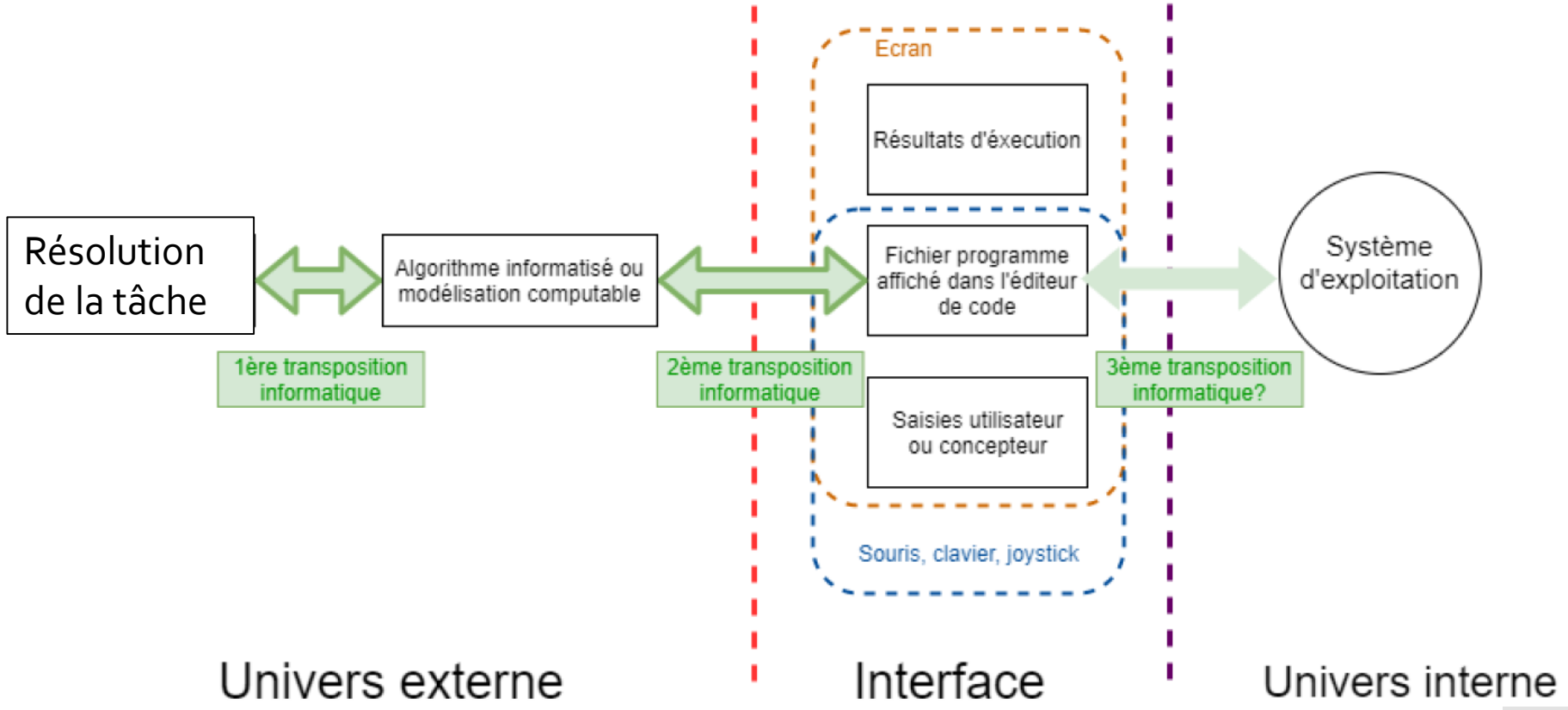
Sensibilisation à la didactique de l'informatique (2)

Laurianne Foulquier d'après un travail de **Christophe Declercq**

Laurianne.foulquier@u-bordeaux.fr

Bilan séance précédente



Extrait du mémoire de Géraldine Olivier Master MEEF 2021

I. Compétences informatiques

II. Retour sur les deux TP

Si vous deviez identifier des compétences majeures en informatique dans le contexte de la programmation, quelles seraient-elles et pourquoi ?

I. Compétences informatiques « Computational thinking »

II. Présentation d’un exemple

- L'expression "computational thinking" introduite par Jeannette Wing fait référence à des compétences et des habiletés humaines, pouvant être développées à l'occasion d'activité de programmation, et transférables à bien d'autres situations de type "résolution de problème".
- Il ne s'agit pas de penser comme une machine, mais de décrire les compétences cognitives en jeu pour résoudre - entre autres - un problème informatique - ou plutôt pour le faire résoudre par une machine.
- Il s'agit donc d'activité cognitive de haut niveau et donc bien d'une activité humaine. L'énumération des compétences en jeu fait débat. La traduction française aussi fait débat. "Thinking" aurait pu être traduit par "réflexion". Le terme "computationnel" n'existant pas, l'adjectif "calculatoire" est le plus proche mais semble réducteur.

I. Compétences informatiques

II. Présentation d'un exemple

- **Evaluer** : capacité à attribuer mentalement une valeur (résultat, type…) à un programme donné.
- **Anticiper** : capacité à se mettre dans la posture du programmeur qui doit décrire dans un algorithme l'enchaînement séquentiel/répétitif/conditionnel des instructions, avant même le début de son exécution.
- **Décomposer** : capacité à transformer un problème complexe en un ensemble de problèmes plus simples équivalents au problème initial.
- **Généraliser** : capacité à inférer un problème général à partir d'une instance de ce problème, et à repérer dans un problème particulier la répétition de traitements ou de données suivant un même schéma.
- **Abstraire** : capacité à "faire abstraction" des informations non pertinentes et à créer des solutions où la manière dont un problème est résolu peut être "abstraite" à l'aide d'une interface pertinente.

I. Compétences informatiques

II. Présentation d'un exemple

Extrait du préambule commun aux programmes de première et de terminale NSI

Il permet de développer des compétences :

- analyser et modéliser un problème en termes de flux et de traitement d'informations ;
- **décomposer** un problème en sous-problèmes, reconnaître des situations déjà analysées et réutiliser des solutions ;
- **concevoir des solutions algorithmiques** ;
- traduire un algorithme dans un langage de programmation, en spécifier les interfaces et les interactions, comprendre et réutiliser des codes sources existants, développer des processus de mise au point et de validation de programmes ;
- mobiliser les concepts et les technologies utiles pour assurer les fonctions d'acquisition, de mémorisation, de traitement et de diffusion des informations ;
- développer des capacités d'**abstraction** et de **généralisation**.

EVALUER

I. Compétences informatiques
EVALUER

II. Présentation d'un exemple

- Capacité à attribuer mentalement une valeur (résultat, type…) à un programme donné.

- L'évaluation d'un programme - lui donner une valeur - peut se faire de différentes manières dans différents domaines. Le plus simple est bien sûr d'évaluer le résultat que donne le programme à l'exécution. Mais il est aussi utile de savoir évaluer le **type** du résultat sans chercher nécessairement à connaître sa valeur.

I. Compétences informatiques EVALUER

II. Présentation d’un exemple

Evaluer c'est aussi donner la valeur du résultat de l'exécution - ce que fournit l'interprète.

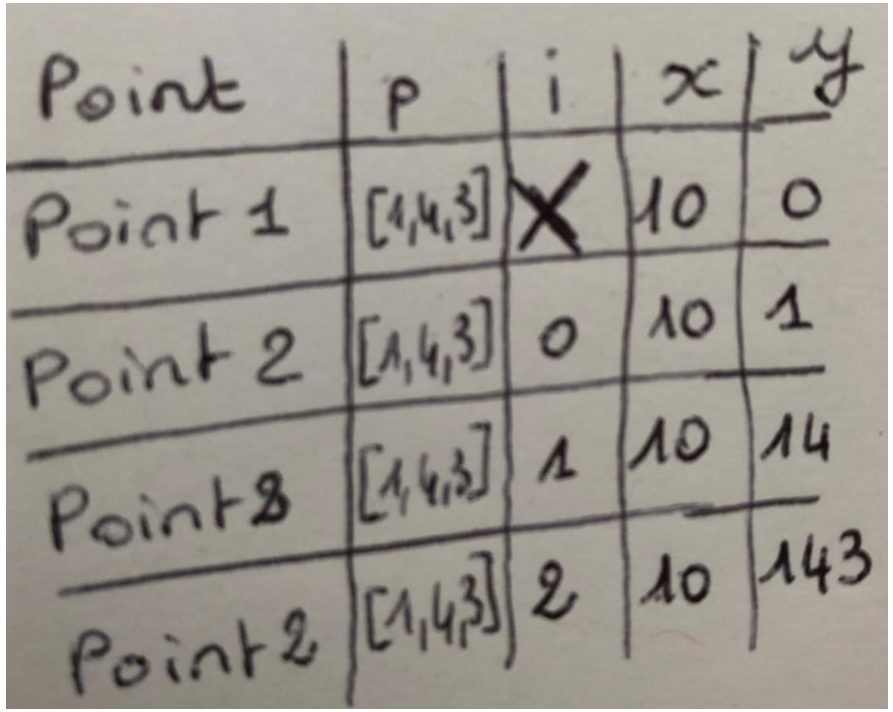
L'évaluation mentale peut être concrète. On peut construire à la main un tableau d'exécution du programme en notant sur chaque ligne les valeurs prises par les variables à certains points d'observation du programme :

I. Compétences informatiques EVALUER

II. Présentation d'un exemple

Exemple : on évalue le résultat du programme suivant.

```
def EvaluerPolynome(p,x):
    y = 0 # Point d'observation 1
    for i in range (len(p)):
        y = y * x + p[i]  # Point d'observation 2
    return(y)
EvaluerPolynome([1,4,3], 10)
```



Point	p	i	x	y
Point 1	[1,4,3]	X	10	0
Point 2	[1,4,3]	0	10	1
Point 3	[1,4,3]	1	10	14
Point 2	[1,4,3]	2	10	143

I. Compétences informatiques EVALUER

II. Présentation d'un exemple

TP : La bataille navale

Exercice 1

On considère le programme (rudimentaire, à ce stade) de bataille navale.

```
a=4
b=7
print('A vous de jouer')
x=int(input('Donner la coordonnee x :'))
y=int(input('Donner la coordonnee y :'))
if x==a and y==b:
    print('Coulé')
elif x==a or y==b:
    print('En vue')
else:
    print('A leau')
```

1) Décrire **en français**, étape par étape, le comportement de ce programme.
2) Proposer un jeu de test satisfaisant pour tester la conformité de ce programme avec un embryon de bataille navale.
3) *En TP : Implémenter ce script et le tester conformément à la question 2).*

Le but de ce TP est d'améliorer petit à petit ce programme pour qu'il remplisse des conditions plus proches d'une réelle partie de bataille navale.

x	y	x==a ?	y==b ?	Message affiché	Joue-t-on encore ?
2	4	$x \neq a$	$y \neq b$	"à l'eau"	Oui
4	5	$x = a$	$y \neq b$	"en vue"	oui
2	7	$x \neq a$	$y = b$	"en vue"	oui
4	7	$x = a$	$y = b$	"coulé"	non

I. Compétences informatiques EVALUER

II. Présentation d'un exemple

Algorithmique papier-crayon - Variable

On considère le programme suivant écrit en pseudo-langage informatique :

Variable : f_1 (type entier)
Variable : c_0 (type entier)

1 $f_1 \leftarrow 3 \times 4$
2 $f_1 \leftarrow f_1 \times 2$
3 $c_0 \leftarrow f_1 - 20$
4 $c_0 \leftarrow c_0 + f_1$
5 **Afficher**(c_0)

1. Pour chaque instruction numérotée :
   - Décrire précisément avec des phrases courtes en français ce qui se passe dans la mémoire de travail de l'ordinateur
   - Faire un dessin de la mémoire de travail, et représenter ce qu'y fait l'algorithme au cours de son exécution
2. Faire un tableau récapitulatif donnant le contenu de chaque variable du programme à la fin de chaque instruction

	Contenu de la variable f_1	Contenu de la variable c_0
A la fin de l'instruction 1 A la fin de l'instruction 2 ⋮		

3. Que va afficher le programme ci-dessous ?

I

a)

1/ L'ordinateur créer une variable f_1 et le processeur fait $3 \times 4 = 12$ et met 12 dans la variable qui l'a appelé f_1.

2/ L'ordi prend la variable f_1 et le processeur va la multiplier par. et efface ce qu'il y a dans la variable et le remplace par le résultat du dernier calcul soit 24

3/ Il créer une variable c_0 et il met dedans le résultat donné par le proc en faisant $24 - 20 = 4$.

4/ Le PC prend la variable c_0 et demande au processeur d'additionner 4 et 24 soit 28 et met ce résultat dans la variable c_0 en écrasant le dernier résultat.

5/ L'ordi va afficher ce qu'il y a dans la variable c_0

(1)

1). Reserve moi un espace dans la mémoire vide
· donne lui le nom « f 1 »
· envoie au processeur le calcul 3×4.
· stocke le résultat dans l'espace mémoire

2). Recupère le contenu de la variable « f 1 »
· envoie au processeur le calcul (contenu de la variable) × 2.
· stocke le résultat dans l'espace mémoire

3). Reserve moi un autre espace mémoire
· donne lui le nom « Co »
· Récupère le contenu de la variable « f 1 » c'est a dire 24.
· envoie au processeur le calcul (contenu de la variable) - 20.
· stake le résultat de ce calcul dans l'espace mémoire.

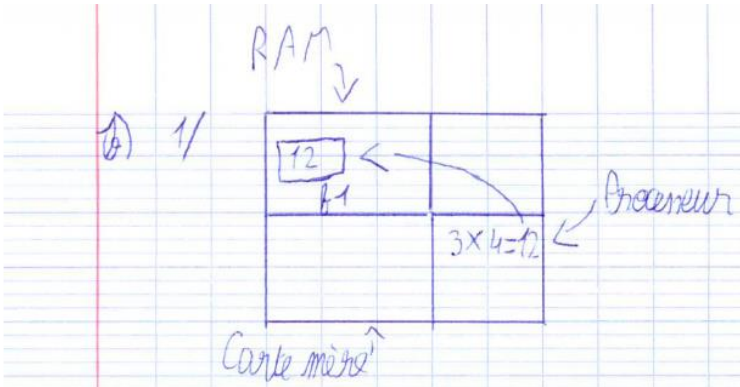
2)

	Contenue de la variable f1	Contenue de la variable co
Instruction 1	(3 x 4) = 12	∝
Instruction 2	((3 x 4) x 2) = 24	∝
Instruction 3	24	((3x4)x2) - 20 = 4
Instruction 4	24	((3x4)x2) - 20 + (3x4) = 28
Instruction 5	∝	28

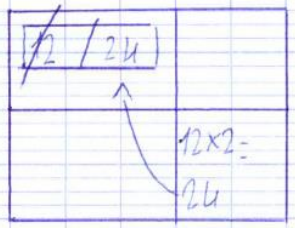
2.

	Contenue de la variable f1	Contenue de la variable co
1	3x4 = 12	x
2	12 x 2 = 24	x
3	24	24 - 20 = 4
4	24	4 + 24 = 28
5	24	28

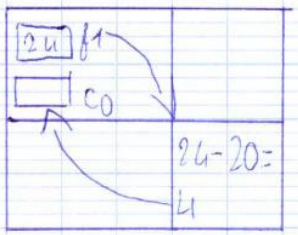
b) 1/



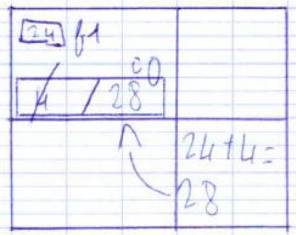
2/



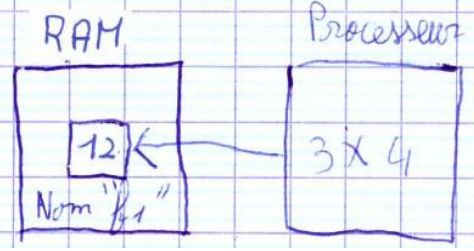
3/



4/



1) Reserve moi un espace mémoire pour que je puisse faire référence à cette espace, donne lui le nom "f1" et stocke le resultat de 3 x 4 sous forme binaire dans l'espace mémoire appelé "f1"



I. Compétences informatiques EVALUER

II. Présentation d'un exemple

- On peut aussi **évaluer** symboliquement. Dans ce cas on évalue *à la main* le résultat en nommant les valeurs des paramètres : `EvaluerPolynome([a2,a1,a0], x)` et en exécutant symboliquement chacune des affectations.

Point	p	x	y
Point 1	[a2,a1,a0]	x	0
Point 2	[a2,a1,a0]	x	a2
Point 2	[a2,a1,a0]	x	a2 * x + a1
Point 2	[a2,a1,a0]	x	a2 x**2 + a1 x + a0

Dans ce cas l'interprète Python ne peut vérifier le résultat, mais cela permet de raisonner sur le programme et d'obtenir une formulation qui correspond bien à ce que l'on souhaitait calculer.

I. Compétences informatiques EVALUER

II. Présentation d'un exemple

Python 3.6

```
1  def listSum(numbers):
2    if not numbers:
3      return 0
4    else:
5      (f, rest) = numbers
6      return f + listSum(rest)
7
8  myList = (1, (2, (3, None)))
9  total = listSum(myList)
```

Edit this code

line that just executed

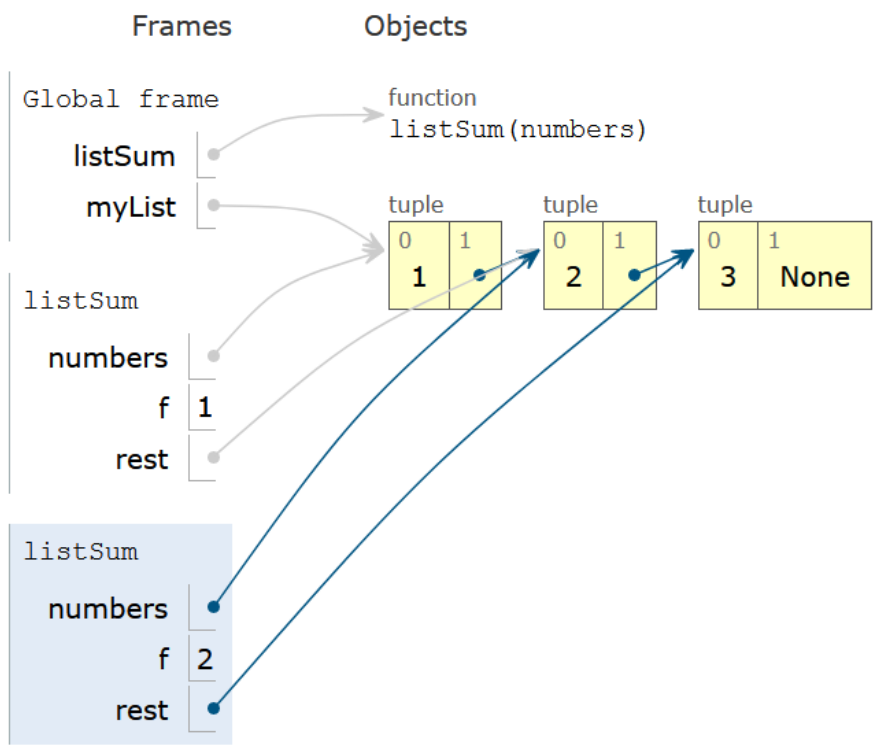
next line to execute

< Prev Next >

Step 11 of 22

Rendered by Python Tutor

Customize visualization (NEW!)



http://pythontutor.com/

Fichier Édition Affichage Historique Marque-pages Outils Aide

Accueil | ENT UB | Zimbra: Réception (13) | WeTransfer | Python Tutor - Visualize Python | pythontutor, vidéo 1, présentati

pythontutor.com/visualize.html#mode=edit

Débuter avec Firefox | Moodle Université de ... | Zimbra: Réception (9)

Get live help for free in the Python tutoring Discord chat room

Write code in Python 3.6

```
1  def somme(L):
2      s=0
3      for i in L:
4          s=s+i
5      return s
6
7  L=[1,3,7,4,9]
8
9  somme(L)
10
```

Visualize Execution | Live Programming Mode

hide exited frames [default] | inline primitives, don't nest objects [default]

draw pointers as arrows [default]

I. Compétences informatiques EVALUER

II. Présentation d'un exemple

Un exemple:

- Soit l'expression suivante prévue pour déterminer si une année est bissextile :

```
an % 4==0 and (not an % 100 == 0 or an
% 400 == 0)
```

On peut au préalable calculer le type du résultat de cette expression en fonction du type des arguments. On sait que `==` prend deux arguments de type quelconque et donne un résultat de type `bool`, que `or` et `and` donne des résultats de type `bool` donc l'expression complète donne un résultat de type `bool`.

On peut vérifier :

```
an = 2019
```

```
type(an % 4==0 and (not an % 100 == 0
or an % 400 == 0))
```

I. Compétences informatiques EVALUER

II. Présentation d'un exemple

Des erreurs fréquentes :

```
a=[input("premier entier"),input("deuxieme
entier"),input("troisieme entier")]
a.sort()
print(a)
```

I. Compétences informatiques EVALUER

II. Présentation d'un exemple

On peut **évaluer** le temps d'exécution d'un programme ou le nombre d'affectations ou de tests exécutés.

```
Tab = [2, 8, 6, 12, 4, 9]
def tri (t):
    for i in range (len(t)-1):
        for j in range(i+1,len(t)):
            if t[j]<t[i]:
                t[i],t[j] = t[j],t[i]

tri(Tab)
```

on peut évaluer à 15 le nombre de tests et à 5 le nombre d'affectations.
Cette évaluation peut se faire pour une donnée concrète: on peut alors instrumenter le programme pour lui faire faire cette évaluation en y ajoutant des compteurs.

I. Compétences informatiques EVALUER

II. Présentation d'un exemple

- De manière générale, **évaluer** un programme consiste donc à regarder ce programme comme une donnée et à en calculer une valeur par une méthode particulière. Ce changement de plan du programmeur consiste à regarder son programme tel qu'il est - et non tel qu'il aurait voulu qu'il soit.
- La compétence *évaluer* est fondamentale pour mettre au point un programme.

ANTICIPER

I. Compétences informatiques ANTICIPER

II. Présentation d'un exemple

- Capacité à se mettre dans la posture du programmeur qui doit décrire dans un algorithme l'enchaînement séquentiel/répétitif/conditionnel des instructions, avant même le début de son exécution.
- C'est une compétence fondamentale pour la conception d'algorithmes. C'est aussi un des principaux obstacles didactiques rencontrés par les programmeurs débutants.

« Une propriété difficile à intégrer [...] est le caractère différé d'une exécution du programme » J. Rogalski, 1986

I. Compétences informatiques ANTICIPER

II. Présentation d'un exemple

Anticiper c'est imaginer

- C'est la part créative du travail du programmeur d'imaginer par quel chemin de calculs intermédiaires on peut passer, pour aller des informations disponibles aux informations que l'on souhaite calculer.
- Anticiper les étapes successives du traitement et les différents cas à envisager, c'est tout l'*art de programmer*.

The Art of Computer Programming, Donald Knuth.

=> Importance du travail papier-crayon

I. Compétences informatiques ANTICIPER

II. Présentation d'un exemple

Dans le contexte de l'enseignement de spécialité en classe de 1ere

- Même si cette compétence est abordée très tôt dans l'apprentissage de la programmation, et si l'obstacle associé est dépassé très vite, toutes les situations problèmes requérant une solution algorithmique font appel à la maîtrise de cette compétence.
- La difficulté d'une situation vient de la longueur de la chaine d'anticipation nécessaire et de la nécessité d'inventer des états intermédiaires pour anticiper le passage d'un état initial à un état final.
- Cette compétence est liée à la compétence **décomposer** car pour anticiper la résolution d'une situation complexe, il est souvent utile de profiter d'états intermédiaires identifiés pour séparer le problème.
- Le modèle d'exécution sous jacent et sa compréhension par l'élève ont une influence sur la difficulté à anticiper correctement.

I. Compétences informatiques ANTICIPER

II. Présentation d’un exemple

Dans le contexte lié à l'utilisation d'un langage de programmation

- L'ordre d'évaluation des expressions et d'exécution des instructions sont des caractéristiques à prendre en compte au moment d'anticiper l'écriture d'un programme.
- La compétence d'anticipation n'est donc pas seulement une capacité d'analyse *descendante* allant du problème vers sa solution algorithmique. Ce peut aussi être une démarche *ascendante* par composition d'instructions en anticipant leur effet pour aller des informations disponibles vers les informations à calculer.

I. Compétences informatiques ANTICIPER

II. Présentation d'un exemple

Écrire un programme permettant d'afficher le plus petit de trois nombres entrés au clavier.
Voici les programmes produits par quatre élèves :

a=int(input("a=?"))
b=int(input("b=?"))
c=int(input("c=?"))
if(a<b):
  if(a<c):
  min=a
  else:
  min=c
elif(b<c):
  min=b
else:
  min=c
print("Le plus petit de ces trois entiers est",min)
	>>> def pluspetit(x,y,z):
	  if x<y and x<z:
	  print (x, "est le plus petit des trois nombres")
	  elif y<x and y<z:
	  print (y, "est le plus petit des trois nombres")
	  else:
	  print (z, "est le plus petit des trois nombres")

def pluspetit1(x,y,z):
  n=x
  if y<=n:
  n=y
  if z<=n:
  n=z
  return n
	a=[input("premier entier"),input("deuxieme entier"),input("troisieme entier")]
	a.sort()
	print(a)

I. Compétences informatiques ANTICIPER

II. Présentation d'un exemple

Exercice 3 : Le programme fait jouer l'utilisateur jusqu'à ce qu'il coule le bateau, dans la limite de 7 coups.

1) Décrire **en français** les différentes étapes du programme « jusqu'à ce qu'il coule le bateau et dans la limite de 7 coups ».

2) **Choix du type de boucle**

En algorithmique, il existe deux types de boucles : les boucles `POUR`, lorsque l'on souhaite répéter une série de commandes un certain nombre déterminé de fois, et les boucles `TANT QUE`, lorsque l'on souhaite répéter une série de commandes *tant que* une certaine condition est vérifiée ⇝ voir le poly de programmation.

a) Dans une partie, savez-vous *a priori* combien de fois on passe dans la boucle qui fait jouer l'utilisateur ? En déduire le type de boucle à utiliser.

b) Traduire la condition « jusqu'à ce qu'il coule le bateau » en une condition du type « FOR » ou « TANT QUE », conformément à la réponse à la question précédente. Exprimer cette condition en fonction des variables du programme (par exemple, les coordonnées `x` et `y` choisies).

c) Traduire la condition « jusqu'à ce qu'il coule le bateau **et** tant qu'il reste des munitions » en une condition du type « FOR » ou « TANT QUE », conformément à la question a). Penser également à les exprimer en fonction des données du programme.

d) *En Python : Implémenter la boucle pour que la partie dure jusqu'à ce que le bateau coule et dans la limite de 7 coups.*

Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek

DECOMPOSER

I. Compétences informatiques DECOMPOSER

II. Présentation d'un exemple

Capacité à transformer un problème complexe en un ensemble de problèmes plus simples équivalents au problème initial.

Cette compétence est fondamentale car c'est elle qui permet d'envisager le traitement de problèmes arbitrairement complexes par réduction à des problèmes plus simples ou déjà connus.

I. Compétences informatiques DECOMPOSER

II. Présentation d'un exemple

Décomposition par cas

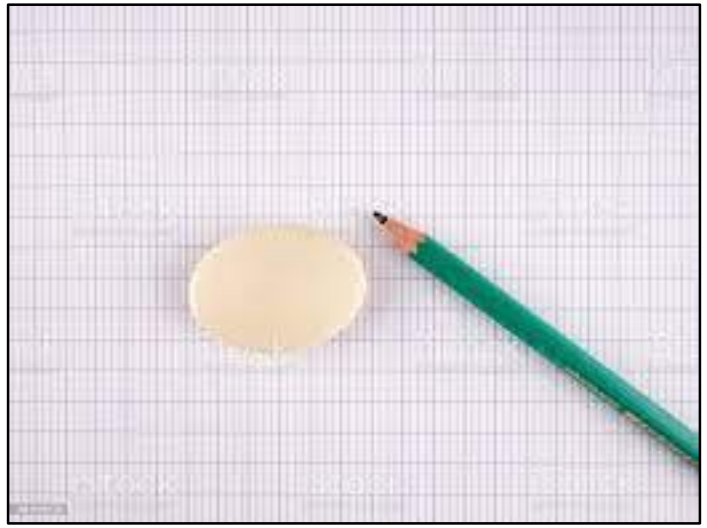
- **Exemple** : le calcul du nombre de jours d'un mois peut se décomposer en 3 cas distincs pour traiter les mois de 31 jours, les mois de 30 jours et le cas particulier du mois de février.
- **Exemple** : le problème de la résolution d'une équation du second degré peut se décomposer en 3 sous-problèmes distincts selon la valeur du discriminant négatif, nul ou strictement positif.
- **Exemple** : le problème de la compression d'une image peut être séparé en plusieurs sous-problèmes selon le format d'enregistrement de l'image à compresser et selon le souhait de l'utilisateur de compresser avec ou sans perte d'information.
- On peut dire en analysant ces exemples que la décomposition peut se pratiquer à tous les niveaux de la conception d'un programme : du niveau le plus fin pour séparer quelques instructions au niveau le plus global pour séparer des traitements demandant chacun plusieurs centaines ou milliers de lignes de code.

I. Compétences informatiques DECOMPOSER

II. Présentation d'un exemple

Des outils :

Aides à la représentation .....



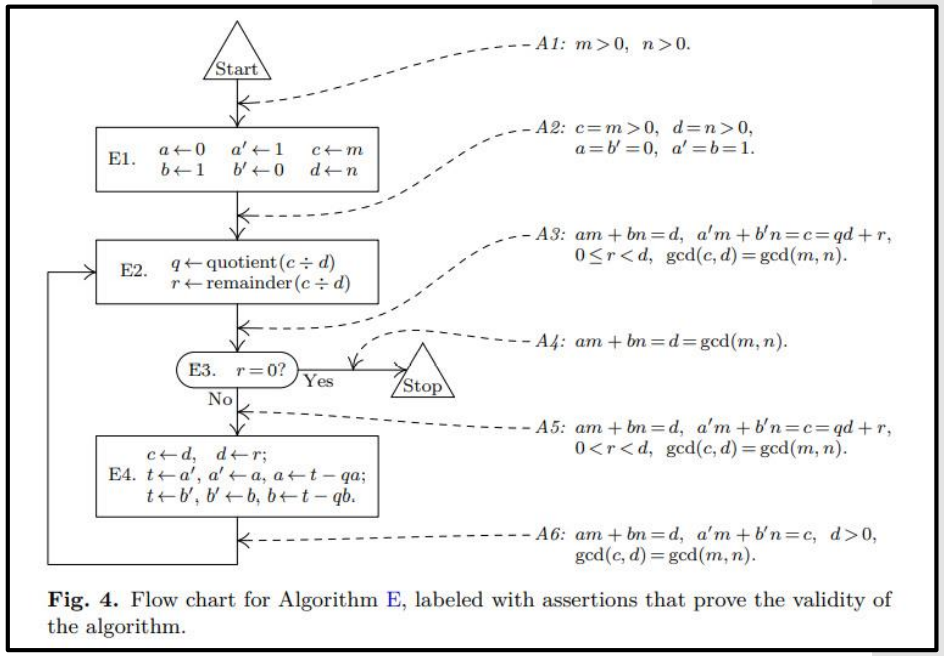
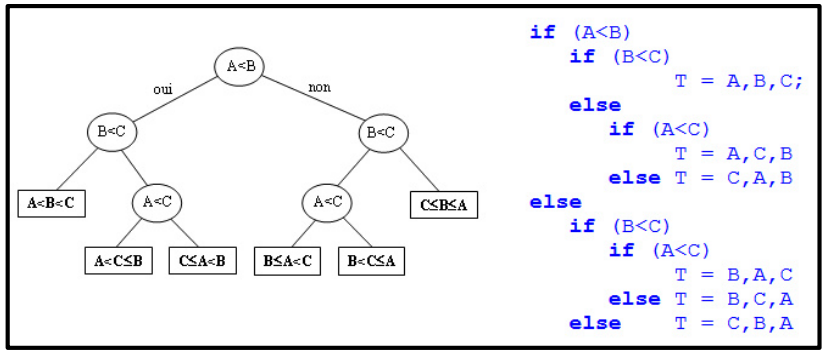
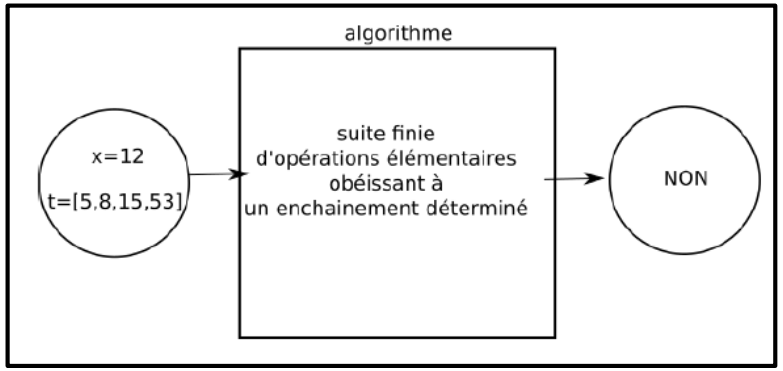


Fig. 4. Flow chart for Algorithm E, labeled with assertions that prove the validity of the algorithm.




```
if (A<B)
   if (B<C)
         T = A,B,C;
   else
      if (A<C)
         T = A,C,B
      else T = C,A,B
else
   if (B<C)
      if (A<C)
         T = B,A,C
      else T = B,C,A
   else    T = C,B,A
```



I. Compétences informatiques DECOMPOSER

II. Présentation d'un exemple

Décomposition séquentielle : importance des fonctions

- **Exemple** : programmer la soustraction de deux nombres en utilisant le codage en complément à deux, peut se décomposer en 6 étapes consistant à coder chacun des nombres a et b, puis à calculer le code de -b en inversant les bits et en ajoutant 1 au code de b, à additionner les codes obtenus puis à décoder le résultat.

```
def soustraction(a,b):
    codea = coder(a)
    codeb = coder(b)
    codeinvb = inverserbits(codeb)
    codemoinsb = ajouter1 (codeinvb)
    codeamoinsb = additionnercodes(codea,
codemoinsb)
    amoinsb = decoder(codeamoinsb)
    return(amoinsb)
```

- Cette mise en œuvre conserve la décomposition suivie. Des variables intermédiaires ont été imaginées et nommées pour contenir les résultats intermédiaires de la décomposition choisie.
- La solution ne sera complète et exécutable que lorsque toutes les fonctions intermédiaires utilisées auront bien été définies.

I. Compétences informatiques DECOMPOSER

II. Retour sur les deux TP

Mise en oeuvre

- Le résultat d'une démarche de conception utilisant la décomposition d'un problème en problèmes plus simples peut être montré explicitement en utilisant une notion de sous-programme / fonction / procédure pour conserver dans le programme écrit la démarche de décomposition suivie.
- La démarche peut aussi être cachée en recollant ensemble dans un seul et même traitement toutes les parties élémentaires issues de la décomposition. Ceci est le plus souvent déconseillé car cela donne des fragments de programmes plus complexes et donc moins simples à comprendre.

I. Compétences informatiques DECOMPOSER

II. Retour sur les deux TP

Conclusion

Savoir décomposer est une compétence permettant d'appréhender des problèmes complexes. Sa mise en œuvre passe par l'utilisation systématique de primitives de programmation permettant la structuration des programmes - fonctions et procédures mais aussi objets et méthodes. Ces mécanismes permettent aussi, par ailleurs, de mettre en œuvre des démarches de généralisation et d'abstraction.

GENERALISER

I. Compétences informatiques GENERALISER

II. Présentation d'un exemple

Capacité à inférer un problème général à partir d'une instance de ce problème, et à repérer dans un problème particulier la répétition de traitements ou de données suivant un même schéma.

C'est la notion de fonction **avec paramètres** qui permet de mettre une œuvre un programme qui peut résoudre un problème général, dont les instances particulières correspondent aux application de cette fonction avec des valeurs particulières des paramètres.

I. Compétences informatiques GENERALISER

II. Présentation d'un exemple

Exemple: On veut calculer la durée de voyage d'un voyageur ayant pris successivement un long courrier pendant exactement 18h20mn30s suivi d'un vol moyen courrier de durée 6h45mn50s. Ecrire un programme Python permettant de résoudre ce problème.

```
h1 = 18
mn1 = 20
s1 = 30
total1 = 18 * 3600 + 20 * 60 + 30
h2 = 6
mn2 = 45
s2 = 50
total2 = 6 * 3600 + 45 * 60 + 50
total = total1 + total2
mn = total // 60
s = total - mn * 60
h = mn // 60
mn = mn - h * 60
print(h, "h", mn , "mn", s , "s")
```

```
h1 = 18
mn1 = 20
s1 = 30
total1 = 18 * 3600 + 20 * 60 + 30
h2 = 6
mn2 = 45
s2 = 50
total2 = 6 * 3600 + 45 * 60 + 50
total = total1 + total2
mn = total // 60
s = total - mn * 60
h = mn // 60
mn = mn - h * 60
print(h, "h", mn , "mn", s , "s")
```

```
def temps(h, mn, s):
    return (h * 3600 + mn * 60 + s)

total1 = temps(h1, mn1, s1)
total2 = temps(h2, mn2, s2)
```

```
def additionner_et_afficher(h1, mn1,
s1, h2, mn2, s2):
    total1 = temps(h1, mn1, s1)
    total2 = temps(h2, mn2, s2)
    total = total1 + total2
    mn = total // 60
    s = total % 60
    h = mn // 60
    mn = mn % 60
    print(h, "h",mn , "mn",s , "s")

additionner_et_afficher(18, 20, 30, 6,
45, 50)
```

I. Compétences informatiques GENERALISER

II. Retour sur les deux TP

Généralisation par une fonction en paramètre

Dans le cas où l'on repère, entre deux traitements, un même schéma d'algorithme mais cependant quelques différences dans les calculs effectués on peut utiliser la notion de fonction en paramètre.

```
def somme(liste):
    resultat = 0
    for elt in liste:
        resultat = resultat + elt
    return(resultat)

somme([1, 2, 3, 4, 5])
```

```
def produit (liste):
    resultat = 1
    for elt in liste:
        resultat = resultat * elt
    return(resultat)

produit([1, 2, 3, 4, 5])
```

On peut **généraliser** ces deux fonctions en définissant une fonction accumuler qui reçoit comme paramètre une « fonction » opération supposée recevoir deux paramètres.

```
def somme(liste):
    resultat = 0
    for elt in liste:
        resultat = resultat + elt
    return(resultat)

somme([1, 2, 3, 4, 5])
```

```
def produit (liste):
    resultat = 1
    for elt in liste:
        resultat = resultat * elt
    return(resultat)

produit([1, 2, 3, 4, 5])
```

```
def accumuler(operation, neutre, liste):
    resultat = neutre
    for elt in liste:
        resultat =operation(resultat, elt)
    return(resultat)
```

```
def addition(x,y):
    return(x + y)

accumuler(addition, 0, [1,2,3,4,5])
```

```
def multiplication(x,y):
    return(x * y)

accumuler(multiplication, 1,
[1,2,3,4,5])
```

I. Compétences informatiques GENERALISER

II. Présentation d'un exemple

```
def accumuler(operation, neutre, liste):
    resultat = neutre
    for elt in liste:
        resultat =operation(resultat, elt)
    return(resultat)
```

```
def addition(x,y):
    return(x + y)

accumuler(addition, 0, [1,2,3,4,5])
```

```
def multiplication(x,y):
    return(x * y)

accumuler(multiplication, 1,
[1,2,3,4,5])
```

On a ainsi, à partir du même schéma, décliné plusieurs instances d'une solution à un problème plus général

I. Compétences informatiques GENERALISER

II. Présentation d'un exemple

Conclusion

- Généraliser est une compétence de haut niveau qui permet au programmeur de résoudre des problèmes plus généraux et ensuite de réutiliser pour des instances particulières des parties de programme déjà écrites.
- La notion de paramètre, utilisée pour instancier des valeurs particulières ou des fonctions particulières est le mécanisme permettant de mettre en oeuvre la généralisation.

ABSTRAIRE

I. Compétences informatiques ABSTRAIRE

II. Présentation d'un exemple

- Capacité à "faire abstraction" des informations non pertinentes et à créer des solutions où la manière dont un problème est résolu peut être "abstraite" à l'aide d'une interface pertinente.

I. Compétences informatiques ABSTRAIRE

II. Présentation d'un exemple

Abstraire avec les données

- Abstraire avec les données consiste à *encapsuler* un certain nombre d'informations en utilisant une structure de données composée qui peut éventuellement masquer le détail du codage des informations.
- **Exemple** : Le programme suivant peut effectuer un calcul de temps si les fonctions intermédiaires utilisées sont définies pour enregistrer les informations sous un format adapté et y accéder. La représentation choisie n'est pas visible à ce niveau. Elle est abstraite. Les informations fournies consistent en des nombres d'heures, minutes et secondes.

```
t1 = temps(18, 20, 30)
t2 = temps(6, 45, 50)
total = additionner_temps(t1, t2)
    afficher_temps (total)
```

I. Compétences informatiques ABSTRAIRE

II. Présentation d'un exemple

```
def temps(h,mn,s):
    return({'h':h, 'mn':mn,'s':s})

def additionner_temps(t1,t2):
    s = t1['s']+t2['s']
    mn = t1['mn']+t2['mn']
    h = t1['h']+t2['h']
    return({'h':h + (mn+s//60)//60,
'mn':(mn+s//60)%60,'s':s%60})

def afficher_temps(t):
    print(t['h'], "h", t['mn'], "mn", t['s'], "s")

t1 = temps(18, 20, 30)
t2 = temps(6, 45, 50)
total = additionner_temps(t1, t2)
afficher_temps (total)
25 h 6 mn 20 s
```

Une autre mise en œuvre peut choisir de tout convertir en secondes.

I. Compétences informatiques ABSTRAIRE

II. Présentation d'un exemple

```
def temps(h,mn,s):
    return((h*60 + mn)*60 + s)

def additionner_temps(t1,t2):
    return(t1 + t2)

def afficher_temps(t):
    print(t // 3600, "h",(t//60)%60 , "mn", t%60, "s")

t1 = temps(18, 20, 30)
t2 = temps(6, 45, 50)
total = additionner_temps(t1, t2)
afficher_temps (total)
25 h 6 mn 20 s
```

Pour l'utilisateur de cet ensemble de fonctions, le choix de la structure de données n'est pas nécessairement visible. Le résultat final est le même dans les deux mises en oeuvre.

I. Compétences informatiques ABSTRAIRE

II. Présentation d'un exemple

Abstraire avec les fonctions

- Les fonctions sont de manière générale un outil d'abstraction qui peut aussi masquer la méthode de calcul utilisée, y compris quand il n'y a pas de question de choix de structure de données.
- **Exemple** : les deux fonctions fact ci-dessous, calculent bien chacune la factorielle d'un entier positif. Elles sont interchangeables pour l'utilisateur.

```
def fact(n):
    f = 1
    for i in range(n):
        f = f * (i+1)
    return(f)

fact(10)
3628800
```

```
def fact(n):
    return(1 if n==0 else n*
fact(n-1))

fact(10)
3628800
```

I. Compétences informatiques ABSTRAIRE

II. Présentation d'un exemple

Conclusion

- Abstraire en cachant soit les données soit les algorithmes, par des fonctions, permet au programmeur de simplifier l'écriture de ses programmes.
- La combinaison des deux méthodes aboutit à la programmation orientée objet où données et méthodes sont encapsulées à l'intérieur d'une classe.

I. Compétences informatiques

II. Présentation d'un exemple

Retour sur des extraits des TP A et B

Exercice 4

Ecrire un programme qui demande à l'utilisateur d'entrer les coordonnées de deux points A et B dans un repère orthonormé puis qui calcule la distance AB.

Exercice 5

a) Ecrire un programme qui demande à l'utilisateur d'entrer 3 nombres puis qui les écrit dans l'ordre croissant (tri croissant).

b) Ecrire un programme qui demande à l'utilisateur d'entrer 3 nombres puis qui les écrit dans l'ordre décroissant (tri décroissant).

Extrait TP B

Concevoir un programme qui compare trois nombres (indication : On pourra appeler plusieurs fois la fonction max…)

I. Quelques travaux d'élèves

II. Présentation d'un exemple

TP : La bataille navale

Exercice 1

On considère le programme (rudimentaire, à ce stade) de bataille navale.

```
a=4
b=7
print('A vous de jouer')
x=int(input('Donner la coordonnee x :'))
y=int(input('Donner la coordonnee y :'))
if x==a and y==b:
    print('Coulé')
elif x==a or y==b:
    print('En vue')
else:
    print('A leau')
```

1) Décrire **en français**, étape par étape, le comportement de ce programme.
2) Proposer un jeu de test satisfaisant pour tester la conformité de ce programme avec un embryon de bataille navale.
3) *En TP : Implémenter ce script et le tester conformément à la question 2).*

Le but de ce TP est d'améliorer petit à petit ce programme pour qu'il remplisse des conditions plus proches d'une réelle partie de bataille navale.

Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek

I. Quelques travaux d'élèves

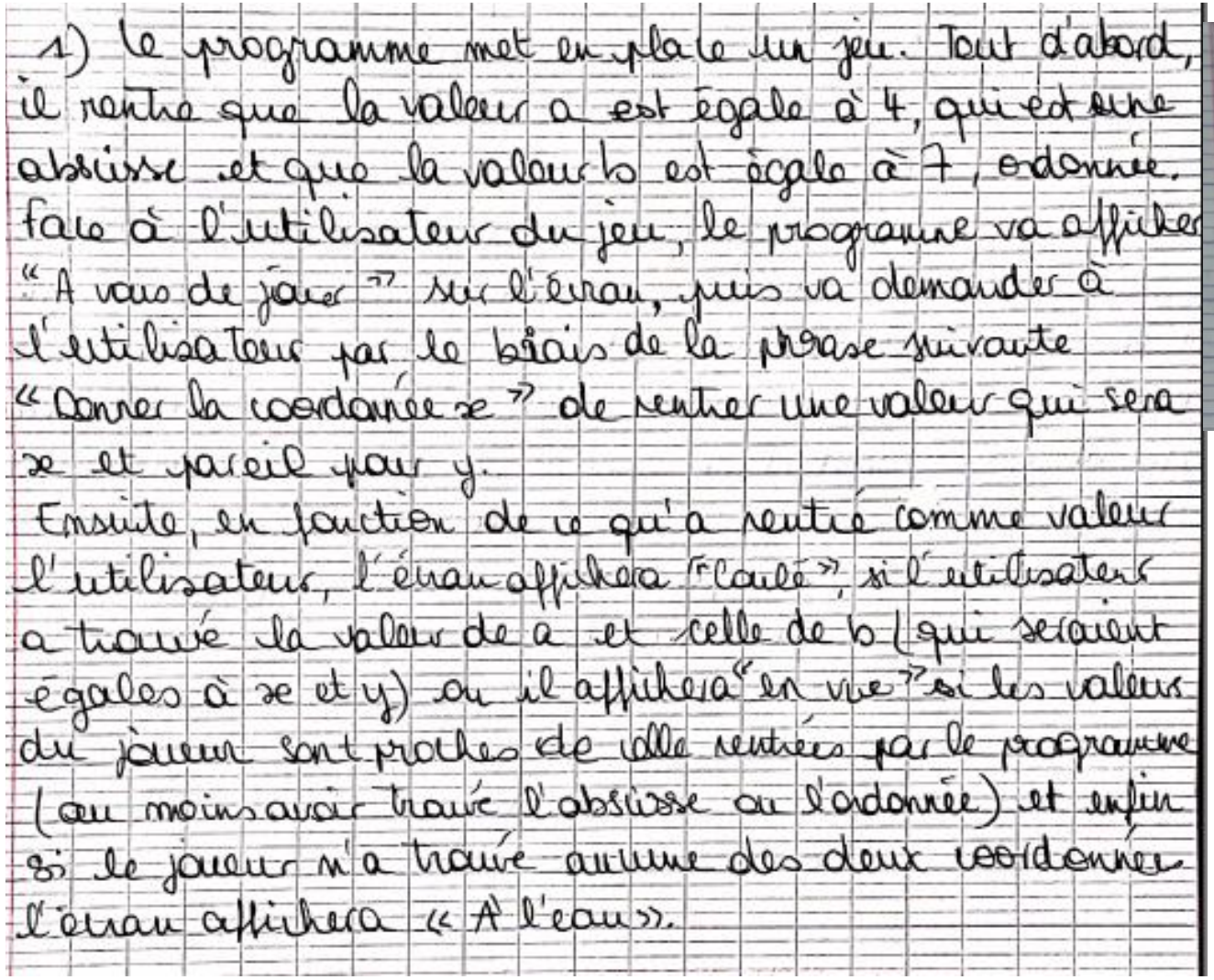
II. Erreur et obstacle, apports didactiques

Coup de pouce 1 : Exercice 1

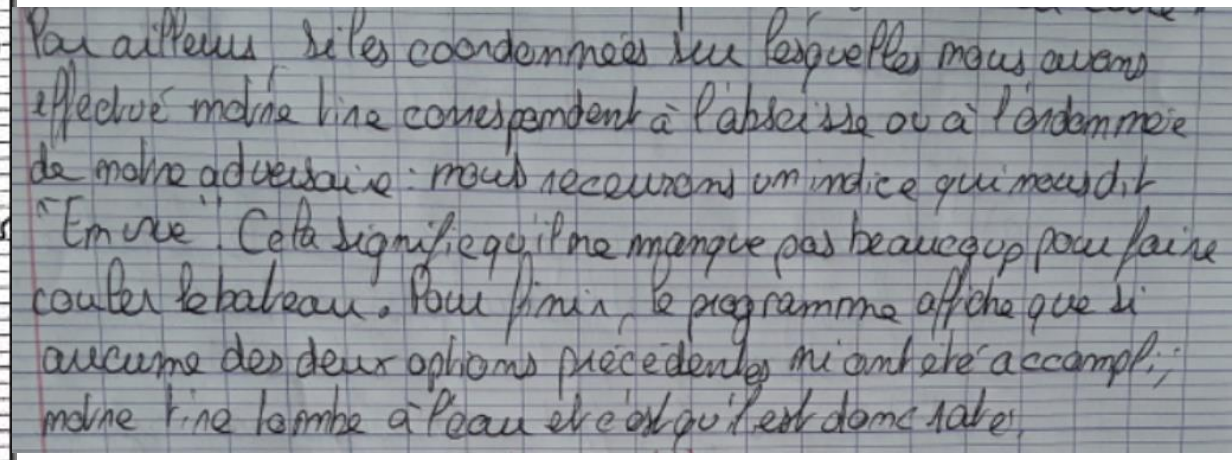
La position du bateau est donnée par $a = 4$ et $b = 7$. Compléter le tableau suivant.

x	y	Message affiché
2	4	
4	5	
2	7	
4	7	

Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek



1) Le programme met en place un jeu. Tout d'abord, il rentre que la valeur a est égale à 4, qui est une abscisse et que la valeur b est égale à 7, ordonnée. Face à l'utilisateur du jeu, le programme va afficher « A vous de jouer » sur l'écran, puis va demander à l'utilisateur par le biais de la phrase suivante « Donner la coordonnée x » de rentrer une valeur qui sera x et pareil pour y.
Ensuite, en fonction de ce qu'a rentré comme valeur l'utilisateur, l'écran affichera "Coulé", si l'utilisateur a trouvé la valeur de a et celle de b (qui seraient égales à x et y) ou il affichera "en vue" si les valeurs du joueur sont proches de celle rentrées par le programme (au moins avoir trouvé l'abscisse ou l'ordonnée) et enfin si le joueur n'a trouvé aucune des deux coordonnées l'écran affichera « A l'eau ».

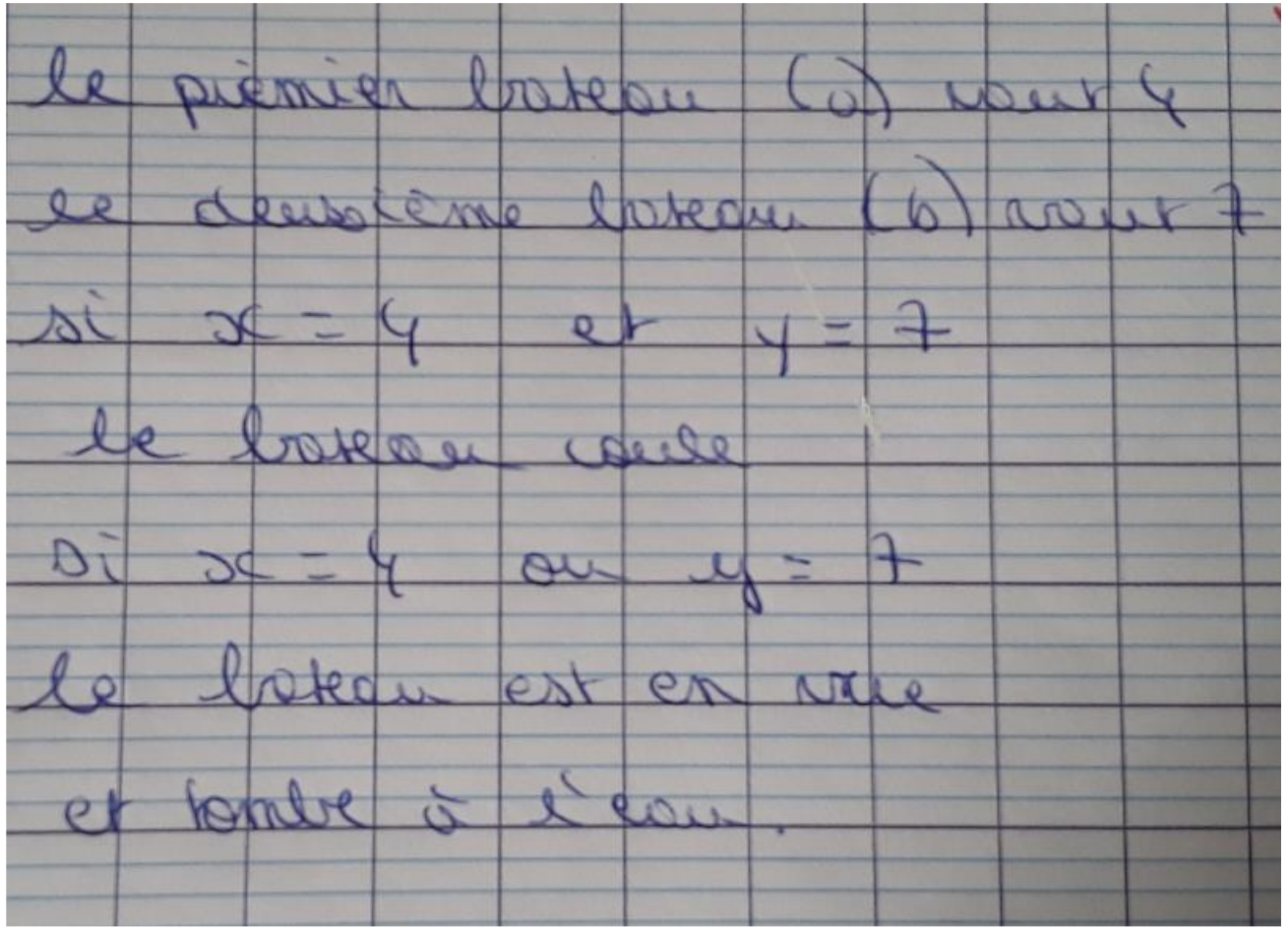


Par ailleurs si les coordonnées sur lesquelles nous avons effectué notre tir correspondent à l'abscisse ou à l'ordonnée de notre adversaire : nous recevrons un indice qui nous dit "En vue". Cela signifie qu'il ne manque pas beaucoup pour faire couler le bateau. Pour finir, le programme affiche que si aucune des deux options précédentes ne s'est réalisé; notre tir a tombé à l'eau et c'est qu'il est donc raté.

```
a=4
b=7
print('A vous de jouer')
x=int(input('Donner la coordonnee x :'))
y=int(input('Donner la coordonnee y :'))
if x==a and y==b:
    print('Coulé')
elif x==a or y==b:
    print('En vue')
else:
    print('A leau')
```

Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek

Décrire en français, étape par étape, le comportement de ce programme.



Décrire en français, étape par étape, le comportement de ce programme.

```
a=4
b=7
print('A vous de jouer')
x=int(input('Donner la coordonnee x :'))
y=int(input('Donner la coordonnee y :'))
if x==a and y==b:
    print('Coulé')
elif x==a or y==b:
    print('En vue')
else:
    print('A leau')
```

Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek

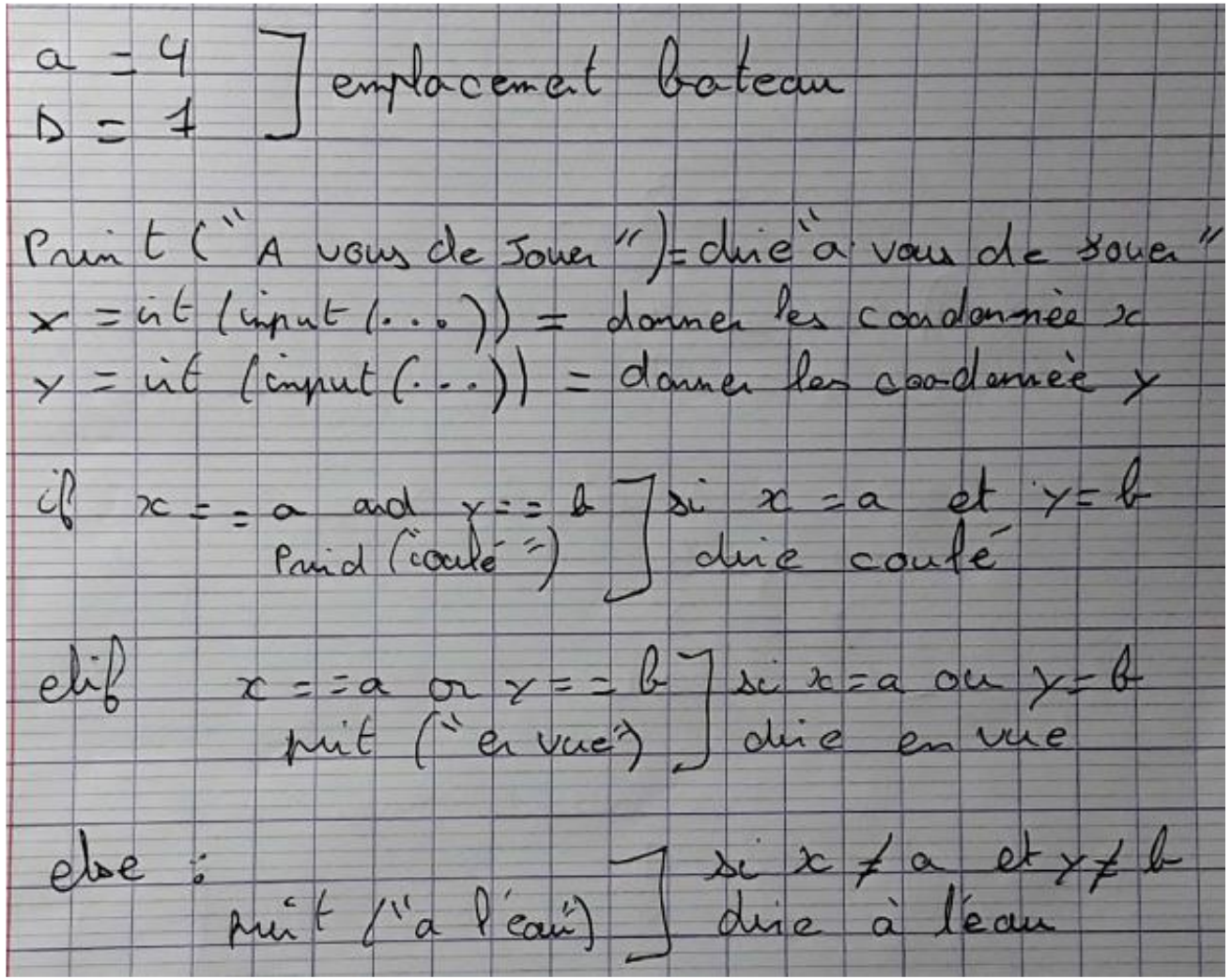
2) Imaginons le joueur joue à son écran toutes les coordonnées $x = 6$ et $y = 10$, alors comme $x \neq a$ et $y \neq b$, l'écran affichera « À l'eau ».
Ensuite dans un autre « round », le joueur rentre $x = 4$ et $y = 9$, comme $x = a$ mais $y \neq b$ alors l'écran affichera, « En vue ».
Enfin, lorsque le joueur rentrera d'autres valeurs telle que $x = 4$ et $y = 7$, comme $x = a$ et $y = b$, alors l'écran affichera « Coulé ».
Le bien, à tester ☺

2) On peut tester les sorties du programme en choisissant 4 coordonnés différentes. Admettons que l'on choisisse $x = 3$ et $y = 3$. A ce moment là le programme affichera "à l'eau". Maintenant avec $x = 4$ et $y = 10$ ou bien $x = 10$ et $y = 7$, le programme affichera "en vue". Pour finir avec $x = 4$ et $y = 7$ le programme affichera "coulé".

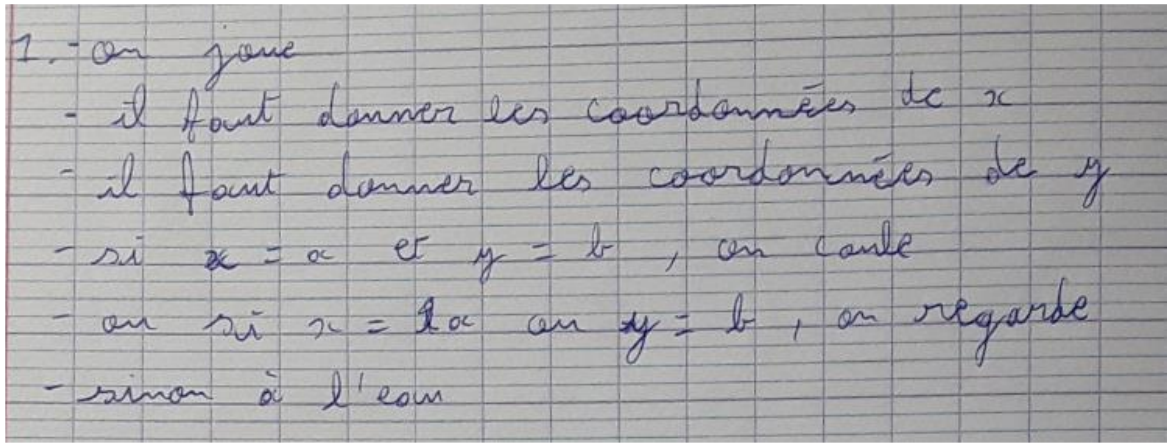
Proposer un jeu de tests satisfaisant pour tester la conformité de ce programme.

```
a=4
b=7
print('A vous de jouer')
x=int(input('Donner la coordonnee x :'))
y=int(input('Donner la coordonnee y :'))
if x==a and y==b:
    print('Coulé')
elif x==a or y==b:
    print('En vue')
else:
    print('A leau')
```

Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek

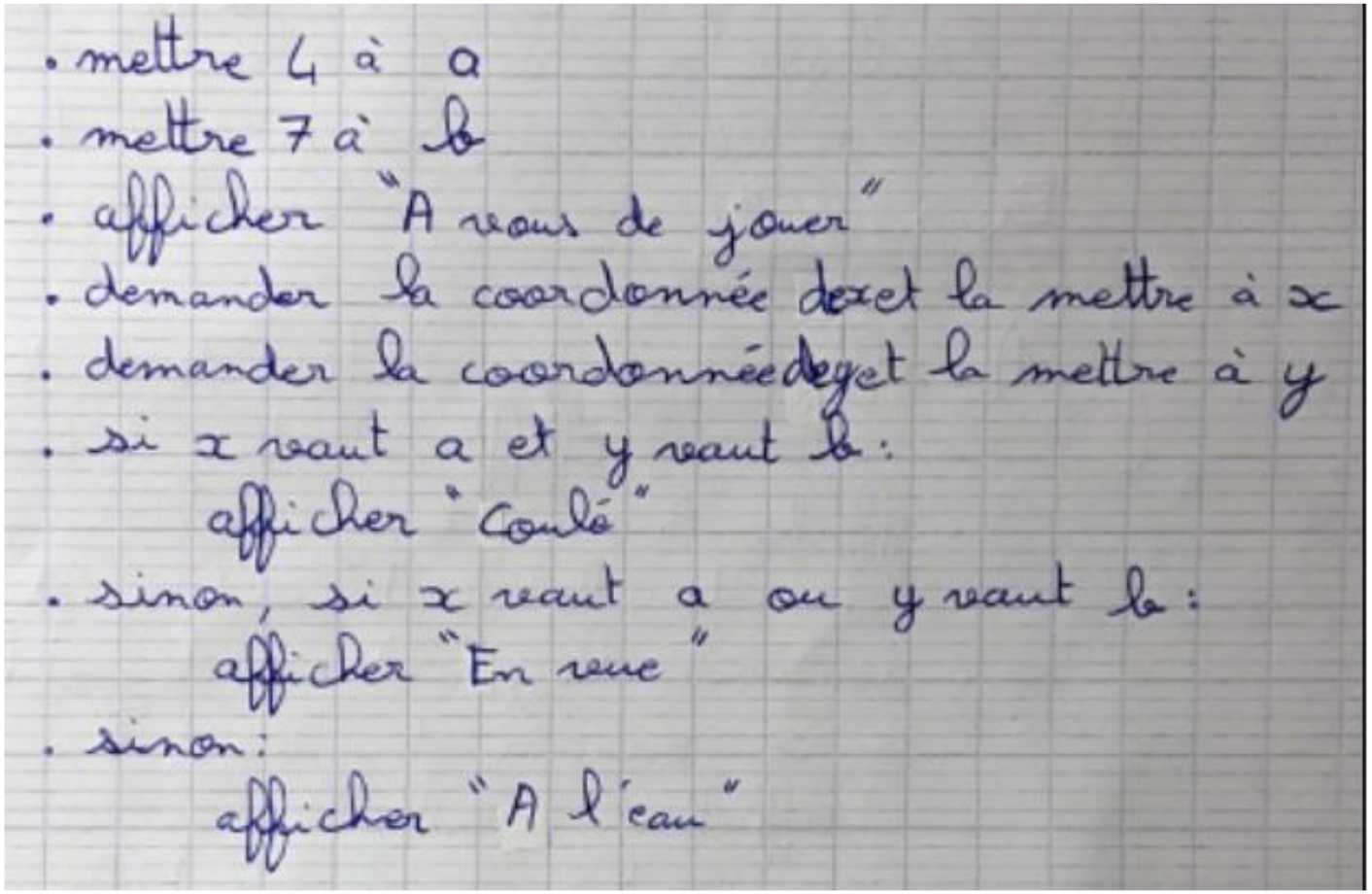


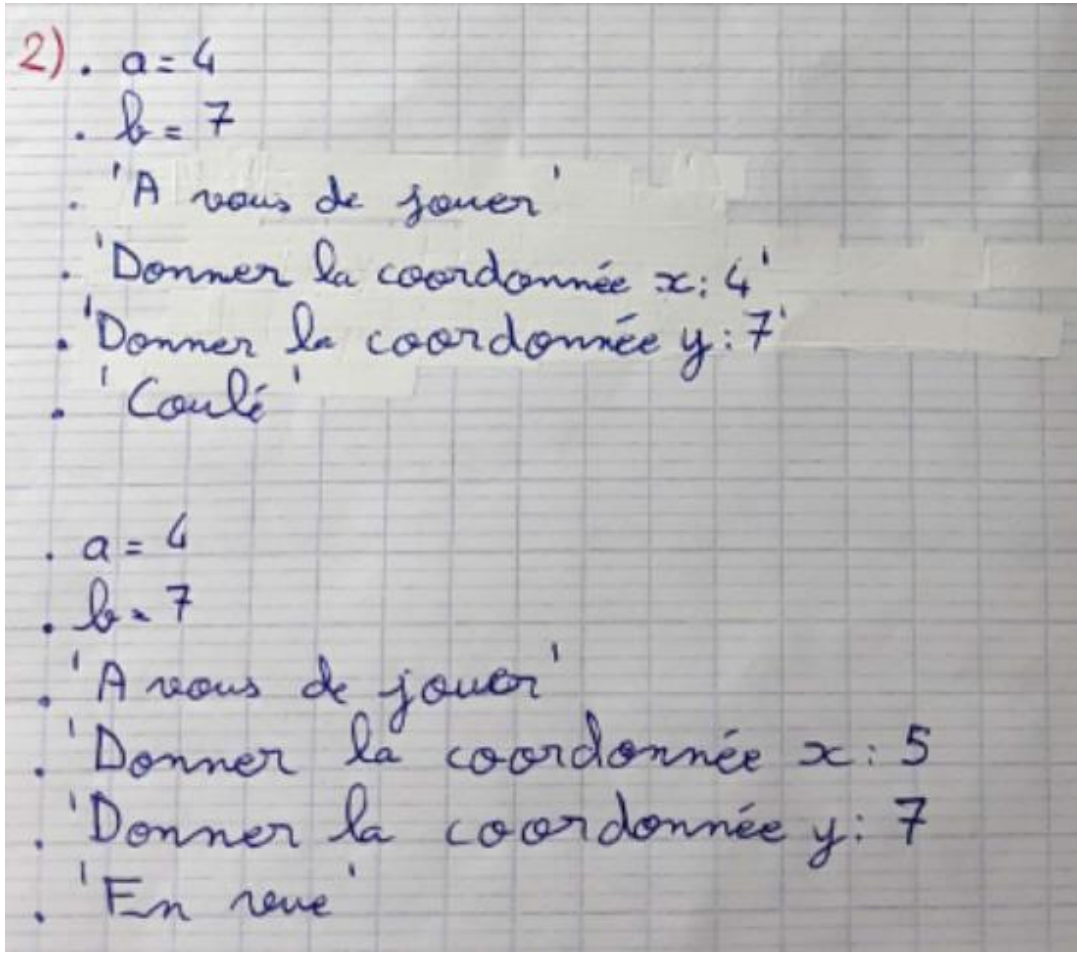
Proposer un jeu de tests satisfaisant pour tester la conformité de ce programme.




```
a=4
b=7
print('A vous de jouer')
x=int(input('Donner la coordonnee x :'))
y=int(input('Donner la coordonnee y :'))
if x==a and y==b:
    print('Coulé')
elif x==a or y==b:
    print('En vue')
else:
    print('A leau')
```

Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek





Proposer un jeu de tests satisfaisant pour tester la conformité de ce programme.

Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek

1. le programme correspond à une partie de bataille navale, la valeur a est de 4 et b est de 7.
le programme affiche "A vous de jouer", ensuite il va demander au joueur de saisir une valeur pour "x" et pour "y" par "donner la coordonnée x/y :".
Si "x" a comme valeur "a" et "y" a comme valeur "b", il sera affiché "coulé", si le joueur trouve les valeurs de "a" et de "b" ou approximativement, le programme affichera "En vue". Mais si le joueur n'a trouvé aucune valeur le programme affichera "A l'eau".

2. Prenons pour valeurs : $x = 1$ et $y = 6$ avec $a = x$ et $b = y$, l'écran affichera "Coulé"
Si $x = 1$ et $y = 6$ avec $a = x$ mais $b \neq y$, l'écran affichera "En vue".
Et si $x = 1$ et $y = 6$ avec $a \neq x$ et $b \neq y$, l'écran affichera alors "A l'eau"

Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek

I. Quelques travaux d'élèves

II. Présentation d'un exemple

Exercice 2 : Choix aléatoire de la position du bateau dans un carré de 10 sur 10.

1) Proposer, **en français**, comment modifier le programme pour que la position initiale du bateau soit choisie aléatoirement dans un carré de 10 sur 10.
2) *En Python : utiliser la librairie Python* `random` *à l'aide de la commande* `from random import *`, *et la commande* `randint(valeur_min,valeur_max)` *afin de générer des valeurs entières aléatoires comprises entre 1 et 10 pour déterminer la position du bateau.*

Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek

I. Quelques travaux d'élèves

II. Présentation d'un exemple

Exercice 3 : Le programme fait jouer l'utilisateur jusqu'à ce qu'il coule le bateau, dans la limite de 7 coups.

1) Décrire **en français** les différentes étapes du programme « jusqu'à ce qu'il coule le bateau et dans la limite de 7 coups ».

2) **Choix du type de boucle**
En algorithmique, il existe deux types de boucles : les boucles `POUR`, lorsque l'on souhaite répéter une série de commandes un certain nombre déterminé de fois, et les boucles `TANT QUE`, lorsque l'on souhaite répéter une série de commandes *tant que* une certaine condition est vérifiée ⇝ voir le poly de programmation.

   a) Dans une partie, savez-vous *a priori* combien de fois on passe dans la boucle qui fait jouer l'utilisateur ? En déduire le type de boucle à utiliser.

   b) Traduire la condition « jusqu'à ce qu'il coule le bateau » en une condition du type « FOR » ou « TANT QUE », conformément à la réponse à la question précédente. Exprimer cette condition en fonction des variables du programme (par exemple, les coordonnées `x` et `y` choisies).

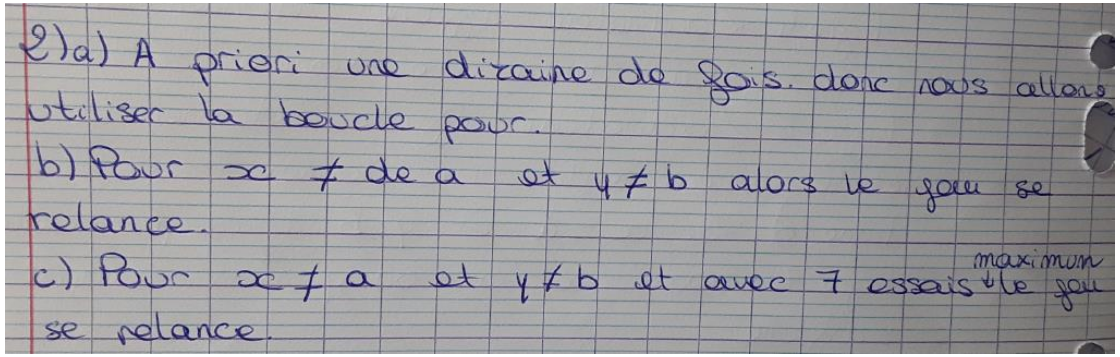
   c) Traduire la condition « jusqu'à ce qu'il coule le bateau **et** tant qu'il reste des munitions » en une condition du type « FOR » ou « TANT QUE », conformément à la question a). Penser également à les exprimer en fonction des données du programme.

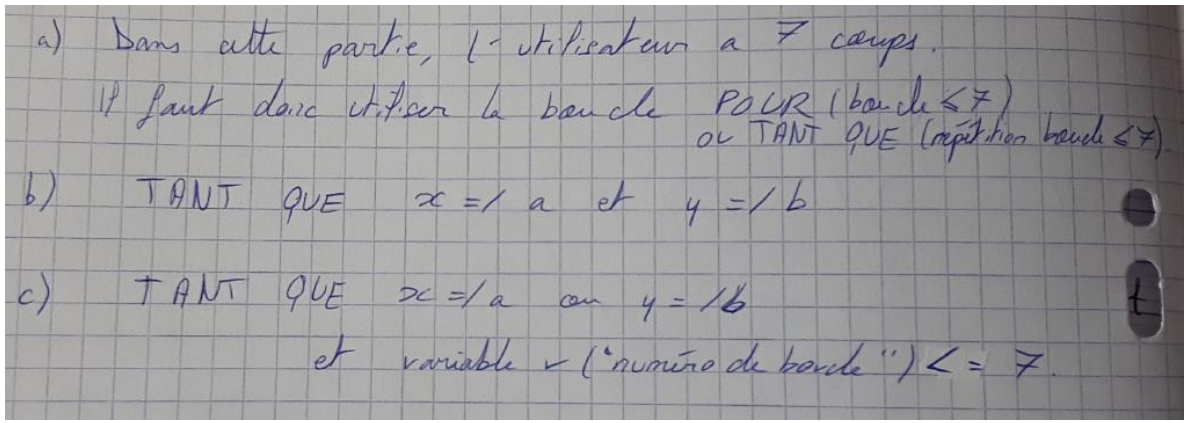
   d) *En Python : Implémenter la boucle pour que la partie dure jusqu'à ce que le bateau coule et dans la limite de 7 coups.*

Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek

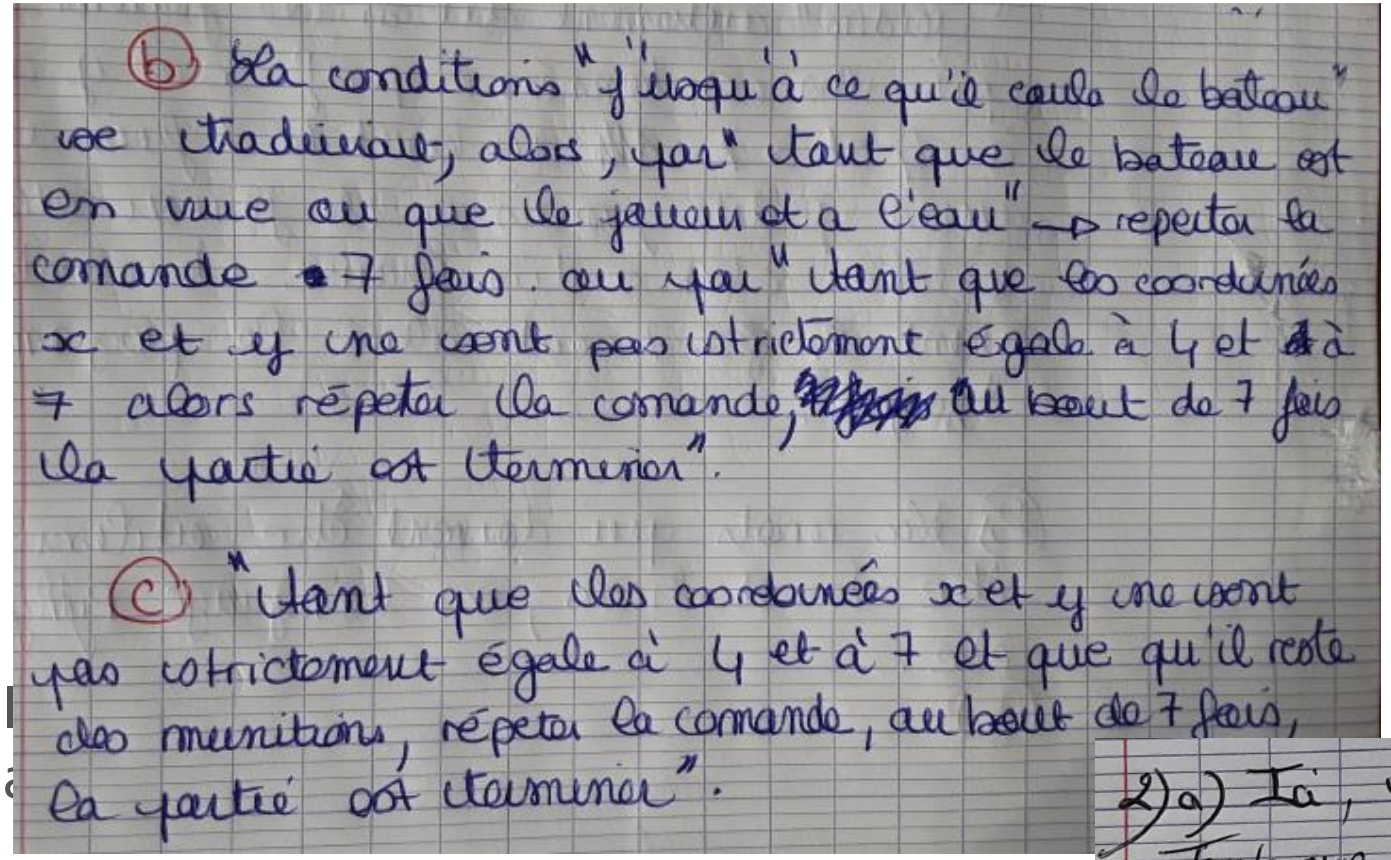
I. Quelques travaux d'élèves

II. Présentation d'un exemple



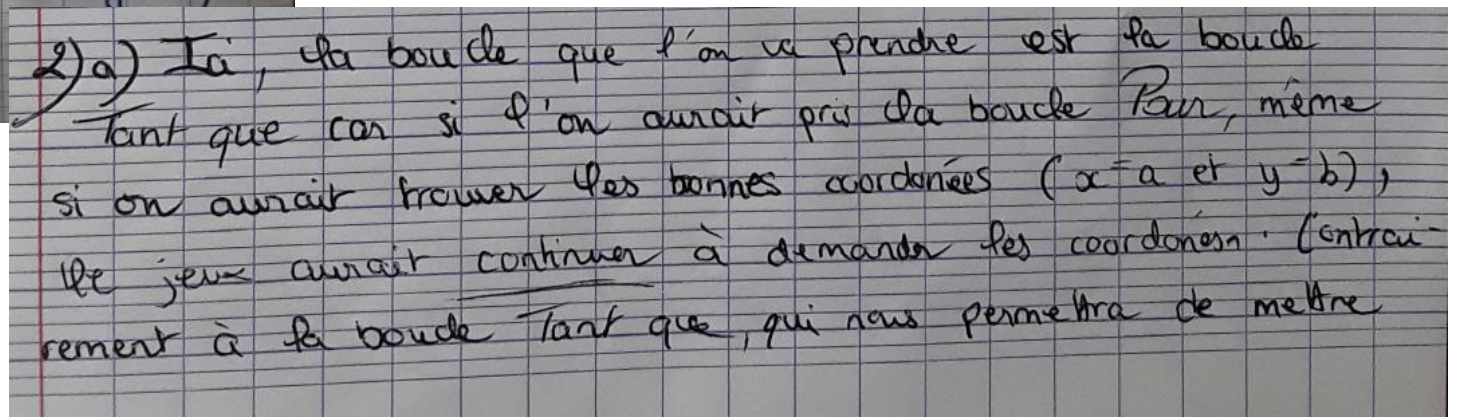


Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek



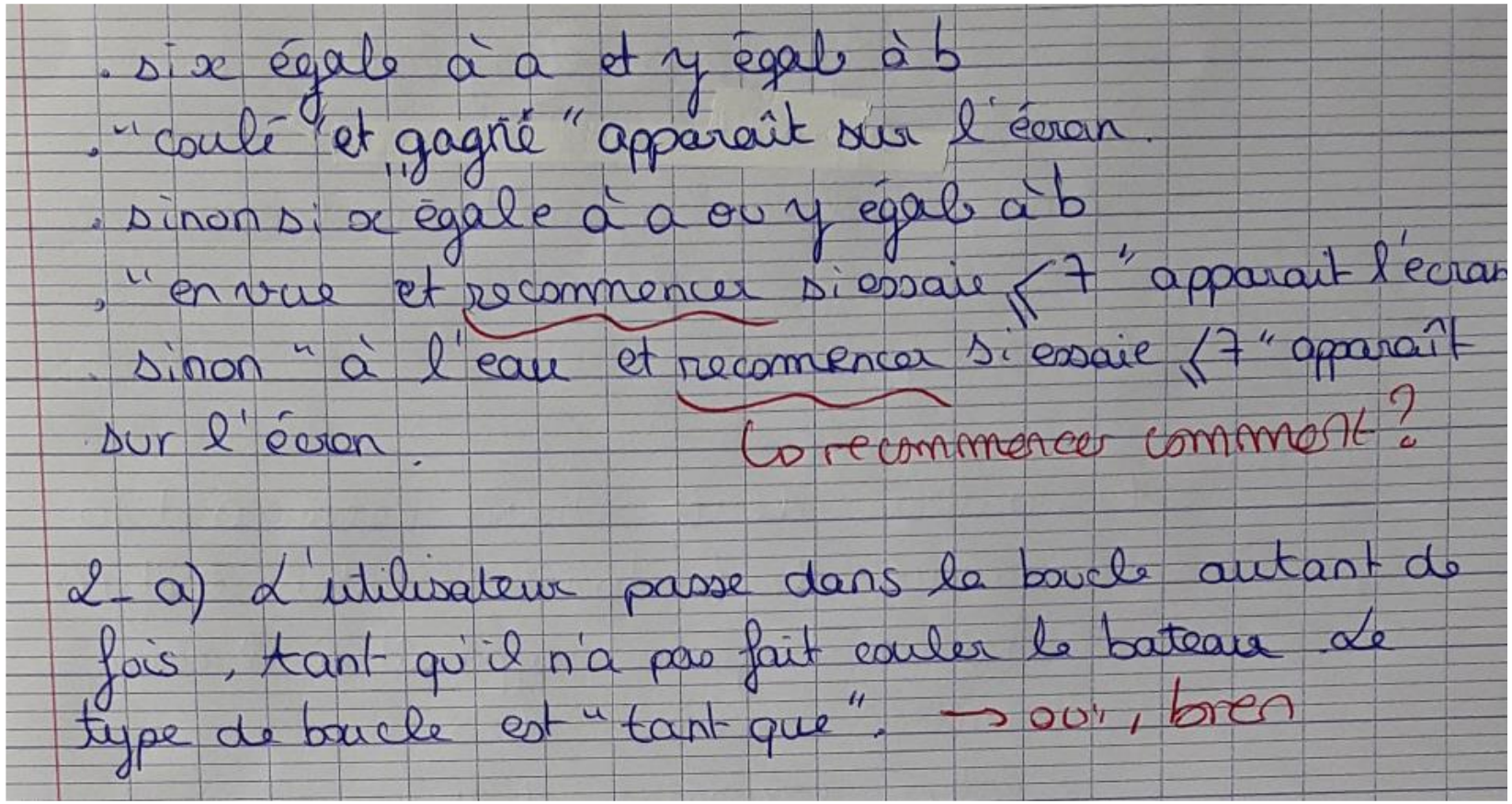
b) La conditions "jusqu'à ce qu'il coule le bateau" se traduirait, alors, par "tant que le bateau est en vue ou que le joueur est à l'eau" → repeter la comande 7 fois. ou par "tant que les coordonnées x et y ne sont pas strictement égale à 4 et à 7 alors répeter la comande, au bout de 7 fois la partie est terminer".

c) "tant que les coordonnées x et y ne sont pas strictement égale à 4 et à 7 et que qu'il reste des munitions, répeter la comande, au bout de 7 fois, la partie est terminer".

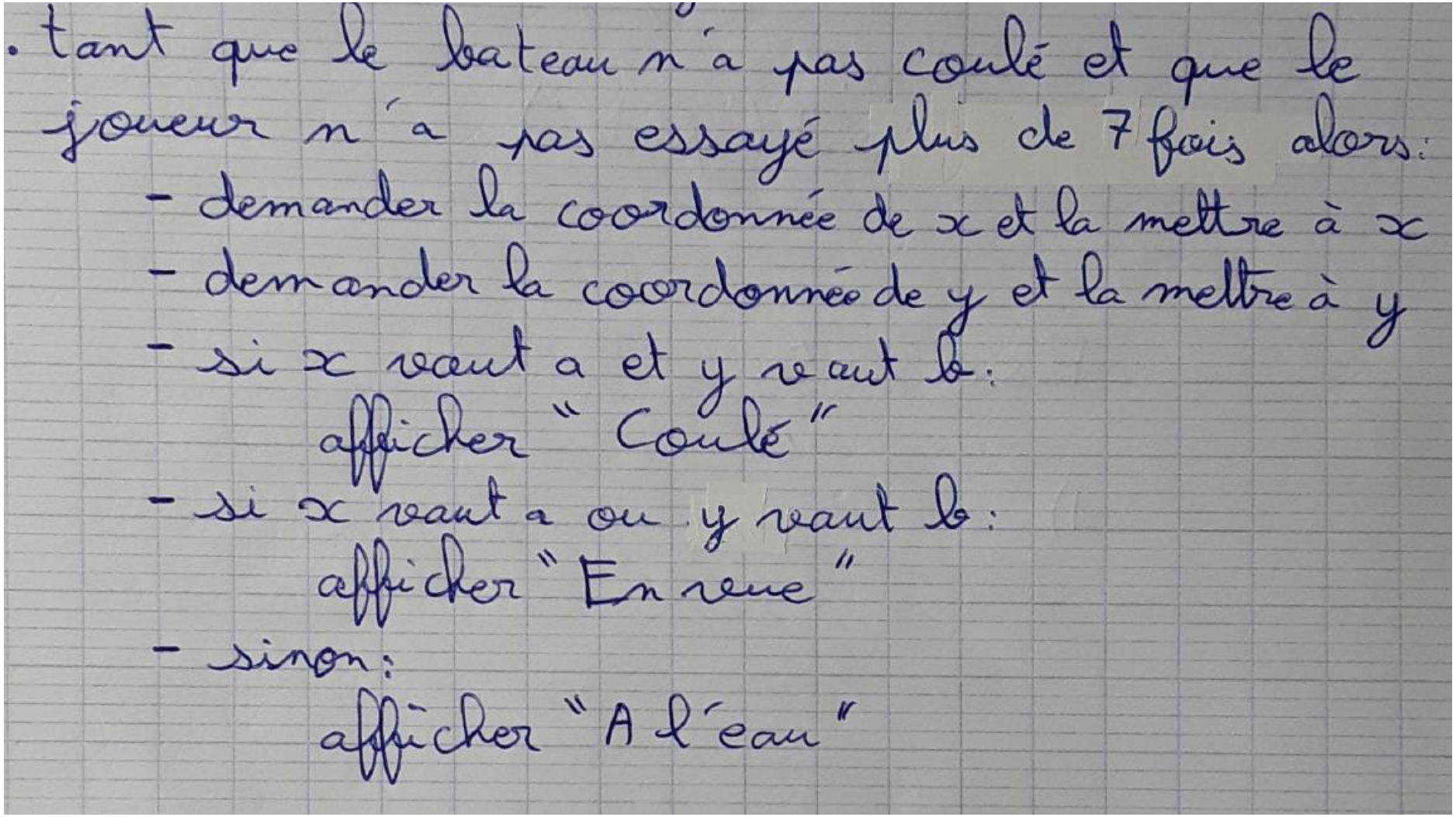


2)a) Ici, la boucle que l'on va prendre est la boucle Tant que car si l'on avait pris la boucle Pour, même si on aurait trouver les bonnes coordonées (x=a et y=b), le jeux aurait continuer à demander les coordonées. Contrairement à la boucle Tant que, qui nous permetra de mettre

Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek


si x égale à a et y égal à b
"coulé" et "gagné" apparaît sur l'écran.
sinon si x égale à a ou y égal à b
"en vue et recommencer si essaie ≠ 7" apparait l'écran
sinon "à l'eau et recommencer si essaie ≠ 7" apparaît sur l'écran.
→ recommencer comment ?
2- a) L'utilisateur passe dans la boucle autant de fois, tant qu'il n'a pas fait couler le bateau. Le type de boucle est "tant que". → oui, bien

Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek



Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek

I. Quelques travaux d'élèves

II. Erreur et obstacle, apports didactiques

Coup de pouce 2 : Exercice 3

On se place dans le cas où la position du bateau est donnée par $a = 4$ et $b = 7$.

1) Exprimer la condition « le bateau est coulé » en fonction des variables x et y.
2) Traduire en français la condition « jusqu'à ce que le bateau soit coulé » en une condition « tant que ».
3) Compléter le tableau suivant.

x	y	`x==a` ?	`y==b` ?	Message affiché	Joue-t-on encore ?
2	4				
4	5				
2	7				
4	7				

4) En s'aidant des questions précédentes, traduire la condition « jusqu'à ce que le bateau soit coulé » à l'aide d'une structure de type « TANT QUE » et en fonction des variables x et y.

Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek

I. Quelques travaux d'élèves

II. Erreur et obstacle, apports didactiques

Coup de pouce 3 : Exercice 3

On souhaite désormais ajouter la condition « dans la limite de 7 coups ». Pour cela, on introduit une variable `nombre_coups` qui vaut 0 au début du jeu, et qui sera augmentée de 1 à chaque tentative.

1) Quelle est la commande (en langage naturel, puis en Python) à indiquer pour augmenter la variable `nombre_coups` de 1 ?
2) Exprimer la condition « dans la limite de 7 coups » à l'aide d'une structure de type « tant que » et en fonction de la variable `nombre_coups` : `TANT QUE nombre_coups...`
3) Ajouter cette condition à la condition « jusqu'à ce que le bateau soit coulé » précédemment définie, au sein de la boucle `TANT QUE` et à l'aide d'un connecteur logique `OR` (« ou ») ou `AND` (« et »).

Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek

I. Quelques travaux d'élèves

II. Erreur et obstacle, apports didactiques

Exercice 4 : Le programme annonce au joueur s'il a perdu ou gagné et le nombre de coups joués.

1) Traduire **en français** à quelle(s) condition(s) le programme va renvoyer « gagné » ou « perdu ».
2) Quelle est le type de structure algorithmique à utiliser ?
   Traduire les conditions de la question 1) en langage algorithmique.
3) *En Python : à l'aide d'une instruction* `if, else`, *indiquer à la fin du programme si vous avez gagné ou non, et le nombre de coups joués. Penser à utiliser une variable qui compte le nombre de coups !*

Exercice 5 : Une partie de bataille navale est constituée de 5 manches.

1) Si on veut faire 5 manches, il faut répéter le programme ci-dessus cinq fois. Exprimer **en français** le déroulement d'une partie de 5 manches.
2) Cette fois, étant donné que l'on sait le nombre de manches à effectuer, quel est le type de boucle à utiliser ?
   Traduire le déroulement de la question 1) en langage algorithmique.
3) *En Python : implémenter cette boucle pour répéter 5 fois le programme développé précédemment. À la fin, faire afficher combien de manches ont été gagnées.*

Extrait du mémoire de Camille Sutour M2 MEEF 2019 à partir d'un TP proposé dans le livre de G.Dowek

Bilan

- Intégrer lors de la préparation des cours la réflexion autour des compétences identifiées
- Prendre en compte les travaux des élèves et repérer les principales erreurs afin d'identifier ce qui pose problème pour voir comment l'aborder et amorcer la discussion dans une perspective d'enseignement:
  - La notion de variable
  - L'affectation
  - La complexité
  - Le typage
  - La récursivité…
- Donner des clés aux élèves pour qu'ils identifient et comprennent leurs erreurs

Bibliographie et sitographie

- C.Declercq, diaporama formation DIU de Nantes
- http://pythontutor.com/
- Dagiene, V., Sentance, S. et Stupuriene, G. (2017). Developing a Two-Dimensional Categorization System for Educational Tasks in Informatics. *Informatica*, 28(1), 23–44.