TP 2 : Expressions et calculs avec Python

◇ On commencera par lancer PYZO avec l'icône sur le bureau.

Exercice 1 *Quelques calculs avec le module* **`math`**. On utilisera en général ce module en écrivant :

```
from math import *
```

au début d'un programme ou dans une console. On a ainsi accès directement à toutes les fonctions du module **`math`**. On pourra par exemple écrire :

```
print(pi,sin(pi),cos(pi))
print(e,exp(1),log(e))
```

La documentation du module **`math`** est accessible sur

`https://docs.python.org/3/library/math.html`

(a) Trouver la fonction du module **`math`** permettant de calculer $n!$ et l'utiliser pour calculer 8!.
(b) Quelle est la différence entre les fonctions **`ceil`** et **`floor`** du module **`math`** ? Dans quel(s) cas ces deux fonctions donnent-elles le même résultat ?
(c) Déterminer le PGCD de 1778 et 9732 en utilisant la fonction appropriée du module **`math`**.
(d) Comment calcule-t-on $\ln(x)$ avec les fonctions du module **`math`** ?

Exercice 2 *Datation au carbone* 14. À la mort d'un être vivant, le carbone 14 présent dans son organisme se désintègre au fil des années de sorte que, si p est la proportion de C_{14} restante au bout de N années, alors $N = -8310\ln(p)$.

(a) Écrire une fonction **`datation_C14(p)`** qui calcule N en fonction de p.
(b) Écrire une fonction **`proportion_C14(N)`** qui calcule p en fonction de N.
(c) La momie Ötzi retrouvée dans un glacier en 1991 contenait 52.8% du C_{14} initial à 1% près. Donner un encadrement de son âge.
(d) On considère que la datation au carbone 14 ne peut se faire que pour des éléments datant de moins de 50 000 ans et est approximative au delà de 35 000 ans. Donner les pourcentages de carbone 14 correspondant à ces durées.

Exercice 3 *Nombres complexes avec* PYTHON *et module* **`cmath`**. On peut définir et calculer avec des nombres complexes. Un nombre complexe $z = a + \mathrm{i}b$ s'écrit avec PYTHON sous la forme **`a+bj`** ou **`complex(a,b)`**. Attention : le nombre complexe i se note **`complex(0,1)`** ou **`0+1j`** ou plus simplement **`1j`**.

(a) Tester dans une console les commandes suivantes :

```
z1=2+1j
z2=complex(1,2)
z1+z2
z1*z2
z1/z2
z1**2
```

```
abs(z1)
z1.real
z1.imag
z1.conjugate()
```

⚠ Il y a bien des parenthèses après **conjugate** (et pas après **real** et **imag**).

(b) La documentation du module **cmath** est accessible sur

https://docs.python.org/3/library/cmath.html

Regarder les différentes fonctions proposées. Rechercher dans la documentation à quoi correspondent les fonctions **phase**, **polar** et **rect** ?

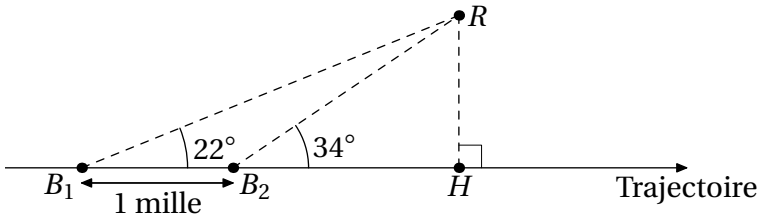
(c) Écrire sous forme algébrique les nombres complexes :

$$\frac{1-2\mathrm{i}}{3+\mathrm{i}}, \quad \frac{2+\mathrm{i}}{2-\mathrm{i}}, \quad \frac{(1-4\mathrm{i})(1-\mathrm{i})}{(1+\mathrm{i})^2}$$

Déterminer également le module et un argument de chacun de ces nombres.

(d) On considère les nombres $a = 3$, $b = 6$ et $c = 8+\mathrm{i}$. Vérifier numériquement, avec PYTHON, que $\arg(c) + \arg(c-a) + \arg(c-b) = \pi/4$ (à 2π près).

Exercice 4 *Calculs de distances.* Un bateau navigue en suivant un certain cap. À un instant donné, le bateau est en position B_1 et le commandant repère un récif R sous un angle de 22°. Un mille plus loin, en position B_2, le commandant repère ce même récif R sous un angle de 34°.



(a) Calculer la distance HR (distance entre le récif et la trajectoire). On pourra commencer par exprimer $\tan(22°)$ et $\tan(34°)$ en fonction des distances HR, B_1H et B_2H.
⚠ Vérifier dans la documentation du module **math** si les fonctions trigonométriques utilisent des radians ou des degrés.

(b) Écrire une fonction **distance_recif(d,angle1,angle2)** qui calcule et renvoie la distance entre la trajectoire à partir des deux angles mesurés (exprimés en degrés) et de la distance entre les deux mesures.

Corrections

Ex 1. On utilise la fonction **factorial** :

```
from math import *

print(factorial(8))
```

```
40320
```

La fonction **ceil** renvoie le plus petit entier supérieur (ou égal) à x tandis que **floor** renvoie le plus grand entier inférieur (ou égal) à x. On a par exemple **floor**$(2.5) = 2$ et **ceil**$(2.5) = 3$. Lorsque x est entier, les deux valeurs sont égales à x. La fonction **gcd** permet d'obtenir le PGCD :

```
print(gcd(1778,9732))
```

```
2
```

La fonction **log** correspond au logarithme de base e, ou encore logarithme népérien c'est à dire ln :

```
print(log(exp(5)))
```

```
5.0
```

Ex 2. Puisque $N = -8310\ln(p)$, on a $p = \exp(-N/8310)$. On définit donc les fonctions :

```
from math import *

def datation_C14(p):
    return -8310*log(p)

def proportion_C14(N):
    return exp(-N/8310)
```

La proportion de carbone 14 dans la momie est comprise entre 0.518 et 0.538 :

```
print(datation_C14(0.518),datation_C14(0.538))
```

```
5466.152105165254 5151.34173339713
```

L'âge de cette momie est donc compris entre 5151 ans et 5467 ans.

```
print(proportion_C14(35000),proportion_C14(50000))
```

```
0.014819772346644028 0.0024373420137081075
```

Après 35000 ans, il reste environ 1,5% du carbone 14 initial et environ 0.2% après 50000.

Ex 3. La fonction **abs** donne le module d'un nombre complexe, **phase** donne un argument (celui compris entre $-\pi$ et π) :

```
import cmath

z = 1+1j
print(abs(z),cmath.phase(z))
```

```
1.4142135623730951 0.7853981633974483
```

La fonction **polar** donne la forme polaire d'un nombre complexe (couple module, argument) et la fonction **rect** réalise l'opération inverse (construit un nombre complexe à partir de son module et de son argument) :

```
print(cmath.polar(z))
print(cmath.rect(1,cmath.pi/2))
```

```
(1.4142135623730951, 0.7853981633974483)
(6.123233995736766e-17+1j)
```

```
print((1-2j)/(3+1j))
print((2+1j)/(2-1j))
print((1-4j)*(1-1j)/(1+1j)**2)
```

```
(0.1-0.7j)
(0.6+0.8j)
(-2.5+1.5j)
```

```
a = 3
b = 6
c = 8+1j
print(cmath.phase(c)+cmath.phase(c-a)+cmath.phase(c-b),cmath.pi/4)
```

```
0.7853981633974483 0.7853981633974483
```

Ex 4. On a tout d'abord :

$$\tan(22°) = \frac{HR}{B_1H}$$
$$\tan(34°) = \frac{HR}{B_2H}$$

et ainsi :

$$B_1H = \frac{HR}{\tan(22°)}$$
$$B_2H = \frac{HR}{\tan(34°)}$$

Comme $B_1H - B_2H = B_1B_2$ (distance connue), on obtient :

$$B_1B_2 = HR\left(\frac{1}{\tan(22°)} - \frac{1}{\tan(34°)}\right) = HR\frac{\tan(34°) - \tan(22°)}{\tan(22°)\tan(34°)}$$

et ainsi :

$$HR = B_1B_2\frac{\tan(22°)\tan(34°)}{\tan(34°) - \tan(22°)}$$

Pour l'application numérique, on note que les fonctions trigonométriques de PYTHON fonctionnent en radians. On convertit en radians un angle exprimé en degrés en divisant par 180 et en multipliant par π (ou alors on utilise la fonction **radians** du module **math**) :

```
from math import *

a1 = 22*pi/180
a2 = 34*pi/180
B1B2 = 1
print("Distance HR =",B1B2*tan(a1)*tan(a2)/(tan(a2)-tan(a1)))
```

```
Distance HR = 1.0075303981590067
```

On peut utiliser ceci pour définir une fonction :

```
def distance_recif(d,angle1,angle2):
    a1 = angle1*pi/180
    a2 = angle2*pi/180
    return d*tan(a1)*tan(a2)/(tan(a2)-tan(a1))

print(distance_recif(1,22,34))
```

```
1.0075303981590067
```