UNIDAD 1. INTRODUCCIÓN A PYTHON

Loreto Pelegrín Castillo

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
Región de Murcia
Sef
Servicio Regional DE EMPLEO y FORMACIÓN
AVANZA

Índice:

1

Introducción

1. Introducción

- ¿Qué es Python?

- Es un lenguaje de programación muy popular y versátil, conocido por su sintaxis clara y fácil de leer, lo que lo hace ideal tanto para principiantes como para programadores experimentados.
- ¿Qué lo hace especial?
  - Legibilidad: Su código se parece mucho al lenguaje natural, lo que facilita su comprensión y escritura.

1. Introducción

- Versatilidad: Se utiliza en una amplia gama de aplicaciones, desde desarrollo web y análisis de datos hasta inteligencia artificial y aprendizaje automático.
- Amplia comunidad: Cuenta con una gran comunidad de usuarios que comparten recursos, bibliotecas y soluciones.
- Open source: Es gratuito y de código abierto, lo que significa que puedes usarlo y modificarlo libremente.

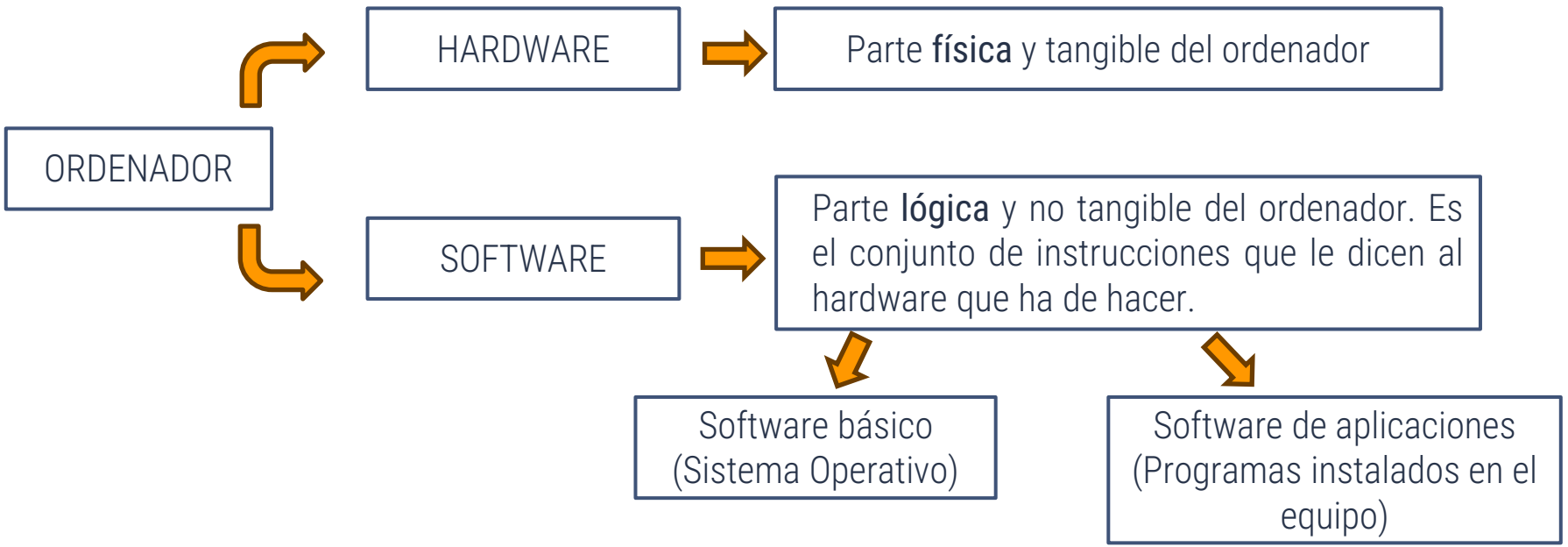
1. Introducción

- https://docs.python.org/es/3/tutorial/
- https://survey.stackoverflow.co/2024/technology#most-popular-technologies-language
- https://www.tiobe.com/tiobe-index/

2

Fundamentos de la programación

2. Fundamentos de la programación



2. Fundamentos de la programación

Conceptos:

- **Algoritmo:** Conjunto de pasos a seguir para solucionar un problema especifico, de manera secuencial y finita. Que sea secuencial implica que el conjunto de pasos debe tener un orden bien determinado.

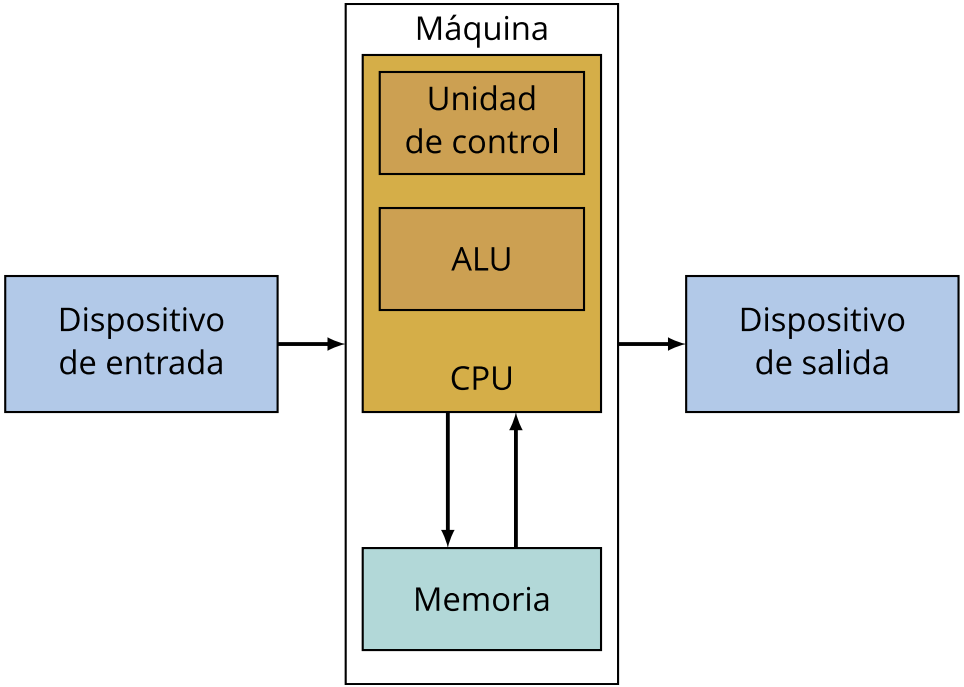
- **Programa:** Uso de lenguajes informáticos para implementar algoritmos en un ordenador.

2. Fundamentos de la programación

Conceptos:

- Características de un programa:
  - Conjunto finito de líneas.
  - Modificación fácil.
  - **Eficiente,** ejecución rápida y que ocupe el menor espacio posible en la memoria del ordenador.
  - **Bien estructurado**, facilita la lectura, implementación y modificación.

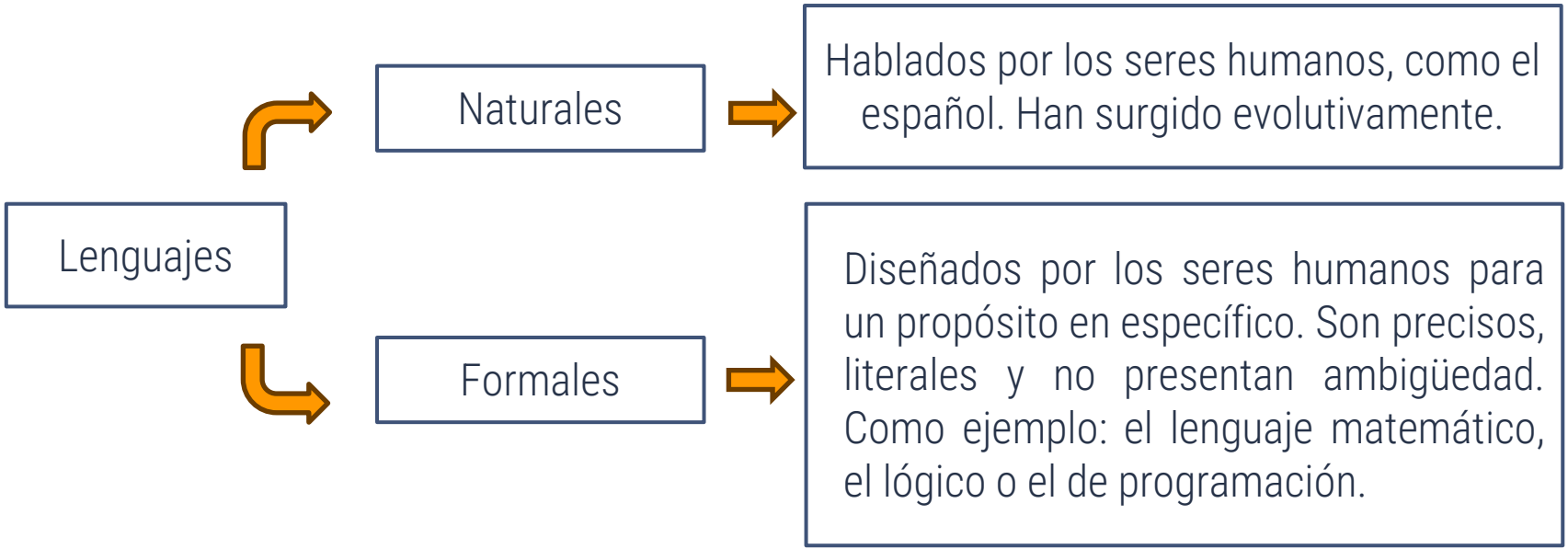
2.1. Arquitectura de un ordenador



La arquitectura de un ordenador muestra como están distribuidos internamente los elementos físicos para que un ordenador funcione.

Todos los ordenadores se construyen siguiendo patrones, el mas conocido: **Arquitectura de Von Neumann.**

2.2. Lenguajes de programación



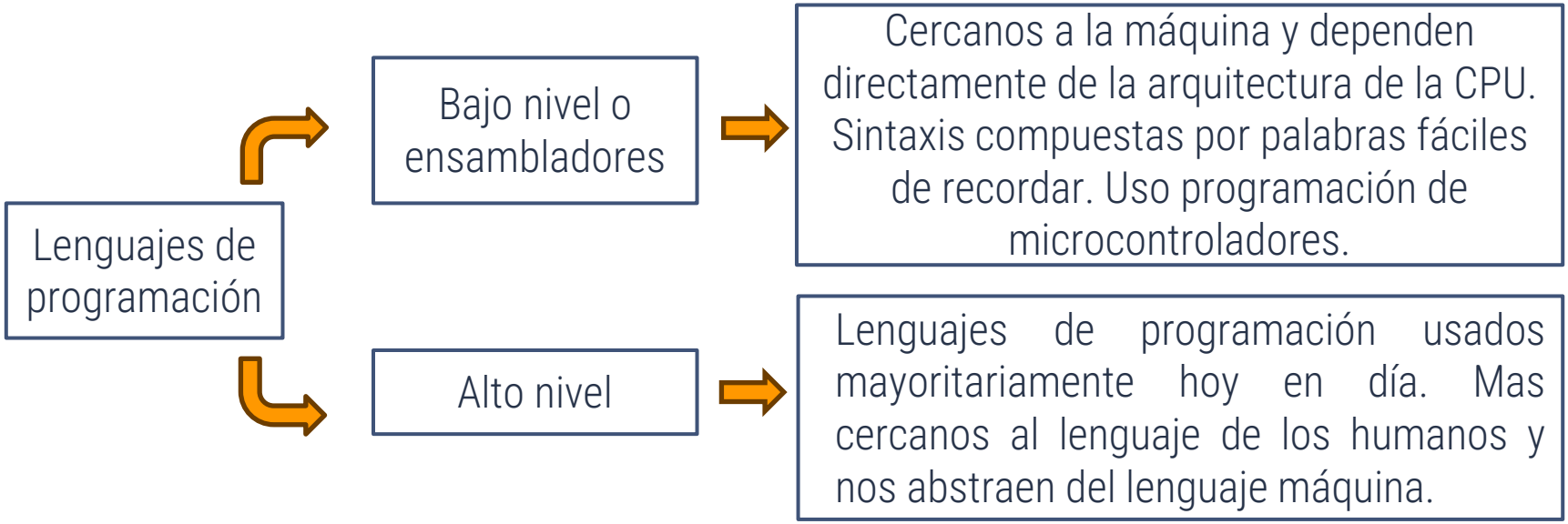
2.2. Lenguajes de programación

Todos los lenguajes de programación terminan traducido a lo que se conoce como **lenguaje máquina.** Este es el único lenguaje de la CPU que los ordenadores pueden entender. Se compone de secuencias de bits representadas por cadenas de 0 y 1.

Lenguaje de máquina - tabla binaria.

11001010	00010111	11110101	00101011
00010111	11110101	00101011	00101011
11001010	00010111	11110101	00101011
00010111	11110101	00101011	00101011
11001010	11110101	00101011	00101011
11001010	11001010	11110101	00101011
11001010	11110101	00101011	00101011
11001010	00010111	11110101	00101011
00010111	11110101	00101011	00101011
11001010	11110101	00101011	00101011

2.2. Lenguajes de programación



2.3. Pseudocódigo

- Los algoritmos son independientes de los lenguajes de programación en los que se implementan, por lo tanto, a la hora de diseñarlos se utiliza un lenguaje intermedio, llamado **pseudocódigo**.

- Es un lenguaje natural no tan preciso, pero que ya posee ciertas **palabras reservadas** con las que podemos presentar la información del conjunto de instrucciones que llevará a cabo el algoritmo como paso previo a su posterior programación en el lenguaje que se elija.

2.4. Diagramas de flujo

También es posible representar gráficamente los algoritmos, para eso tenemos diferentes técnicas, la mas conocida, **diagramas de flujo.**

Este tipo de diagramas se utilizan cajitas de varias formas. Dentro de esas cajas se escriben los pasos que seguir del algoritmo y se unen mediante flechas, que son las que representan el flujo que seguir por el programa.

2.5. Pseudocódigo y diagrama de flujo

Pseudocódigo:

```
INICIO
    Num1, Num2,Suma  : ENTERO
    ESCRIBA  "Diga dos números: "
    LEA Num1, Num2
    Suma ←   Num1 + Num2
    ESCRIBA "La Suma es:", Suma
FIN
```

Diagrama de flujo:

3

Empezando con Python

3. Empezando con Python

- Python fue creado por el programador Guido van Rossum, a finales de la década de los 80 y comienzos de los 90. Es un lenguaje que posee una sintaxis sencilla y muy legible.

- Es un lenguaje **orientado a objetos**, en el los problemas se tratan mediante un tipo de datos abstractos llamados objetos, que poseen unos atributos y comportamientos determinados.

3. Empezando con Python

- Se caracteriza por un tipado dinámico, es decir, a la hora de declarar una variable no es necesario declarar el tipo de dato, bien sea, numero, cadena de texto u otro tipo diferente.

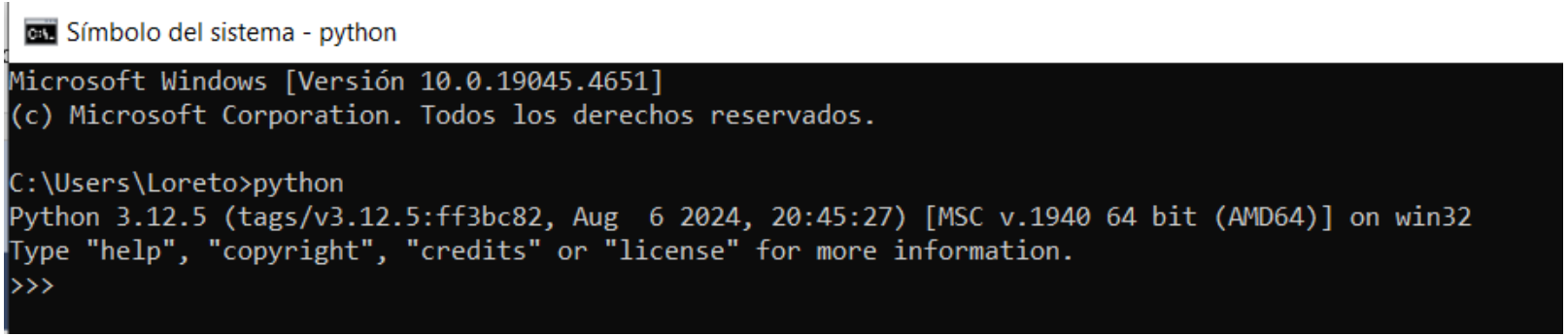
- Es de código abierto, lo que es posible manipularlo internamente y adaptarlo a nuestras necesidades.

3.1. Primeros pasos

- Instalar Python: https://www.python.org/downloads/

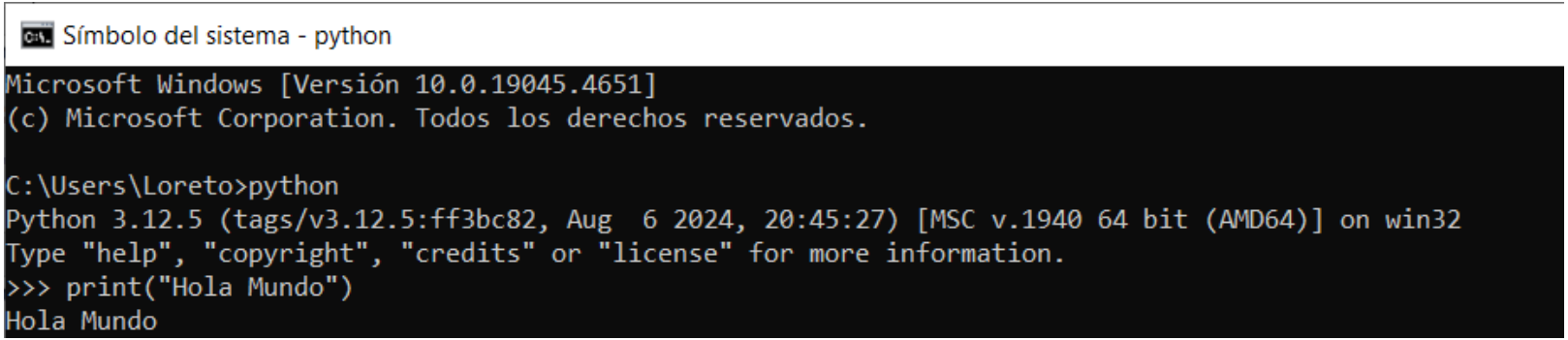
3.1. Primeros pasos

- Una vez instalado. Abrimos la consola de Windows, CMD.
- Escribimos Python. Nos debe salir >>>, promp de Python.
- Y ya podemos empezar a programar.



3.1. Primeros pasos

- El primer programa que se suele escribir en cualquier lenguaje de programación es: “Hola Mundo”.
- print(“Hola Mundo”)



3.1.1. Visual Studio Code

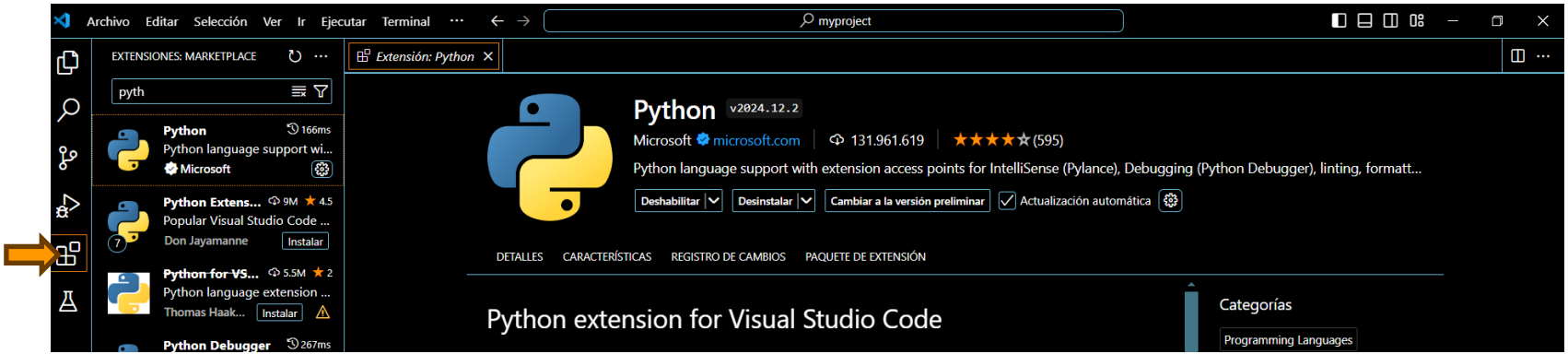
- La consola de Python es muy útil, pero para facilitar el trabajo se utilizan editores de código.

- En nuestro caso vamos a instalar : **Visual Studio Code**
  https://code.visualstudio.com/

- Una vez instalado el programa, hay que añadir diferentes extensiones para que se pueda programar en Python.

3.1.1. Visual Studio Code

- La primera extensión o plugin para poder programar en Python es el lenguaje en sin mismo:

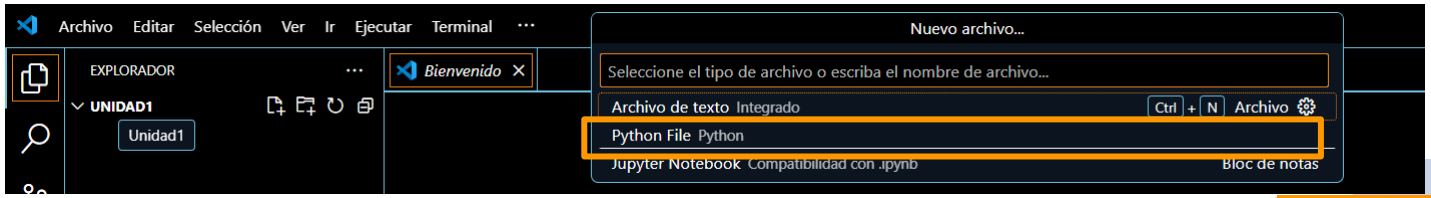


3.1.1. Visual Studio Code

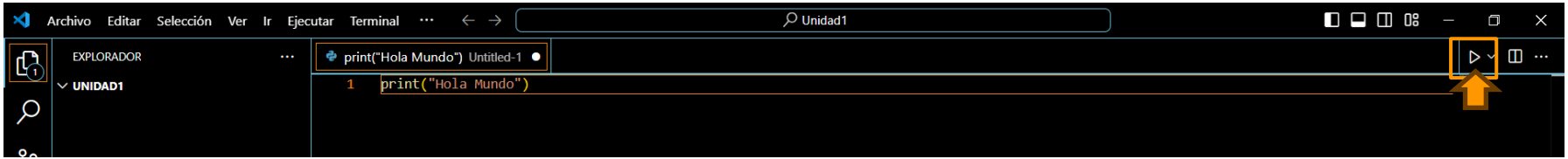
- Para ponerlo en español (opcional):

3.1.1. Visual Studio Code

- Como crear un programa en Visual Studio Code:
- Crear una carpeta, nombre que se desee, para poder almacenar los programas.
- En Visual Studio Code:
  - Archivo -> Abrir Carpeta (para asignar el directorio)
  - Archivo -> Nuevo Archivo, y seleccionamos Python File.



3.1.1. Visual Studio Code



- Para ejecutar el programa se pulsa en el Play de la parte superior derecha.
- Para que se ejecute se debe guardar el programa.

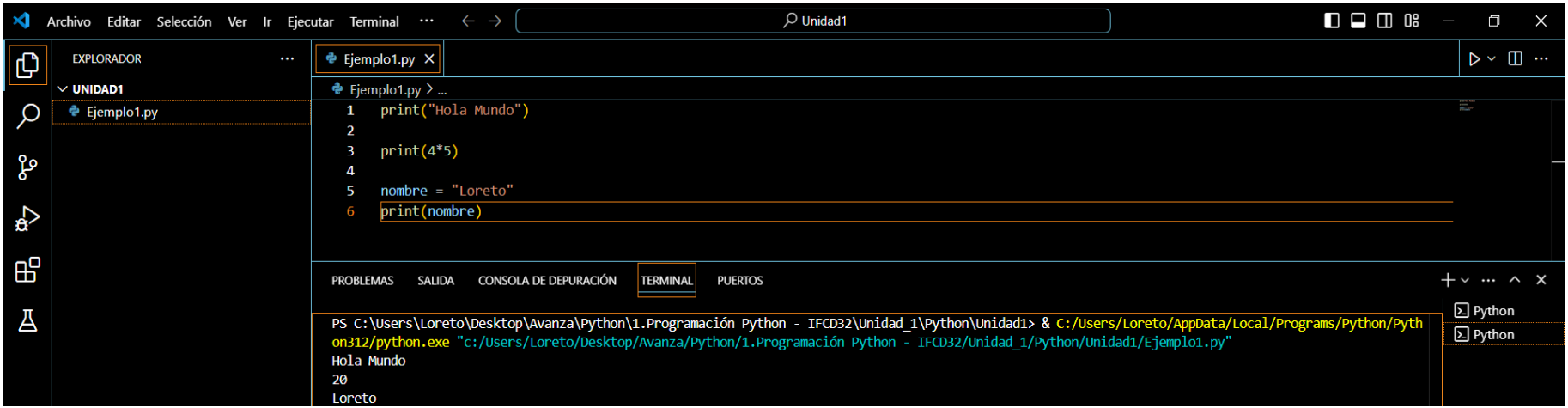
3.1.1. Visual Studio Code

- Para poder ejecutar el archivo o programa que hayamos creado, es necesario guardarlo.
- La extensión de un archivo Python es **.py**
- Archivo -> Guardar o Guardar como.
- Guardado rápido: la tecla **Ctrl** y la tecla **s** a la vez.

3.2. Print()

- El comando **print()** es una función especial de Python que nos va a permitir mostrar por pantalla de la consola líneas de texto, el contenido de las variables y las expresiones a evaluar.
- Prueba en Visual Studio Code:
  - print(4*5)
  - nombre = “Loreto”
  - print(nombre)

3.2. Print()


Archivo Editar Selección Ver Ir Ejecutar Terminal
Unidad1
EXPLORADOR
UNIDAD1
Ejemplo1.py
Ejemplo1.py
Ejemplo1.py > ...
1 print("Hola Mundo")
2
3 print(4*5)
4
5 nombre = "Loreto"
6 print(nombre)
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS
PS C:\Users\Loreto\Desktop\Avanza\Python\1.Programación Python - IFCD32\Unidad_1\Python\Unidad1> & C:/Users/Loreto/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/Loreto/Desktop/Avanza/Python/1.Programación Python - IFCD32/Unidad_1/Python/Unidad1/Ejemplo1.py"
Hola Mundo
20
Loreto
Python
Python

3.3. Cometarios

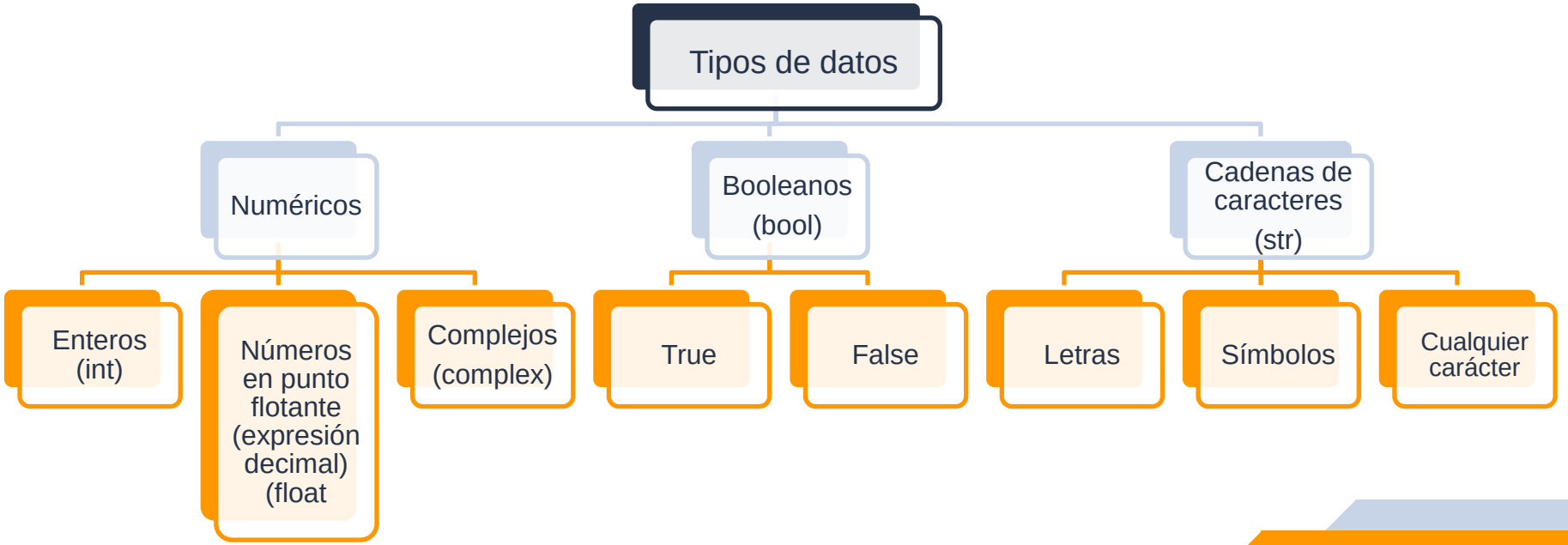
- Una de las mejores practicas que podemos realizar como programadores es **comentar nuestro código**.
- Consiste en hacer anotaciones en el código que nos permite explicar el funcionamiento de este.
- En Python hay 2 tipos de comentarios:
  - De una línea:

```
#Comentario de una linea
```

  - Varias líneas:

```
"""Cometario de
varias lineas """
```

3.4. Tipos de datos



3.4. Tipos de datos

- Para conocer el tipo de dato de un valor en Python, esta la función type(), la cual indicará la clase de datos a la que el valor pertenece.
- Probar:
  - type(3)
  - type(8.12)
  - type(1+2j)
  - type(True)
  - type(mesa)

Comprobación de que un valor entero es diferente a un tipo diferente al carácter:

type(5) == type(“5”)

3.4.1. Casting o conversión de tipos

- Esta operación consiste en convertir voluntariamente un tipos de dato en otro. A veces puede conllevar una perdida de información.

- Para realizarlo usaremos funciones con el tipo de dato al que vamos a convertir el dato que introduciremos entre paréntesis:

```
"""Conversion de un punto flotante o float en un entero o int.
Se pierde información ya que solo coge la parte entera."""
numero = int(8.945)
print(numero)
```

3.4.1. Casting o conversión de tipos

```
#Conversion de un entero en float
numero1 = float(8)
print(numero1)
```

```
#Conversion de un numero en una cadena
cadena = str(9876)
print(cadena)
```

Aplicación Practica

- Una empresa nos pide crear un programa que almacene los datos de los clientes: El nombre, la edad, si esta casado o no, el número de hijos y el gasto mensual. ¿Qué tipo de datos se usaría para cada dato?

Solución

- Elegir los tipos de datos es importante, ya que nos va a condicionar el tipo de operaciones que vamos a llevar a cabo.

- Para el nombre del cliente: cadena de caracteres o string.
- Edad y Número de hijos: enteros.
- Gasto medio mensual: float
- Cliente Casado: Boolean, ya que solo puede admitir 2 respuestas, si o no.

3.5. Variables y palabras reservadas

- **Variable:** espacio reservado en la memoria del ordenador, en el cual que se almacenará un valor determinado. Ese valor podrá cambiar durante el tiempo de ejecución del programa.
- En Python, las variables se crean cuando se definen por primera vez, es decir, cuando se les asigna el primer valor. Para crear una variable se utiliza el signo =
  - primer_dia = “Lunes”
- Se puede asignar varias variables en una misma línea:
  - a,b,c = 1, “hola”, 5.5

3.5. Variables y palabras reservadas

- **Constante**: es una variable definida con un valor asignado, y que no se modifica durante el tiempo de ejecución.
- En Python no existen las contantes, por lo tanto si se quiere crear un valor fijo, se declarará una variable en **letras mayúsculas**.
  - PI = 3.14159

3.5. Variables y palabras reservadas

- Para pedir información al usuario y guardarla en una variable, se puede usar la función **input(),** esta le pide el dato por consola y guarda el valor que se le introduzca en la variable.

- Esta función guarda el valor como una **cadena o string,** para cambiar a otro tipo de dato hay que realizar un casting.

```
#Uso de input para guardar datos
nombre = input("¿Como te llamas? ")
print("Mi nombre es: "+nombre)
```

3.5. Variables y palabras reservadas

- Python es un lenguaje muy flexible a la hora de asignar nombres a las variables y constantes. Pero existe en la comunidad de Python un conjunto de pautas en común que se siguen a la hora de nombrar variables, comentar código, …, no es obligatorio, es recomendable seguirlas.

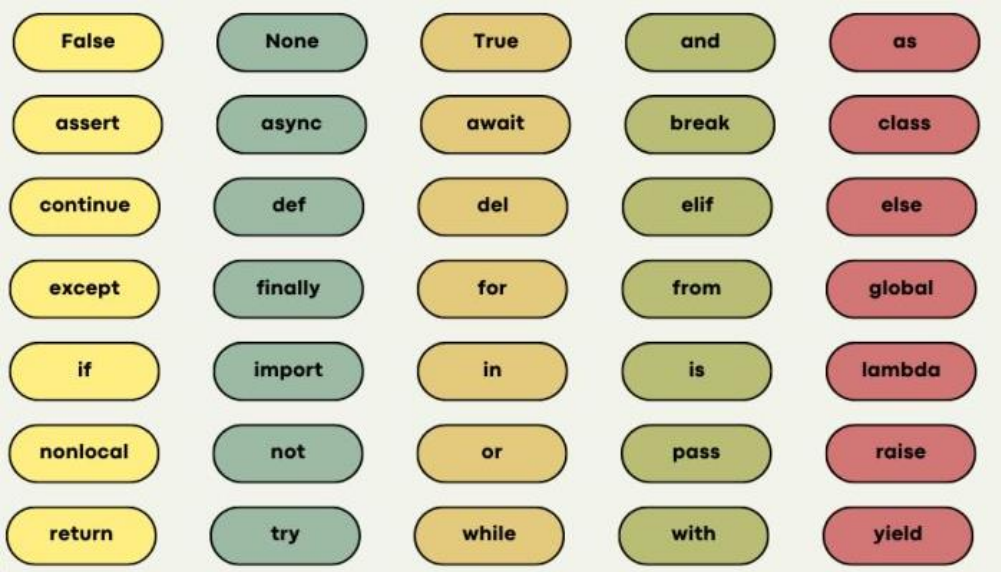
- A este conjunto de normas se las conoce como “guía de estilo PEP 8”.

3.5. Variables y palabras reservadas

- Hay determinadas cosas que no están permitidas en Python, y nos dará error si las realizamos:
  - Python es case sensitive, es decir, distingue entre mayúsculas y minúsculas. Para Python no es lo mismo la variable Nombre que nombre.
  - Se pueden usar números y letras en el nombre de la variable, pero siempre deben comenzar por una letra.
  - Se puede nombrar una variable comenzando por guion bajo: _nombre.

3.5. Variables y palabras reservadas

- No usar tildes en el nombre de las variables.
- No se pueden usar las palabras reservadas del lenguaje, ya que dará error.



3.6. Expresiones y operadores

- **Sentencia**: es una instrucción que el interprete de Python puede ejecutar, por ejemplo, cuando asignamos un valor a una variable.
- **Expresión:** Es conjunto de variables, operadores y valores. Al introducirla en la consola, Python evalúa y muestra un valor.

```
"""Evaluación de la expresion con variables,
valores y operadores"""
valor = 98
(valor + 56)*63
```

3.6. Expresiones y operadores

- **Operadores**: Símbolos especiales que ejecutan algún tipo de calculo computacional. Se clasifican en:
  - **Operadores de comparación:** Permiten comparar 2 o mas valores. El resultado siempre será True o False.

```
x == x #x es igual a x
x != y #x es distinto de y
x > y  #x es mayor que y
x < y  #x es menor que y
x >= y #x es mayor o igual que y
x <= y #x es menor o igual que y
```

```
#Ejemplos
"hola" == "HOLA"  #False
"telefono" != "Telefono"  #True
78 > 48 #True
8 <= 8  #True
```

3.6. Expresiones y operadores

- **Operadores aritméticos:** Permiten realizar operaciones aritméticas entre los operandos.

```
x + y  #Operador suma
x - y  #Operador resta
x * y  #Operador de multiplicacion
x / y  #Operador de division exacta
x // y #Operador de division entera
x ** y #Operador de potencia
x % y  #Operador de modulo (devuelve el resto de una division)
```

3.6. Expresiones y operadores

- **Operadores lógicos:** Permiten operar con valores booleanos, operaciones aritméticas entre los operandos.
  - **Operador and**: devuelve true cuando ambas condiciones se cumplen, en cualquier otro caso devuelve false.

```
True and False  #False
True and True   #True
False and False #False
```

3.6. Expresiones y operadores

- **Operador or**: devuelve true cuando una o ambas condiciones se cumple. Solamente devolverá False cuando ambas condiciones no se cumplan.

```
True or False  #True
True or True   #True
False or False #False
```

- **Operador not:** devuelve el valor opuesto al valor booleano que negamos.

```
not True  #False
not False #True
```

3.7. Estructuras de control

Identación: Es una especie de sangría que se deja en los lenguajes de programación para estructurar mejor el código escrito y para visualmente se puedan ver más fácilmente los bloques de código.

No en todos los lenguajes es obligatoria la identación, pero si recomendado. Pero en el caso de **Python si es obligatorio**, ya que no utiliza elementos de delimitación, llaves o puntos.

La identación en Python suele ser de 4 espacios en blanco. (Una tabulación)

3.7. Estructuras de control

Estructuras de control: Son bloques de código que nos permiten agrupar un conjunto de instrucciones determinadas y guiar el flujo del programa.

- En Python existen las siguientes estructura de control:
  - Estructuras condicionales
  - Estructuras repetitivas
  - Otros (break, continue)

3.7.1. Estructuras condicionales

- Es el tipo de estructura que evalúa una condición que dar como resultado **True o False**, a partir de ese resultado booleano el programa tomará un camino u otro.

- Para las estructuras condiciones se usan las palabras reservadas:
  - if (si)
  - else (sino)
  - elif (sino con condición)

3.7.1. Estructuras condicionales

- La palabra reservada **if** permite introducir una condición que de ser cierta tomará el valor True y hará que se ejecute el código que hay dentro del bloque.
- La condición se debe escribir entre paréntesis. Ejemplo:

```
edad = 20 #Se le asigna el valor 20 a la variable edad

"""La condicion consiste en que la variable edad tenga un valor
superior a 18, esta condicion se cumple y por lo tanto se
ejecuta lo que hay dentro del bloque de codigo"""

if(edad > 18):
    print("Eres mayor de edad")
```

3.7.1. Estructuras condicionales

- La palabra reservada **else** se usa tras el **if,** esta permite que, si la condición del bloque perteneciente al if no se ejecutara porque no es cierta, pasa directamente al bloque else y ejecuta esa instrucción. No es necesario escribir ninguna condición en el bloque else.

```
edad = 15 #Se le asigna el valor 15 a la variable edad

"""En este caso no se cumple la condicion del bloque if,
por lo tanto se ejecutará la condición del bloque else"""

if(edad > 18):
    print("Eres mayor de edad")
else:
    print("Todavia eres menor de edad")
```

3.7.1. Estructuras condicionales

- La palabra reservada **elif** permite introducir una o varias condiciones tras el **if**.

```
numero = -6 #Se le asigna un valor negativo a numero

""" Se va a comprobar que el numero sea positivo, negativo o 0,
en este caso el numero es negativo"""
if numero > 0:
    print("El número es positivo.")
elif numero < 0:
    print("El número es negativo.")
else:
    print("El número es cero.")
```

3.7.1. Estructuras condicionales

- La palabra reservada **elif** permite introducir una o varias condiciones tras el **if**. En un mismo bloque solo puede haber un if y else seguidos, pero puede haber tanto elif como se necesiten.

```
numero = -6 #Se le asigna un valor negativo a numero

""" Se va a comprobar que el numero sea positivo, negativo o 0,
en este caso el numero es negativo"""
if numero > 0:
    print("El número es positivo.")
elif numero < 0:
    print("El número es negativo.")
else:
    print("El número es cero.")
```

3.7.1. Estructuras condicionales

- Bloques anidados: Bloque de sentencias condicionales dentro de otro

```
nota = 8.9
if(nota == 10):
    print ("Matricula de honor")
else:
    if( nota <5):
        print("Estas Suspenso")
    elif(nota <= 6.9 and  nota >= 5):
        print("Estas aprobado")
    elif(nota <= 7.9 and nota >= 6):
        print("Tienes un bien")
    elif(nota <= 8.9 and nota >= 7 ):
        print("Tienes un notable")
    else:
        print("Sobresaliente")
```

1º Bloque

2º Bloque

3.7.1. Estructuras condicionales

- Realizar los ejercicios de estructuras condicionales.
  - Crear una carpeta llamada Condicional-NombreAlumno.
  - Dentro de esa carpeta, crear un archivo para cada ejercicio.
  - Comprimir la carpeta y subir a la tarea llamada: Ejercicios Estructuras condicionales.

3.7.2. Estructuras repetitivas

- Este tipo de estructuras también llamadas **Bucles,** nos permiten ejecutar un bloque de código varias veces.

- El bloque se sigue ejecutando mientras se siga cumpliendo la condición que se ha establecido.
- Hay 2 palabras reservadas para este tipo de estructuras:
  - for
  - while

3.7.2. Estructuras repetitivas

- **For:** Repetir un bloque un número predeterminado de veces.
- Cada repetición se llama Iteración.

```
for variable in iterable:
        cuerpo del buque
```

- Ej: Tenemos una caja de bombones. El bucle for sería una instrucción que te dice:
- Coge un bombón, cómelo.
- Así hasta que se acabe la caja.

3.7.2. Estructuras repetitivas

- **Range():** Función que permite crear una secuencia de números que van desde 0 por defecto hasta el número que se pasa como parámetro menos 1.
  - Ej: Range(6): Crea una secuencia de números del 0 al 5.
- Va de la mano con el bucle for:

```
#Este bucle imprimirá 3 veces "Buenos dias"
for i in range(3):
    print("Buenas dias")
```

```
#Muestra por pantalla del 1 al 5
for numero in range(1, 6):
    print(numero)
```

3.7.2. Estructuras repetitivas

- Se pueden pasar hasta tres parámetros separados por coma, donde el primer es el inicio de la secuencia, el segundo el final y el tercero el salto que se desea entre números. Por defecto se empieza en 0 y el salto es de 1.
- range(inicio, fin, salto)

```
for i in range(5, 20, 2):
    print(i) #5,7,9,11,13,15,17,19
```

```
for i in range (5, 0, -1):
    print(i) #5,4,3,2,1
```

3.7.2. Estructuras repetitivas

- **Nota:** cuando se utiliza print(), al final de la línea se agrega un salto de línea automáticamente.
- Sin embargo, el argumento end permite personalizar este comportamiento.
- Por ejemplo: Al especificar end=", ", indicamos que en lugar de un salto de línea, se debe imprimir una coma y un espacio. Esto es útil cuando queremos construir una cadena de texto de forma más personalizada.

```
print(i, end=", ")
```

3.7.2. Estructuras repetitivas

- Ejercicios de clase:
- 1. Escribir un programa que pida al usuario una palabra y la muestre 10 veces por pantalla.
- 2. Escribir un programa que pida al usuario un número entero positivo y muestre por pantalla todos los números impares desde 1 hasta ese número separados por comas.
- 3. Escribir un programa que pida al usuario un número entero positivo y muestre por pantalla la cuenta atrás desde ese número hasta cero separados por comas.

3.7.2. Estructuras repetitivas

- **while:** Se usará cuando se quiera repetir un bucle **desconociendo las iteraciones**. Estas se repetirán mientras la condición que se establezca se siga cumpliendo.

```
numero = 9
while(numero > 0):
    print(numero)
    numero = numero -1
```

```
while True:
    print("Bucle infinito")
```

3.7.2. Estructuras repetitivas

- Es posible tener un while en una sola línea, algo muy útil si el bloque que queremos ejecutar es corto.
- En el caso de tener mas de una sentencia, las debemos separar con ;.

```
x = 5
while x > 0: x-=1; print(x)
```

3.7.2. Estructuras repetitivas

- Else y while
- Algo no muy corriente en otros lenguajes de programación pero si en Python, es el uso de la cláusula else al final del while.

```
x = 5
while x > 0:
    x -=1
    print(x) #4,3,2,1,0
else:
    print("El bucle ha finalizado")
```

3.7.2. Estructuras repetitivas

- Ejercicios de clase:
- 1. Escribir un programa que almacene la cadena de caracteres contraseña en una variable, pregunte al usuario por la contraseña hasta que introduzca la contraseña correcta.
- 2. Calcular el factorial de un número
- 3. Sucesión de Fibonacci en Python.
  - En matemáticas, la sucesión de Fibonacci es una sucesión infinita de números naturales, donde el siguiente numero se consigue sumando los dos anteriores.

3.7.3. Otros (break, continue)

- **Break:** Permite salir de un bloque de código si se cumple una determinada condición y pasa a la siguente instrucción. Se puede usar con los bucles for y while.

```
valor = 5
"""En este caso cuando la variable valor sea igual
a 5 se saldrá del bucle y se pasará a la siguente
instruccion del programa si la hubiera"""

while (valor >0):
    if(valor== 5):
        break
    print(valor)
    valor = valor - 1
```

3.7.3. Otros (break, continue)

- **Continue:** Permite saltarse una iteración sin salir del bucle, devuelve la posición al principio del bucle y se sigue ejecutando hasta que termina.
- Se puede usar tanto para for como while.

```
"""Cuando el valor sea igual a 3 se saltará este paso
Se devuelvo el control al inicio del bucle
Se seguirá ejecutando el bucle"""
for valor in range(7):
    if(valor==3):
        continue
    print(valor)
```

¡Gracias!