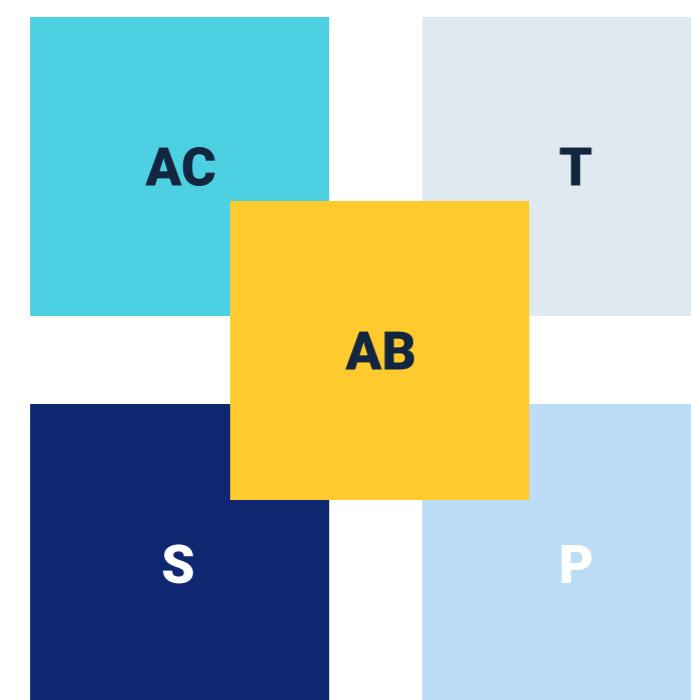


Fundamentos y lógica de programación

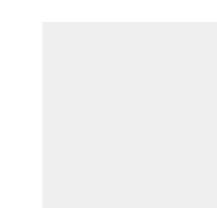
En este componente formativo se presentan los fundamentos de la programación como base para iniciar en el universo de la analítica de datos, se abordan también temas como la codificación de la lógica de programación, empleando herramientas de vanguardia y metodologías que permiten crear programas con las mejores prácticas, garantizando un código reutilizable, escalable, seguro y de fácil mantenimiento.

[Iniciar >](#)

PRIMARIO
#138AF8



SECUNDARIO
#1B3F5E



NEUTRAL 1
#EFEFEF



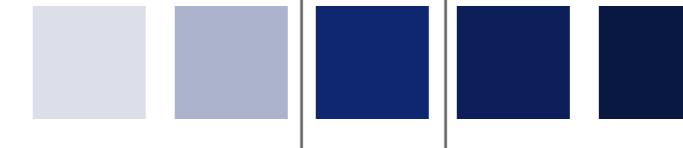
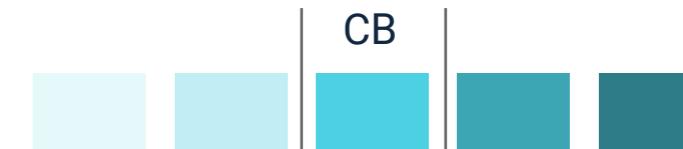
ACENTO CONTENIDO
#5ED1A9



ACENTO BOTONES
#FFD947



NEUTRAL 2
#F9F7EC



i Introducción

Apreciado aprendiz, bienvenido a este componente formativo, donde se abordarán los principios básicos de programación, los fundamentos de la lógica de programación y los paradigmas y estándares de programación. Podrá desarrollar desde programas sencillos hasta sofisticados códigos; posteriormente, aprenderá a manejar estructuras concretas: variables, tipos de datos, instrucciones de decisión, de repetición, entre otras. Finalmente, aprenderá a programar en un entorno de desarrollo en línea Google Colab que permite ejecutar código escrito en lenguaje Python, herramienta esencial para crear soluciones de software que le servirán de insumo, para adentrarse en el campo de la analítica de datos.

En el siguiente video conocerá, de forma general, la temática que se estudiará a lo largo del componente formativo.

¡Le deseamos una experiencia de aprendizaje significativa y memorable!

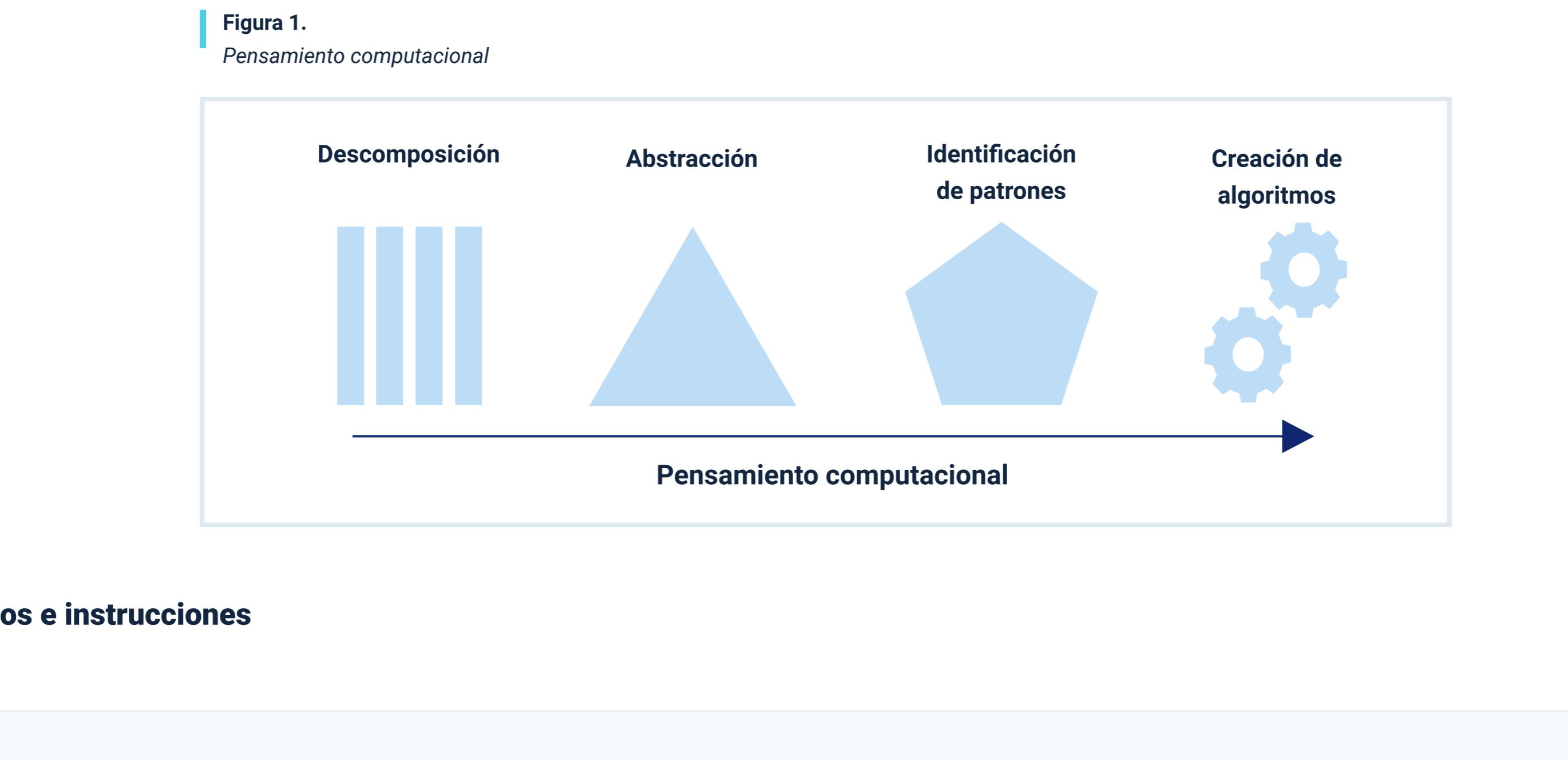


VIDEO

1 Fundamentos y lógica de Programación

Los fundamentos y la lógica de programación es la parte inicial del recorrido, es aquí, donde se adoptarán los conceptos clave y la lógica que apoyará el resto del camino, de modo que, se pueda resolver un problema complejo utilizando técnicas que permitan convertirlos en unidades más pequeñas y fáciles de solucionar.

Pensamiento computacional



El pensamiento computacional fomenta las habilidades que tienen las personas para resolver problemas, ya que, este tipo de pensamiento implica descomponer un problema en unidades más pequeñas e identificar los patrones y características comunes de estas partes, para luego enfocarse en la información relevante y finalmente, solucionar el problema paso a paso a través de algoritmos.

Figura 1.
Pensamiento computacional



Algoritmos e instrucciones

Un algoritmo en términos generales es una secuencia de pasos que se deben ejecutar en un orden predefinido, con el fin de realizar una tarea específica o resolver un problema en particular. Un claro ejemplo los algoritmos, son los manuales, porque en ellos encontramos los pasos para resolver los problemas, como los pasos para armar un artefacto o qué hacer en caso de que se presente algún inconveniente o falla con un dispositivo.

Diagramas de flujo

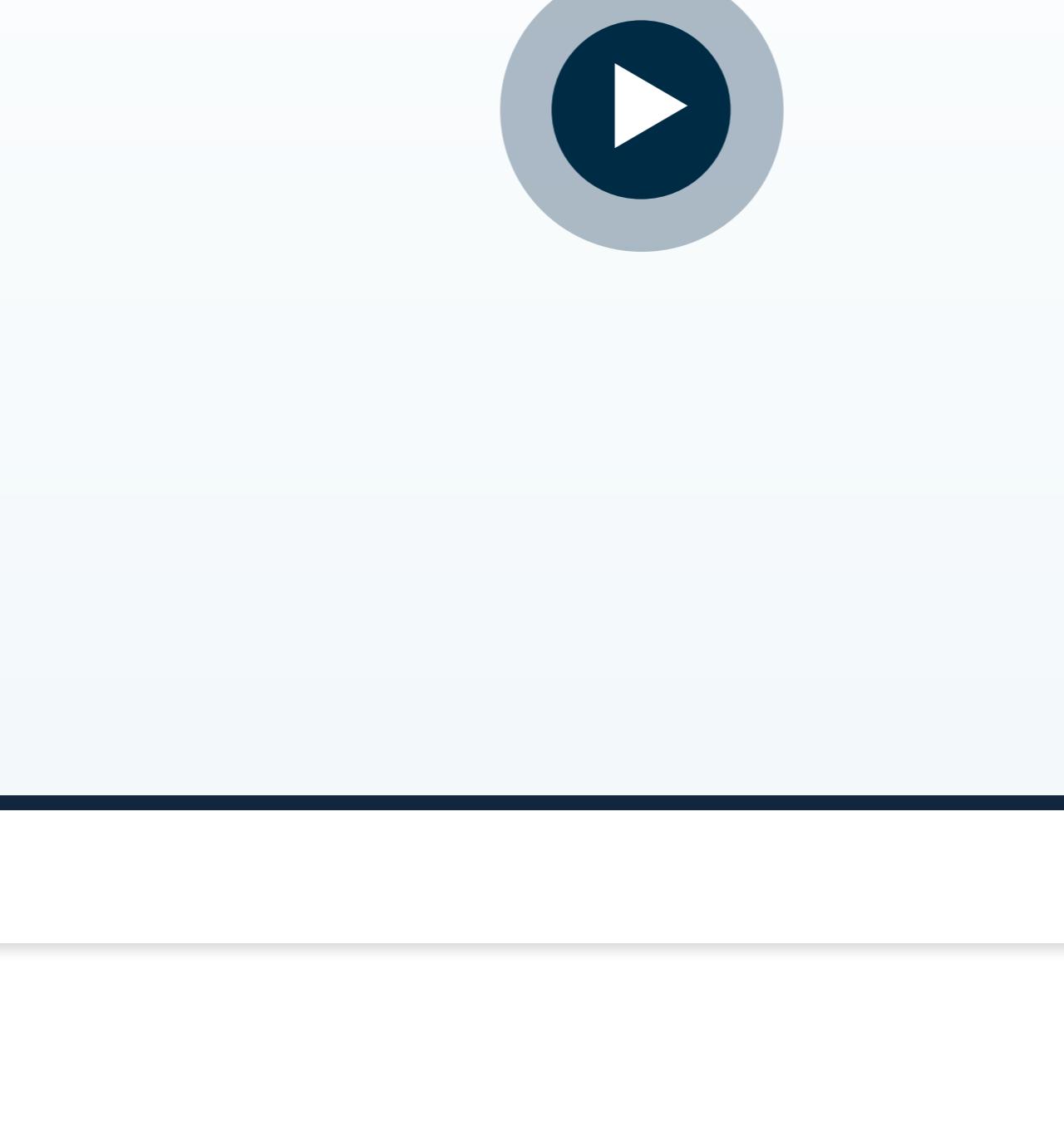
Los diagramas de flujo son representaciones gráficas de los algoritmos para representar de manera clara, puntual y organizada la información que se desea tratar. También son llamados fluojogramas.

Los fluojogramas cuentan con una simbología establecida por el Instituto Nacional Estadounidense de Estándares (ANSI). A continuación, se relacionan los símbolos más usados:



La siguiente figura muestra un ejemplo de diagrama de flujo para determinar si un número es positivo o negativo:

Figura 2
Ejemplo de diagrama de flujo



El uso de fluojogramas puede acarrear algunos beneficios pero, al mismo tiempo, algunas desventajas:

Ventajas de los diagramas de flujo:

- > Ayudan a comprender la lógica de la programación.
- > Contribuyen en la detección de errores.
- > Facilitan la identificación de procesos.
- > Son útiles para mostrar al cliente las fases o transacciones que realiza el sistema.

Desventajas de los diagramas de flujo:

- > Son difíciles para realizar cambios en su estructura o diseño.
- > No son recomendables para la lógica de programas complejos.
- > Pueden retrasar el desarrollo del software.

Existen herramientas como **PSeInt** que asisten a los estudiantes para dar sus primeros pasos en programación. **PSeInt** cuenta con la opción de crear fluojogramas.

A continuación, encontrará un video que muestra la instalación de esta herramienta en un sistema operativo Windows y la creación de un diagrama de flujo, como ejemplo:

VIDEO

Pseudocódigo

El pseudocódigo es un lenguaje que se emplea para documentar los programas paso a paso, de la forma más detalladamente posible. No existe ninguna sintaxis para escribir un pseudocódigo, por lo que se encuentran muchas maneras según la forma particular de escribir del autor.

Un pseudocódigo no tiene reglas predefinidas y no se utiliza para programar, sino que se emplea para familiarizarse y acercarse a los lenguajes de programación (Trejos, 2021).

Preste atención a lo expuesto en el siguiente esquema:

Figura 3
Algoritmo área del rectángulo

Algoritmo área_rectángulo

Inicio

Escribir "Cálculo del área del rectángulo"

Escribir "Digite la base (en centímetros)"

Ler base

Escribir "Digite la altura (en centímetros)"

Ler altura

altura ← base * altura

Escribir "Área es:", base, "centímetros cuadrados"

Fin

Es importante tener mayor claridad sobre el concepto **pseudocódigo**; para lograrlo, analice con atención el ejemplo que se expone en la siguiente tabla:

Tabla 2
Ejemplo de algoritmo, versus representación en pseudocódigo

Algoritmo	Pseudocódigo
Paso 1. Solicitar los datos de un usuario (nombre, valor por hora, número de horas laboradas).	Inicio
Paso 2. Condicionado al número total de horas trabajadas en la semana:	Defina Nombre= ' ', ValorHr = 0 Defina Hrs_Lab = 0, Salario = 0 Presentar 'Digite los datos del usuario' Capturar Nombre, ValorHr, Hrs_Lab Si Hrs_Lab <=40
a Si el número de horas no es mayor a 40, hacer el cálculo sin horas extras. b Si el número de horas es mayor a 40, realizar el cálculo del sueldo, teniendo en cuenta que, la hora extra se cancela con 1.5% más por encima del valor de la hora normal.	Entonces Salario= Hrs_Lab * ValorHr Sino Salario=40 * ValorHr + (Hrs_Lab - 40)*(1.5*ValorHr) FinSi Presentar 'El salario de', Nombre, 'es', Salario
Paso 3. Presentar el nombre del usuario y el respectivo salario que va a recibir.	Fin

Preste atención a lo expuesto en el siguiente esquema:

Figura 3
Algoritmo área del rectángulo

Algoritmo área_rectángulo

Inicio

Escribir "Cálculo del área del rectángulo"

Escribir "Digite la base (en centímetros)"

Ler base

Escribir "Digite la altura (en centímetros)"

Ler altura

altura ← base * altura

Escribir "Área es:", base, "centímetros cuadrados"

Fin

Es importante tener mayor claridad sobre el concepto **pseudocódigo**; para lograrlo, analice con atención el ejemplo que se expone en la siguiente tabla:

Tabla 2
Ejemplo de algoritmo, versus representación en pseudocódigo

Algoritmo	Pseudocódigo
Paso 1. Solicitar los datos de un usuario (nombre, valor por hora, número de horas laboradas).	Inicio
Paso 2. Condicionado al número total de horas trabajadas en la semana:	Defina Nombre= ' ', ValorHr = 0 Defina Hrs_Lab = 0, Salario = 0 Presentar 'Digite los datos del usuario' Capturar Nombre, ValorHr, Hrs_Lab Si Hrs_Lab <=40
a Si el número de horas no es mayor a 40, hacer el cálculo sin horas extras. b Si el número de horas es mayor a 40, realizar el cálculo del sueldo, teniendo en cuenta que, la hora extra se cancela con 1.5% más por encima del valor de la hora normal.	Entonces Salario= Hrs_Lab * ValorHr Sino Salario=40 * ValorHr + (Hrs_Lab - 40)*(1.5*ValorHr) FinSi Presentar 'El salario de', Nombre, 'es', Salario
Paso 3. Presentar el nombre del usuario y el respectivo salario que va a recibir.	Fin

2 Entorno de desarrollo

Los entornos de desarrollo o entornos de desarrollo integrado, también llamados IDE (por sus siglas en inglés *Integrated Development Environment*), son aplicaciones informáticas especializadas que emplean los programadores para escribir el código fuente en cualquiera de los lenguajes de programación existentes como Python, PHP, Javascript, Java, C, C++, C#, GO, Ruby entre otros.

Este tipo de aplicaciones ofrecen herramientas como la depuración de errores y automatización de compilaciones locales, que le proporcionan al programador o desarrollador de software, ser más eficientes en la codificación de los programas.

¿Por qué los programadores utilizan los entornos de desarrollo integrado (IDE)?

Los IDE permiten que los programadores inicien a desarrollar nuevas aplicaciones con mayor rapidez, debido a que, con ellos no se requiere preparar ni cambiar de forma manual las herramientas que hacen parte de los procesos de configuración. También aportan en la optimización del tiempo con el auto relleno inteligente de código, lo cual suprime la necesidad de escribir secuencias enteras de caracteres.

Otra característica fundamental de los IDE es el análisis de código en tiempo real, esto permite que, mientras se está codificando, se puedan visualizar los errores en las líneas de código y hasta las posibles correcciones.

Es importante destacar que es posible crear programas sin utilizar ningún IDE o que los programadores diseñen un IDE personalizado que integre varias herramientas de forma manual; como los editores de texto Notepad+, VIM, Sublime Text, entre otros. Sin embargo, en los entornos empresariales, es altamente beneficioso minimizar tiempos con la estandarización de un IDE, a invertirlo en las opciones manuales, por lo que la mayor parte de las organizaciones prefieren los IDE preconfigurados que se adaptan a sus casos de usos particulares.

Tipos de IDE

Existen opciones tanto comerciales como open source en el mercado que suplen la demanda existente. Entre las características que distinguen los IDE resaltan las siguientes:

Cantidad de lenguajes soportados:

- Sistemas operativos soportados:
- Performance:
- Extensiones y Plugins:

Cantidad de lenguajes soportados: existen IDE que solamente son compatibles con un lenguaje de programación, por lo tanto, son mejores para un estilo de programación en especial, entre ellos, tenemos como ejemplo a IntelliJ, que se conoce como un IDE exclusivo para Java. Por otra parte, están los IDE que soportan varios lenguajes de programación, como es el caso de Eclipse, con el que se puede programar en Java, Python, JavaScript, PHP, C++, entre otros.

Lenguajes compilados y lenguajes interpretados

Cuando se inicia en el mundo del desarrollo de software, la selección del lenguaje de programación es uno de los factores que más se tiene en cuenta para dar los primeros pasos. También es fundamental conocer si el lenguaje elegido es compilado o interpretado.

Lenguaje compilado

Este tipo de lenguaje de programación traduce el código fuente en sentencias que la máquina puede entender. Estas sentencias son agrupadas en un archivo binario ejecutable, que, por lo general, está limitado a un determinado sistema, permitiendo que se puedan ejecutar sin la instalación de otro programa complementario, ya que todo lo que necesita para funcionar correctamente se encuentra embebido en ese único archivo ejecutable. Entre los lenguajes compilados está C, C++, Swift, Go, Ada, entre otros.

Lenguaje interpretado

Este tipo de lenguaje de programación ejecuta el código fuente directamente en la máquina sin necesidad de traducirlo a un formato ejecutable. Los lenguajes interpretados incluyen Python, JavaScript, PHP, entre otros. La ejecución es más lenta que la de los lenguajes compilados, pero ofrece una mayor flexibilidad y portabilidad.

Lenguajes compilados versus interpretados

La tabla que se observa enseguida, relaciona algunas diferencias o contraposiciones entre lenguajes compilados y lenguajes interpretados:

Tabla 2
Lenguajes compilados versus interpretados

Lenguajes compilados	Lenguajes interpretados
Para que la aplicación se ejecute, primero se debe terminar de codificar el programa.	Se ejecuta instrucción a instrucción, por lo tanto, no se requiere terminar de codificar el programa para que funcione.
La compilación se realiza una sola vez y solamente será necesario volver a compilar cuando se requieran cambios en el código fuente.	Cada vez que se ejecute la aplicación se debe interpretar el código fuente.
Se emplean mayormente en aplicaciones de escritorio.	Se utilizan principalmente en aplicaciones web.
Se usan cuando la prioridad es la eficiencia.	Se aplican cuando la prioridad es la portabilidad, en otras palabras, cuando la preocupación es que la aplicación pueda ejecutarse sobre cualquier plataforma.
La aplicación obtenida con estos lenguajes es más eficiente y rápida.	La aplicación tiende a ser un poco más lenta que las aplicaciones compiladas.
Los archivos ejecutables por lo general tienen un tamaño considerable en disco.	Los ejecutables generados son de un reducido tamaño y no consumen mucho espacio en disco.

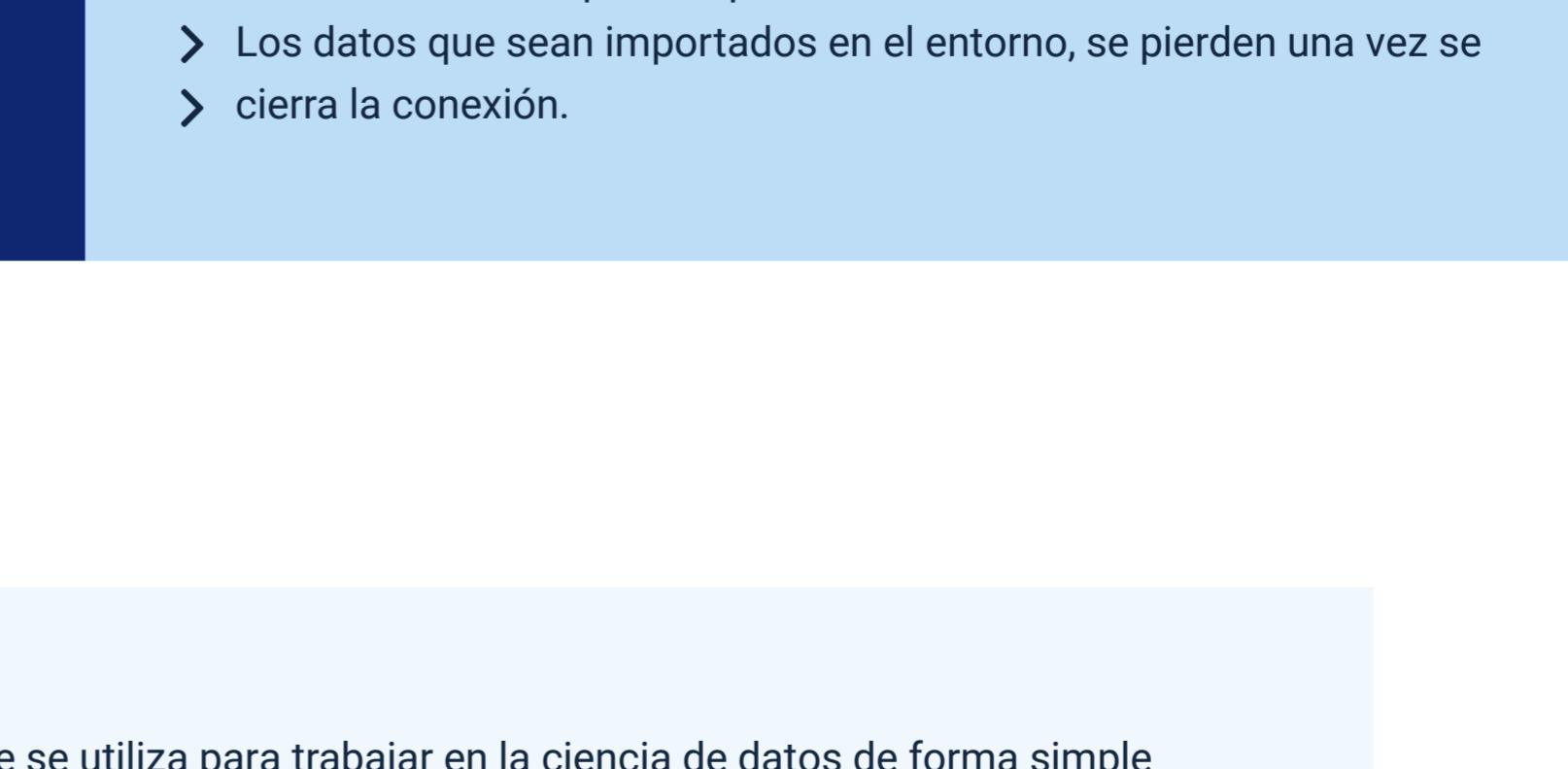
Python y/o R

La analítica de datos es un campo de trabajo emocionante que combina habilidades estadísticas y cuantitativas avanzadas, con habilidades de programación del mundo real. Los aspirantes a científicos de datos pueden considerar especializarse en muchos lenguajes de programación. En este componente formativo, el enfoque estará en el uso de Python.

La habilidad de programar es fundamental en la gestión y analítica de datos; no obstante, hay algunos lenguajes como Python, R, MATLAB y SQL que se destacan como los principales y más usados en dicha gestión.

Ventajas de programar en Python

- Rápido de aprender: su curva de aprendizaje es corta.
- Potente y multifuncional: se utiliza prácticamente en todas las ramas de la tecnología.
- Simple y práctico: para crear aplicaciones se necesitan menos líneas de código que en otros lenguajes como Java, C, C++, entre otros.
- Amplia documentación: Python cuenta con una enorme comunidad que soporta el lenguaje y es de gran utilidad cuando se necesita apoyo para resolver cualquier duda en su manejo.
- colección de bibliotecas: las bibliotecas también conocidas como librerías, son importantes porque permiten la reutilización de código y aumentan la rapidez durante el desarrollo.



Para conocer el proceso de instalación de Python y las herramientas necesarias para programar, visualice con suma atención el siguiente video:

VIDEO

Analice la infografía que se propone a continuación y descubra los cinco campos de aplicación de Python:

Descargar

Google Colab

Colab, también conocido como Colaboratory, permite programar y ejecutar Python en el navegador Web; no requiere configuraciones y tolera contenido con facilidad.

A continuación, le invitamos a conocer esta herramienta y algunas ventajas y desventajas que puede traer su uso:

¿Qué es Google colab?

Colab es un entorno colaborativo que posibilita la ejecución del lenguaje de programación Python sin necesidad de instalar ningún componente de forma local, ya que, es netamente implementado en la nube de Google y no requiere configuración previa para iniciar a programar.

Jupyter es un entorno de desarrollo basado en la web, que se utiliza para trabajar en la ciencia de datos de forma simple y enfocada en documentos. Al igual que Google Colab, Jupyter permite, tanto código como celdas de texto que incorporan el formato markdown (lenguaje de marcado sencillo que se utiliza para agregar formato a los textos dentro de un documento).

Sus ventajas y desventajas son:

Ventajas de Google Colab:

- Librerías preinstaladas.
- Acceso gratuito a GPUs (unidad de procesamiento gráfico).
- Almacenamiento en la nube.
- Permite la colaboración.
- No necesita configuración para ser utilizado.

Desventajas de Google Colab:

- No se puede ejecutar sin conexión a Internet.
- Limitado para trabajar con volúmenes de datos que sean demasiado grandes.
- Limitado para trabajar con volúmenes de datos que requieren recursos de hardware altos para su procesamiento.
- Los datos que sean importados en el entorno, se pierden una vez se cierra la conexión.

Conozcamos ahora sobre el entorno de desarrollo Jupyter.

Jupyter es un entorno de desarrollo basado en la web, que se utiliza para trabajar en la ciencia de datos de forma simple y enfocada en documentos. Al igual que Google Colab, Jupyter permite, tanto código como celdas de texto que incorporan el formato markdown (lenguaje de marcado sencillo que se utiliza para agregar formato a los textos dentro de un documento).

Ventajas de Jupyter:

- Es compatible con los principales lenguajes de programación empleados en la gestión y analítica de datos como Python y R.
- Se puede utilizar sin conexión a Internet.
- Es open source y gratuito.
- Proporciona opciones de colaboración con JupyterHub.
- Permite el control de versiones en sus Notebooks.

Desventajas de Jupyter:

- Es tedioso en el proceso de depuración.
- No contiene todas las herramientas que posee un IDE.
- Induce a crear código complejo e incomprensible.

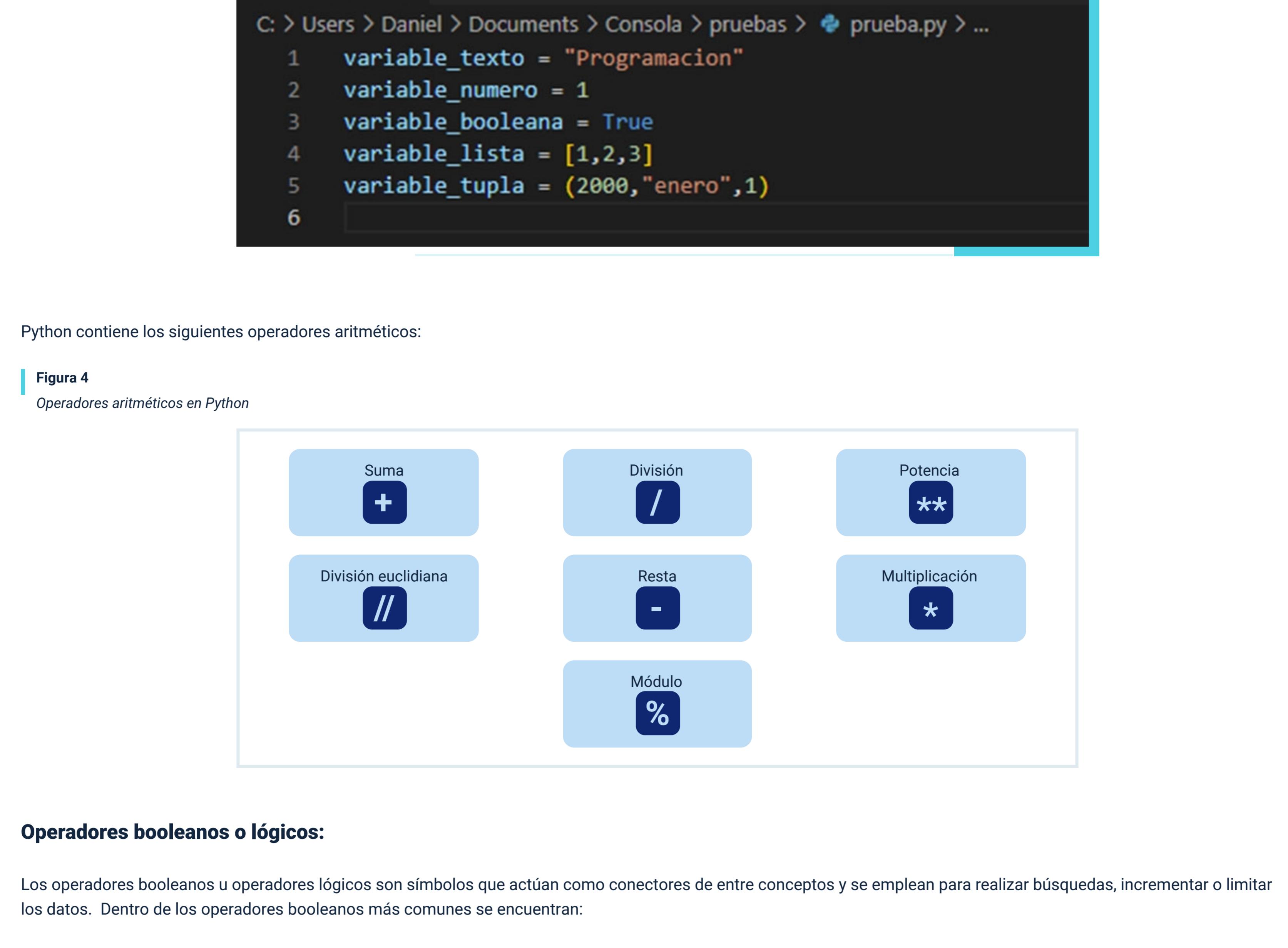
3 Principios básicos de programación

Programar consiste en escribir detalladamente órdenes que una máquina debe seguir para completar una tarea. Estas órdenes se escriben en un lenguaje que la máquina puede entender de forma directa o a través de un intérprete o compilador.

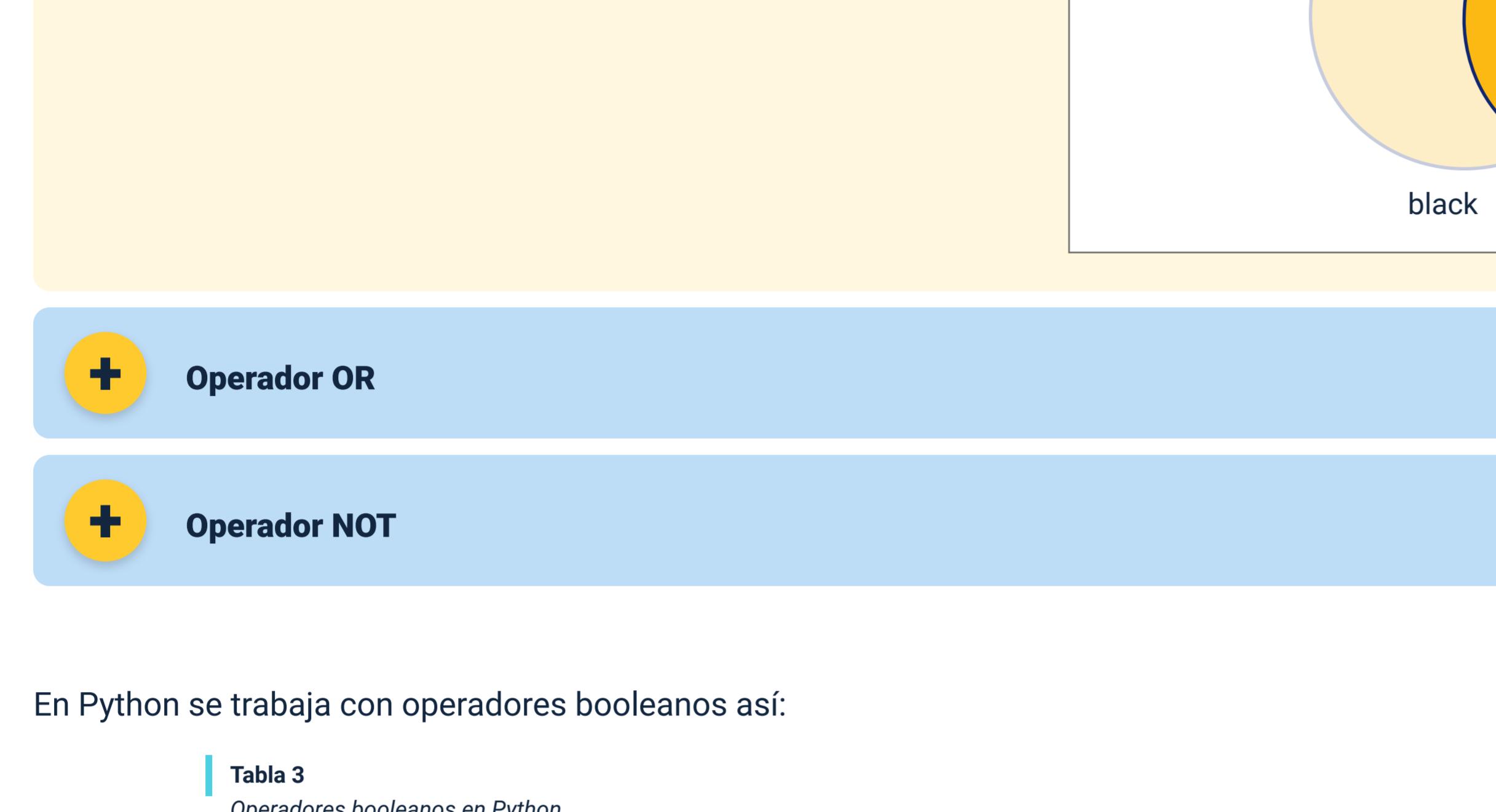
En esta unidad se estudiarán las herramientas más comunes de las que disponen todos los lenguajes de programación.

Tipos de variables

Una variable es un espacio reservado en la memoria del computador donde se guardan datos u objetos como números, imágenes, texto, entre otros.



Variables y operadores booleanos



```
prueba.py
C: > Users > Daniel > Documents > Consola > pruebas > prueba.py > ...
1 variable_texto = "Programacion"
2 variable_numero = 1
3 variable_boleana = True
4 variable_lista = [1,2,3]
5 variable_tupla = (2008,"enero",1)
6
```

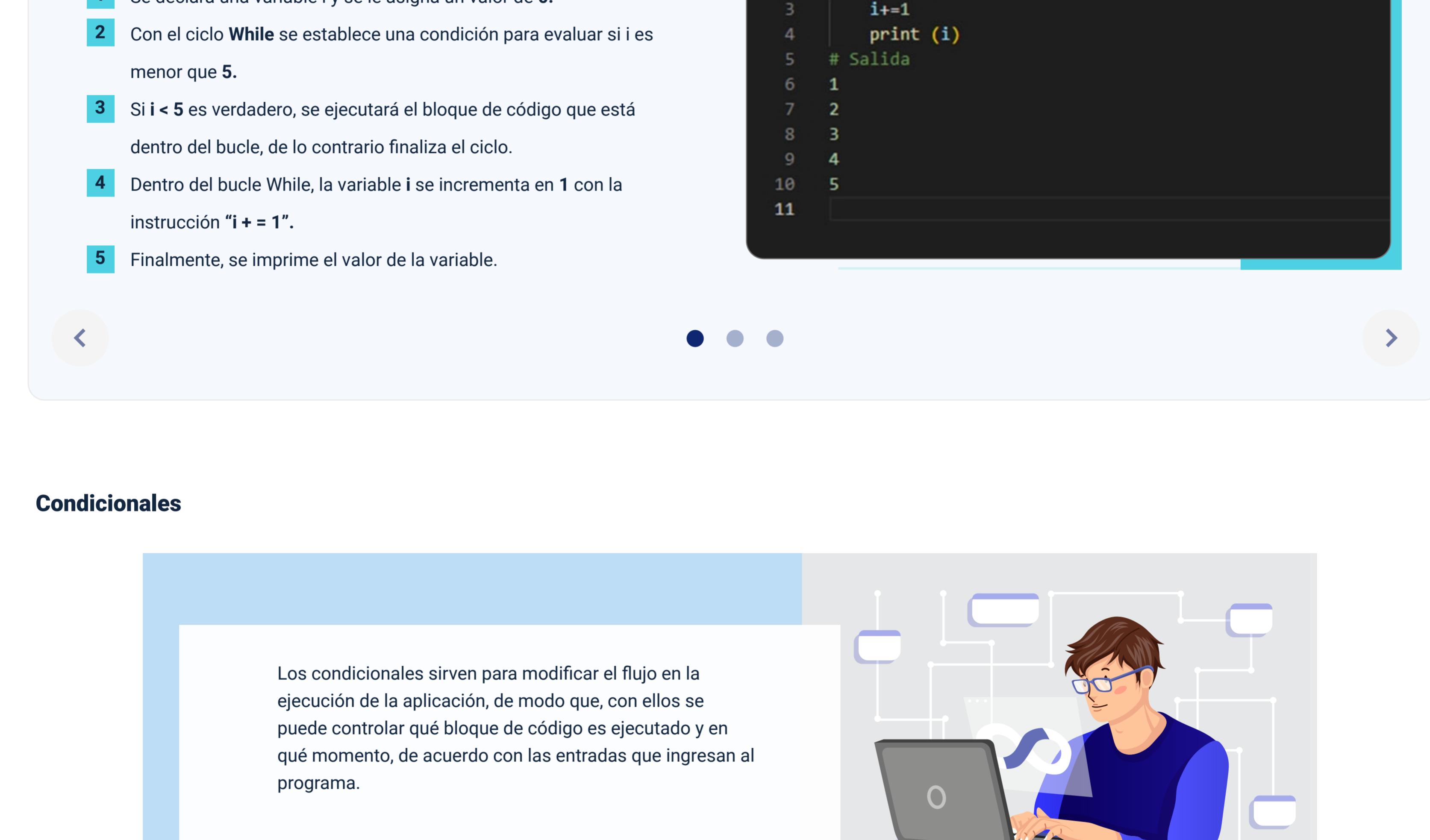
Python contiene los siguientes operadores aritméticos:

Figura 4
Operadores aritméticos en Python



Operadores booleanos o lógicos:

Los operadores booleanos o operadores lógicos son símbolos que actúan como conectores de entre conceptos y se emplean para realizar búsquedas, incrementar o limitar los datos. Dentro de los operadores booleanos más comunes se encuentran:



En Python se trabaja con operadores booleanos así:

Figura 3
Operadores booleanos en Python

```
prueba.py
C: > Users > Daniel > Documents > Consola > pruebas > prueba.py > ...
1 print(True and True) # True
2 print(True or False) # True
3 print(False and True) # False
4 print(False or False) # False
5
```

Figura 6
Uso de operador OR en Python

```
prueba.py
C: > Users > Daniel > Documents > Consola > pruebas > prueba.py > ...
1 print(True or True) # True
2 print(True or False) # True
3 print(False or True) # True
4 print(False or False) # False
5
```

Figura 7
Uso del operador NOT en Python

```
prueba.py
C: > Users > Daniel > Documents > Consola > pruebas > prueba.py > ...
1 print(not True) # False
2 print(not False) # True
3 print(not not not not True) # True
4
```

Ciclos (Do While, While y For)

En programación, los ciclos o bucles son secuencias de instrucciones que se ejecutan varias veces, hasta cumplir con una condición, o que la condición inicial deje de cumplirse.

Cabe mencionar que existen dos tipos de ciclos:

- ✓ Definidos: son los que tienen una cantidad de iteraciones conocidas.
- ✓ No definidos: son los que tienen un número de iteraciones desconocidas.

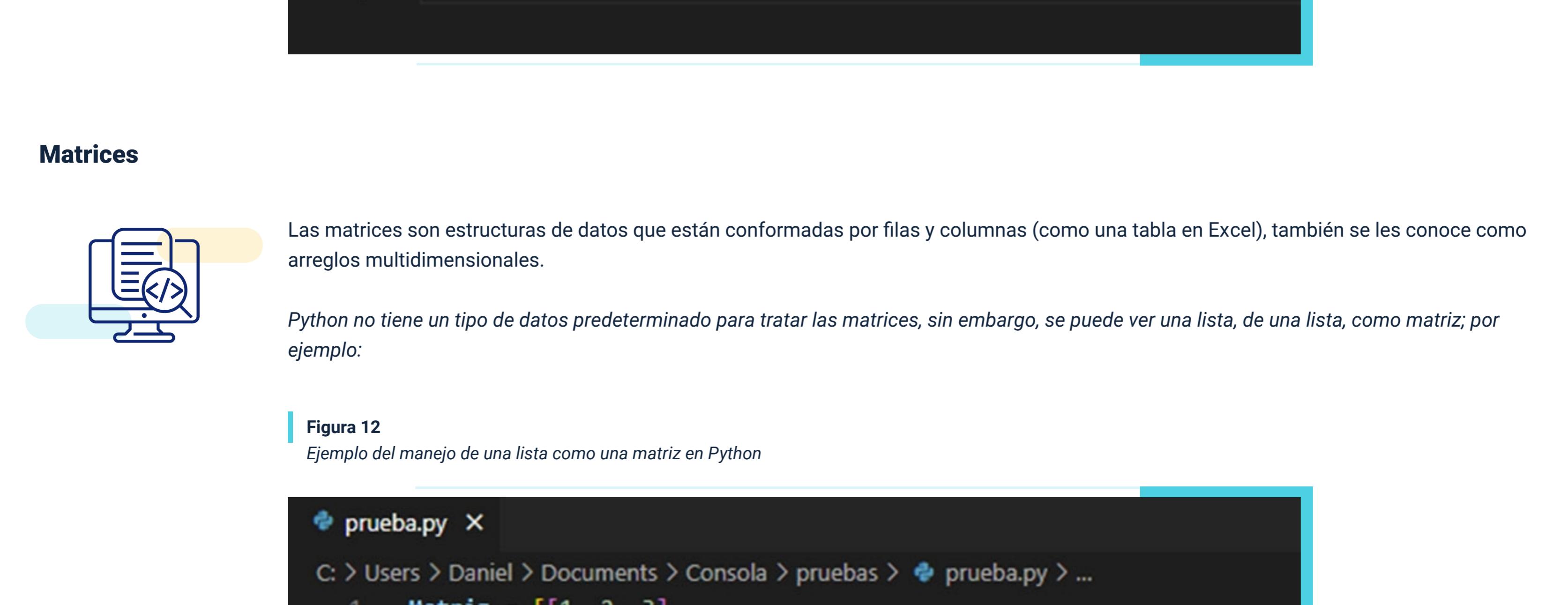
En el ejemplo de la imagen se observa que:

- 1 Se declara una variable i y se le asigna un valor de 0.
- 2 Con el ciclo While se establece una condición para evaluar si i es menor que 5.
- 3 Si i < 5 es verdadero, se ejecutará el bloque de código que está dentro del bucle, de lo contrario finalizará el ciclo.
- 4 Dentro del bucle While, la variable i se incrementa en 1 con la instrucción "i+=1".
- 5 Finalmente, se imprime el valor de la variable.

Ejemplo de ciclo While en el que se imprimen los números del 1 al 5.

```
prueba.py
C: > Users > Daniel > Documents > Consola > pruebas > prueba.py > ...
1 i=0
2 while i<5:
3     i+=1
4     print(i)
5 # Salida
6
7 1
8 2
9 3
10 4
11 5
```

Condicionales



Código Python para realizar la división de dos números.

```
prueba.py
C: > Users > Daniel > Documents > Consola > pruebas > prueba.py > ...
1 n = 10
2 d = 2
3 if n != 0:
4     print(n/d)
5     # Salida
6     # 5.0
```

Arreglos

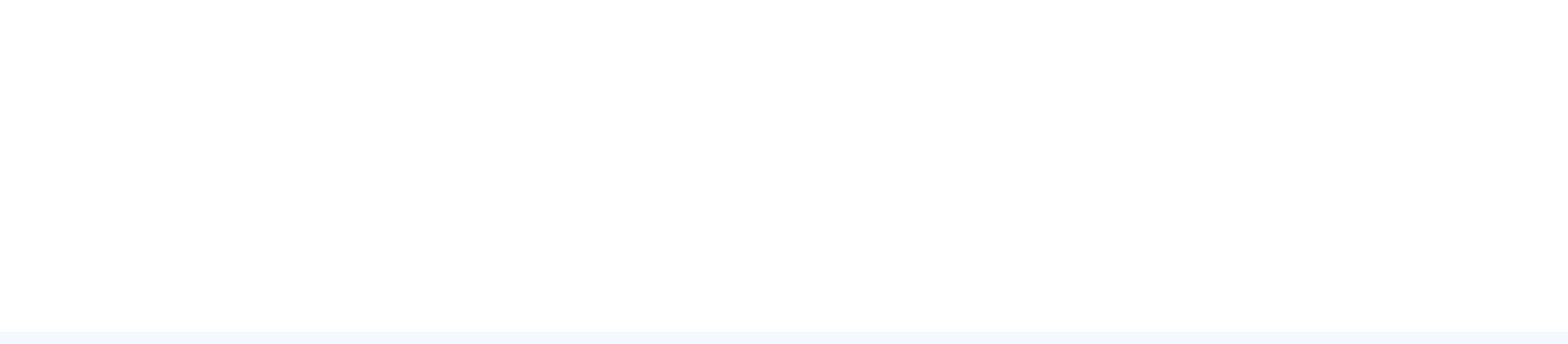


Figura 8
Ejemplo de un arreglo en Python

```
prueba.py
C: > Users > Daniel > Documents > Consola > pruebas > prueba.py > ...
1 arreglo = [1,2,3,4,5]
2 print(arreglo[1])
3
4 # Salida:
5 # 2
```

En el ejemplo de la imagen, el resultado sería imprimir el dato que se encuentra en la posición 1 del arreglo, el cual es el número 2, ya que, las posiciones empiezan con el índice 0.

Listas y tuplas

Las listas, a diferencia de los arreglos, permiten guardar datos de cualquier tipo y pueden ser dinámicas o variables, para proporcionar flexibilidad en la manipulación de la información.

La siguiente lista está conformada por un número entero, una cadena de caracteres y un número flotante:

Figura 9
Ejemplo de una lista en Python

```
prueba.py
C: > Users > Daniel > Documents > Consola > pruebas > prueba.py > ...
1 lista = [100, "Programacion", 2.5]
2 print(lista[0]) #100
3 print(lista[1]) #programacion
4 print(lista[2]) #2.5
5 print(lista[-1]) #2.5, imprime la ultima posicion de la lista
```

Las tuplas, a diferencia de las listas, no son dinámicas. Esto quiere decir, que no permiten modificaciones en sus elementos, pero en términos de rendimiento, son más eficientes:

Analice el ejemplo de la siguiente imagen:

Figura 10
Ejemplo de una tupla en Python

```
prueba.py
C: > Users > Daniel > Documents > Consola > pruebas > prueba.py > ...
1 tupla = 10, 20, ('x', 'y'), 30
2 print(tupla) # (10, 20, ('x', 'y'), 30)
3 print(tupla[2][0]) #x
4
```

Funciones

Figura 11
Ejemplo de una función en Python

```
prueba.py
C: > Users > Daniel > Documents > Consola > pruebas > prueba.py > ...
1 def funcion_ejemplo(parametros):
2     a=c+b
3     return # codigo
4
5
```

En Python, para crear una función, se emplea con la palabra reservada `def`.

A continuación, se define de forma general, la estructura de una función:

Figura 12
Ejemplo del manejo de una lista como una matriz en Python

```
prueba.py
C: > Users > Daniel > Documents > Consola > pruebas > prueba.py > ...
1 Matriz = [[1, 2, 3], [4, 5, 6]]
2
3
```

Las matrices son estructuras de datos que están conformadas por filas y columnas (como una tabla en Excel), también se les conoce como arreglos multidimensionales.

Python no tiene un tipo de datos predeterminado para tratar las matrices, sin embargo, se puede ver una lista, de una lista, como matriz; por ejemplo:

Analice el ejemplo de la siguiente imagen:

Figura 13
Ejemplo de una matriz en Python

```
prueba.py
C: > Users > Daniel > Documents > Consola > pruebas > prueba.py > ...
1 Matriz = [[1, 2, 3], [4, 5, 6]]
2
3
```

4 Paradigmas y estándares de programación

Los paradigmas y estándares de programación son los modos o estilos que se utilizan para programar.

Los lenguajes de programación pueden adoptar uno o varios paradigmas. Python es un lenguaje multiparadigma.

Entre los paradigmas de programación más utilizados se encuentran los siguientes:

Paradigma imperativo	Paradigma declarativo	Paradigma orientado a objetos
----------------------	-----------------------	-------------------------------

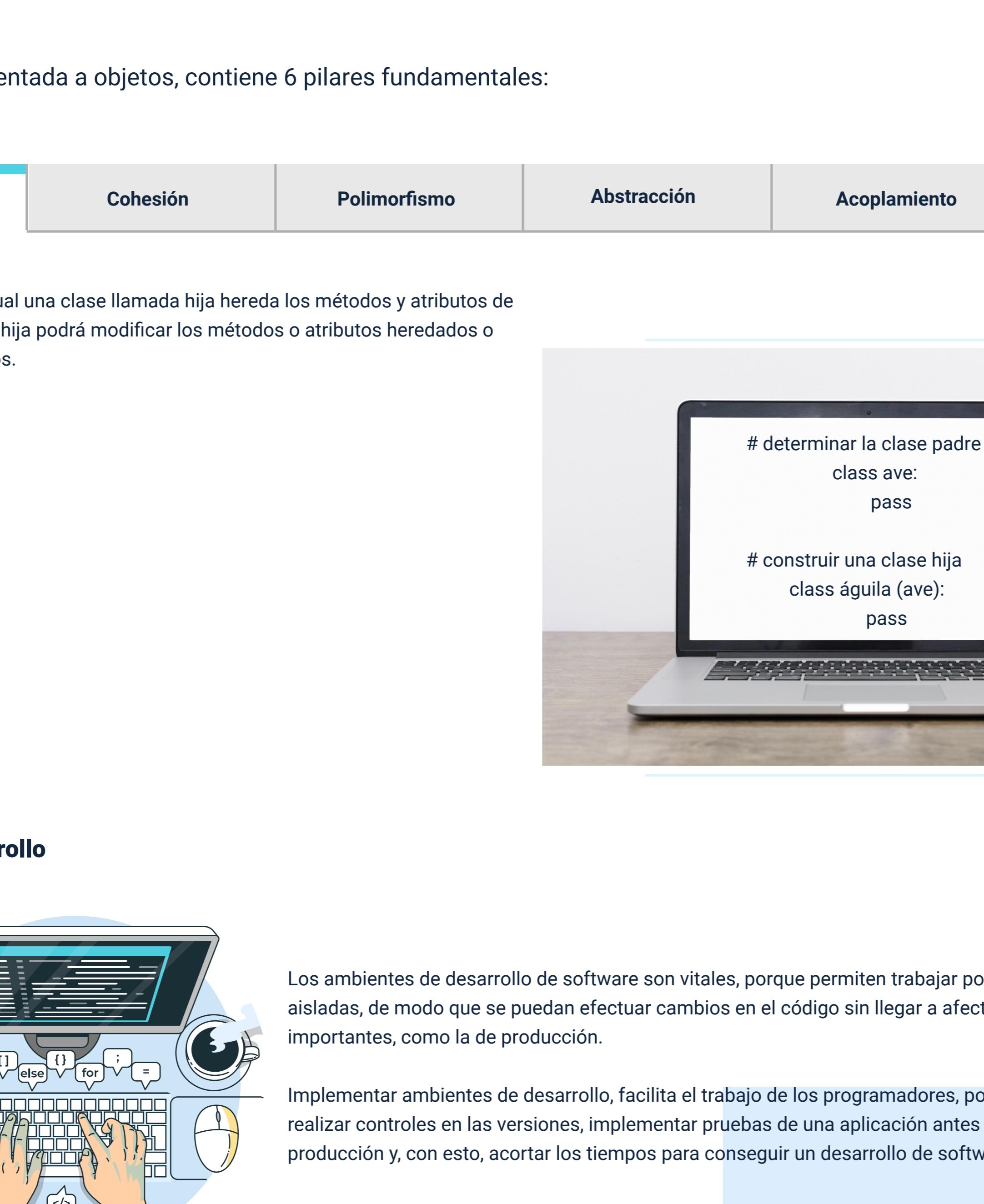
Paradigma imperativo

El desarrollador describe las instrucciones o las órdenes y la máquina las ejecuta paso a paso.

Algunos lenguajes de programación de este paradigma son: Pascal, Python, C, C++, Basic, entre otros.

Programación estructurada

La programación estructurada hace parte del paradigma imperativo y está diseñado para promover la claridad, calidad y el tiempo de desarrollo de las aplicaciones, utilizando las siguientes estructuras:



Programación orientada a objetos

La programación orientada a objetos en POO, por su sigla, es un paradigma de programación que permite organizar el código de formas similar a como se piensa en el mundo real o en lo cotidiano; por ejemplo, artículos como: vehículo, perro, mesa o computador, son representados como **clases** que poseen unas características que, en el caso del vehículo, podrían ser la marca, el modelo, el cilindraje, entre otras.

Estas características son llamadas **atributos**. De igual manera, las clases poseen una colección de funciones, en el caso del vehículo, podría ser trasladar o cargar, las cuales se llaman **métodos**.

Por último, está el concepto de **objeto** que identifica a cada elemento de una clase, es decir vehículo es la clase, pero una camioneta, un bus, una moto, sería un objeto de la clase vehículo.



La programación orientada a objetos, contiene 6 pilares fundamentales:

Herencia	Cohesión	Polimorfismo	Abstracción	Acoplamiento	Encapsulamiento
----------	----------	--------------	-------------	--------------	-----------------

Es el mecanismo por el cual una clase llamada hija hereda los métodos y atributos de una clase padre. La clase hija podrá modificar los métodos o atributos heredados o también crear unos nuevos.



Ambientes de desarrollo

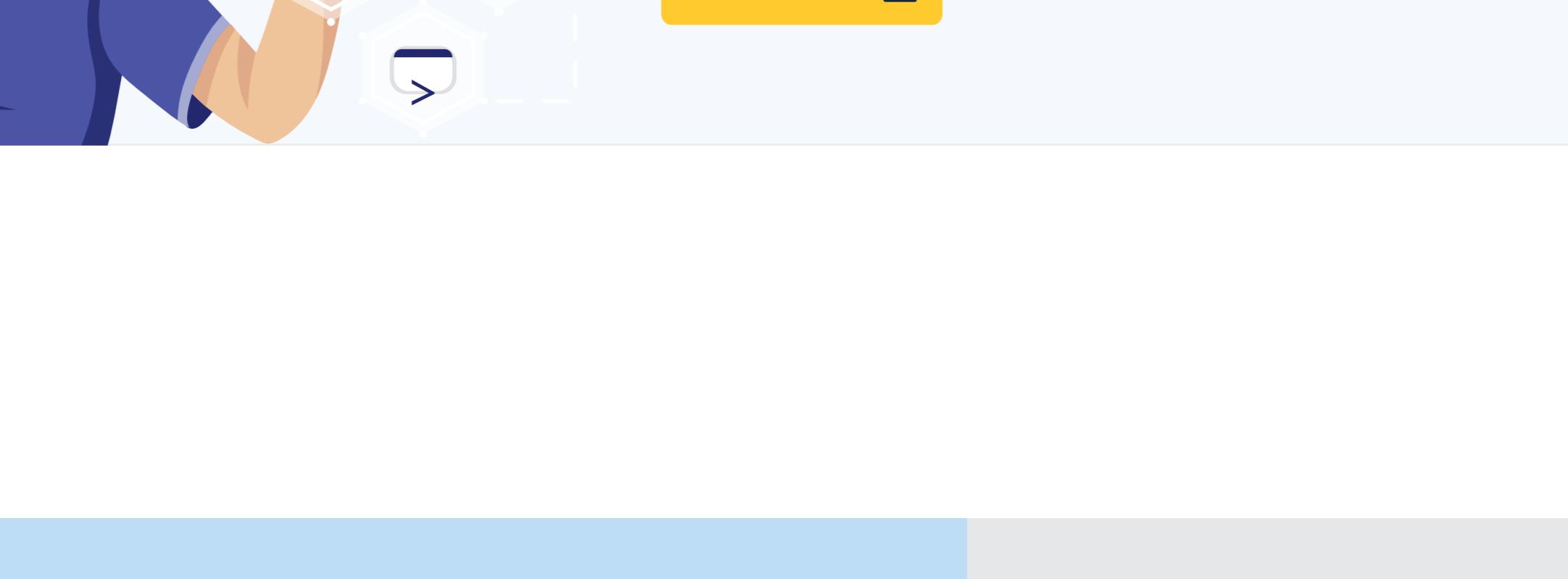


Los ambientes de desarrollo de software son vitales, porque permiten trabajar por capas aisladas, de modo que se puedan efectuar cambios en el código sin llegar a afectar las capas importantes, como la de producción.

Implementar ambientes de desarrollo, facilita el trabajo de los programadores, porque permite realizar controles en las versiones, implementar pruebas de una aplicación antes de salir a producción y, con esto, acortar los tiempos para conseguir un desarrollo de software ágil.

Los ambientes de desarrollo de software más utilizados son:

Figura 14
Ambientes de desarrollo de software



Buenas prácticas de programación

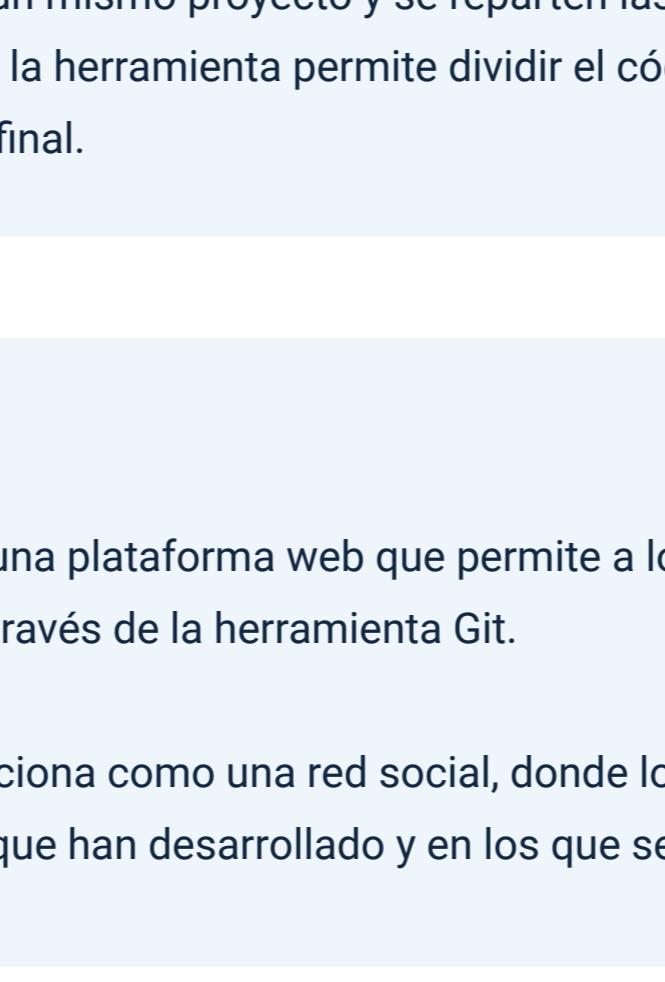
Para crear software práctico, eficiente, seguro, fácil de mantener y desplegar, se debe implementar una serie de principios, estándares y técnicas que conforman las buenas prácticas de programación.

A continuación, se presenta un conjunto de buenas prácticas de programación:



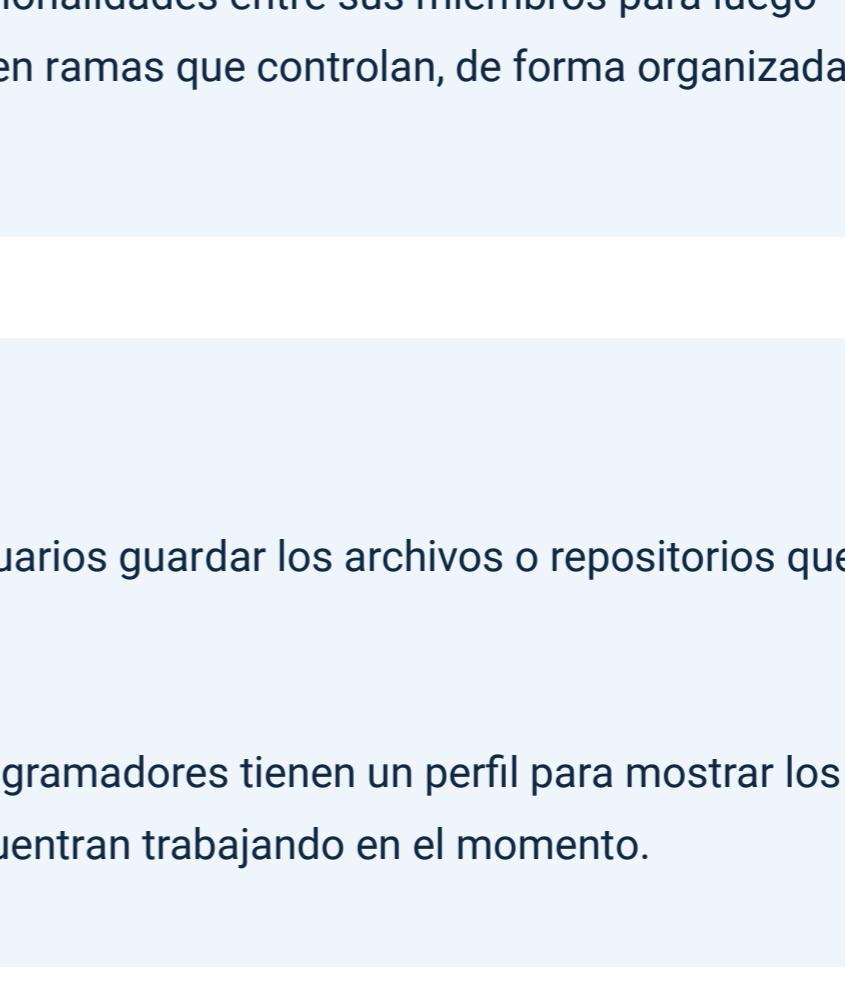
Priorizar la legibilidad:

Es fundamental que los desarrolladores puedan escribir un código que sea entendible para cualquiera, ya que un código complejo, es difícil de mantener y por lo tanto gasta más recursos y tiempo.



Documentar el código:

se deben insertar comentarios para evitar confusiones y facilitar el trabajo a terceros que accedan, de forma colaborativa, al proyecto de software.



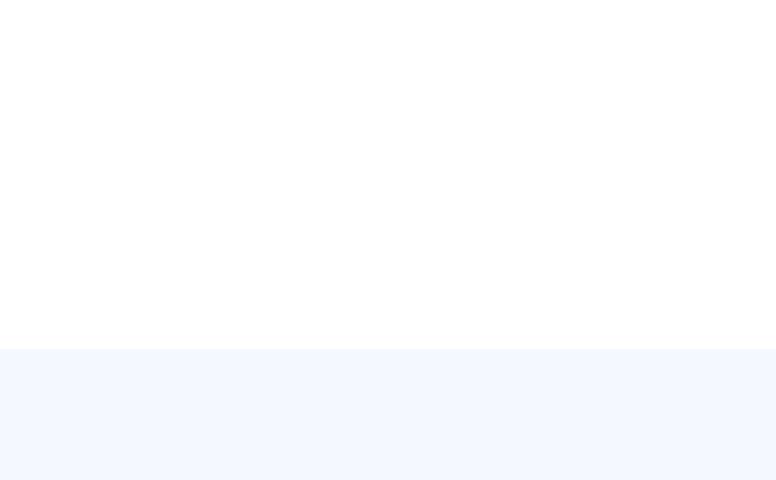
Realizar pruebas de código:

es necesario que en el proceso de desarrollo, se planifiquen pruebas periódicas para encontrar errores a tiempo y poder corregirlos, antes que pasen a la fase de producción.



Git y GitHub

Versionar el código fuente es sin duda una de las mejores prácticas que los equipos de desarrolladores utilizan para registrar los responsables de los cambios en las líneas de código cuando se trabaja en proyectos colaborativos.



Git:

Git es una herramienta de control de versiones muy potente, que permite a los desarrolladores registrar, históricamente, todos los cambios que se realizan en los archivos que componen los proyectos de desarrollo.

Git es muy práctico para trabajar colaborativamente en equipos de desarrollo, porque, cuando se trabaja en un mismo proyecto y se reparten las funcionalidades entre sus miembros para luego integrarlas, la herramienta permite dividir el código en ramas que controlan, de forma organizada, la aplicación final.

GitHub:

GitHub es una plataforma web que permite a los usuarios guardar los archivos o repositorios que generan a través de la herramienta Git.

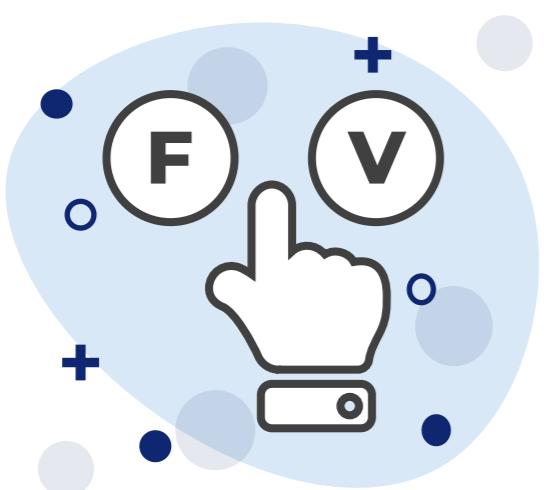
GitHub funciona como una red social, donde los programadores tienen un perfil para mostrar los proyectos que han desarrollado y en los que se encuentran trabajando en el momento.

Recuerde explorar los demás recursos que se encuentran disponibles en este componente formativo; para ello, diríjase al menú principal, donde encontrará la síntesis, una actividad didáctica para reforzar los conceptos estudiados, material complementario, entre otros.



Actividad didáctica

Falso / verdadero

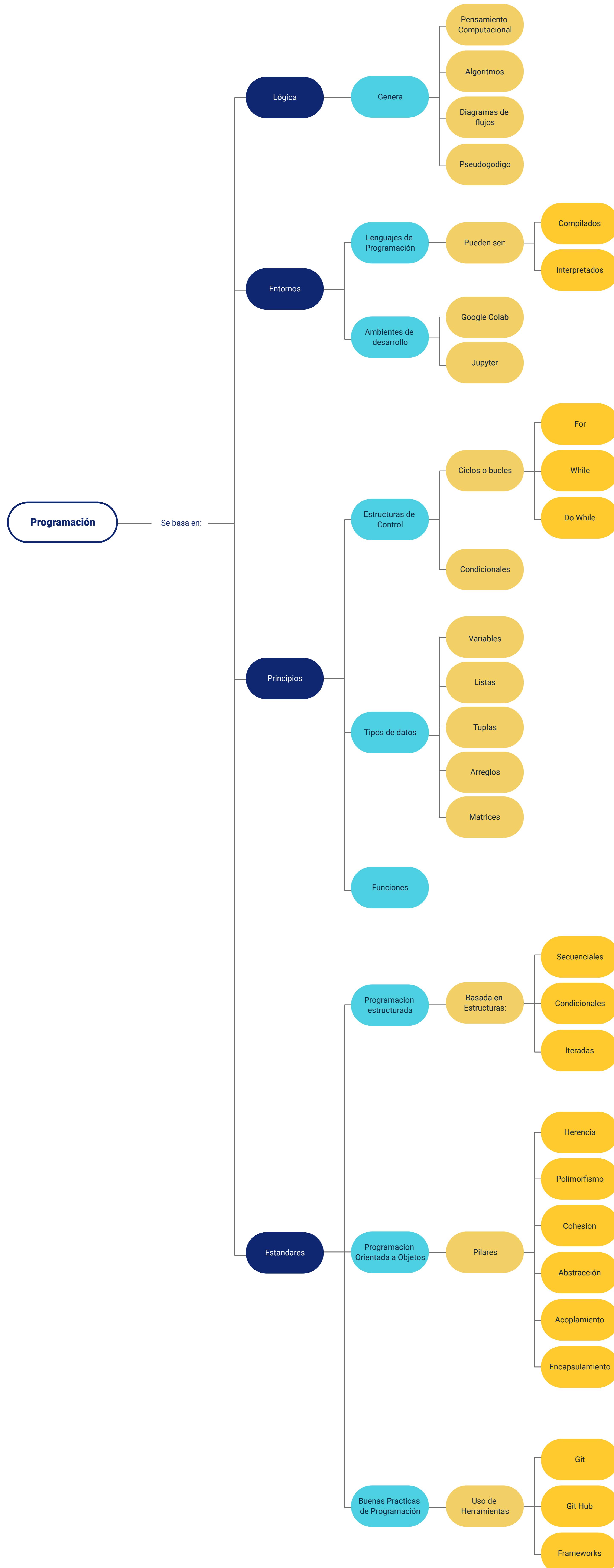


Apreciado aprendiz, a continuación, encontrará una serie de preguntas que deberá resolver, con el objetivo de evaluar la aprehensión de los conocimientos expuestos en este componente formativo:

Falso / verdadero

Realizar

El siguiente mapa integra los criterios y especificidades de los conocimientos expuestos en el presente componente formativo:





Portada actividad

800 x 800



Imagen acompañamiento Actividad

350 x 480