C. Hinard, M. Miallier,
T. Prévost

Collège Croas ar Pennoc,
28 avril 2022

« L CODENT L CRÉENT »

Séance 2 (et 3 ?)

I
Rétrospective : On a appris quoi la dernière fois ?

On sait, la dernière fois remonte déjà à un bon moment qui, dans la rapidité de l'instant couplée aux enjeux qui sont ceux de la classe de troisième (coucou le brevet !), on a l'impression que c'était il y a une éternité...Mais pas de panique, on va refaire un tour de ce qui a été vu, jusque-là, comme ça on pourra repartir sur de bonnes bases !

1.1. Notre ami Python

Python est « un langage de programmation interprété, multi-paradigme et multiplate-formes [...] doté d'un typage dynamique fort » [1].

Mais concrètement, si on oublie les définitions longues et compliquées (moyennement intéressantes, on va se l'avouer...), Python est un langage informatique, autrement dit une manière de parler à un ordinateur afin de lui demander toutes sortes de choses (et c'est incroyable le nombre de choses que l'ordinateur sait faire).

Surtout, Python a le gros avantage de pouvoir être écrit mais surtout compris assez simplement, en étant assez proche du langage naturel, mais aussi celui de reconnaître tout seul le type d'une variable (c'est ça le typage dynamique).

1.2. Les bases

On dispose d'un bon nombre d'opérations de base en Python. À commencer par les 4 opérations de base :

Opération	Symbole	Exemple	Résultat de l'exemple
`addition`	`+`	`5 + 2`	
`soustraction`	`-`	`5 - 2`	
`multiplication`	`*`	`5 * 2`	
`division`	`/`	`5 / 2`	

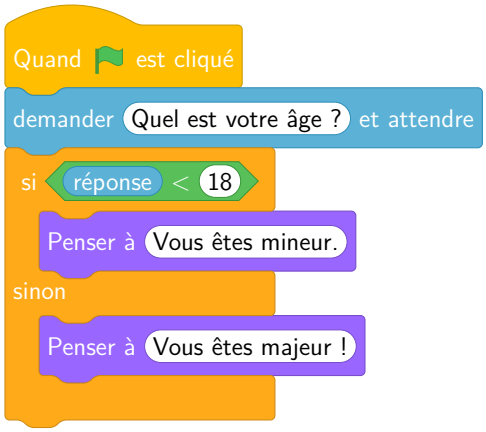
1. Merci Wikipédia !

Mais on a aussi une palanquée d'autres opérations dans notre inventaire :

Opération	Symbole	Exemple	Résultat de l'exemple
reste de la division	%	5 % 2	
puissance	**	5 ** 2	
division entière	//	5 // 2	
inférieur	<	5 < 2	
supérieur	>	5 > 2	
inférieur ou égal	<=	5 <= 2	
supérieur ou égal	>=	5 >= 2	
égalité	==	5 == 2	
différence	!=	5 != 2	
et	and	5 and 2	
ou	or	5 or 2	
non	not	5 not 2	

1.3. If, then, else

```
age = input("Quel est votre âge ?")
if int(age) < 18:
  print("Mineur !")
else:
  print("Mineur !")
```



II
Et aujourd'hui, commence par quoi ?

2.1. Boucle for

1. Recopier les programmes ci-dessous et les faire tourner. Que remarque-t-on ?

```
for i in range(1, 10):
    print(i)
```

```
for i in range(10):
    print(i)
```

2. Compléter le programme Python ci-dessous permettant de calculer la somme des nombres de 1 à 20 :

```
somme = ...
for i in range(1, ...):
  somme = ...
print(somme)
```

3. De la même manière, écrire un programme Python qui affiche la table de multiplication de 5.

2.2. Boucle `while`

1. Faire tourner les programmes suivants, et observer le résultat :

```
i = 0
while i != 10:
  print(i)
  i = i + 1
```

```
i = 0
while i < 10:
  print(i)
  i = i + 1
```

```
i = 0
while i < 10:
  print(i)
```

Que remarque-t-on avec le dernier ? Pourquoi ?

2. Compléter le programme Python ci-dessous permettant de trouver à partir de quel nombre la somme des entiers dépasse 1000 :

```
somme = ...
i = ...
while somme < ...:
  somme = ...
  i = ...
print(...)
```

3. Un avion, initialement à une vitesse de 180 nœuds, accélère à chaque seconde de 10 nœuds. Écrire un programme Python qui calcule le temps nécessaire pour que l'avion atteigne sa vitesse de croisière de 340 nœuds.

III
Mais ensuite ?

3.1. Fonctions

Tout comme en maths, il est possible de définir des fonctions en Python. Contrairement à un programme tel qu'on a pu en écrire, une fonction est un morceau de code que l'on écrit une seule fois et qui peut être réutilisé à l'infini. Généralement, une fonction prend une variable en entrée, et lui fait subir une suite d'opérations avant de donner un résultat, mais il existe aussi des fonctions qui ne prennent rien en entrée, ou ne renvoient rien en sortie.

Par exemple, la fonction $f : x \mapsto 2x$ s'écrit en python :

```
def f(x):
  return 2*x
```

À noter qu'une fonction Python peut recevoir une infinité de variables en entrée (comme en maths), de tous types. Par exemple, la fonction Python correspondant au calcul de $f : x \mapsto ax + b$, avec a et b choisis par l'utilisateur, s'écrit :

```
def f(x, a, b):
  return a * x + b
```

Une fois la fonction écrite, il suffit de l'appeler avec les arguments qu'on veut pour obtenir le résultat. Par exemple, écrire `f(3, 2, 1)` dans un programme Python ou la console donnera le résultat 7 (car $2 \times 3 + 1 = 7$).

- **À vous !**

1. Compléter la fonction Python ci-dessous qui calcule la somme des nombres de 1 à n :

```
def somme(n):
  somme = ...
  for i in range(1, ...):
    somme = ...
  return somme
```

Appeler `somme(10)`. Que se passe-t-il ? Appeler alors `print(somme(10))`. Est-ce que ça marche ?

2. Recopier le code suivant, l'exécuter, et voir ce qu'il se passe :

```
def double(x):
   print(2 * x)
x = double(3)
print(x)
```

Quelle modification lui apporter pour qu'elle fonctionne ?

3. Compléter la fonction Python ci-dessous qui prend en entrée une liste de nombres et qui renvoie uniquement les nombres pairs de cette liste :

```
def nombres_pairs(liste):
    liste_pairs = ...
    for nombre in ...:
        if nombre ...:
            liste_pairs.append(nombre)
    return ...
```

4. Écrire une fonction Python `premiere_derniere_lettre(mot)`qui prend en entrée une chaîne de caractères, et qui renvoie uniquement sa première et dernière lettre. Par exemple, `premiere_derniere_lettre("bonjour")` donnera `"b...r"`.

IV
Et maintenant, on dessine !

4.1. Présentation du module `turtle`

4.2. À vous !

V
Quelques exercices pour occuper votre temps libre

On propose une liste d'exercices de programmation de différentes difficultés, permettant de combler vos éventuelles heures de permanence (ou votre temps libre à la maison). De nombreux éditeurs Python existent en ligne, vous permettant d'écrire votre code et de le faire tourner.

Plus un exercice comporte d'étoiles, plus il est difficile [2].

À vos claviers !

5.1. Table de 12

Proposer un programme Python qui affiche la table de multiplication de 12.

5.2. Table de n

Écrire une fonction Python qui prend en entrée un entier n, et qui affiche sa table de multiplication.

(*Indice : la fonction ne renvoie rien, elle affiche elle-même la table ligne par ligne*).

5.3. Le juste prix $\star$

On souhaite écrire en Python un jeu du juste prix. Les règles sont les suivantes :

- Un premier joueur choisit un nombre entre 1 et 100 ;
- Le second joueur doit deviner ce nombre. Pour cela, il fait une proposition à chaque tour :
  - Si le nombre qu'il propose est plus petit que le nombre choisi, programme lui indique qu'il est en-dessous,
  - Si le nombre qu'il propose est plus grand que le nombre choisi, programme lui indique qu'il est au-dessus,
  - Si le nombre qu'il propose est égal au nombre choisi, le jeu est terminé ;
- Si le joueur ne trouve pas le nombre au bout de 10 essais, il a perdu.

1. Écrire une fonction Python `situe_nombre(n, p)` qui situe un nombre p proposé par rapport au nombre n choisi. Il renverra soit `plus petit` si p est plus petit que n, soit `plus grand` si p est plus grand que n, soit `juste` si p est égal à n.
2. Proposer une boucle permettant de jouer jusqu'à ce que le joueur dépasse le nombre d'essais maximum.
3. Compléter alors, grâce aux questions précédentes, la fonction `juste_prix(n)` permettant de jouer au juste prix à partir d'un nombre n proposé par le premier joueur :

2. toutes proportions gardées bien sûr, ils sont tous faisables !

```
n = int(input(...))
def juste_prix(n):
    nombre_choisi = ...
    nombre_propose = ...
    essai = ...
    while ... and essai <= 10:
        print(situe_nombre(...))
        essai = ...
        nombre_propose = ...
    if essai > 10:
        print(...)
    else:
        print("Gagné !")
    print("Partie terminée.")
```

5.4. Trouveur de code ⋆

On souhaite écrire une fonction Python permettant de trouver un code à 4 chiffres. Pour cela, ce programe essaiera toutes les combinaisons possibles jusqu'à trouver la bonne.

1. Proposer une manière de générer une seule combinaison de 4 chiffres à partir d'opérations élémentaires (`+`, `*`).
2. Proposer une manière de parcourir tous les nombres de 0 à 9.
3. En déduire un empilement de boucles permettant de parcourir toutes les combinaisons à 4 chiffres, de `0000` à `9999`.
4. Finalement, combiner tout ça dans une fonction `trouve_code(n)` qui, à partir d'un code n à 4 chiffres, essaie toutes les combinaisons jusqu'à trouver le bon code. Elle renverra la combinaison qu'elle a trouvée. Vérifier alors si cette combinaison correspond bien au code de départ.
5. Pour les matheuses : combien de combinaisons possibles pour un code à 4 chiffres ? À 6 chiffres ? S'il faut une seconde pour essayer 10 combinaisons, combien de temps faut-il au plus pour trouver le code à 4 chiffres d'un iPhone ? Même question pour un code à 6 chiffres. Conclure.

5.5. Nombres premiers ⋆⋆

On souhaite écrire une fonction Python qui donne la liste de tous les nombres premiers inférieurs à un certain nombre. Pour cela, on se souvient qu'un nombre premier est un entier $n > 1$ ayant pour seuls diviseurs lui-même et 1. Par exemple, 3 est premier car il n'a que 1 et 3 comme diviseurs, tandis que 4 ne l'est pas car il a 1, 2 et 4 comme diviseurs.

1. Rappeler une manière en Python de parcourir tous les nombres de 1 à n.

2. On rappelle qu'un entier n est divisible par $k \leq n$ si le reste de la division entière de n par k est égal à 0. Rappeler comment accéder à ce résultat avec Python.
3. Écrire alors une fonction `est_premier(n)` qui renverra `True` si n est premier, `False` sinon.
4. Compléter alors la fonction `premiers(n)` qui renvoie la liste des nombres premiers inférieurs à n :

```
def premiers(n):
    liste = ...
    for ...:
        if est_premier(...):
            liste.append(...)
    return ...
```

5.6. Les pyramides ⋆⋆

On souhaite écrire une fonction Python `pyramide(n)` qui, à partir d'un entier n, renvoie la pyramide de nombres de hauteur n. Par exemple, `pyramide(5)` renverra la pydamide suivante :

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Proposer une solution pour la fonction `pyramide(n)` en utilisant deux boucles imbriquées.

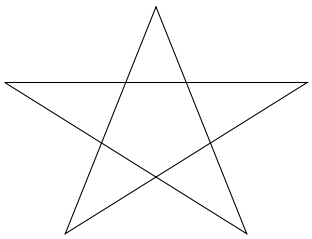
5.7. Les pyramides, le retour ⋆⋆⋆

De la même manière que pour la pyramide simple, proposer une fonction `pyramide_retour(n)` qui, à partir d'un entier n, renvoie une pyramide de hauteur n, qui monte puis redescend. Par exemple, `pyramide_retour(5)` renverra la pydamide suivante :

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

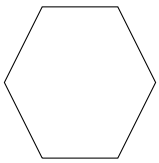
5.8. Si on dessinait ?

On souhaite, à l'aide du module `turtle`, dessiner la figure suivante (ou au moins une qui lui ressemble) :



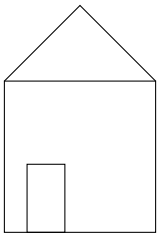
5.9. Hexagone

Proposer un programme Python qui, grâce au module `turtle`, dessine un hexagone de taille quelconque comme celui-ci :



5.10. Maison

Dessiner, grâce au module `turtle`, une maison qui ressemble (plus ou moins) à celle-ci :



On pourra également lui ajouter une cheminée (et pourquoi pas une fenêtre) ?