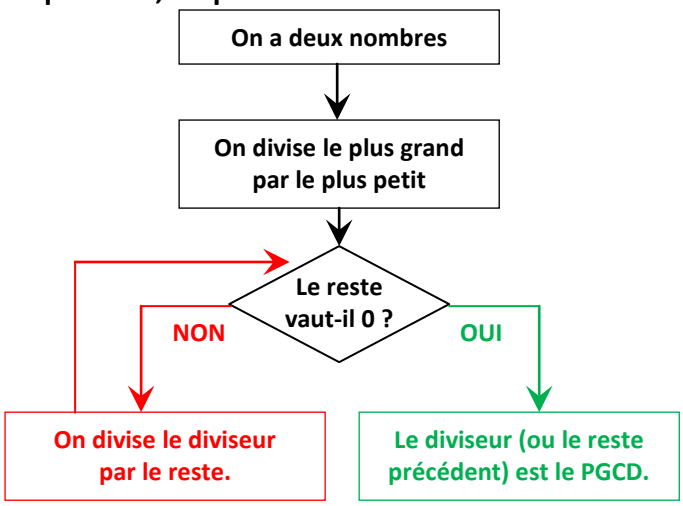
Chapitre 7 ~ Algorithmique

I – Algorithme sur un exemple connu

1. Rappel de l'algorithme d'Euclide (PGCD)

Définition

L'algorithme d'Euclide est un procédé qui permet de calculer le PGCD de deux nombres entiers naturels *a* et *b*. Schématiquement, on peut le noter :



Exemple : Calculer le PGCD de 320 et 460.

$460 = 320 \times 1 + 140 \quad \Rightarrow \quad 320 = 140 \times 2 + 40 \quad \Rightarrow \quad 140 = 40 \times 3 + 20 \quad \Rightarrow \quad 40 = 20 \times 2 + 0.$

2. Application

Définition

Un algorithme est une liste d'instructions à effectuer pas à pas afin d'accomplir une tâche précise. Il doit être *systématique* (= fonctionne quelles que soient les valeurs de départ) et *mécanique* (= pas besoin de réfléchir, il suffit d'appliquer la méthode !).

Exemples :
- l'algorithme d'Euclide présenté ci-dessus ;
- le partage d'un segment en 5 parties de longueurs égales ;
- le flocon de von Koch ;
- la construction d'un pentagone régulier ;
- …

Il peut être utile de savoir coder les algorithmes ci-dessus dans une calculatrice ou un logiciel de calcul formel (qui sait utiliser les lettres). Avant, il faut donner l'algorithme que l'on traduira plus tard dans le langage de programmation souhaité (malheureusement, ils sont tous différents : TI, Casio, Python, Maple, …). Pour l'algorithme d'Euclide, on dirait alors

```
Demander l'entier le plus grand a
Demander l'entier le plus petit b
Faire division euclidienne de a par b (qui donne q et r)
Tant que r ≠ 0
        a ← b, b ← r
        q ← quotient, r ← reste
Fin de boucle
Afficher b
```

Voici ce que cela donne sur TI et langages de programmation :

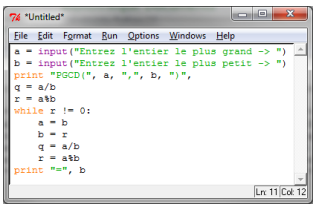
TI-82 & TI-83 Plus

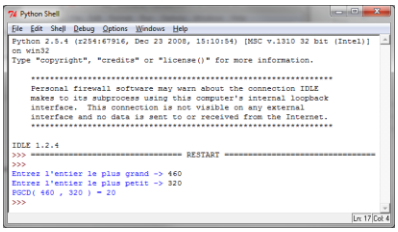
```
PROGRAM:PGCD
:Input "NB LE + GRAND : ",A
:Input "NB LE - GRAND : ",B
:int(A/B)→Q
:A-B*Q→R
:While R≠0
:B→A
:R→B
:int(A/B)→Q
:A-B*Q→R
:End
:Disp "PGCD=",B
```

↓

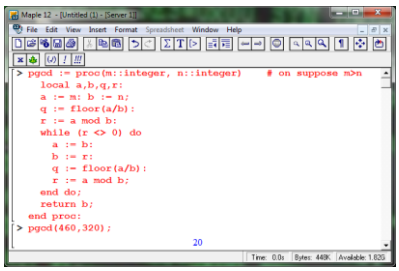
```
NB LE + GRAND :
460
NB LE - GRAND :
320
PGCD=
                20
              Done
```

Python





Maple



Nous allons travailler cette année avec Python car il présente de nombreux avantages :
- il est gratuit, prêt à l'emploi (pas d'installation : récupérer le zip et le décompresser sur une clé USB) ;
- il est facile et assez intuitif, surtout pour ceux qui n'ont jamais fait de programmation.

Remarques

- Pour récupérer Python, rendez-vous sur le site www.capes-de-maths.com, rubrique Lycée puis Seconde, récupérer le fichier « Python.zip », et décompressez-le sur une clé USB ou dans un répertoire de votre disque dur ! Rien de plus simple…
- L'interface de Python est en mode interactif (on tape des instructions qu'on valide une à une), mais on peut aussi écrire des programmes (comme celui-ci-dessus) en appuyant sur Control + N, puis à la fin de l'écriture en tapant F5.

II – Python

1. Calculs

Dans Python, les calculs s'écrivent de façon naturelle... à quelques exceptions près !

```
>>> 4*3+1                          # On utilise l'étoile du clavier comme signe multiplié
13

>>> a = 5
>>> 2*a+3*(1+4)                    # Attention, on ne peut omettre aucun signe multiplié !
25                                 # Taper 2a+3(1+4) génère une erreur.

>>> 14/4                           # Surprise : Ici les opérandes sont entiers donc Python
3                                  # fait une division entière !
>>> 14.0/4                         # En revanche, si au moins un des opérandes est réel
3.5                                # alors le résultat sera un réel.

>>> 14%4                           # L'opérateur % donne le reste de la division entière
2                                  # Ici, 14 = 4 * 3 + 2

>>> 2**3                           # 2 puissance 3
8

>>> from math import *             # Les fonctions et constantes mathématiques ont été
>>> sqrt(4)                        # regroupées dans le module « math ».
2.0                                # Avant de pouvoir les utiliser, il faut charger ce module
>>> pi                             # avec « from math import * », sans quoi on a des erreurs...
3.1415926535897931
```

2. Variables

Une variable permet de stocker une valeur que l'on compte réutiliser plus tard.

```
>>> a = 5                          # On utilise le signe égal pour « affecter » une nouvelle
>>> b = 2*a+5                      # valeur à une variable.
>>> a = a+1                        # Attention : Ce signe égal n'a rien à voir avec celui d'une
                                   # identité ou d'une équation !!

>>> nom = "Fontanet"               # Pour faciliter la clarté d'un programme, on essaye de
>>> code_postal = "78000"          # donner des noms significatifs aux variables !
```


Remarques

- Le nom d'une variable commence par une lettre puis est suivi de lettres, de chiffres ou de soulignés. Attention : les lettres doivent être non accentuées.
- Le nom d'une variable est « sensible à la casse » : `Prenom` et `prenom` ne désignent pas la même variable.

Exercices :

1. Quelles sont les variables dont le nom va déclencher une erreur :
   a) « Pépite2 » b) « 2pepites » c) « pepite.2 » d) « pepi2te »
   e) « tourne-vis » f) « tourne_vis » g) « tourneVis » h) « tourne vis »
2. Soit les instructions : `a=5; b=3; c=a+b; a=2; c=b-a`. Combien vaut `c` ?
3. Soit les instructions : `a=5; b=3; a=a+b; b=a+b; c=a+b`. Combien vaut `c` ?
4. On a deux variables `a` et `b`. Quelles instructions taper pour permuter les contenus de `a` et `b` ?
5. Même chose avec trois variables `a`, `b` et `c`. Quelles instructions taper pour avoir `a` → `b` ; `b` → `c` et `c` → `a` ?

3. Afficher une valeur (« output »)

En mode interactif, il suffit de taper un calcul ou le nom d'une variable pour afficher le résultat de ce calcul ou le contenu de cette variable.

```
>>> gateau = "charlotte au chocolat"
>>> gateau
'charlotte au chocolat'
>>> x = 2.5
>>> x**2+2*x+1
12.25
```

En revanche, dans un script, nous devrons utiliser l'instruction `print`.

```
age = 15
print "J'ai", age, "ans, bientôt", age+1                    # Affiche : J'ai 15 ans, bientôt 16

                                                            # Affiche sur deux lignes :
print "Bonjour"                                             #   Bonjour
print "les amis"                                            #   les amis

print "Bonjour",                                            # Affiche sur une seule ligne :
print "les amis"                                            #   Bonjour les amis
```


Remarques

- `print` permet d'afficher tout ce que l'on veut : chaînes de caractères, nombres, variables, résultats de calculs,...
- Dans la même instruction `print`, on peut afficher plusieurs choses en les séparant par des virgules.
- À la fin d'un `print`, Python passe à la ligne sauf si ce `print` se termine par une virgule.

4. Demander une valeur dans un script (« input »)

Lors de l'exécution d'un script, on peut « poser une question » à l'utilisateur.

- Si la réponse attendue est un nombre, on utilisera la fonction `input()`
- Si la réponse attendue est une chaîne de caractères, on utilisera la fonction `raw_input()`

```
largeur = input("Entrer la largeur du rectangle : ")
longueur = input("Entrer la longueur du rectangle : ")
print "La surface est de", largeur*longueur

ville = raw_input("Dans quelle ville habitez-vous ? ")
print "Oh, mais", ville, "est un très bel endroit !"
```

Exercices :

1. Écrire un programme qui demande un nombre et qui affiche son carré.
2. Écrire un programme qui demande un rayon et qui affiche le périmètre du cercle et l'aire du disque.
3. Écrire un programme qui demande deux entiers positifs et affiche la division euclidienne du premier par le second sous la forme : `14 = 4 * 3 + 2`
   (*rappel* : Avec des entiers, l'opérateur `/` donne le quotient entier et l'opérateur `%` donne le reste.)
4. Écrire un programme qui demande un nombre de secondes et le converti en heures/minutes/secondes.

	En classe : 1 p. 12	Exercices : 61 p. 39 + 15 p. 63

5. Tests (« if »)

Pour réaliser un test en Python, on utilise : `if elif else` (**`else`** veut dire « sinon » et **`elif`** est la contraction de « else if »).

```
nationalité = raw_input("Quelle est votre nationalité ?  ")
if nationalité == "anglais" or nationalité == "américain":
    print "hello !"                                  # 1er bloc d'instructions exécuté uniquement
    print "glad to meet you"                         # si la condition ci-dessus est vraie
elif nationalité == "français":
    print "salut !"                                  # 2ème bloc d'instructions
else:
    print "moi pas comprendre"                       # 3ème bloc d'instructions
```


Remarques

- En Python, un bloc d'instructions commence par « : » suivi d'une indentation (= retrait de paragraphe, touche Tab) et se termine à la fin de cette indentation.
- Derrière `if` et `elif`, Python attend une condition qui doit être `True` ou `False`. Pour cela, on peut utiliser un opérateur de comparaison (`==`, `!=`, `<`, `>`, `<=`, `>=`) ou en combiner plusieurs avec `and` , `or` ou `not`.
- Attention à ne pas confondre :
  `nationalité == "anglais"` (avec deux signes égal) qui est une comparaison
  et `nationalité = "anglais"` (avec un seul signe égal) qui est une affectation de variable.
- On peut utiliser `if` sans `elif` ni `else` ou avec un seul des deux. On peut également mettre plusieurs `elif`.

Exercices :

Programmes :

1. Écrire un programme testant si un nombre est pair ou non.
   On pourra utiliser l'opérateur `%` qui donne le reste de la division entière de deux nombres.
2. Écrire un programme testant si un nombre appartient aux ensembles suivants : $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}^+$, $\mathbb{R}$.
   On pourra utiliser la fonction **`int()`** qui retourne la partie entière d'un nombre.
3. Écrire un programme qui demande trois nombres distincts puis les classe en ordre croissant.
   On pourra entrer les trois nombres en une seule instruction :
   `a,b,c = input("Entrez 3 nombres distincts en les séparant par une virgule : ")`
4. Écrire un programme qui demande à l'utilisateur les valeurs de a et b, puis résout $a\,x + b = 0$.
   (Ne pas oublier de traiter le cas ou a est nul...)

Écriture de conditions :

5. Que vaut : « `(1<x and x<=5) or x==12` » si $x = 3$? et si $x = 5$? et si $x = 7$? et si $x = 12$?
6. Parmi les conditions ci-dessous, deux sont toujours équivalentes : lesquelles ?
   « not (a or b) », « (not a) or (not b) », « (not a) and (not b) »
7. Parmi les expressions ci-dessous, quelles sont celles qui sont correctes ?
   « `a == 1 or 2` », « `a == 1 or a == 2` », « `(a == 1) or (a == 2)` »

TP : 1 p. 56 + 4 p. 137		Exercices : 20 p. 92

6. Boucles (« while »)

Dans un algorithme, il arrive que l'on doive répéter le même bloc d'instructions plusieurs fois. Plutôt que de réécrire ce bloc plusieurs fois dans le script, on fait ce que l'on appelle une boucle. En Python, il y a deux types de boucles : les boucles `while` et les boucles **`for`**.
Commençons ici par les boucles `while` qui permettent d'exécuter une boucle « tant qu'une » condition est vérifiée.

```
reponse = raw_input('Comment dit-on "salut" en anglais ?  ')
while reponse != "hello":
    print "Désolé, ce n'est pas ça !"
    reponse = raw_input("Allez, on réessaye :  ")
print "Yes, you 're right !!!"
```


Remarques

- Ne pas oublier, ni les deux points en fin de ligne `while`, ni l'indentation.
- Pour insérer `"salut"` entre guillemets dans le message de départ, on a délimité la chaîne de caractères par des apostrophes !

Exercices :

1. Écrire un programme qui demande un nombre supérieur à un, puis l'encadre entre deux puissances consécutives de deux. (Par exemple qui encadre 5 entre 2^2 et 2^3)
2. Écrire un programme qui demande deux nombres positifs (en premier le plus petit, puis le plus grand) et affiche tous les entiers compris entre ces deux nombres.
3. Même exercice avec deux nombres de signes quelconques et pas forcément dans l'ordre…
4. Écrire un programme qui choisit un entier au hasard entre 0 et 100 puis demande à l'utilisateur de le deviner en lui répondant à chaque essai « trop petit » ou « trop grand ». Le programme s'arrête quand l'utilisateur a trouvé le bon nombre. *(Pour savoir comment générer un nombre aléatoire avec Python, cherchez sur Internet !)*
5. Écrire un programme qui permet à l'utilisateur d'entrer une série de notes puis qui calcule la moyenne de ces notes dès que la dernière note entrée n'est pas comprise entre 0 et 20. (On ne sait donc pas à l'avance combien de notes vont être entrées)
6. Même principe que le programme précédent sauf qu'au lieu de calculer la moyenne, on renvoie la meilleure note et la moins bonne.
7. La population de Bigcity augmente de 3% par an. Écrire un programme qui permet de déterminer dans combien d'années elle aura doublé.
8. Écrire un programme qui demande un entier puis détermine s'il est premier ou non.

TP : 2 p. 56 + 1 p. 88		Exercices : 44 p. 95

7. Boucles (« for »)

Les boucles **`for`** permettent d'exécuter une boucle « pour » toutes les valeurs d'une liste.

```
>>> for i in [12, 45, 7]:
        print i
12
45
7
```

La liste à parcourir est souvent définie par la fonction `range([début,] fin [, pas])`

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(-10, -100, -30)
[-10, -40, -70]

>>> for i in range(3):
        print i
0
1
2
```


Remarque

`range(10)` s'arrête à 10 non compris !

Exercices :

1. Écrire un programme qui demande un entier puis affiche la table de multiplication de cet entier. (Ex : 4 × 1 = 4 ; 4 × 2 = 8 ; 4 × 3 = 12 ; … ; 4 × 10 = 40)
2. Écrire un programme qui affiche les carrés des entiers de 1 à 10
3. Écrire un programme qui calcule la somme des 100 premiers entiers naturels (de 1 à 100 compris) en utilisant une boucle `for` puis écrire un autre programme qui fait la même chose avec une boucle `while`.
4. Écrire un programme qui demande un entier naturel puis calcule la factorielle de cet entier. (La factorielle d'un entier n est notée $n!$ et vaut $1 \times 2 \times 3 \times 4 \times \cdots \times n$)
5. Écrire un programme qui demande un entier naturel n, puis affiche les n premiers nombres pairs.
6. Écrire un programme qui demande un entier naturel puis affiche tous les diviseurs de cet entier.
7. Écrire un programme qui demande un nombre n, puis affiche une suite de 10 nombres qui commence avec n et ou chacun des termes qui suit est le carré du précédent.

8. Divers

a) *Le point-virgule :*

```
>>> a=10 ; print a              # On peut écrire plusieurs instructions sur une même ligne
10                              # en les séparant par un point-virgule.
```

b) *Les fonctions de conversions de types :*

```
>>> a="3.5" ; print a
'3.5'
>>> b=float(a) ; print b          # Conversion en nombre réel.
3.5
>>> c=int(b) ; print c            # Conversion en entier relatif et troncature si nécessaire.
3
>>> d=str(c) ; print d            # Conversion en chaîne de caractères.
'3'
```

c) *La fonction eval() :*

```
>>> x = 2                         # La fonction eval évalue la chaîne de caractère passée
>>> eval('x+1')                   # en paramètre et retourne sa valeur.
3
```

d) *Les nombres aléatoires :*

```
>>> from random import *          # Commencer par charger le module « random » avant d'utiliser
                                  # les fonctions ci-dessous.

>>> random()                      # Retourne un réel x tel que 0.0 <= x < 1.0
0.38094480454925794
>>> uniform(1, 10)                # Retourne un réel x tel que 1.0 <= x < 10.0
5.030133066130043
>>> randint(1, 10)                # Retourne un entier n tel que 1 <= n <= 10
3
```

e) *Les listes Python :*

```
>>> a=[1, 5, 8, 3, 7]            # On définit ici ce que l'on appelle une liste
>>> print a                       # en la délimitant par des crochets.
[1, 5, 8, 3, 7]

>>> a[0]                          # a[0] est le premier élément de la liste. (et non a[1] !)
1
>>> a[-2]                         # a[-2] est le 2ème élément de la liste en partant de la fin.
3
>>> a[2]=10 ; print a             # On peut modifier n'importe quel élément d'une liste.
[1, 5, 10, 3, 7]
>>> max(a)                        # Retourne le plus grand élément de la liste.
10
>>> min(a)                        # Retourne le plus petit élément de la liste.
1
>>> len(a)                        # Retourne le nombre d'éléments de la liste.
5
>>> sorted(a)                     # Retourne la liste triée.
[1, 3, 5, 7, 10]
```

f) *Les commentaires :*

Une grande partie du travail d'un programmeur consiste à reprendre des anciens scripts pour les adapter à un nouveau besoin ou pour les améliorer. Du coup, dès qu'il y a un passage un peu délicat dans un script, il est très fortement conseillé d'insérer des commentaires afin de faciliter la compréhension.
Les commentaires commencent avec un `#` et se terminent en fin de ligne.

```
a=1000                            # Ceci est un commentaire tout à fait inutile !!
```

g) *Le module « tortue » :*

Le module « turtle » permet de pouvoir dessiner dans la fenêtre du logiciel. Pour l'importer, saisir `from turtle import *`. Voici les principales fonctions de ce module :

* `reset()` Efface l'écran
* `goto(x,y)` Se déplace au point de coordonnées (x ; y)
* `forward(l)` Avance d'une longueur l
* `backward(l)` Recule d'une longueur l
* `left(a)` Tourne à gauche de a degrés
* `right(a)` Tourne à droite de a degrés
* `up()` Lève le crayon (ne dessine plus)
* `down()` Abaisse le crayon

Voir les exemples du livre p. 16.