5 PROGRAMACIÓN ESTRUCTURADA

¿Qué es un programa?

Un programa es una secuencia de instrucciones para un ordenador.

Los primeros lenguajes de «alto nivel» (similares al lenguaje humano) permitían crear programas formados por una lista de órdenes que se procesaban de principio a fin, salvo que, en algún instante, se indicase que se debía saltar a otra posición del programa:

```
10 PRINT "Hola"
20 GOTO 10
```

Esta forma de trabajar provocaba que, a medida que el tamaño de los programas iba aumentando, estos resultaran cada vez más difíciles de leer y corregir, puesto que era necesario analizar paso a paso de qué punto a qué otro punto del programa se saltaba: es lo que se llamó *código espagueti*.

Como alternativa, a mediados de los años sesenta se diseñaron lenguajes que incorporaban estructuras más legibles, que permitían que un fragmento de un programa se repitiera mientras se cumpliera una cierta condición o descomponer un programa en bloques, denominados *subrutinas*. Este hecho supuso el comienzo de la programación estructurada, que se aplicó a lenguajes como ALGOL, PL/I o Pascal.

Con el paso de los años, se han diseñado otras metodologías más apropiadas para crear programas de gran tamaño (formados por miles de órdenes), como la programación orientada a objetos.

COMPROMISO ODS

Un programa informático puede ser una valiosa herramienta paradar a conocer los Objetivos de Desarrollo Sostenible.

1. Crea, al finalizar esta unidad, un programa sencillo que muestre una lista numerada del 1 al 17, con los nombres de los objetivos. Al introducir el número correspondiente, deberá mostrar un resumen breve sobre a qué se refiere ese objetivo concreto, empleando para ello dos listas de datos.
2. **Mesa redonda.** ¿Qué ventaja tendría la programación de un juego para conocer mejor los ODS frente a un texto que los explique?

En anayaeducacion.es encontrarás información y vídeos sobre las principales metas de los 17 Objetivos de Desarrollo Sostenible.

1 LENGUAJES, COMPILADORES E INTÉRPRETES

1.1. Lenguajes de bajo nivel y alto nivel

Un **programa** consiste en una **secuencia de instrucciones** para un **ordenador.** Al igual que las personas utilizamos distintos idiomas para comunicarnos, existen diversos lenguajes con los que podemos dar órdenes a un dispositivo.

El problema radica en que el lenguaje que un ordenador entiende, esto es, el **«código máquina»,** que es el ejemplo más claro de lenguaje de bajo nivel, es muy diferente a los idiomas hablados por las personas. Resulta difícil de aprender y es fácil cometer errores con él. Por eso, es habitual emplear otros más parecidos a los de los humanos (los denominados ***lenguajes de alto nivel***) y utilizar herramientas que conviertan el programa a código máquina. Normalmente, estos son muy similares al inglés, aunque más estrictos.

```
PRINT "Hola"
```

Fig. 1. Escritura en pantalla usando un lenguaje de alto nivel (BASIC).

El concepto de *algoritmo* está muy relacionado con el de *programa,* aunque, a veces, se confunden al pensar que son lo mismo. En este caso, un **algoritmo** es la **secuencia de pasos** (finita y sin ambigüedades) necesaria para resolver un cierto problema. Esta expresión no se usa solo en informática, sino también en otras ciencias, como las matemáticas y la lógica. Un **programa** es el **resultado de implementar un algoritmo,** empleando un sistema informático.

1.2. Compiladores e intérpretes

Las **herramientas** encargadas de convertir nuestro programa escrito en lenguaje de alto nivel (lo que se conoce como ***programa fuente***) en código máquina, obteniendo un programa **ejecutable,** son los **compiladores.** Si este detecta algún error en el programa fuente, mostrará el pertinente mensaje de aviso y no se creará ningún ejecutable.

```
PRINT "Hola"
Hola

ESCRIBE "Hola"
Syntax error
```

Fig. 2. Comportamiento de un intérprete: análisis línea a línea en tiempo real.

Por su parte, un **intérprete** es otro tipo de **traductor.** La diferencia con el compilador es que, en los intérpretes, no se crea ningún programa ejecutable capaz de funcionar por sí solo. En el momento de ponerlo en funcionamiento, el intérprete convierte cada orden del programa fuente en su equivalente en código máquina y la pone en marcha, y luego la siguiente, y así sucesivamente. Si encuentra un error, el programa se detiene en ese punto, pero las órdenes anteriores ya se habrán ejecutado.

Esto supone que, normalmente, un **programa interpretado comenzará** a funcionar **antes que un programa compilado** (pues no es necesario traducirlo todo para empezar, sino solo la primera orden), **pero será más lento** en los procesos de cálculo intensivo (porque cada orden se puede tener que traducir varias veces: tantas como se ejecute dicha orden).

1.3. Pseudocódigo

Aunque los lenguajes de alto nivel se asemejan al lenguaje natural, es habitual no usar ninguno concreto al plantear, inicialmente, los pasos necesarios para resolver un problema, sino **emplear uno ficticio,** no tan estricto, que puede escribirse, incluso, en castellano. Este recibe el nombre de ***pseudocódigo:***

```
PEDIR numero1
PEDIR numero2
SI numero2 ≠ 0
    ESCRIBIR "Su división es ", numero1/numero2
SI NO
    ESCRIBIR "No se puede dividir entre cero"
```

1.4. Lenguajes más extendidos

Existen multitud de lenguajes de programación. Muchos de ellos son de **propósito general** (sirven para crear programas de cualquier tipo), mientras que otros están **especialmente diseñados para ciertas tareas,** por lo que pueden resultar menos adecuados para realizar otras distintas. Por ejemplo, hay lenguajes específicos para hacer cálculos matemáticos complejos, para representar y manipular imágenes, para programar robots, etc.

Existen *rankings* de valoraciones, como el índice TIOBE, que **ordenan** los lenguajes de mayor a menor **popularidad,** a partir de ciertos criterios, como la cantidad de búsquedas realizadas sobre ellos en Google y otros buscadores, o la cantidad de programas escritos en ese lenguaje que se pueden encontrar en almacenes de programas fuente, como GitHub y SourceForge.

Algunos de los **lenguajes de programación más extendidos** son **C, C++, C#, Java, JavaScript, PHP** y **Python. C** se suele usar para **crear sistemas operativos y programas** que deban acceder al *hardware;* **Java,** para **aplicaciones** en **servidores web** de gran tamaño y para el sistema **Android; C++,** en **aplicaciones de escritorio y videojuegos,** y **Python** se emplea, cada vez más, como **lenguaje** de propósito **general,** pero también en sectores como el del **análisis de datos.** Se trata de un lenguaje potente, con una sintaxis razonablemente sencilla, y para el que hay multitud de **bibliotecas,** que permiten aplicarlo a muy distintos ámbitos. Además, su intérprete está disponible para la gran mayoría de sistemas operativos actuales, de modo que se pueden crear aplicaciones para LliureX, macOS o Windows, entre otros.

1.5. Hola, mundo

El primer programa que se suele crear, al comenzar a trabajar con un lenguaje de programación, consiste en escribir algo en pantalla. Con frecuencia, ese texto que escribe el programa es un **saludo** al mundo que le rodea, un «Hola, mundo». En otros más antiguos y simples, como BASIC, esto se podía conseguir así:

```
PRINT "Hola, mundo"
```

La mayoría de los lenguajes modernos añaden nuevas posibilidades, como el **acceso a Internet** o **la manipulación de textos** de gran tamaño y de imágenes, pero, a cambio, pueden complicar un programa simple. Por ejemplo, el equivalente al programa anterior, escrito en Java, sería:

```
class HolaMundo {
  public static void main( String args[] ) {
    System.out.print( "Hola, mundo" );
  }
}
```

Por el contrario, **Python** supone una vuelta a la **simplicidad.** En este lenguaje, un programa básico tendría esta apariencia:

```
print("Hola, mundo")
```

```
Hola, mundo
```

Esta sencillez aparente no quiere decir que Python sea un lenguaje menos potente que Java, sino que se ha diseñado pensando en que su curva de aprendizaje sea menos pronunciada.

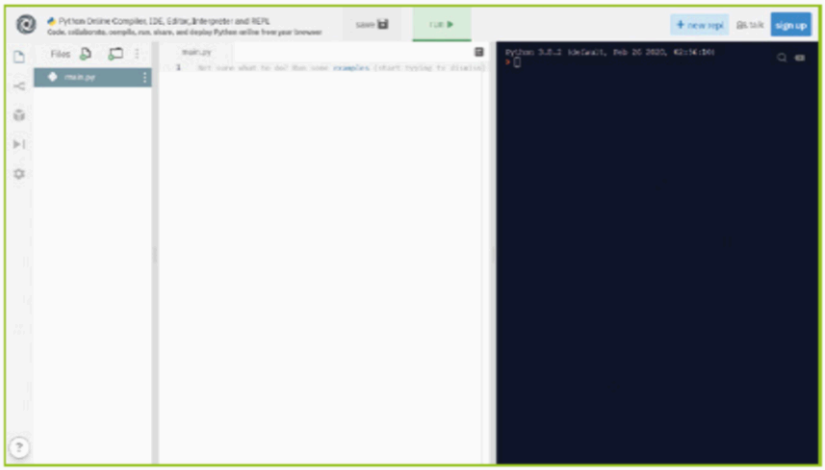
En **anayaeducacion.es** hay una práctica guiada que trata las peculiaridades de la sintaxis de Python.

1.6. Probar un programa Python en un navegador web

Resultan, cada vez más frecuentes, los **entornos de programación *online,*** que permiten teclear y probar un programa **sin necesidad de instalar** nada en nuestro equipo. Además, son **independientes** de nuestro sistema operativo, por lo que se podrían emplear, incluso, desde una tableta o un teléfono inteligente.

Es habitual que estos estén divididos en **dos paneles,** uno al lado del otro. Normalmente, el panel **izquierdo** se deberá emplear para **teclear** el programa, mientras que el panel **derecho** mostrará los **resultados.**

Para **lanzar** el programa, existirá un botón como **Run** o **Execute**, o uno que recuerde al botón **Play** de los equipos de música y multimedia.



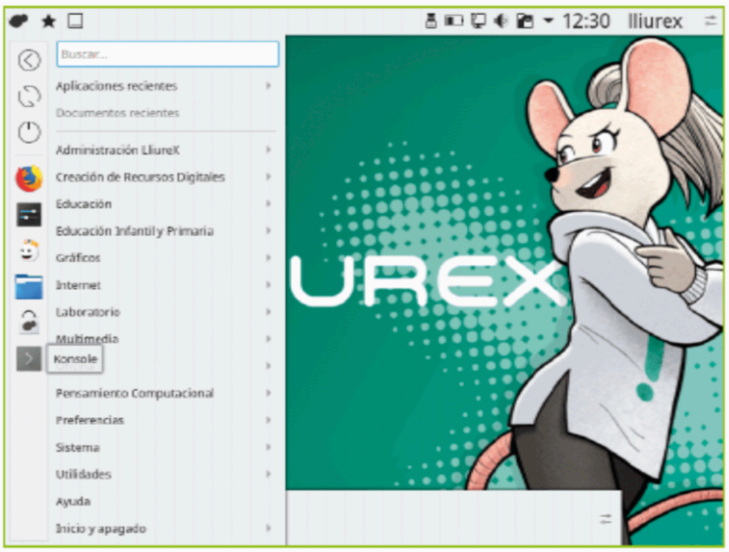
Entornos de programación *online*

Existen muchos entornos de programación *online*. Basta con teclear «Python 3 online compiler» en un buscador, como Google, para encontrar diferentes sitios web, por ejemplo, repl.it, onlinegdb, tutorialspoint o paiza.io.

1.7. Probar un programa en LliureX

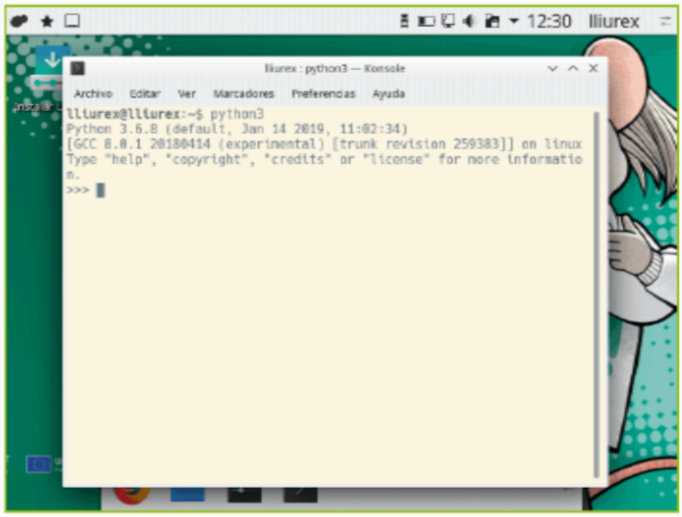
En LliureX, al igual que en la mayoría de distribuciones de Linux, es habitual que Python esté preinstalado y sea accesible a través del menú del sistema, en un apartado llamado **Desarrollo** o **Programación**.

En caso de no existir ese menú, Python se podría lanzar desde un terminal (como Konsole):



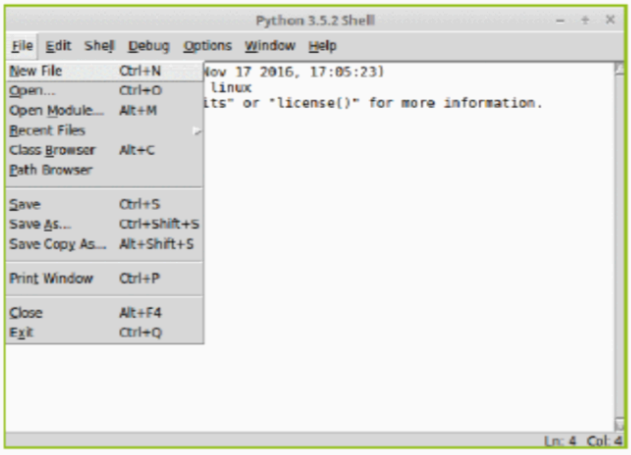
En anayaeducacion.es encontrarás un videotutorial sobre entornos *online* para programar con Python.

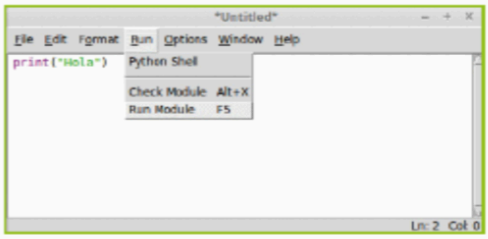
Así, desde la consola, bastaría con teclear «python3», para entrar en su entorno básico de programación.



Si existe un apartado para Python en el menú del sistema, es habitual que aparezca también la opción **Idle**, que es el **editor de texto** que viene preinstalado como parte de la distribución estándar de Python, el cual será más cómodo que el entorno básico y suficiente para programas cuya complejidad no sea muy grande.

Una vez dentro de **Idle**, para crear programas formados por varias órdenes, se deberá acceder al menú **File** y usar la opción **New File**.



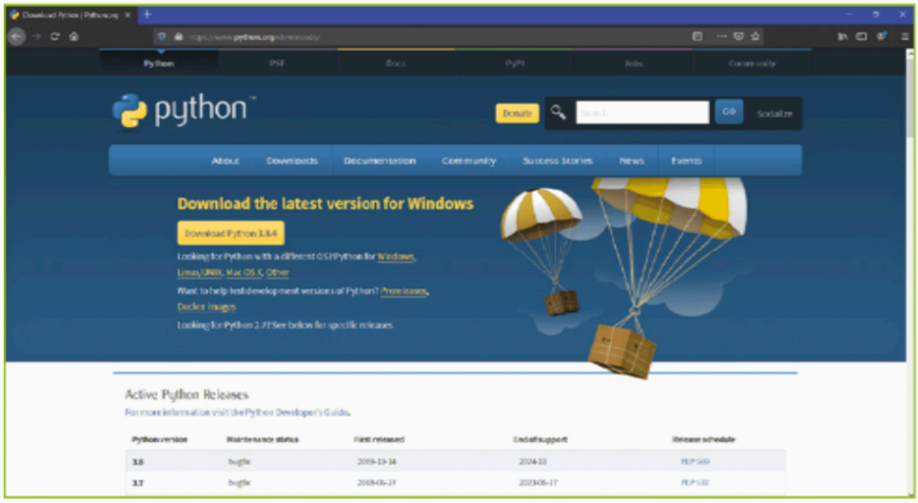


Tras teclear el programa, será necesario acceder al menú **Run** y escoger la opción **Run Module**.

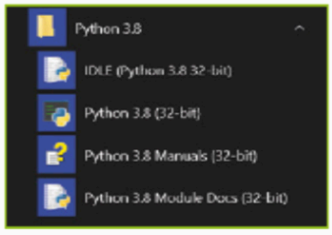
Si el programa no se hubiera guardado, en ese momento se dará la posibilidad para ello.

1.8. Probar un programa en Windows

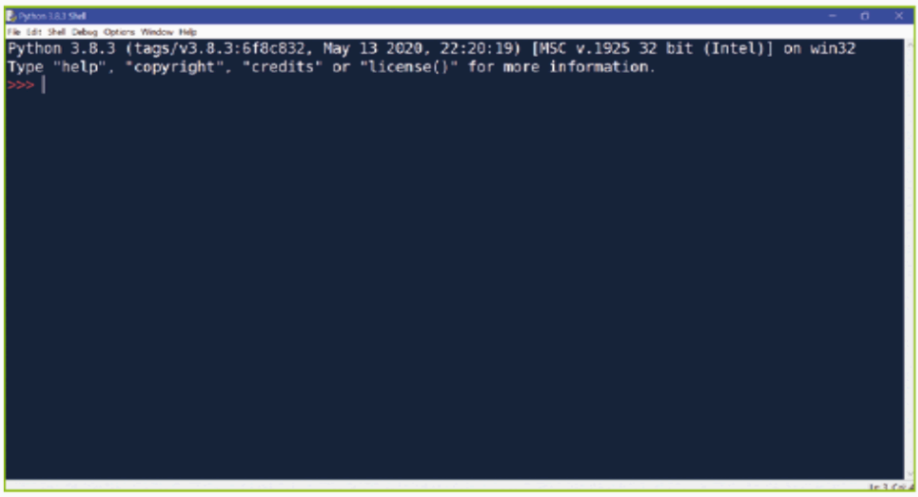
Windows, al contrario que LliureX, no incluye Python preinstalado, pero se puede descargar e instalar, de forma sencilla, desde su página web, www.python.org.



Tras instalarlo, tanto el entorno básico como **Idle** estarán disponibles en el menú de **Inicio**, dentro de la carpeta **Python**.



El manejo de ambos será idéntico al de la versión para Lliurex o cualquier otro sistema operativo.



Para programas más complejos, existe la alternativa de emplear **entornos más avanzados,** algunos de los cuales cuentan con versiones gratuitas para uso general o para uso educativo, como PyCharm, creado por JetBrains.

Actividades

1. Crea un programa que escriba tu nombre en pantalla y pruébalo.
2. Diseña un programa que primero escriba «Hola» y después, en la siguiente línea, tu nombre.
3. Si no estás usando un entorno web, sino uno de escritorio (como Lliurex, Windows o macOS), comprueba el contenido de la carpeta en la que has dejado tu programa fuente.
   a) ¿Aparece en ella algún ejecutable (fichero con el mismo nombre que tu programa, pero terminado en una extensión como **.exe**)?
   b) A partir de la observación de esa carpeta, ¿opinas que Python es un lenguaje compilado o interpretado? Argumenta la respuesta en tu cuaderno.

2
UN PROGRAMA QUE CALCULA

2.1. Realizar operaciones prefijadas

Realizar operaciones matemáticas en Python es fácil: basta con indicar la operación **sin encerrarla entre comillas:**

```
print( 5 + 7 )
```

→ 12

Si se desea, es posible incluir **espacios** adicionales entre los paréntesis, o separar los números de los operadores, para conseguir una **mejor legibilidad.**

Las operaciones de **suma, resta, multiplicación** y **división** son sencillas. Sin embargo, una operación muy habitual en programación, pero que suele resultar más difícil de comprender, es el **resto** o **módulo:** el resultado de 21%5 es el resto de dividir 21 entre 5; es decir, 1:

```
print( 21 + 5 )
print( 21 - 5 )
print( 21 * 5 )
print( 21 / 5 )
print( 21 % 5 )
```

→
```
26
16
105
4.2
1
```

Otras dos operaciones frecuentes que permite realizar Python, y que no están disponibles en todos los lenguajes de programación, son la **potencia** (que se expresa con **dos asteriscos** seguidos) y la **división entera** (sin decimales, con **dos barras** de división seguidas):

```
print( 21 ** 2 )
print( 21 // 2 )
```

→
```
441
4
```

Operaciones básicas en Python

Operador	Operación
+	Suma
-	Resta, negación
*	Multiplicación
**	Potencia
/	División
//	División entera
%	Resto

2.2. Escribir varios textos

Es posible escribir varios mensajes en la misma línea con una única orden **print**. Para ello, solo es necesario separarlos mediante **comas.** Cada coma se reflejará en pantalla como un espacio en blanco, separando los distintos mensajes. Por supuesto, se pueden incluir tanto **textos prefijados** (entre comillas) como **operaciones** (sin comillas).

```
print( "21 + 5 =", 21 + 5 )
```

→ 21 + 5 = 26

Si se desea que una orden **print** no avance a la línea siguiente, porque otra posterior deba mostrar su resultado en la misma línea, la primera de ambas deberá terminar con **end=""**. **Esta alternativa no incluye espacios intermedios en blanco.**

```
print( "21 + 5 = ", end="" )
print( 21 + 5 )
```

→ 21 + 5 = 26

Desde Python 3.6, se permite usar **cadenas con formato** (en inglés, *f-strings*, abreviatura de *formatted string literals*). Estas comienzan con una letra **f** e indican, entre llaves, aquello que se deba analizar y sustituir por su valor, como en este ejemplo:

```
print( f"21 + 5 = {21+5}")
```

→ 21 + 5 = 26

2.3. Comentarios

Las líneas que comiencen por un símbolo de **almohadilla** o *hash* (**#**) se consideran **comentarios.** Estos se incluyen para **ayudar** al programador o la programadora a recordar mejor qué se pretendía **con un fragmento** del programa o cuál es **su cometido.** Para el ordenador, se comportan como si no se hubiera escrito nada:

```
# Prueba de la potencia y la división entera
print( 21 ** 2 )
print( 21 // 2 )
```

Uso de caracteres internacionales

En los comentarios y en los textos que se muestran en pantalla, se puede escribir texto con acentos, eñes y cualquier otro carácter internacional.

Por el contrario, en los nombres de las variables, es recomendable usar solo el alfabeto inglés, como buena costumbre aplicable a otros lenguajes de programación que sean menos tolerantes que Python.

En versiones antiguas de Python (anteriores a Python 3), quizá incluso los acentos en los comentarios o en los textos en pantalla sean problemáticos. En ese caso, será necesario añadir al principio del programa la siguiente línea:

```
# coding: utf-8
```

2.4. Pedir datos a la persona usuaria

Un programa en el que todos los datos están prefijados es poco útil; no tiene mucho sentido que sea preciso modificar el código fuente y relanzar el programa cada vez que se hace necesario probar otros nuevos. Por el contrario, lo habitual es que esos **datos** sean **introducidos** por el usuario o la usuaria —la alternativa más sencilla— o se **lean desde un fichero, una base de datos** o una **red de ordenadores** (Internet o cualquier otra).

Para **leer datos** en Python, se emplea la orden **input()**, y será necesario disponer de un lugar donde guardarlos, un **espacio de memoria** al que se le dará un nombre: es lo que se conoce como una ***variable.*** En otros lenguajes de programación, es necesario declarar aquellas que un programa va a emplear, detallando, además, el tipo de datos que se van a guardar (por ejemplo, si será un texto o un número entero). En Python, esto no es necesario, ya que el propio **sistema se encargará de deducir** qué **tipo de datos** debe guardar cada variable **y de almacenarlos** correctamente. Por ejemplo, el siguiente programa pregunta a la persona usuaria su nombre y luego la saluda:

```
print("Dime tu nombre:")
nombre = input()
print("Hola,", nombre)
```

```
Dime tu nombre:
Enrique
Hola, Enrique
```

Ese programa puede abreviarse un poco, porque, entre los paréntesis de la orden **input()**, se puede indicar un mensaje de aviso, que ayude al usuario o la usuaria a saber qué tipo de información debe introducir. En este caso, y al contrario que en el ejemplo anterior, el texto se introducirá en la misma línea, a continuación del aviso, y sin ningún espacio adicional de separación:

```
nombre = input("Dime tu nombre: ")
print("Hola,", nombre)
```

```
Dime tu nombre: Enrique
Hola, Enrique
```

En ambos ejemplos, se pide a la persona que introduzca su nombre, y el texto que teclee se almacena en un espacio de memoria, al que se podrá acceder usando la palabra «nombre». Esta palabra se ha elegido por legibilidad, pero también podría haber sido algo tan breve como «n», o algo más detallado, como «nombreDelUsuario». Los **nombres** de las variables deben **empezar por una letra, no deben contener espacios** intermedios, y **respetar las mayúsculas y minúsculas** que se hayan escogido, de modo que el siguiente programa daría un mensaje de error al llegar a la segunda línea, porque la palabra «nombre» empieza por mayúsculas en la orden **print**:

```
nombre = input("Dime tu nombre: ")
print("Hola,", Nombre)
```

```
Dime tu nombre: Enrique
Traceback (most recent call last):
  File "saludo.py", line 2, in <module>
    print("Hola,", Nombre)
NameError: name 'Nombre' is not defined
```

2.5. Operaciones con datos numéricos introducidos por el usuario o la usuaria

Si lo que se desea es que el usuario o la usuaria **introduzca un número,** será necesario indicar a Python que el resultado de **input()** debe ser tratado como un número. En programación, se suele distinguir entre **números enteros** (sin decimales) y **números reales** o **de coma flotante** (que sí permiten decimales, pero pueden tener cierta pérdida de precisión y, según el sistema concreto, ser algo más lentos de manipular que los números enteros).

Así, el siguiente programa pide un número entero a la persona usuaria y, cuando se introduce, calcula su triple:

```
# Triple de un número
n = int ( input("Dime un numero: ") )
print("El triple de tu numero es", n*3)
```

```
Dime un numero: 15
El triple de tu numero es 45
```

Tal como se puede observar, para que un dato introducido por la persona se tome como **número entero,** basta con encerrarlo dentro de **int()**.

Es habitual que un programa emplee **varias variables,** tanto si se van a solicitar varios datos al usuario o la usuaria como si se van a realizar cálculos intermedios. Por ejemplo, se podría calcular la suma de dos números reales así:

```
# Sumar dos datos introducidos por el usuario
n1 = float ( input("Dime un número: ") )
n2 = float ( input("Dime otro número: ") )
suma = n1 + n2
print("Su suma es", suma)
```

```
Dime un numero: 23
Dime otro numero: 58
Su suma es 81
```

2.6. Funciones matemáticas

Existen muchas funciones matemáticas incorporadas en Python, como estas:

- Raíz cuadrada: **math.sqrt(x)**
- x elevado a y: **math.pow(x,y)**
- Coseno: **math.cos(x)**
- Seno: **math.sin(x)**
- Tangente: **math.tan(x)**
- Exponencial de x (*e* elevado a x): **math.exp(x)**
- Logaritmo natural (o neperiano) en base *e:* **math.log(x)**
- Logaritmo en base 10: **math.log10(x)**

Estas funciones **no son parte del lenguaje base,** sino de la **biblioteca math**, por lo que, para utilizarlas, el programa deberá comenzar con la línea **import math**, como en el siguiente ejemplo:

```
# Raíz cuadrada de un número

import math

n = float ( input("Dime un número: ") )
raiz = math.sqrt(n)
print( "Su raíz cuadrada es", raiz )
```

```
Dime un numero: 8
Su raíz cuadrada es
2.8284271247461903
```

División de números enteros

Al contrario que en algunos lenguajes de programación (en general, C y los que derivan de él, como C++ o Java), la división de dos números enteros se calcula con decimales en Python, es decir **print(7/2)** escribiría 3.5, mientras que en otros lenguajes el resultado de la orden equivalente es un número sin decimales (3).

Actividades

4 Crea un programa que calcule la diferencia entre 102 y 37, y otro que calcule el producto de 84 y –23.

5 Realiza un programa que pida dos números enteros y calcule la división del primero entre el segundo.

Para las **funciones trigonométricas,** el ángulo se indica en **radianes** en vez de en grados. Así, si el dato original está en grados, habrá que convertirlo a radianes, teniendo en cuenta la equivalencia 180 grados = π radianes.

El número π, en Python, se representa como **math.pi**, tal como se puede ver en el siguiente ejemplo:

```
# Funciones matemáticas: coseno

import math

anguloGrados = 45
anguloRadianes = anguloGrados * math.pi / 180
print( "El coseno de", anguloGrados, "es", math.cos(anguloRadianes) )
```

```
El coseno de 45 es
0.7071067811865476
```

2.7. Partir líneas largas

A pesar de que, por lo general, Python es más compacto que otros lenguajes de programación, en ocasiones una orden ocupará más que el ancho visible de la pantalla, lo que puede disminuir la legibilidad del programa, al tener que hacer *scroll* a un lado y a otro, para revisarlo.

En la práctica, se recomienda que una **línea no esté formada por más de 80 caracteres,** y que si una orden concreta va a ser más larga, **se descomponga en varias líneas.**

Existen dos formas de **partir líneas: dentro de un paréntesis,** se podrá partir la línea (lo deseable, tras una coma o un operador matemático) y **tabular** la continuación **a la derecha** (habitualmente, cuatro espacios); o bien, **en líneas que no contengan paréntesis,** se puede emplear la **barra invertida** (\ ), para indicar que una determinada **orden no está completa,** sino que continúa en la línea siguiente:

➜ Dentro de un paréntesis

```
# Coseno y orden que abarca dos líneas (1)

import math

anguloGrados = 45
anguloRadianes = anguloGrados * math.pi / 180
print( "El coseno de", anguloGrados, "es",
   math.cos(anguloRadianes) )
```

➜ En líneas que no contengan paréntesis

```
# Coseno y orden que abarca dos líneas (2)

import math

anguloGrados = 45
anguloRadianes = anguloGrados * \
   math.pi / 180
print( "El coseno de", anguloGrados, "es",
   math.cos(anguloRadianes) )
```

Actividades

6 Crea un programa que calcule la raíz cuadrada del número que elija el usuario o la usuaria.

7 Diseña un programa que calcule la raíz cúbica del número introducido por la persona usuaria (pista: deberás elevar ese número a 1/3).

8 Elabora un programa que pida, al usuario o la usuaria, una cantidad de millas terrestres y calcule su equivalente en kilómetros (1 milla = 1,609 km).

9 Realiza un programa que pida cinco números reales a la persona usuaria y calcule su media.

10 Diseña un programa que calcule el seno de ángulos con los siguientes grados: 30, 45, 60 y 90 °.

11 Si se ingresan *E* euros en un banco, con un interés *R*, durante *N* periodos de tiempo, el dinero que se obtendrá finalmente vendrá dado por el interés compuesto $(E \cdot (1+R)^N)$. A partir de estos datos, compón un programa que calcule el interés para los datos introducidos la persona usuaria.

3
TOMA DE DECISIONES

3.1. if

Para **comprobar** si se cumple una **determinada condición** e indicar qué pasos se deben dar en dicho caso, se emplea la orden **if** («si» condicional). Una vez escrita, tras la palabra **if**, se señalará la condición que se desea evaluar, seguida de dos puntos (**:**).

La orden u órdenes que se deban realizar, en caso de que se cumpla esa condición, se detallará en la línea o líneas siguientes, tabulando un poco más a la derecha (de nuevo, cuatro espacios es el estándar más aceptado por regla general):

```
# Condiciones con if: comprobar si un número es positivo

numero = int ( input("Escribe un numero: ") )
if numero > 0 :
   print("El numero es positivo.")
```

```
Escribe un numero: 5
El numero es positivo.
```

3.2. Operadores relacionales: <, <=, >, >=, ==, !=

Igual que se observa en el ejemplo del apartado anterior, el símbolo **mayor que** (**>**) se emplea para comprobar si un número es mayor que otro. Por su parte, el símbolo **menor que** (**<**) también es sencillo, pero los demás operadores de comparación son un poco menos evidentes, tal como se puede observar en la siguiente tabla:

Relación de operadores y sus significados

Operador	Operación
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual a
!=	No igual a (Distinto de)

Los espacios intermedios

Los espacios intermedios junto a los paréntesis y operadores no son necesarios, pero en ocasiones el programa puede resultar más legible con espacios adicionales.

A efectos prácticos, es lo mismo escribir

```
if x > y :
   print( "x es mayor" )
```

que esta alternativa más compacta:

```
if x>y:
   print("x es mayor")
```

```
# Comprobar el valor de una variable

numero = int ( input("Escribe un numero: ") )
if numero == 0 :
   print("Has escrito cero.")
```

```
Escribe un numero: 0
Has escrito cero.
```

Es importante tener en cuenta que los operadores formados por **dos símbolos,** como >=, **no** deben tener un **espacio** entre ambos símbolos, o no se reconocerán correctamente, tal como muestra el siguiente ejemplo:

```
# Error al comprobar el valor de una variable

numero = int ( input("Escribe un numero: ") )
if numero > = 0 :
   print("Es positivo o cero.")
```

```
File "main.py", line 3
  if numero > = 0:
             ^
SyntaxError: invalid syntax
```

3.3. El caso contrario: else

También es habitual indicar lo que debe hacer el programa **si no se cumple una determinada condición,** para lo cual se añade la cláusula **else**. Al igual que ocurre con el bloque que sigue a **if**, las órdenes que siguen a **else** deben tabularse a la derecha:

```
# if y else

numero = int ( input("Escribe un numero: ") )
if numero == 0 :
   print("Has escrito cero.")
else :
   print("Has escrito un numero distinto de cero.")
```

Cláusula else

En Python y otros muchos lenguajes de programación, la palabra «else» no puede existir por sí sola, sino como parte opcional de una orden **if**, y según cada lenguaje concreto, quizá de alguna otra. Por eso, no se suele hablar de «la orden **else**», sino de «la cláusula **else**».

3.4. Bloques de órdenes

Tal como ya se ha anticipado, cuando sea necesario **ejecutar varias órdenes** al cumplirse una condición, estas deberán **tabularse** más **a la derecha:**

```
# Bloques de órdenes

numero = int ( input("Escribe un numero: ") )
if numero == 0 :
   print("Has escrito cero.")
   print("Tambien puedes escribir negativos.")
else :
   print("Has escrito un numero distinto de cero.")
```

```
Escribe un numero: 0
Has escrito cero.
Tambien puedes escribir negativos.
```

```
Escribe un numero: 2
Has escrito un numero distinto de cero.
```

Un ejemplo adicional puede ser multiplicar dos números introducidos por la persona usuaria, pero pidiendo el segundo número solo en caso de que el primero no sea cero:

```
print("Vamos a multiplicar dos números...")
n1 = int ( input("Dime el primero: ") )
if n1 == 0 :
   print("Al multiplicar por cero obtendrás cero.")
else :
   n2 = int ( input("Dime el segundo: ") )
   print("Su producto es... ", end ="")
   print(n1*n2)
```

```
Vamos a multiplicar dos números...
Dime el primero: 0
Al multiplicar por cero obtendrás cero.
```

```
Vamos a multiplicar dos números...
Dime el primero: 2
Dime el segundo: 3
Su producto es... 6
```

Actividades

12 Crea un programa que pida un número entero al usuario o la usuaria e indique si es par o no. Para ello, deberá comprobar si el resto que obtiene al dividir dicho número entre dos es 0: **if numero % 2 == 0**.

13 Realiza un programa que pida a la persona usuaria dos números enteros y diga cuál es mayor.

14 Elabora un programa que pida dos números a la persona usuaria. Si el segundo no es 0, mostrará la división de ambos; en caso contrario, aparecerá un mensaje de error.

15 Crea un programa que pida dos números e indique si el primero es múltiplo del segundo.

3.5. Encadenar condiciones: and, or, not

Las condiciones se pueden **encadenar** con **and** (y), **or** (o), **not** (no), etc., igual que se indica en la tabla del margen. Así, se pueden escribir expresiones como:

Relación de operadores y sus significados

Operador	Significado
and	Y
or	O
not	No

```
# Condiciones múltiples

numero = int ( input("Escribe un numero: ") )
if not (numero == 0):
   print ("No es cero.")
if (numero == 2) or (numero == 3):
   print ("Es dos o tres.")
if (numero >= 2) and (numero <= 7):
   print ("Esta entre 2 y 7 (incluidos).")
```

```
Escribe un numero: 3
No es cero.
Es dos o tres.
Esta entre 2 y 7 (incluidos).
```

3.6. Expresiones condicionales

Existe otra forma de **asignar un valor a una variable,** en función de **si se cumple una condición** o no. Se trata de la ***expresión condicional*** u ***operador ternario,*** que equivale a otra forma de escribir la orden **if**, en la que actúa como operador.

A título de ejemplo, un planteamiento habitual para calcular el mayor de dos números sería:

```
# if "convencional"

a = 3
b = 5
if a>b:
   mayor = a
else:
   mayor = b
print ("El mayor es", mayor)
```

```
El mayor es 5
```

Pero Python permite también formularlo de la siguiente forma, más compacta pero (según el caso) también algo menos legible:

```
# if como expresión condicional

a = 3
b = 5
mayor = a if a>b else b
print ("El mayor es", mayor)
```

```
El mayor es 5
```

Es decir, su sintaxis es:

```
variable = valor1 if condición else valor2
```

Equivale a decir «si se cumple la condición, toma el primer valor; si no se cumple, toma el segundo valor».

Otro ejemplo sería una forma compacta de ver si un número es par:

```
dato = 6
esPar = "Si" if dato%2 == 0 else "No"
print ("Es par?", esPar)
```

```
Es par? Si
```

3.7. elif

Cuando se deben analizar varias **condiciones consecutivas,** el hecho de tener que tabular cada nuevo bloque más a la derecha puede hacer que el programa adquiera la apariencia de una escalera y resulte más **difícil de leer,** tal como ocurre con el siguiente programa, que escribe el nombre de los números del 1 al 5:

```
# Condiciones sin elif

numero = int ( input("Introduce un numero del 1 al 5: ") )
if numero == 1 :
  print("Uno")
else :
  if numero == 2 :
    print("Dos")
  else :
    if numero == 3 :
      print("Tres")
    else :
      if numero == 4 :
        print("Cuatro")
      else :
        if numero == 5 :
          print("Cinco")
        else :
          print("Valor incorrecto!")
```

La alternativa es usar la orden **elif**, que permite **enlazar** los **else : if** de una forma más compacta y evitar ese aumento de las tabulaciones:

```
# Condiciones con elif

numero = int ( input("Introduce un numero del 1 al 5: ") )
if numero == 1 :
  print("Uno")
elif numero == 2 :
  print("Dos")
elif numero == 3 :
  print("Tres")
elif numero == 4 :
  print("Cuatro")
elif numero == 5 :
  print("Cinco")
else :
  print("Valor incorrecto!")
```

En otros lenguajes, como los que proceden de C, existe una orden **switch**, para comprobar varios posibles valores de una variable. Esa orden no existe en Python, sino que se debe emplear bloques **if-elif**. Este planteamiento hace el programa un poco menos legible, pero también más versátil, porque, normalmente, la orden **switch** solo permite ver valores exactos, y no operadores como **>** o **<=**.

Actividades

16 Elabora un programa que pida, al usuario o la usuaria, dos números enteros y diga: «Uno de los números es positivo», «Los dos números son positivos» o bien «Ninguno de los números es positivo», según corresponda.

17 Crea un programa que pida tres números reales e indique el valor numérico del mayor de ellos (no que informe de cuál es su posición, ya que algún valor puede estar repetido).

18 Diseña, usando **if encadenados** en vez del operador **%**, un programa que pida un número del 1 al 10 y diga si es múltiplo de 3 o no.

19 Crea un programa, usando **elif** en vez de **%**, que pida un número del 1 al 10 y diga si es múltiplo de 3 o no.

20 Haz, usando **elif**, un programa que pida un número entero del 1 al 10 y escriba el nombre de la nota correspondiente (por ejemplo, tanto para el 7 como para el 8 deberá escribir «Notable»).