Programación para Física y Astronomía

Departamento de Física.

April 26, 2022

Universidad Andrés Bello®

python powered

Resumen

Ordenando Listas

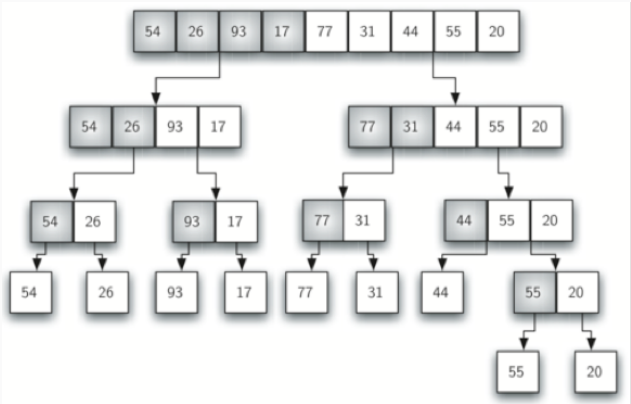
Variables y Rangos

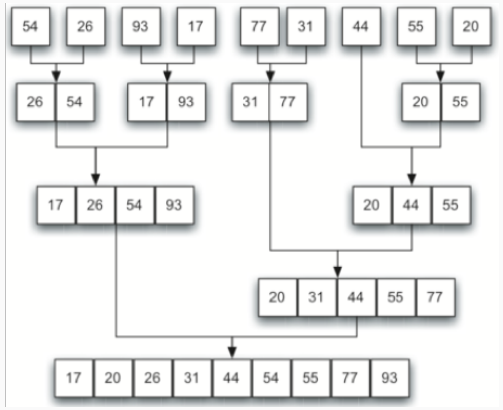
- Errores numéricos

Ordenando Listas

Ordenando Datos

1. Merge sort: algoritmo basado en la estrategia de divide y vencerás. Algoritmo recursivo que continuamente divide una lista en mitades:
   - Si la lista esta vacía o tiene un ítem, está ordenada por definición.
   - Si la lista posee más de un ítem, se divide la lista y recursivamente se invoca al algoritmo nuevamente.
   - Una vez que las dos mitades están ordenadas, se desarrolla la operación de merge(mezcla), mezclando las listas y retornando la lista ordenada.





Ordenando Datos

```
1  def mergeSort(lista):
2    print("Dividiendo ",lista)
3    if len(lista)>1:
4    centro = len(lista)//2
5    mit_izq = lista[:centro]
6    mit_der = lista[centro:]
7    mergeSort(mitad_izq)
8    mergeSort(mitad_der)
9    i=0
10   j=0
11   k=0
12   while i<len(mit_izq) and j<len(mit_der):
13     if mitad_izq[i] < mitad_der[j]:
14       lista[k]=mitad_izq[i]
15       i=i+1
16     else:
17       lista[k]=mitad_der[j]
18       j=j+1
19     k=k+1
```

```
20   while i < len(mitad_izq):
21     lista[k]=mitad_izq[i]
22     i=i+1
23     k=k+1
24   while j < len(mitad_der):
25     lista[k]=mitad_der[j]
26     j=j+1
27     k=k+1
28   print("Mezclando ",lista)
29 milista = [54,26,93,17,77,31,44,55,20]
30 mergeSort(milista)
31 print(milista)
```

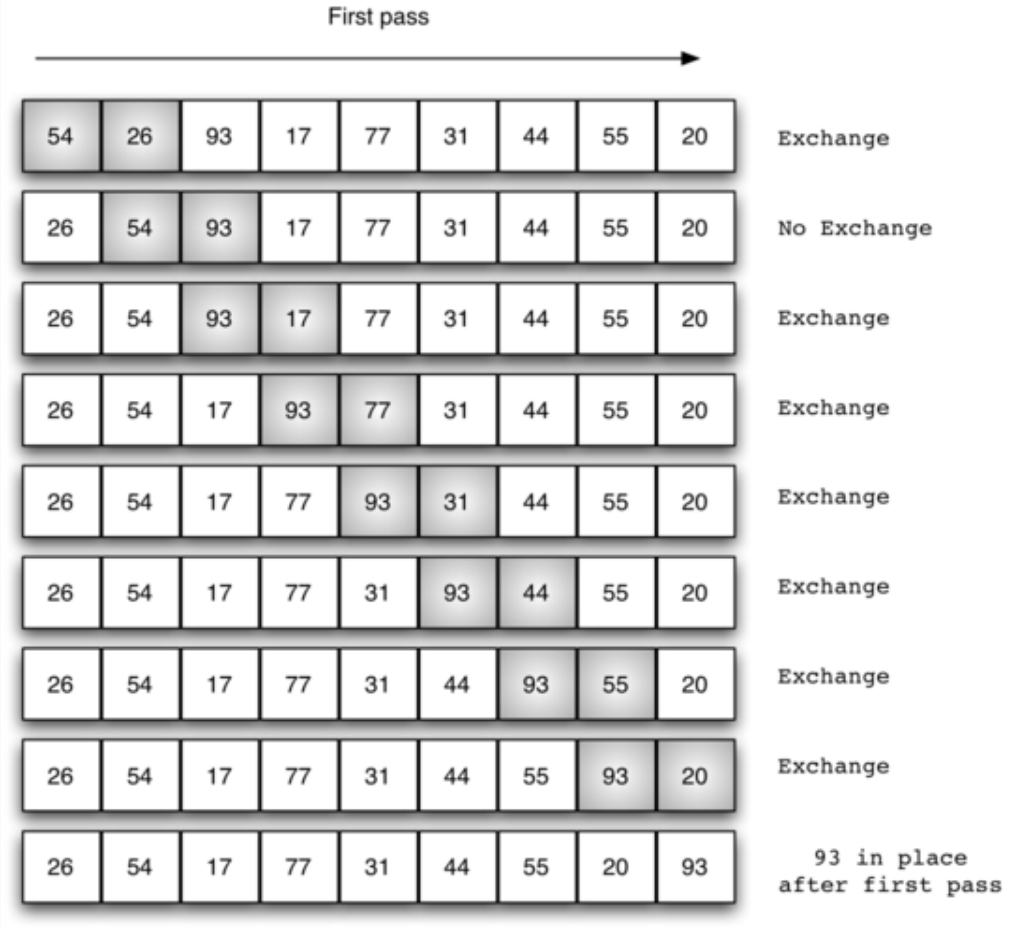
Ordenando Datos

Pero existen muchos algoritmos más, diseñados para ordenar listas de números, entre ellos usted encontrará.

- Merge Sort: Dividir la lista, y reordenar. Divide y vencerás.
- Insertion Sort: Basado en comparación, un elemento a la vez.
- Bubble Sort: Basado en compración de pares e intercambios (*swap*)
- QuickSort: Es un algoritmo basado en comparación. Mediante un pivote.
- HeapSort: Usa un salto binario para ordenar la estructura de los elementos.
- Counting Sort: Ordenamiento para enteros.

Bubble Sort

Como primer ejercicio, deberán implementar el algoritmo Bubble sort. Puede definir una lista cualquiera en su código.



Variables y Rangos

Variables y Rangos

Hasta el momento hemos visto varios elementos básicos de programación en Python que nos permiten trabajar y resolver diversos problemas en Física. Sin embargo hay aspectos de exactitud y velocidad que no hemos abordado y que son muy relevantes.

- Los computadores tienen limitaciones. Estos no pueden guardar números reales con infinito número de decimales. Existe un límite para el número mayor y menor que se puede guardar.
- Se pueden efectuar cálculos muy rápidos, pero no infinitamente rápidos.

Variables y Rangos

Hemos visto ejemplos de uso de **variables** que incluyen, `int`, `float`, `complex`, etc. Sin embargo estas variables no pueden guardar números que sean arbitrariamente grandes.
Por ejemplo:

El valor más grande que se puede almacenar en una variable de tipo `float` es $\sim 10^{308}$. Existe también su correspondiente valor negativo $\sim -10^{308}$

Recordemos que para representar números grandes podemos usar la notación científica, usando e para denotar el exponente, por ejemplo:

`2e9` significa 2×10^9 y `1.602e-19` significa 1.602×10^{-19}.
Nota: números especificados en notación científica son siempre `float`.

Variables y Rangos

Si las variables exceden el número mayor de tipo `float` que se puede almacenar, se dice que la variable se ha desbordado (*overflowed*). Pruebe el siguiente programa:

```
1 for i in range(10):
2  v=1.0e305*(10**i)
3  print(v)
```

En Python, cuando la variable es desbordada esta recibe, como vimos con el programa anterior, un valor especial "`inf`" que denota infinito.

Variables y Rangos

Por otra parte, también existe un límite para el número más pequeño que se puede almacenar en una variable:

El valor más pequeño que se puede almacenar en una variable de tipo `float` es $\sim 10^{-323}$.

En este caso, si el computador intenta llegar a números más pequeños se dice que se produce un *underflowed* y le designa simplemente cero a la variable.

Ejercicio

Proponga un programa para probar lo indicado anteriormente.

¿Y qué pasa con los enteros? En Python los **enteros** pueden ser representados con precisión arbitraria. No existe límite para el tamaño de un entero. [1]

[1] https://www.tutorialspoint.com/what-is-the-maximum-possible-value-of-an-integer-in-python

Errores numéricos

Los números del tipo `float` son representados en el computador con una precisión **finita**.
En Python el estándar de precisión es de 16 cifras significativas. Por ejemplo: los números como π y $\sqrt{2}$, los cuales son irracionales, pueden solo ser representados aproximadamente.

Real vs Python

valor real de π:	`3.14159265358979323846...`
valor en Python:	`3.141592653589793`
diferencia:	`0.00000000000000023846...`

La diferencia entre el valor real y el valor en Python es llamado *error de redondeo*.

Redondeo

Cualquier valor, cuyo valor real tenga más de 16 cifras significativas, sufrirá de error de redondeo en Python.

Errores numéricos

Cuando realizamos aritmética con números flotantes, los resultados son sólo garantizados con la precisión de 16 cifras, aunque los números sean expresados exactamente.

Si sumamos 1.1 y 2.2 en Python el resultado será: 3.3000000000000003. Entonces:

```
1 x=1.1+2.2
2 if x==3.3:
3  print("Verdadero")
4 else:
5  print("Falso")
```

¿Cuál creen que es el resultado?

Una importante consecuencia del *error de redondeo* es que **nunca se debe usar una declaración `if` para probar la calidad de dos `float`.**

Errores numéricos

Para solucionar el caso anterior se recomienda usar un número como `epsilon`, escogido apropiadamente, que indica la precisión con la que estoy realizando la comparación:

```
epsilon=1e-12
x=1.1+2.2
if abs(x-3.3)<epsilon:
 print("Verdadero")
else:
 print("Falso")
```

A tener en cuenta

Los números flotantes son representados como fracciones en base 2 por el hardware de la máquina.
Por ejemplo: la fracción decimal 0.125 puede ser representada por 1/10+2/100+5/1000 y el mismo número en fracción binaria es 0.001 que puede ser representado por 0/2+0/4+1/8.

IEEE-754

Desafortunadamente, muchas fracciones decimales no pueden ser representadas exactamente en fracciones binarias, lo que implica que el valor binario que puede ser almacenado por el computador con el estándar IEEE-754 debe ser truncado.

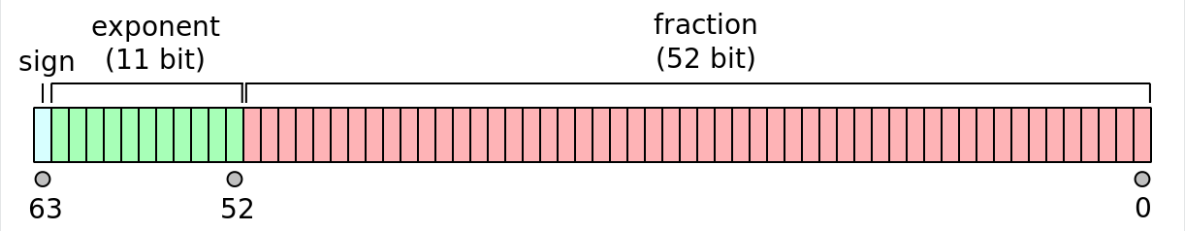


Figure 1: IEEE-754 "Double-precision":

Error de redondeo

El error de redondeo (ϵ) en un número se define como la cantidad que se debería agregar al número calculado por el computador para obtener el valor real.

```
1 from math import sqrt
2 x=sqrt(2)
```

Obtenemos al final $x = \sqrt{2} - \epsilon$. En general si `x` es exacto hasta 16 cifras significativas, entonces el error de redondeo tendrá un tamaño de $x/10^{16}$.

Es usualmente aceptable asumir que el error es un número aleatorio con una desviación estándar $\sigma = Cx$, donde $C \backsimeq 10^{-16}$. El error de redondeo es similar al error que se mide en el laboratorio de experimentos y por lo tanto, las reglas para la combinación de errores son las mismas.

Ejemplo: $\sigma = \sqrt{\sigma_1^2 + \sigma_2^2}$ para el caso de la suma o diferencia de x_1 y x_2.

Error de redondeo

Si queremos apreciar el efecto real del error de redondeo, podemos extender el resultado al calcular la suma de N números $x_1 ... x_N$, con errores que tengan desviación estándar $\sigma_i = C x_i$.

$$\sigma = C\sqrt{N}\sqrt{\overline{x^2}}$$

Error de redondeo

Entre los problemas más comunes están:

- Suma de números muy variados: Si hay números mucho más pequeños que otros, estos pueden perderse.
- Sustracción de números:

```
1 x=1000000000000000
2 y=1000000000000001.2345678901234
```

Si calculamos $y - x$, dado que python almacena solo 16 cifras significativas, el resultado será $y - x = 1.25$ en vez de 1.2345678901234. El error entre el valor real y el calculado es de un 1.25%

Error de redondeo

Consideremos los siguientes números:

$$x = 1, y = 1 + 10^{-14}\sqrt{2}$$

de donde podemos ver que:

$$10^{14}(y - x) = \sqrt{2}$$

```
1  from math import sqrt
2  x=1.0
3  y=1.0+(1e-14)*sqrt(2)
4  print((1e14)*(y-x))
5  print(sqrt(2))
```

```
1.42108547152
1.41421356237
```

Fin