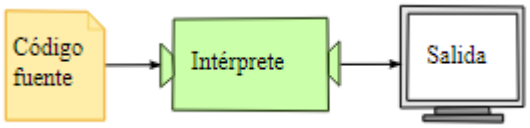
Principales Características del Lenguaje Python

Python es un lenguaje de programación creado por Guido van Rossum a finales de los ochenta, y que gracias a sus características ha llegado a ser un lenguaje muy conocido en la actualidad. A continuación se listan las principales características que este lenguaje posee:

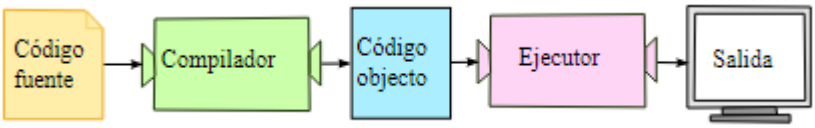
- **Simple:**
  Python es un lenguaje muy simple, por lo que es muy fácil iniciarse en este lenguaje. El pseudo-código natural de Python es una de sus grandes fortalezas.
  Propósito General:
  Usando el lenguaje Python se puede crear todo tipo de programas; programas de propósito general y también se pueden desarrollar páginas Web.
- **Open Source:**
  Debito a la naturaleza de Python de ser Open Suorce; ha sido modificado para que pueda funcionar en diversas plataformas (Linux, Windows, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE y PocketPC).Al ser Open Source es gratuito.
- **Lenguaje Orientado a Objetos:**
  Al ser un Lenguaje Orientado a Objetos es construido sobre objetos que combinan datos y funcionalidades.
- **Lenguaje de Alto Nivel:**
  Al programar en Python no nos debemos preocupar por detalles de bajo nivel, (como manejar la memoria empleada por el programa).
- **Incrustable:**
  Se puede insertar lenguaje Python dentro un programa C/C++ y de esta manera ofrecer las facilidades del scripting.
- **Extensas Librerías:**
  Python contiene una gran cantidad de librerías, tipos de datos y funciones incorporadas en el propio lenguaje, que ayudan a realizar muchas tareas comunes sin necesidad de tener que programarlas desde cero.
  Las librerías pueden ayudar a hacer varias cosas como expresiones regulares, generación de documentos, evaluación de unidades, pruebas, procesos, bases de datos, navegadores web, CGI, ftp, correo electrónico, XML, XML-RPC, HTML, archivos WAV, criptografía, GUI, y también otras funciones dependientes del Sistema.
- **Sintaxis clara:**
  Python tiene una sintaxis muy visual, gracias a que maneja una sintaxis identada (con márgenes), que es de caracter obligatorio. Para separar los bloques de código en Python se debe tabular hacia dentro. Esto ayuda a que todos los programadores adopten las mismas notaciones y que los programas hechos en Python tengan un aspecto muy similar.

Python es un lenguaje de programación **interpretado** cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma, lo que significa que cualquier sistema es compatible con el lenguaje siempre y cuando exista un intérprete programado para él.

Hay dos tipos de programas que traducen lenguajes de alto nivel a lenguajes de bajo nivel: **intérpretes** y **compiladores**. Un intérprete lee un programa de alto nivel y lo ejecuta, lo que significa que lleva a cabo lo que indica el programa. Es decir, traduce el programa poco a poco, leyendo y ejecutando cada comando.



Un compilador lee el programa y lo traduce completo antes de su ejecución. En este caso, al programa de alto nivel se le llama **código fuente**, y el programa traducido es llamado **código objeto** o **programa ejecutable**. Una vez que un programa ha sido compilado, puede ser ejecutado repetidamente sin necesidad de más traducción.



Muchos de los lenguajes modernos usan ambos tipos de programas de traducción. Estos lenguajes se traducen primero a un lenguaje de bajo nivel, llamado **código de bytes**, y después son interpretados por un programa denominado **máquina virtual**. Aunque Python usa ambos tipos de programas de traducción, usualmente se le considera un lenguaje interpretado debido a la manera en que los programadores interactúan con él.

Identificadores, Palabras Reservadas, Comentarios, Sentencias, Tipos básicos de datos, Constantes, Variables, Expresiones.

La sintaxis de un lenguaje define los elementos de dicho lenguaje y cómo se combinan para formar un programa. Los elementos típicos de cualquier lenguaje son los siguientes:

- Identificadores
- Tipos de datos
- Palabras reservadas
- Comentarios
- Sentencias
- Constantes
- Variables
- Expresiones
- Operadores

A lo largo de las páginas que siguen examinaremos en detalle cada uno de estos elementos.

Identificadores

Un identificador es un nombre que identifica a una variable, a un método o función miembro, a una clase. Todos los lenguajes tienen ciertas reglas para componer los identificadores:

- Todos los identificadores han de comenzar con una letra, o el carácter subrayado (_).
- Luego sigue con una secuencia de letras, números y guiones bajos. Los espacios no están permitidos dentro de los identificadores.
- Distingue entre letras mayúsculas y minúsculas
- No se pueden utilizar las palabras reservadas como identificadores

Además de estas restricciones, hay ciertas convenciones que hacen que el programa sea más legible, pero que no afectan a la ejecución del programa. La primera y fundamental es la de encontrar un nombre que sea significativo, de modo que el programa sea lo más legible posible. El tiempo que se pretende ahorrar eligiendo nombres cortos y poco significativos se pierde con creces cuando se revisa el programa después de cierto tiempo.

Tipo de identificador	Convención	Ejemplo
nombre de una clase	Comienza por letra mayúscula	String, Rectangulo, CinematicaApplet
nombre de función	comienza con letra minúscula	calcularArea, getValue, setColor
nombre de variable	comienza por letra minúscula	area, color, appletSize
nombre de constante	En letras mayúsculas	PI, MAX_ANCHO

Palabras Reservadas

Python reserva palabras para describir la estructura del programa, y no permite que se usen como identificadores. Cuando en un programa nos encontramos con que un nombre no es admitido pese a que su formato es válido, seguramente se trata de una de las palabras de esta lista, a la que llamaremos de *palabras reservadas*. Esta es la lista completa de las palabras reservadas de Python:

```
and       del       from     not      while
as        elif      global   or       with
assert    else      if       pass     yield
break     except    import   print
class     exec      in       raise
continue  finally   is       return
def       for       lambda   try
```

Comentarios

Un comentario es un texto adicional que se añade al código para explicar su funcionalidad, bien a otras personas que lean el programa, o al propio autor como recordatorio. Los comentarios son una parte importante de la documentación de un programa. Los comentarios son ignorados por el intérprete, debemos añadirlos al código para que pueda entenderse mejor.

La programación orientada a objetos facilita mucho la lectura del código, por lo que no se precisa hacer tanto uso de los comentarios como en los lenguajes estructurados. En Python existen dos tipos de comentarios

- Comentarios en una sola línea
- Comentarios de varias líneas

Como podemos observar un comentario en varias líneas es un bloque de texto situado entre el símbolo de comienzo del bloque **" " "** (tres dobles comillas consecutivas), y otro de terminación del mismo **" " "**. Teniendo en cuenta este hecho, los programadores diseñan comentarios como el siguiente:

```
 """--------------------------------|
|   (C) Roberto Garcia              |
|   Fecha: Marzo 2018               |
|   Programa: Manejo de archivos    |
|-------------------------------- """
```

Habitualmente, usaremos comentarios en una sola línea **para lo cual el comentario debe comenzar con #**, ejemplo:

```
# Esto es un comentario de una sola línea
mi_variable = 15
```

```
mi_variable = 15 # Este comentario es de una línea también
```

Comentarios en la misma línea del código deben separarse con dos espacios en blanco. Luego del símbolo # debe ir un solo espacio en blanco.

Sentencias

Una sentencia es una orden que se le da al programa para realizar una tarea específica, esta puede ser: mostrar un mensaje en la pantalla, inicializar una variable, llamar a una función, etc.

Tipos básicos de datos.

La computadora procesa datos y los convierte en información significativa.

Un Dato: es la expresión general que describe los objetos con los cuales trabaja el computador.

Información: son los datos procesados, los cuales contienen datos significativos.

Los datos pueden ser: *Numéricos* y *No-numéricos*.

- Numéricos se representan en dos formas como números **Enteros**, los cuales corresponden a números que no tienen componente decimal, que pueden ser positivos y negativos, ejemplo: 1, 45, -5, 12345, 1000000, -295, etc. Los números **Reales** siempre presentan componentes decimales pueden ser también positivos y negativos, ejemplo: 1.0, 23.8, -12345.456, 0.125, etc.
- No-numéricos pueden representarse como cadenas de caracteres y lógicos. Una cadena de caracteres es un secuencia de letras números y símbolos especiales que comienzan y terminan con una doble comilla (apostrofe) ejemplo: "La casa es roja", "Roberto Martínez", "Urbanización El Trébol calle # 25 casa # 35", y un dato lógico es aquel que solo puede tomar uno de dos valores "verdadero" o "falso" (True o False).

La computadora tiene la capacidad de manipular estos datos con las conocidas cuatro operaciones básicas: suma, resta, multiplicación y división para los datos numéricos, para los no-numéricos las operaciones no son tan conocidas pudiendo citar la concatenación que es la unión de dos cadenas.

Constantes: valores que no cambian durante la ejecución de un programa. Estas pueden ser:

- Enteras: representadas por datos numéricos enteros.
- Reales: representadas pos datos numéricos reales.
- De cadena: representado por un dato no-numérico de cadena.
- Lógicas: representado por un dato no-numérico lógico.

Variables: valores que cambian durante la ejecución de un programa, desde el punto de vista del computador se puede decir que son áreas de memoria que se reservan para almacenar datos de un determinado tipo de cuyo contenido puede variar durante la ejecución de un programa, identificadas con un nombre o identificador valido.

Tipos de variables:

- **Enteras:** cuyo contenido es un constante numérica entera. En Python estos están en el rango de valores está entre -2147483648 y +2147483647, denominados enteros cortos, los enteros largos están en el rango de valores entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807. Es de hacer notar que el rango de valores se refiere al grupo de valores que podemos almacenar en una variable de un determinado tipo.
- **Reales:** cuyo contenido es una constante numérica real. En Python estos están en el rango de valores comprendido entre $\pm 2.2250738585072020 \times 10^{-308}$ a $\pm 1.7973931348623157 \times 10^{308}$.
- **De Cadena:** cuyo contenido es una constante de cadena.
- **Lógicas:** cuyo contenido son constantes lógicas (True, False).

En Python no se declaran las variables como en otros lenguajes, aquí solamente con colocar el nombre de las variables e igualarla a un valor la variable asumirá el tipo en función del valor asignado de acuerdo a la siguiente sintaxis:

Nombre_Variable = Valor_de_la_variable

Esto es lo que se denomina el **tipado dinámico** el cual consiste en que no es necesario declarar el tipo de dato que almacenara una variable sino que su tipo se determina en tiempo de ejecución según el valor que se le asigne, por ejemplo:

```
Edad = 12 # Esta sería una variable entera corta
Numero = 123L # Esta sería una variable entera larga
```

Observe que en el último ejemplo hemos agregado al final del número la letra L para indicar el tipo entero largo

```
Impuesto = 123.45 # Esta sería una variable real
Nombre = "Andrés Gómez" # Esta sería una variable de cadena
Bandera = True # Esta sería una variable lógica
```

En Python es posibles hacer la asignación múltiple por ejemplo:

```
x = p = Z = 0 # Tanto x, p y z son variables enteras con el valor de cero
genero = estado = False # Tanto genero como estados tendran el valor de False
```

Expresión: es una combinación de operandos y operadores que al ser evaluados arrojan un resultado. Los **operadores** son símbolos especiales que toman los valores de los operandos y producen nuevos valores, **operando** valores que son afectados por los operadores, básicamente son variables, constantes o funciones.

Los operadores en función de la cantidad de operandos que afectan pueden ser *Unarios* y *Binarios*, un operador binario es aquel que afecta a dos operandos, mientras que uno unario afecta a un solo operando.

Los operadores se clasifican en:

- Aritméticos.
- Relaciónales.
- Lógicos.
- De asignación.

Operadores Aritméticos: sirven para realizar operaciones aritméticas básicas como son:

Símbolo	Identificación
**	Exponenciación binario
-	Cambio de signo, unario
*	Multiplicación binario
/	División real, binario
%	Modulo, binario
//	División entera, binario
+	Suma, binario
-	Resta, binario

Exponenciación: cuando *a* y *b* son dos números enteros la operación a^b puede definirse como a**b como equivalente a la potencia. Sin embargo cierto número de problemas físicos concretos llevaron a tratar de generalizar la fórmula anterior a valores de *b* no enteros. Cuando *b* = 1/2 la operación equivale a una raíz cuadrada, si b = 1/3 seria la raíz cubica.

Cambio de signo: cambia el signo de su operando, si A es una variable entera cuyo contenido es -3 entonces –A implica como resultado 3.

+, -, *: estos tres operadores efectúan la suma, resta y multiplicación respectivamente tanto de reales como enteros dando como resultado un numero real o entero dependiendo del tipo de los operando haciendo notar que si un operando es entero y el otro es real el resultado será real.

Operadores de la división: aquí encontramos tres operadores distintos los cuales son:

/: Realiza la división real o división ordinaria; el resultado es de tipo real, se usa con operadores reales y enteros ejemplos: ¾ = 0.75, 4/2 = 2.0; 3.5/2 = 1.75.

%: Retorna el residuo o resto de una división:

3 % 4 → 3, 4 % → 0, 3 % 2 → 1.

7.5 // 3 → 1.5

//: Retorna la parte entera de una división:

3 // 4 → 0, 4 // 2 → 2, 3 // 2 →1.

7.5 // 3 → 2.

Operadores Relaciónales: son operadores que comprueban una relación entre dos operandos dando como resultado una constante lógica (true, false), estos operadores son todos binarios, y son:

Operador	Significado	Ejemplo	Resultado
==	Igual a	1 **==** 2	False
!=	Diferente de	1 **!=** 2	True
>	Mayor que	1 **>** 2	False
<	Menor que	1 **<** 2	True
>=	Mayor o igual que	1 **>=** 2	False
<=	Menor o igual que	1 **<=** 2	True

Se pueden aplicar a cualquier tipo de dato estándar: entero, real, de caracter. Cuando se utilizan con caracteres esta comparación se hace en base a una secuencia de ordenamiento la cual se establece en un código internacional llamado código **ASCII**, en el cual los caracteres alfabéticos de la "A" a la "Z" quedan representados por los valores enteros comprendidos entre el 65 y el 90, los caracteres alfabéticos en minúscula de la "a" a la "z" representados por los enteros comprendidos entre el 97 y el 122, de tal manera que si hacemos las siguientes comparaciones sus resultados serán:

"A" **>** "B" False
"a" **<** "z" True
"A" **<** "a" True

Al usar operadores relacionales los operandos que se comparan deben ser del mismo tipo, con la excepción de que es posible comparar enteros con reales.

Operadores Lógicos: estos operadores se utilizan en expresiones lógicas, siendo una expresión lógica propiamente dicha aquella que consiste en una combinación de constantes lógicas y variables lógicas unidas mediante los operadores lógicos not, and, or, (siendo unario not) dando como resultado una constante lógica.
not: este operador hace que su operando tome el valor lógico opuesto ejemplo: not(true) → false, not(false)→ true.
and: la expresión lógica donde se use tiene el valor de true cuando todos sus operandos son true, basta que uno de los operandos sea falso para que el resultado general sea falso, ejemplo: sean P y Q dos variables lógicas:

P	Q	P and Q
True	True	True
True	False	False
False	True	False
False	False	false

or: la expresión lógica donde se use tiene el valor de true cuando al menos uno de sus operandos es true en caso contrario tendrá el valor false ejemplo: sean P y Q dos variables lógicas

P	Q	P or Q
True	True	True
True	False	True
False	True	True
False	False	false

Ejemplos de expresiones lógicas:

Expresión Lógica	Resultado
(1 **>** 0) and (3 **=** 3)	True
(0 **<** 5) or (8 **>** 5)	True
(5 **<=** 7) and (2 **>** 4)	False
not(5 **<** **>** 5)	True
not(1 **<** 2)	False

Operador de Asignación: la operación de asignación asigna el valor de la expresión de la derecha a la variable situada a la izquierda y se representa con el símbolo **=**, el formato general de la expresión de asignación es:

Variable = Expresión

Ejemplo: Total = 5, significa que a la variable Total se le asigna el valor 5, la acción de asignar es destructiva ya que le valor que tuviera la variable antes de la asignación se pierde y se reemplaza por el nuevo valor.

Ejemplo: sean Z, A y B variables numéricas Z = A + B, aquí la asignación se ejecuta en dos pasos básicos:

1. Se determina el valor de expresión (A + B) que está a la derecha del operador de asignación.
2. Se almacena el valor obtenido en la variable cuyo nombre aparece a la izquierda del operador de asignación sustituyendo así su valor anterior.

Es posible utilizar la misma variable tanto a la derecha y como a la izquierda del operador de asignación, puesto que primero se evalúa la expresión, tomando el valor actual de la variable ejemplo Total = Total + 1, aquí estamos indicando que el valor actual de Total se incrementa en uno y el resultado se le asigna a la misma variable, de manera que si el valor actual de Total es 5 una vez evaluada la expresión el valor de Total será 6.

El tipo de la variable de la izquierda debe ser igual al tipo del valor resultante de la expresión de la derecha. Aparte de este operador existen otros operadores de asignación como son: -=, +=, *=, /=, %=, //= estos función de la siguiente manera:

a += 1 esto es equivalente a: a = a + 1
b *= c esto es equivalente a: b = b * c

Orden de precedencia de los operadores.

La precedencia de los operadores queda establecida en la siguiente tabla:

Prioridad	Operador
1	**
2	-
3	*, /, //, %
4	+. -
5	<, >, <=, >=
6	==, **!=**
7	=, %=, //=, /=, *=, -=, +=, **=
8	and, or, not

Considérese la siguiente asignación:

V = 3 + 6 * 13

¿Qué valor recibe la variable V? si se procesa de izquierda a derecha efectuando la suma antes que la multiplicación el resultado será 117, pero si se evalúa de derecha a izquierda el resultado será 81, ¿Cuál es el correcto?, para resolver el problema se han establecido algunas reglas para evaluar las expresiones, específicamente, sobre el orden en que deben realizarse las operaciones individuales con cada operador, por lo que a cada operador se le asigna una precedencia o prioridad. Los operadores que tengan mayor precedencia se procesan primero y enseguida se continúa de izquierda a derecha, de acuerdo con esto la expresión anterior primero se evaluara la multiplicación ya que esta tiene mayor precedencia que la suma dando como resultado 81, sea la expresión:

8 + 7 * 3 + 4 * 5

La multiplicación por tener mayor prioridad se ejecutara primero, dando los resultados intermedios 21 y 20 quedando la expresión así: 8 +21 + 20, enseguida se procesan las sumas de izquierda a derecha dando como resultado final 49.

Cuando hay dos o más operadores de igual precedencia, la evaluación se efectúa de izquierda a derecha, ejemplo:

8 + 7 * 8 – 4 / 2

Primero se evalúan la multiplicación y luego la división dando como resultados intermedios 56 y 2, quedando la expresión:

8 + 56 – 2

Segundo como la suma y la resta están a un mismo nivel de precedencia, primero se efectúa la suma y luego la resta dando como resultado: 62

Supongamos que quisiéramos en la expresión anterior que la suma se evaluara primero que las otras operaciones, para ello debemos cambiar la prioridad de los operadores utilizando los paréntesis -( )- , el uso de los mismos indica que las operaciones encerradas entre paréntesis deben evaluarse primero, quedando la expresión así:

(8 + 7) * 8 – 4 / 2

Esto da como resultado 118, evaluándose primero la suma seguido de la multiplicación y la división quedando para el final la resta. Sea la siguiente expresión:

(3 * (6 + 2) * 8)

Los paréntesis interiores se procesan primero, dando resultado 8 este valor se multiplica por 3 resultando 24 y este a su vez por 8 dando como resultado final 192.

Operadores de cadenas

Con las cadenas podemos usar el operador + y *, el primero lo usaremos para concatenar cadenas de caracteres, este procedimiento permite unir cadenas de caracteres ejemplo:

"La casa" + "Azul" → "La casaAzul"

"La casa" + " " + "Azul" → "La casa Azul"

En el caso del operador * el efecto es el siguiente:

"La casa" * 3 → "LacasaLacasaLacasa" que sería equivalente a:

"La casa" + "La casa" + "La casa"

Estructuras de Control

Cualquier programa, sin importa su complejidad se puede construir combinando estas dos estructuras: *Secuencial y estructuras de control.*

Estructura Secuencial: significa que los pasos de un programa se ejecutan en secuencia, uno detrás del otro, en el orden en que están situados en el programa.

Estructuras de Control: son estructuras que afectan el flujo normal de un programa, estas se dividen en *Selectivas* y *Repetitivas (Ciclos)*.

Estructuras Selectivas: permiten determinar un sentido de acción en el flujo del programa sobre la base de la evaluación de una determinada condición, se usan para tomar decisiones lógicas, en ellas se evalúa una expresión lógica y en función del resultado de la misma se ejecutan o no una secuencia de pasos en el programa.

Estructuras Repetitivas o ciclos: permiten repetir automáticamente un grupo de instrucciones, ya sea un número determinado de veces o mientras que una condición particular se cumpla. Son aquellas que permiten que una operación o conjunto de ellas se repitan muchas veces.

Identación

Para hablar de estructuras de control en Python debemos hablar de Identación, que significa hacer espacios en blanco hacia la derecha para mover una línea de código, en Python se aplica la identación para indicar que las instrucciones identadas forman un bloque de código asociado a una misma estructura de control.

Estructura selectiva if ... elif ... else ...

Representamos una estructura de selección con las palabras If (Si), estas estructuras pueden ser:

- Simples
- Dobles.
- Múltiples.

La estructura de **selección simple** If ejecuta una determinada instrucción o instrucciones cuando se cumple una determinada condición.

Sintaxis

```
if condición:
    instrucción
    instrucción
```

La primera línea contiene la condición a evaluar después del if y es una expresión lógica. Esta línea debe terminar siempre por dos puntos (:), a continuación viene el bloque de órdenes que se ejecutan cuando la condición se cumple (es decir, cuando la condición es True). Es importante señalar que este bloque debe ir identado, puesto que Python utiliza el mismo para reconocer las líneas que forman un bloque de instrucciones. Al escribir dos puntos (:) al final de una línea, el IDE sangrará automáticamente las líneas siguientes. Para terminar un bloque, basta con volver al principio de la línea. Un ejemplo de esto sería:

```
numero = int(input("Escriba un número positivo: "))
if numero < 0:
    print("¡Le he dicho que escriba un número positivo!")
print("Ha escrito " + str(numero))
```

El programa anterior pide un número positivos al usuario y almacena la respuesta en la variable "numero". Después comprueba si el número es negativo. Si lo es, el programa avisa que no era eso lo que se había pedido. Finalmente, el programa imprime siempre el valor introducido por el usuario.
La estructura anterior es muy limitada porque normalmente se requiere de una que permita elegir entre dos opciones o alternativas posibles al evaluar una expresión lógica, en este caso usaremos la estructura if..else (**estructura selectiva doble**), su sintaxis es:

```
if condición:
    instrucción
    instrucción
else:
    instrucción
    instrucción
```

La primera línea contiene la condición a evaluar. Esta línea debe terminar siempre por dos puntos (:), a continuación viene el bloque de órdenes que se ejecutan cuando la condición se cumple, todas las instrucciones que están entre los dos puntos y el else (es decir, cuando la condición es True), este bloque debe ir identado,

Después viene la línea con la orden **else**, que indica a Python que el bloque que viene a continuación se tiene que ejecutar cuando la condición no se cumpla (es decir, cuando sea False). Esta línea también debe terminar siempre por dos puntos (:). La línea con la orden **else** no debe incluir nada más que el **else** y los dos puntos, seguido está el bloque de instrucciones identado que corresponde al **else**. Un ejemplo de esto sería:

```
edad = int(input("¿Cuántos años tiene? "))
if edad < 18:
    print("Es usted menor de edad")
else:
    print("Es usted mayor de edad")
print("¡Fin!")
```

El programa anterior pregunta la edad al usuario y almacena la respuesta en la variable "edad". Después comprueba si la edad es inferior a 18 años. Si esta comparación es cierta, el programa escribe que es menor de edad y si es falsa escribe que es mayor de edad. Finalmente el programa siempre se despide, ya que la última instrucción está fuera de cualquier bloque y por tanto se ejecuta siempre.

Sentencias condicionales anidadas

Una sentencia condicional puede contener a su vez otra sentencia condicional anidada. Se pueden anidar tantas sentencias condicionales como se desee. Ejemplo:

```
print("Piense un número de 1 a 4.")
print("Conteste S (sí) o N (no) a mis preguntas.")
primera = input("¿El número pensado es mayor que 2? ")
if primera == "S":
    segunda = input("¿El número pensado es mayor que 3? ")
    if segunda == "S":
        print("El número pensado es 4.")
    else:
        print("El número pensado es 3")
else:
    segunda = input("¿El número pensado es mayor que 1? ")
    if segunda == "S":
        print("El número pensado es 2.")
    else:
        print("El número pensado es 1.")
print("¡Fin!")
```

El programa anterior "adivina" el número pensado por el usuario, el cual debe pensar en un numero en el intervalo 1 – 4, a continuación el programa hace una serie de preguntas para adivinar el numero pensado por usted.

Selección multiple: if ... elif ... else ...

La estructura de control **if ... elif ... else ...** permite encadenar varias condiciones. **elif** es una contracción de **else if**. La sintaxis en Pyhton sería:

```
if condición_1:
    bloque_1
elif condición_2:
    bloque_2
else:
    bloque_3
```

- Si se cumple la condición_1, se ejecuta el bloque_1
- Si no se cumple la condición_1 pero sí que se cumple la condición_2, se ejecuta el bloque_2
- Si no se cumplen ni la condición_1 ni la condición_2, se ejecuta el bloque_3.

Esta estructura es equivalente a la siguiente estructura de **if ... else ...** anidados:

```
if condición_1:
    bloque_1
else:
    if condición_2:
        bloque_2
    else:
        bloque_3
```

Se pueden escribir tantos bloques **elif** como sean necesarios. El bloque else (que es opcional) se ejecuta si no se cumple ninguna de las condiciones anteriores y debe ser la última de la estructura si se usa, esto significa que no puede haber un else antes de un elif.

Ejemplo:

```
edad = int(input("¿Cuántos años tiene? "))
if edad < 0:
    print("No se puede tener una edad negativa")
elif edad >= 0 and edad < 18:
    print("Es usted menor de edad")
else:
    print("Es usted mayor de edad")
```

Otra forma de escribir el código anterior seria:

```
edad = int(input("¿Cuántos años tiene? "))
if edad < 0:
    print("No se puede tener una edad negativa")
elif edad < 18:
    print("Es usted menor de edad")
else:
    print("Es usted mayor de edad")
```

Estructuras Repetitivas

En muchos problemas se requiere que algunos cálculos o grupos de instrucciones se repitan una y otra vez, a continuación procederemos a examinar las diferentes estructuras que se utilizan para construir secciones de código repetitivas, como son: for y while

Las estructuras que repiten una secuencia de instrucciones un número de veces se denominan **ciclos**, denominándose **iteración** a la repetición de una secuencia de instrucciones. Por ejemplo queremos determinar el promedio de notas de un curso de Programación, para ello debemos leer todas las notas del curso y añadir cada una a una variable que llamaremos suma que contiene las sucesivas sumas parciales; esta variable se inicializa a cero, incrementándose con el valor de cada nota cada vez que una de ellas se lea, al final dividiremos la variable suma entre el total de alumnos que hay en el curso.
Como podemos observar en este caso el ciclo está constituido por: la petición de una nota, leer la nota y acumular la nota en una variable llamada suma, por cada alumno que tenga el curso se producirá una iteración de este ciclo o sea se pedirá una nota, se leerá la nota y se acumulara la nota en la variable suma y así sucesivamente hasta que se lea la nota del último alumno del curso. Este es un ejemplo típico de tareas repetitivas que incluye entrada de datos, acumulaciones parciales de estos datos, deteniéndose el proceso cuando se lee el último dato de la lista.

Definiciones básicas:
Variable contadora: variable que se modifica así misma para llevar el computo de veces que ocurre un suceso, evento o situación, este tipo de variables su incremento es un valor constante: **A = A + valor_constante** o **A += valor_conatnate.**
Variable acumuladora: variable que se modifica así misma para acumular un valor variable: **B = B + valor_variable** o **B += valor_variable.**
Variable Bandera: variable que cambia su valor para indicar que algún suceso o evento ha tenido lugar, cambia de un valor a otro como de cero a uno o de falso a cierto.
Porcentaje: número de elementos de una lista que cumplen con una condición entre la cantidad de elementos a los cuales se les comprobó si cumplen o no con la condición por cien.
Promedio: sumatoria de un valor de una lista entre número de elementos sumados.

Ciclos For… in.

Este tipo de estructura repetitiva se caracteriza porque las instrucciones del cuerpo del ciclo se ejecutan *un número determinado de veces* y de modo automático se controla el número de iteraciones o pasos a través del bloque de instrucciones del ciclo.
Sintaxis:

```
for variable in range():
    bloque_de_código
```

range genera una **lista inmutable de números enteros en sucesión aritmética.**

- Inmutable significa que no se pueden modificar.
- Una sucesión aritmética es una sucesión en la que la diferencia entre dos términos consecutivos es siempre la misma.

Para definir un **range** usaremos una de las siguientes alternativas:

- *range*(n,m,s) cumple: rango >= n y rango < m en incrementos de s.
- *range*(n,m) cumple: rango >= n y rango < m en incrementos de 1.
- *range*(m) cumple: rango >= 0 y rango < m en incrementos de 1.

Ejemplos simples para ilustrar el comportamiento del **for** con **range**.

Caso 1
```
for contador in range(3, 11, 2):
    print(contador)

Salida:
3
5
7
9
```

Caso 2
```
for contador in range(5, 9):
    print(contador)

Salida:
5
6
7
8
```

Caso 3
```
for contador in range(4):
    print(contador)

Salida:
0
1
2
3
```

En caso de que el ciclo no se incremente si no que se decremente tenemos el siguiente ejemplo:

```
for contador in range(26, 10, -4):
    print(contador)

Salida:
26
22
18
14
```

Ejemplo en el cual pediremos la nota obtenida por 4 alumnos en un curso de computación y procederemos a calcular el promedio del curso.

```
nota = 0
suma = 0
promedio = 0.0
for x in range(4):
    nota = int(input("Nota de un alumno: "))
    suma += nota
promedio = suma / 4
print("Promedio del curso: " + str(promedio))
```

Entrada y Salida:

```
Nota de un alumno: 10
Nota de un alumno: 5
Nota de un alumno: 12
Nota de un alumno: 13
Promedio del curso: 10.0
```

Las primeras tres líneas se inicializan tres variables las dos primeras enteras y la tercera real.
El encabezado del ciclo for nos indica que se ejecutar un ciclo for de 4 iteraciones la variable x controlara el valor de cada iteración.
Por cada iteración del ciclo se pedirá por consola la nota de un alumno que se almacena en la variable nota y esta se acumulara en la variable suma.
Al terminar el ciclo se calcula el promedio y se muestra por consola su valor.

Ciclos While...

Un bucle **while** permite repetir la ejecución de un grupo de instrucciones mientras se cumpla una condición (es decir, mientras el resultado de evaluar la condición sea **True**).

La sintaxis del bucle **while** es la siguiente:

```
while condición:
    cuerpo del ciclo
```

Cuando el flujo del programa alcanza la instrucción **while**, Python evalúa la condición y, si es cierta (True), ejecuta el grupo de instrucciones que forman parte del cuerpo del ciclo. Una vez ejecutado el cuerpo del ciclo, se repite el proceso (se evalúa de nuevo la condición y, si es cierta, se ejecuta de nuevo el cuerpo del ciclo) una y otra vez mientras la condición sea cierta. Únicamente cuando la condición sea falsa, el cuerpo del ciclo no se ejecutará y continuará la ejecución del resto del programa.

La variable o las variables que aparezcan en la condición se suelen llamar variables de control. Las variables de control deben definirse antes del ciclo **while** y modificarse en el cuerpo del ciclo **while**.

Ejemplo en el cual el programa escribe los números del 1 al 3.

```
i = 1
while i <= 3:
    print(i)
    i += 1
print("Programa terminado")
```

```
Salida:
1
2
3
Programa terminado
```

En el ejemplo anterior se inicializa la variable i en 1 al encontrar el ciclo while se evalua la condición si i es menor o igual a 3 como el valor de la variable es 1 la codicion es True, se ejecuta el cuerpo del ciclo, imprimiéndose el valor de i y luego incrementando el valor de i en 1 con lo cual i es igual a 2, el flujo del programa retorna a la cabecera del ciclo y se vuelve a evaluar la condición en este caso el valor de i es 2 por lo tanto es menor o igual a 3 siendo el resultado de evaluar la condición True, se ejecuta el cuerpo del ciclo de nuevo ahora el valor de i es 3 el flujo del programa retorna a la cabecera del ciclo se evalúa la condición siendo está de nuevo True, se ejecuta el cuerpo del ciclo y ahora el valor de i es 4 el flujo del programa retorna a la cabecera del ciclo y al evaluar la condición se observa que esta es False ya que el valor de i no es menor o igual a 3, se termina el ciclo ejecutándose la instrucción que esta a continuación.

El ejemplo anterior se podría haber programado con un ciclo **for**. Cuando usar un ciclo **for** y cuando un ciclo **while**, el for lo usaremos cuando conozcamos de antemano el número de veces que el ciclo se va a ejecutar, en el caso del while esta información no la tengamos a mano por lo que el ciclo lo controlaremos mediante el uso de una condición, por ejemplo porque los datos los proporciona el usuario. El siguiente ejemplo se pide un número positivo al usuario una y otra vez hasta que el usuario lo haga correctamente:

```
numero = int(input("Escriba un número positivo: "))
while numero < 0:
    print("¡Ha escrito un número negativo! Inténtelo de nuevo")
    numero = int(input("Escriba un número positivo: "))
print("Gracias por su colaboración")
```

Entrada y salida:

```
Escriba un número positivo: -4
¡Ha escrito un número negativo! Inténtelo de nuevo
Escriba un número positivo: -8
¡Ha escrito un número negativo! Inténtelo de nuevo
Escriba un número positivo: 9
Gracias por su colaboración
```

Ciclos infinitos

Si la condición del ciclo se cumple siempre, el ciclo no terminará nunca de ejecutarse y tendremos lo que se denomina un **ciclo infinito**. Los bucles infinitos no intencionados deben evitarse pues significan perder el control del programa. Para interrumpir un ciclo infinito, hay que pulsar la combinación de teclas **Ctrl+C**. Al interrumpir un programa se mostrará un mensaje de error que indica que el programa fue interrumpido por teclado (**KeyboardInterrupt**). Por desgracia, es fácil programar involuntariamente un bucle infinito, por lo que es inevitable hacerlo de vez en cuando, sobre todo cuando se está aprendiendo a programar.

Funciones matemáticas

Las funciones matemáticas en Python están concentradas en módulo (biblioteca) denominada **math**. Para su utilización es necesario realizar una importación al módulo o programa en desarrollo. La importación se realiza mediante la instrucción:

import math.*

A continuación se presenta un resumen de las funciones más utilizadas de la biblioteca **math**.

Función	código	ejemplo
Valor absoluto	math.fabs	math.fabs(-3)
Función exponencial	math.exp	math.exp(1)
Potencial	math.pow(base,exponente)	math.pow(2,3)
raiz cuadrada	math.sqrt	math.sqrt(3.0)
Coseno de un ángulo medido en radianes	math.cos	math.cos(0.7)
Seno de un agulo medido en radianes	math.sin	math.sin(0.707)
Tangente de un ángulo medido en radianes	math.tan	math.tan(1)
Conversión de radianes a grados	math.degree	math.degree(0,707)
Obtención de valor de pi	math.pi	math.pi
obtención de valor de e	math.e	math.e
Arcoseno de un valor (ángulo)	math.acos	math.acos(0.56)
Arcseno de un valor (ángulo)	math.asin	math.asin(0.7)
Conversión de grados a radianes	math.radians	math.radians(30)