02ex_NumberRepresentation

March 14, 2020

1. Write a function that converts number representation (bin<->dec<->hex)

In [1]:
```
mapDecHex = {0:0, 1:1, 2:2, 3:3, 4:4, 5:5, 6:6, 7:7, 8:8, 9:9, 10:'a', 11:'b', 12:'c',
mapHexDec = {'0':0, '1':1, '2':2, '3':3, '4':4, '5':5, '6':6, '7':7, '8':8, '9':9, 'a'

def BinToDec (x):
    y=str(x)
    result= ""
    decimal=0
    if x==0:
        return '0'
    for i,j in zip(map(str.lower,y),range(len(y))):
        k=(int)(mapHexDec[i])
        decimal += k*2**(len(y)-j-1)
    while decimal>0 :
        result += str(mapDecHex[decimal % 10])
        decimal =decimal//10
    return result[::-1] #revert the string


def HexToDec (x):
    y=str(x)
    result= ""
    decimal=0
    if x==0:
        return '0'
    for i,j in zip(map(str.lower,y),range(len(y))):
        k=(int)(mapHexDec[i])
        decimal += k*16**(len(y)-j-1)
    while decimal>0 :
        result += str(mapDecHex[decimal % 10])
        decimal =decimal//10
    return result[::-1]

def DecToBin (x):
    y=str(x)
    result= ""
```

```
    decimal=0
    if x==0:
        return '0'
    for i,j in zip(map(str.lower,y),range(len(y))):
        k=(int)(mapHexDec[i])
        decimal += k*10**(len(y)-j-1)
    while decimal>0 :
        result += str(mapDecHex[decimal % 2])
        decimal =decimal//2
    return result[::-1]


def DecToHex (x):
    y=str(x)
    result= ""
    decimal=0
    if x==0:
        return '0'
    for i,j in zip(map(str.lower,y),range(len(y))):
        k=(int)(mapHexDec[i])
        decimal += k*10**(len(y)-j-1)
    while decimal>0 :
        result += str(mapDecHex[decimal % 16])
        decimal =decimal//16
    return result[::-1]

print(BinToDec(101))
print(HexToDec("c"))

print(DecToBin(6))
print(DecToHex(12))
```

```
5
12
110
c
```

2. Write a function that converts a 32 bit word into a single precision floating point (i.e. interprets the various bits as sign, mantissa and exponent)

```
In [2]: a='zzzz'
        import math
        b=bin(int.from_bytes(a.encode(), 'big'))
        print(b)

        f=1.0
        e=0
```

```
for i in range(11,33):
    if b[i]=='1' :
        f+=pow(2, -i+10)
        #print(f)
for i in range(3, 11):
    if b[i]=='1' :
        e+=pow(2, 10-i)

print("mantissa: ", f)
print("exponent: ", e)
f*=pow(2,e-127)*pow(-1, int(b[2]) )
print ("number: ",  f)
```

```
0b11110100111101001111010011110 10
mantissa:  1.9137253761291504
exponent:  233
number:  -1.5525984779021514e+32
```

3. Write a program to determine the underflow and overflow limits (within a factor of 2) for python on your computer.

Tips: define two variables inizialized to 1 and halve/double them enough time to exceed the under/over-flow limits

```
In [3]: a=1.0
        b=1.0
        #da migliroare

        while a!=0. :
            tempA = a
            a /=2
        while b/b==1. :
            tempB = b
            b *=2

        print("Underflow limit", tempA)
        print("Overflow limit", tempB)
```

```
Underflow limit 5e-324
Overflow limit 8.98846567431158e+307
```

4. Write a program to determine the machine precision

Tips: define a new variable by adding a smaller and smaller value (proceeding similarly to prob. 2) to an original variable and check the point where the two are the same

```
In [4]: a=1.0
        b=a
```

```
import math
print(a.hex(), b.hex())

for i in range (1, 100):
    if (a.hex()==(b+pow(10, -i)).hex()):
        print("they are the same at ", i, " iteration")
        break
    else:
        print("different at ", i)
```

```
0x1.0000000000000p+0 0x1.0000000000000p+0
different at  1
different at  2
different at  3
different at  4
different at  5
different at  6
different at  7
different at  8
different at  9
different at  10
different at  11
different at  12
different at  13
different at  14
different at  15
they are the same at  16  iteration
```

5. Write a function that takes in input three parameters a, b and c and prints out the two solutions to the quadratic equation $ax^2 + bx + c = 0$ using the standard formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

(a) use the program to compute the solution for $a = 0.001$, $b = 1000$ and $c = 0.001$

(b) re-express the standard solution formula by multiplying top and bottom by $-b \mp \sqrt{b^2 - 4ac}$ and again find the solution for $a = 0.001$, $b = 1000$ and $c = 0.001$. How does it compare with what previously obtained? Why?

(c) write a function that compute the roots of a quadratic equation accurately in all cases

```
In [5]: #MANCA PUNTO C

        import math
        from decimal import Decimal
        def solve1 (a, b, c):
            d=math.sqrt(b*b-4*a*c)
            return (-b+d)/(2*a) , (-b-d)/(2*a)
```

```
def solve2 (a, b, c):
    d=math.sqrt(b*b-4*a*c)
    return (-b+d)*(-b-d)/((2*a)*(-b-d)),(-b-d)*(-b+d)/((2*a)*(-b+d))

def solve3 (a,b,c):
    a=Decimal(a)
    b=Decimal(b)
    c=Decimal(c)
    d=Decimal(math.sqrt(b*b-4*a*c))
    return Decimal((-b+d)/(2*a)) , Decimal((-b-d)/(2*a))

print(solve1(0.001, 1000, 0.001))
print(solve2(0.001, 1000, 0.001))
print(solve3(0.001, 1000, 0.001))
```

```
(-9.999894245993346e-07, -999999.999999)
(-9.999894245993346e-07, -999999.9999990001)
(Decimal('-9.999894245993345767711238822E-7'), Decimal('-999999.9999989999897587189540'))
```

6. Write a program that implements the function $f(x) = x(x1)$

(a) Calculate the derivative of the function at the point $x = 1$ using the derivative definition:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}$$

with $\delta = 10^2$. Calculate the true value of the same derivative analytically and compare with the answer your program gives. The two will not agree perfectly. Why not?

(b) Repeat the calculation for $\delta = 10^4, 10^6, 10^8, 10^{10}, 10^{12}$ and 10^{14}. How does the accuracy scales with δ?

```
In [6]: def func (x):
            return x*(x-1)

        def NumDer (x0, delta):
            return (func(x+delta)-func(x))/delta

        def AnalitycDer(x0): #d/dx x(x-1)= 2x-1
            return 2*x0-1

        x=1
        print (func(x))
        print ("Analitic: ",AnalitycDer(x))
        print("Numeric: ")
        for i in range (2, 16, 2):
            print ("Delta 1e-",i,NumDer(x,10**(-i)))
```

```
0
Analitic:  1
Numeric:
Delta 1e- 2 1.010000000000001
Delta 1e- 4 1.0000999999998899
Delta 1e- 6 1.0000009999177333
Delta 1e- 8 1.0000000039225287
Delta 1e- 10 1.000000082840371
Delta 1e- 12 1.0000889005833413
Delta 1e- 14 0.9992007221626509
```

7. Consider the integral of the semicircle of radius 1:

$$I = \int_{-1}^{1} \sqrt{(1 - x^2)} \mathrm{d}x$$

which it's known to be $I = \frac{\pi}{2} = 1.57079632679....$ Alternatively we can use the Riemann definition of the integral:

$$I = \lim_{N \to \infty} \sum_{k=1}^{N} h y_k$$

with $h = 2/N$ the width of each of the N slices the domain is divided into and where y_k is the value of the function at the $k-$th slice.

(a) Write a programe to compute the integral with $N = 100$. How does the result compares to the true value?

(b) How much can N be increased if the computation needs to be run in less than a second? What is the gain in running it for 1 minute?

```
In [7]: import math
        import numpy as np
        from math import pi

        def func(x):
            return math.sqrt(1.0-x*x)

        def integrate(iterations, xmin, xmax):
            delta=(xmax-xmin)/iterations
            integral=np.sum([func(xmin+delta*i)*delta for i in range(0, iterations+1)])
            return integral

        N=100
        print("N= ",N)
        print("My integral: ", integrate(N, -1,1))
        print("True value:  ", pi/2)
        print("Difference:  ", integrate(N, -1,1)-pi/2)

        N1=3200000
```

```
%time integrate(N1, -1,1)
print("N= ",N1)
print("Difference:  ", integrate(N1, -1,1)-pi/2)
```

```
N=  100
My integral:  1.56913425554925
True value:   1.5707963267948966
Difference:   -0.0016620712456465458
CPU times: user 884 ms, sys: 289 ms, total: 1.17 s
Wall time: 844 ms
N=  3200000
Difference:   -2.905296003774538e-10
```

In [ ]: