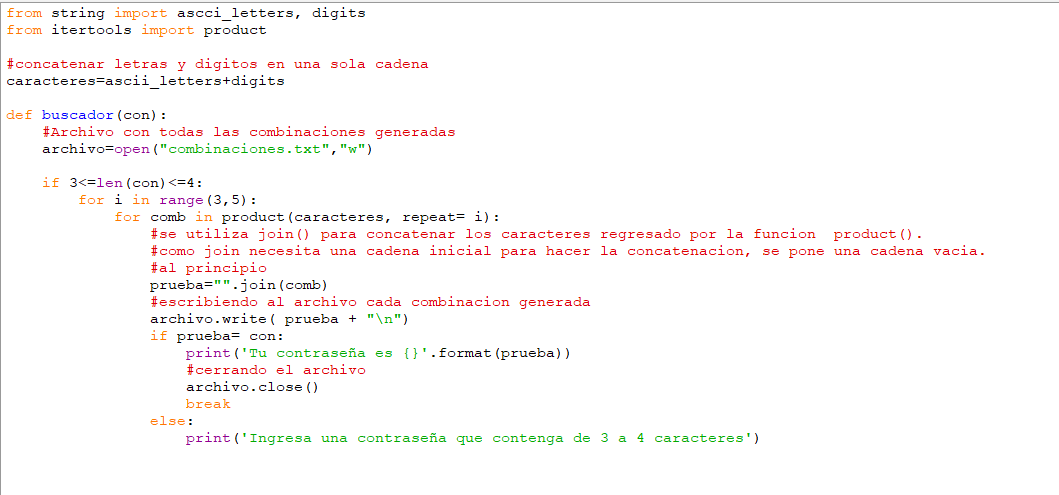
**Objetivo:**

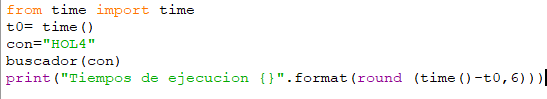
El objetivo de esta guía es implementar, al menos, dos enfoques de diseño (estrategias) de algoritmos y analizar las implicaciones de cada uno de ellos.

**Desarrollo**

*Fuerza Bruta.*

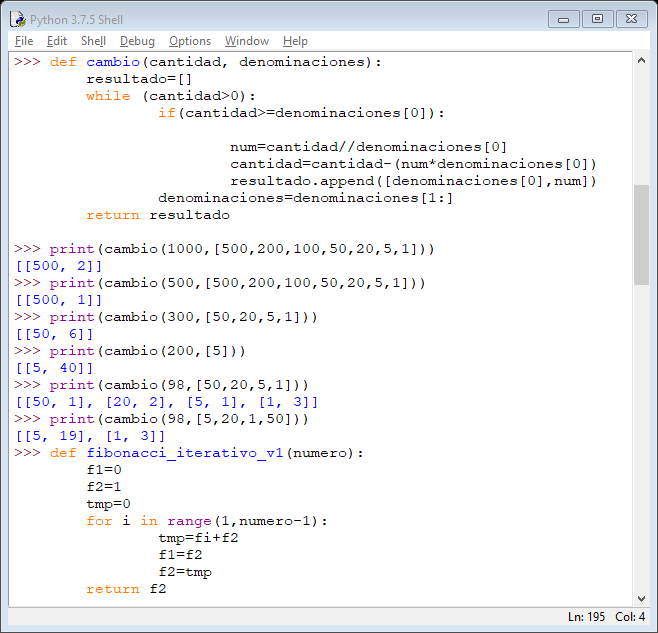
El objetivo de resolver problemas por medio de fuerza es bruta es hacer una búsqueda exhaustiva de todas las posibilidades que lleven a la solución del problema. Un ejemplo de esto es encontrar una contraseña haciendo una combinación exhaustiva de caracteres alfanuméricos generando cadenas de cierta longitud. La desventaja de resolver problemas por medio de esta estrategia es el tiempo que toman.





*Algoritmos ávidos (greedy).*

Esta estrategia se diferencia de fuerza bruta porque va tomando una serie de decisiones en un orden específico, una vez que se ha ejecutado esa decisión, ya no se vuelve a considerar. En comparación con fuerza bruta, ésta puede ser más rápida; aunque una desventaja es que la solución que se obtiene no siempre es la más óptima.



\*nota: el uso del operador de división, la diferencia entre / y // es que el primer operador se realizar para números reales y el segundo para números enteros.

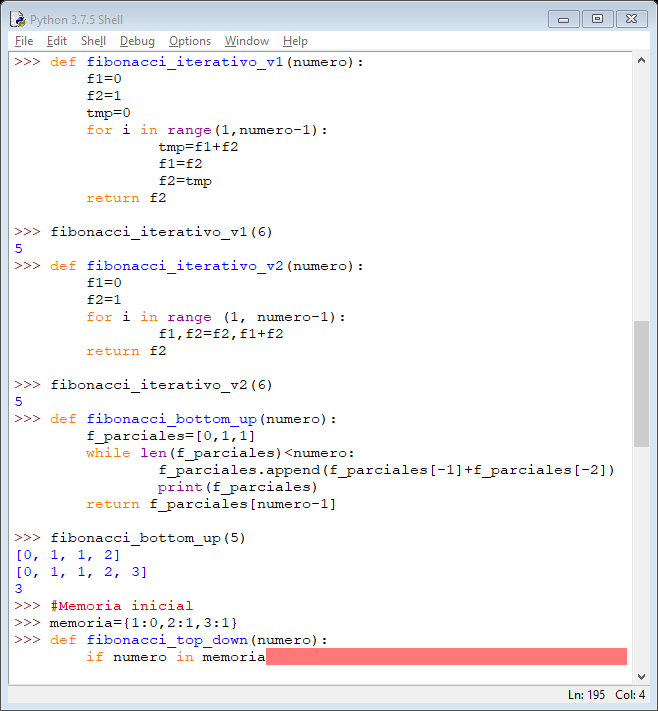
El problema consiste en regresar el cambio de monedas, de cierta denominación, usando el menor número de éstas. Este problema se resuelve escogiendo sucesivamente las monedas de mayor valor hasta que ya no se pueda seguir usándolas y cuando esto pasa, se utiliza la siguiente de mayor valor. Sin embargo la solución del problema debe estar ordenado de mayor a menor, de lo contrario se resuelve el problema pero no de la manera óptima.

*Bottom-up (programación dinámica)*

El objetivo de esta estrategia es resolver un problema a partir de subproblemas que ya han sido resueltos. La solución final se forma a partir de la combinación de una o más soluciones que se guardan en una tabla, ésta previene que se vuelvan a calcular las soluciones.

En este ejemplo se aprecia la sucesión de Fibonacci. La sucesión de Fibonacci es una sucesión infinita de números enteros cuyos primeros dos elementos son 0 y 1, los siguientes números son calculados por la suma de los dos anteriores.

Se presenta la implementación iterativa para calcular la sucesión de Fibonacci.



Como se puede apreciar la aplicación de la estrategia de bottom-up. Donde se parte de las soluciones

F(0)=0, F(1)=1, f(2)=1

Estas soluciones previas son almacenadas en tablas de soluciones f\_parciales. f\_parciales=[0,1,1]. No hace el cálculo de los primeros números sino que toma las soluciones previas

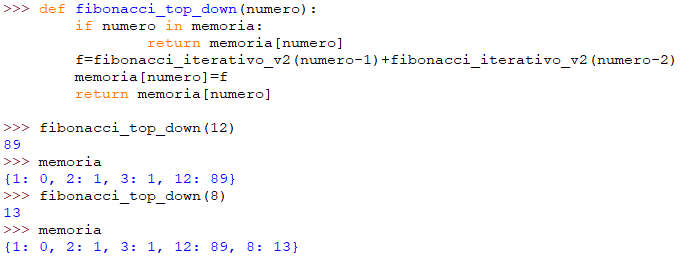
[0, 1, 1] Datos iniciales

[0, 1, 1, 2] Primera iteración, se calcula n-1 = 1, y n - 2 = 1;

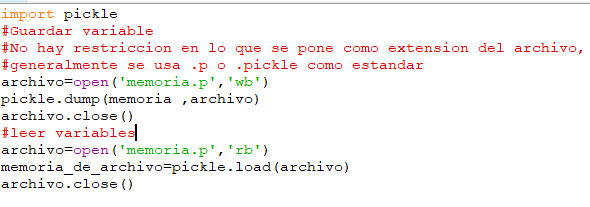
[0, 1, 1, 2, 3] Segunda iteración, se calcula n-1 = 2, y n - 2 = 1;

*Top-down*

A diferencia de bottom-up, aquí se empiezan a hacer los cálculos de n hacia abajo. Además, se aplica una técnica llamada memorización la cual consiste en guardar los resultados previamente calculados, de tal manera que no se tengan que repetir operaciones. Al aplicar la estrategia top-down, se utiliza un diccionario (memoria) el cual va a almacenar valores previamente calculados. 



Se calculan n-1 y n-2 usando la versión iterativa. Una ventaja es que una vez que ya se calcularon, se guardan en una memoria, que en este caso es un diccionario; en dado caso de que se necesite un valor que ya ha sido calculado, sólo regresa y ya no se realizan los cálculos.

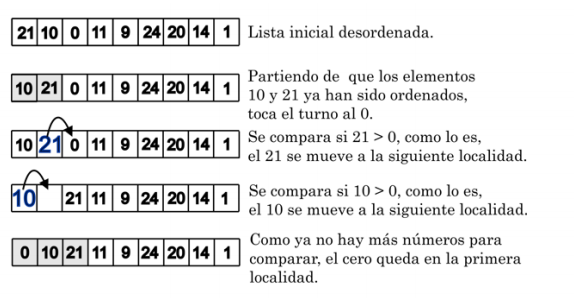
Los archivos que se generan con pickle están en binario, por lo que no se puede leer a simple vista la información que contienen, como se haría desde un archivo de texto plano. 

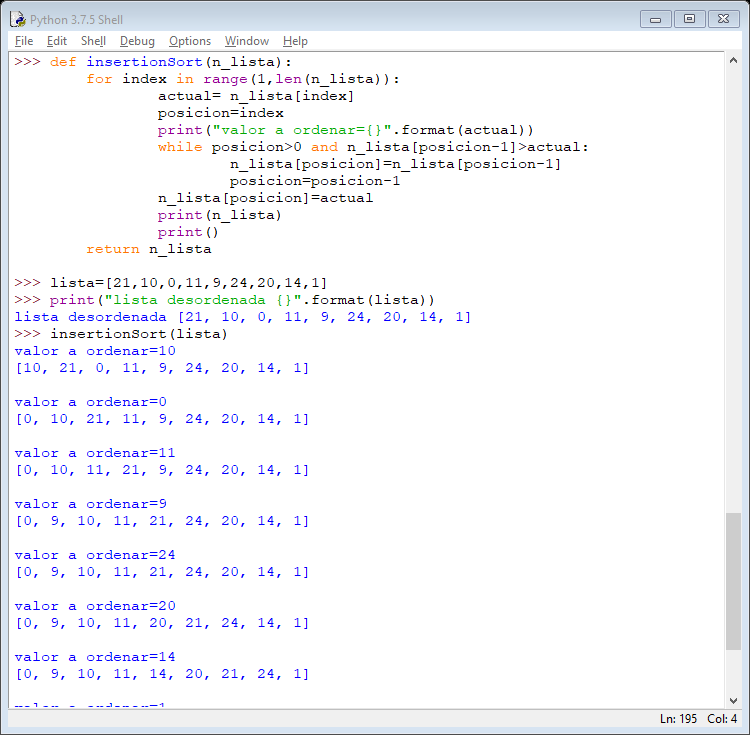
***Incremental.***

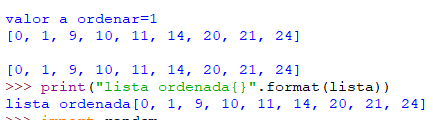
Es una estrategia que consiste en implementar y probar que sea correcto de manera paulatina, ya que en cada iteración se va agregando información hasta completar la tarea.

*Insertion sort*

Insertion sort ordena los elementos manteniendo una sublista de números ordenados empezando por las primeras localidades de la lista. Al principio se considera que el elemento en la primera posición de la lista está ordenado. Después cada uno de los elementos de la lista se compara con la sublista ordenada para encontrar la posición adecuada.







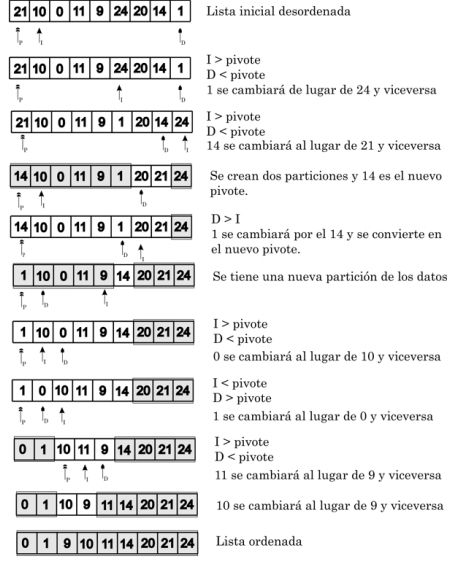
***Divide y vencerás.***

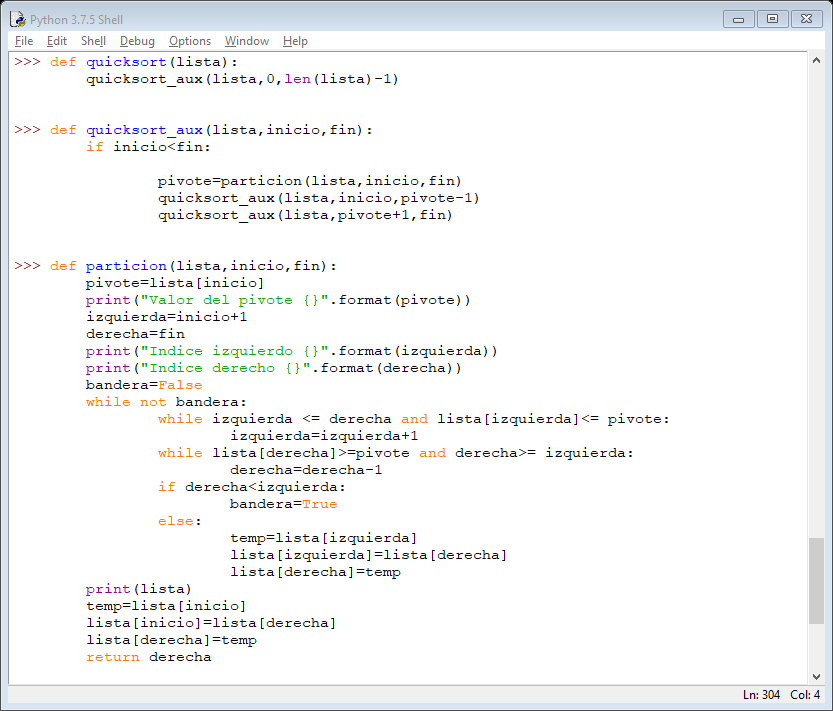
Es una estrategia que consiste en:

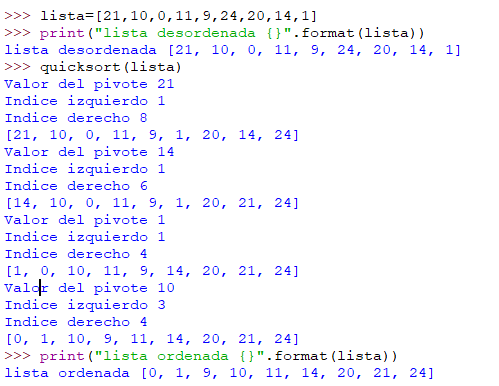
* Dividir el problema en subproblemas hasta que son suficientemente simples que se pueden resolver directamente.
* Después las soluciones son combinadas para generar la solución general del problema.

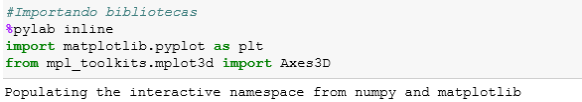
*Quick sort.*

Quicksort es un ejemplo de resolver un problema por medio de la estrategia divide y vencerás. En Quicksort se divide en dos el arreglo que va a ser ordenado y se llama recursivamente para ordenar las divisiones. La parte más importante en Quicksort es la partición de los datos. Lo primero que se necesita es escoger un valor de pivote el cual está encargado de ayudar con la partición de los datos. El objetivo de dividir los datos es mover los que se encuentran en una posición incorrecta con respecto al pivote. La siguiente figura muestra un ejemplo de cómo se ordena una lista.







*Medición y graficas de los tiempos de ejecución* 

Las funciones en Python pueden ser guardadas en archivos individuales (insertionSort.py) o varias en un sólo archivo (quickSort.py).

En dado caso de que se quiera llamar más funciones que estén en un mismo archivo se pueden escribir los nombres de las funciones separados por nombres: \*from file\_name import función1, función2, función3\*

*Modelo RAM*

Cuando se realiza un análisis de complejidad utilizando el modelo RAM, se debe contabilizar las veces que se ejecuta una función o un ciclo, en lugar de medir el tiempo de ejecución.

**Conclusión.**

En esta práctica, en lo personal se me hizo muy interesante por los métodos para poder resolver diferentes problemas comunes, con ayuda de indicaciones que anteriormente se vieron en la práctica de introducción a Python.

Asimismo se debe de analizar y desarrollar todos los aspectos a utilizar para un buen funcionamiento del programa a crear, el apoyo de bibliotecas y de funciones para un óptico, eficiente y eficaz código que haga un gran cambio en la programación. También conocí como poder influir algunas características para implementarlo en otros campos de trabajo.