SAYISAL FİZİK (COMPUTATIONAL PHYSICS)

SAYI TEMSİLLERİ BİT VE BAYT

En küçük hafıza birimine bit adı verilir(binary digit). Bir bit, 0 veya 1 değerini alabilir. Sayılar bilgisayarda ikilik tabanda temsil edilir. Sekiz bitten oluşan hafıza birimine bayt adı verilir. 2^8=256 farklı değer alabilir. 0,1,....255 sayısına kadar temsil edebilir. Temsil edilmesi bakımında iki gruba ayrılır:

TAMSAYILAR

Belli bir büyüklüğe kadar tamsayılar bilgisayarda tam doğru olarak temsil edilir. Ayrılan bayt sayısına göre tamsayıların temsil aralığı değişir:

16 BİT YADA 2 BAYTLIK TAMSAYILAR

En basit temsilde, tamsayılar 16 bit yani 2 baytlık alanda saklanır. Her bit 2 değer alabildiğine göre 2^16=65536 sayıda değer alabilir. Negatif ve pozitif aralıklara dağıtırsak

```
-32768< 2-baytlık tamsayılar < 32767
```

aralığında olabilirler. Negatif sayıların birinci biti 1 alınır. Örnek olarak, 13 sayısının temsilini görelim. Bu iki sayıyı ikilik tabanda yazarsak,

$$13 = \mathbf{1} \times 2^3 + \mathbf{1} \times 2^2 + \mathbf{0} \times 2 + \mathbf{1} \times 2^0 = (1101)_2$$

olur. Buna göre, 13 sayısı hafızada şöyle saklanır:

0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1

$= 13$

-13 sayısı ise, birinci biti tersine çevirerek elde edilir:

1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1

$= -13$

32 BİT YADA 4 BAYTLIK TAMSAYILAR

Daha büyük aralıkta tamsayılar gerektiğinde 32 bit= 4 bayt yer kullanılır. Bu alana 2^32=4294967296 sayıda değer alabilir; bu değer negatif ve pozitif sayılara şu şekilde paylaştırılır:

```
-2147483648 < 32-baytlık tamsayılar < +2147483647
```

Bu sayılarda ikilik tabanda aynı şekilde gösterilir. 16 bitlik sayılar "int" komutuyla tanımlanır. 32 bitlik sayılar ise "long" komutuyla yada "L" harfi eklenerek tanımlanır. Bu tanım python 2.x sürümleri için geçerli. Python 3.x sürümlerinde "int" tipindeki değişkenler 64 bite kadar olan sayıları tutabilmektedir.

Örneğin; a=59, b=578329567L olsun.

In [32]:

```
a=59
b=7832956789125768738
print(type(a))
print(type(b))
"""type() fonksiyonu değişkenin tipini verir."""
```

```
<class 'int'>
<class 'int'>
```

Out[32]:

```
'type() fonksiyonu değişkenin tipini verir.'
```

REEL SAYILAR

Reel sayılar ancak sınırlı sayıda haneyle temsil edilirler. √2= 1.41421... sayısının ondalık hanesi istenildiği kadar uzataılabilir, ne zaman (√2)² görürsek 2 değerini alırız. Fakat bilgisayarda 7-8 hane belleğe alınır ve tüm işlemler bununla yapılır. Reel sayıların hafızada saklanma kuralı

```
±0.XXX*10^Y
```

şeklindedir.

```
923.68 = +0.92368*10^3
0.00000874 = 0.874*10^-5
-3.56 = -0.356*10^9
```

Buna göre reel sayıların 3 bileşeni vardır:

- İşaret (±),
- Ondalık Hanesindeki Sayılar (mantis, yani yukarıdaki örneklere göre 92368, 874, 356),
- Üs (3, -5, 9).

Birçok dilde karşılaşılan reel sayı karmaşası python'da bulunmamaktadır. Reel sayılar "float" komutuyla çift-duyarlı yani 64-bitlik tanımlanırlar. Bir değişkene ondalık değer girdiğinzde otomatik olarak "float" tipinde tanımlar. Anlamadıysanız merak etmeyin örnekte gayet anlaşılır olacak.

In [39]:

```
e=2.71
print(e)
print(type(e))
d=30.00
print(d)
print(type(d))
h=5
print(h)
print(type(h))
g=float(h)
print("tip dönüşümü yapmış olduk\n",type(g))
```

```
2.71
<class 'float'>
30.0
<class 'float'>
5
<class 'int'>
tip dönüşümü yapmış olduk
 <class 'float'>
```

SAYISAL HATA TÜRLERİ

1. YUVARLAMA HATASI

Reel sayılar hafızalarını sınırlı duyarlılıkta saklayabiliyordu. Örneğin pi sayısı: π = 3.14159265359 Bazı derleyiciler 6-7 haneden sonrasını doğrudan kesip atar, bazılarıda bir önceki haneye yuvarlar, hangisi olursa olsun buna yuvarlama hatası denir. Her zaman A-A=0 olmayabilir. Örneğin a-(a/b)*b ifadesini hesaplayan basit bir program yazalım.

In [41]:

```
from math import *

a=cos(1.1)
b=sin(1.1)

z= a-(a/b)*b
print(z)
```

```
-5.551115123125783e-17
```

Kağıt üzerinde yapıldığında sonucun "0" olacağı gayet açıktı. Sonucun "0" olmaması a ve b sayılarının bellekte sınırlı ondalık haneye sahip olmasıdır.

1. KESME HATASI

Bir hesaplamada kullanılan matematiksel yöntemin yaklaşık oluşundan kaynaklanan hata türüne kesme hatası denir. Örneğin f(x) fonksiyonunun bir a noktası civarındaki Taylor açlımını gözönüne alalım "x = a + h" alınırsa,

$$f(a+h) = f(a) + h\,f'(a) + \frac{h^2}{2}f''(a) + \frac{h^3}{6}f'''(a) + \frac{h^4}{24}f^{(4)}(a) + \cdots$$

olur. Buradan 1. türevi çekip alırsak,

$$f'(a) = \frac{f(a+h) - f(a)}{h} - \frac{h}{2}\left[f''(a) + \frac{h^3}{6}f'''(a) + \frac{h^4}{24}f^{(4)}(a) + \cdots\right]$$

olur. Pratik hesaplarda bu sonsuz terimli seriyi bir yerde kesmek gerekir, çünkü bilgisayarda sonsuz sayıda işlem yapılamaz. Örneğin, sadece ilk terimi alırsak, kalan terimler h ile orantılı bir hata oluştururlar:

$$f'(a) \approx \frac{f(a+h) - f(a)}{h} + O(h)$$

Burada O(h) yazılımına "h mertebesinden hata" adı verilir.

Kesme hatası, kullanılan yöntem ve sayısal değerlere bağlı olarak, bizim elimizde olan bir hatadır. Örneğin türev hesabında hata payını azaltmak için 2 şey yapabiliriz:

- Taylor serisini 2. veya daha yüksek türevli terimlerden sonra kesebiliriz. Bu matematiksel yöntemin iyileştirilmesi demektir.
- Hata payı O(h) mertebesinde olduğuna göre, h adım uzunluğunu küçültürsek, kesme hatasıda küçülecektir. Bu sayısal yöntemin iyileştirilmesi anlamına gelir. Örneğin, 1. türevli Taylor açılımında h = 0.01 aldığımızda yapılan hata, h = 0.1 alınarak yapılan hatandan yaklaşık 10 kez daha küçük olur.

Ancak, h adım uzunluğunu küçültmenin de bir sınırı vardır. Belli bir değerden daha küçük olduğunda, bu kez yuvarlama hatası etkin olmaya başlar. Çünkü, çok küçük sayılarla çok büyük sayıları birlikte kullanmış oluruz.

Bu etkiyi görmek için f'(a) türevini,

$$f'(a) \approx \frac{f(a+h) - f(a)}{h}$$

ifadesiyle hesaplayalım. Örnek olarak, f(x) = sinx fonksiyonuna a = 1 noktasındaki türevini değişik h adım uzunluklarıyla hesaplayalım ve gerçek türevi olan cosx ile karşılaştıralım.

In [77]:

```
from math import *

def f(x):
    return sin(x)

a=1.0
h=1.0

for i in range(14):
    f1=(f(a+h)-f(a))/h
    h=h/10.0
    print("%.14f" %h,"%.10f" %f1,"%.10f" % float(cos(a)-f1))


print("\nTürevin gerçek değeri", cos(a))
```

```
0.10000000000000 0.0678264420 0.4724758639
0.01000000000000 0.4973637525 0.0429385533
0.00100000000000 0.5360859810 0.0042163249
0.00010000000000 0.5398814804 0.0004208255
0.00001000000000 0.5402602314 0.0000420744
0.00000100000000 0.5402980985 0.0000042074
0.00000010000000 0.5403018851 0.0000004207
0.00000001000000 0.5403022640 0.0000000418
0.00000000100000 0.5403023029 0.0000000030
0.00000000010000 0.5403023584 -0.0000000525
0.00000000001000 0.5403022474 0.0000000585
0.00000000000100 0.5403011372 0.0000011687
0.00000000000010 0.5403455461 -0.0000432402
0.00000000000001 0.5395683900 0.0007339159

Türevin gerçek değeri 0.5403023058681398
```

Bu sonuçlara bakarak şu gözlemleri yapabiliriz:

- h adım uzunluğu küçüldüğünde, gerçekten de kesme hatası azalmaktadır. Türevi en doğru veren sonuç h =(0.1*10^9)
- Fakat, h adımı daha da küçüldüğünde sonuçlarda hata payı artmaktadır. Bu etki yuvarlama hatasından, yani bilgisayarda reel sayıların temsilinin sınırlı oluşundan kaynaklanmaktadır.