Por razones de optimización, se ha limitado el máximo número de ejecuciones en Edube a solo algunos segundos únicamente. Ve a Sandbox, copia y pega el código que está arriba, ejecuta el programa y espera unos segundos. Tu programa debe detenerse después de unos segundos. Ahora abre IDLE, y ejecuta el mismo programa ahí -¿Puedes notar alguna diferencia?

Consejo: La característica mencionada anteriormente de la función input() puede ser utilizada para pedirle al usuario que termine o finalice el programa. Observa el siguiente código:

nombre = input("Ingresa tu nombre: ")

print("Hola, " + nombre + ". ¡Un gusto conocerte!")

print("\nPresiona la tecla Enter para finalizar el programa.")

input()

print("FIN.")

3. El resultado de la función input() es una cadena. Se pueden unir cadenas unas con otras a través del operador de concatenación (+). Observa el siguiente código:

num1 = input("Ingresa el primer número: ") # Ingresa 12

num2 = input("Ingresa el segundo número: ") # Ingresa 21

print(num1 + num2) # el programa regresa 1221

4. También se pueden multiplicar (\* - replicación) cadenas, por ejemplo:

miEntrada = ("Ingresa Algo: ") # Ejemplo: hola

print(miEntrada \* 3) # Salida esperada: holaholahola

**Ejercicio 1**

¿Cuál es la salida del siguiente código?

x = int(input("Ingresa un número: ")) # el usuario ingresa un 2

print(x \* "5")

55

**Ejercicio 2**

¿Cuál es la salida esperada del siguiente código?

x = input("Ingresa un número: ") # el usuario ingresa un 2

print(type(x))

<class 'str'>

**¡Felicidades! Has completado el Módulo 2**

¡Bien hecho! Has llegado al final del Módulo 2 y has completado un paso importante en tu educación de programación en Python. Aquí hay un breve resumen de los objetivos que has cubierto y con los que te has familiarizado en el Módulo 2:

* Los métodos básicos de formateo y salida de datos ofrecidos por Python, junto con los tipos principales de datos y operadores numéricos, sus relaciones mutuas y enlaces.
* El concepto de variables y la manera correcta de darles nombre.
* El operador de asignación, las reglas que rigen la construcción de expresiones.
* La entrada y conversión de datos.

Ahora estás listo para tomar el cuestionario del módulo e intentar el desafío final: La Prueba del Módulo 2, que te ayudará a evaluar lo que has aprendido hasta ahora.