

## TP 2 : Racines $n$ -ièmes de l'unité

L'objectif de ce TP est de réaliser une *procédure* Maple permettant de représenter un polygone du plan dont les sommets sont les points d'affixes les racines  $n$ -ièmes de l'unité.

Il s'agit dans un premier temps de se familiariser avec certaines notions de programmation et certains objets Maple.

## 1 La notion de liste

### 1.1 Séquences

Il s'agit d'une succession ordonnée d'opérandes séparées par des virgules. Exemples :

```
A := 2, a+1, cos, 2*9;
```

La séquence vide est définie par `B := NULL;`.

Pour rajouter un élément à une séquence, on ajoute l'élément précédé d'une virgule :

```
A := A, x;
```

La fonction `seq` permet de créer des séquences sans énumérer tous les éléments de la séquence.

**Exemple :** Exécuter la commande suivante : `C := seq(k**2, k=0..10);`

La  $k$ -ième opérande d'une séquence `S` est retournée à l'aide de la commande `S[k]`.

On peut déterminer la somme, le produit, le maximum ou le minimum d'une séquence à l'aide des fonctions `sum`, `add`, `mul`, `product` `max`, `min`...

**Exercice 1.** Créer une séquence qui contient tout les carrés des entiers de 0 à 15 puis donner la somme et le produit des éléments de cette séquence.

### 1.2 Liste

Une liste est une séquence entre crochets `[]`.

La fonction `nops` donne le nombre d'opérandes d'une liste. Pour extraire la  $k$ -ième opérande d'une liste `L` on peut utiliser les commandes `op(k,L)` ou `L[k]`. La fonction `op` renvoie la séquence formée des éléments de la liste.

**Exercice 2.** Exécuter la série de commandes suivantes. Que réalise-t-on ici ?

```
>A := [seq(1/i, i=1..5)];
```

```
>A := [op(A), 1/6];
```

```
>A[6];
```

## 2 Procédures

### 2.1 définition

Une procédure est un groupe d'instructions placées dans un même bloc débutant par `nom :=proc(n)` et se terminant par `end;`.

Une procédure permet de réaliser des fonctions.

**Exercice 3.** Exécuter les lignes de commandes suivantes. Quelle opération réalise la procédure ?

```
>puissance :=proc(n) ;
2**n ;
end ;
>puissance(3) ;
```

## 2.2 Structure conditionnelle : boucles if.

La commande `if` permet de tester une propriété. La syntaxe générale pour une telle commande est la suivante :

```
if condition then instruction ; else instruction ; end if.
```

**Remarque :** pensez à bien fermer votre boucle avec `end if`.

**Exercice 4.** Exécuter la procédure suivante. Que réalise-t-on ici ?

```
>vab :=proc(x)
if x>0 then x else -x ;
end if ;
end ;
```

**Exercice 5.** Réaliser une procédure `ent` avec un entier  $n$  en entrée qui renvoie la partie entière de  $\frac{n}{2}$ . Indication : utiliser la commande `type(x, integer)`.

## 2.3 Structure itérative : boucles for.

Le principe d'une boucle itérative est de réaliser une action un certain nombre de fois jusqu'à qu'une condition soit réalisée. La syntaxe générale est :

```
for variable from début to fin while condition do instruction ; end do ;
```

La boucle se termine quand  $k$  arrive à la valeur de `fin` imposée ou quand la `condition` est réalisée.

Par exemple, la procédure suivante calcule la somme des entiers de 1 à  $n$ . Exécuter puis analyser le fonctionnement de cette procédure :

```
>somme :=proc(n)
local k,S ;
S :=0 ; for k from 1 to n do S :=S+k ; end do ;
end ;
```

**Exercice 6.** A l'aide d'une boucle `for`, créer une procédure `fact` dont la variable d'entrée est  $n$ , qui retourne le nombre  $n!$ .

**Exercice 7.** A l'aide d'une boucle `for` et de la commande `while`, créer une procédure `sominv` dont la variable d'entrée est un réel positif  $a$  et qui renvoie le plus petit entier  $n$  tel que :

$$\sum_{k=1}^n \frac{1}{k} \geq a.$$

*Remarque :* il n'est pas a priori évident que cet algorithme se termine.

**Exercice 8.** A l'aide d'une boucle `for` et de la fonction `isprime`, créer une procédure `premier` dont la variable d'entrée est un entier  $n$  et qui renvoie la liste des  $n$  premiers nombres premiers.

**Exercice 9.** La suite de syracuse  $(s_n)_{n \geq 1}$  est définie de la manière suivante :  $s_1 \in \mathbb{N}$  et

$$s_{n+1} = \begin{cases} \frac{s_n}{2} & \text{si } s_n \text{ est pair} \\ 3s_n + 1 & \text{si } s_n \text{ est impair} \end{cases}$$

On conjecture que pour toute valeur initiale  $s_1$ , il existe un rang  $n$  tel que  $s_n = 1$ . Créer une procédure `syracuse` dont la variable d'entrée est  $x = s_1$ , qui détermine le plus petit entier  $n$  tel que  $s_n = 1$  (on ne sait donc pas si cet algorithme se termine).

### 3 Réalisation d'un polygone régulier

#### 3.1 Les complexes avec Maple

On rappelle que le nombre complexe  $i$  se note en langage Maple `I`. Concernant les nombres complexes, on dispose des fonctions : `Re`, `Im`, `conjugate`, `polar`, `evalc`...

Maple connaît bien sur la notation exponentielle pour les complexes : utiliser pour cela la fonction `exp`.

**Exemple.** Exécuter les lignes suivantes :

```
>exp(2*I*pi/3) ;
>evalc(exp(2*I*pi/3)) ;
```

**Exercice 10.** Déterminer la partie réelle et la partie imaginaire de  $\left(\frac{1+2i}{3-i}\right)^5$ . Donner une valeur approchée pour son argument.

**Exercice 11.** Créer une procédure dont la variable d'entrée est un entier  $n$  et qui renvoie la liste des racines  $n$ -ième de l'unité.

**Exercice 12.** A l'aide d'une seule racine de l'unité  $\omega = e^{\frac{2i\pi}{n}}$  et d'une boucle `for` créer une procédure qui retourne la liste des racines  $n$ -ième de l'unité (i.e il s'agit de générer la liste  $[1, \omega, \omega^2, \dots, \omega^{n-1}]$ ).

#### 3.2 Points du plan

Maple dispose d'un package `geometry` qui regroupe des fonctions utiles pour la géométrie. Il n'est pas nécessaire d'utiliser ce package pour ce TP mais ce n'est pas interdit.

Un point du plan est reperé par un couple  $[x, y]$ .

**Exercice 13.** Exécuter la commande suivante. Qu'a-t-on tracé ?

```
>plot([[1,0], [0,1], [-1,0], [1,0]])
```

La commande `complexplot` permet de représenter des nombres complexes dans le plan. Cette commande appartient au *package* `plot` ; il faut donc la charger à l'aide de la commande `with(plots)`.

**Exercice 14.** A l'aide de la commande `complexplot`, tracer les point du plan d'affixes  $1, j$  et  $j^2$ .

#### 3.3 Synthèse

A l'aide de tout ce que vous venez d'apprendre, créer une procédure dont la variable d'entrée est un entier  $n$ , et qui trace un polygone régulier à  $n$  côtés (i.e. qui trace les points d'affixes les racines  $n$ -ièmes de l'unité).

Pour les plus courageux, créer une procédure dont les variables d'entrées sont un entier  $n$  et un complexe  $z \in \mathbb{C}^*$  qui trace les points d'affixes les racines  $n$ -ièmes de  $z$ .