Travaux pratiques de mathématiques - Méthode RSA Python

$\mathcal{S}$ nb-tp3py.1. Test de primalité. [1 - B. Ischi 24-25]

(1) Rédiger un script *Python3* qui détermine si un nombre n choisi par l'utilisateur est premier en utilisant une boucle **while**:

```
>python3 ex.py
Tapez un nombre n entier >1: n= 4657
4657  est premier
>python3 ex.py
Tapez un nombre n entier >1: n= 4659
4659  n'est pas premier
```

(2) Rédiger un script *Python3* qui détermine si un nombre n choisi par l'utilisateur est premier en utilisant une boucle **for** et la commande **break**:

```
>python3 ex.py
Tapez un nombre n entier >1: n= 4999
4999  est premier
>python3 ex.py
Tapez un nombre n entier >2: n= 4997
4997  n'est pas premier
```

$\mathcal{S}$ nb-tp3py.2. Liste de nombres premiers. [1 - B. Ischi 24-25]

Rédiger un programme *Python3* qui affiche tous les nombres premiers $\leq n$ où n est un nombre entier ≥ 3 choisi par l'utilisateur. **NB** Un nombre k est premier si et seulement si $p \nmid k$ pour tout premier $p < k$. Commencer par créer une fonction **def** test(n, liste): qui renvoie True si aucun des nombres contenus dans la liste liste ne divise n.

```
>python3 ex.py
Tapez un nombre entier n >3: n= 5000
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109,
 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239,
 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379,
 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521,
 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661,
 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827,
 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991,
 997, 1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1087, 1091, 1093, 1097, 1103, 1109,
 1117, 1123, 1129, 1151, 1153, 1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223, 1229, 1231, 1237, 1249, 1259,
 1277, 1279, 1283, 1289, 1291, 1297, 1301, 1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381, 1399, 1409, 1423,
 1427, 1429, 1433, 1439, 1447, 1451, 1453, 1459, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523, 1531, 1543,
 1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613, 1619, 1621, 1627, 1637, 1657, 1663, 1667,
 1669, 1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783, 1787, 1789, 1801, 1811, 1823,
 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987,
 1993, 1997, 1999, 2003, 2011, 2017, 2027, 2029, 2039, 2053, 2063, 2069, 2081, 2083, 2087, 2089, 2099, 2111, 2113,
 2129, 2131, 2137, 2141, 2143, 2153, 2161, 2179, 2203, 2207, 2213, 2221, 2237, 2239, 2243, 2251, 2267, 2269, 2273,
 2281, 2287, 2293, 2297, 2309, 2311, 2333, 2339, 2341, 2347, 2351, 2357, 2371, 2377, 2381, 2383, 2389, 2393, 2399,
 2411, 2417, 2423, 2437, 2441, 2447, 2459, 2467, 2473, 2477, 2503, 2521, 2531, 2539, 2543, 2549, 2551, 2557, 2579,
 2591, 2593, 2609, 2617, 2621, 2633, 2647, 2657, 2659, 2663, 2671, 2677, 2683, 2687, 2689, 2693, 2699, 2707, 2711,
 2713, 2719, 2729, 2731, 2741, 2749, 2753, 2767, 2777, 2789, 2791, 2797, 2801, 2803, 2819, 2833, 2837, 2843, 2851,
 2857, 2861, 2879, 2887, 2897, 2903, 2909, 2917, 2927, 2939, 2953, 2957, 2963, 2969, 2971, 2999, 3001, 3011, 3019,
 3023, 3037, 3041, 3049, 3061, 3067, 3079, 3083, 3089, 3109, 3119, 3121, 3137, 3163, 3167, 3169, 3181, 3187, 3191,
 3203, 3209, 3217, 3221, 3229, 3251, 3253, 3257, 3259, 3271, 3299, 3301, 3307, 3313, 3319, 3323, 3329, 3331, 3343,
 3347, 3359, 3361, 3371, 3373, 3389, 3391, 3407, 3413, 3433, 3449, 3457, 3461, 3463, 3467, 3469, 3491, 3499, 3511,
 3517, 3527, 3529, 3533, 3539, 3541, 3547, 3557, 3559, 3571, 3581, 3583, 3593, 3607, 3613, 3617, 3623, 3631, 3637,
 3643, 3659, 3671, 3673, 3677, 3691, 3697, 3701, 3709, 3719, 3727, 3733, 3739, 3761, 3767, 3769, 3779, 3793, 3797,
 3803, 3821, 3823, 3833, 3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907, 3911, 3917, 3919, 3923, 3929, 3931, 3943,
 3947, 3967, 3989, 4001, 4003, 4007, 4013, 4019, 4021, 4027, 4049, 4051, 4057, 4073, 4079, 4091, 4093, 4099, 4111,
 4127, 4129, 4133, 4139, 4153, 4157, 4159, 4177, 4201, 4211, 4217, 4219, 4229, 4231, 4241, 4243, 4253, 4259, 4261,
 4271, 4273, 4283, 4289, 4297, 4327, 4337, 4339, 4349, 4357, 4363, 4373, 4391, 4397, 4409, 4421, 4423, 4441, 4447,
 4451, 4457, 4463, 4481, 4483, 4493, 4507, 4513, 4517, 4519, 4523, 4547, 4549, 4561, 4567, 4583, 4591, 4597, 4603,
 4621, 4637, 4639, 4643, 4649, 4651, 4657, 4663, 4673, 4679, 4691, 4703, 4721, 4723, 4729, 4733, 4751, 4759, 4783,
 4787, 4789, 4793, 4799, 4801, 4813, 4817, 4831, 4861, 4871, 4877, 4889, 4903, 4909, 4919, 4931, 4933, 4937, 4943,
```

```
4951, 4957, 4967, 4969, 4973, 4987, 4993, 4999]
```

$\mathcal{S}$ nb-tp3py.3. Algorithme d'Euclide pour le pgcd. [1 - B. Ischi 24-25]

Rédiger un script *Python3* qui détermine par l'algorithme d'Euclide le pgcd des nombres a et b choisis par l'utilisateur:

```
>python3 ex.py
a= 124475554909
b= 123529474163
pgcd( 124475554909 , 123529474163 )= 4999
```

$\mathcal{S}$ nb-tp3py.4. Algorithme d'Euclide étendu pour le pgcd. [1 - B. Ischi 24-25]

Rédiger un script *Python3* qui détermine par l'algorithme d'Euclide étendu le pgcd des nombres a et b choisis par l'utilisateur et qui donne deux nombres r et s tels que

$$\text{pgcd}(a, b) = r \cdot a + s \cdot b$$

Voici un exemple d'exécution du code:

```
>python3 ex.py
a=? 124475554909
b=? 123529474163
pgcd( 124475554909 , 123529474163 )= 4999 = 11833662 * 124475554909 + -11924293 * 123529474163
```

$\mathcal{S}$ nb-tp3py.5. Base 2. [1 - B. Ischi 24-25]

Rédiger un programme *Python3* qui transforme un nombre donné par l'utilisateur en base 2. Voici un exemple de fonctionnement du programme:

```
Tapez un nombre: 1602
[0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1]
```

$\mathcal{S}$ nb-tp3py.6. Transformation d'un texte en nombres. [1 - B. Ischi 24-25]

Rédiger un programme *Python3* qui transforme un texte donné par l'utilisateur (sans accents) en blocs de nombres. Voici un exemple de fonctionnement du programme:

```
Tapez un texte (sans accents SVP): Bonjour
Longueur des blocs: 3
[7237442, 7696234, 2105458]
```

Le programme procède comme suit:

(1) Des espaces sont ajoutés à la fin du texte afin que sa longueur soit un multiple de 3:

```
"Bonjour" ---> "Bon"   "jou"   "r  "
```

(2) Chaque bloc est transformé en nombre. Une suite de trois lettres peut être interprétée comme un nombre de trois chiffres écrit en base 256 à l'envers en considérant le code `ASCII` de chaque lettre. Rappelons que les codes `ASCII` des lettres du mot "Bonjour" sont

$$\text{Bonjour} \mapsto 66\ 111\ 110\ 106\ 111\ 117\ 114$$

De plus, le code `ASCII` d'un espace est le nombre 32. Ainsi:

$$\begin{aligned}
\text{Bon} &\mapsto 66 + 111 \cdot 256 + 110 \cdot 256^2 = 7237442 \\
\text{jou} &\mapsto 106 + 111 \cdot 256 + 117 \cdot 256^2 = 7696234 \\
\text{r_ _} &\mapsto 114 + 32 \cdot 256 + 32 \cdot 256^2 = 2105458
\end{aligned}$$

$\mathcal{S}$ nb-tp3py.7. Puissance: algorithme naïf. [1 - B. Ischi 24-25]

Rédiger un programme *Python3* qui calcule

$$k^e \bmod n$$

où k, e et n sont des nombres entiers donnés par l'utilisateur, avec l'algorithme naïf. Comparer avec la fonction *Python3* pow. Voici un exemple de fonctionnement du programme:

```
k=2
e=500000000
n=13
k^e mod n= 9  (pownaif)
k^e mod n= 9  (pow Python3)
```

$\mathcal{S}$ nb-tp3py.8. Alogorithme rapide. [1 - B. Ischi 24-25]

Rédiger un programme *Python3* qui calcule

$$k^e \bmod n$$

où k, e et n sont des nombres entiers donnés par l'utilisateur, avec l'algorithme rapide. Comparer avec la fonction *Python3* pow. Voici un exemple de fonctionnement du programme:

```
k=13283482736498237492439
e=617523712387192379128738172937129366712537615 37
n=19827893719237189238
k^e mod n= 14675959440647468521  (powcdc)
k^e mod n= 14675959440647468521  (pow Python3)
```

$\mathcal{S}$ nb-tp3py.9. Clé RSA. [1 - B. Ischi 24-25]

(1) Déterminer l'action de la méthode generate.nextprime de la classe ntheory du module sympy en exécutant le script *Python3* suivant

```
1 import random
2 import sympy
3 n=random.randint(10**2,10**3-1)
4 print(n)
5 p=sympy.ntheory.generate.nextprime(n)
6 print(p)
```

(2) Rédiger un programme *Python3* qui génère une clé RSA avec des nombres premiers p et q de 300 chiffres choisis au hasard et un nombre e de 100 chiffres également choisi au hasard. Voici un exemple d'exécution du programme

```
p= 91695698391968019317755165705868574397897513923639387817519245628062996981723300278161679938307928041
79424187835791202499100636863874249345797019125320097657037407058063176098557544836896269131886872352630
91478926562869995656083434945684333615774100415773896719266084769356758411386836522079144602 7819

q= 77428691995704471806171202566726265746914703710240600433974302495985994795832393582345785115967602 0
43427462682298080321921293949451442364160578416272063327681139051499240087045216777294871217657010810 1
74724244073300814722993908816499711191319634017270736084032435645765511092299572939651463753265652 57

n= 70998779881227055839089943056453689240338369837932183069701441124276760550233338922496258974409224668
40617524318009497791578040162334392901321954812682393939965157173507255353157647371025401925745569661445
62152426841953478048901268617833008785256936156068223630686777214929715415271778321552130301565415772987
80894529823379784402981849502282370537859518474109914148749028860244274126886652365373086742345427016657
71277053189553607023806120057517187717750609476009425297113664632580025782646989850683209824701569891699
65015416194541424477269072843385398174224106623726328204213386512222773216013408844 83

e= 48637804972321527221519252101111200559223035237422424018572550326881647702757577601056371857963022927
```

```
d= 701852756618429489728886536236602104839384301898891203689001038768872415823022628537401955983439904157
829179491905125144612152072388757020568502677941074389162276857190113582654055504726940542456793841308785
479520096669540431513766367686199715928835724211804755568673749224384005492930497969779947502036312811232
913055878510888015437366776746665236321750938932052568933427534922133534621357831391274447446975219750800
943910465366015568597883477601432959253448566947629532656784482648432046101885812823239467924867601824084
070840214759696625786295525082226835029154458720117365963849313116910179432440 63

Contrôle (d*e-1)%phin= 0
```

$\mathcal{S}$ nb-tp3py.10. Cryptage et décryptage RSA. [1 - B. Ischi 24-25]

(1) Rédiger un programme *Python3* qui demande à l'utilisateur de
   (a) taper un texte (sans accents),
   (b) donner le n de la clé publique du destinataire,
   (c) donner le e de la clé publique du destinataire,

   et qui encrypte avec la méthode RSA le message en le décomposant par blocs de 10 lettres (**NB** on doit avoir $n > 256^{10}$). Voici un exemple du fonctionnement du programme:

```
Tapez un texte: Ceci est un message secret

Clé publique du destinataire: n=709987798812270558390899430564536892403383698379321830697014411242767605502333389224962589744092246684061752431800949779157804016233439290132195481268239393996515717350725535315764737102540192574556966144562152426841953478048901268617833008785256936156068223630686777214929715415271778321552130301565415772987808945298233797844029818495022823705378595184741099141487490288602442741268866523653730867423454270166577127705318955360702380612005751718771775060947600942529711366463258002578264698985068320982470156989169965015416194541424477269072843385398174224106623726328204213386512222773216013408844830 

Clé publique du destinataire: e=486378049723215272215192521011112005592230352374224240185725502 6881647702757577601056371857963022927

Message crypté:  [180658865610552767793030731578975979165090590776496900650583087400855631280303490321442037429483492079957409485455802613830251860209660003602298903513542551285696012300707632646443704358460005904067718442779376420193677700959146183378846286879853182566490367968814174997113990455236251788236593067302965586409935129174321700610561433135177963224941329131755661752779604290779457307826840109833607666806327159185710037045294238232683453111955844444571641402188506639104220751759593398746802164754115683356182450272661194847579928167008980479368384591939784502548648068827832192042122147857650681918680 56,2102342035733659887984326435393979383434306034577952537258189506474531827154685435992648653179243095024394700566262674701413318569230586201931892482259595994962834078175860012958285766455585742389274133233437233183300523675776036328400162604517565550247862642871341459261076163324964194583180323738920546982185272349649066412993278380656981666001062138140881164349426992119870635867446030969406664590597512974642708931117880729964861489470340720523341201189447144601036197064350222274384648525924918767571333985469533591418081058198611168311005981576118838994692934687949542782352201858535903534454040,72861750631477945320506081680881533620431882761792191533141257755495168115208742342833027290360316331483692892658206758454605390008128388127196077823861591240238864827186147259163870534697727541186539629564739511852934888374704166874906744699275102362578402580070809061011015652451502793621225704723739501179139421678612922658225637980275483056129304574167420938840543348404119652155088932917117077702778778855262543835638122901754811722054864551782696237760418072168079405561438871664637661203014897881753580925590704506434887345591044369812530471541100675269752813144648492017596259426274706347426 13]
```

(2) Rédiger un programme *Python3* permet de décrypter le message crypté par le programme du point (1). Le programme demande à l'utilisateur de
   (a) taper le message crypté (sans les parenthèses),
   (b) donner le n de la clé publique du destinataire,
   (c) donner le d de la clé privée du destinataire,

   et qui affiche le message décrypté Voici un exemple du fonctionnement du programme:

```
Message crypté: 180658865610552767793030731578975979165090590776496900650583087400855631280303490321442037429483492079957409485455802613830251860209660003602298903513542551285696012300707632646443704358460005904067718442779376420193677700959146183378846286879853182566490367968814174997113990455236251788236593067302965586409935129174321700610561433135177963224941329131755661752779604290779457307826840109833607666806327159185710037045294238232683453111955844444571641402188506639104220751759593398746802164754115683356182450272611948475799281670089804793683845919397845025486480688278321920421221478576506819186805 6,
```

```
210234203573365988798432643539397938343430603457795253725818950647453182715468543599264865317924309502439470056626267470141331856923058620193189248225959599496283407817586001295828576645558574238927413323343723318330052367577603632840016260451756555024786264287134145926107616332496419458318032373892054698218527234964904664129932783806569816660010621381408811643494269921198706358674460309694066645905975129746427089311178807299648614894703407205233412011894471446010361970643502222743846485259249187675713339854695335914180810581986111683110059815761188389946929346879495427823522018585359035344540 40,
728617506314779453205060816808815336204318827617921915331412577554951681152087423428330272903603163314836928926582067584546053900081283881271960778238615912402388648271861472591638705346977275411865396295647395118529348883747041668749067446992751023625784025800708090610110156524515027936212257047237395011791394216786129226582256379802754830561293045741674209388405433484041196521550889329171170777027787788552625438356381229017548117220548645517826962377604180721680794055614388716646376612030148978817535809255907045064348873455910443698125304715411006752697528131446484920175962594262747063474261 3
Clé publique du destinataire: n=709987798812270558390899430564536892403383698379321830697014411242767605502333389224962589744092246684061752431800949779157804016233439290132195481268239393996515717350725535315764737102540192574556966144562152426841953478048901268617833008785256936156068223630686777214929715415271778321552130301565415772987808945298233797844029818495022823705378595184741099141487490288602442741268866523653730867423454270166577127705318955360702380612005751718771775060947600942529711366463258002578264698985068320982470156989169965015416194541424477269072843385398174224106623726328204213386512222773216013408844833209246791248676018240840708402147596966257862955250822268350291544587201173659638493131169101794324406 3

Clé privée du destinataire: d=701852756618429489728886536236602104839384301898891203689001038768872415823022628537401955983439904157829179491905125144612152072388757020568502677941074389162276857190113582654055504726940542456793841308785479520096669540431513766367686199715928835724211804755568673749224384005492930497969779947502036312811232913055878510880154373667767466652363217509389320525689334275349221335346213578313912744474469752197508009439104653660155685978834776014329592534485669476295326567844826484320461018858128232394679248676018240840708402147596966257862955250822268350291544587201173659638493131169101794324406 3

Texte décrypté:  Ceci est un message secret
```

NB Pour transformer le message crypté en liste de nombre, on peut utiliser le code *Python3* suivant:

```python
M0=input("Message crypté: ")
M0=M0.split(",")
M=[]
for x in M0:
    M.append(int(x))
```