Chapitre 2

Devine mon nombre !

2.1. Thèmes abordés dans ce chapitre

- commentaires
- modules externes, import
- variables
- boucle `while`
- condition : `if... elif.... else`
- la fonction de conversion `int`
- `input()`
- exceptions

2.2. Règles du jeu

Ce jeu est très simple. L'ordinateur tire un nombre au hasard entre 1 et 30 et vous avez cinq essais pour le trouver. Après chaque tentative, l'ordinateur vous dira si le nombre que vous avez proposé est trop grand, trop petit, ou si vous avez trouvé le bon nombre.

Exemple de partie

```
J'ai choisi un nombre entre 1 et 30
A vous de le deviner en 5 tentatives au maximum !
Essai no  1
Votre proposition : 15
Trop petit
Essai no  2
Votre proposition : 22
Trop grand
Essai no  3
Votre proposition : 17
Trop grand
Essai no  4
Votre proposition : 16
Bravo ! Vous avez trouvé 16 en 4 essais
```

Remarque : les nombres en gras ont été entrés au clavier par le joueur.

2.3. Code du programme

devine.py

```
# Devine mon nombre

from random import randint

nbr_essais_max = 5
nbr_essais = 1
borne_sup = 30
mon_nombre = randint(1,borne_sup)     # nombre choisi par l'ordinateur
ton_nombre = 0                        # nombre proposé par le joueur

print("J'ai choisi un nombre entre 1 et",borne_sup)
print("A vous de le deviner en",nbr_essais_max,"tentatives au maximum !")

while ton_nombre != mon_nombre and nbr_essais <= nbr_essais_max:
    print("Essai no ",nbr_essais)
    ton_nombre = int(input("Votre proposition : "))
    if ton_nombre < mon_nombre:
        print("Trop petit")
    elif ton_nombre > mon_nombre:
        print("Trop grand")
    else:
        print("Bravo ! Vous avez trouvé",mon_nombre,"en",nbr_essais,"essai(s)")
    nbr_essais += 1

if nbr_essais>nbr_essais_max and ton_nombre != mon_nombre :
    print("Désolé, vous avez utilisé vos",nbr_essais_max,"essais en vain.")
    print("J'avais choisi le nombre",mon_nombre,".")
```

2.4. Analyse du programme

Reprenons ce programme ligne par ligne pour l'expliquer en détails.

2.4.1. Commentaires

```
# Devine mon nombre
```

Ceci est un commentaire. Les commentaires n'ont pas d'influence sur le programme lui-même ; ils sont là pour aider à la lecture et à la compréhension du code.

Règle 1

Le commentaire ne doit pas être redondant avec le code. Inutile de commenter des choses évidentes ! D'une manière générale, mieux le code est écrit, moins il y aura besoin de commentaires.

Règle 2

Pour déterminer ce qu'il faut indiquer dans le commentaire, se poser la question « pourquoi ? » et non pas « comment ? ». En effet, on arrivera souvent à comprendre ce que fait une fonction sans commentaires, mais on ne verra pas toujours son utilité.

2.4.2. Variables

```
nbr_essais_max = 5
nbr_essais = 1
borne_sup = 30
mon_nombre = randint(1,borne_sup)     # nombre choisi par l'ordinateur
ton_nombre = 0                        # nombre proposé par le joueur
```

Nous avons ici cinq variables qu'il faut initialiser. Cela signifie qu'il faut leur donner une valeur de départ. Si on ne le fait pas, l'interpréteur Python va envoyer le message d'erreur du genre :

NameError: name 'nbr_essais_max' is not defined

C'est au moment où l'on initialise une variable que l'interpréteur Python la crée. On peut voir une variable comme une boîte qui va contenir une valeur : ce peut être un nombre, une chaîne de caractères, une liste, etc. Écrire `nbr_essais = 1` a pour effet de déposer dans cette boîte la valeur entière `1`. On ne pourra pas mettre autre chose que des nombres entiers dans cette variable par la suite.

Dans la variable `mon_nombre` va être stockée une valeur aléatoire entière, qui changera à chaque exécution du programme. Il est à noter que si l'on avait omis la ligne

```
from random import randint
```

l'interpréteur Python aurait écrit le message d'erreur : NameError: name 'randint' is not defined

```
print("J'ai choisi un nombre entre 1 et",borne_sup)
print("A vous de le deviner en",nbr_essais_max,"tentatives au maximum !")
```

Ces deux lignes écrivent à l'écran le texte entre guillemets, ainsi que les valeurs contenues dans les variables `borne_sup` et `nbr_essais_max`. En l'occurrence, on verra s'écrire sur l'écran :

```
J'ai choisi un nombre entre 1 et 30
A vous de le deviner en 5 tentatives au maximum !
```

Règles pour les noms des variables

Le nom d'une variable est composé des lettres de a à z, de A à Z, et des chiffres 0 à 9, mais il ne doit pas commencer par un chiffre.

Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que $, #, @, etc. sont interdits, à l'exception du caractère _ (souligné). Le tiret (-) est bien sûr interdit puisqu'il correspond aussi à la soustraction.

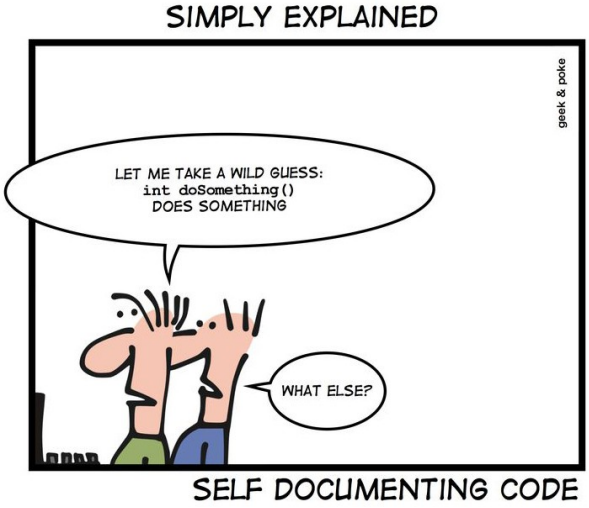
La casse est significative : `spam` et `Spam` sont des variables différentes !

Python compte 33 mots réservés qui ne peuvent pas non plus être utilisés comme noms de variable (ils sont utilisés par le langage lui-même) :

`and`	`as`	`assert`	`break`	`class`	`continue`	`def`
`del`	`elif`	`else`	`except`	`False`	`finally`	`for`
`from`	`global`	`if`	`import`	`in`	`is`	`lambda`
`None`	`nonlocal`	`not`	`or`	`pass`	`raise`	`return`
`True`	`try`	`while`	`with`	`yield`		

Prenez l'habitude d'écrire l'essentiel des noms de variables en caractères minuscules (y compris la première lettre). Il s'agit d'une convention largement respectée. N'utilisez les majuscules ou les soulignés qu'à l'intérieur du nom, pour en augmenter la lisibilité. Par exemple : *finDeMot* ou *fin_de_mot*.

Utilisez des noms de variable qui ont un sens afin d'augmenter la compréhension du programme. Cela vous évitera d'ajouter des commentaires pour expliquer l'utilité de ces variables.



Affectations

En Python, on peut assigner une valeur à plusieurs variables simultanément. Exemple :

```
a = b = 3
```

On peut aussi effectuer des affectations *parallèles* à l'aide d'un seul opérateur :

```
a, b = 3, 2.54
```

Dans cet exemple, les variables `a` et `b` prennent simultanément les nouvelles valeurs 3 et 2.54. Cela est particulièrement utile quand on veut échanger les valeurs de deux variables. Il suffit d'écrire :

```
a, b = b, a
```

Comme les affectations sont simultanées, les nouvelles valeurs de `a` et `b` seront respectivement 2.54 et 3.

Notons enfin au passage qu'une instruction du type :

```
a + 1 = 3
```

est tout à fait illégale !

Opérations sur les variables entières

Dans notre programme, toutes les variables sont du type entier. Les opérations que l'on peut faire avec les entiers sont les suivantes :

Les priorités sont les mêmes que sur une calculatrice standard. On peut utiliser des parenthèses pour changer les priorités.

Symbole	Nom	Exemple	Résultat
+	Addition	3+4	7
–	Soustraction	8–3	5
*	Multiplication	5*2	10
//	Division entière	14//3	4
%	Reste de la division entière	14%3	2
/	Division	14/3	4.666...
**	Élévation à la puissance	3**4	81

Tableau 2.1: opérateurs sur les nombres entiers

2.4.3. Boucle while (tant que)

```
while ton_nombre != mon_nombre and nbr_essais <= nbr_essais_max:
    print("Essai no ",nbr_essais)
    ton_nombre = int(input("Votre proposition : "))
    if ton_nombre < mon_nombre:
        print("Trop petit")
    elif ton_nombre > mon_nombre:
        print("Trop grand")
    else:
        print("Bravo ! Vous avez trouvé",mon_nombre,"en",nbr_essais,"essai(s)")
    nbr_essais += 1
```

En informatique, on dit *indenté* plutôt que *décalé à droite*.

Voici une *boucle Tant que*. Tant que la valeur stockée dans `nom_nombre` sera différente de la valeur stockée dans `ton_nombre` et que le nombre d'essais effectués sera inférieur ou égal au nombre d'essais maximum, alors toute la partie du code qui est indentée vers la droite sera exécutée en boucle.

Symbole	Nom	Exemple	Résultat
==	égal	3 == 3	True
!=	différent	3 != 4	True
>	supérieur	5 > 5	False
>=	supérieur ou égal	5 >= 5	True
<	inférieur	6 < 2	False
<=	inférieur ou égal	0 <= 1	True

Tableau 2.2: opérateurs de comparaison

2.4.4. Incrémentation

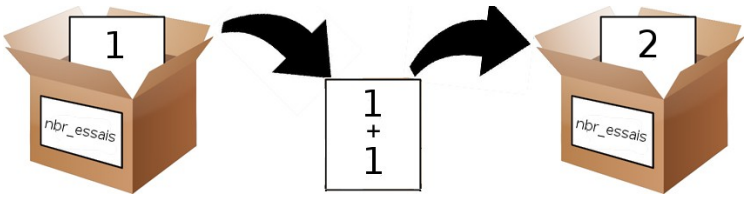
L'incrémentation est une des instructions les plus déroutantes pour un débutant :

```
nbr_essais += 1
```

On peut aussi écrire cette instruction ainsi, sans que cela soit beaucoup plus clair :

```
nbr_essais = nbr_essais + 1
```

Voici ce qui se passe. On prend la valeur de la variable `nbr_essais`, on y ajoute 1, puis on remet le résultat dans la variable `nbr_essais`. Donc, si `nbr_essais` avait la valeur 1, après l'instruction `nbr_essais += 1` il aura la valeur 2.



Beaucoup de boucles se terminent par une incrémentation, comme dans notre programme. Faute de quoi, la condition d'arrêt (ici `nbr_essais <= nbr_essais_max`) ne sera jamais satisfaite, ce qui provoquera une boucle infinie. C'est une des erreurs les plus courantes en programmation.

Exercice 2.1

Écrivez un programme qui demande à l'utilisateur d'écrire des mots.
Tant que l'utilisateur n'aura pas pressé sur la touche 'Enter' toute seule, l'ordinateur demandera un nouveau mot. Le programme écrira ce mot, précédé de numéro de rang.
Poliment, l'ordinateur écrira « Au revoir » avant que le programme se termine.

2.4.5. Boucles imbriquées

Il est tout à fait possible, et parfois nécessaire, d'imbriquer une boucle dans une autre boucle.
Par exemple, imaginons que l'on veuille écrire toutes les heures et minutes d'une journée. Cela commencera par 0 heure 0 minute, 0 heure 1 minute, 0 heure 2 minutes, ..., 0 heure 59 minutes, 1 heure 0 minute, 1 heure 1 minute, ...
On voit qu'une première boucle parcourra les heures, tandis que la deuxième parcourra les minutes, et que l'on incrémentera l'heure seulement quand 60 minutes seront passées.

```
h=0
while h<24:
    m=0
    while m<60:
        print(h,"heure(s) ",m,"minute(s)")
        m+=1
    h+=1
```

Exercice 2.2

Ajoutez une boucle au programme ci-dessus pour écrire les secondes.

Exercice 2.3

Partie 1

Écrivez un programme qui affiche les 20 premiers termes de la table de multiplication par 53. Le résultat commencera comme ceci :

```
1 x 53 =   53
2 x 53 =  106
3 x 53 =  159
```

Aide : Il existe une fonction de formatage qui permet d'aligner joliment les nombres en colonne. Par exemple,

```
print('{:4d}'.format(a))
```

écrira l'entier `a` sur 4 espaces et le calera sur la droite.
La page de référence pour le formatage est :
http://docs.python.org/py3k/library/string.html#string-formatting

Partie 2

Modifiez la partie 1 pour obtenir un programme qui affiche toutes les tables de multiplication de 2 à 12. Chaque table comprendra 12 lignes. Alignez joliment les nombres dans les tables.

2.4.6. Conversion de types

```
ton_nombre = int(input("Votre proposition : "))
```

La fonction `input` permet à l'utilisateur d'entrer une chaîne de caractères (*string* en anglais) au clavier. En écrivant `int(input())`, on transforme le type string en un type entier. La variable `ton_nombre` contiendra donc un nombre entier, qui pourra être utiliser comme tel.

Exercice 2.4 : livret

Écrivez un programme Python pour tester votre livret. Le programme demandera d'abord quel est le plus grand nombre (Nmax) qu'il pourra utiliser. Puis il proposera 10 questions de livret avec deux nombres tirés au sort dans l'intervalle [2, Nmax].

En cas de réponse erronée, le programme reposera la même question, tant que la bonne réponse n'aura pas été donnée.

Exemple d'une partie

```
nombre maximum ? 15

6 x 12 = 72
14 x 7 = 98
8 x 9 = 72
5 x 12 = 60
13 x 14 = 183
Faux ! Réessayez !
13 x 14 = 192
Faux ! Réessayez !
13 x 14 = 182
4 x 5 = 20
9 x 7 = 63
8 x 8 = 64
12 x 14 = 168
10 x 12 = 120
```

2.4.7. Conditions

```
if ton_nombre < mon_nombre:
    print("Trop petit")
elif ton_nombre > mon_nombre:
    print("Trop grand")
else:
    print("Bravo ! Vous avez trouvé",mon_nombre,"en",nbr_essais,"essais")
```

Quand le joueur propose un nombre, il y a trois possibilités : soit son nombre est trop petit, soit il est trop grand, soit c'est le bon nombre. Ces trois possibilités correspondront à trois réponses différentes de l'ordinateur.

Cela se traduira en Python par l'utilisation des instructions `if... elif... else...` On aurait pu écrire plusieurs instructions au lieu d'une. Il aurait suffi de garder le même décalage. Par exemple :

```
if ton_nombre < mon_nombre:
    print("Trop petit")
    print("Dommage!")
```

Exercice 2.5 : livret sous pression du temps

Améliorez le programme de l'exercice 2.4 : à la fin de la partie, affichez le nombre d'erreurs (avec la bonne orthographe) et le temps utilisé pour répondre aux dix questions.

Aide : dans le module `time` se trouve la fonction `time()`, qui donne le nombre de secondes écoulées depuis le 1er janvier 1970 à 00:00:00.

Exercice 2.6 : calcul mental

Améliorez le programme de l'exercice 2.5. Cette fois-ci, on ne veut pas se contenter d'exercer les multiplications, mais aussi l'addition, la soustraction et la division entière. L'opération à tester sera tirée au sort pour chaque question.

Exercice 2.7

Écrivez un programme qui simule 1000 lancers d'une pièce de monnaie. Vous afficherez seulement le nombre de piles et le nombre de faces obtenus.

Écrivez une version avec une boucle `for` (voir chapitre 1) et une autre avec une boucle `while`.

Exercice 2.8 : les paquets de cartes Hearthstone

Améliorez le programme de l'exercice 1.8.
Générez *n* paquets de cartes ***Hearthstone*** valides.
Comptez le nombre total de cartes de chaque rareté afin de vérifier que les pourcentages obtenus sont « proches » des pourcentages théoriques (plus vous générerez de paquets, plus les pourcentages seront proches de la théorie).

Exemple de sortie avec 10 paquets (50 cartes) :

```
R C R C C
C R C C C
C R C C L !
C R R C C
C C C C C non valide
C C C R C
C C R R C
R C C L C !
C C C C C non valide
C C C R C
R C C C C
E C E C C

2 légendaire(s): 4.00 %
2 épique(s): 4.00 %
12 rare(s): 24.00 %
34 communes: 68.00 %
```

Vous ajouterez un point d'exclamation après les paquets contenant au moins une légendaire.

Exercice 2.9

Modifiez le code du § 2.3 :

1. Avant de commencer le jeu, le programme demandera le prénom du joueur. Quand René trouvera le bon nombre, le programme le félicitera par la phrase :
   `Bravo, René ! Vous avez trouvé 16 en 4 essai(s).`
2. À la fin de la partie, le programme proposera une nouvelle partie au joueur, qui répondra par oui ou non. Quand le joueur arrêtera de jouer, le programme indiquera le pourcentage de réussite et le nombre moyen de tentatives pour trouver le nombre (on ne comptabilisera le nombre de coups qu'en cas de réussite).

Exemple de partie

```
Quel est votre prénom ? René
J'ai choisi un nombre entre 1 et 30
A vous de le deviner en 5 tentatives au maximum !
Essai no  1
Votre proposition : 15
Bravo, René ! Vous avez trouvé 15 en 1 essai(s)
Voulez-vous rejouer (o/n) ? o
J'ai choisi un nombre entre 1 et 30
A vous de le deviner en 5 tentatives au maximum !
Essai no  1
Votre proposition : 1
Trop petit
Essai no  2
Votre proposition : 9
Trop petit
Essai no  3
Votre proposition : 14
Trop petit
Essai no  4
Votre proposition : 19
```

```
Trop petit
Essai no  5
Votre proposition : 21
Trop petit
Désolé, vous avez utilisé vos 5 essais en vain.
J'avais choisi le nombre 25 .
Voulez-vous rejouer (o/n) ? n
Pourcentage de réussite: 50.0 %
Nombre moyen de tentatives: 1.0
```

2.5. Exceptions

devine-try.py

Si l'utilisateur entre autre chose qu'un nombre entier, le programme va générer une erreur et s'arrêter. Pour éviter cela, il faut remplacer la ligne :

```
ton_nombre = int(input("Votre proposition : "))
```

par

```
while True:
    try:
        ton_nombre = int(input("Votre proposition : "))
        break
    except ValueError:
        print("Réponse non valide. Réessayez !")
```

C'est une boucle infinie (`while True`) dont le seul moyen de sortir (`break`) est d'entrer un nombre entier. Tant que l'exception `ValueError` est levée, un message d'erreur créé par l'utilisateur est affiché à l'écran et le programme repose la question.

Exercice 2.10

Inversons les rôles ! C'est l'ordinateur qui essaiera de deviner le nombre que vous avez en tête et vous lui indiquerez si le nombre qu'il proposera est trop petit ou trop grand.

Convention : entrez 1 si le nombre de l'ordinateur est trop grand, 2 s'il est trop petit et 0 si l'ordinateur a trouvé le bon nombre.

Prévoyez les cas où l'utilisateur écrit autre chose qu'une des trois réponses attendues.

Exercice 2.11 : Risk

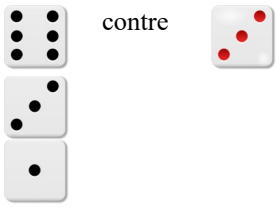
Au jeu « Risk », les combats se déroulent ainsi : l'assaillant désigne par leurs noms le territoire visé et le territoire attaquant. L'assaillant prend trois dés de même couleur (rose). Il lance autant de dés qu'il engage d'armées (3 au maximum).

Exemple : Un assaillant a 4 armées sur l'Ontario. Il attaque le Québec avec maximum 3 armées à chaque coup de dés mais il peut attaquer, s'il le désire, avec seulement 1 ou 2 armées en jetant 1 ou 2 dés. Le défenseur, lui, ne peut lancer que 2 dés au maximum, même s'il a 3 armées ou plus sur son territoire.

On compare séparément chacun de ses dés avec ceux de l'adversaire en commençant par le plus fort de chaque côté. Les dés les plus forts gagnent. Mais, en cas d'égalité de points, le défenseur l'emporte, même s'il lance moins de dés.

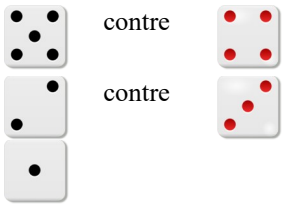
Voici plusieurs configurations de dés. L'assaillant a les dés noirs et le défenseur les dés rouges. L'assaillant peut attaquer avec 1, 2 ou 3 dés; le défenseur peut utiliser 1 ou 2 dés.

Exemple 1



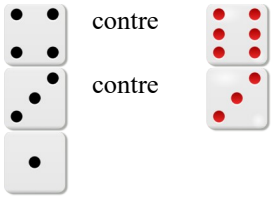
Le défenseur perd 1 armée.

Exemple 2



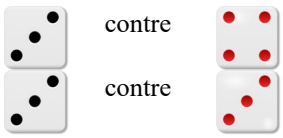
Chaque adversaire perd 1 armée.

Exemple 3



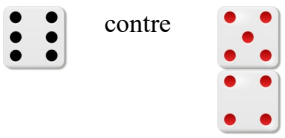
À égalité des points, le défenseur l'emporte. L'assaillant perd 2 armées.

Exemple 4



L'assaillant perd 2 armées.

Exemple 5



Le défenseur perd 1 armée.

Calculez à l'aide d'un programme et notez dans le tableau ci-après la probabilité de chacun des événements y figurant, en fonction du nombre de dés que chacun des deux joueurs choisit de lancer.

		Nombre de dés lancés par le défenseur	
		2	**1**
Nombre de dés lancés par l'assaillant	**3**	L'assaillant perd 2 armées : % Le défenseur perd 2 armées : % Chaque joueur perd 1 armée : %	L'assaillant perd 1 armée : % Le défenseur perd 1 armée : %
	2	L'assaillant perd 2 armées : % Le défenseur perd 2 armées : % Chaque joueur perd 1 armée : %	L'assaillant perd 1 armée : % Le défenseur perd 1 armée : %
	1	L'assaillant perd 1 armée : % Le défenseur perd 1 armée : %	L'assaillant perd 1 armée : % Le défenseur perd 1 armée : %

2.6. Ce que vous avez appris dans ce chapitre

- On peut écrire dans le code d'un programme des commentaires qui aideront à sa lisibilité. Il faut savoir les utiliser avec intelligence (§ 2.4.1). Un bon programmeur écrit des programmes lisibles !
- Vous savez tout sur les variables (§ 2.4.2) et comment s'en servir : noms autorisés, initialisation, affectation, incrémentation (§ 2.4.4).
- Il y plusieurs genres d'affectation en Python : simples, multiples, simultanées (§ 2.4.2).
- En plus de la boucle `for` vue rapidement au chapitre 1, il existe une boucle `while` (§ 2.4.3). On utilise plutôt la boucle `for` quand on sait d'avance combien d'itérations on devra effectuer (par exemple, lancer 100 fois un dé), mais une boucle `while` fait aussi l'affaire. La boucle `while` s'utilise quand on ne sait pas au départ le nombre d'itérations.
- On peut imbriquer des boucles.
- Il y a différents types de données : les chaînes de caractères, les entiers, les réels, les booléens.
- La fonction `input()` retourne toujours une chaîne de caractères (§ 2.4.6). On peut convertir cette chaîne de caractères en un entier (avec `int`) ou en un réel (avec `float`) .
- Le tableau 2.1 résume toutes les opérations possibles avec les entiers et le tableau 2.2 les opérateurs de comparaison.
- Les exceptions sont utiles pour éviter des erreurs qui pourraient provoquer l'arrêt du programme (§ 2.5).
- Il existe une fonction de formatage qui permet d'aligner joliment les nombres en colonne (exercice 2.3).