



Universidad
Tecnológica
del Perú

**FACULTAD DE
INGENIERÍA**

**Domina
Python y
Conquista el
Mundo de los
Datos.**



PYTHON Y EL MUNDO DE LOS DATOS



SESIÓN 01

Introducción a Python

Logros de Aprendizaje

General

Desarrollar competencias en programación con Python, desde fundamentos básicos hasta el manejo avanzado de datos con NumPy y Pandas. Los participantes aprenderán a implementar estructuras, funciones, conceptos de POO (herencia y encapsulamiento) y librerías de análisis, integrando estos conocimientos en un caso práctico para resolver problemas reales y presentar resultados de manera efectiva.

Sesión

Al finalizar la sesión, el estudiante desarrolla programas básicos con Python haciendo uso de variables, estructuras de control y estructuras de datos, como listas y diccionarios.

PYTHON Y EL MUNDO DE LOS DATOS



SESIÓN 01

Introducción a Python

- Historia y características de Python.
- Descarga e instalación de Visual Studio Code con Python.
- Variables en Python: Tipos de datos.
- Estructuras de control: if, for, while.
- Operaciones básicas con listas y diccionarios.

INICIO - Conocimientos Previos

¿Por qué aprender a programar?



<https://www.youtube.com/watch?v=qHAilBtZ3nE>

UTILIDAD - Sesión 01

- ¿Con qué finalidad se desarrolla el software?
- ¿Por qué conocer los aspectos fundamentales de Python?

Utilidad:

Conocer los fundamentos de la programación con Python sienta las bases para desarrollar aplicaciones para la gestión y análisis de datos.

```
class BigFile:
    def __init__(self, datadir, ndims):
        idfile = os.path.join(datadir, "id.txt")
        self.names = [x.strip() for x in str.split(open(idfile).read()) if x.strip()]
        self.name2index = dict(zip(self.names, range(len(self.names))))
        self.ndims = ndims
        self.featurefile = os.path.join(datadir, "feature.bin")
        print "[BigFile] %d features, %d dimensions" % (len(self.names), self.ndims)
        print "        binary: %s" % self.featurefile
        print "        txt: %s" % idfile

    def read(self, requested, isname=True):
        if isname:
            index_name_array = [(self.name2index[x], x) for x in requested if x in self.names]
        else:
            assert(min(requested) >= 0)
            assert(max(requested) < len(self.names))
            index_name_array = [(x, self.names[x]) for x in requested]
            index_name_array.sort()
        vecs = seq_read(self.featurefile, self.ndims, [x[0] for x in index_name_array])
        return [x[1] for x in index_name_array], vecs

    def shape(self):
        return [len(self.names), self.ndims]
```


Historia de Python

- Concebido a finales de los años 80 como un lenguaje para ser **interpretado** y orientado a la enseñanza.
- Lanzado en 1991. Con el tiempo se **convirtió** en una herramienta esencial para programadores, ingenieros e investigadores, tanto en el ámbito académico como industrial.
- Creado por el informático holandés **Guido van Rossum**, conocido durante muchos años con el *título* de **BDFL** (**B**enevolent **D**ictator for **L**ife).
- El nombre **Python** se debe a la afición de Guido por el programa de la BBC *Monty Python's Flying Circus*, del célebre grupo de humoristas británico **Monty Python**.



Características de Python

- Es un lenguaje de programación multiparadigma.
- Es multiplataforma.
- Sencillo de aprender, al ser un lenguaje simple y minimalista.
- Es interpretado.
- Usa tipado dinámico.
- De propósito general, casi todo puede programarse, por ejemplo: aplicaciones de escritorio, aplicaciones web, juegos, aplicaciones a medida, aplicaciones de inteligencia artificial, ciencia de los datos, etc.



Características de Python

- El éxito de **Python** reside no solo en su simplicidad, sino que sobre él se ha construido una enorme cantidad de herramientas para todo tipo de dominios de aplicación.
- Gran parte de los programas escritos en **Python** en computación científica y ciencia de datos está codificado utilizando el siguiente grupo de paquetes, consolidados casi como estándar:

NumPy

Almacenamiento y
computación
eficiente de
matrices
multidimensionales

SciPy

Colección de
herramientas
de cálculo
numérico

Pandas

Permite manipular,
filtrar, agrupar y
transformar datos,
así como el acceso
a las BD que
puedan contenerlos

Matplotlib

Conjunto de
funciones para la
creación de
figuras y gráficos
de gran calidad

SciKit-Learn

Paquete de
herramientas con
los algoritmos más
usuales de
aprendizaje
automático

Características de Python

- Además, de necesitarse alguna herramienta para dar tratamiento a un conjunto de datos, es muy probable que, dentro de la amplia *Comunidad Python*, esa herramienta ya esté programada y sea de *dominio público*.
- Obviamente, para aprovechar el poder de este ecosistema, es necesario familiarizarse con el lenguaje **Python**.



Entorno de desarrollo Integrado IDE de Python

- PyCharm.
- PyDev.
- Sublime Text 3.
- ATOM.
- VIM.
- Visual Studio Code.



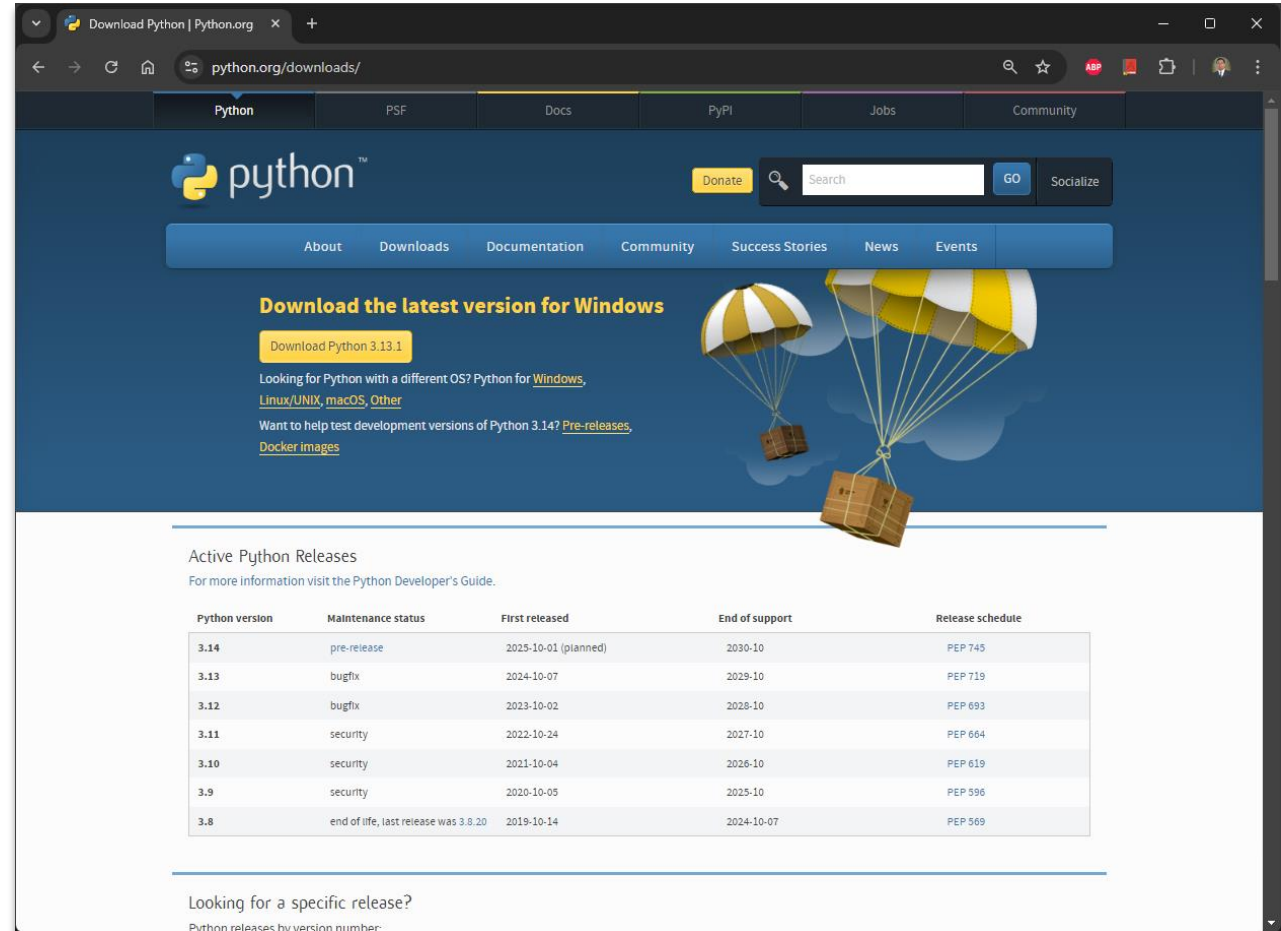
Visual Studio Code

Descarga e Instalación de Python.

Paso 01:

Descargar Python de la siguiente ruta:

<https://www.python.org/downloads/>



The screenshot shows the Python.org website's download page. The main heading is "Download the latest version for Windows" with a button to "Download Python 3.13.1". Below this, there are links for "Python for Windows, Linux/UNIX, macOS, Other" and "Pre-releases, Docker images". The page also features a section for "Active Python Releases" with a table of release details.









Python version	Maintenance status	First released	End of support	Release schedule
3.14	pre-release	2025-10-01 (planned)	2030-10	PEP 745
3.13	bugfix	2024-10-07	2029-10	PEP 719
3.12	bugfix	2023-10-02	2028-10	PEP 693
3.11	security	2022-10-24	2027-10	PEP 664
3.10	security	2021-10-04	2026-10	PEP 619
3.9	security	2020-10-05	2025-10	PEP 596
3.8	end of life, last release was 3.8.20	2019-10-14	2024-10-07	PEP 569

Descarga e Instalación de Python.

Paso 02: Clic en “Download” de la versión Python 3.12.3 del 9 de abril 2024.

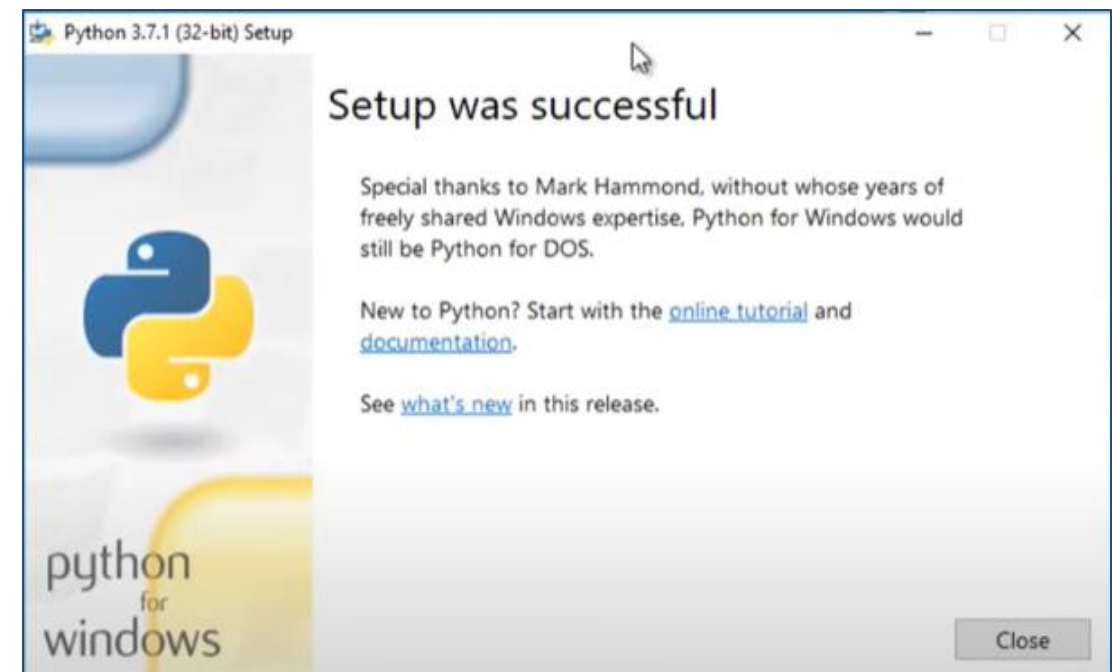
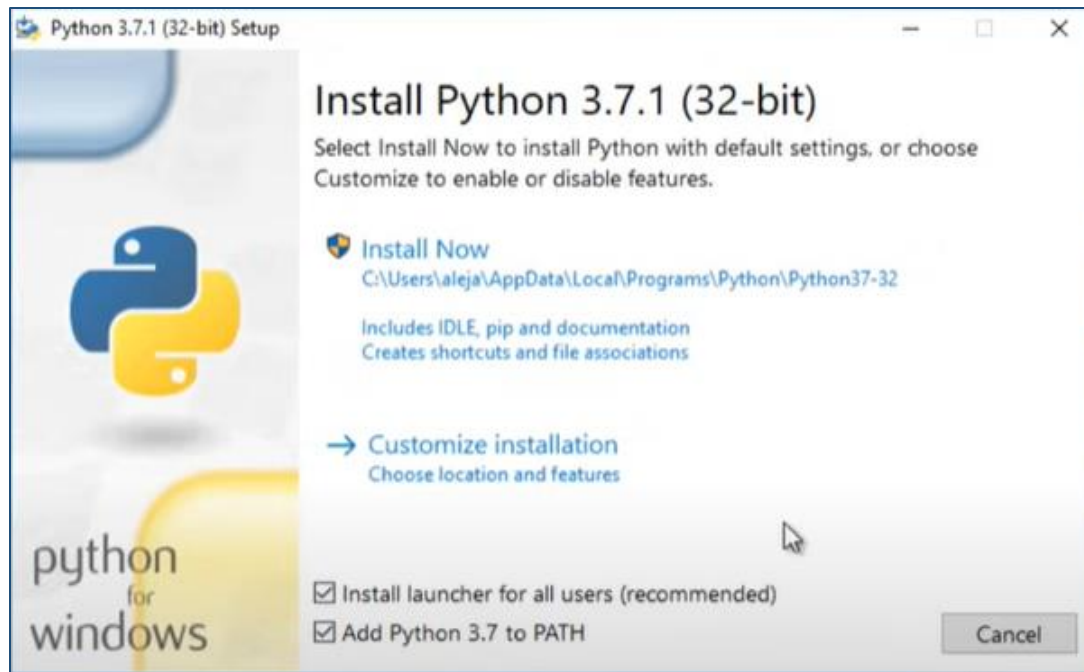
Looking for a specific release?

Python releases by version number:

Release version	Release date	Click for more	
Python 3.12.3	April 9, 2024	 Download	Release Notes
Python 3.11.9	April 2, 2024	 Download	Release Notes
Python 3.10.14	March 19, 2024	 Download	Release Notes
Python 3.9.19	March 19, 2024	 Download	Release Notes
Python 3.8.19	March 19, 2024	 Download	Release Notes
Python 3.11.8	Feb. 6, 2024	 Download	Release Notes
Python 3.12.2	Feb. 6, 2024	 Download	Release Notes
Python 3.12.1	Dec. 8, 2023	 Download	Release Notes

Descarga e Instalación de Python.

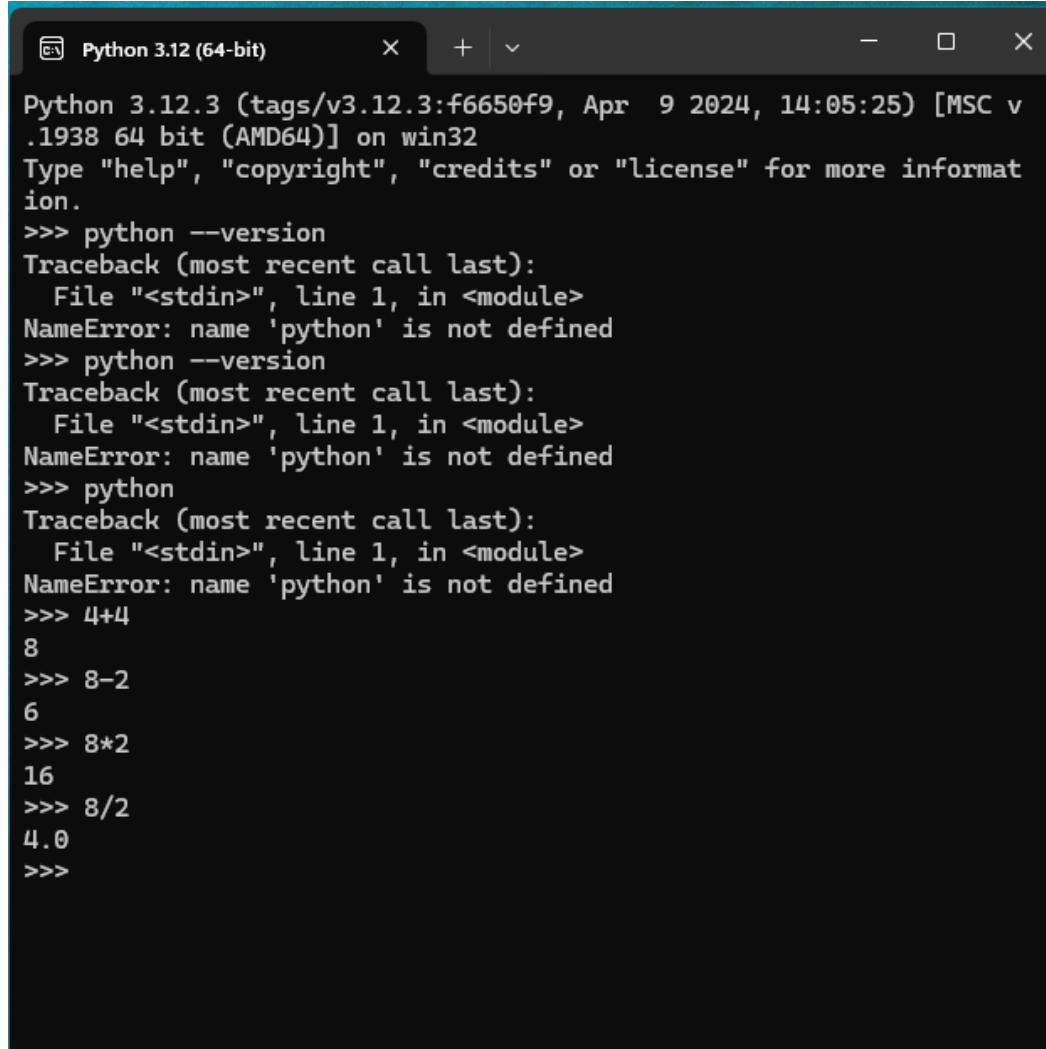
Paso 03: Instalar Python 3.12.3 del 9 de abril 2024.



Descarga e Instalación de Python.

Paso 04:

Probar consola de Python 3.12.3



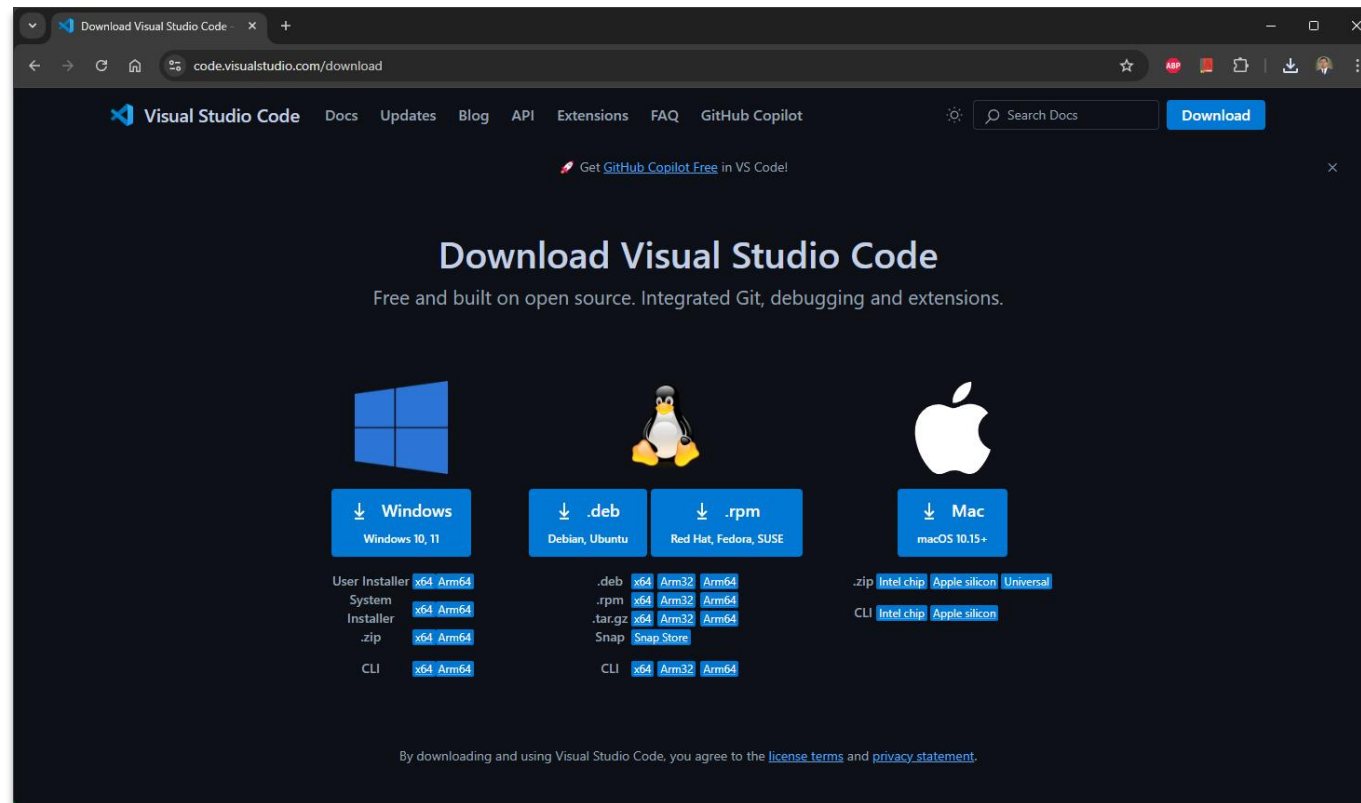
```
Python 3.12 (64-bit)
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v
.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more informat
ion.
>>> python --version
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'python' is not defined
>>> python --version
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'python' is not defined
>>> python
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'python' is not defined
>>> 4+4
8
>>> 8-2
6
>>> 8*2
16
>>> 8/2
4.0
>>>
```

Descarga e Instalación del Visual Studio Code

Paso 01:

Instalar Visual Studio Code de la siguiente ruta:

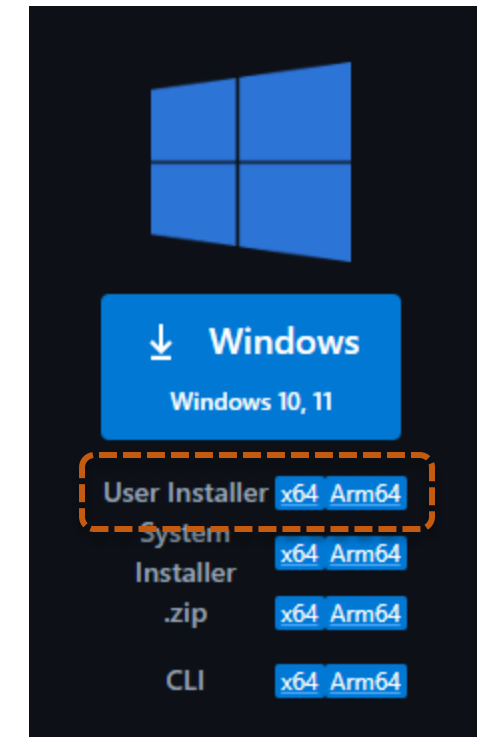
<https://code.visualstudio.com/download>



Paso 02:

Descargar el Instalador gratuito:

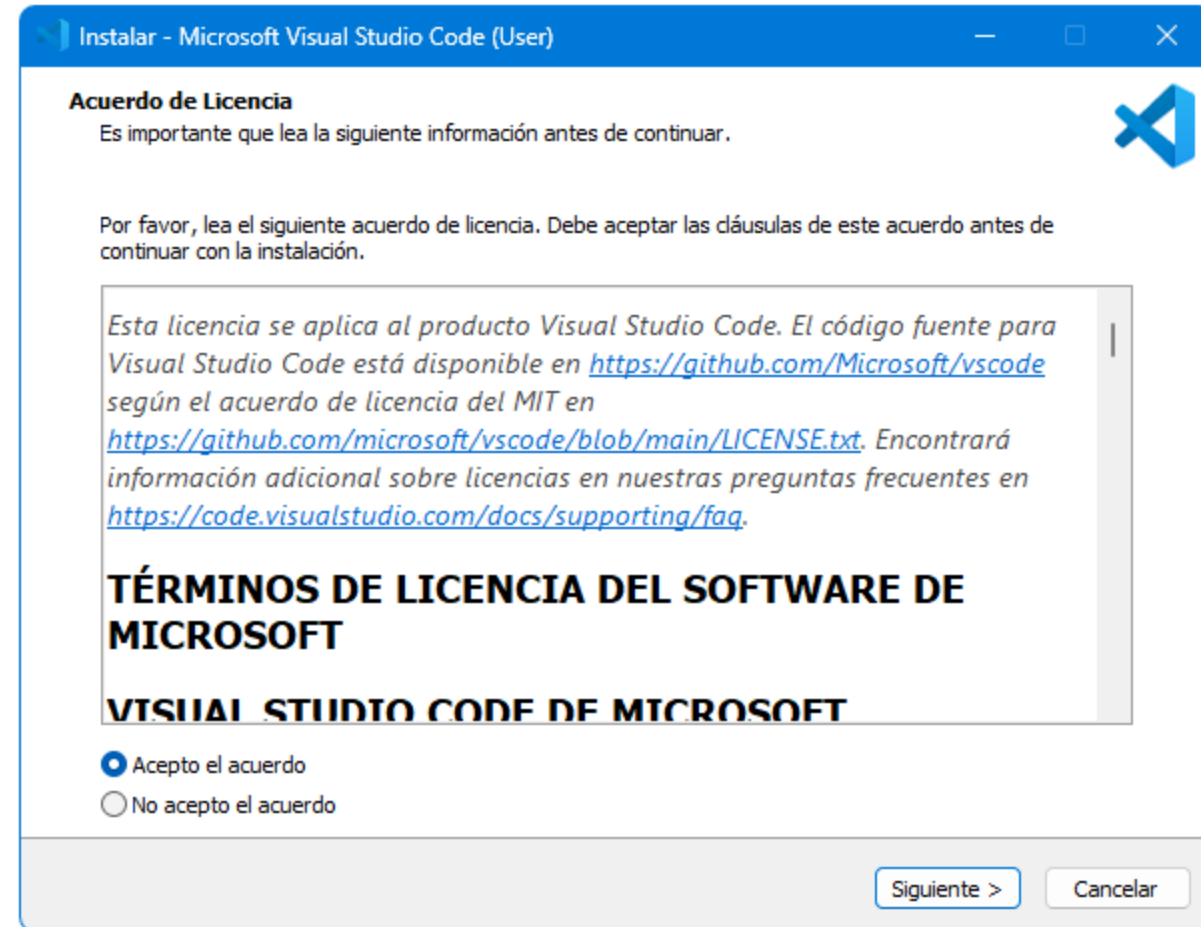
Windows x64 Instalador de usuario



Descarga e Instalación del Visual Studio Code

Paso 03:

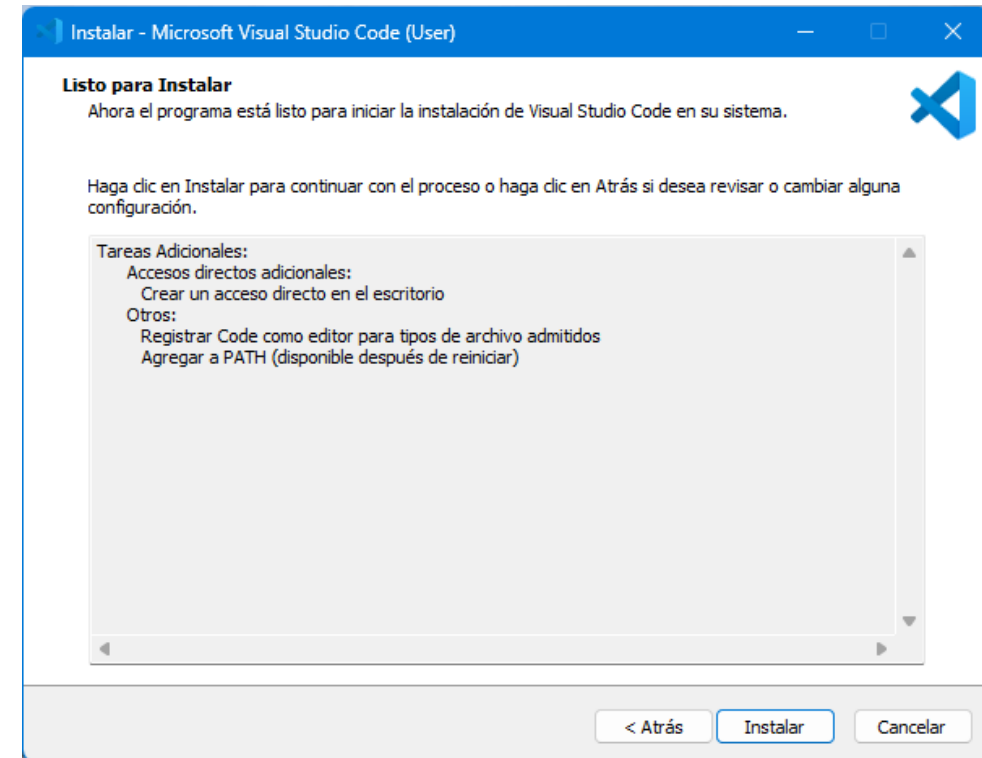
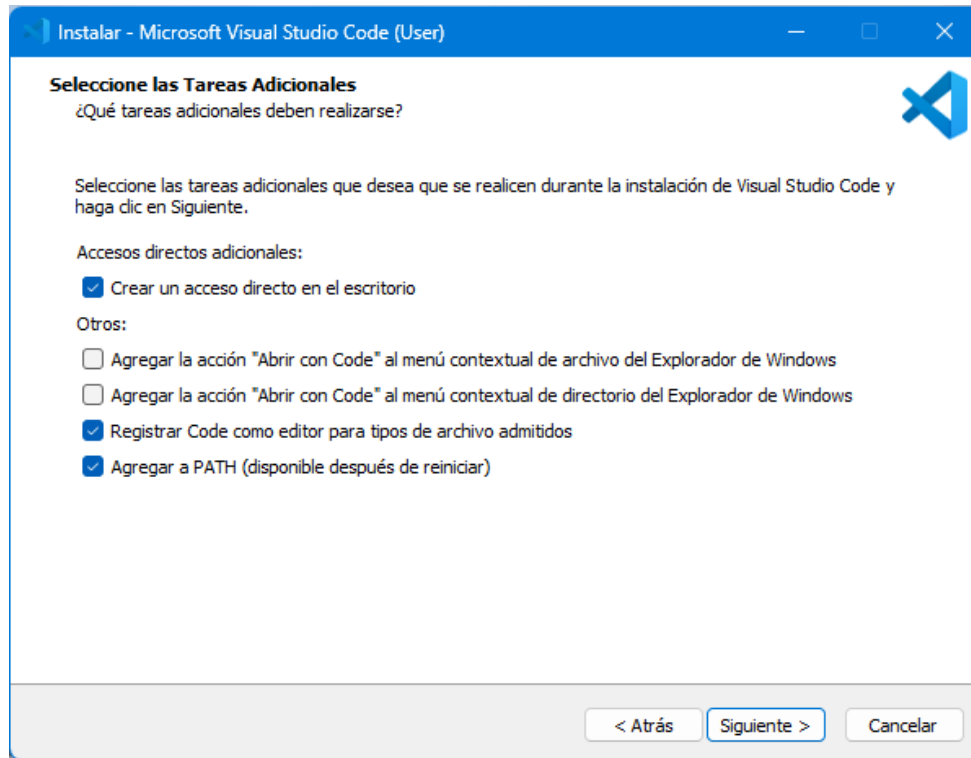
Ejecuta el instalador y Acepta el acuerdo de la licencia gratuita.



Descarga e Instalación del Visual Studio Code

Paso 04:

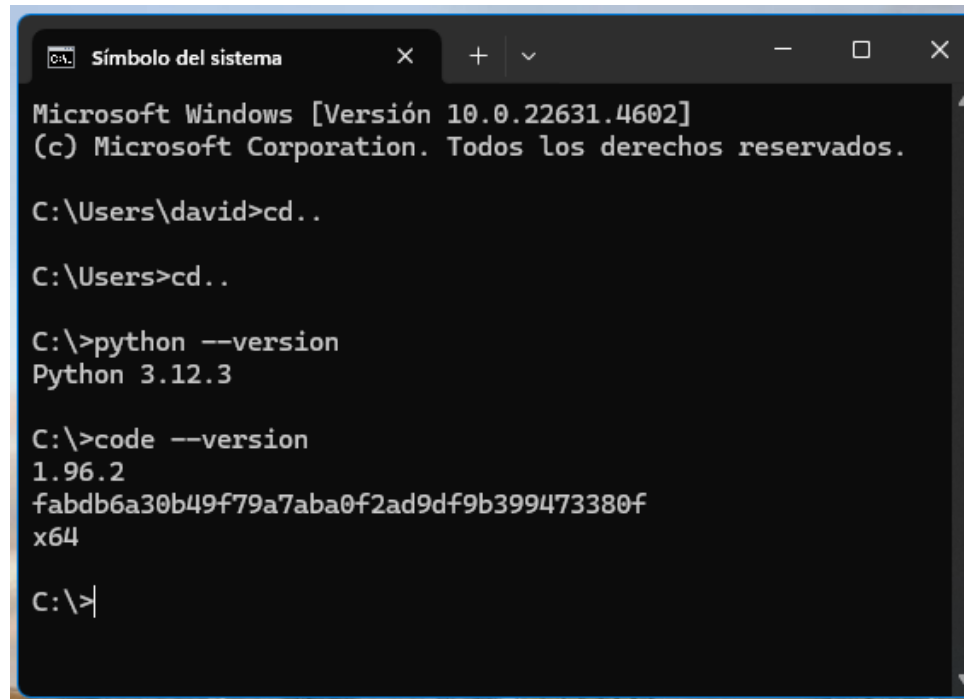
Continúa con los siguientes pasos de la instalación.



Configuración de VS Code para Python

Paso 01:

A través de una terminal o símbolo del sistema, verificar las versiones de Python y Visual Studio Code.



```
Símbolo del sistema
Microsoft Windows [Versión 10.0.22631.4602]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\david>cd..

C:\Users>cd..

C:\>python --version
Python 3.12.3

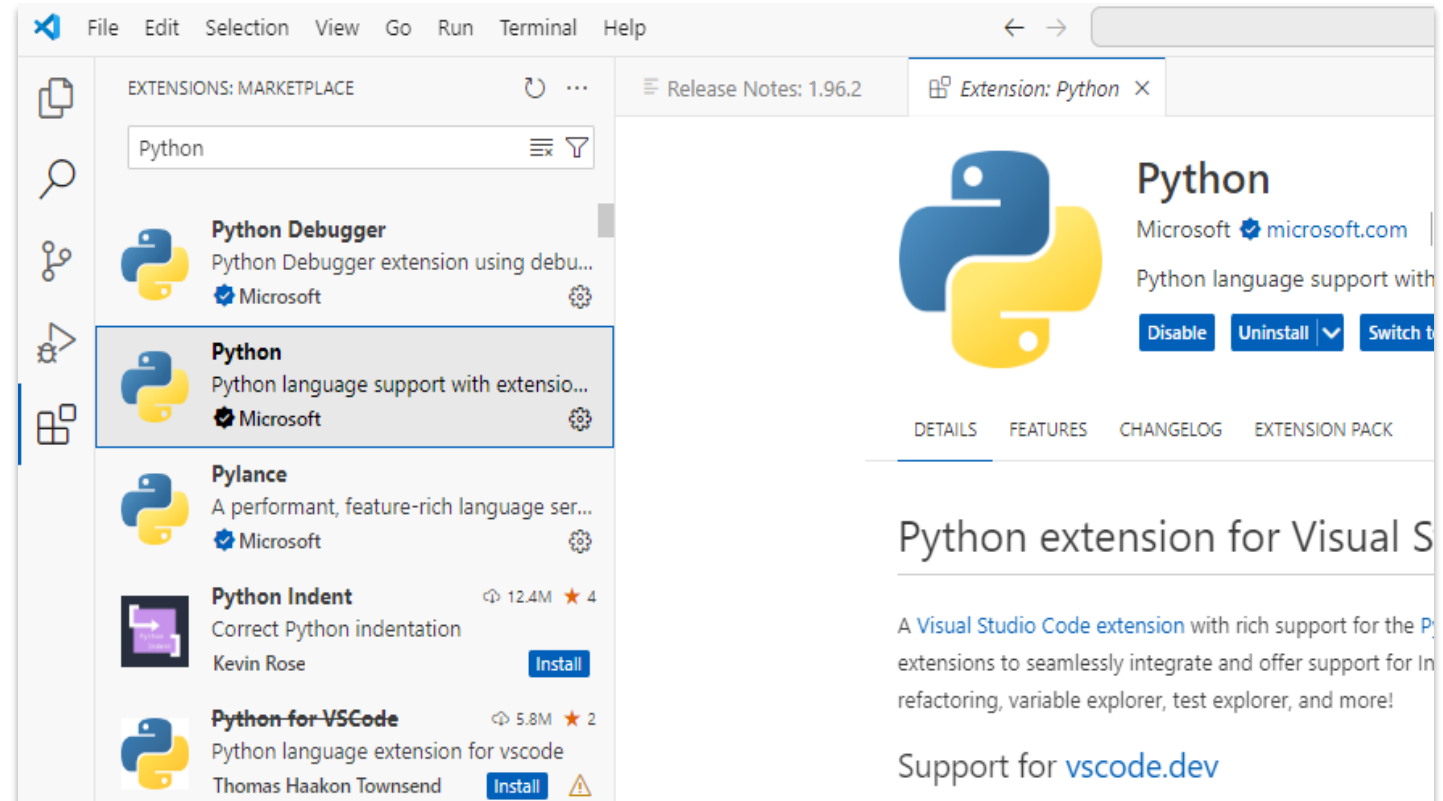
C:\>code --version
1.96.2
fabdb6a30b49f79a7aba0f2ad9df9b399473380f
x64

C:\>|
```


Configuración de VS Code para Python

Paso 02:

- Ingresar a Visual Studio Code y en la sección extensiones, escribir “Python”.
- Luego elegir la extensión “Python de Microsoft”.
- Posteriormente se debe instalar la extensión.



Variable en Python: Tipos de datos

¿Recuerdas qué es una variable?

```
a = 3.0
b = 4.0
c = (a ** 2 + b ** 2) ** 0.5
print("c =", c)
```

Declaración

- Reserva de espacio de memoria.
- La variable no debe ser usada aún.



Inicialización

- Primera asignación de contenido a la variable.
- Necesaria siempre antes de su uso.



Utilización

- La variable es utilizada en procesos del programa.
- Su contenido puede cambiar N veces.

Variable en Python: Tipos de datos

Tipo	Clase	Notas	Ejemplo
int	Entero	Número entero	30
float	Decimal	Coma flotante	3.1416
bool	Booleano	Valor verdadero o falso	True, False
str	Cadena	Inmutable	'Hola'
list	Secuencia	Mutable	[3.0, 'Hola']
tuple	Secuencia	Inmutable	(3.0, 'Hola')
set	Conjunto	Mutable, sin orden, sin duplicados	set([3.0, 'Hola'])
frozenset	Conjunto	Inmutable, sin orden, sin duplicados	frozenset([3.0, 'Hola'])
dict	Diccionario (Mapa)	Pares clave:valor	{'clave1':4, 'clave2': 'Hola'}

Instrucciones básicas en Python

Instrucción	Descripción
<code>print()</code>	imprime expresiones en pantalla.
<code>input()</code>	Solicita datos desde teclado.
<code>type()</code>	Devuelve el tipo de dato del argumento.
<code>int()</code>	Convierte un valor a entero.
<code>float()</code>	Convierte un valor a float.
<code>str()</code>	Convierte un valor a cadena de texto.

Operadores en Python

Operador	Descripción	Ejemplo
+	Suma	<code>r = 3 + 2 # r es 5</code>
-	Resta	<code>r = 4 - 7 # r es -3</code>
*	Multiplicación	<code>r = 2 * 6 # r es 12</code>
**	Potencia	<code>r = 2 ** 6 # r es 64</code>
/	División	<code>r = 3.5 / 2 # r es 1.75</code>
//	División entera	<code>r = 3.5 // 2 # r es 1.0</code>
%	Residuo (módulo)	<code>r = 7 % 2 # r es 1</code>

Ingreso de Datos por teclado

Función input()

- Permite obtener texto escrito por teclado.
- La función hace que el programa se detenga y espere a que el usuario escriba algo.
- Al finalizar pulsar la tecla **Enter**, como muestra la imagen.

```
nombre = input("¿Cuál es tu nombre? ")  
print("Bienvenid@, ", nombre)
```

```
¿Cuál es tu nombre? Fernando  
Bienvenid@, Fernando
```

Conversiones de tipos

- Si se quiere que Python interprete la entrada como un número entero o real, deben utilizarse las funciones `int()` y `float()`, respectivamente:

```
edad = int(input("Ingrese su edad: "))  
print("Usted tiene", edad, "años")
```

```
Ingrese su edad: 28  
Usted tiene 28 años
```

Si ingresa un número decimal la función `int()` devolverá un error

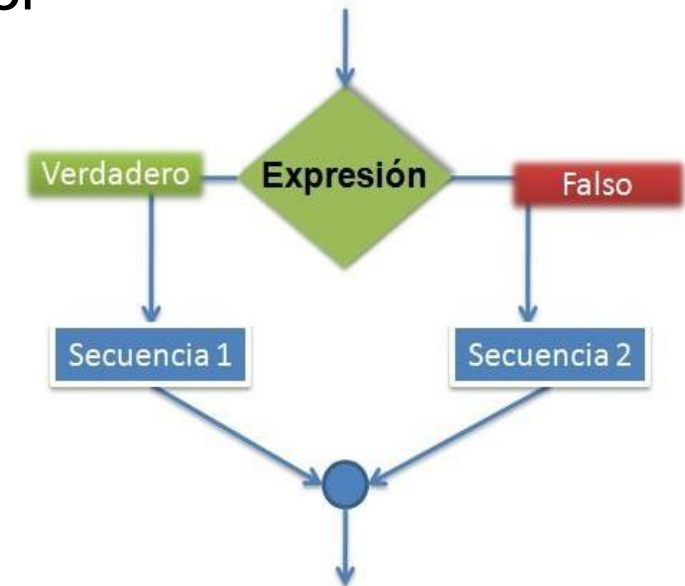
```
dinero = float(input("Ingrese un monto: "))  
print("Usted tiene", dinero, "soles")
```

```
Ingrese un monto: 20.5  
Usted tiene 20.5 soles
```

Si ingresa un número entero la función `float()` no devolverá un error

Estructuras de Control: if, for, while.

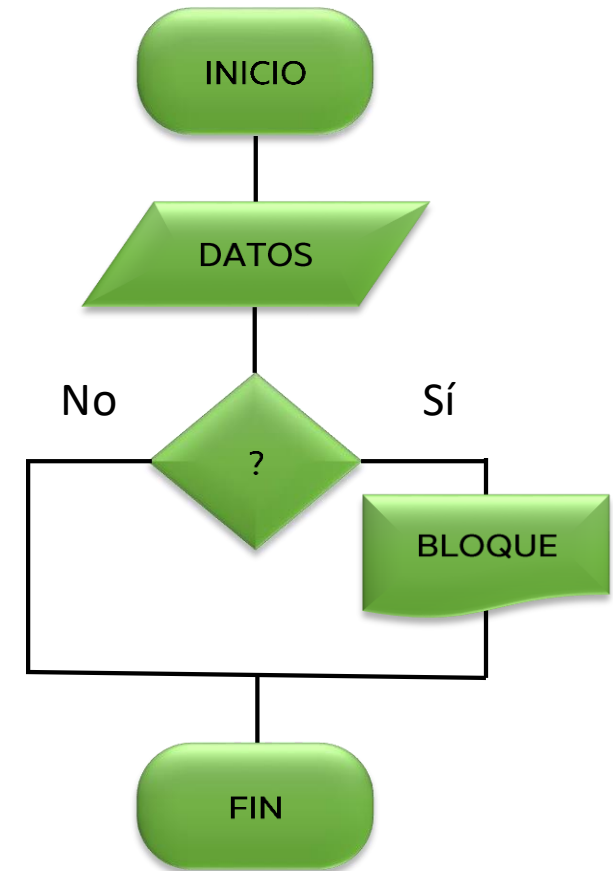
- Las estructuras condicionales permiten la ejecución selectiva de las instrucciones, dependiendo del valor de la expresión asociada con ellas.
- Los operadores de comparación son necesarios para la evaluación de las condiciones.
- Las diferentes estructuras condicionales son las siguientes:
 - La condicional simple: if
 - La condicional doble: if...else



Estructuras de Control: if, for, while.

- La estructura de control **if** permite que un programa ejecute instrucciones cuando se cumpla una condición.
- Una condicional simple en Python se escribe de la siguiente manera:

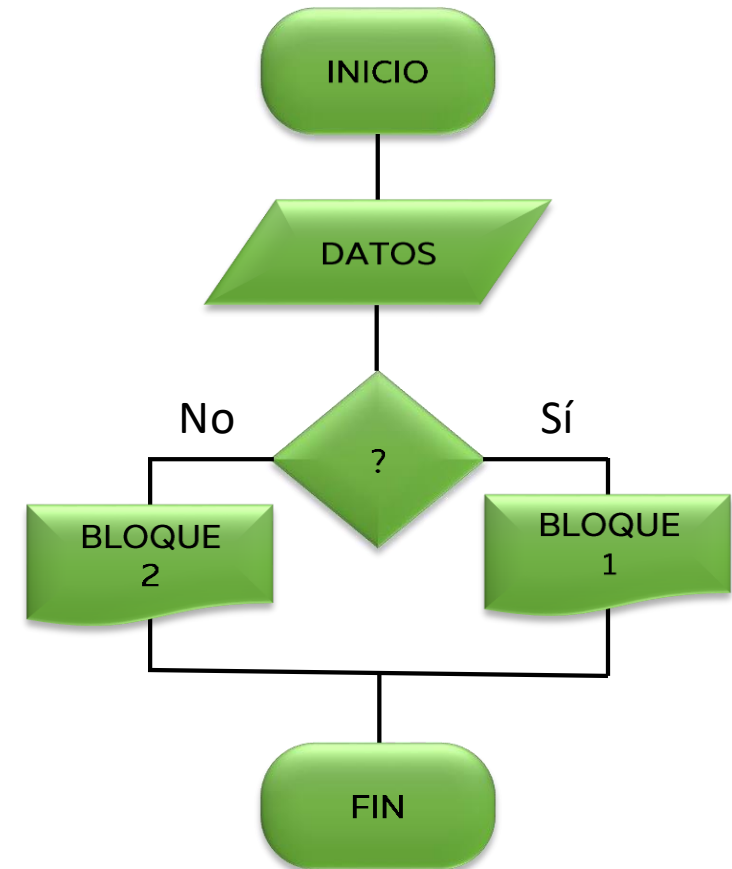
```
if condicion:  
    # Instrucciones a ejecutar si  
    # la condición es cierta  
    # y que pueden ocupar varias líneas
```



Estructuras de Control: if, for, while.

- La estructura condicional doble **if...else** permite ejecutar unas instrucciones cuando se cumpla una condición y otras instrucciones cuando no.
- Una condicional doble en Python se escribe de la siguiente manera:

```
if condicion:  
    # Instrucciones a ejecutar si  
    # la condición es cierta  
else:  
    # Instrucciones a ejecutar si  
    # la condición no es cierta
```



Estructuras de Control: if, for, while.

ENUNCIADO

- Escribe un programa en Python que permita el ingreso de dos números y luego muestre un mensaje indicando si el segundo número ingresado es mayor o menor que el primero.



Estructuras de Control: if, for, while.

SOLUCIÓN:

```
# Entrada de datos

a = int(input("Ingrese primer número: "))
b = int(input("Ingrese segundo número: "))

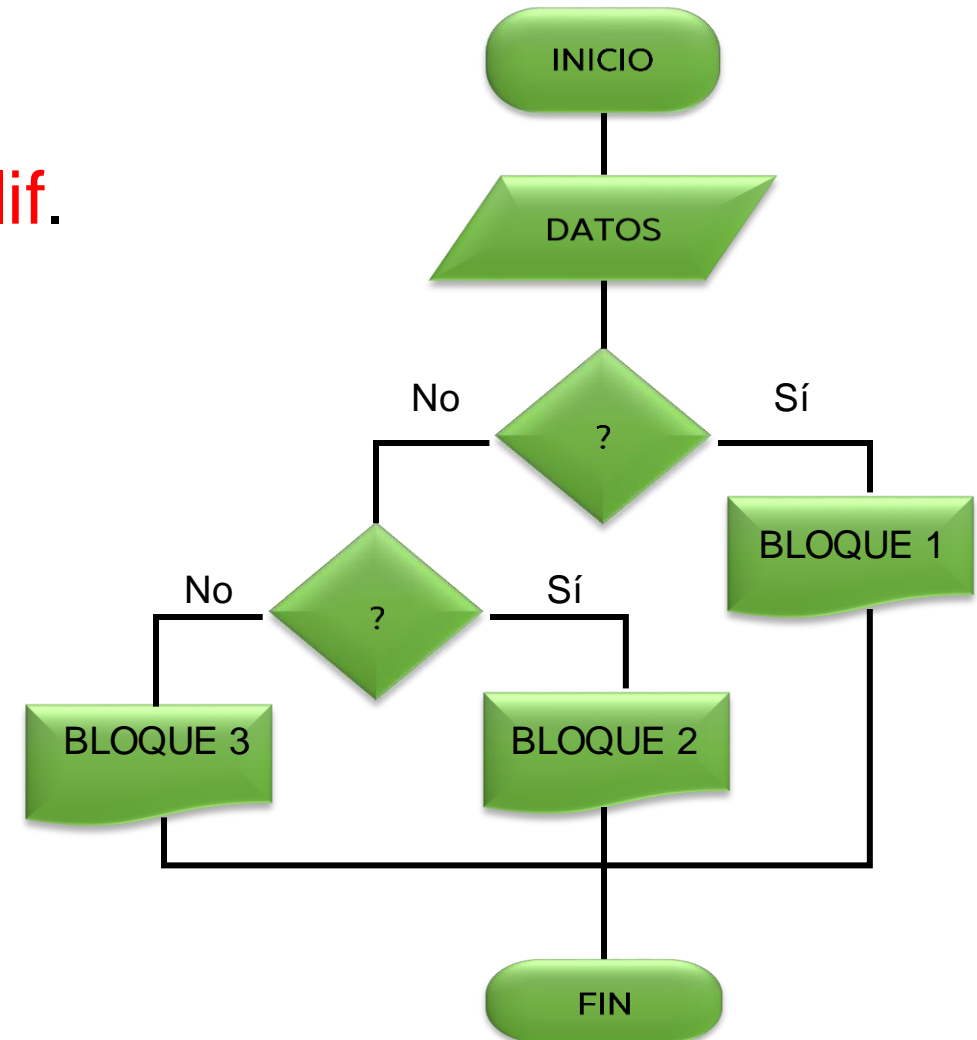
# Evaluación de los números
# e impresión de mensajes
if b > a:
    print(b, " es mayor que ", a)
else:
    print(b, " es menor que ", a)
```



Estructuras de Control: if, for, while.

- La instrucción **if...else** se puede extender añadiendo la instrucción **elif**.
- La estructura **if...elif...else** permite encadenar varias condiciones.
- **elif** es una contracción de **else if**.

```
if condicion_1:  
    # bloque 1  
elif condicion_2:  
    # bloque 2  
else:  
    # bloque 3
```



Estructuras de Control: if, for, while.

ENUNCIADO

- Una tienda vende un producto a precios unitarios que dependen de la cantidad de unidades adquiridas de acuerdo a la tabla mostrada:
- Adicionalmente, si el cliente adquiere más de 50 unidades la tienda le descuenta el 15% del importe de la compra; en caso contrario, sólo le descuenta el 5%.
- Diseñe un programa que determine el importe de la compra, el importe del descuento y el importe a pagar por la compra de cierta cantidad de unidades del producto.

Cantidad comprada	Precio por unidad
Entre 1 y 25	27.7
Entre 26 y 50	25.5
Entre 51 y 75	23.5
76 o más	21.5

Estructuras de Control: if, for, while.

```
# Entrada de datos
```

```
unidades = int(input("Ingrese unidades: "))
```

```
# Calculamos el importe a pagar  
# and (y), or (o), not (negación)
```

```
if unidades >= 1 and unidades <= 25:  
    importeCompra = unidades * 27.7  
elif unidades >= 26 and unidades <= 50:  
    importeCompra = unidades * 25.5  
elif unidades >= 51 and unidades <= 75:  
    importeCompra = unidades * 23.5  
elif unidades >= 76:  
    importeCompra = unidades * 21.5
```

```
# Calculamos el importe de descuento
```

```
if unidades > 50:  
    importeDescuento = 0.15 * importeCompra  
else:  
    importeDescuento = 0.05 * importeCompra
```

```
# Calcular el importe a pagar  
importePago = importeCompra - importeDescuento
```

```
# Salida de resultados  
print("Importe de compra:", importeCompra)  
print("Importe de descuento:",  
      importeDescuento)  
print("Importe a pagar:", importePago)
```

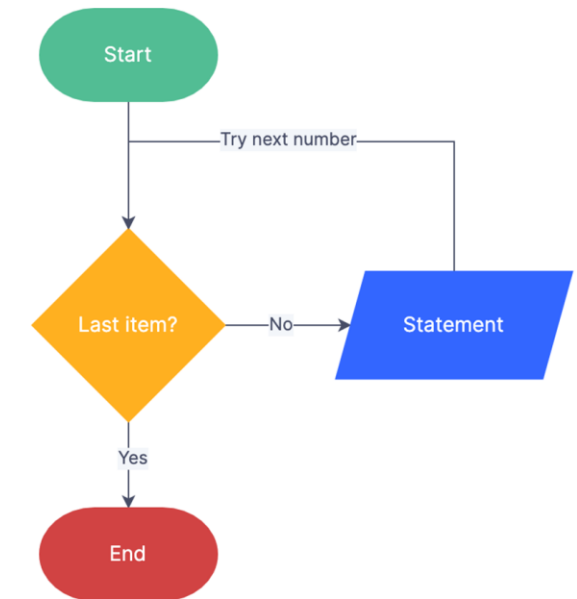
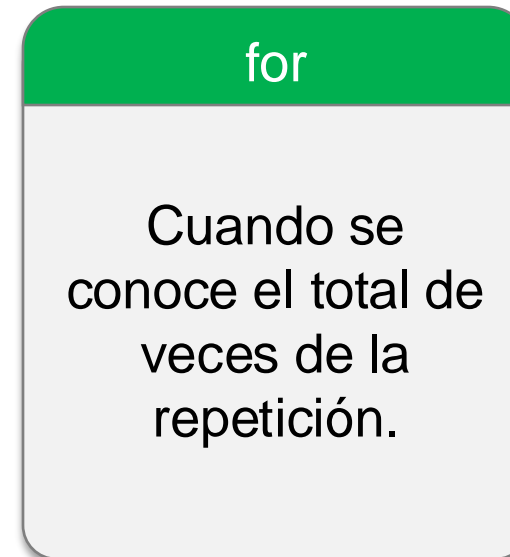
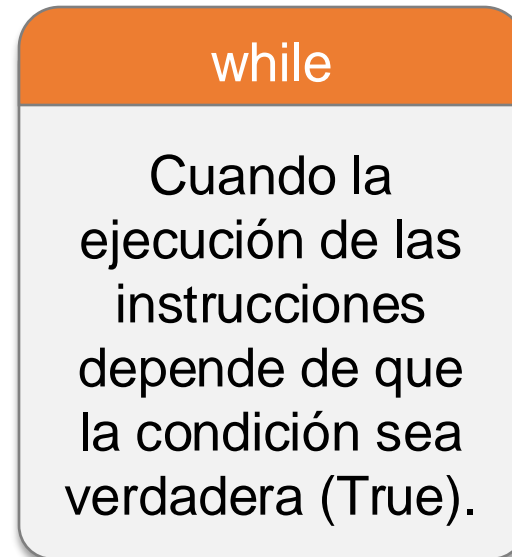
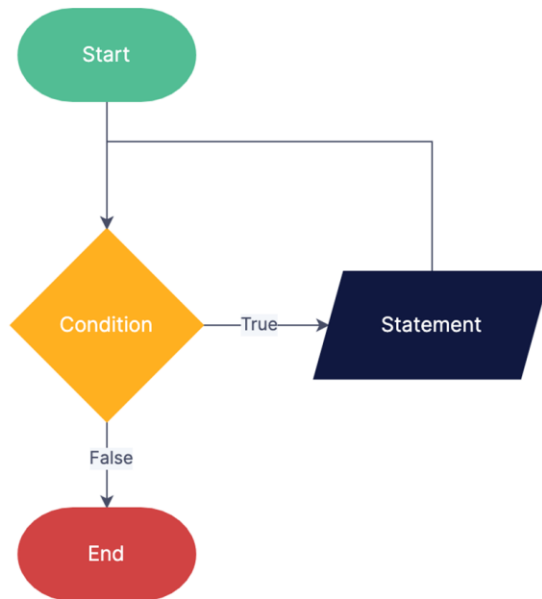
Estructuras de Control: if, for, while.

- Las estructuras repetitivas se utilizan cuando se quiere que un conjunto de instrucciones se ejecuten un número finito de veces.
 - Ejemplos: escribir algo en pantalla cierta cantidad de veces, mover un objeto de un punto a otro, cierta cantidad de pasos, o hacer una operación matemática cierta cantidad de veces.
- A estas estructuras se les conoce también como estructuras iterativas o bucles.
- A las instrucciones por repetir se le conoce como el cuerpo del bucle, y al hecho de repetir la secuencia de instrucciones se denomina iteración.



Estructuras de Control: if, for, while.

- Tipos de bucles:



Estructuras de Control: if, for, while.

- Sintaxis del bucle while:

```
while condicion:  
    # cuerpo del bucle
```

- Tipos de bucle while:
 - while controlado por conteo.
 - while controlado por evento.

Estructuras de Control: if, for, while.

```
print('While controlado por conteo')
print("-----")
print("Sumador hasta el 10")
```

```
sum = 0
num = 1
```

```
while sum <= 10:
    sum += num
    num += 1
```

```
print("La suma es " + str(sum))
```

```
print("While controlado por evento")
print("-----")
print("Calcular promedio")
```

```
promedio = 0.1
total = 0
contar = 0
```

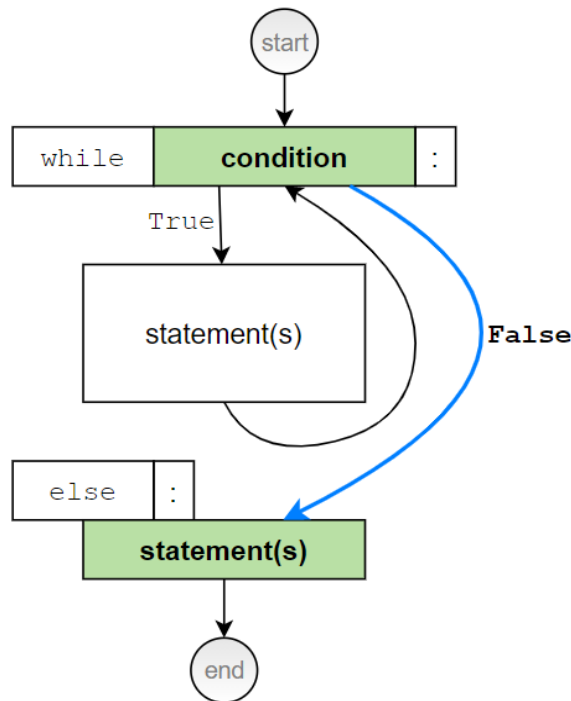
```
nota = int(input("Ingresa un valor (-1 para salir): "))
```

```
while nota != -1:
    total += nota
    contar += 1
    nota = int(input("Ingresa un valor (-1 para salir): "))
```

```
promedio = total / contar
print("El promedio es " + str(promedio))
```

Estructuras de Control: if, for, while.

- La cláusula **while-loop-else**, es exclusiva de Python.
- El bloque **else** solo se ejecuta si la condición de **while** es falsa.



```
palabra = 'universidad'
letraBuscada = 'v'
i = 0

while i < len(palabra):
    if (palabra[i] == letraBuscada):
        print(letraBuscada, 'encontrada!')
        break # se interrumpe el bucle
    i += 1
else:
    print(letraBuscada, "no encontrada")
```

Listas en Python

- Las listas en Python son estructuras de datos para almacenar múltiples elementos en un solo contenedor.
- Son secuenciales, es decir, los elementos tienen un orden determinado, cada uno con un índice, que comienza desde 0.
- Pueden contener elementos de diferentes tipos: números, cadenas, booleanos, etc., lo que las hace muy flexibles.
- Se definen con corchetes [], por ejemplo:

```
mi_lista = [1, 2, 3, "hola", True]
```

Operaciones básicas con listas

Operación	Ejemplo
Obtener el valor de un elemento usando su índice.	<code>mi_lista[0]</code> # devuelve el primer elemento
Cambiar el valor de un elemento en una posición específica.	<code>mi_lista[1] = 10</code> # cambia el segundo elemento a 10
Añadir un elemento al final de la lista.	<code>mi_lista.append(5)</code>
Insertar un elemento en una posición específica.	<code>mi_lista.insert(1, 7)</code>
Eliminar la primera aparición de un elemento en la lista.	<code>mi_lista.remove(3)</code>
Eliminar el elemento en una posición específica y devolverlo.	<code>mi_lista.pop(2)</code>
Vaciar la lista completamente.	<code>mi_lista.clear()</code>
Obtener el índice de la primera aparición de un valor en la lista.	<code>mi_lista.index(7)</code>
Verificar si un elemento existe en la lista.	<code>7 in mi_lista</code> # devuelve True o False
Obtener el número de elementos en la lista.	<code>len(mi_lista)</code>
Invertir el orden de los elementos en la lista.	<code>mi_lista.reverse()</code>
Ordenar los elementos de la lista (<i>solo funciona con elementos comparables</i>).	<code>mi_lista.sort()</code>

Diccionarios en Python

- Los diccionarios en Python son estructuras de datos que almacenan pares de claves y valores.
- A diferencia de las listas, que se indexan por números enteros, los diccionarios utilizan claves, que pueden ser de cualquier tipo inmutable (*como cadenas, números o tuplas*), y permiten acceder a los valores asociados.
- Los diccionarios se definen utilizando llaves { }, y cada par de clave-valor se separa por dos puntos :, por ejemplo:

```
mi_diccionario = {"nombre": "Juan", "edad": 30, "ciudad": "Tacna"}
```

Operaciones básicas con diccionarios

Operación	Ejemplo
Obtener el valor asociado a una clave.	<code>mi_diccionario["nombre"]</code> # devuelve el valor asociado a "nombre"
Añadir un nuevo par clave-valor o modificar el valor de una clave existente.	<code>mi_diccionario["edad"] = 31</code> # cambia el valor de "edad" a 31
Eliminar un par clave-valor del diccionario.	<code>del mi_diccionario["ciudad"]</code>
Obtener una vista de todas las claves del diccionario.	<code>mi_diccionario.keys()</code>
Obtener una vista de todos los valores del diccionario.	<code>mi_diccionario.values()</code>
Obtener una vista de todos los pares clave-valor del diccionario.	<code>mi_diccionario.items()</code>
Verificar si una clave está presente en el diccionario.	<code>"nombre" in mi_diccionario</code>
Obtener un valor para una clave, o un valor por defecto si la clave no existe.	<code>mi_diccionario.get("edad", 0)</code> # devuelve 0 si "edad" no existe
Eliminar un par clave-valor y devolver el valor eliminado.	<code>mi_diccionario.pop("edad")</code>
Eliminar todos los pares clave-valor del diccionario.	<code>mi_diccionario.clear()</code>

Práctica – Sesión 1

- Ejercicio 01: Escribimos un programa en Python que solicite el nombre de un alumno y tres calificaciones (enteros). El programa debe calcular y mostrar el promedio (con dos decimales) con el siguiente formato:
(ejemplo) “Juan, su promedio es: 16.50”



Práctica – Sesión 1

- Solución:

input() permite ingresar datos desde la consola

```
nombre = input("Ingrese su nombre: ")

nota1 = int(input("Ingrese nota 1: "))
nota2 = int(input("Ingrese nota 2: "))
nota3 = int(input("Ingrese nota 3: "))

promedio = (nota1 + nota2 + nota3) / 3.0

print(f"{nombre}, su promedio es {promedio}")
print(f"{nombre}, su promedio es {promedio:.2f}")
```



int() convierte texto
a número entero

anteponiendo "f" a una expresión
podemos interpolar variables

Práctica – Sesión 1

- Ejercicio 02: Trabajando con listas.
Creamos una lista de nombres de países y luego usamos funciones del lenguaje para agregar, eliminar y ordenar los elementos. Finalmente, utilizamos bucles para recorrer la lista y mostrar sus elementos.



Práctica – Sesión 1

definimos una lista usando corchetes [] y separando los valores con una coma

```
listaPaíses = ['Perú', 'Argentina', 'Chile', 'Brasil', 'Colombia']  
print(listaPaíses)
```

agregamos al final

```
listaPaíses.append('Bolivia')  
listaPaíses.append('Cuba')
```



insertamos en una posición específica

```
listaPaíses.insert(2, 'Ecuador')  
listaPaíses.insert(2, 'Ecuador') # Pueden repetirse los elementos  
listaPaíses.insert(5, 'El Salvador')
```

agregamos una lista

```
listaPaíses.extend(['México', 'Estados Unidos', 'Canadá'])  
  
print(listaPaíses)
```

eliminamos elementos

```
listaPaíses.remove('Chile')  
listaPaíses.remove('Estados Unidos')  
# listaPaíses.remove('Francia') # Error! elemento no existe
```

Práctica – Sesión 1

ordenamos la lista

```
listaPaises.sort()  
print(listaPaises)
```



invertimos el orden

```
listaPaises.reverse()  
print(listaPaises)
```

bucle while

```
i = 0  
while i < len(listaPaises):  
    print(listaPaises[i])  
    i += 1
```



recorrido de colección
usando for

```
for pais in listaPaises:  
    print(pais)
```

recorrido usando for
con rango e índice

```
for i in range(0, len(listaPaises)):  
    print(listaPaises[i])
```

formateando la salida

```
for idx in range(0, len(listaPaises)):  
    print(f"{idx + 1}. {listaPaises[idx]}")
```

-
- A top-down view of a white surface crowded with a variety of personal electronic items. In the upper left, there's a round silver clock and a white power adapter. Next to it is a black smartphone. To the right of the clock is a yellow digital clock. Further right is a white battery pack and a black game controller. In the center, there's a black tablet with an orange border, a small yellow and black action camera, and a black smartphone. To the right of the tablet is a black mouse and a white keyboard. In the bottom left, there's a blue and white calculator, a small white cube, and a pink fan. On the bottom right, there's a light blue hair clipper and a small white USB drive. Various cables are scattered throughout the scene.

Práctica – Sesión 1

definimos un diccionario
usando llaves { } y
separando los pares
clave:valor con comas

```
dicProductos = { 'Teclado': 128.50, 'Monitor': 368.00,  
                 'Mouse': 58.7, 'Cámara Web': 284.30}  
print(dicProductos)
```



Actualizamos y agregamos
elementos

```
dicProductos.update({'Mouse': 62.53}) # actualizamos un elemento  
dicProductos.update({'Impresora': 690.20}) # agregamos un elemento  
print(dicProductos)
```

Accedemos a los valores
mediante las claves

```
print(dicProductos['Cámara Web'])  
print(dicProductos.get("Monitor"))
```

pop() retorna el valor del elemento cuya clave se
proporciona y extrae el elemento del diccionario

```
print(dicProductos.pop('Teclado'))  
print(dicProductos)
```

popitem() extrae el último elemento del
diccionario

```
ultimoItem = dicProductos.popitem()  
print(f"{ultimoItem[0]} = {ultimoItem[1]}")
```

Práctica – Sesión 1

Accediendo a la lista de claves y valores

```
print(dicProductos.keys())  
print(dicProductos.values())
```



imprimimos las claves del diccionario

```
for clave in dicProductos.keys():  
    print(clave)
```



imprimimos los valores del diccionario

```
for valor in dicProductos.values():  
    print(valor)
```

accedemos a los elementos (tuplas)

```
for elemento in dicProductos.items():  
    print(elemento) # tupla  
    print(elemento[0]) # clave  
    print(elemento[1]) # valor
```

Práctica – Sesión 1

- Ejercicio 04: Trabajando con condicionales. Clasificar edades en categorías usando condicionales. Mostrar el número de edades por categoría.



Práctica – Sesión 1

```
dicCategorias = { 'No válido': 0, 'Niño': 0,  
                  'Adolescente': 0, 'Adulto': 0, 'Adulto Mayor': 0}  
continuar = True
```

```
while continuar:  
    edad = int(input('Ingrese una edad: '))  
    if edad < 0:  
        dicCategorias['No válido'] += 1  
    elif edad <= 12:  
        dicCategorias['Niño'] += 1  
    elif edad <= 17:  
        dicCategorias['Adolescente'] += 1  
    elif edad <= 64:  
        dicCategorias['Adulto'] += 1  
    else:  
        dicCategorias['Adulto Mayor'] += 1  
    continuar = input("Desea continuar? s/n") == 's'
```



```
print("Edades ingresadas clasificadas:")  
for categoria in dicCategorias.items():  
    print(f'{categoria[0]}: {categoria[1]}')
```

definimos un diccionario con categorías fijas para contabilizar las edades ingresadas

solicitamos una edad

contabilizamos la edad según su categoría

preguntamos si se desea continuar

imprimimos los resultados usando salida con formato e interpolación de variables

Espacio práctico (tarea) – Sesión 1

¡Ahora inténtalo tú!


- Ejercicio propuesto: crea una lista de nombres y muestra aquellos que comiencen con una letra específica.



Espacio práctico (tarea) – Sesión 1

- Solución:

```
lista_nombres = ['Juan', 'Pedro', 'María', 'Roberto',  
                 'Carla', 'Luis', 'Bernardo', 'Javier', 'Rocío']  
continuar = True  
  
while continuar:  
    letra = input('Ingrese una letra: ').upper()  
    contador = 0  
    for nombre in lista_nombres:  
        if (nombre.startswith(letra)):  
            contador += 1  
            print('- ' + nombre)  
  
    if contador == 0:  
        print('No se encontraron nombres con la inicial proporcionada.')  
    else:  
        print(f'{contador} nombres encontrados.')  
  
    continuar = input("Desea continuar? s/n: ") == 's'  
  
print('Adiós!')
```



Cierre - Sesión 01

- Completa el diagrama sobre los tipos de datos en Python.



Cierre - Sesión 01

- Menciona algunas similitudes/diferencias entre una lista y un diccionario.

Lista	Diccionario

Cierre - Sesión 01

- Describe cómo se utilizan las estructuras condicionales y repetitivas en Python.

if

while

for



Gracias por su atención