6

Fonctions prédéfinies

L'un des concepts les plus importants en programmation est celui de fonction[24]*. Les fonctions permettent en effet de décomposer un programme complexe en une série de sous-programmes plus simples, lesquels peuvent à leur tour être décomposés en fragments plus petits, et ainsi de suite. D'autre part, les fonctions sont réutilisables : si nous disposons d'une fonction capable de calculer une racine carrée, par exemple, nous pouvons l'utiliser un peu partout dans nos programmes sans avoir à la ré-écrire à chaque fois.*

Affichage : la fonction print()

Nous avons bien évidemment déjà rencontré cette fonction. Précisons simplement ici qu'elle permet d'afficher n'importe quel nombre de valeurs fournies en arguments (c'est-à-dire entre les parenthèses). Par défaut, ces valeurs seront séparées les unes des autres par un espace, et le tout se terminera par un saut à la ligne.

Vous pouvez remplacer le séparateur par défaut (l'espace) par un autre caractère quelconque (ou même par aucun caractère), grâce à l'argument « sep ». Exemple :

```
>>> print("Bonjour", "à", "tous", sep ="*")
Bonjour*à*tous
 >>> print("Bonjour", "à", "tous", sep ="")
Bonjouràtous
```

De même, vous pouvez remplacer le saut à la ligne terminal avec l'argument « end » :

```
>>> n =0
>>> while n<6:
...     print("zut", end ="")
...     n = n+1
...
zutzutzutzutzut
```

[24]Sous Python, le terme « fonction » est utilisé indifféremment pour désigner à la fois de véritables fonctions mais également des **procédures**. Nous indiquerons plus loin la distinction entre ces deux concepts proches.

Interaction avec l'utilisateur : la fonction input()

La plupart des scripts élaborés nécessitent à un moment ou l'autre une intervention de l'utilisateur (entrée d'un paramètre, clic de souris sur un bouton, etc.). Dans un script en mode texte (comme ceux que nous avons créés jusqu'à présent), la méthode la plus simple consiste à employer la fonction intégrée **input()**. Cette fonction provoque une interruption dans le programme courant. L'utilisateur est invité à entrer des caractères au clavier et à terminer avec <*Enter*>. Lorsque cette touche est enfoncée, l'exécution du programme se poursuit, et la fonction fournit en retour une chaîne de caractères correspondant à ce que l'utilisateur a entré. Cette chaîne peut alors être assignée à une variable quelconque, convertie, etc.

On peut invoquer la fonction **input()** en laissant les parenthèses vides. On peut aussi y placer en argument un message explicatif destiné à l'utilisateur. Exemple :

```
prenom = input("Entrez votre prénom : ")
print("Bonjour,", prenom)
```

ou encore :

```
print("Veuillez entrer un nombre positif quelconque : ", end=" ")
ch = input()
nn = int(ch)              # conversion de la chaîne en un nombre entier
print("Le carré de", nn, "vaut", nn**2)
```

Remarque importante

- La fonction **input()** renvoie toujours une chaîne de caractères[25]. Si vous souhaitez que l'utilisateur entre une valeur numérique, vous devrez donc convertir la valeur entrée (qui sera donc de toute façon de type string) en une valeur numérique du type qui vous convient, par l'intermédiaire des fonctions intégrées **int()** (si vous attendez un entier) ou **float()** (si vous attendez un réel). Exemple :

```
>>> a = input("Entrez une donnée numérique : ")
Entrez une donnée numérique : 52.37
>>> type(a)
<class 'str'>
>>> b = float(a)      # conversion de la chaîne en un nombre réel
>>> type(b)
<class 'float'>
```

Importer un module de fonctions

Vous avez déjà rencontré des *fonctions intégrées* au langage lui-même, comme la fonction **len()**, par exemple, qui permet de connaître la longueur d'une chaîne de caractères. Il va de soi cependant qu'il n'est pas possible d'intégrer toutes les fonctions imaginables dans le corps standard de Python, car il en existe virtuellement une infinité : vous apprendrez d'ailleurs très bientôt comment en créer vous-même de nouvelles. Les fonctions intégrées au langage sont relative-

[25] Dans les versions de Python antérieures à la version 3.0, la valeur renvoyée par input() était de type variable, suivant ce que l'utilisateur avait entré. Le comportement actuel est en fait celui de l'ancienne fonction raw_input(), que lui préféraient la plupart des programmeurs.

ment peu nombreuses : ce sont seulement celles qui sont susceptibles d'être utilisées très fréquemment. Les autres sont regroupées dans des fichiers séparés que l'on appelle des *modules* .

> *Les modules sont des fichiers qui regroupent des ensembles de fonctions*[26].

Vous verrez plus loin combien il est commode de découper un programme important en plusieurs fichiers de taille modeste pour en faciliter la maintenance. Une application Python typique sera alors constituée d'un programme principal, accompagné de un ou plusieurs modules contenant chacun les définitions d'un certain nombre de fonctions accessoires.

Il existe un grand nombre de modules pré-programmés qui sont fournis d'office avec Python. Vous pouvez en trouver d'autres chez divers fournisseurs. Souvent on essaie de regrouper dans un même module des ensembles de fonctions apparentées, que l'on appelle des *bibliothèques*.

Le module *math*, par exemple, contient les définitions de nombreuses fonctions mathématiques telles que *sinus*, *cosinus*, *tangente*, *racine carrée*, etc. Pour pouvoir utiliser ces fonctions, il vous suffit d'incorporer la ligne suivante au début de votre script :

```
from math import *
```

Cette ligne indique à Python qu'il lui faut inclure dans le programme courant *toutes* les fonctions (c'est là la signification du symbole « joker » *) du module *math*, lequel contient une bibliothèque de fonctions mathématiques pré-programmées.

Dans le corps du script lui-même, vous écrirez par exemple :
`racine = sqrt(nombre)` pour assigner à la variable `racine` la racine carrée de `nombre`,
`sinusx = sin(angle)` pour assigner à la variable `sinusx` le sinus de `angle` (en radians !), etc.

Exemple :

```
# Démo : utilisation des fonctions du module <math>

from math import *

nombre = 121
angle = pi/6          # soit 30°
                      # (la bibliothèque math inclut aussi la définition de pi)
print("racine carrée de", nombre, "=", sqrt(nombre))
print("sinus de", angle, "radians", "=", sin(angle))
```

L'exécution de ce script provoque l'affichage suivant :

```
racine carrée de 121 = 11.0
sinus de 0.523598775598 radians = 0.5
```

Ce court exemple illustre déjà fort bien quelques caractéristiques importantes des fonctions :

- une fonction apparaît sous la forme d'*un nom quelconque associé à des parenthèses*
  exemple : **sqrt()**
- dans les parenthèses, on *transmet* à la fonction un ou plusieurs *arguments*
  exemple : **sqrt(121)**

[26] En toute rigueur, un module peut contenir aussi des définitions de variables ainsi que des *classes*. Nous pouvons toutefois laisser ces précisions de côté, provisoirement.

- la fonction fournit une *valeur de retour* (on dira aussi qu'elle « retourne », ou mieux : qu'elle « renvoie » une valeur)
  exemple : **11.0**

Nous allons développer tout ceci dans les pages suivantes. Veuillez noter au passage que les fonctions mathématiques utilisées ici ne représentent qu'un tout premier exemple. Un simple coup d'œil dans la documentation des bibliothèques Python vous permettra de constater que de très nombreuses fonctions sont d'ores et déjà disponibles pour réaliser une multitude de tâches, y compris des algorithmes mathématiques très complexes (Python est couramment utilisé dans les universités pour la résolution de problèmes scientifiques de haut niveau). Il est donc hors de question de fournir ici une liste détaillée. Une telle liste est aisément accessible dans le système d'aide de Python :

Documentation HTML → *Python documentation* → *Modules index* → *math*

Au chapitre suivant, nous apprendrons comment créer nous-mêmes de nouvelles fonctions.

Exercices

> *Note : dans tous ces exercices, utilisez la fonction* `input()` *pour l'entrée des données.*

6.1 Écrivez un programme qui convertisse en mètres par seconde et en km/h une vitesse fournie par l'utilisateur en miles/heure. (*Rappel : 1 mile = 1609 mètres*)

6.2 Écrivez un programme qui calcule le périmètre et l'aire d'un triangle quelconque dont l'utilisateur fournit les 3 côtés.
(Rappel : l'aire d'un triangle quelconque se calcule à l'aide de la formule d'Héron :

$$S = \sqrt{d \cdot (d-a) \cdot (d-b) \cdot (d-c)}$$

dans laquelle d désigne la longueur du demi-périmètre, et a, b, c celles des trois côtés.)

6.3 Écrivez un programme qui calcule la période d'un pendule simple de longueur donnée.

La formule qui permet de calculer la période d'un pendule simple est $T = 2\pi\sqrt{\frac{l}{g}}$,

l représentant la longueur du pendule et ***g*** la valeur de l'accélération de la pesanteur au lieu d'expérience.

6.4 Écrivez un programme qui permette d'encoder des valeurs dans une liste. Ce programme devrait fonctionner en boucle, l'utilisateur étant invité à entrer sans cesse de nouvelles valeurs, jusqu'à ce qu'il décide de terminer en frappant <*Enter*> en guise d'entrée. Le programme se terminerait alors par l'affichage de la liste. Exemple de fonctionnement :

```
Veuillez entrer une valeur : 25
Veuillez entrer une valeur : 18
Veuillez entrer une valeur : 6284
Veuillez entrer une valeur :
[25, 18, 6284]
```

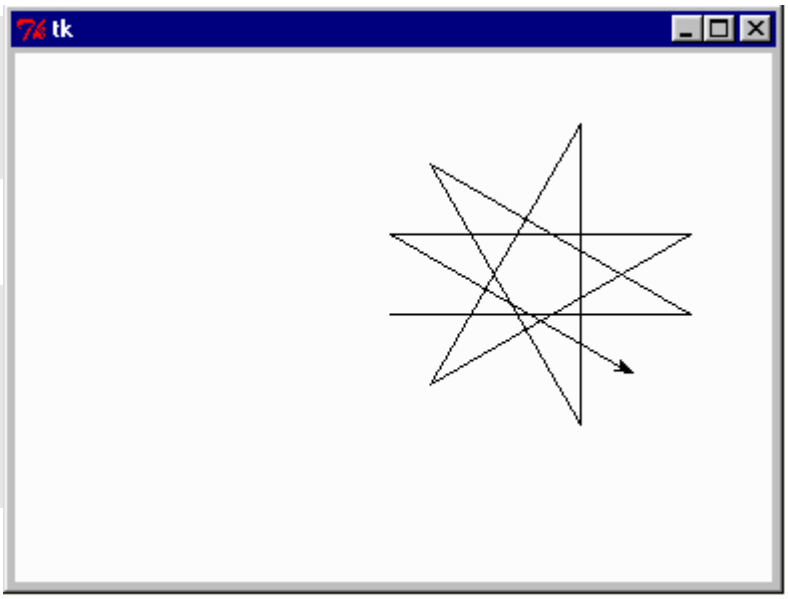
Un peu de détente avec le module turtle

Comme nous venons de le voir, l'une des grandes qualités de Python est qu'il est extrêmement facile de lui ajouter de nombreuses fonctionnalités par importation de divers *modules* .

Pour illustrer cela, et nous amuser un peu avec d'autres objets que des nombres, nous allons explorer un module Python qui permet de réaliser des « graphiques tortue », c'est-à-dire des dessins géométriques correspondant à la piste laissée derrière elle par une petite « tortue » virtuelle, dont nous contrôlons les déplacements sur l'écran de l'ordinateur à l'aide d'instructions simples.

Activer cette tortue est un vrai jeu d'enfant. Plutôt que de vous donner de longues explications, nous vous invitons à essayer tout de suite :

```
>>> from turtle import *
>>> forward(120)
>>> left(90)
>>> color('red')
>>> forward(80)
```



L'exercice est évidemment plus riche si l'on utilise des boucles :

```
>>> reset()
>>> a = 0
>>> while a <12:
        a = a +1
        forward(150)
        left(150)
```

Attention cependant : avant de lancer un tel script, assurez-vous toujours qu'il ne comporte pas de boucle sans fin (voir page 29), car si c'est le cas vous risquez de ne plus pouvoir reprendre le contrôle des opérations (en particulier sous *Windows*).

Amusez-vous à écrire des scripts qui réalisent des dessins suivant un modèle imposé à l'avance. Les principales fonctions mises à votre disposition dans le module *turtle* sont les suivantes :

`reset()`	On efface tout et on recommence
`goto(x, y)`	Aller à l'endroit de coordonnées x, y
`forward(distance)`	Avancer d'une distance donnée
`backward(distance)`	Reculer
`up()`	Relever le crayon (pour pouvoir avancer sans dessiner)
`down()`	Abaisser le crayon (pour recommencer à dessiner)
`color(couleur)`	*couleur* peut être une chaîne prédéfinie ('red', 'blue', etc.)
`left(angle)`	Tourner à gauche d'un angle donné (exprimé en degrés)
`right(angle)`	Tourner à droite
`width(épaisseur)`	Choisir l'épaisseur du tracé
`fill(1)`	Remplir un contour fermé à l'aide de la couleur sélectionnée
`write(texte)`	Écrire un texte. *texte* doit être une chaîne de caractères

Il existe aussi la fonction **speed** qui permet de modifier la vitesse de la tortue. Du moins rapide au plus rapide : speed('slowest') - speed('slow') - speed('normal') - speed('fast') - speed('fastest').

Véracité/fausseté d'une expression

Lorsqu'un programme contient des instructions telles que **while** ou **if**, l'ordinateur qui exécute ce programme doit évaluer la véracité d'une condition, c'est-à-dire déterminer si une expression est *vraie* ou *fausse*. Par exemple, une boucle initiée par **while c<20:** s'exécutera aussi longtemps que la condition **c<20** restera *vraie*.

Mais comment un ordinateur peut-il déterminer si quelque chose est *vrai* ou *faux* ?

En fait – et vous le savez déjà – un ordinateur ne manipule strictement que des nombres. Tout ce qu'un ordinateur doit traiter doit d'abord toujours être converti en valeur numérique. Cela s'applique aussi à la notion de vrai/faux. En Python, tout comme en *C*, en *Basic* et en de nombreux autres langages de programmation, on considère que toute valeur numérique autre que zéro est « vraie ». Seule la valeur zéro est « fausse ». Exemple :

```
ch = input('Entrez un nombre entier quelconque')
n =int(ch)
if n:
    print("vrai")
else:
    print("faux")
```

Le petit script ci-dessus n'affiche « faux » que si vous entrez la valeur 0. Pour toute autre valeur numérique, vous obtiendrez « vrai ».

Ce qui précède signifie donc qu'une expression à évaluer, telle par exemple la condition **a > 5**, est d'abord convertie par l'ordinateur en une valeur numérique (**1** si l'expression est vraie, et **zéro** si l'expression est fausse). Ce n'est pas tout à fait évident, parce que l'interpréteur Python est doté d'un dispositif qui traduit ces deux valeurs en « True » ou « False » lorsqu'on les lui demande explicitement. Exemple :

```
>>> a, b = 3, 8
>>> c = (a < b)
>>> d = (a > b)
>>> c
True
>>> d
False
```

(L'expression **a < b** est évaluée, et son résultat (« vrai ») est mémorisé dans la variable **c**. De même pour le résultat de l'expression inverse, dans la variable **d**[27].)

À l'aide d'une petite astuce, nous pouvons quand même vérifier que ces valeurs **True** et **False** sont en réalité les deux nombres 1 et 0 « déguisés » :

```
>>> accord = ["non", "oui"]
>>> accord[d]
```

[27] Ces variables sont d'un type entier un peu particulier : le type « booléen ». Les variables de ce type ne peuvent prendre que les deux valeurs True et False (en réalité, 1 et 0).

```
non
>>> accord[c]
oui
```

(En utilisant les valeurs des variables **c** et **d** comme indices pour extraire les éléments de la liste **accord**, nous confirmons bien que False =0 et True =1.)

Le petit script ci-après est très similaire au précédent. Il nous permet de tester le caractère « vrai » ou « faux » d'une chaîne de caractères :

```
ch = input("Entrez une chaîne de caractères quelconque")
if ch:
    print("vrai")
else:
    print("faux")
```

Vous obtiendrez « faux » pour toute chaîne *vide* , et « vrai » pour toute chaîne contenant au moins un caractère. Vous pourriez de la même manière tester la « véracité » d'une liste, et constater qu'une liste vide est « fausse », alors qu'une liste ayant un contenu quelconque est « vraie »[28].

L'instruction `if ch:` , à la troisième ligne de cet exemple, est donc équivalente à une instruction du type : `if ch !="":` , du moins de notre point de vue d'êtres humains.

Pour l'ordinateur, cependant, ce n'est pas tout à fait pareil. Pour lui, l'instruction **if ch:** consiste à vérifier directement si la valeur de la variable **ch** est une chaîne vide ou non, comme nous venons de le voir. La seconde formulation, par contre : **if ch != "":** lui impose de commencer par comparer le contenu de **ch** *à la valeur d'une autre donnée que nous lui fournissons par notre programme* (une chaîne vide), puis à évaluer *ensuite* si le résultat de cette comparaison est lui-même « vrai » ou « faux » (ou en d'autres termes, à vérifier si ce résultat est lui-même « True » ou « False »). Cela lui demande donc deux opérations successives, là où la première formulation ne lui en demande qu'une seule. La première formulation est donc plus performante.

Pour les mêmes raisons, dans un script tel celui-ci :

```
ch =input("Veuillez entrer un nombre : ")
n =int(ch)
if n % 2:
     print("Il s'agit d'un nombre impair.")
else:
     print("Il s'agit d'un nombre pair.")
```

il est plus efficace de programmer la troisième ligne comme nous l'avons fait ci-dessus, plutôt que d'écrire explicitement : `if n % 2 != 0` , car cette formulation imposerait à l'ordinateur d'effectuer deux comparaisons successives au lieu d'une seule.

Ce raisonnement « proche de la machine » vous paraîtra probablement un peu subtil au début, mais croyez bien que cette forme d'écriture vous deviendra très vite tout à fait familière.

---

[28] Les autres structures de données se comportent d'une manière similaire. Les *tuples* et les *dictionnaires* que vous étudierez plus loin (au chapitre 10) sont également considérés comme « faux » lorsqu'ils sont vides, et « vrais » lorsqu'ils possèdent un contenu.

Révision

Dans ce qui suit, nous n'allons pas apprendre de nouveaux concepts mais simplement utiliser tout ce que nous connaissons déjà, pour réaliser de vrais petits programmes.

Contrôle du flux – utilisation d'une liste simple

Commençons par un petit retour sur les branchements conditionnels (il s'agit peut-être là du groupe d'instructions le plus important, dans n'importe quel langage !) :

```
# Utilisation d'une liste et de branchements conditionnels

print("Ce script recherche le plus grand de trois nombres")
print("Veuillez entrer trois nombres séparés par des virgules : ")
ch =input()
# Note : l'association des fonctions eval() et list() permet de convertir
# en liste toute chaîne de valeurs séparées par des virgules :[29]
nn = list(eval(ch))
max, index = nn[0], 'premier'
if nn[1] > max:                        # ne pas omettre le double point !
    max = nn[1]
    index = 'second'
if nn[2] > max:
    max = nn[2]
    index = 'troisième'
print("Le plus grand de ces nombres est", max)
print("Ce nombre est le", index, "de votre liste.")
```

Dans cet exercice, vous retrouvez à nouveau le concept de « bloc d'instructions », déjà abondamment commenté aux chapitres 3 et 4, et que vous devez absolument assimiler. Pour rappel, les blocs d'instructions sont délimités par *l'indentation*. Après la première instruction **if**, par exemple, il y a deux lignes indentées définissant un bloc d'instructions. Ces instructions ne seront exécutées que si la condition **nn[1] > max** est vraie.

La ligne suivante, par contre (celle qui contient la deuxième instruction **if**) n'est pas indentée. Cette ligne se situe donc au même niveau que celles qui définissent le corps principal du programme. L'instruction contenue dans cette ligne est donc toujours exécutée, alors que les deux suivantes (qui constituent encore un autre bloc) ne sont exécutées que si la condition **nn[2] > max** est vraie.

En suivant la même logique, on voit que les instructions des deux dernières lignes font partie du bloc principal et sont donc toujours exécutées.

Boucle while – instructions imbriquées

Continuons dans cette voie en imbriquant d'autres structures :

```
1#  # Instructions composées <while> - <if> - <elif> - <else>
2#
```

[29] En fait, la fonction **eval()** évalue le contenu de la chaîne fournie en argument comme étant une expression Python dont elle doit renvoyer le résultat. Par exemple : **eval("7 + 5")** renvoie l'entier **12**. Si on lui fournit une chaîne de valeurs séparées par des virgules, cela correspond pour elle à un ***tuple***. Les tuples sont des séquences apparentées aux listes. Ils seront abordés au chapitre 10 (cf. page 152).

```
 3#  print('Choisissez un nombre de 1 à 3 (ou 0 pour terminer) ', end=' ')
 4#  ch = input()
 5#  a = int(ch)              # conversion de la chaîne entrée en entier
 6#  while a:                 # équivalent à : < while a != 0: >
 7#      if a == 1:
 8#          print("Vous avez choisi un :")
 9#          print("le premier, l'unique, l'unité ..."
10#      elif a == 2:
11#          print("Vous préférez le deux :")
12#          print("la paire, le couple, le duo ...")
13#      elif a == 3:
14#          print("Vous optez pour le plus grand des trois :")
15#          print("le trio, la trinité, le triplet ...")
16#      else :
17#          print("Un nombre entre UN et TROIS, s.v.p.")
18#      print('Choisissez un nombre de 1 à 3 (ou 0 pour terminer) ', end =' ')
19#      a = int(input())
20#  print("Vous avez entré zéro :")
21#  print("L'exercice est donc terminé.")
```

Nous retrouvons ici une boucle **while**, associée à un groupe d'instructions **if**, **elif** et **else**. Notez bien cette fois encore comment la structure logique du programme est créée à l'aide des indentations (... et n'oubliez pas le caractère « : » à la fin de chaque ligne d'en-tête !).

À la ligne 6, l'instruction **while** est utilisée de la manière expliquée à la page 55 : pour la comprendre, il suffit de vous rappeler que toute valeur numérique autre que zéro est considérée comme « vraie » par l'interpréteur Python. Vous pouvez remplacer cette forme d'écriture par : « **while a != 0** : » si vous préférez (rappelons à ce sujet que l'opérateur de comparaison **!=** signifie « est différent de »), mais c'est moins efficace.

Cette « boucle while » relance le questionnement après chaque réponse de l'utilisateur (du moins jusqu'à ce que celui-ci décide de « quitter » en entrant une valeur nulle :).

Dans le corps de la boucle, nous trouvons le groupe d'instructions **if**, **elif** et **else** (de la ligne 7 à la ligne 17), qui aiguille le flux du programme vers les différentes réponses, ensuite une instruction **print()** et une instruction **input()** (lignes 18 & 19) qui seront exécutées dans tous les cas de figure : notez bien leur niveau d'indentation, qui est le même que celui du bloc **if**, **elif** et **else**. Après ces instructions, le programme boucle et l'exécution reprend à l'instruction **while** (ligne 6).

À la ligne 19, nous utilisons la composition pour écrire un code plus compact, qui est équivalent aux lignes 4 & 5 rassemblées en une seule.

Les deux dernières instructions **print()** (lignes 20 & 21) ne sont exécutées qu'à la sortie de la boucle.

Exercices

6.5 Que fait le programme ci-dessous, dans les quatre cas où l'on aurait défini au préalable que la variable **a** vaut 1, 2, 3 ou 15 ?

```
if a !=2:
    print('perdu')
elif a ==3:
    print('un instant, s.v.p.')
else :
    print('gagné')
```

6.6 Que font ces programmes ?

a)
```
a = 5
b = 2
if (a==5) & (b<2):
    print('"&" signifie "et"; on peut aussi utiliser\
           le mot "and"')
```

b)
```
a, b = 2, 4
if (a==4) or (b!=4):
    print('gagné')
elif (a==4) or (b==4):
    print('presque gagné')
```

c)
```
a = 1
if not a:
    print('gagné')
elif a:
    print('perdu')
```

6.7 Reprendre le programme c) avec a = 0 au lieu de a = 1.
Que se passe-t-il ? Conclure !

6.8 Écrire un programme qui, étant données deux bornes entières a et b, additionne les nombres multiples de 3 et de 5 compris entre ces bornes. Prendre par exemple a = 0, b = 32 ; le résultat devrait être alors 0 + 15 + 30 = 45.
Modifier légèrement ce programme pour qu'il additionne les nombres multiples de 3 ou de 5 compris entre les bornes a et b. Avec les bornes 0 et 32, le résultat devrait donc être : 0 + 3 + 5 + 6 + 9 + 10 + 12 + 15 + 18 + 20 + 21 + 24 + 25 + 27 + 30 = 225.

6.9 Déterminer si une année (dont le millésime est introduit par l'utilisateur) est bissextile ou non. Une année A est bissextile si A est divisible par 4. Elle ne l'est cependant pas si A est un multiple de 100, à moins que A ne soit multiple de 400.

6.10 Demander à l'utilisateur son nom et son sexe (M ou F). En fonction de ces données, afficher « Cher Monsieur » ou « Chère Mademoiselle » suivi du nom de la personne.

6.11 Demander à l'utilisateur d'entrer trois longueurs a, b, c. À l'aide de ces trois longueurs, déterminer s'il est possible de construire un triangle. Déterminer ensuite si ce triangle est rectangle, isocèle, équilatéral ou quelconque. Attention : un triangle rectangle peut être isocèle.

6.12 Demander à l'utilisateur qu'il entre un nombre. Afficher ensuite : soit la racine carrée de ce nombre, soit un message indiquant que la racine carrée de ce nombre ne peut être calculée.

6.13 Convertir une note scolaire N quelconque, entrée par l'utilisateur sous forme de points (par exemple 27 sur 85), en une note standardisée suivant le code ci-dessous :

Note	Appréciation
N >= 80 %	A
80 % > N >= 60 %	B
60 % > N >= 50 %	C
50 % > N >= 40 %	D
N < 40 %	E

6.14 Soit la liste suivante :
```
['Jean-Michel', 'Marc', 'Vanessa', 'Anne', 'Maximilien',
 'Alexandre-Benoît', 'Louise']
```

Écrivez un script qui affiche chacun de ces noms avec le nombre de caractères correspondant.

6.15 Écrire une boucle de programme qui demande à l'utilisateur d'entrer des notes d'élèves. La boucle se terminera seulement si l'utilisateur entre une valeur négative. Avec les notes ainsi entrées, construire progressivement une liste. Après chaque entrée d'une nouvelle note (et donc à chaque itération de la boucle), afficher le nombre de notes entrées, la note la plus élevée, la note la plus basse, la moyenne de toutes les notes.

6.16 Écrivez un script qui affiche la valeur de la force de gravitation s'exerçant entre deux masses de 10 000 kg , pour des distances qui augmentent suivant une progression géométrique de raison 2, à partir de 5 cm (0,05 mètre).

La force de gravitation est régie par la formule $F = 6{,}67 \,.\, 10^{-11} . \frac{m \cdot m'}{d^2}$

Exemple d'affichage :

```
d = .05 m :  la force vaut  2.668 N
d = .1 m  :  la force vaut  0.667 N
d = .2 m  :  la force vaut  0.167 N
d = .4 m  :  la force vaut  0.0417 N
```

etc.

6.17 A l'aide du module turtle, reproduisez le dessin ci-dessous :