Índice

[**Programación Orientada a Objetos en Python** 1](#_Toc48830838)

[**Tipos de datos abstractos y clases, Instancias** 5](#_Toc48830839)

[**Decomposición** 7](#_Toc48830840)

[**Abstracción** 7](#_Toc48830841)

[**Encapsulación, getters and setters** 7](#_Toc48830842)

[**Herencia** 8](#_Toc48830843)

[**Polimorfismo** 8](#_Toc48830844)

[**Introducción a la complejidad algorítmica** 9](#_Toc48830845)

[**Notación asintótica** 9](#_Toc48830846)

[**Clases de complejidad algorítmica** 10](#_Toc48830847)

[**Búsqueda lineal** 12](#_Toc48830848)

[**Búsqueda binaria** 12](#_Toc48830849)

[**Ordenamiento de burbuja** 12](#_Toc48830850)

[**Ordenamiento por inserción** 12](#_Toc48830851)

[**Ordenamiento por mezcla** 15](#_Toc48830852)

[**Ambientes virtuales** 15](#_Toc48830853)

[**¿Por qué graficar?** 17](#_Toc48830854)

[**Graficando simple** 18](#_Toc48830855)

[**Introducción a la optimización** 18](#_Toc48830856)

[**El problema del morral** 19](#_Toc48830857)

# **Programación Orientada a Objetos en Python**

Uno de los elementos más importantes de los lenguajes de programación es la utilización de clases para organizar programas en módulos y abstracciones de datos.

Las clases se pueden utilizar de muchas diversas maneras. Pero en este artículo hablaremos de cómo utilizarlas en el contexto de la programación orientada a objetos.

La clave para entender la programación orientada a objetos es pensar en objetos como agrupaciones de datos y los métodos que operan en dichos datos.

Por ejemplo, podemos representar a una persona con propiedades como nombre, edad, género, etc. y los comportamientos de dicha persona como caminar, cantar, comer, etc. De la misma manera podemos representar unos audífonos con propiedades como su marca, tamaño, color, etc. y sus comportamientos como reproducir música, pausar y avanzar a la siguiente canción.

Puesto de otra manera, la programación orientada a objetos nos permite modelar cosas reales y concretas del mundo y sus relaciones con otros objetos.

Las ideas detrás de la programación orientada a objetos tienen más de 50 años y han sido ampliamente aceptadas y practicadas en los últimos treinta. A mediados de la década de los setenta se comenzaron a escribir artículos académicos explicando los beneficios de esta aproximación a la programación. También durante esos años se comenzaron a escribir los primeros lenguajes de programación que incorporaban estas ideas (como Smalltalk y CLU). Pero no fue hasta la llegada de Java y C++ que la idea consiguió un número importante de seguidores.

Hasta ahora, en el curso previo de **Introducción al Pensamiento Computacional con Python** hemos utilizado programación orientada a objetos de manera implícita. Cuando decimos "Los objetos son las principales cosas que un programa de Python manipula. Cada objeto tiene un tipo que define qué cosas puede realizar un programa con dicho objeto.", nos estamos refiriendo a las ideas principales de la programación orientada a objetos. Hemos utilizado los tipos lista y diccionario, entre muchos otros, así como los métodos asociados a dichos tipos.

Así como los creadores de un lenguaje de programación sólo pueden diseñar una fracción muy pequeña de todas las funciones útiles (como abs, float, type, etc.), también pueden escribir una fracción muy pequeña de los tipos útiles (int, str, dict, list, etc.). Ya sabemos los mecanismos que nos permiten crear funciones, ahora veremos los mecanismos que nos permiten crear nuevos tipos (o clases).

**Clases en Python**

Las estructuras primitivas con las que hemos trabajado hasta ahora nos permiten definir cosas sencillas, como el costo de algo, el nombre de un usuario, las veces que debe correr un bucle, etc. Sin embargo, existen ocasiones cuando necesitamos definir estructuras más complejas, por ejemplo, un hotel. Podríamos utilizar dos listas: una para definir los cuartos y una segunda para definir si el cuarto se encuentra ocupado o no.



Sin embargo, este tipo de organización rápidamente se sale de control. ¿Qué tal que quisiéramos añadir más propiedades, cómo si el cuarto ya fue aseado o no? ¿Si el cuarto tiene cama doble o sencilla? Esto nos lleva a una falta fuerte de organización y es justamente el punto que justifica la existencia de clases.

Las clases nos permiten crear nuevos tipos que contiene información arbitraria sobre un objeto. En el caso del hotel, podríamos crear dos clases **Hotel()** y **Cuarto()** que nos permitiera dar seguimiento a las propiedades como número de cuartos, ocupación, aseo, tipo de habitación, etc.

Es importante resaltar que las clases sólo proveen estructura. Son un molde con el cual podemos construir objetos específicos. La clase señala las propiedades que los hoteles que modelemos tendrán, pero no es ningún hotel específico. Para eso necesitamos las instancias.

**Instancias**

Mientras que las clases proveen la estructura, las instancias son los objetos reales que creamos en nuestro programa: un hotel llamado **PlatziHotel** o **Hilton**.

Otra forma de pensarlo es que las clases son como un formulario y una vez que se llena cada copia del formulario tenemos las instancias que pertenecen a dicha clase. Cada copia puede tener datos distintos, al igual que cada instancia es distinta de las demás (aunque todas pertenecen a una misma clase).

Para definir una clase, simplemente utilizamos el keyword class. Por ejemplo:



Una vez que tenemos una clase llamada Hotel podemos generar una instancia

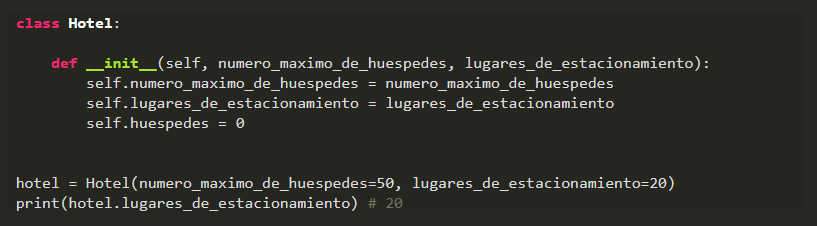
llamando al constructor de la clase.



**Atributos de la instancia**

Todas las clases crean objetos y todos los objetos tienen atributos. Utilizamos el método especial \_\_init\_\_ para definir el estado inicial de nuestra instancia.

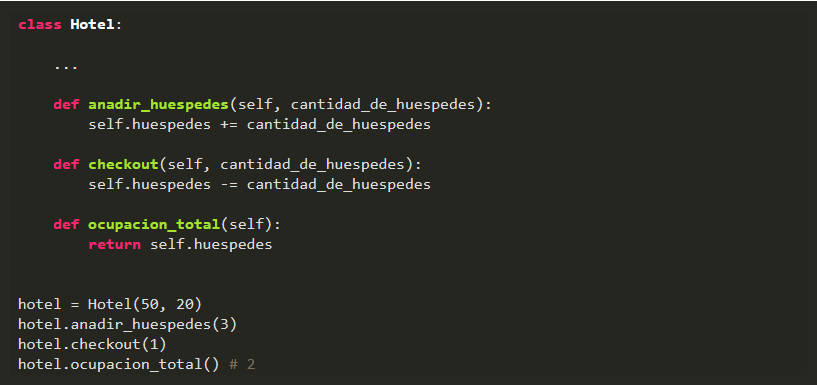
Recibe como primer parámetro obligatorio self (que es simplemente una referencia a la instancia).



**Métodos de instancia**

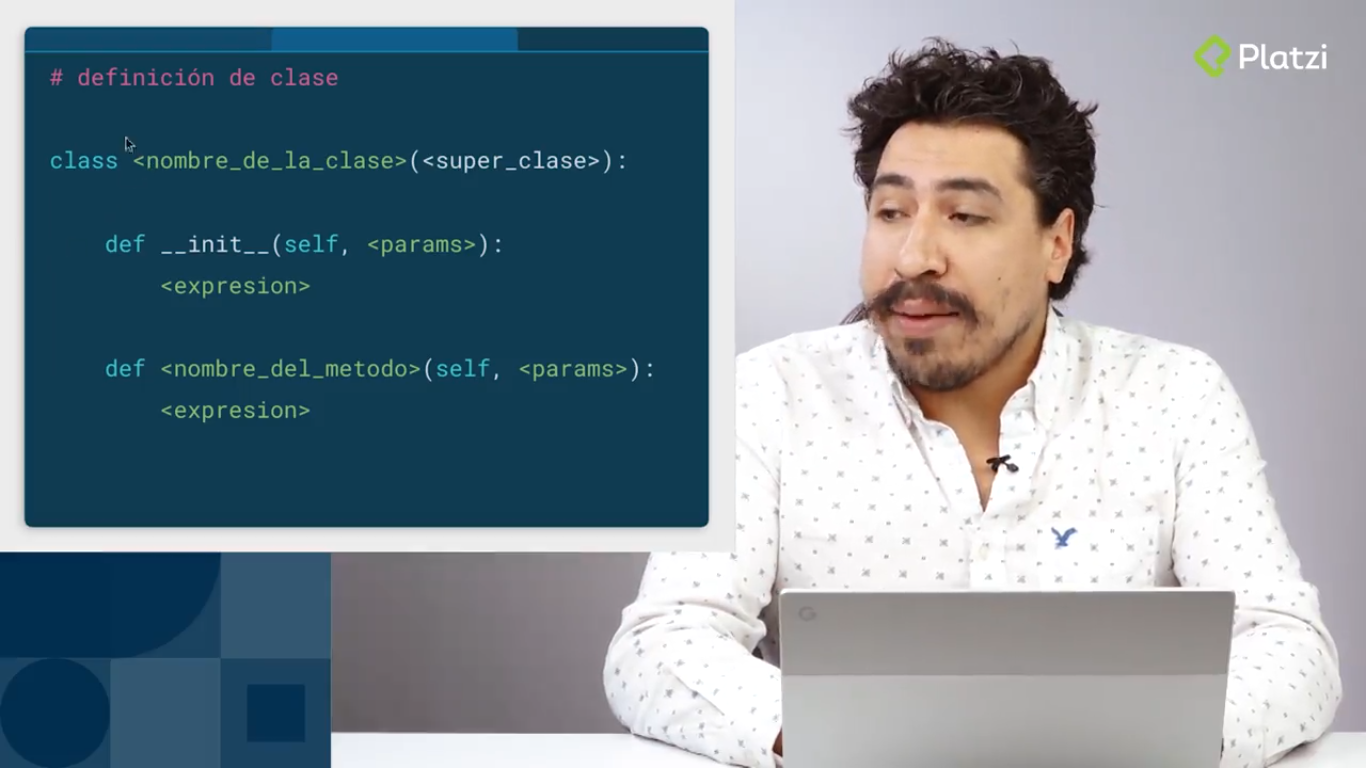
Mientras que los atributos de la instancia describen lo que representa el objeto, los métodos de instancia nos indican qué podemos hacer con las instancias de dicha clase y normalmente operan en los mencionados atributos.

Los métodos son equivalentes a funciones dentro de la definición de la clase, pero todos reciben self como primer argumento.



# **Tipos de datos abstractos y clases, Instancias**

* En python **todo** es un objeto y tiene un tipo.
  + Tiene una representación de datos en memoria y formas de interactuar con ellos.
* Formas de interactuar con un objeto
  + Creación.
  + Manipulación.
  + Destrucción.



**Instancias**

* Mientras que la clase es un molde, a los objetos creados se los conoce como instancias.
* Cuando se crea una instancia, se ejecuta el método **\_\_init\_\_**
* Todos los métodos de una clase reciben como primer parámetro **self**.
* Los atributos de clases nos permiten:
  + Representar datos.
  + Procedimientos para interactuar con los mismos (métodos)
  + Mecanismos para esconder la representación interna.
* Se accede a los atributos con la notación del punto.
* Puede tener atributos privados. Por convención comienzan con \_.

# **Decomposición**

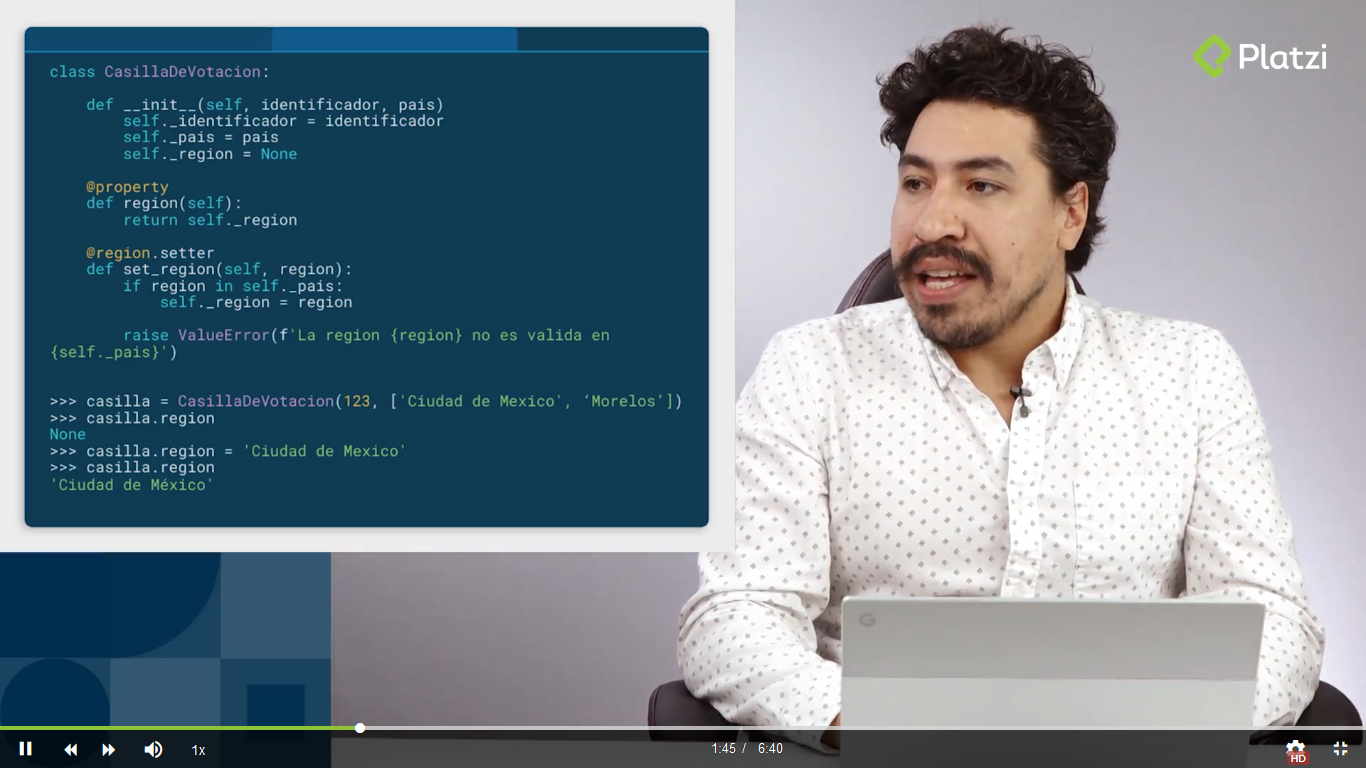
* Se trata de partir un programa en problemas más pequeños.
* Las clases permiten crear mayores abstracciones en forma de componentes.
* Cada clase se encarga de una parte del problema y el programa se vuelve más fácil de mantener.

# **Abstracción**

* Es enfocarnos en la información relevante.
* Separar la información central de los detalles secundarios.
* Podemos utilizar variables y métodos (públicos y privados.)

# **Encapsulación, getters and setters**

* Permite agrupar datos y su comportamiento.
* Controla el acceso a dichos datos.
* Previene modificaciones no autorizadas.



# **Herencia**

* Nos permite modelar una jerarquía de clases.
* Permite compartir comportamiento común en la jerarquía.
* Al padre se lo conoce como superclase y al hijo como subclase.

# **Polimorfismo**

* Es la habilidad de tomar varias formas.
* En python, nos permite cambiar el comportamiento de una superclase para adaptarlo a la subclase.

# **Introducción a la complejidad algorítmica**

* ¿Por qué comparamos la eficiencia de un algoritmo?

Nos permite evaluar la eficiencia entre dos algoritmos y esto a su vez nos permite predecir el tiempo que nos vamos a tardar en resolver un problema.

* Complejidad temporal y complejidad espacial.
  + Complejidad temporal: Describe la cantidad de tiempo que lleva ejecutar un algoritmo.
  + Complejidad espacial: Es la cantidad de recursos utilizados por el algoritmo.
* Podemos definirlo como T**(n)**.

**Aproximaciones**

* Cronometrar el tiempo con el que corre un algoritmo.
* Contar los pasos con una medida abstracta de operación.
* Contar los pasos conforme nos aproximemos al infinito.

# **Notación asintótica**

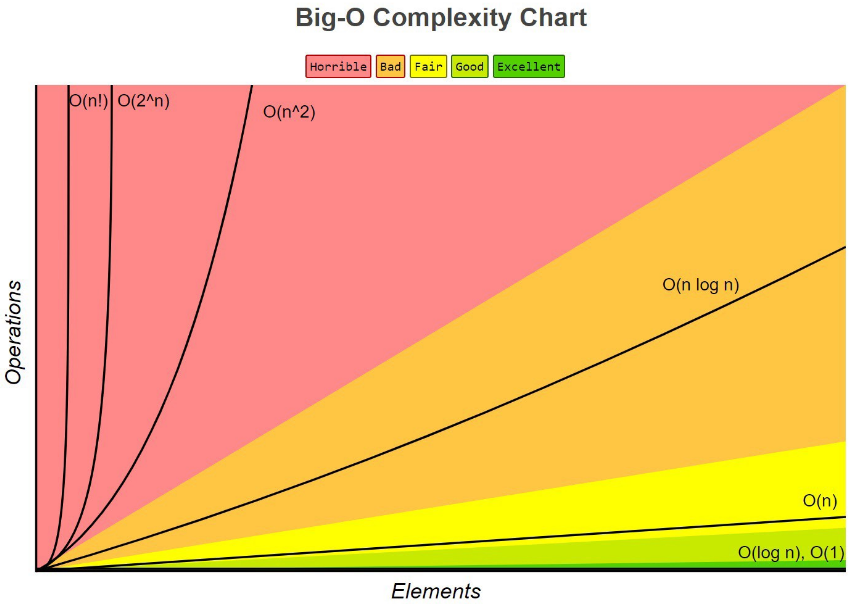
El termino asintótico significa **conforme se acerca al infinito**.

* No importan variaciones pequeñas conforme nos acercamos al infinito.
* En enfoque se centra en lo que pasa conforme el tamaño del problema se acerca al infinito.
* Existen tres casos:
  + **El mejor de los casos**, se encuentra el resultado en la primera posición de la lista.
  + **El promedio**, se encuentra el resultado en el medio de la lista.
  + **El peor de los casos**, se encuentra el resultado en la ultima posición de la lista.
* Big O.
* Nada más importa el termino de mayor tamaño.

# **Clases de complejidad algorítmica**

Existen distintos tipos de complejidad algorítmica:

* **O(1) Constante**: no importa la cantidad de input que reciba, siempre demorara el mismo tiempo.
* **O(n) Lineal**: la complejidad crecerá de forma proporcional a medida que crezca el input.
* **O(log n) Logarítmica**: nuestra función crecerá de forma logarítmica con respecto al input. Esto significa que en un inicio crecerá rápido, pero luego se estabilizará.
* **O(n log n) Log lineal**: crecerá de forma logarítmica pero junto con una constante.
* **O(n²) Polinomial**: crecen de forma cuadrática. No son recomendables a menos que el input de datos en pequeño.
* **O(2^n) Exponencial**: crecerá de forma exponencial, por lo que la carga es muy alta. Para nada recomendable en ningún caso, solo para análisis conceptual.
* **O(n!) Factorial**: crece de forma factorial, por lo que al igual que el exponencial su carga es muy alta, por lo que jamás utilizar algoritmos de este tipo.



# **Búsqueda lineal**

Busca en todos los elementos de manera secuencial.

# **Búsqueda binaria**

* Divide y conquista.
* El problema se divide en 2 en cada iteración.

Si se busca una sola vez en la lista podemos hacer la búsqueda lineal, si vamos a buscar varias veces dentro de la lista podríamos ordenar la lista

# **Ordenamiento de burbuja**

Es un algoritmo que recorre una lista repetidamente que necesita ordenarse. Compara elementos adyacentes y los intercambia si están el orden incorrecto. Este procedimiento se repite hasta que no se requieren más intercambios, lo que indica que la lista se encuentra ordenada.

# **Ordenamiento por inserción**

El ordenamiento por inserción es uno de los algoritmos más comunes que estudian los Científicos del Cómputo. Es intuitivo y fácil de implementar, pero es muy ineficiente para listas de gran tamaño.

Una de las características del ordenamiento por inserción es que ordena en “su lugar.” Es decir, no requiere memoria adicional para realizar el ordenamiento ya que simplemente modifican los valores en memoria.

La definición es simple:

* Una lista es dividida entre una sublista ordenada y otra sublista desordenada.

Al principio, la sublista ordenada contiene un solo elemento, por lo que por definición se encuentra ordenada.

* A continuación, se evalúa el primer elemento dentro la sublista desordenada para que podamos insertarlo en el lugar correcto dentro de la lista ordenada.
* La inserción se realiza al mover todos los elementos mayores al elemento que se está evaluando un lugar a la derecha.
* Continua el proceso hasta que la sublista desordenada quede vacía y, por lo tanto, la lista se encontrará ordenada.

Veamos un ejemplo:

Imagina que tienes la siguiente lista de números:

7, 3, 2, 9, 8

Primero añadimos 7 a la sublista ordenada:

**7**, 3, 2, 9, 8

Ahora vemos el primer elemento de la sublista desordenada y lo guardamos en una variable para mantener el valor. A esa variable la llamaremos **valor\_actual**.

Verificamos que 3 es menor que 7, por lo que movemos 7 un lugar a la derecha.

**7**, 7, 2, 9, 8 (valor\_actual=3)

3 es menor que 7, por lo que insertamos el valor en la primera posición.

**3**, **7**, 2, 9, 8

Ahora vemos el número 2, 2 es menor que 7 por lo que lo movemos un espacio a la derecha y hacemos lo mismo con 3.

**3**, **3**, **7**, 9, 8 (valor\_actual=2)

Ahora insertamos 2 en la primera posición.

**2**, **3**, **7**, 9, 8

9 es más grande que el valor más grande de nuestra sublista ordenada por lo que lo insertamos directamente en su posición.

**2**, **3**, **7**, 9, 8

El último valor es 8. 9 es más grande que 8 por lo que lo movemos a la derecha:

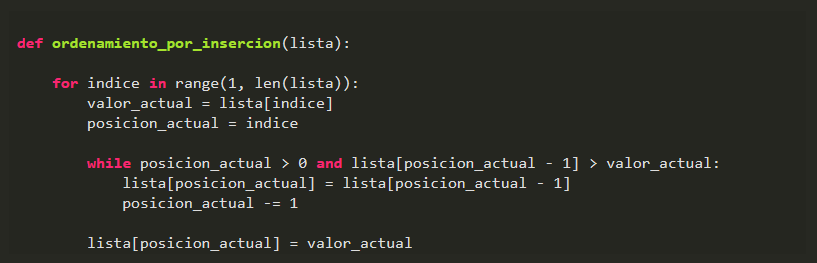
**2**, **3**, **7**, **9**, 9 (valor\_actual=8) 8 es más grande que 7, por lo que procedemos a insertar nuestro valor\_actual.

**2**, **3**, **7**, **8**, **9**

Ahora la lista se encuentra ordenada y no quedan más elementos en la sublista desordenada.

Antes de ver la implementación en Python, trata de implementarlo por ti mismo y compártenos tu algoritmo en la sección de comentarios.

Esta es una forma de implementar el algoritmo anterior:



# **Ordenamiento por mezcla**

Es un algoritmo que divide y conquista. Primero divide una lista en partes iguales hasta que quedan 1 o 0 elementos. Luego las recombina en forma ordenada.

# **Ambientes virtuales**

Son muy importante dentro de python.

**¿Por qué?**

* Permiten aislar el ambiente para poder instalar diversas versiones de paquetes.
* A partir de python 3 se incluye en la librería estándar en el módulo **venv**.
* Ningún ingeniero profesional de python traba sin ellos.

**Pip**

* Nos permite descargar programas de terceros para utilizar en nuestros programas.
* Permite compartir nuestros paquetes con terceros.
* Permite especificar la versión del paquete que necesitamos.

**Crear un ambiente virtual en windows**

1. Entras a CMD.
2. Entras a tu carpeta con el comando cd.
3. Pones el comando “**python -m venv nombre\_de\_tu\_espacio\_virtual**".
4. Entras a "**nombre\_de\_tu\_espacio\_virtual\Scripts\activate.bat**" y presionamos Enter.
5. Por último, tipeas “pip install nombre\_de\_librería” que se desea descargar.

**Algunos comandos pip**

* **search**: busca un paquete.



* **install**: instala un paquete.



* **show**: muestra detalles del paquete instalado.



**uninstall**: eliminar un paquete.



* **list**: retorna la lista de paquetes en el ambiente actual.



* **freeze**: se utiliza para congelar los paquetes y su versión actual.

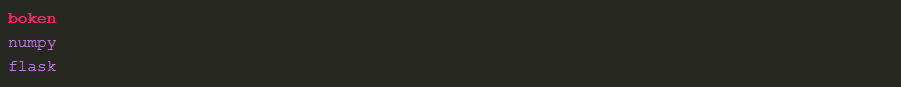


**deactivate**: Sale del entorno virtual.



* Para más comando [aquí](https://pip.pypa.io/en/stable/reference/pip/).

Una práctica muy útil es crear un archivo con el nombre de **requirements.txt** y colocar los paquetes que necesitemos uno debajo del otro.



Y luego usas para instalar



# **¿Por qué graficar?**

* Nos permite reconocer patrones.
* Predicción de una serie.
* Simplifica la interpretación y las conclusiones acerca de los datos.

# **Graficando simple**

* Bokeh es una librería que nos permite construir graficas complejas de manera rápida y con comandos simples.
* Permite exportar a varios formatos como html, notebooks, imágenes, etc.
* Bokeh se puede utilizar en el servidor con Flask y Django.

# **Introducción a la optimización**

* El concepto de optimización permite resolver muchos problemas de manera computacional.
* Una función objetivo que debemos maximizar o minimizar.
  + **Maximizar**: Encontrar el **input** que nos regrese el **output** más **alto**.
  + **Minimizar**: Encontrar el **input** que nos regresa el **output** más **bajo** dentro de una función muy específica.
* Una serie de limitantes que debemos respetar.

# **El problema del morral**

Imaginemos que somos un ladrón que queremos entrar a un museo o casa y tenemos un gran problema, que sólo hay una mochila y hay muchas más cosas de las que se puede cargar, entonces el ladrón deberá escoger cual de las cosas tendrá que llevarse para obtener el mayor valor posible.