



Tecnicatura Universitaria
en Programación

Universidad Tecnológica Nacional
Facultad Regional Avellaneda

Seminario de nivelación
Programación Inicial

Índice de temas:

1. **Introducción a la programación:**

- Introducción
- Lenguajes naturales vs lenguajes de programación
- ¿Qué compone un lenguaje?
- Lenguaje máquina vs lenguaje de alto nivel
- Compilación vs Interpretación
- Algoritmo

2. **Introducción a Python:**

- ¿Qué es Python?
- ¿Quién creó Python?
- Un proyecto de programación por pasatiempo
- ¿Qué hace que Python sea tan especial?
- Cómo descargar, instalar y configurar Python
- Comenzando tu trabajo con Python
- Tu primer programa

3. **Primer programa – Literales – Tipo de datos:**

- Primer programa
- Literales – los datos en sí mismos
- Enteros
- Flotantes
- Enteros vs Flotantes
- Booleanos
- Cadenas
- Comentarios en línea, en bloque.

4. **Operadores:**

- Operadores – herramientas de manipulación de datos.

5. **Variables – Constantes:**

- Variables – cajas con forma de datos
- Nombres de Variables
- Cómo crear una variable
- Cómo emplear una variable
- Cómo asignar un nuevo valor a una variable ya existente
- Constantes en Python.

6. **Interacción con el Usuario:**

- Interacción con el Usuario
- La función input() con un argumento
- El resultado de la función input()
- Conversión de tipos
- Más sobre input()
- Operadores cadena
- Conversiones de tipos una vez más

7. **Porcentajes – Incrementar – Decrementar:**

- Porcentaje
- Operadores de Incremento – Decremento.

8. **Toma de decisiones – comparadores:**

- Preguntas y Respuestas
- Comparación: operador de igualdad
- Operadores comparación
- Desigualdad: el operador no es igual a (!=)
- Operadores de comparación: mayor que
- Operadores de comparación: mayor o igual que
- Operadores de comparación: menor o igual que
- Haciendo uso de las respuestas.

1. Introducción

1.1 ¿Cómo funciona un programa de computadora?

Un programa hace que una computadora sea usable. Sin un programa, una computadora, incluso la más poderosa, no es más que un objeto. Del mismo modo, sin un reproductor, un piano no es más que una caja de madera.

Las computadoras pueden realizar tareas muy complejas, pero esta habilidad no es innata. La naturaleza de una computadora es bastante diferente.

Solo puede ejecutar operaciones extremadamente simples. Por ejemplo, una computadora no puede comprender el valor de una función matemática complicada por sí misma, aunque esto no está fuera del alcance de la posibilidad en un futuro cercano.

Las computadoras contemporáneas solo pueden evaluar los resultados de operaciones muy fundamentales. , como sumar o dividir, pero pueden hacerlo muy rápido y pueden repetir estas acciones prácticamente cualquier cantidad de veces.



Imagina que quieres saber la velocidad media que has alcanzado durante un viaje largo. Conoces la distancia, conoces el tiempo, necesitas la velocidad.

Naturalmente, la computadora podrá calcular esto, pero la computadora no es consciente de cosas como la distancia, la velocidad o el tiempo. Por lo tanto, es necesario instruir a la computadora para:

- aceptar un número que represente la distancia;
- aceptar un número que represente el tiempo de viaje;
- divide el valor anterior por el segundo y almacenar el resultado en la memoria;
- mostrar el resultado (que representa la velocidad promedio) en un formato legible.

Estas cuatro simples acciones forman un programa. Por supuesto, estos ejemplos no están formalizados y están muy lejos de lo que la computadora puede entender, pero son lo suficientemente buenos para ser traducidos a un idioma que la computadora pueda aceptar.

El lenguaje es la palabra clave.

1.2 Lenguajes naturales vs lenguajes de programación

Un lenguaje es un medio (y una herramienta) para expresar y registrar pensamientos. Hay muchos lenguajes a nuestro alrededor. Algunos de ellos no requieren ni hablar ni escribir, como el lenguaje corporal; es posible expresar tus sentimientos más profundos muy precisamente sin decir una palabra.

Otro lenguaje que utilizas cada día es tu lengua materna, que utilizas para manifestar tu voluntad y reflexionar sobre la realidad. Las computadoras también tienen su propio lenguaje, llamado lenguaje **máquina**, que es muy rudimentario.

Una computadora, incluso la más sofisticada técnicamente, está desprovista de cualquier rastro de inteligencia. Se podría decir que es como un perro bien adiestrado: responde sólo a un conjunto predeterminado de comandos conocidos.

Los comandos que reconoce son muy simples. Podemos imaginar que la computadora responde a órdenes como "toma ese número, divide por otro y guarda el resultado".

Un conjunto completo de comandos conocidos se llama **lista de instrucciones**, a veces abreviada IL (por sus siglas en inglés). Los diferentes tipos de computadoras pueden variar según el tamaño de sus IL y las instrucciones pueden ser completamente diferentes en diferentes modelos.

Nota: los lenguajes máquina son desarrollados por humanos.

Ninguna computadora es actualmente capaz de crear un nuevo idioma o lenguaje. Sin embargo, eso puede cambiar pronto. Por otro lado, las personas también usan varios idiomas muy diferentes, pero estos idiomas se crearon ellos mismos. Además, todavía están evolucionando.

Cada día se crean nuevas palabras y desaparecen las viejas. Estos lenguajes se llaman **lenguajes naturales**.

1.3 ¿Qué compone a un lenguaje?

Podemos decir que cada lenguaje (máquina o natural, no importa) consta de los siguientes elementos:

un conjunto de símbolos utilizados para formar palabras de un determinado lenguaje (por ejemplo, el alfabeto latino para el inglés, el alfabeto cirílico para el ruso, el kanji para el japonés, y así sucesivamente)

Un alfabeto: un conjunto de símbolos utilizados para formar palabras de un determinado lenguaje (por ejemplo, el alfabeto latino para el inglés, el alfabeto cirílico para el ruso, el kanji para el japonés, y así sucesivamente)

Un léxico: (también conocido como diccionario) un conjunto de palabras que el lenguaje ofrece a sus usuarios (por ejemplo, la palabra "computadora" proviene del diccionario en inglés, mientras que "cmoptrue" no; la palabra "chat" está presente en los diccionarios de inglés y francés, pero sus significados son diferentes)

Una sintaxis: un conjunto de reglas (formales o informales, escritas o interpretadas intuitivamente) utilizadas para precisar si una determinada cadena de palabras forma una oración válida (por ejemplo, "Soy una serpiente" es una frase sintácticamente correcta, mientras que "Yo serpiente soy una" no lo es)

Una semántica: un conjunto de reglas que determinan si una frase tiene sentido (por ejemplo, "Me comí una dona" tiene sentido, pero "Una dona me comió" no lo tiene)

1.4 Lenguaje máquina vs lenguaje de alto nivel

El **IL** es, de hecho, el **alfabeto de un lenguaje máquina**. Este es el conjunto de símbolos más simple y primario que podemos usar para dar comandos a una computadora. Es la lengua materna de la computadora.

Desafortunadamente, esta lengua materna está muy lejos de la lengua materna humana. Ambos (computadoras y humanos) necesitamos algo más, un lenguaje común para computadoras y humanos, o un puente entre los dos mundos diferentes.

Necesitamos un lenguaje en el que los humanos puedan escribir sus programas y un lenguaje que las computadoras pueden usar para ejecutar los programas, uno que es mucho más complejo que el lenguaje de máquina y, sin embargo, mucho más simple que el lenguaje natural.

Estos lenguajes a menudo se denominan lenguajes de programación de alto nivel. Son al menos algo similares a los naturales en que usan símbolos, palabras y convenciones legibles para los humanos. Estos lenguajes permiten a los humanos expresar comandos a las computadoras que son mucho más complejos que los que ofrecen las IL.

Un programa escrito en un lenguaje de programación de alto nivel se denomina **código fuente** (en contraste con el código máquina ejecutado por las computadoras). Del mismo modo, el archivo que contiene el código fuente se denomina **archivo fuente**.

1.5 Compilación vs Interpretación

La programación informática es el acto de componer los elementos del lenguaje de programación seleccionado en el orden que provocará el efecto deseado. El efecto podría ser diferente en cada caso específico – depende de la imaginación, el conocimiento y la experiencia del programador.

Por supuesto, dicha composición tiene que ser correcta en muchos sentidos:

- **alfabéticamente** – un programa debe estar escrito en un alfabeto reconocible, como romano, cirílico, etc.
- **léxicamente** – cada lenguaje de programación tiene su diccionario y hay que dominarlo; afortunadamente, es mucho más simple y pequeño que el diccionario de cualquier idioma natural;
- **sintácticamente** – cada idioma tiene sus reglas y hay que obedecerlas;
- **semánticamente** – el programa tiene que tener sentido.

Desafortunadamente, un programador también puede cometer errores con cada uno de los cuatro sentidos anteriores. Cada uno de ellos puede hacer que el programa se vuelva completamente inútil.

Supongamos que has escrito con éxito un programa. ¿Cómo persuadimos a la computadora para que lo ejecute? Tienes que convertir tu programa en lenguaje de máquina. Afortunadamente, la traducción la puede hacer una computadora, lo que hace que todo el proceso sea rápido y eficiente.

Hay dos formas diferentes de **transformar un programa de un lenguaje de programación de alto nivel a un lenguaje de máquina**:

Compilación:

El programa fuente se traduce una sola vez al obtener un archivo que contiene el código máquina. Ahora se puede distribuir el archivo en todo el mundo; el programa que realiza esta traducción se llama compilador o traductor.

Interpretación:

Cualquier usuario del código fuente puede traducir el programa cada vez que se debe ejecutar. El programa que realiza este tipo de transformación se denomina intérprete, ya que interpreta el código cada vez que se pretende ejecutar.

1.6 Algoritmo

Es un conjunto de pasos lógicos y estructurados que nos permiten dar solución a un problema.

La importancia de un algoritmo radica en desarrollar un razonamiento lógico matemático a través de la comprensión y aplicación de metodologías para la resolución de problemáticas, éstas problemáticas bien pueden ser de la propia asignatura o de otras disciplinas como matemáticas, química y física que implican el seguimiento de algoritmos, apoyando así al razonamiento crítico deductivo e inductivo.

No podemos apartar nuestra vida cotidiana los algoritmos, ya que al realizar cualquier actividad diaria los algoritmos están presentes, aunque pasan desapercibidos, por ejemplo:

Al levantarnos cada día para hacer nuestras labores hacemos una serie de pasos una y otra vez; eso es aplicar un algoritmo.

Estructura de un Algoritmo:

Todo algoritmo consta de tres secciones principales:

- **Entrada:** Es la introducción de datos para ser transformados.
- **Proceso:** Es el conjunto de operaciones a realizar para dar solución al problema.
- **Salida:** Son los resultados obtenidos a través del proceso.

Metodología para la descomposición de un algoritmo:

- **Definición del problema:**
En esta etapa se deben establecer los resultados y objetivos que se desea para poder saber si los datos que se tienen son suficientes para lograr los fines propuestos.
- **Análisis:**
Una vez definido el problema se deberán organizar los datos de tal manera que sean susceptibles de usar en los cálculos siguientes.
- **Diseño:**
En esta etapa se proponen soluciones a los problemas a resolver, por lo que se realiza una toma de decisiones aplicando los conocimientos adquiridos y utilizando los datos existentes.
- **Verificación o prueba de escritorio:**
Se consideran resultados previstos para datos conocidos a fin de que al probar cada una de sus partes podamos ir comprobando que el algoritmo sirve o requiere modificarse.

¿QUÉ ES PROGRAMAR? (5:01 min):

<https://www.youtube.com/watch?v=9nkwitj8evs>

¿QUÉ ES UN ALGORITMO? (5:48 min):

<https://www.youtube.com/watch?v=dQ-j0Noadac>

2. Introducción a Python

2.1 ¿Qué es Python?

Python es un lenguaje de programación de alto nivel, interpretado, orientado a objetos y de uso generalizado con semántica dinámica, que se utiliza para la programación de propósito general.

Aunque puede que conozcas a la python como una gran serpiente, el nombre del lenguaje de programación Python proviene de una vieja serie de comedia de la BBC llamada Monty Python's Flying Circus.

En el apogeo de su éxito, el equipo de Monty Python estaba realizando sus escenas en vivo para audiencias en todo el mundo, incluso en el Hollywood Bowl.

Dado que Monty Python es considerado uno de los dos nutrientes fundamentales para un programador (el otro es la pizza), el creador de Python nombró el lenguaje en honor al programa de televisión.



2.2 ¿Quién creó Python?

Una de las características sorprendentes de Python es el hecho de que en realidad es el trabajo de una persona. Por lo general, los grandes lenguajes de programación son desarrollados y publicados por grandes compañías que emplean a muchos profesionales, y debido a las normas de derechos de autor, es muy difícil nombrar a cualquiera de las personas involucradas en el proyecto. Python es una excepción.

No existen muchos lenguajes de programación cuyos autores sean conocidos por su nombre. Python fue creado por Guido van Rossum, nacido en 1956 en Haarlem, Países Bajos. Por supuesto, Guido van Rossum no desarrolló y evolucionó todos los componentes de Python.

La velocidad con la que Python se ha extendido por todo el mundo es el resultado del trabajo continuo de miles de (muy a menudo anónimos) programadores, testers, usuarios (muchos de ellos no son especialistas en TI) y entusiastas, pero hay que decir que la primera idea (la semilla de la que brotó Python) llegó a una cabeza: la de Guido.



2.3 Un proyecto de programación por pasatiempo

Las circunstancias en las que se creó Python son un poco desconcertantes. Según Guido van Rossum:

En diciembre de 1989, estaba buscando un proyecto de programación de "pasatiempo" que me mantendría ocupado durante la semana de Navidad. Mi oficina (...) estaría cerrada, pero tenía una computadora en casa y no mucho más en mis manos. Decidí escribir un intérprete para el nuevo lenguaje de scripting en el que había estado pensando últimamente: un descendiente de ABC que atraería a los hackers de Unix/C. Elegí Python como el título de trabajo para el proyecto, estando en un estado de ánimo ligeramente irreverente (y un gran fanático de Monty Python's Flying Circus). Guido van Rossum.

Los objetivos de Python:

En 1999, Guido van Rossum definió sus objetivos para Python:

- Un lenguaje **fácil e intuitivo** tan poderoso como los de los principales competidores.
- De código abierto, para que cualquiera pueda contribuir a su desarrollo.
- El código que es tan comprensible como el inglés simple.
- Adecuado para tareas cotidianas, permitiendo tiempos de desarrollo cortos.

Unos 20 años después, está claro que todas estas intenciones se han cumplido. Algunas fuentes dicen que Python es el lenguaje de programación más popular del mundo, mientras que otros afirman que es el tercero o el quinto.

2.4 ¿Qué hace que Python sea tan especial?

¿Por qué los programadores, jóvenes y viejos, experimentados y novatos, quieren usarlo? ¿Cómo fue que las grandes empresas adoptaron Python e implementaron sus productos al usarlo?

Existen muchas razones. Ya hemos enumerado algunas de ellas, pero vamos a enumerarlas de una manera más práctica:

- Es **fácil de aprender** - el tiempo necesario para aprender Python es más corto que en muchos otros lenguajes; esto significa que es posible comenzar la programación real más rápido.
- Es **fácil de enseñar** - la carga de trabajo de enseñanza es menor que la que necesitan otros lenguajes; esto significa que el profesor puede poner más énfasis en las técnicas de programación generales (independientes del lenguaje), no gastando energía en trucos exóticos, extrañas excepciones y reglas incomprensibles.
- Es **fácil de utilizar para escribir software nuevo** - a menudo es posible escribir código más rápido cuando se emplea Python.
- Es **fácil de entender** - a menudo, también es más fácil entender el código de otra persona más rápido si está escrito en Python.
- Es **fácil de obtener**, instalar y desplegar - Python es gratuito, abierto y multiplataforma; no todos los lenguajes pueden presumir de eso.

2.5 Cómo descargar, instalar y configurar Python

Existen varias formas de obtener tu propia copia de Python 3, dependiendo del sistema operativo que utilices. Debido a que el navegador le dice al sitio web al que se ingresó, el sistema operativo que se utiliza, el único paso que se debe seguir es hacer clic en la versión de Python que se desea.

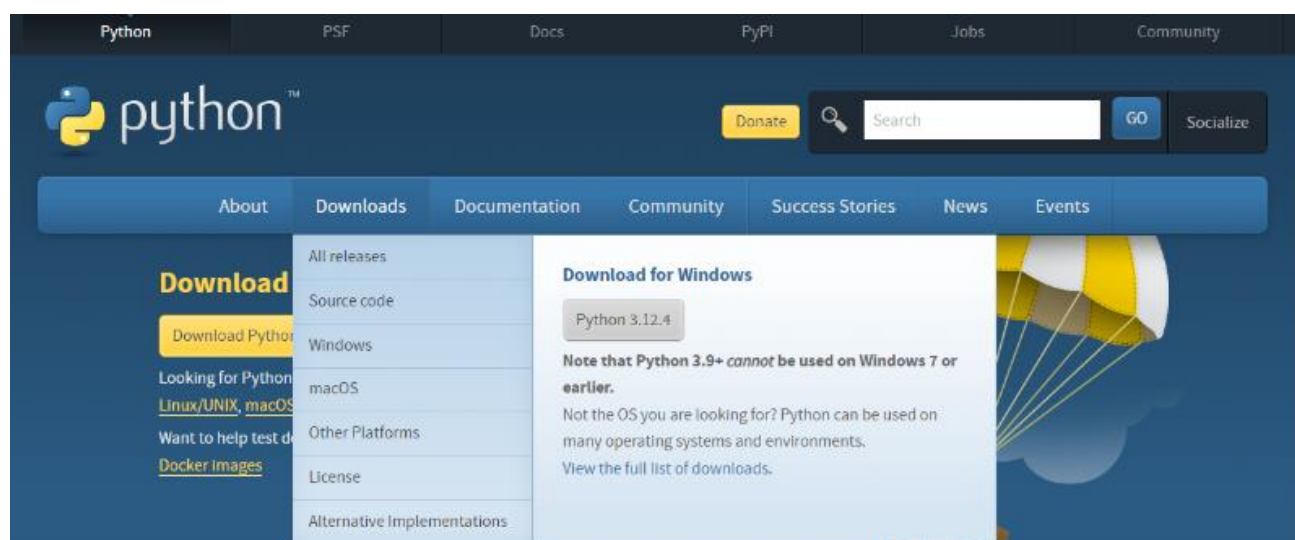
En este caso, selecciona Python 3. El sitio siempre te ofrece la última versión.

Si eres un usuario de Windows, utiliza el archivo .exe descargado y sigue todos los pasos.

Deja las configuraciones predeterminadas que el instalador sugiere por ahora, con una excepción: observa la casilla de verificación denominada Agregar Python 3.x a PATH y selecciónala.

Esto hará las cosas más fáciles.

Si eres un usuario de macOS, es posible que ya se haya preinstalado una versión de Python 2 en tu computadora, pero como estaremos trabajando con Python 3, aún deberás descargar e instalar el archivo .pkg correspondiente desde el sitio de Python.



2.6 Comenzando tu trabajo con Python

Ahora que tienes Python 3 instalado, es hora de verificar si funciona y de utilizarlo por primera vez.

Este será un procedimiento muy simple, pero debería ser suficiente para convencerte de que el entorno de Python es completo y funcional.

Existen muchas formas de utilizar Python, especialmente si vas a ser un desarrollador de Python.

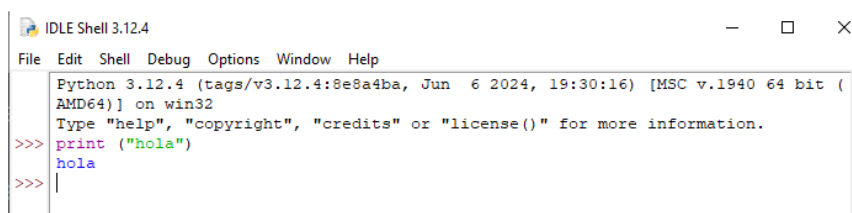
Para comenzar tu trabajo, necesitas las siguientes herramientas:

- Un **editor** que te ayudará a escribir el código (debe tener algunas características especiales, no disponibles en herramientas simples); este editor dedicado te dará más que el equipo estándar del sistema operativo, puede ser [Visual Studio Code](#).
- Una **consola** en la que puedas ejecutar tu código recién escrito y detenerlo por la fuerza cuando se sale de control.
- Una herramienta llamada depurador, capaz de ejecutar tu código paso a paso y te permite inspeccionarlo en cada momento de su ejecución.

Además de sus muchos componentes útiles, la instalación estándar de Python 3 contiene una aplicación muy simple pero extremadamente útil llamada IDLE.

IDLE es un acrónimo de: **I**ntegrated **D**evelopment and **L**earning **E**nvironment (Desarrollo Integrado y Entorno de Aprendizaje).

Navega por los menús de tu sistema operativo, encuentra IDLE en algún lugar debajo de Python 3.x y ejecútalo. Esto es lo que deberías ver:



2.7 Tu primer programa

Ahora es el momento de escribir y ejecutar tu primer programa en Python 3. Por ahora, será muy simple.

La ventana del editor actualmente no tiene título, pero es una buena práctica comenzar a trabajar

Ahora solo coloca una línea en tu ventana de editor y presiona enter.

```
print("Hola mundo UTN...")
```

Viste en el IDLE:

```
print("hola mundo UTN")
```

```
hola mundo UTN
```

Es importante recordar que durante la cursada trabajaremos con el Editor de Código Visual Studio Code, no olvidar ver el video de como instalarlo:

PYTHON CON VISUAL STUDIO CODE (5:30):

<https://www.youtube.com/watch?v=B2NnELeRKUE&t=1s>

INSTALACIÓN DE PYTHON (4:13 min):

<https://www.youtube.com/watch?v=6XbFEDPZ1P4&t=1s>

3. Primer Programa – Literales – Tipo de Datos

3.1 Primer programa

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.

La función print()

Observa la línea de código de abajo:

```
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 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.

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

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

Como pueden 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 - pueden modificarla, ingresar contenido nuevo, o borrar 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.

HOLA MUNDO EN PYTHON (4:27 min):

<https://www.youtube.com/watch?v=IQy2urna3IA&t=1s>

Instrucciones:

Ya has visto un programa de computadora que contiene una invocación de función. La invocación de una función es uno de los muchos tipos posibles de instrucciones de Python.

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 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).

3.2 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*.

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

Comencemos con un sencillo experimento - observa el fragmento de código y luego ejecútalo y compara los resultados.

```
primerPrograma.py U X
python > pythonPrueba > primerPrograma.py
1 print("2")
2 print(2)
```

Así se verá en la terminal el resultado:

```
Programa.py
2
2
```

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

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 máquina (una serie de bits). La función `print()` es capaz de mostrar ambos en una forma legible.

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

3.3 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 fraccionaria.

La distinción es muy importante, y la frontera entre estos dos tipos de números es muy estricta. Ambos tipos difieren significativamente en cómo son almacenados en una computadora y en el rango de valores que aceptan.

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

Por ahora, dejemos los números flotantes a un lado 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**.

3.4 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 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.

3.5 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.

3.6 Booleanos

Los valores booleanos en Python son un tipo de dato fundamental que representan una de dos posibles respuestas: **True** (verdadero) o **False** (falso). Estos valores se utilizan ampliamente en la programación para realizar comparaciones y tomar decisiones basadas en condiciones específicas. Aquí hay una explicación detallada de los valores booleanos en Python:

Valores Booleanos

En Python, los valores booleanos se representan mediante las palabras clave **True** y **False**. Estos son equivalentes a los valores numéricos 1 y 0, respectivamente, pero se utilizan específicamente para indicar estados de verdad.

Ejemplo de valores booleanos:

```
a = True
b = False

print(a) # Output: True
print(b) # Output: False
```

3.7 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."**

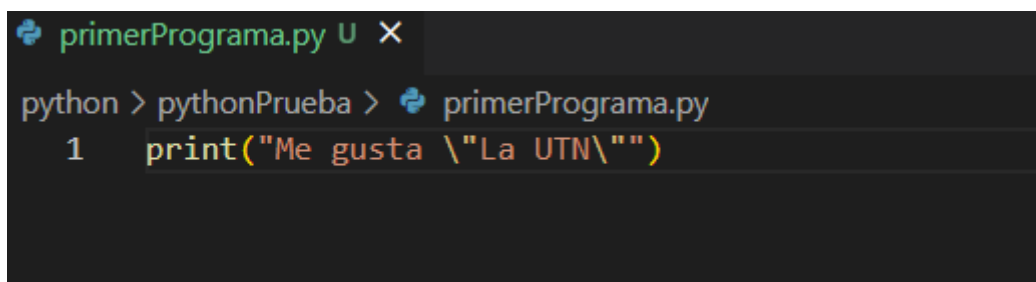
Sin embargo, hay una cuestión. Cómo se puede codificar una comilla dentro de una cadena que ya está delimitada por comillas.

Supongamos que se desea mostrar un muy sencillo mensaje:

```
Me gusta "La UTN"
```

¿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:



```
primerPrograma.py U X
python > pythonPrueba > primerPrograma.py
1 print('Me gusta \"La UTN\"')
```

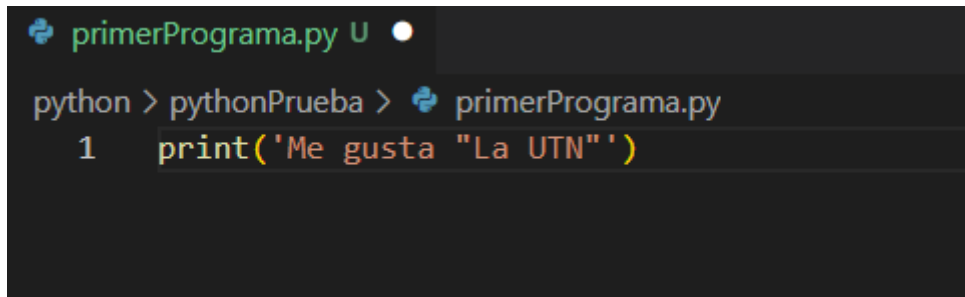
Nota: ¿Existen dos comillas con escape en la cadena - puedes observar ambas?

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:



```
python > pythonPrueba > primerPrograma.py
1  print('Me gusta "La UTN"')
```

3.8 Comentarios en línea, en bloque

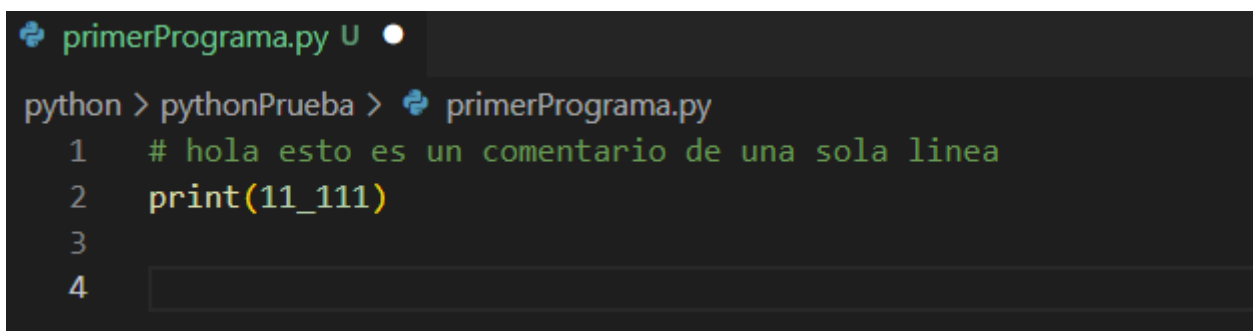
Quizá en algún momento será necesario poner algunas palabras en el código dirigidas no a Python, sino a las personas quienes estén leyendo el código con el fin de explicarles como es que funciona, o tal vez especificar el significado de las variables, también para documentar quien es el autor del programa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



```
python > pythonPrueba > primerPrograma.py
1  # hola esto es un comentario de una sola linea
2  print(11_111)
3
4
```


Si se desea colocar un comentario que abarca varias líneas, se debe colocar triples comillas simples:

```
python > pythonPrueba > primerPrograma.py
1  '''
2  Hola
3  eso es un comentario
4  multilinea
5  '''
6  print("hola multilinea")
7
8
```

COMENTARIOS EN PYTHON (2:12 min):

<https://www.youtube.com/watch?v=DzrGcOeqfAc>

4. Operadores

4.1 Operadores – herramientas de manipulación de datos

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)
```

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

Hemos 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**.

Operadores básicos:

Un operador 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 `+` (más) 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:

- `+` (suma)
- `-` (resta)
- `*` (multiplicación)
- `/` (división)

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.

Exponenciación:

Observa los ejemplos en la ventana del editor:

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

Multiplicación:

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

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

División:

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

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

Ejecuta el código y analiza los resultados.

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

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.)
```

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 – intenta predecir el resultado y después ejecútalo:

```
print(14 % 4)
```

```
print(10%3)
```

```
print(10%2)
```

Suma:

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

De nuevo, observa el siguiente fragmento de código:

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

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 ultima línea de código del siguiente fragmento:

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

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 recordarás de la escuela que las multiplicaciones preceden a las sumas.

Seguramente recordarás 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.

TIPOS DE DATOS EN PROGRAMACIÓN (4:06 min):

<https://www.youtube.com/watch?v=UUxSQ-rCeXs>

EXPRESIÓN VS INSTRUCCIÓN (2:54 min):

<https://www.youtube.com/watch?v=nVDq-GXQkNk>

OPERADORES ARITMÉTICOS EN PYTHON (5:20 min):

<https://www.youtube.com/watch?v=uIYqLzM5yVM&t=1s>

VARIABLES EN PROGRAMACIÓN (4:30 min):

<https://www.youtube.com/watch?v=IHtqbquNzHk>

5. Variables – Constantes

5.1 Variables – cajas con forma de datos

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 cómo 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" (o "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, tu debes decidir cuantas variables deseas utilizar en tu programa.

También las debes de nombrar.

5.2 Nombres de Variables

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

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

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:

- `MyVariable`
- `i`
- `l`
- `t34`
- `Exchange_Rate`
- `counter`
- `days_to_christmas`
- `TheNameIsTooLongAndHardlyReadable`
- `_`

Estos nombres de variables también son correctos:

- `Adiós_Señora`
- `sûr_la_mer`
- `Einbahnstraße`
- `переменная`.

Python te permite usar no solo letras latinas sino también caracteres específicos de idiomas que usan otros alfabetos.

Ahora veamos algunos nombres incorrectos:

- `10t` (no comienza con una letra)
- `!important` (no comienza con una letra)
- `exchange rate` (contiene un espacio).

El [PEP 8 -- Style Guide for Python Code](#) recomienda la siguiente convención de nomenclatura para variables y funciones en Python:

- Los nombres de las variables deben estar en minúsculas, con palabras separadas por guiones bajos para mejorar la legibilidad (por ejemplo, `var`, `my_variable`)
- Los nombres de las funciones siguen la misma convención que los nombres de las variables (por ejemplo, `fun`, `my_function`)
- También es posible usar letras mixtas (por ejemplo, `myVariable`), pero solo en contextos donde ese ya es el estilo predominante, para mantener la compatibilidad retroactiva con la convención adoptada.

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 clave 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.

5.3 Cómo crear una variable

¿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 mas de los cuales aun 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 fragmento en el editor:

```
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.

Como puedes ver, **print()** tiene otro lado: también puede manejar variables. ¿Sabes cuál será el resultado del fragmento? Ejecuta el código para verificar.

5.4 Cómo emplear una variable

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

```
var = 1  
account_balance = 1000.0  
client_name = 'John Doe'  
print(var, account_balance, client_name)  
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 ocasionará 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 = "3.8.5"
print("Python version: " + var)
```

5.5 Cómo asignar un nuevo valor 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:

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

La sentencia 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`, sumale 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)
```

5.6 Constantes en Python

Terminamos esta introducción a Python señalando que, a diferencia de otros lenguajes, en Python no existen las constantes.

Entendemos como constante una variable que una vez asignado un valor, este no se puede modificar. Es decir, que a la variable no se le puede asignar ningún otro valor una vez asignado el primero.

Se puede simular este comportamiento, siempre desde el punto de vista del programador y atendiendo a convenciones propias, pero no podemos cambiar la naturaleza mutable de las variables.

Una convención para trabajar con constantes en python es definir el nombre con mayusculas como el siguiente ejemplo:

```
TITULO_CURSO = "curso introduccion"
```

VARIABLES EN PYTHON (5:01 min):

<https://www.youtube.com/watch?v=IJckwi9La4>

6. Interacción con el usuario

6.1 La función `input()`

Ahora se introducirá una nueva función, la cual pareciese ser un reflejo de la función `print()`, `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 siguiente ejemplo:

```
print("Introduce algun texto en el cursor titilando...")
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 mas 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.

Ahora crearemos el mismo ejemplo pero guardaremos el valor del `input()` en una variable para así poder reutilizarla a futuro:

```
print("Introduce algun texto en el cursor parpadeando")
valorInputGuardado = input()
print(valorInputGuardado)
```

En el código anterior el valor ingresado por la función `input`, se almacena en la variable creada "`valorInputGuardado`", lo cual en la línea siguiente se imprime ese valor con la función `print()`.

6.2 La función `input()` con un argumento

La función `input()` puede hacer otra cosa: puede avisar al usuario sin ninguna ayuda de `print()`.

Hemos modificado un poco nuestro ejemplo, mira el código:

```
algo= input("Escribir lo que sea...")
print("Hmm...", algo, "...¿en serio?")
```

Nota:

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

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

6.3 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 multiplicarlos, para dividirlos entre algo o por algo.

```
numero = input("Ingresa un número:")
multiplicacion = numero * 2
print(numero, "multiplicado por 2 es:", numero)
```

¿Qué es lo que ocurre? Python debió haberte dado una salida un poco confusa si ingresamos un número, por ejemplo, el 2. Lo que está sucediendo es que python está replicando una cadena dos veces lo cual está realizando una operación un poco extraña a la que pedimos.

6.4 Conversión de tipos

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

Sus nombres indican cual es su función:

- La función int() toma un argumento (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 mas 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.

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

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.

```
numero = int(input("Ingresa un número: "))
algo = numero * 2
print(algo, "multiplicado por 2 es", numero)
```

6.5 Más sobre input()

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:

```
leg_a = float(input("Ingresa la longitud del primer cateto: "))
leg_b = float(input("Ingresa la longitud del segundo cateto: "))
hypo = (leg_a**2 + leg_b**2) **.5
print("La longitud de la hipotenusa es:", hypo)
```

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

Toma en cuenta que en el programa que puede ver en el editor, la variable hypo se usa con un solo propósito - guardar el valor calculado entre la ejecución de la línea de código contigua.

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:

```
leg_a = float(input("Ingresa la longitud del primer cateto: "))
leg_b = float(input("Ingresa la longitud del segundo cateto: "))
print("La longitud de la hipotenusa es:", (leg_a**2 + leg_b**2) **.5)
```

6.6 Operadores cadena

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

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

Los hemos visto en acción cuando sus argumentos son (flotantes o enteros, no hay diferencia).

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

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

```
string + string
```

Simplemente concatena (junta) dos cadenas en una. Por supuesto, puede ser utilizado más de una vez en una misma expresión, y en tal contexto se comporta con enlazado del lado izquierdo.

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 sencillo programa muestra el signo + en su segundo uso:

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

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:

```
string * number
number * string
```

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

Por ejemplo:

- "James" * 3 produce "JamesJamesJames"
- 3 * "an" produce "ananan"
- 5 * "2" (o "2" * 5) produce "22222" (no 10!)

6.7 Conversiones de tipos una vez más

A estas alturas ya sabes cómo 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 numero a una cadena, lo cual es más fácil y seguro - esta operación es posible hacerla siempre.

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

```
str(number)
```

CONCATENAR DATOS EN PYTHON (6:38 min):

<https://www.youtube.com/watch?v=-5QQp0W6kv4>

ENTRADA DE DATOS (INPUT) EN PYTHON (7:02 min):

<https://www.youtube.com/watch?v=Wd3J3Jh0Nko&t=1s>

7. Porcentaje – Incrementar – Decrementar

7.1 Porcentaje

Los porcentajes son una forma de expresar un número como una fracción de 100. Por ejemplo, el 50% es igual a 50/100 o 0.5. En Python, podemos trabajar con porcentajes usando operaciones matemáticas básicas.

Cálculo de porcentajes

1. Convertir un número a un porcentaje:

Para convertir un número a un porcentaje, multiplica el número por 100.

```
# Convertir un número a porcentaje
```

```
numero = 0.75
```

```
porcentaje = numero * 100
```

2. Encontrar el porcentaje de un número:

Para encontrar el porcentaje de un número, multiplica el número por el porcentaje deseado y divide por 100.

```
numero = 200
```

```
porcentaje = 15
```

```
resultado = (numero * porcentaje) / 100
```

3. Incrementar un número por un porcentaje:

Para incrementar un número por un porcentaje, primero encuentra el porcentaje del número y luego añádelo al número original.

```
numero = 100
```

```
porcentaje = 20
```

```
incremento = (numero * porcentaje) / 100
```

```
nuevo_numero = numero + incremento
```

7.2 Operadores de Incremento-decremento

Es importante recordar que en Python no existe el operador ++ ni --, para lograr el mismo resultado se utilizan los operadores compuestos:

```
x = x+1
```

```
x += 1
```

En ambos casos se incrementa en 1 el valor de la x, así mismo también se puede realizar con el operador -, te dejamos una lista de los operadores compuestos:

+=	x += 2	x = x + 2
-=	x -= 2	x = x - 2
*=	x *= 2	x = x * 2
/=	x /= 2	x = x / 2
%=	x %= 2	x = x % 2
//=	x //= 2	x = x // 2
**=	x **= 2	x = x ** 2

8. Toma de decisiones - Comparadores

8.1 Preguntas y Respuestas

Un programador escribe un programa y el programa hace preguntas.

Una computadora ejecuta el programa y proporciona las respuestas. El programa debe ser capaz de reaccionar de acuerdo con las respuestas recibidas.

Afortunadamente, las computadoras solo conocen dos tipos de respuestas (**valores Booleanos**):

- Si, es cierto.
- No, esto es falso.

Nunca obtendrás una respuesta como *Déjame pensar...*, *no lo sé*, o *probablemente sí, pero no lo sé con seguridad*.

Para hacer preguntas, Python utiliza un conjunto de operadores muy especiales. Revisemos uno tras otro, ilustrando sus efectos en algunos ejemplos simples.

8.2 Comparación: operador de igualdad

Pregunta: ¿son dos valores iguales?

Para hacer esta pregunta, se utiliza el `==` (igual igual) operador.

No olvides esta importante distinción:

- `=` es un operador de asignación, por ejemplo, `a = b` asigna a la variable `a` el valor de `b`;
- `==` es una pregunta *¿Son estos valores iguales?* así que `a == b` compara `a` y `b`.

Es un operador binario con enlazado del lado izquierdo. Necesita dos argumentos y verifica si son iguales.

Ahora vamos a hacer algunas preguntas. Intenta adivinar las respuestas.

Pregunta #1:

¿Cuál es el resultado de la siguiente comparación?

```
print(2 == 2)
```

Pregunta #2:

¿Cuál es el resultado de la siguiente comparación?

```
print(2 == 3).
```

Pregunta #3:

¿Cuál es el resultado de la siguiente comparación?

```
print(1 == 2)
```

8.3 Operadores comparación

Igualdad: El operador *igual a* (`==`)

El operador `==` (igual a) compara los valores de dos operandos. Si son iguales, el resultado de la comparación es `True`. Si no son iguales, el resultado de la comparación es `False`.

Observa la comparación de igualdad a continuación - ¿Cuál es el resultado de esta operación?

```
print(var == 0)
```

Toma en cuenta que no podemos encontrar la respuesta si no sabemos qué valor está almacenado actualmente en la variable `var`.

Si la variable se ha cambiado muchas veces durante la ejecución del programa, o si se ingresa su valor inicial desde la consola, Python solo puede responder a esta pregunta en el tiempo de ejecución del programa.

Ahora imagina a un programador que sufre de insomnio, y tiene que contar las ovejas negras y blancas por separado siempre y cuando haya exactamente el doble de ovejas negras que de las blancas.

La pregunta será la siguiente:

```
print(ovejas_negras == (2 * ovejas_blancas))
```

Entonces, vamos a practicar la comprensión del operador == - ¿puedes adivinar la salida del código a continuación?

```
var = 0 # Asignando 0 a var
print(var == 0)

var = 1 # Asignando 1 a var
print(var == 0)
```

8.4 Desigualdad: el operador no es igual a (!=)

El operador != (no es igual a) también compara los valores de dos operandos. Aquí está la diferencia: si son iguales, el resultado de la comparación es **False**. Si no son iguales, el resultado de la comparación es **True**. Ahora echa un vistazo a la comparación de desigualdad a continuación - ¿puedes adivinar el resultado de esta operación?

```
var = 0 # Asignando 0 a var
print(var != 0)

var = 1 # Asignando 1 a var
print(var != 0)
```

8.5 Operadores de comparación: mayor que

También se puede hacer una pregunta de comparación usando el operador > (mayor que).

Si deseas saber si hay más ovejas negras que blancas, puedes escribirlo de la siguiente manera:

```
ovejas_negras = 10
ovejas_blancas = 15
print(ovejas_negras > ovejas_blancas)
```

8.6 Operadores de comparación: mayor o igual que

El operador *mayor que* tiene otra variante especial, una variante no estricta, pero se denota de manera diferente que la notación aritmética clásica: >= (mayor o igual que).

Hay dos signos subsecuentes, no uno.

Ambos operadores (estrictos y no estrictos), así como los otros dos que se analizan en la siguiente sección, son operadores binarios con enlazado del lado izquierdo, y su prioridad es mayor que la mostrada por == y !=.

Si queremos saber si tenemos mayor cantidad de ovejas incluidos el numero actual, nos hacemos la siguiente pregunta, observen ambos códigos :

```
ovejas_negras = 10
ovejas_blancas = 10
print(ovejas_negras > ovejas_blancas)

ovejas_negras = 10
ovejas_blancas = 10
print(ovejas_negras >= ovejas_blancas)
```

8.7 Operadores de comparación: menor o igual que

Como probablemente ya hayas adivinado, los operadores utilizados en este caso son: El operador < (menor que) y su hermano no estricto: <= (menor o igual que).

Observa este ejemplo simple:

```
ovejas_negras = 10
ovejas_blancas = 10
print(ovejas_negras > ovejas_blancas)

ovejas_negras = 10
ovejas_blancas = 10
print(ovejas_negras <= ovejas_blancas)
```

8.8 Haciendo uso de las respuestas

¿Qué puedes hacer con la respuesta (es decir, el resultado de una operación de comparación) que se obtiene de la computadora?

Existen al menos dos posibilidades: primero, puedes memorizarlo (almacenarlo en una variable) y utilizarlo más tarde. ¿Cómo haces eso? Bueno, utilizarías una variable arbitraria como esta:

```
valorVerdad = 10 > 5
valorMentira = 10 > 15

print(valorVerdad == valorMentira)
```

OPERADORES RELACIONALES EN PYTHON (4:23 min):



<https://www.youtube.com/watch?v=HIWyamKYl4M>

9. Extras:

- **Guía de ejercicios 1 – Introducción a Python:**
https://www.utnfravirtual.org.ar/pluginfile.php/552102/mod_resource/content/3/Actividad%20Pr%C3%A1ctica%20-%20Python%20Unidad%201%20%281%29.pdf
-  **Cuestionario 1:**
<https://www.utnfravirtual.org.ar/mod/quiz/view.php?id=207776>
- **Ejercicios Prácticos – Python Unidad 1-A:**
https://www.utnfravirtual.org.ar/pluginfile.php/552102/mod_resource/content/5/Ejercicios%20Pr%C3%A1cticos%20-%20Python%20Unidad%201-A%20%281%29.pdf
- **Ejercicios Prácticos – Python Unidad 1-B:**
https://www.utnfravirtual.org.ar/pluginfile.php/555376/mod_resource/content/6/Ejercicios%20Pr%C3%A1cticos%20-%20Python%20Unidad%201-B.pdf
- **Cuestionario Repaso General Unidad 1 – Python:**
<https://www.utnfravirtual.org.ar/mod/quiz/view.php?id=207227>

10. Desafíos:

Primer desafío

 Bienvenido a los desafíos 

Hola, te doy la bienvenida a los desafíos, un lugar donde podrás practicar tus conocimientos y compartirlos a la comunidad del curso de Nivelación con Python.

En este ejercicio, tu desafío es utilizar la función `print` para imprimir tres mensajes y publicarlos en el foro de tu división en la sección **Foro de comunicación -> Consultas**. Para ello, debes utilizar la función `print` tres veces en el siguiente orden con los siguientes mensajes:

- "Hola soy de la división..... y soy del barrio de....."
- "Tengo años"
- "Mis hobbies son....."

Recuerda prestar atención a los espacios y mayúsculas en los mensajes, ya que son importantes para que tu respuesta sea correcta. Un ejemplo de cómo utilizar la función `print` se encuentra a continuación:

```
print('Hola mundo Académico!!')
```

Segundo Desafío:

En este desafío, vas a practicar lo que has aprendido acerca de variables en Python. Una variable es un lugar en la memoria donde se puede almacenar un valor, por ejemplo, un número o una cadena de texto. En Python, las variables se declaran utilizando el nombre de la variable, seguido de un signo igual y el valor que se desea asignar a la variable. Por ejemplo:

```
texto = 'Mi primer texto'
print(texto)
```

Recuerda que las variables pueden ser modificadas a lo largo de un programa, tu reto es modificar el valor de la variable text para que sea un número y luego imprimirlo utilizando la función print.

Modifica el valor de la variable texto por un número y luego imprímelo 📌

Luego modifica el valor de la variable texto por una cadena de texto y vuélvelo a imprimir 📌

Tercer Desafío:

🔔 Bienvenido a los desafíos

En este desafío, vas a practicar lo que has aprendido acerca de contadores en Python.

Cree un programa que al ingresar un número entero, vaya incrementando de a una unidad y se muestre por consola:

Explicar

```
EJ:
el número ingresado fue 10
11
12
13
14
15
16....
```

Recuerda que Python no tiene el operador de incremento ++

Comparte tu código en el foro de comunicacion 📌

11. TP

TP: ES_Facturaciones

Enunciado:

Para el departamento de facturación:

- A. Ingresar tres precios de productos y mostrar la suma de los mismos.
- B. Ingresar tres precios de productos y mostrar el promedio de los mismos.
- C. ingresar tres precios de productos sumarlos y mostrar el precio final (más IVA 21%).

Una vez finalizado el TP, Comparte el código en los foros 