**Programación orientada a objetos en Python:**

Clases para organizar programas en módulos y abstracciones de datos. 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.

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 solo 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.)

**Clases en Python:**

Las estructuras primitivas 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 solo 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:

Texto

Descripción generada automáticamente con confianza media

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).

Texto

Descripción generada automáticamente

**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.

Texto

Descripción generada automáticamente

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

En Python todo es un objeto y tiene un tipo como enteros o Strings, listas o diccionarios, y en programación se conocen como tipos de datos abstractos. Que todo sea un objeto quiere decir que todo tiene una representación en memoria, y que todo tenga un tipo quiere decir que se puede encapsular los datos y el comportamiento dentro de un solo objeto.

Los tipos vienen de las matemáticas, permiten modelar el mundo, cualquier cosa imaginable tiene un tipo y Python permite comenzar a modelar el mundo.

Los objetos se pueden crear tantas veces como se quiera, se pueden manipular, y se pueden destruir, en algunos lenguajes hay que destruir la referencia, pero en Python el garbage colector lo destruye automáticamente.

Existen varias ventajas en estos objetos, como decomponerlos en partes más pequeños, generar abstracciones como en un restaurante pedir una hamburguesa sin preocuparse como se genera esta, y también es posible encapsular, o sea esconder ciertos datos que no son importantes para los que utilizan las clases y objetos, y simplemente se manejan internamente.

Los datos abstractos se van a definir los datos abstractos, luego se le da el nombre de la clase, luego se puede referir una super clase, y finalmente se utiliza el método init, en Python el primer parámetro es self y finalmente los parámetros de inicialización, y dentro de la clase esta la expresión. Las funciones dentro de las clases se llaman métodos, se definen con def y el nombre del método.

El constructor es el primer método que se ejecuta cuando se genera una instancia de la clase

Texto

Descripción generada automáticamente

Las clases son un tipo de molde, es una forma de definir las características, y estas las van a poder utilizar todas las instancias de esta clase, en el anterior ejemplo se inicializa una instancia de la clase persona.

A los objetos se les conoce como instancia, esto significa que por ejemplo tenemos un molde de una botella y cada ves que hacemos una botella, esta será una instancia del monde, asimismo, las instancias pueden tener atributos distintos como el material y el color pero el molde sigue siendo el mismo.

Cada ves que generamos una nueva instancia se ejecuta el constructor, en Python se conoce como dunder init, dunder es duble under que signofoca doble guion bajo.

Los atributos de clase son las variables de instancia, o sea cuando se inicializa en el constructor con self, y los métodos, además, Existen un tercer tipo de variable de instancia que son las variables privadas, en otros lenguajes se utilizara la palabra private, pero en Python se utiliza una convención en este caso un guion bajo seguido del nombre del método o variable. (en Python siempre se utilizan convenciones)

El método isinstance permite saber si algún objeto es instancia de una clase.

**Decomposición**:

Es simplemente partir un problema en problemas mas pequeños, por ejemplo en una computadora puedes partirla en la CPU, la memoria. Etc. Las clases permiten generar los componentes mas pequeños, una ventaja es que ayuda a tener un software que se puede mantener de manera mas sencilla, una clase con muchas líneas de código es muy difícil de entender, entonces las clases ayudan a partir el problema y que cada pedazo sea más fácil de mantener.

Cada clase se vuelve una parte del problema y el programa es más fácil de entender.

Hay un tipo de parámetro llamado default keyword, directamente se le dice el tipo.

**Abstracción:**

La abstracción es enfocarse en la información relevante, es generar una interfaz mediante la cual se puede interactuar con cualquier tipo de objeto sin quedarse en los detalles o en cómo funciona el objeto. Hay que separar la información relevante de la que no es relevante, esto se puede hacer con variables y métodos privados o públicos.

La abstracción se puede ver en muchos aspectos, por ejemplo en elevador, solo nos preocupamos de darle click para ir arriba o abajo y el piso, pero no nos preocupamos como funcionan las poleas o como funciona internamente el elevador.

Para implementar la abstracción se genera una interfaz, la interfaz con la que se interactúa con las clases, de otra forma seria muy difícil implementarlo, porque sería necesario saber cómo funciona, pero lo que importa es saber como interactuar con los algoritmos, y no preocuparnos de la implementación interna.

La programación orientada a objetos permite ver directamente el objeto, entender como funciona e implementar los métodos que hace que funcione. Siempre es necesario pensar que es publico y que es privado.

En física una abstracción puede ser las formulas que describen un comportamiento.

**Funciones: base de los decoradores:**

Los decoradores son una forma sencilla de llamar funciones de orden mayor, es decir, funciones que toman otra función como parámetro y/o retornan otra función como resultado. De esta forma un decorador añade capacidades a una función sin modificarla. Un ejemplo de esto son las llantas de un automóvil. Si les colocas cadenas para la nieve, el automóvil aún puede andar y además extiende su funcionalidad para conducirse en otros terrenos. Una función decorador devuelve una función.

Imagen que contiene Texto

Descripción generada automáticamente

Funciones como objetos de primera-clase: en Python las funciones son objetos de primera-clase, es decir, que pueden ser pasados y utilizados como argumentos al igual que cualquier otro objeto (strings, enteros, flotantes, listas, etc.).

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

Las primeras dos funciones son obvias en su resultado, donde nos mostrarán un mensaje con el valor de la variable nombre. La tercera función puede ser más compleja de predecir, ya que toma otra función como entrada. Veamos qué pasa cuando colocamos una función como atributo:

Texto

Descripción generada automáticamente

Pongamos atención en cómo la función consume\_funciones() se escribe con paréntesis para ser ejecutada, mientras que la función presentarse y estudiemos\_juntos solo hace referencia a estas.

Funciones anidadas: puedes colocar funciones dentro de otra función.

Pantalla negra con letras blancas

Descripción generada automáticamente

Si llamamos a la función funcion\_mayor tendremos la siguiente salida:

Texto

Descripción generada automáticamente

Debemos considerar que las funciones anidadas dentro de funcion\_mayor no se ejecutan hasta que se llama a esta primera, siendo muestra del scope o alcance de las funciones. Si las llamamos obtendremos un error

**Setters, getters y decorador property:**

Ejemplo: Sucede que la función wrapper() recibió la la función zumbido() cómo parámetro y coloca su salida entre los otros dos prints.

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente con confianza media

Todo lo que sucede se conoce en programación como metaprogramación (metaprogramming), ya que una parte del programa trata de modificar a otra durante el tiempo de compilación. En tanto un decorador básicamente toma una función, le añade alguna funcionalidad y la retorna. Definitivamente la forma en que decoramos la función es compleja, pero afortunadamente Python lo tiene en cuenta y podemos utilizar decoradores con el símbolo @. Volviendo al mismo ejemplo de funcion\_decoradora(), podemos simplificarlo así:

Texto

Descripción generada automáticamente

En solo tres líneas de código tenemos el mismo resultado que escribir zumbido = funcion\_decoradora(zumbido).

**¿Qué son getters y setters?**

A diferencia de otros lenguajes de programación, en Python los getters y setters tienen el objetivo de asegurar el encapsulamiento de datos. Cómo habrás visto, si declaramos una variable privada en Python al colocar un guión bajo al inicio de esta (\_) y normalmente son utilizados para: añadir lógica de validación al momento de obtener y definir un valor y, para evitar el acceso directo al campo de una clase. La realidad es que en Python no existen variables netamente privadas, pues aunque se declaren con un guión bajo podemos seguir accediendo a estas. En Programación Orientada a Objetos esto es peligroso, pues podemos alterar el método de alguna clase y tener efectos colaterales que afecten la lógica de nuestra aplicación.

Clases sin getters y setters: Veamos un ejemplo con una clase que almacena un dato de distancia recorrida en millas (mi) y lo convierte a kilómetros (km):

Texto

Descripción generada automáticamente

Ahora creemos un objeto que haga referencia a un viaje:

Texto

Descripción generada automáticamente

Utilizando getters y setters: Incluyamos un par de métodos para obtener la distancia y otro para que no acepte valores inferiores a cero, pues no tendría sentido que un vehículo recorra una distancia negativa. Estos son métodos getters y setters:

Texto

Descripción generada automáticamente

El método getter obtendrá el valor de la distancia que y el método setter se encargará de añadir una restricción. También debemos notar cómo distancia fue reemplazado por \_distancia, denotando que es una variable privada. Si probamos nuestro código funcionará, la desventaja es que cualquier aplicación que hayamos creado con una base similar deberá ser actualizado. Esto no es nada escalable si tenemos cientos o miles de líneas de código.

**Función property():**

Esta función está incluida en Python, en particular crea y retorna la propiedad de un objeto. La propiedad de un objeto posee los métodos getter(), setter() y del().

En tanto la función tiene cuatro atributos: property(fget, fset, fdel, fdoc) :

* fget : trae el valor de un atributo.
* fset : define el valor de un atributo.
* fdel : elimina el valor de un atributo.
* fdoc : crea un docstring (cadena de caracteres de documentación, le permite documentar su código) por atributo.

Texto

Descripción generada automáticamente

Aunque en este ejemplo hay una sola llamada a print, tenemos tres líneas como salida pues esta llama a los primeros dos métodos. Por lo que la propiedad distancia es una propiedad de objeto que ayuda a mantener el acceso de forma privada.

**Decorador @property:**

Este decorador es uno de varios con los que ya cuenta Python, el cual nos permite utilizar getters y setters para hacer más fácil la implementación de la programación orientada a objetos en Python cambiando los métodos o atributos de las clases de forma que no modifiquemos el código.

Texto

Descripción generada automáticamente

De esta manera usamos el decorador @property para utilizar getters y setters de una forma más prolija e incluimos una nueva funcionalidad a nuestro método definir\_distancia() , al mismo tiempo protegemos el acceso a nuestras variables privadas y cumplimos con el principio de encapsulación.

**Encapsulación, getters and setters:**

La encapsulación permite agrupar datos y comportamientos en un mismo lugar, esto se ve con las clases, las clases permiten tener variables de instancia, métodos internos ya sea privados o públicas, y todo estará en la clase.

Lo importante de la encapsulación también tiene que ver con la programación defensiva, y permite controlar el acceso a los datos y a la modificación de los datos, esto porque Python es muy abierto, y es necesarios controlar cuando y como se modifica una clase o una propiedad, y cuando y como se extrae información de la clase.

En el siguiente ejemplo se observan los decoradores con el símbolo @, y para definir que una función es una propiedad, simplemente le ponemos @property, y en la siguiente función que nos permite modificar el valor se utiliza el @nombre.setter.

* getter: @property
* setter: @nombre.setter

Dentro de esas funciones se puede determinar cuando y como acceder y modificar esas variables.

Texto

Descripción generada automáticamente

**Herencia:**

La herencia permite modelar una jerarquía de objetos, compartir cierto comportamiento entre estos objetos, esto ayuda a tener el código organizado y a que tenga sentido, a las clases se les conoce como super clases, y las que heredan el comportamiento se les conoce como subclases.

Si por ejemplo tenemos una jerarquía de vehículos, y la super clase tiene como propiedades avanzar y frenar, además hay vehículos terrestres y aéreos, y así se puede seguir creando clasificaciones:

Imagen que contiene Escala de tiempo

Descripción generada automáticamente

El comportamiento que se va definiendo en la super clase se puede ir heredando a las subclases, y así se genera una mayor especialización. Esto permite que el código se mantenga limpio y que nos enfoquemos en las clases y comportamientos que importan, y permite generar una jerarquía completa, donde se define a que nivel corresponde cada comportamiento.

en otros lenguajes se utilizaría la palabra extend para referenciar a la super clase, pero en Python se pone entre paréntesis el nombre de la clase como si fuera un parámetro. Super permite obtener una referencia directa de la súper clase.

Un aspecto importantísimo de la Herencia es la posibilidad de reutilizar código. La reutilización es uno de los pilares de la POO, de manera que evitemos reinventar la rueda cada vez. Si tenemos un comportamiento que es común entre una serie de objetos de la misma categoría, este comportamiento debe enviarse a una clase superior que permita compartirlo con sus clases hijas. Esto facilita la mantenibilidad del código haciéndolo más estable.

**Polimorfismo:**

Si retomamos el ejemplo de los vehículos, cada subclase se desplaza pero de manera distinta, esto quiere decir polimorfismo. En otras palabras, el polimorfismo es la habilidad de tomar varias formas, y Python permite cambiar el comportamiento de la super clase para adaptarlo a la subclase.

En otros lenguajes se utiliza overwrite, pero en Python simplemente se toma el nombre del método de la super clase e implementarlo de otra forma en la subclase.

(cuando un método tiene como uno parámetro self significa que cuando lo ejecutemos no va a recibir parámetros)

**Introducción a la complejidad algorítmica:**

La complejidad algorítmica permite comparar la eficiencia entre dos algoritmos, y permite predecir el tiempo que toma resolver un problema, esto es importante cuando se trata de muchos datos. Es importante entender si el algoritmo va a ser capaz de resolver un problema en determinado tiempo.

La complejidad puede ser temporal o espacial, la primera es cuánto tiempo va a tardar, y la segunda es cuanto va a ocupar en memoria, pero aquí solo se va a tratar la temporal definida como T(n).

Hay muchas formas de implementar la función T, la primera es contar el tiempo que demora la ejecución, el problema es que las diferencias de tiempo dependerán de muchos factores, como por ejemplo cuantos procesos tiene el sistema operativo, entonces no siempre ayuda a comparar algoritmos. Una segunda idea es comenzar a contar los pasos que toma llegar a una solución, o sea como un contador, así se pueden contar comparaciones, operaciones matemáticas. etc. el problema con lo anterior es que la solución puede variar de implementación a implementación dentro de un mismo algoritmo, e introduce nuevos términos que tienden a ser irrelevantes.

La forma mas correcta de medir es contando los pasos pero con una medida asintótica, o sea conforme se acerca al infinito.

**Conteo abstracto de operación:**

Queremos contar los pasos del siguiente programa, esto lo haremos de forma matemática de la siguiente manera

Esto sale de sumar la cantidad de pasos que se haría en cada loop, entonces obtenemos un polinomio. Si x es pequeña el termino que mas suma es la constante, pero esto cambia si x es grande, por ejemplo si x es igual a 100, el término que más pesaría seria x cuadrada, y mientras más se acerca x a infinito, menos importancia tiene la constante.

Texto

Descripción generada automáticamente

**Notación asintótica:**

Cuando se habla de crecimiento asintótico se refiere a como el input crece al infinito, entonces cuanto más crezca los términos pequeños no importan. Es posible pensar en el mejor de los casos, en el caso promedio, y en el peor de los casos, como un algoritmo de búsqueda el mejor de los casos es que el dato que se busca este en la primera posición, pero siempre hay que medir el algoritmo en el peor de los casos, aunque en promedio encontraríamos la solución en la mitad de la lista. en Big O notation solo importa el termino de mayor tamaño.

Texto

Descripción generada automáticamente

En el ejemplo tenemos dos los que depende de n, o sea cada loop tendrá n pasos, si quisiéramos generar el termino en Big O notation tendría un crecimiento lineal. Si por ejemplo el segundo loop crecería en n al cuadrado, este sería el único termino que importaría y la función seria cuadrática.

Texto

Descripción generada automáticamente

Finalmente podemos poner un loop dentro de otro loop, y lo que sucede es que se multiplican los termino, en lugar de sumados, y la función seria cuadrática.

Texto

Descripción generada automáticamente

En el ejemplo del algoritmo de Fibonacci, se están generando dos llamadas de Fibonacci, o sea que el crecimiento es exponencial.

Texto

Descripción generada automáticamente

**Clases de complejidad algorítmica:**

* O(1) El tiempo es contante, siempre demora lo mismo
* O(n) Se crece de manera proporcional conforme crezca el input
* O(log n) Logarítmica, nuestra función crece de forma logarítmica con respecto al input
* O(n log n) Log lineal, se crece de manera logarítmica pero también con una constante
* O(n\*\*2) Polinomial
* O(2\*\*n) Exponencial

Gráfico

Descripción generada automáticamente

Un algoritmo que crece en logaritmo de n es un muy buen algoritmo, como la búsqueda lineal, el algoritmo lineal es bueno y se ve mucho en lista, el algoritmo n log n una parte se divide cada vez más pequeña pero la otra es constante, pero en los algoritmos polinomiales hay demasiados pasos, y no podríamos utilizarlos para resolver problemas con muchos datos, finalmente los algoritmos exponenciales no son muy útiles a menos que sean muy muy pocos inputs, o sea que solo solucionan problemas pequeños. Existen los O(n!) que ya está en la categoría no utilizables.

Las bases de datos de Google que más escalan lo hacen en logaritmo de n.

Tabla

Descripción generada automáticamente

**Búsqueda lineal:**

Busca en todos los elementos de manera secuencial, puede ser ordenado o no, y el peor de los casos es que el elemento esta al final

La línea de código que dice if \_\_name\_\_ == '\_\_main\_\_' quiere decir que si el archivo se ejecuta desde la consola, eso es lo que queremos que se ejecute primero.

**Búsqueda binaria:**

Divide y conquistaras, el problema divide en 2 en cada iteración y este algoritmo tiene el problema que asume que la lista esta ordenada, y no existe un buen algoritmo para ordenar, entonces hay que pensar si vale la pena ordenar o no la lista.

Existe una función para ordenar una lista directamente y es la función sorted:



Casi siempre para optimizar el tiempo hay que consumir mas memoria, pero para optimizar memoria hay que utilizar más tiempo. El peor de los casos de la búsqueda lineal es cuando el resultado esta al final, pero el peor de los casos de la búsqueda binaria es

Cuando se pone // significa división de enteros, lo que quiere decir que no interesan los decimales

**Ordenamiento de burbuja:**

El ordenamiento de burbuja es un algoritmo que recorre repetidamente una lista que necesita ordenarse, compara elementos adyacentes y los intercambia si están en el orden incorrecto, este procedimiento se repite hasta que no se requieran más intercambios.

Este algoritmo de ordenamiento da la garantía que después de la primera ronda, el valor mas grande va a estar hasta el final, por tanto es muy útil cuando se busca encontrar el valor mas grande.

Si tenemos una lista demasiado grande este algoritmo no va a funcionar, porque el crecimiento es O(n\*\*2)

**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 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 evalua 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.

Imagina que tienes la siguiente lista de números:

Primero añadimos 7 a la sublista ordenada:

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.

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

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.

Ahora insertamos 2 en la primera posición.

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

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

8 es más grande que 7, por lo que procedemos a insertar nuestro valor\_actual.

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

Texto

Descripción generada automáticamente

**Ordenamiento por mezcla:**

Merge sort en inglés inventado por John Ford Newman, es un ordenamiento que utiliza la estrategia de divide y vencerás, lo que hace es dividir la lista en sub listas cada ves mas pequeñas, hasta que quedan de 1 o 0 elementos (estas listas están ordenadas por definición), luego empieza a comparar listas cada vez más pequeñas hasta que ya tenemos una lista completamente ordenada.

Si nosotros dividimos primero las listas hasta que queden perfectamente ordenadas en una situación cada vez más pequeña (eso significa que tenemos log(n) y luego la reconquistamos en n) significa que podemos generar un ordenamiento muy eficiente.

**Ambientes virtuales:**

Python opera de manera global dentro de la computadora, y hasta ahora los imports vienen de la librería estándar de Python, en otros lenguajes como JavaScript, se depende completamente de la comunidad para otorgar todas las funcionalidades, pero Java si tiene una librería estándar muy fuerte. En Python para poder trabajar con diferentes versiones de librerías es necesario un ambiente virtual, para que el ambiente global no quede contaminado

Por ejemplo si hay un programa con django y luego se quiere desarrollar otro programa con django en otra versión, sin ambientes virtuales la computadora nada mas puede tener un django instalado.

En Python 2 hay que importar una librería para poderlos generar, pero en Python 3 se incorporo directamente a la librería estándar el manejo de ambientes virtuales.

Pip permite descargar programas de terceros, y permite especificar la versión exacta del paquete, entonces pip tiene librerías para todo y es gratis. Además también se pueden compartir las librerías que se hacen. Nunca hay que instalar paquetes de manera global, siempre en un ambiente virtual.

Para windows 10 puede usarse estos comandos, ya que el tutorial fue hecho en otro sistema operativo.

* mkdir graficado
* cd graficado/
* python -m venv env
* env\Scripts\activate.bat
* pip install bokeh

**¿Por qué graficar?:**

Graficar permite identificar los patrones adentro de los datos, esto porque el cerebro esta diseñado para identificar información visual de manera muy rápida. Los gráficos son muy importantes para trabajar con datos, ya que permite predecir cual es el siguiente elemento adentro de una serie, también es posible predecir el programa se está comportando de manera correcta.

Los gráficos es necesario interpretarlos y sacar conclusiones, y el hecho de que sea visual ayudan a esto. Lo primero es identificar si existe o no un patrón, porque no hay necesidad de ver patrones donde no los hay.

**Graficado simple:**

En Python se utiliza la librería llamada bokeh para hacer gráficos, porque permite hacer gráficos de manera rápida y muy simple. En Python existen muchas librerías que funcionan para lo mismo, e incluso hay herramientas por fuera de Python, pero lo importante es decidir si se quieren graficar los datos para entenderlos o si se quiere mostrar la gráfica al usuario, dependerá del caso de uso.

La librería bokeh permite exportar a html, a jupiter notebooks y también imágenes de las gráficas, además también se puede utilizar en el servidor con flash y con django, esto quiere decir que se pueden generar gráficos que los vea el usuario dentro de una aplicación web o una API.

La página principal de bokeh es: docs.bokeh.org

Es necesario encontrar la forma correcta de visualizar, porque hay muchas opciones y no todas permiten extraer conclusiones de buena manera.

**Introducción a la optimización:**

La optimización permite resolver muchos problemas computacionales, ya que muchos problemas se pueden reducir a algoritmos de optimización, o sea hay que maximizar o minimizar una función, eso quiere decir encontrar el input que regresa el output más alto o bajo.

A veces no hay limitantes, pero lo normal es que hallan ciertas limitantes por ejemplo si queremos optimizar vuelos, queremos el vuelo mas barato, en ciertas fechas, sin escalas y que además tenga asientos en salida de emergencia. Otro problema de optimización es el tráfico, este problema lo soluciono waze y se vendió a Google por mucho dinero.

Un problema muy común es el de traveling salesman o vendedor viajero, consiste en dado una serie de ciudades, hay que encontrar la ruta mas eficiente para recorrer todas las ciudades. Hoy en día no existe una solución eficiente para poderlo resolver, los mejores algoritmos son exponenciales lo que quiere decir que si hay demasiadas ciudades no se puede resolver. Esto significa que existen problemas que los mejores algoritmos que tenemos, no pueden resolverlos, o existen problemas que no pueden ser resueltos, o no han encontrado solución.

**El problema del morral:**

Imaginemos que somos un ladro y quiere entrar a una casa, y nada más tiene una mochila, pero hay muchas mas cosas de las que se pueden cargar, entonces hay que escoger que articulo llevar para garantizar el mayor valor posible.

Este problema se puede resolver con greede algorithm si pudiéramos dividir los artículos en pedazos, entonces escoger primero lo mas alto, lo segundo mas alto. Etc. Esto funciona por ejemplo si tenemos oro molido, plata molida y arroz, entonces se llenaría primero todo de oro luego de plata y finalmente lo que quede de arroz.

Si por el contrario no podemos dividir los artículos, el problema seria 1 0 knapsack, y necesitara un algoritmo diferente.

En recursividad sin un caso base, es lo mismo que un infinite loop.