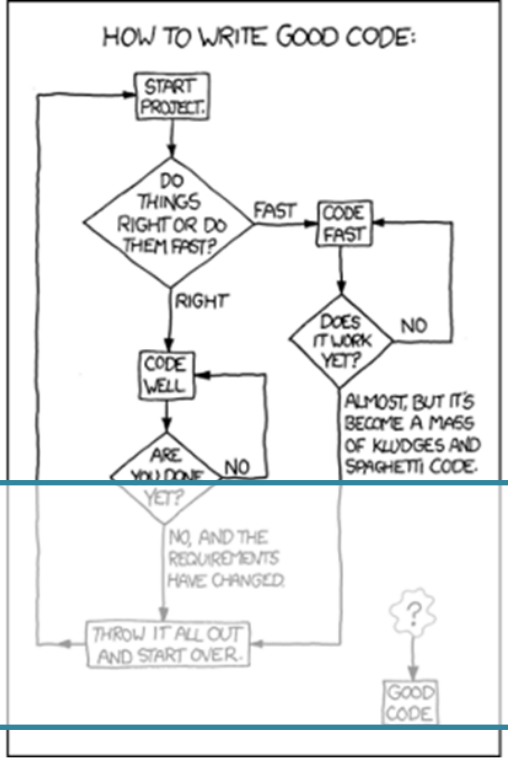
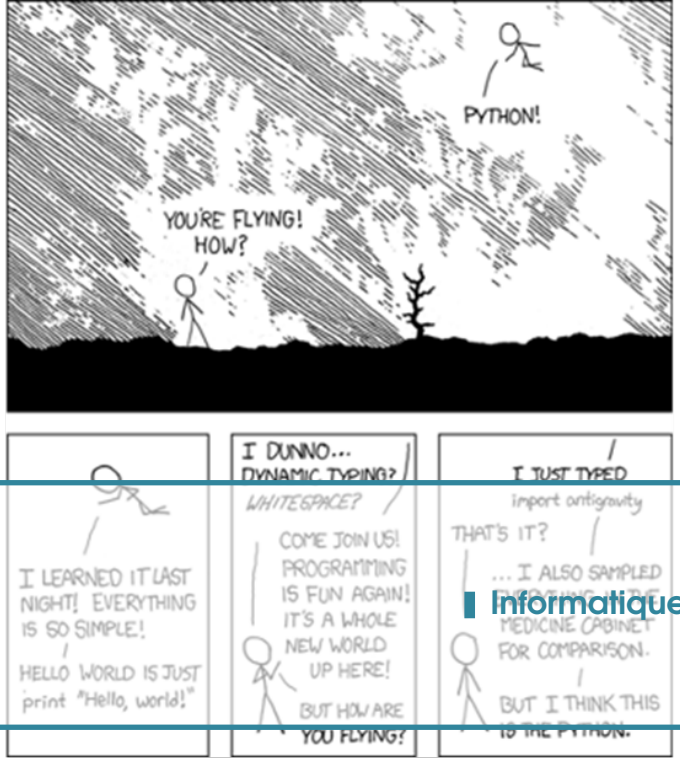




Fiche

Introduction à la programmation en Python
Informatique

1	Définitions	2
2	Suites définies par récurrence	2
3	Algorithmes dichotomiques – Divisier pour régner	2
4	Tracer de figures définies par récursivite	3
5	Activité préparatoire	4
6	QCM	5

E. Durif – X. Pessoles
J.-P. Berne
MPSI

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.

2 Suites définies par récurrence

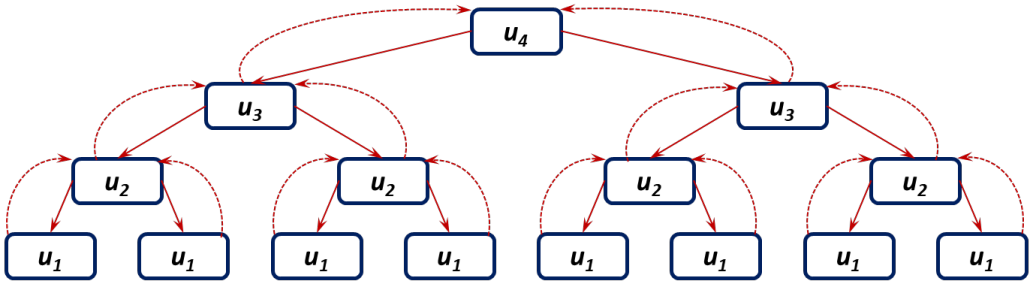
Les suite définies par récurrence pour les quelles $u_n = f(u_{n-1}, u_{n-2}, ...)$ sont des cas d'application directs des fonctions récursives.

Par exemple, soit la suite u_n définie par récurrence pour tout $n \in \mathbb{N}^*$ par $\begin{cases} u_1 = 1 \\ u_{n+1} = \dfrac{u_n + 6}{u_n + 2} \end{cases}$. Il est possible de calculer le n^{e}terme par un algorithme itératif ou un algorithme récursif.

```
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
```

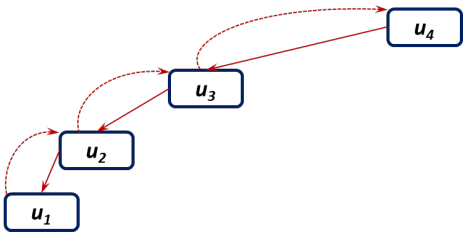
```
def un_rec (n : int) -> float :
    if n == 1 :
        return 1
    else :
        return (un_rec(n-1)+6)/(un_rec(n-1)+2)
```

La figure suivante montre que dans le cas de l'algorithme récursif proposé plusieurs termes sont calculés à plusieurs reprises ce qui constitue une perte de temps et d'espace mémoire.



Une autre formulation de l'algorithme récursif permet très simplement de diminuer le nombre de termes calculés.

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



3 Algorithmes dichotomiques – Divisier pour régner

Les algorithmes dichotomiques se prètent aussi à des formulations récursives. Prenons comme exemple la recherche d'un élément dans une liste triée. L'algorithme de gauche propose une version itérative. L'algorithme de droite une version récursive.

```
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"""
    # Limite gauche de la tranche où l'on recherche e
    g = 0
    # Limite droite de la tranche où l'on recherche e
    d = len(t)-1
    # La tranche où l'on cherche e n'est pas vide
    while g <= d:
        # Milieu de la tranche où l'on recherche e
        m = (g+d)//2
        pivot = t[m]
        if e == pivot: # On a trouvé e
            return True
        elif e < pivot:
            # On recherche e dans la partie gauche de la tranche
            d = m-1
        else:
            # On recherche e dans la partie droite de la tranche
            g = m+1
    return False
```

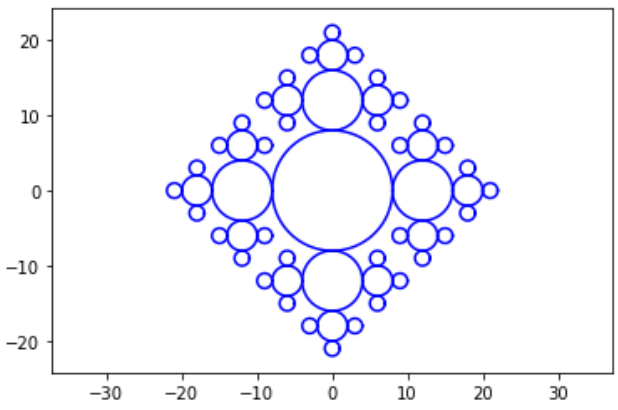
```
def appartient_dicho_rec(e : int , t : list) -> bool:
    """Renvoie un booléen indiquant si e est dans t. Préconditions : t est un tableau de nombres trié par ordre croissant e est un nombre"""
    # Limite gauche de la tranche où l'on recherche e
    g = 0
    # Limite droite de la tranche où l'on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

4 Tracer de figures définies par récursivite

Un grand nombre de figures peuvent être tracés en utilisant des algorithmes récursifs (flocon de Koch, courbe de Peano, courbe du dragon *etc.*).

```
import matplotlib.pyplot as plt
import numpy as np
def cercle(x,y,r):
    theta = np.linspace(0, 2*np.pi, 100) #des points régulièrement espacés dans l'intervalle [0,2pi]
    X = [x+r*np.cos(t) for t in theta] #abscisses de points du cercle C((x,y),r)
    Y = [y+r*np.sin(t) for t in theta] #ordonnées de points du cercle C((x,y),r)
    plt.plot(X,Y,"b") #tracé sans affichage
```

```
def bubble(n, x, y, r, d):
    cercle(x, y, r)
    if n > 1:
        if d != 's':
            bubble(n-1,x,y+3*r/2,r/2,"n")
        if d != 'w':
            bubble(n-1,x+3*r/2,y,r/2,"e")
        if d != 'n':
            bubble(n-1,x,y-3*r/2,r/2,"s")
        if d != 'e':
            bubble(n-1,x-3*r/2,y,r/2,"w")
bubble(4,0,0,8,"")
plt.axis("equal")
plt.show()
```



5 Activité préparatoire

Pour réaliser l'activité associée à ce cours, suivre le lien suivant :

- Sujet : `https://bit.ly/3zkiJgb`
- Corrige : `https://bit.ly/3zl7QKJ`

6 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".