TP6-c.py

```
001| #Exercice n°1
002| def nat_bin(n):
003|     if n==0:
004|         return "0"
005|     b=""
006|     while n!=0:
007|         r=n%2
008|         n=n//2
009|         b=str(r)+b
010|     return b
011|
012| def nat_dec(b):
013|     n=0
014|     for c in b:
015|         n=n*2+int(c)
016|     return n
017|
018| def nat_bin_kbits(n,k):
019|     b=str()
020|     q=n
021|     while q!=0:
022|         q,r=q//2,q%2
023|         b=str(r)+b
024|     b=(k-len(b))*'0'+b
025|     return(b)
026|
027| #Exercice n°2
028| def rel_bin_nbits(r,n):
029|     if r<0:
030|         r=r+2**n
031|     if r==0:
032|         return n*"0"
033|     b=""
034|     q=r
035|     while q!=0:
036|         reste=q%2
```

```
037|          q=q//2
038|          b=str(reste)+b
039|      return (n-len(b))*"0"+b
040|
041| #Exercice n°3
042| def ajoute(n,m,b,i): # somme n+m*b**i
043|      rep=list(n) # rep contien la réponse
044|      if m==[0]:
045|          return rep
046|      for k in range(len(m)+i-len(n)):
047|          rep.append(0) # si m*b**i est plus
long que n
048|      retenue=0
049|      for k in range(len(m)): # somme des
chiffres n[i+k]+m[k]
050|          s=rep[i]+m[k]+retenue
051|          if s>=b:  # traitement de la
retenue
052|              rep[i]=s-b
053|              retenue=1
054|          else:
055|              rep[i]=s
056|              retenue=0
057|          i=i+1
058|      while i<len(n) and retenue==1: # si n
est plus long que m*b**i
059|          s=n[i]+retenue
060|          if s>=b:
061|              rep[i]=s-b
062|              retenue=1
063|          else:
064|              rep[i]=s
065|              retenue=0
066|          i=i+1
067|      if retenue>0: # s'il reste une retenue
finale
068|          rep.append(retenue)
069|      return rep
```

```
070|
071| # Pour tester que cela marche
072| a=[1,1,0,1]
073| b=[1,0,0,1]
074| assert ajoute([0],b,2,0)==b
075| assert ajoute(a,[1],2,0)==[0,0,1,1]
076| assert ajoute(a,b,2,0)==[0,0,1,0,1]
077|
078|
079| def produit1(n,c,b):
080|     if c==[0]:
081|         return [0]
082|     if c==[1]:
083|         return n
084|     # en base deux, c'est terminé
085|     retenue=0
086|     prod=len(n)*[0]
087|     for i in range (len(n)):
088|         p=n[i]*c[0]+retenue
089|         prod[i]=p%b
090|         retenue=p//b
091|     if retenue>0:
092|         prod.append(retenue)
093|     return prod
094|
095| # Pour tester que cela marche
096| assert produit1([1,0,1,1],[0],2)==[0] # en
base 2
097| assert produit1([1,0,1,1],[1],
2)==[1,0,1,1]
098| assert produit1([2,3,4],[2],10)==[4,6,8] #
en base 10
099| assert produit1([3,4,5],[3],10)==[9,2,6,1]
100| assert produit1([2,3,4],[5],
16)==[10,15,4,1] # en base 16
101| assert produit1([2,3,4],[6],
16)==[12,2,9,1]
102|
```

```
103| def produit(n,m,b):
104|     prod=[]
105|     for i in range(len(m)):
106|         p=produit1(n,[m[i]],b) # m[i] est
un nombre à un chiffre
107|         prod=ajoute(prod,p,b,i)
108|     return prod
109|
110| assert produit([1,0,1],[1,1],2)==[1,1,1,1]
111| assert produit([1,1,1],[1,1],
2)==[1,0,1,0,1]
112| assert produit([7,2,8],[5,2],
10)==[5,7,6,0,2]
113| assert produit([4,1],[4],16)==[0,5]
114|
115| #Exercice n°4
116| def decimal_be(b):
117|     n=len(b)
118|     d=b[0]
119|     for i in range(1,n):
120|         d=b[i]+256*d
121|     return d
122|
123| def decimal_le(b):
124|     n=len(b)
125|     d=b[n-1]
126|     for i in range(1,n):
127|         d=b[n-1-i]+256*d
128|     return d
129|
130|
131|
132|
133|
134|
135|
136|
```