Exercice 1

Qu'affichera le programme suivant ?

```
x=17
def f(x):
  x=(x+3)%6
  print("x=",x)

def g():
  global x
  x=(x+3)%6
  print("x=",x)

def h(x):
  x=(x+3)%6
  print("x=",x)
  return x

f(x)
print("x=",x)
g()
print("x=",x)
x=h(x)
print("x=",x)
```

```
x=2
x=17
x=2
x=2
x=5
x=5
```

Exercice 2

Dans cet exercice "Algorithme de décomposition primaire d'un entier", on se propose d'écrire un algorithme pour décomposer un entier en produit de nombres premiers
On définit la valuation p-adique [de n] pour p nombre premier et n entier naturel non nul.
Si p divise n, on note $v_p(n)$ le plus grand entier k tel que p^k divise n.
Si p ne divise pas n, on pose $v_p(n) = 0$.
L'entier $v_p(n)$ s'appelle la valuation p-adique de n.

1) Écrire une fonction **estPremier(n)** qui prend en argument un entier naturel non nul n et qui renvoie le booléen True si n est premier et le booléen False sinon.

```
def estPremier(n):
  for i in range(2,n//2+1):
    if n%i==0:
      return False
  return True
```

2) En déduire une fonction **liste_premiers(n)** qui prend en argument un entier naturel non nul n et renvoie la liste des nombres premiers inférieurs ou égaux à n.

```
def liste_premiers(n):
  return [i for i in range(2,n+1) if estPremier(i)]
```

3) Écrire une fonction **valuation_p_adique(n,p)** qui prend en arguments un entier naturel n non nul et un nombre premier p et renvoie la valuation p−adique de n. Par exemple, puisque $40 = 2^3 * 5$, valuation_p_adique(40,2) renvoie 3, valuation_p_adique(40,5) renvoie 1 et valuation_p_adique(40, 7) renvoie 0.

```
def valuation_p_adique(n,p):
  k=0
  while n%p**k==0:
    k=k+1
  return k-1
```

Exercice 3

Un ensemble mathématique est un groupement d'objets distincts, appelés éléments de cet ensemble.
La théorie des ensembles est l'étude des propriétés et des opérations sur des ensembles (appartenance, inclusion, réunion, . . .). Elle représente une branche essentielle des mathématiques.
On s'intéresse ici aux algorithmes réalisant quelques traitements sur des ensembles mathématiques finis de nombres entiers.
Notation d'un ensemble fini d' éléments : Si E est un ensemble fini de N éléments $(N < 0)$, $e_0, e_1, ..., e_{N-1}$, alors E sera note ainsi $E = \{e_0, e_1, ..., e_{N-1}\}$.
Il s'agit de représenter les ensembles par des liste : E = [$e_0, e_1, ..., e_{N-1}$]

1) Ecrire une fonction **estEnsemble(E)** qui prend en argument une liste E et renvoie True si E est un ensemble, sinon elle renvoie False.

```
def estEnsemble(E):
  for x in E:
    if E.count(x)>=2:
      return False
  return True
```

2) Ecrire une fonction **appartient(E,x)** qui prend en argument une liste E et un entier x, elle renvoie True si x ∈ E, sinon elle renvoie False.

```
def appartient(E,x):
  return x in E
```

3) Ecrire une fonction **intersection(E,F)** qui prend en argument deux listes E et F, elle renvoie $G = E \cap F$.

```
def intersection(E,F):
  return [x for x in E if x in F]
```

4) Ecrire une fonction **union(E,F)** qui prend en argument deux listes E et F, elle renvoie $G = E \cup F$.

```
def union(E,F):
  return E +[x for x in F if x not in E]
```

5) Ecrire une fonction **égale(E,F)** qui prend en argument deux listes E et F, elle renvoie True si E = F, sinon elle renvoie False.

```
def egale(E,F):
  E.sort()
  F.sort()
  return E==F
```

6) Ecrire une fonction **inclus(E,F)** qui prend en argument deux listes E et F, elle renvoie True si $E \subset F$, sinon elle renvoie False.

```
def inclus(E,F):
  return [x for x in E if x not in F]==[]
```

7) Ecrire une fonction **complement(E,Ω)** qui prend en argument deux listes E et Ω, elle renvoie $\overline{E}$, le complément de E dans Ω.

```
def complement(E,omega):
  return [x for x in omega if x not in E]
```

8) Ecrire une fonction **DifferenceSymetrique(E,F,Ω)** qui prend en argument trois listes E, F et Ω, elle renvoie la différence symétrique entre E et F.
on note $E \bigtriangleup F = (E \cap \overline{F}) \cup (F \cap \overline{E})$

```
def DifferenceSymetrique(E,F,omega):
  return union(intersection(E,complement(F,omega)),intersection(complement(E,omega),F)))
```

9)Ecrire une fonction **ProduitCartesien(E,F)** qui prend en argument deux listes E et F, elle renvoie le produit cartesien de E et F. on note $E \times F = \{(x,y) | x \in E, y \in F\}$

```
def ProduitCartesien(E,F):
  return [(x,y) for x in E for y in F]
```