CHAPITRE 1

RAPPELS DE PYTHON – SYNTAXE ET STRUCTURES DE DONNÉES

```
''' Dans les premiers exemples, on précise la signature des fonctions
    mais ce n'est pas indispensable. '''

## page 1

def f(x: float) -> float:
    """ entrée : flottant x
        sortie : valeur de x**2+x-1
    """
    return x**2 + x - 1

from math import exp

def g(x: float) -> float:
    """ entrée : flottant x
        sortie : valeur de exp(x+1)
    """
    return exp(x+1)


## page 2

def dernier(ch: str) -> str:
    """ entrée : chaîne de caractères non vide
        sortie : dernier caractère
    """
    return ch[-1] # ou ch[len(ch)-1]


def f(a: float, b: float, x:float) -> float:
    """ entrée : a, b, x flottants
        précondition : a < b
        sortie : valeur de la fonction de l'énoncé en x
    """
    if x >= b:
        return 1
    elif x <= a:
        return 0
    else:
        return (x-a)/(b-a)


## page 3

def impairs(n: int) -> int:
    """ entrée : entier naturel n
        sortie : somme des n premiers nombres impairs
    """
    s = 0
    for k in range(n):
        s = s + 2*k+1
    return s
```

```python
## page 4
''' Dans l'exemple suivant "plus grande puissance de 2" est interprété
    comme étant l'exposant de ladite puissance (i.e. 4 pour 2**4=16). '''

def puissance(n: int) -> int:
    """ entrée : entier naturel n
        sortie : plus grande puissance de 2 inférieure ou égale à n
    """
    p = 0
    while 2**p <= n:
        p = p + 1
    return p-1

def puissance_bis(n: int) -> int:
    """ entrée : entier naturel n
        sortie : plus grande puissance de 2 inférieure ou égale à n
    """
    p = 0
    puiss = 1
    while puiss <= n:
        p = p + 1
        puiss = puiss * 2
    return p-1



## sur les listes

def appartient(a, L):
    ind = 0
    while ind < len(L):
        if L[ind] == a:
            return True
        ind += 1
    return False

def appartientbis(L, a):
    ind = 0
    while ind < len(L) and L[ind] <= a:
        if L[ind] == a:
            return True
        ind += 1
    return False

def seuil(L, x):
    lis = []
    for y in L:
        if y >= x:
            lis.append(y)
    return lis

def extremes(L):
    m, M = L[0], L[0]
    ind = 1
    while ind < len(L):
        if L[ind] > M:
            M = L[ind]
        if L[ind] < m:
            m = L[ind]
        ind += 1
    return (m, M)

def indmax(L):
    i, M = 0, L[0]
    ind = 1
    while ind < len(L):
        if L[ind] >= M:
            M = L[ind]
            i = ind
        ind += 1
    return i
```

```python
def nombre(lis, x, n):
    c = 0
    for element in lis:
        if element == x:
            c += 1
    return c == n

def deuxieme(lis):
    a = lis[0]
    b = lis[1]
    if a > b:
        a, b = b, a
    for k in range(2,len(lis)):
        c = lis[k]
        if c > b:
            a, b = b, c
        elif c > a:
            a = c
    return(a)

def distmin(lis):
    n = len(lis)
    d = abs(lis[1]-lis[0])
    for i in range(n):
        for j in range(n):
            e = abs(lis[i] - lis[j])
            if 0 < e < d:
                d = e
    return d

# un jeu de tests pour cette dernière fonction:
assert distmin([2, 5, 6, 0, 11]) == 1
assert distmin([2, 5, 6, 0, 6, 11]) == 1
assert distmin([2, 6, 5, 0, 11]) == 1



## Sur les piles

# implémentation avec des listes

def creerPile():
    return []

def estVide(p):
    return p == []

def empiler(p, x):
    p.append(x)

def depiler(p):
    if estVide(p):
        return "pile vide"
    else:
        return p.pop()



def sommet(p):
    """ entrée : p pile
        sortie : sommet de p """
    if not estVide(p):
        a = depiler(p)
        empiler(p, a)
        return a
```

```
def echange(p):
    """ entrée : p pile
        sortie : échange des deux éléments au sommet de p;
                 ne fait rien si p a au plus un élément """
    if not estVide(p):
        a = depiler(p)
        if not estVide(p):
            b = depiler(p)
            empiler(p, a)
            empiler(p, b)
        else:
            empiler(p, a)


def taille(p):
    """ entrée : p pile
        sortie : taille de p (entier), sans détruire p """
    n = 0
    q = creerPile()
    while not estVide(p):
        empiler(q, depiler(p))
        n += 1
    while not estVide(q):
        empiler(p, depiler(q))
    return n



## Sur les files

# implémentation avec deque

from collections import deque

def creerFile():
    return deque()

def estVide(f):
    return f == deque([])

def enfiler(f, x):
    return f.append(x)

def defiler(f):
    return f.popleft()


def consulter(f):
    """ entrée : file f
        précondition : f non vide
        sortie : renvoie la valeur de tête de f sans modifier f """
    g = creerFile()
    a = defiler(f)
    enfiler(g, a)
    while not estVide(f):
        enfiler(g, defiler(f))
    while not estVide(g):
        enfiler(f, defiler(g))
    return a



## Sur les dictionnaires

def compte(ch):
    d = {}
    for l in ch:
        if l in d:
            d[l] += 1
        else:
            d[l] = 1
    return d
```

```
test = "sapristi, voici venir ici des ex-pcsi !"
print(test)
print(compte(test))


def traduction(dico):
    dinv = {}
    for word in list(dico.items()):
        for mot in word[1]:
            if mot in dinv:
                dinv[mot].append(word[0])
            else:
                dinv[mot] = [word[0]]
    return dinv

test = {'make' : ['faire', 'fabriquer', 'réaliser'], 'do' : ['faire'], 'realize' : ['réaliser'
    , 'prendre conscience', 'parvenir'], 'produce' : ['réaliser', 'produire', 'fabriquer']}
print(test)
print(traduction(test))



## Exemples pages 18 à 20

# Exemple 1

def sa(g):
    """ entrée : dictionnaire donnant la liste d'adjacence g d'un graphe non orienté
        sortie : couple nb de sommets, nb d'arêtes de g
    """
    s, a = 0, 0
    for t in g:
        s += 1
        a += len(g[t])
    return s, a//2


G1 = {'a' : ['b'],
      'b' : ['a', 'c', 'd'],
      'c' : ['b', 'd'],
      'd' : ['b', 'c'],
      'e' : ['f'],
      'f' : ['e'],
      'g' : []}
assert sa(G1) == (7, 5)

G2 = {'a': [('b', 8), ('c', 2), ('d', 1)],
      'b': [('a', 8), ('c', 4), ('d', 6), ('e', 1)],
      'c': [('a', 2), ('b', 4), ('e', 2)],
      'd': [('a', 1), ('b', 6)],
      'e': [('b', 1), ('c', 2)] }
assert sa(G2) == (5, 7)


# Exemple 2

def degs(g, s):
    """ entrée : dictionnaire donnant la liste d'adjacence g d'un graphe non orienté,
                   somme s de g
        sortie : degré du sommet s de g
    """
    return len(g[s])


assert degs(G1, 'a') == 1
assert degs(G1, 'b') == 3
assert degs(G1, 'g') == 0
```

```python
# Exemple 3

def ajar(g, l):
    """ entrée : dictionnaire donnant la liste d'adjacence g d'un graphe non orienté,
                 liste de deux sommets [a, b] de g
        sortie : dictionnaire donnant la liste d'adjacence du graphe
                 obtenu à partir de g en ajoutant une éventuelle arête entre a et b
    """
    a, b = l
    if b not in g[a]:
        g[a].append(b)
        g[b].append(a)


# Exemple 4

def aretes(gr):
    """
    Renvoie la liste des arêtes du graphe gr (représenté par une liste d'adjacence).
    Chaque arête {u, v} est représentée par la liste [u, v] ou [v, u]
    """

    ar = []

    for s1 in gr:
        for s2 in gr[s1]:
            if  [s1, s2] not in ar and [s2, s1] not in ar:
                ar.append([s1, s2])

    return ar


# Exemple 5

def degre_liste(gr, s):
    """
    Renvoie le degré de s dans le graphe gr, représenté par une liste d'adjacence.
    """
    return len(gr[s])


def listed(gr, d):
    """
    Renvoie la liste classée par ordre croissant lexicographique
    des sommets du graphe gr dont le degré est d.
    """

    return sorted([s for s in gr if degre_liste(gr, s) == d])


# Exemple 6

def graphe_complet(n):
    """
    Renvoie le dictionnaire correspondant au graphe complet ayant n
    sommets dont les sommets sont les entiers de 0 à n-1.
    """

    gr = {}
    for i in range(n):
        gr[i] = [k for k in range(n) if k != i]
    return gr


# Exemple 7

def biparti(gr, S):
    for s in gr:
        for v in gr[s]:
            if (s in S and v in S) or (s not in S and v not in S):
                return False
    return True
```