EJERCICIO 9

APRENDICES :juan camilo enciso camacho y juan david ospina

1 ¿ Qué es un lenguaje de programación?

En [informática](https://concepto.de/informatica/), se conoce como lenguaje de programación a un programa destinado a la construcción de otros [programas informáticos](https://concepto.de/programa-informatico/). Su nombre se debe a que comprende un [lenguaje formal](https://concepto.de/lenguajes-formales/) que está diseñado para organizar [algoritmos](https://concepto.de/algoritmo-en-informatica/) y procesos lógicos que serán luego llevados a cabo por un ordenador o sistema informático, permitiendo controlar así su comportamiento físico, lógico y su [comunicación](https://concepto.de/comunicacion/) con el usuario humano.

×

Dicho lenguaje está compuesto por símbolos y reglas sintácticas y semánticas, expresadas en forma de instrucciones y relaciones lógicas, mediante las cuales se construye el [código fuente](https://concepto.de/codigo-fuente/) de una aplicación o pieza de software determinada. Así, puede llamarse también lenguaje de programación al resultado final de estos [procesos](https://concepto.de/proceso/) creativos.

La implementación de lenguajes de programación permite el trabajo conjunto y coordinado, a través de un conjunto afín y finito de instrucciones posibles, de diversos programadores o arquitectos de [software](https://concepto.de/software/), para lo cual estos lenguajes imitan, al menos formalmente, la lógica de los [lenguajes humanos](https://concepto.de/lenguaje/) o naturales.

No deben confundirse, sin embargo, con los distintos tipos de lenguaje informático. Estos últimos representan una categoría mucho más amplia, en donde están contenidos los lenguajes de programación y muchos otros [protocolos informáticos](https://concepto.de/protocolo-informatico/), como el [HTML](https://concepto.de/html/) de las [páginas web](https://concepto.de/pagina-web/).

Fuente: [https://concepto.de/lenguaje-de-programacion/#ixz 8Fp q1w 19Z](https://concepto.de/lenguaje-de-programacion/#ixzz8Fpq1w19Z)

2 ¿ Qué diferencia hay entre un intérprete y un compilador?

Compilador e intérprete son dos herramientas fundamentales para la programación de [software](https://muytecnologicos.com/diccionario-tecnologico/software). Ambos se utilizan para convertir el código fuente escrito por los desarrolladores en un lenguaje que el [sistema informático](https://muytecnologicos.com/diccionario-tecnologico/sistema-informatico) entienda. Con todo, hay algunas diferencias clave entre ellos.

En primer lugar, un compilador toma el código fuente como una sola entrada y lo traduce a un formato binario. Por otro lado, un intérprete interpreta ejecuta instrucciones línea a línea sin convertir el código fuente en binario. En segundo lugar, la velocidad de ejecución es mejor con compiladores que con intérpretes debido a que el formato binario es mucho más rápido para ser procesado por la computadora. Por otro lado, los intérpretes son más flexibles que los compiladores ya que no necesitan generar código binario antes de ejecutarlo.

Otra diferencia importante es la facilidad de depuración de errores: los compiladores suelen detectarlos antes del tiempo de ejecución. Mientras que los intérpretes sólo detectan errores cuando se está ejecutando el programa. Lo que indica que los errores detectados por compiladores pueden corregirse con mayor facilidad y rapidez.

Para resumir, hay varias diferencias entre compilador e intérprete:

* Un compilador traduce el código fuente a formato binario antes de la ejecución. Por otro lado, un intérprete interpreta las instrucciones línea a línea sin conversiones previas.
* Los compiladores generalmente tienen velocidades de ejecución superiores a las de los intérpretes.
* Los intérpretes son más flexibles ya que no necesitan generar código binario previamente.
* Los errores detectados por los compiladores pueden corregirse con mayor facilidad y rapidez.

3 ¿ que es un programa ?

Un **programa informático** es un conjunto ordenado de instrucciones, escritas en un lenguaje de programación, para realizar una tarea en particular dentro de una computadora. En palabras sencillas, es una secuencia de órdenes que le indican a una computadora qué hacer.

Por lo regular, estos programas son desarrollados por personas que se dedican profesionalmente a la escritura de código, los cuales se conocen como **programadores**.

Los programas poseen en su núcleo un **código fuente**, que es el texto legible por los humanos y escrito en un lenguaje de programación.

Finalmente, un **lenguaje de programación** es la herramienta que permite que una persona (un programador) pueda codificar un cúmulo de instrucciones lógicas (el programa informático), para que una computadora pueda ejecutarla y que esta responda llevando a cabo una serie determinada de acciones.

Algunos ejemplos de lenguajes de programación, son:

* Python.
* Java.
* C++.
* JavaScript.
* Kotlin.
* PHP.
* Ruby.
* Swift.

4 tipos de errores en programación

ERRORES DE SINTAXIS

Son todos aquellos que se generan por infringir las normas de escritura de un lenguaje. Suelen deberse a olvidos o a desconocimiento (programadores principiantes) y comprenden falta o mal uso de elementos separadores (comas, puntos y comas, dos puntos, etc.), palabras mal escritas (por ejemplo Mientras en vez de Mientras o Finalizar en vez de Finalizar).

ERRORES POR PROCESOS NO VÁLIDOS

Cada lenguaje tiene sus particularidades o formas de concepción. Cualquier sintaxis aparentemente correcta pero que infrinja las normas de construcción de programas que define el lenguaje dará lugar a un error. Las posibilidades son muy variadas. Por citar algunas habituales señalaremos:

* Indeterminaciones matemáticas (p. ej. SQR(a) con a < 0).
* Asignar un valor a la variable que no coincide con el tipo declarado para la variable.
* Uso de variables no declaradas.
* Modificaciones no permitidas del número de localizadores de un array dinámico.
* Llamadas a módulos que no existen.
* No pasar los parámetros o pasar un número incorrecto de parámetros a un módulo genérico.
* Tratar de extraer datos o usar ficheros que no se encuentran.

ERRORES LÓGICOS

Son todos aquellos derivados de un mal diseño de los algoritmos o la arquitectura modular. Las posibilidades son muchas y los efectos posibles también. Pueden ir desde el bloqueo o detención indeseada del programa hasta un resultado incorrecto, una parte del programa que no se ejecuta, etc.

# 5 ¿ Qué es un debugging?

El debugging o depuración es una técnica informática que se utiliza para encontrar y corregir errores en un programa de computadora. Esto permite a los desarrolladores detectar y eliminar rápidamente problemas en su código antes de que el programa se lance al mercado.

Para qué sirve el debugging:

* **Revela los errores del programa**. El debugging nos ayuda a identificar los errores específicos en nuestro código, lo que nos permite solucionarlos inmediatamente.
* **Mejorar la calidad del código**. El debugging nos ayuda a mejorar la calidad de nuestro código, haciendo que nuestra aplicación sea más estable y confiable para los usuarios finales.
* **Mejorar la velocidad de ejecución**. El debugging puede ayudarnos a optimizar el rendimiento de nuestras aplicaciones. En consecuencia, se ejecutarán más rápido y con mayor estabilidad.
* **Reduce las horas de trabajo**. Al eliminar errores rápidamente, el debugging reduce las horas de trabajo necesarias para completar un proyecto, lo que significa menos tiempo y dinero gastados en reparaciones y pruebas adicionales.

6 ¿ Qué es python?

se define como un “**lenguaje de programación versátil**, multiplataforma y multiparadigmaque se destaca por su código legible y limpio”. Cuenta con una licencia de código abierto que permite su utilización en distintos contextos de forma gratuita. A su vez, se emplea en plataformas de alto tráfico como Google, YouTube o Facebook. Python atrae por su sencillez y exactitud en la sintaxis, ya que se trata de un lenguaje como cualquier otro, pero a nivel informático.

Su principal objetivo es la **automatización de procesos,** lo que hará de las tareas algo mucho más simple. En este sentido, Python **crea un código** con gran legibilidad, que ahorra tiempo y recursos. Uno de sus puntos fuertes es que “comprueba los errores sobre la marcha” para solucionarlos cuando afectan a la memoria, lo que mantiene la integridad de la matriz y evita las complicaciones a la hora de escribir el código.

7 explique variables, tipos de datos, operaciones y expresiones básicas de Python

En Python podemos encontrar distintos tipos de datos con diferentes características y clasificaciones. En este tutorial repasaremos los tipos de datos básicos, aunque te introduciré otros tipos de datos que veremos en tutoriales posteriores.

Los tipos de datos básicos de Python son los *booleanos*, los *numéricos* (enteros, punto flotante y complejos) y las *cadenas de caracteres*.

Python también define otros tipos de datos, entre los que se encuentran:

* Secuencias: Los tipos [*list*](https://j2logo.com/python/tutorial/tipo-list-python/), [*tuple*](https://j2logo.com/python/tutorial/tipo-tuple-python/) y [*range*](https://j2logo.com/python/tutorial/tipo-range-python/)
* Mapas: El tipo [*dic*t](https://j2logo.com/python/tutorial/tipo-dict-python/)
* Conjuntos: El tipo [*set*](https://j2logo.com/python/tutorial/tipo-set-python/)
* Iteradores
* Clases
* Instancias
* Excepciones

### Números enteros

El tipo de los números enteros es int. Este tipo de dato comprende el conjunto de todos los números enteros, pero como dicho conjunto es infinito, en Python el conjunto está limitado realmente por la capacidad de la memoria disponible. No hay un límite de representación impuesto por el lenguaje.

Pero tranquilidad, que para el 99% de los programas que desarrolles tendrás suficiente con el subconjunto que puedes representar.

Un número de tipo int se crea a partir de un literal que represente un número entero o bien como resultado de una expresión o una llamada a una función.

Ejemplos:

>>> a = -1 # a es de tipo int y su valor es -1

>>> b = a + 2 # b es de tipo int y su valor es 1

>>> print(b)

## Tipo booleano

En Python la clase que representa los valores booleanos es bool. Esta clase solo se puede instanciar con dos valores/objetos: True para representar verdadero y False para representar falso.

Una particularidad del lenguaje es que cualquier objeto puede ser usado en un contexto donde se requiera comprobar si algo es verdadero o falso. Por tanto, cualquier objeto se puede usar en la condición de un if o un while (son estructuras de control que veremos en tutoriales posteriores) o como operando de una operación booleana.

Por defecto, **cualquier objeto es considerado como verdadero con dos excepciones**:

* Que implemente el método \_\_bool\_\_() y este devuelva False.
* Que impleménte el método \_\_len\_\_() y este devuelva 0.

Además, los siguientes objetos/instancias también son consideradas falsas:

* None
* False
* El valor cero de cualquier tipo numérico: 0, 0.0, 0j, …
* Secuencias y colecciones vacías (veremos estos tipos en otros tutoriales): '', (), [], {}, set(), range(0)

## Tipo cadena de caracteres

Una vez que hemos acabado con los números, es el turno de las letras

Otro tipo básico de Python, e imprescindible, son las secuencias o cadenas de caracteres. Este tipo es conocido como string aunque su clase verdadera es str.

Formalmente, un string es una secuencia **inmutable** de caracteres en formato [**Unicode**](https://es.wikipedia.org/wiki/Unicode).

Para crear un string, simplemente tienes que encerrar entre comillas simples '' o dobles "" una secuencia de caracteres.

Se puede usar indistintamente comillas simples o dobles, con una particularidad. Si en la cadena de caracteres se necesita usar una comilla simple, tienes dos opciones: usar comillas dobles para encerrar el string, o bien, usar comillas simples pero anteponer el carácter \ a la comilla simple del interior de la cadena. El caso contrario es similar.

Veamos todo esto con un ejemplo:

>>> hola = 'Hola "Pythonista"'

>>> hola\_2 = 'Hola \'Pythonista\''

>>> hola\_3 = "Hola 'Pythonista'"

>>> print(hola)

Hola "Pythonista"

>>> print(hola\_2)

Hola 'Pythonista'

>>> print(hola\_3)

Hola 'Pythonista'

A diferencia de otros lenguajes, en Python no existe el tipo «carácter». No obstante, se puede simular con un string de un solo carácter:

>>> caracter\_a = 'a'

>>> print(caracter\_a)

a

Revisa este tutorial si quieres conocer más sobre la clase [str de Python](https://j2logo.com/python/tutorial/tipo-str-python/).

## Otros tipos

Hasta aquí hemos repasado los tipos de datos básicos de Python, sin embargo, el lenguaje ofrece muchos tipos más. Te hago aquí un avance de los más importantes, aunque los veremos en detalle en otros tutoriales.

Además de los tipos básicos, otros tipos fundamentales de Python son las secuencias (list y tuple), los conjuntos (set) y los mapas (dict).

Todos ellos son tipos compuestos y se utilizan para agrupar juntos varios valores.

* Las [listas](https://j2logo.com/python/tutorial/tipo-list-python/) son secuencias mutables de valores.
* Las [tuplas](https://j2logo.com/python/tutorial/tipo-tuple-python/) son secuencias inmutables de valores.
* Los [conjuntos](https://j2logo.com/python/tutorial/tipo-set-python/) se utilizan para representar conjuntos únicos de elementos, es decir, en un conjunto no pueden existir dos objetos iguales.
* Los [diccionarios](https://j2logo.com/python/tutorial/tipo-dict-python/) son tipos especiales de contenedores en los que se puede acceder a sus elementos a partir de una clave única.

>>> lista = [1, 2, 3, 8, 9]

>>> tupla = (1, 4, 8, 0, 5)

>>> conjunto = set([1, 3, 1, 4])

>>> diccionario = {'a': 1, 'b': 3, 'z': 8}

>>> print(lista)

[1, 2, 3, 8, 9]

>>> print(tupla)

(1, 4, 8, 0, 5)

>>> print(conjunto)

{1, 3, 4}

>>> print(diccionario)

{'a': 1, 'b': 3, 'z': 8}

## Conocer el tipo de una variable

Ahora te voy a presentar dos funciones para que puedas jugar con todo lo que hemos visto en este tutorial. Son type() e isinstance():

* type() recibe como parámetro un objeto y te devuelve el tipo del mismo.
* isinstance() recibe dos parámetros: un objeto y un tipo. Devuelve True si el objeto es del tipo que se pasa como parámetro y False en caso contrario.

>>> type(3)

<**class** 'int'>

>>> type(2.78)

<**class** 'float'>

>>> type('Hola')

<**class** 'str'>

>>> isinstance(3, float)

**False**

>>> isinstance(3, int)

**True**

>>> isinstance(3, bool)

**False**

>>> isinstance(**False**, bool)

**True**

## Conversión de tipos

Lo último que veremos en este tutorial sobre tipos de datos es la conversión de tipos.

¿Esto qué significa?

Imagina que tienes una variable edad de tipo string cuyo valor es '25'. Se podría decir que edad, aunque realmente es una cadena de caracteres, contiene un número. Sin embargo, si intentas sumar 10 a edad, el intérprete te dará un error porque edad es de tipo str y 10 un tipo numérico.

>>> edad = '25'

>>> edad = edad + 10

Traceback (most recent call last):

File "<input>", line 1, **in** <module>

TypeError: can only concatenate str (not "int") to str

¿Cómo puedo tratar la variable edad como un número? Convirtiéndola a un tipo numérico, por ejemplo, al tipo int.

Para ello, Python ofrece las siguientes funciones:

* str(): Devuelve la representación en cadena de caracteres del objeto que se pasa como parámetro.
* int(): Devuelve un int a partir de un número o secuencia de caracteres.
* float(): Devuelve un float a partir de un número o secuencia de caracteres.
* complex(): Devuelve un complex a partir de un número o secuencia de caracteres.

Si a las funciones anteriores se les pasa como parámetro un valor inválido, el intérprete mostrará un error.

>>> edad = int(edad) + 10 # Convierte edad a int

>>> edad # edad es un int

35

>>> edad = str(edad) # Convierte edad a str

>>> edad # edad es un str (se muestran las '')

'35'

>>> float('18.66') # Convierte un str a float

18.66

>>> float('hola') # Convierte un str a float (pero no es válido)

Traceback (most recent call last):

File "<input>", line 1, **in** <module>

ValueError: could not convert string to float: 'hola'