**UNIDAD 1 - Paradigma Estructurado**

**Itinerario 1**

¿Qué es un **algoritmo**?

Conjunto ordenado de pasos que permiten obtener un resultado o encontrar la solución a un problema.

Todos los algoritmos deben cumplir tres premisas:

1. **PRECISO** (Orden)

Implica el orden de realización de cada uno de los pasos.

1. **DEFINIDO** (Resultado)

Si se sigue dos veces, se obtiene el mismo resultado.

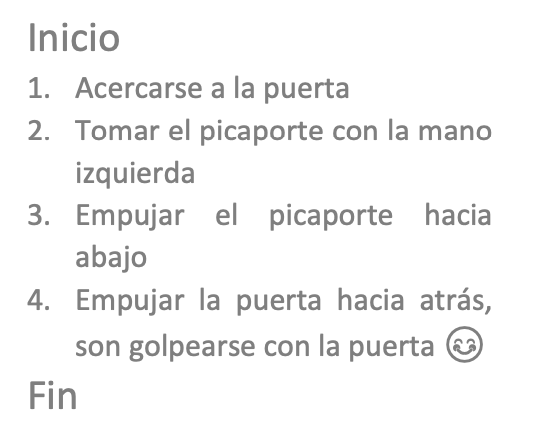
1. **FINITO**

Tiene un número determinado de pasos, implica que tiene un fin.

PSEUDOCÓDIGO

Es una manera de expresar los diferentes pasos de una solución usando una forma más amigable.

Ejemplo:



TÉCNICAS ALGORÍTMICAS

La **ALGORITMIA** es una técnica utilizada para resolver problemas mediante un conjunto de procesos mentales. Consiste en implementar pasos los cuales están validados.

1. ***Enumeración Exhaustiva***: Consiste en probar cada opción posible de manera secuencial. Este método evalúa si cada opción satisface las condiciones del problema planteado hasta encontrar la solución correcta.

Ejemplo: Algoritmo => Recorrer un laberinto hasta salir.

**Inicio**

1- Repetir todo hasta encontrar una salida.

a. Caminar hacia delante mientras no encuentres una abertura para doblar a la izquierda.

i. Si encuentra una salida, girar a la izquierda.

ii. Si se encuentra al final de un camino y no puede avanzar, girar 90 grados.

**Fin**

1. ***Algoritmo de Aproximación***: Es una propuesta para subsanar el costo del método anterior. El objetivo es el mismo pero mejor rendimiento.

Este método proporciona una solución aproximada en lugar de intentar una solución exacta.

El algoritmo comienza con una solución inicial aleatoria y luego itera a través de un conjunto de vecinos de la solución actual. Para cada vecino, el algoritmo comprueba si es mejor que la solución actual. Si es así, el algoritmo reemplaza la solución actual con el vecino.

Ejemplo: Algoritmo => Calcular raíz cuadrada de un número.

**Inicio**

1- Leer número N

2- Leer un valor inicial aproximado A para la raíz cuadrada.

3- Definir una precisión deseada, epsilon.

4- Mientras la diferencia absoluta entre A al cuadrado y N sea mayor que epsilon, repetir:

5- Calcular una nueva aproximación, A, como el promedio entre A y N/A (A = + N/A) / 2)

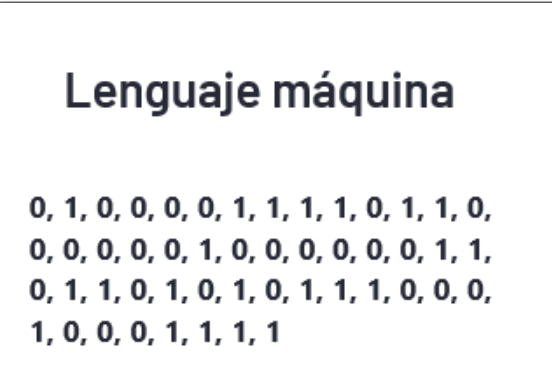
6- Mostrar el valor aproximado de la raíz cuadrada de N: A

**Fin**

Clasificación de los lenguajes de **programación**



Según el estilo de la sintaxis del lenguaje se pueden clasificar en **Bajos** (cercanos a la PC), **Medios** y **Altos** (cercanos al humano).

1. **Lenguaje Máquina**

El lenguaje de la computadora es el código binario. Este es el único lenguaje que la PC entiende.

1. **Lenguaje de Alto Nivel**

Ejemplos: Python, C#, etc.

Propósito: Desarrollar sin un alto grado de dependencia con el hardware que se utilice.

Los lenguajes de alto nivel se deben traducir a lenguaje máquina. Este proceso se denomina compilación o interpretación del código fuente.

* **Interpretado**: Proceso de traducción a lenguaje máquina se hace cada vez que se ejecuta. No tengo un .EXE. Ejemplo: JavaScript.
* **Compilado**: Proceso de traducción se hace por única vez, luego de obtener el .EXE ese archivo se usan sin necesidad de volver a compilar.

¿QUÉ TIENE UN LENGUAJE DE PROGRAMACIÓN?

1. Tipos de INSTRUCCIONES
2. **Secuenciales**: Instrucciones de entrada y salida
3. **Condicionales** (If, else if, switch): permite ,dada una condición, modificar el flujo de ejecución del programa.
4. De **repetición** (for, while, do while): bloques de código que podemos ejecutar varias veces dependiendo de una condición.

PARADIGMAS DE PROGRAMACIÓN

1. **PROGRAMACIÓN ESTRUCTURADA**: También conocido como paradigma *imperativo*. Es un tipo de programación que se define mediante subrutinas y flujos de control.
2. **PROGRAMACIÓN OO**: En este paradigma se construyen modelos de objetos considerando un nivel de abstracción según el problema a resolver.
3. **PROGRAMACIÓN FUNCIONAL**: Es un paradigma *declarativo*. En este paradigma las funciones tienen un rol protagónico, y podrán ser asignadas a variables y ser usadas como entradas y salidas de otras funciones.

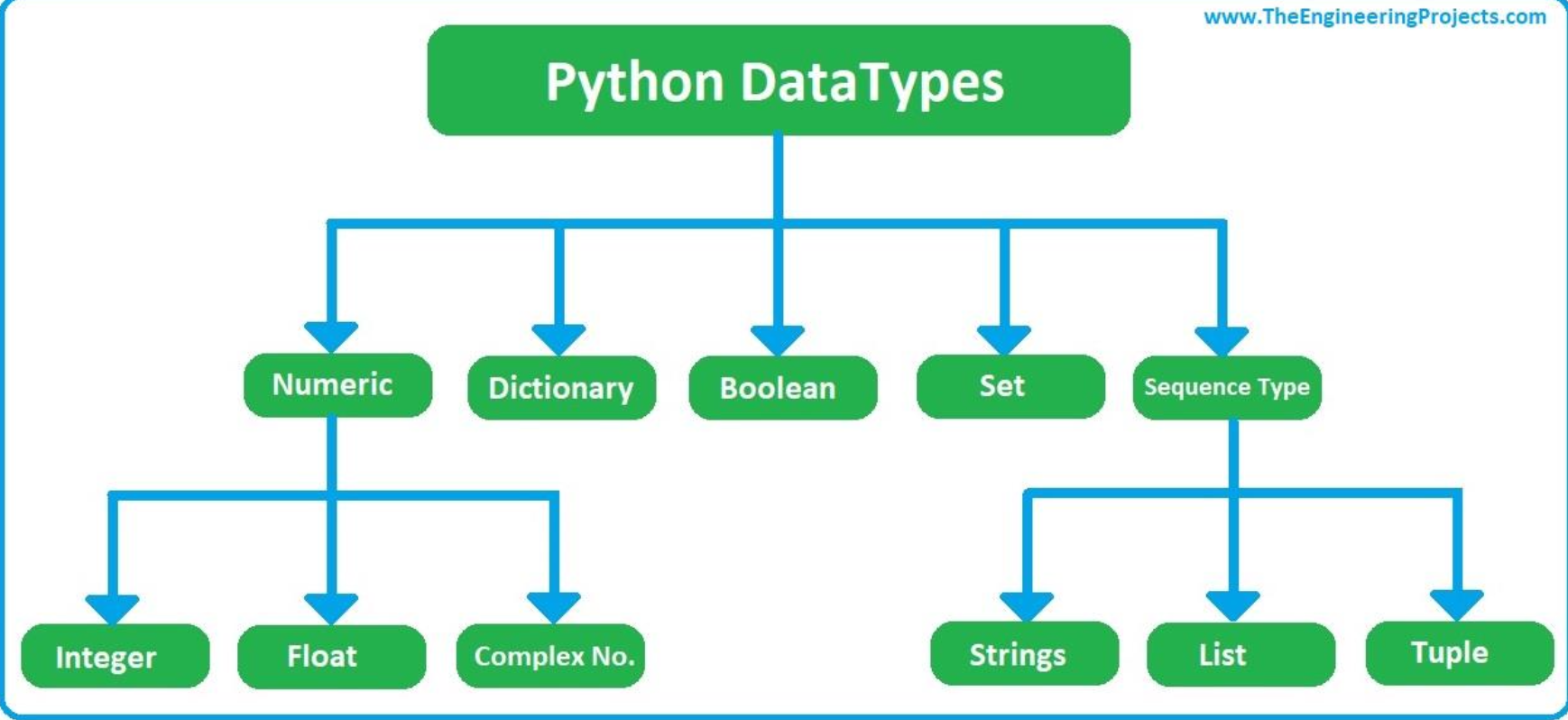
**Itinerario 2**

**Variables**

Una **variable** es un espacio en memoria el cual nos permite identificar un dato que no conocemos y que luego usaremos en el algoritmo.

Cada variable guarda un sólo dato a la vez.

TIPOS DE DATOS



Lista: Se usa [] y es **mutable**.

Tupla: Se usa () y es **inmutable**.

CONVENCIONES

1. **Constantes**: No existe este tipo en Python, pero para indicar que el valor de la variable no va cambiar, pondremos el nombre en mayuscula.

Ejemplo: **PI = 3.14**

1. Los nombres de las variables deben ser representativos.

Ejemplo: Mal uso -> aaa = 12 // Buen uso -> edad = 12

1. El nombre debe comenzar con una letra, o con un guion bajo. Usaremos el **\_** para métodos/variables **privadas**.

Ejemplo: **nombre= “Pablo”** // **\_nombre = “Pablo”**

1. Si el nombre tiene más de una palabra, no puede tener espacios en blanco:

Ejemplo: Mal uso -> mi nombre = “Pablo”

1. No usar palabras reservadas del lenguaje:

Ejemplo: Mal uso -> print = “Hola”

1. Formato ***lowerCamelCase***: Usado para variables. Ejemplo: **nombreDePila** = “Pablo”
2. Formato ***snake\_case:***  Usado para funciones o métodos. Ejemplo: **mi\_funcion()**:
3. Formato ***ScreamingSnakeCase***: Usado para las clases y constantes. Ejemplo: **BorrarPantalla.**

**Asignación de valores a Variables**

| a, b, c = 1, 2, 3 | Se asigna secuencialmente un valor a cada variable. |
| --- | --- |
| a = b = c = 10 | Asignamos 10 a todas las variables. |

CONVERSIÓN DE VARIABLES

Convertir el valor de una variable, es cuando queremos cambiar un valor de un tipo de dato a otro.

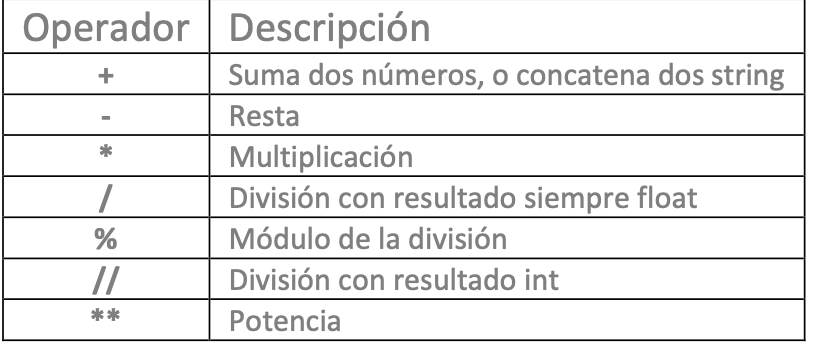
| **str()** | Devuelve un string. |
| --- | --- |
| **int()** | Devuelve un entero |
| **float()** | Devuelve un flotante. |
| **Bool()** | Devuelve un booleano. |

En Python existe dos tipos de conversiones:



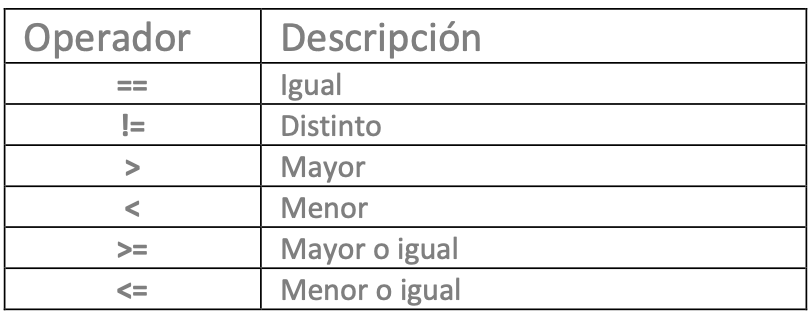
OPERADORES ARITMÉTICOS

Permite hacer operaciones aritméticas con las variables.

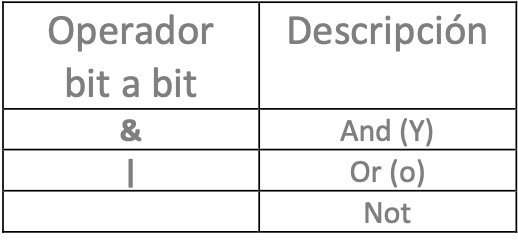


OPERADORES RELACIONALES

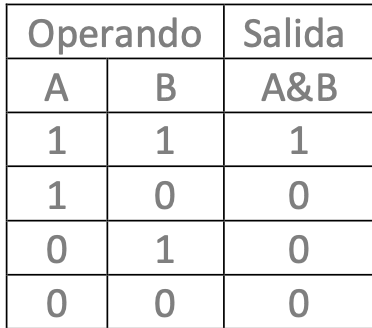
Se utiliza para comparar dos valores y devolver True o False como resultado.



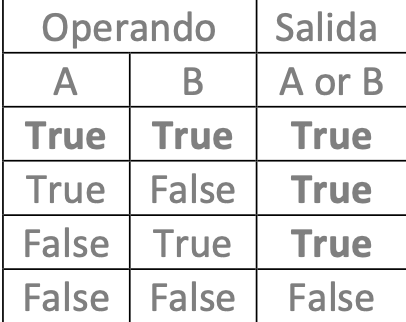
OPERADORES BINARIOS



1. **Tabla de verdad** **&**



1. **Tabla de verdad |**

****

**Comentarios**

1. **De una sola línea**

Ejemplo: # Esto es un comentario

1. **Multilínea**

Ejemplo: ‘’’ Esto es un comentario

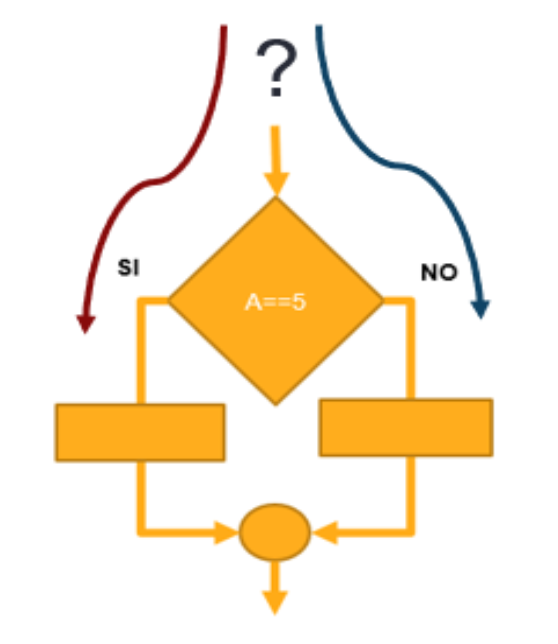
multilínea ‘’’

**Conjunto de Instrucciones**

Todo lenguaje de programación debe tener instrucciones **secuenciales**, de **decisión** y de **repetición**.

En todos los lenguajes tenemos dos tipos de instrucciones: simples y compuestas.

1. **INSTRUCCIONES DE DECISIÓN**: Se usan en todos los programas para crear caminos alternativos en el flujo de ejecución del código.

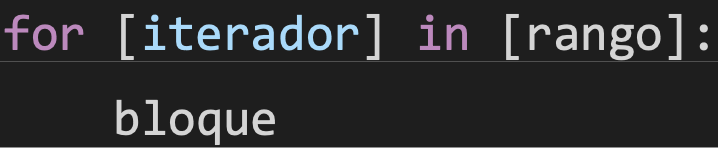


**Itinerario 4 - FUNCIONES**

INSTRUCCION FOR

Instrucción que repite una línea o un bloque de líneas una cantidad finita de veces.

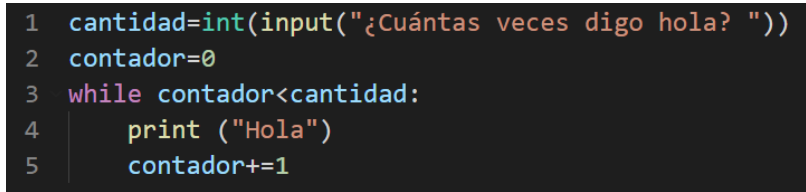
Esta instrucción necesita que se le indique cuantas veces se repite el código. Ejemplo:



El ***iterador*** es una variable que representa, en cada vuelta, el elemento que forma parte de la lista del rango expresado.

INSTRUCCION WHILE

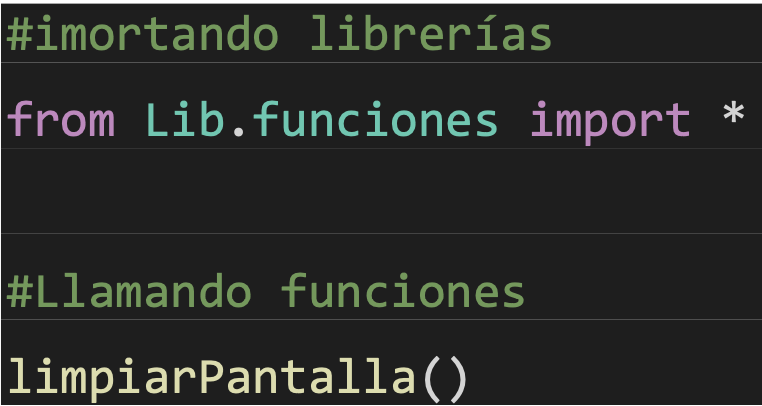
Instrucción que se usa cuando se necesita repetir un bloque de código con un número desconocido de repeticiones. Ejemplo:



**Itinerario 5 - ARCHIVOS, EXCEPCIONES Y EXPRESIONES REGULARES**

**Módulos**

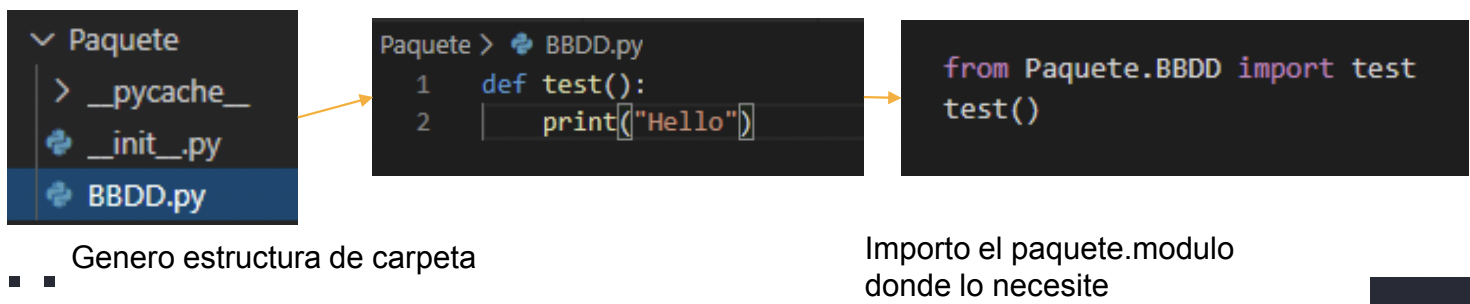
Son entidades que permiten una división lógica de nuestro código para facilitar el mantenimiento y la lectura. En Python, los módulos son objetos de tipo *module*.



Debemos tener una precaución para que esto funcione. El primer archivo de la carpeta debe ser un archivo vacío llamado **\_\_init\_\_.py.** Esta carpeta es un paquete.

**Paquetes**

Los paquetes sirven para organizar los módulos. También son de tipo *module*. Se representan mediante directorios.



**Excepciones**

Las excepciones son errores detectados por Python durante la ejecución del programa.

EXCEPTION

Son los errores personalizados que tiene Python. Python proporciona una lista de excepciones estándares como:

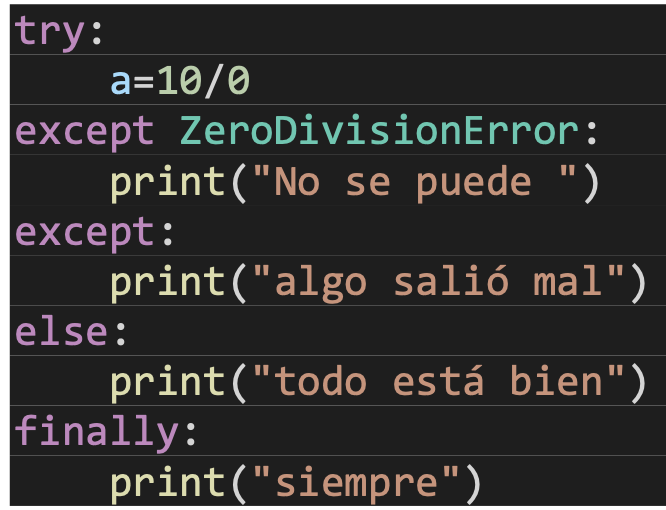
* **Exception**: es la base de todas las excepciones.
* Arithmetic Error: clase base para todos los errores que se producen en un cálculo aritmético.
* FloatingPointError: se genera cuando falla un cálculo de punto flotante.
* ZeroDivisionError: se produce cuando se hace una división por cero.
* EOFError: error cuando se alcanza el final del archivo.
* ImportError: Cuando falla la declaración del import.
* NameError: se genera cuando no se encuentra el nombre del namespace local o global.
* IOError: cuando falla la operación de entrada y salida.
* ValueError: cuando tiene el tipo de valor de argumento incorrecto.

TRY-EXCEPT

En Python, se utiliza el operador **Try** para controlar las excepciones en un bloque de código. Cuando se produce un error, lo atrapa el **except**. En este último, se describe el tratamiento que se le dará al error. Ejemplo:



Por último, tenemos la cláusula **finally**. Se utiliza para ejecutar un código al terminar el try se haya encontrado un error o no. Ejemplo:



**UNIDAD 2 - Paradigma OO**

**Itinerario 8**

La programación OO permite mejorar la legibilidad y además disminuir la cantidad de errores. Por otro lado, facilita la reutilización del código y su escalabilidad. Un ejemplo de lenguaje orientado a objetos: C#, Python, etc.

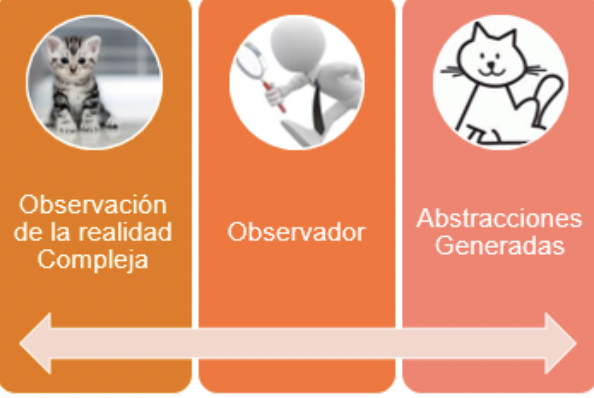
Existen 4 conceptos fundamentales en el paradigma OO:

* Abstracción de datos.
* Encapsulación
* Herencia
* Polimorfismo

PILARES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

La POO se basa en la creación de objetos, entidades que encapsulan datos y comportamiento, imitando el funcionamiento del mundo real.

* **ABSTRACCIÓN**: Cuando decimos abstracción, en el paradigma OO, hablamos de *clases*. Estas abstracciones, representan tipos de entidades.

Ejemplo:

* **ENCAPSULACIÓN**: Cuando decimos encapsulación, el código queda escondido dentro de los objetos. Significa que podemos interactuar con el objeto, pero no sabemos cómo resuelve dicha petición.
* **INSTANCIA**: La acción de crear un objeto se llama *instanciar*. Cuando tenemos la clase, podemos crear todos los objetos necesarios.

CLASES Y OBJETOS

Las clases serán *únicas* pero los objetos sí pueden repetirse. A partir de las clases que programemos, podemos crear los objetos según sea necesario.

**DEFINIR UNA CLASE**

Las clases son descripciones de una abstracción de elementos similares. Dicha descripción consta de expresar las características comunes que observamos. Las acciones que estos elementos pueden implementar, se programan como parte del comportamiento.

Ejemplo:

Clase => Animal

Atributos => - Color - Cantidad de Patas - Altura - Peso

Métodos => - Correr - Saltar - Comer

INTERFAZ O CONTRATO

Al detallar las características y funciones de la clase, podemos especificar su alcance. Es decir, cuáles son privadas y cuáles son públicas. Todo lo público puede ser accesible desde fuera del modelo. Lo que describimos como características y funciones es la *interfaz* (medio a través del cual podemos comunicarnos con el modelo).

Decimos que es un contrato porque, al querer comunicarnos con este modelo, debemos utilizar su interfaz.

**DEFINIR UN OBJETO**

Un objeto es algo que representa una idea, y que además tiene identidad única. También, es una instancia de una clase.



ESTADO

Es el conjunto de todas las propiedades y los valores para cada una de estas propiedades. En general, todos los objetos tienen las mismas propiedades, pero cambian sus valores. Todos los objetos son instancia de una clase, pero no son el mismo objeto.

COMPORTAMIENTO

Es todo lo que el objeto puede hacer.

IDENTIDAD

Cada instancia de un objeto es única. Los valores que toman las propiedades no es la manera de diferenciar objetos.

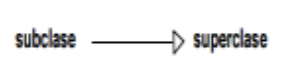
RELACIONES ENTRE CLASES

Es la manera como los elementos pueden colaborar entre sí. Para simplificar, sólo se presentan dos tipos de relaciones: la *herencia* y *asociación*.

1. ***HERENCIA***

Una clase nueva se crea a partir de una existente.

Mediante la herencia, las instancias de una clase hija (ó subclase) pueden acceder tanto a los atributos como a los métodos públicos y protegidos de la clase padre (ó superclase).



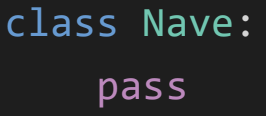
1. ***ASOCIACIÓN***

Se establece cuando dos clases tienen una dependencia de utilización, es decir cuando una clase utiliza atributos y/o métodos de otra para funcionar. Estas dos clases no necesariamente están en jerarquía. La asociación expresa un tipo de colaboración entre ambos elementos. Esto último, también se lo suele llamar “**una relación de uso**”.

CLASES Y OBJETOS EN PYTHON

**CREAR UNA CLASE**

Como ocurre con las variables, debemos definirlas. La palabra reservada que se usa es **class**.



(**pass** es una palabra reservada que se usa para indicar que la clase está vacía).

INSTANCIA DE LA CLASE

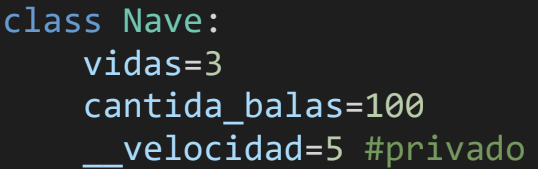
**CREAR UN OBJETO**

Se llama **instancia**, y el código es el siguiente:



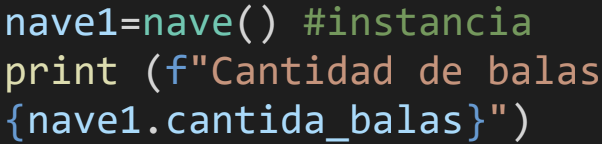
**ATRIBUTOS DE UNA CLASE**

Son las propiedades, es decir la estructura que lo define. Es un par de nombre y valor.



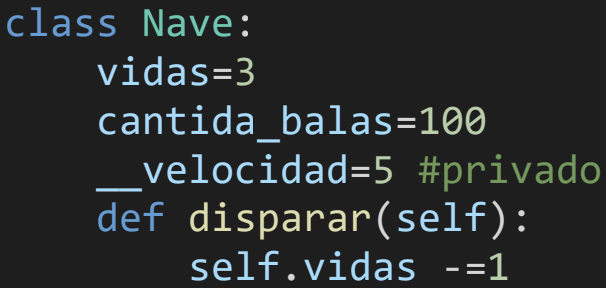
El alcance del atributo determina si se puede ver desde el exterior. En el caso de Python, no podemos definir el alcance como privado, pero podemos emular al anteponer un **doble guión bajo**.

Para utilizar sus atributos, solo navegamos su interfaz:



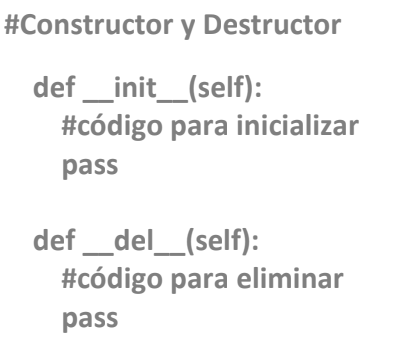
**MÉTODOS DE UNA CLASE**

Se programan como si fuesen funciones, con el agregado de que siempre debe tener por lo menos el parámetro **self**.



**MÉTODO CONSTRUCTOR Y MÉTODO DESTRUCTOR**

Son métodos especiales que se ejecutan en primer lugar cuando se instancia el objeto, y en último lugar cuando se destruye el objeto.



**UNIDAD 3 - Numpy, Pandas y MatplotLib**

**Itinerario 11 - Librería Pandas y MatPlotLib**

LIBRERÍA PANDAS

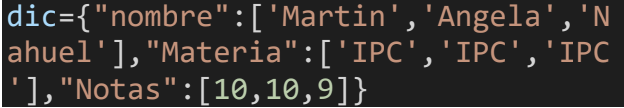
Pandas es una librería que permite trabajar con datos.

**DATAFRAME**

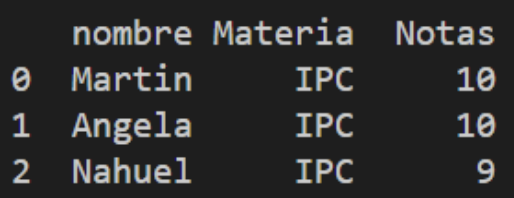
Es un conjunto de datos que se organiza como tablas, ordenado en filas y columnas. Tienen dos índices. Al estar organizado como matrices, tienen uno para filas y otro para columnas.

**CREACIÓN DE UN DATAFRAME**

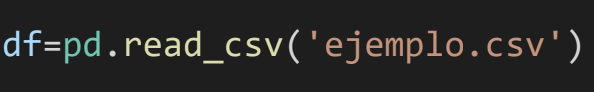
Para crear un DF, primero hay que crear un diccionario, donde el índice corresponde a una columna de la tabla y los datos en cada elemento de las filas.







**CREACIÓN DE UN DF DESDE UN ARCHIVO .CSL O EXCEL**

****

**NOTA:** El archivo debe estar en la misma carpeta sino hay que colocar el ***path*** al archivo.

**[DF].ILOC**

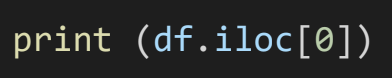
El acceso es tomando como referencia la ubicación del dato en la tabla. Dicha ubicación se representa por las filas y las columnas.

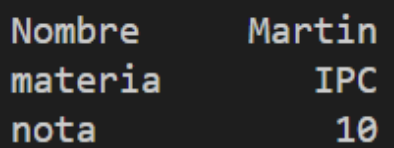
• **df.iloc[filas, columnas]** : devuelve el valor

de la celda

• **df.iloc[i]** : devuelve los datos de toda una

fila

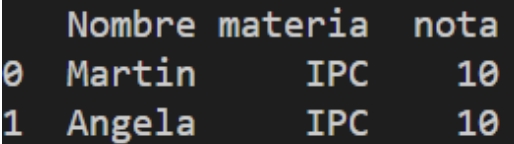




**[DF].LOC**

Devuelve los datos de una fila entera, también permite devolver los datos de más de una fila.

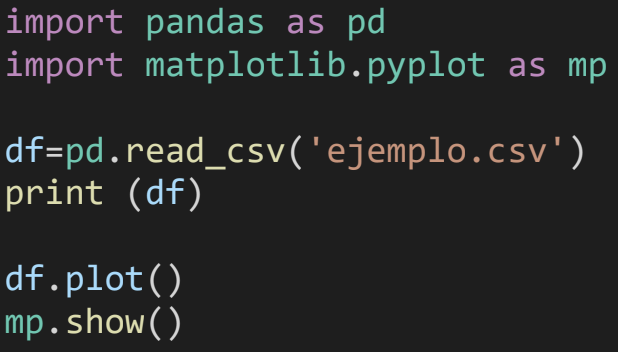


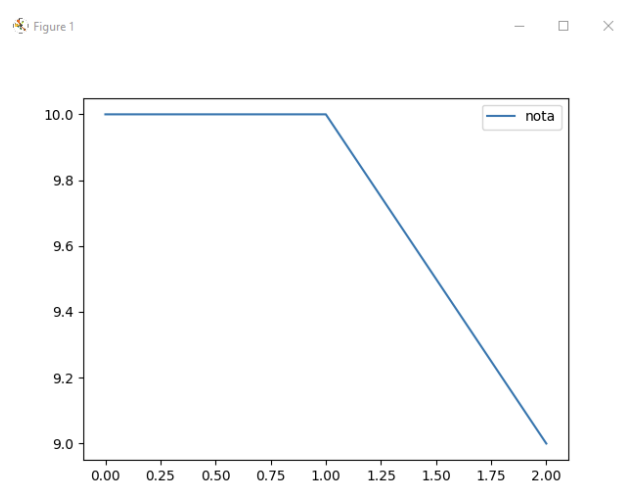


LIBRERÍA MATPLOTLIB

Es una librería de código abierto, que se puede usar para generar gráficos.

**CREACIÓN DE UN GRÁFICO**



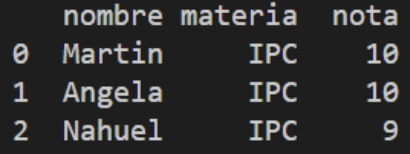


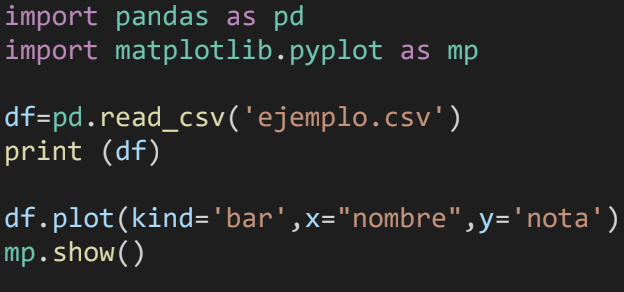
Luego, agregamos al plot tres palabras:

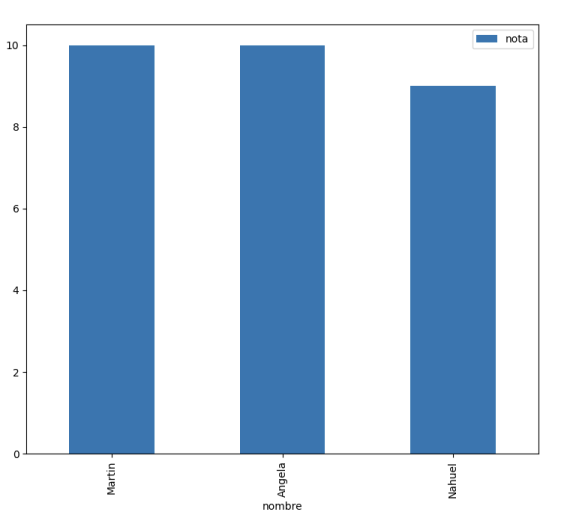
1. **Kind**: el tipo de grafico
2. **X**= valor del eje X
3. **Y**= valor del eje Y

En el siguiente código, creamos un grafico de barras con el nombre y las notas tomando el

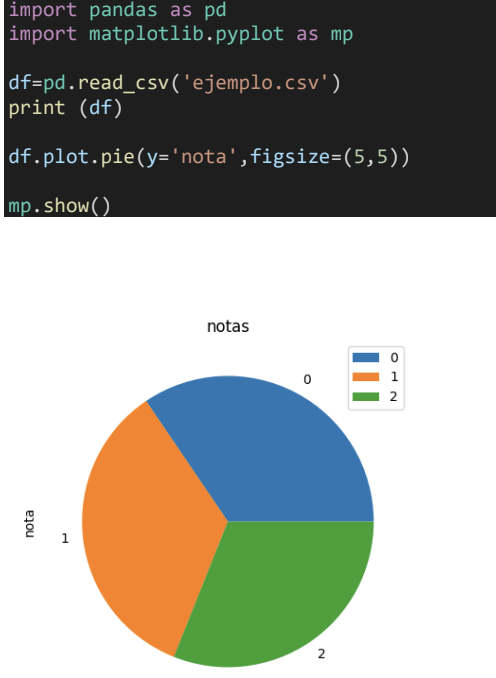
siguiente dataframe







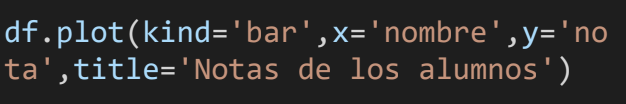
**CREACIÓN DE UN GRÁFICO DE TORTAS**



**PERSONALIZANDO LOS GRÁFICOS**

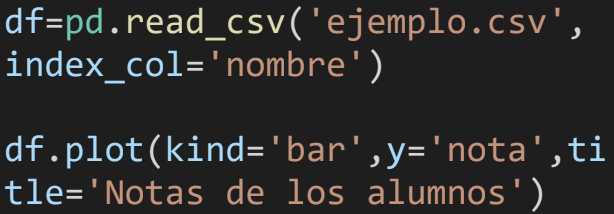
Se pueden modificar los gráficos para que la experiencia sea mas positiva.

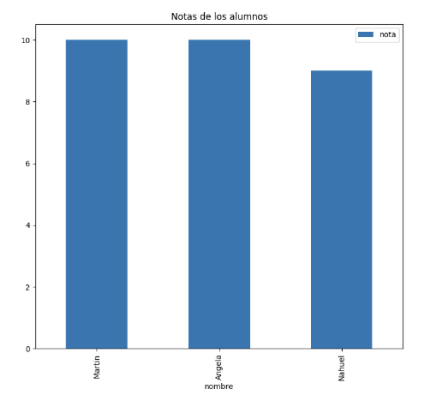
* Agregando títulos



IMPORTANDO CON INDEX

Podemos crear un index personalizado





* Cambiando el color

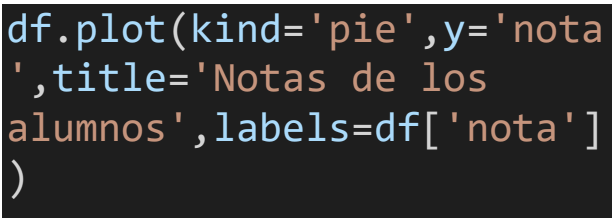
Se puede modificar el color de las barras y de las series



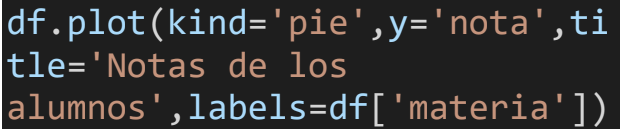
**PERSONALIZANDO LAS ETIQUETAS**

Podemos cambiar las Etiquetas por un dato:

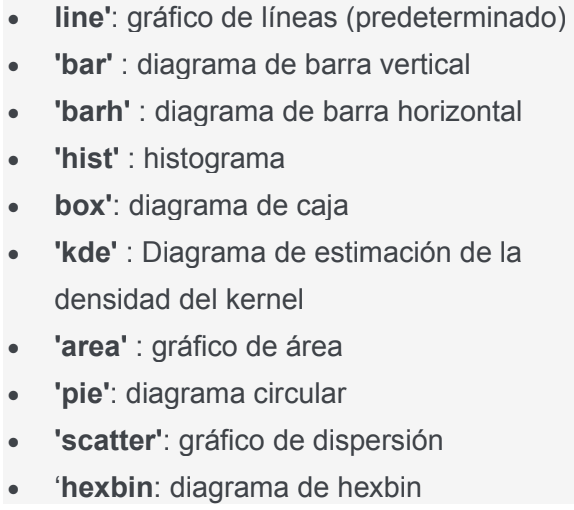
* La nota como etiqueta



* Las materias como etiqueta



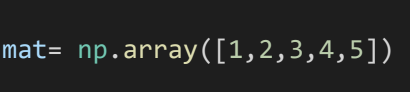
**TIPOS DE GRÁFICOS**



**Itinerario 12 - Librería NumPy**

**Numpy** (numerical Python) es una biblioteca que se utiliza para trabajar con datos númericos. Nos permite trabajar con matrices (estructura multidimensional).

**CREANDO UNA MATRIZ**



**ANALIZANDO UNA MATRIZ**

Se pueden conocer datos de la matriz como la cantidad de elementos o la cantidad de dimensiones que posee.

Métodos que brindan información de la matriz:

1. **TYPE**

¿Cómo saber si es una matriz?



1. **NDIM**

¿Cómo conocer las dimensiones que tiene la matriz?



1. **SHAPE**

¿Cómo saber la cantidad de elementos en cada dimensión?



**CREANDO MATRICES AUTOMÁTICAS**

1. **ZEROS Y ONES**

Se utiliza para crear una matriz y completar los elementos con ceros.



1. **EMPTY**

Se utiliza para crear una matriz vacía.



1. **DTYPE**

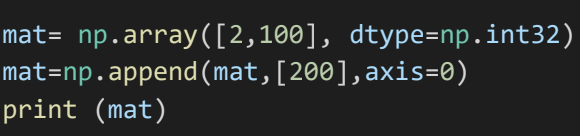
Se utiliza para crear una matriz especificando el tipo de dato.



**AGREGANDO Y ELIMINANDO DATOS DE LA MATRIZ**

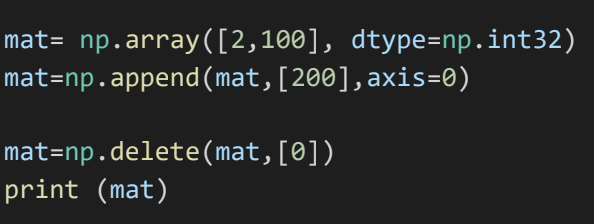
1. **APPEND**

Permite agregar un elemento al final de la matriz. La función append crea una nueva matriz.



1. **DELETE**

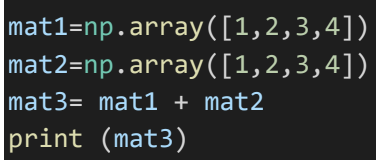
Retorna una nueva estructura con los datos eliminados.



**OPERACIONES ARITMÉTICAS CON MATRICES**

1. **OPERACIONES ( + - \* / )**

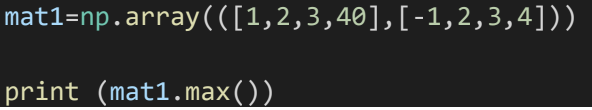
Se pueden sumar dos matrices como si fuesen variables independientes de un solo tipo de dato. También podemos restar, multiplicar y dividir de la misma manera.



**OPERACIONES AVANZADAS**

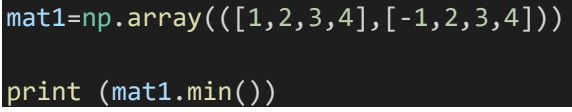
1. **MAX**

Devuelve el valor máximo de los elementos.



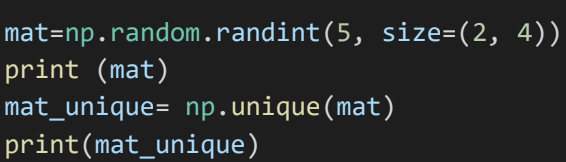
1. **MIN**

Devuelve el valor mínimo de los elementos.



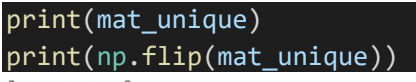
1. **UNIQUE**

Devuelve los valores únicos de la matriz.



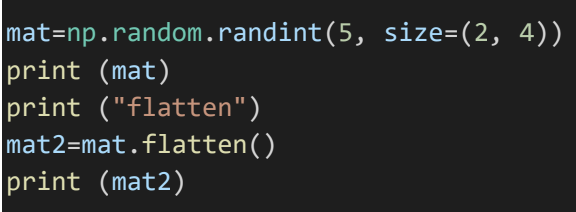
1. **FLIP**

Invierte la matriz.



1. **FLATTEN**

Convierte la matriz en una matriz de una dimensión.



1. **RANDOM**

Crea una matriz con valores randoms.

