Thèmes d'étude

J.-P. Berne – E. Durif – X. Pessoles

Thème : Fonctions récursives.
Commentaires :
- version récusive d'algorithmes dichotomiques ;
- fonctions produisant à l'aide de `print` successifs des figures alphanumériques ;
- dessins de fractales ;
- énumération des sous-listes ou des permutations d'une liste.

On évite de se cantonner à des fonctions mathématiques (factorielles, suites récurrentes). On peut montrer le phénomène de dépassement de la taile de la pile.

1 Définitions

Définition Une fonction récursive est une fonction qui s'appelle elle-même.
On appelle récursion l'appel de la fonction à elle-même.

La programmation est un paradigme de programmation au même titre que la programmation itérative. Un programme écrit de manière récursive peut être traduit de manière itérative, même si dans certain cas, cela peut s'avérer délicat.

Méthode
- Une fonction récursive doit posséder une condition d'arrêt (ou cas de base).
- Une fonction récursive doit s'appeler elle-même (récursion).
- L'argument de l'étape de récursion doit évoluer de manière à se ramener à la condition d'arrêt.

▪ **Exemple** Prenons pour exemple du calcul de $n! = \prod_{i=1}^{n} i$ et $n! = \begin{cases} 1 & \text{si } n = 0 \\ n \times (n-1)! & \text{si } n \geq 1 \end{cases}$.

```
def fact_it(n : int) -> int :
    res = 1
    for i in range (1,n+1): # ou range(n)
        res = res*i
    return res
```

```
def fact_rec(n : int) -> int :
    if n == 0 :
        return 1
    else :
        return n * fact_rec(n-1) # Ré
            cursion
```

▪

2 Exemples

2.1 Suites récurrentes

Soit la suite u_n définie par récurrence pour tout $n \in \mathbb{N}^*$ par $\begin{cases} u_1 = 1 \\ u_{n+1} = \dfrac{u_n + 6}{u_n + 2} \end{cases}$.

```
def un_it (n : int) -> float :
    if n == 1 :
        return 1
    else :
        u = 1
        for i in range(2,n+1):
            u = (u+6)/(u+2)
        return u
```

```
def un_rec (n : int) -> float :
    if n == 1 :
        return 1
    else :
        return (un_rec(n-1)+6)/(un_rec(n-1)
            +2)
```

2.2 Recheche dichotomique

```
def appartient_dicho(e : int , t : list) -> bool:
    """Renvoie un booléen indiquant si e est dans t
    Préconditions : t est un tableau de nombres trié par ordre croissant e est un nombre"""
    g = 0 # Limite gauche de la tranche où l'on recherche e
    d = len(t)-1 # Limite droite de la tranche où l'on recherche e
    while g <= d: # La tranche où l'on cherche e n'est pas vide
```

```
        m = (g+d)//2 # Milieu de la tranche où l'on recherche e
        pivot = t[m]
        if e == pivot: # On a trouvé e
            return True
        elif e < pivot:
            d = m-1 # On recherche e dans la partie gauche de la tranche
        else:
            g = m+1 # On recherche e dans la partie droite de la tranche
    return False
```

```
def appartient_dicho_rec(e : int , t : list) -> bool:
    """Renvoie un booléen indiquant si e est dans t
    Préconditions : t est un tableau de nombres trié par ordre croissant e est un nombre"""
    g = 0 # Limite gauche de la tranche où l'on recherche e
    d = len(t)-1 # Limite droite de la tranche où l'on recherche e
    while g <= d: # La tranche où l'on cherche e n'est pas vide
        m = (g+d)//2 # Milieu de la tranche où l'on recherche e
        pivot = t[m]
        if e == pivot: # On a trouvé e
            return True
        elif e < pivot:
            d = m-1 # On recherche e dans la partie gauche de la tranche
            appartient_dicho_rec(e,t[g:d])
        else:
            g = m+1 # On recherche e dans la partie droite de la tranche
            appartient_dicho_rec(e,t[g:d])
    return False
```


2.3 Exponentiation rapide

L'exponentiation rapide est un algorithme permettant de calculer x^n en utilisant l'algorithme récursif suivant :

$$\text{puissance}(x,n) = \begin{cases} x & \text{si } n = 1 \\ \text{puissance}(x^2, n/2) & \text{si n est pair} \\ x \times \text{puissance}(x^2,(n-1)/2) & \text{si n est impar pair} \end{cases}.$$

```
def expo_it(x : float, n : int) -> float :
    res=1
    while n!=0:
        if n%2==1:
            res = res*x
        x*=x
        n//=2
    return res
```

```
def expo_rec(x : float, n : int) -> float :
    if n==0:
        return 1
    elif n%2==0:
        return expo_rec(x*x,n//2)
    else:
        return x*expo_rec(x*x,n//2)
```

Pour réaliser l'activité associée à ce cours, suivre le lien suivant : `https://bit.ly/3AmRgdH`

QCM

Question 1 *Que retourne la commande suivante* `mystere(4)` *?*

```
def mystere(n):
    if n>0 :
        return mystere(n-2)
    else :
        return n==0
```

1. 0.
2. False.
3. True.
4. L'exécution génère une erreur.

Question 2 *Laquelle de ces fonctions retourne True lorsqu'on exécute* `f(5)` *?*

```
def f1(n):
    if n==0 :
        return True
    else :
        return f(n-2)

def f2(n):
    if n<=0 :
        return True
    else :
        f(n-2)

def f3(n):
    if n<=0 :
        return True
    return f(n-2)

def f4(n):
    if n==0 :
        return True
    f(n-2)
```

1. `f1`.
2. `f2`.
3. `f3`.
4. `f4`.

Question 3 *Quel affichage obtient-on en exécutant* `affiche(3)` *?*

```
def affiche(n):
    print(n)
    if n>=0:
        affiche(n-1)
```

1. 3, 2, 1, 0 (avec des retours à la ligne entre chaque valeurs).
2. 0, 1, 2, 3 (avec des retours à la ligne entre chaque valeurs).
3. 3, 2, 1, 0, -1 (avec des retours à la ligne entre chaque valeurs).
4. 3.

Question 4 *Une seule des fonctions définies ci-dessous retourne* `'ccccc'` *à l'appel de* `replique(5,'c')`. *Déterminer laquelle.*

```
def replique(a,b): # Fonction 1
    if a==1:
        return b
    else :
        return replique( a-1 , b+b)

def replique(a,b): # Fonction 2
    if a==1:
```

```
        return b
    elif a%2 == 0:
        return replique( a-2 , b+b)
    else :
        return b + replique( a-2 , b+b)

def replique(a,b): # Fonction 3
    if a==1:
        return b
    elif a%2 == 0:
        return replique( a//2 , b+b)
    else :
        return b + replique( a//2 , b+b)

def replique(a,b): # Fonction 4
    if a==1:
        return b
    else :
        replique( a-1 , b+b)
```

1. Fonction 1.
2. Fonction 2.
3. Fonction 3.
4. Fonction 4.

Question 5 *Que retourne l'instruction* `copy(3,'A')` *?*

```
def copy(n,s):
    if n==0:
        return s
    return copy(n-1, s+s)
```

1. `'AAA'`.
2. `'AAAAAA'`.
3. `'AAAAAAAA'`.
4. `'3A'`.

Question 6 *Que retourne l'instruction :* `mystere(3,'$')` *?*

```
def mystere(n,s):
    if n==0:
        return s
    return s + mystere(n-1, s)
```

1. `'$$$'`.
2. `'$2$'`.
3. `'$$$$'`.
4. L'exécution déclenche une erreur.

Question 7 *Que retourne la commande* `f(3,4)` *?*

```
def f(a,b):
    if a == 0 :
        return b
    return f(a-1, b+1)
```

1. 4.
2. 5.
3. 6.
4. 7.

Question 8 *Que retourne la commande* `mystere(3)` *?*

```
def mystere(n):
    if n>0 :
        return mystere(n-2)
    else :
        return n==0
```

1. True.
2. False.
3. RecursionError.
4. 0.

Question 9 *On propose de créer une fonction récursive permettant de calculer x^n. Compléter la fonction proposée.*

```
def puissance(x,n):
    if n > 0 :
        return .......
    return 1
```

1. `puissance(x,n-1)`.
2. `x*puissance(x,n-1)`.
3. Quoi que l'on écrive, cette fonction ne donnera pas le résultat attendu.
4. `x**(n-1)*puissance(x,n-1)`.

Question 10 *Que renvoi ce programme en console ?*

```
def ed(L,M=[]):
    if len(L) == 0 : return M
    a=L.pop()
    if a not in M : M.append(a)
    return ed(L,M)
L=[2, 3, 2, 6, 8, 9, 9, 10, 9, 3, 6, 7, 8, 8, 9]
print(ed(L))
```

1. `None`.
2. `[9, 8, 7, 6, 3, 10, 2]`.
3. `[9, 8, 8, 7, 6, 3, 9, 10, 9, 9, 8, 6, 2, 3, 2]`.
4. `[2, 10, 3, 6, 7, 8, 9]`.

Question 11 *Que retourne le programme suivant ?*

```
def A(x):
    if x <= 1 : return x
    return B(x+1)

def B(x) :
    return A(x-2)+4

print(A(4))
```

1. 13.
2. 1.
3. 12.
4. Une erreur de type : "RecursionError : maximum recursion depth exceeded in comparison".