Les cahiers d'Exercices en Programmation : Le langage Python

Apprendre à programmer

Apprenez et entraînez vos acquis

- De très nombreux exercices à réaliser par vous-même

- Des corrigés expliqués Pas à Pas.

AVANT-PROPOS

Ce livre est un cahier d'exercices : il vous propose des énoncés d'exercices et leurs corrigés. Vous allez apprendre le logiciel en vous entrainant à travers des exercices regroupés par thème.

Chaque énoncé vous présente l'exercice à réaliser. Vous trouverez à la fin du cahier le corrigé de chaque exercice. Certaines explications peuvent-être présentes.

METHODOLOGIE

Lors de la réalisation des exercices, vous pourrez remédier à certain problème à l'aide des corrections à la fin du cahier.

Après avoir réalisé tous les exercices de chaque chapitre vous allez pouvoir vérifier les compétences acquises à l'aide du tableau des objectifs.

Celui-ci sert à la cotation du professeur (grille d'évaluation).

Des **légendes ou recommandations** peuvent être présentes dans certains exercices. Celles-ci vous aideront dans vos recherches. Elles ne doivent pas être reproduites dans votre travail.

Chaque point de matière acquis dans un exercice peut être utilisé dans des exercices suivants sans explication.

Table des matières

Avant-Propos

Pourquoi apprendre la programmation informatique aux enfants ?

La programmation encourage la créativité, le raisonnement et la résolution de problèmes.

Opportunité du programmeur :

- Créer quelque chose à partir de rien ;
- Utiliser la logique pour transformer les constructions de programmation en une forme que l'ordinateur peut exécuter ;
- Exploiter ses capacités de résolution afin de trouver ce qui ne va pas.

La programmation est une activité amusante pleine de défis.

Les compétences acquises sont utiles à la fois à l'école et au travail.

Site d'accompagnement

Va sur l'adresse http://www.editions-eyrolles.com/go/pykids.

Tu y trouveras des téléchargements pour tous les exemples du livre et d'autres puzzles de programmation, ainsi que les solutions des puzzles de programmation, si tu te sens perdu et si tu veux comparer ta solution avec celles proposées.

Chapitre 1 : Introduction à Python

Objectif(s) des exercices de ce chapitre.

À la fin de ce chapitre, l'élève sera capable de :

Lancer l'interpréteur
Utiliser la fonction d'affichage : PRINT
Saisir des expressions (*, +, -, /, //, %, **)
Comprendre les expressions
Introduire des commentaires à l'aide du symbole : #

1. Qu'est-ce que le langage Python ?

Python est un langage facile à apprendre. Le code est assez facile à lire. Python possède une console de commandes interactive pour y entrer tes programmes afin de les voir fonctionner. Python propose certaines fonctionnalités qui améliorent fortement l'apprentissage et permettent de rassembler des animations simples pour créer tes propres jeux.

- Python est un langage de programmation interprété (il est lui traduit en quelque sorte au fur et à mesure de l'exécution par l'interpréteur), à ne pas confondre avec un langage compilé (langage qui est, avant de pouvoir l'exécuter, traduit en langage machine (binaire) par un compilateur).
- Il permet de créer toutes sortes de programmes, comme des jeux, des logiciels, des progiciels, etc.
- Il est possible d'associer des bibliothèques à Python afin d'étendre ses possibilités.
- Il est portable, c'est à dire qu'il peut fonctionner sous différents systèmes d'exploitation (Windows, Linux, Mac OS X,…).

Dans les chapitres suivants, tu trouveras des informations pour installer Python, démarrer la console et réaliser des calculs simples, afficher du texte à l'écran, créer des listes, réaliser des opérations élémentaires de contrôle de flux à l'aide d'instructions if, ainsi que des boucles for. Tu comprendras ce qu'elles signifient. Tu apprendras à réutiliser du code dans des fonctions, les éléments fondamentaux des classes et des objets, ainsi que les descriptions de quelques-uns des nombreux modules et fonctions intégrés dans Python.

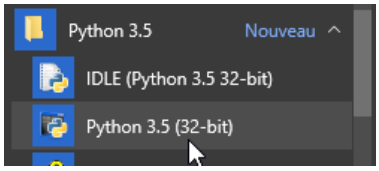
Des chapitres inspectent les graphismes à l'aide de la tortue, sous forme simple ou plus évoluée, ainsi que l'utilisation du module tkinter pour dessiner à l'écran de l'ordinateur.

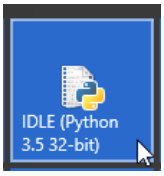
Quand tu auras construit tes connaissances fondamentales de la programmation, tu apprendras à écrire tes propres jeux.

2. Utiliser Python comme une calculette grâce à l'interpréteur de commande

Ce site peut te permettre d'utiliser Python Online (pas recommandé pour la syntaxe) : *https://www.tutorialspoint.com/execute_python_online.php*

Où tu peux aussi lancer l'interpréteur sous Windows (il doit être bien sûr installé) ou l'IDLE :





Dans les exercices futurs si ce n'est pas signalé, quand vous allez voir ces caractères précédents une commande

```
>>> import turtle
>>> aline = turtle.Pen()
>>> karine = turtle.Pen()
```

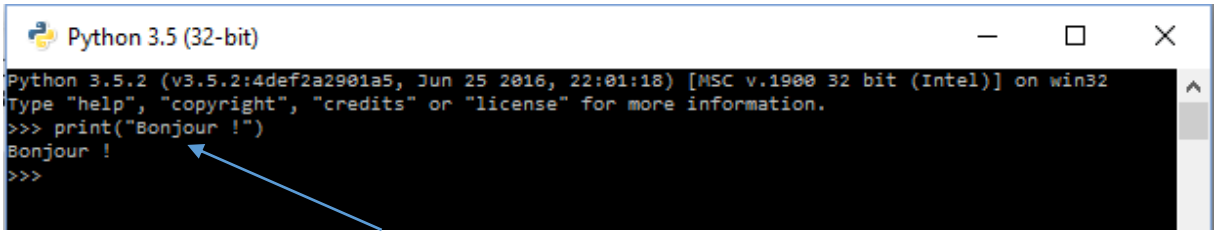
cela veut signaler que vous devez les introduire dans l'IDLE Python.

Sinon c'est à partir de Notepad ++ ou jedit.

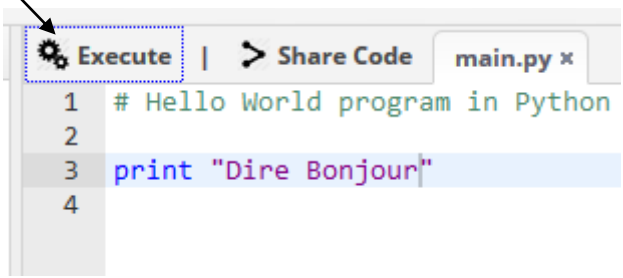
Et sous Mac OSX :

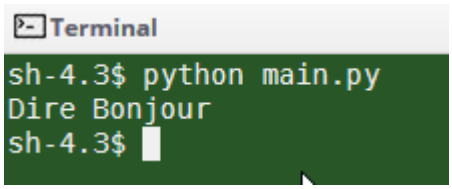
Cherchez un dossier `Python` dans le dossier `Applications`. Pour lancer Python, ouvrez l'application `IDLE` de ce dossier.

Exercice : Commençons l'apprentissage par l'affichage d'une salutation (commande print) et ensuite Enter.



Où à partir du site Python Online :





Ou avec Notepad ++. Mais le fichier doit être exécuté avec l'IDLE Python (je préfère cette manière de travailler).

La console Python (IDLE) pas le site Python Online fonctionne aussi comme une simple calculatrice.

Vous pouvez saisir une expression dont la valeur est renvoyée dès que vous appuyez sur la touche Enter.

Exercez-vous en tapant ces expressions :

```
>>> 2 * 5 + 6 - (100 + 3)
-87
>>> 7 / 2; 7 / 3
3.5
2.3333333333333335
>>> 34 // 5; 34 % 5 # quotient et reste de la division euclidienne de 34 par 5
6
4
>>> 2 ** 7  # pour l'exponentiation (et non pas 2^7 !)
128
```

// : Obtenir la partie entière d'une division

% : Modulo ou reste de la division

: Introduction d'un

Chapitre 2 : Éditer, exécuter et sauvegarder un programme Python avec Notepad++

Objectif(s) de ce chapitre.

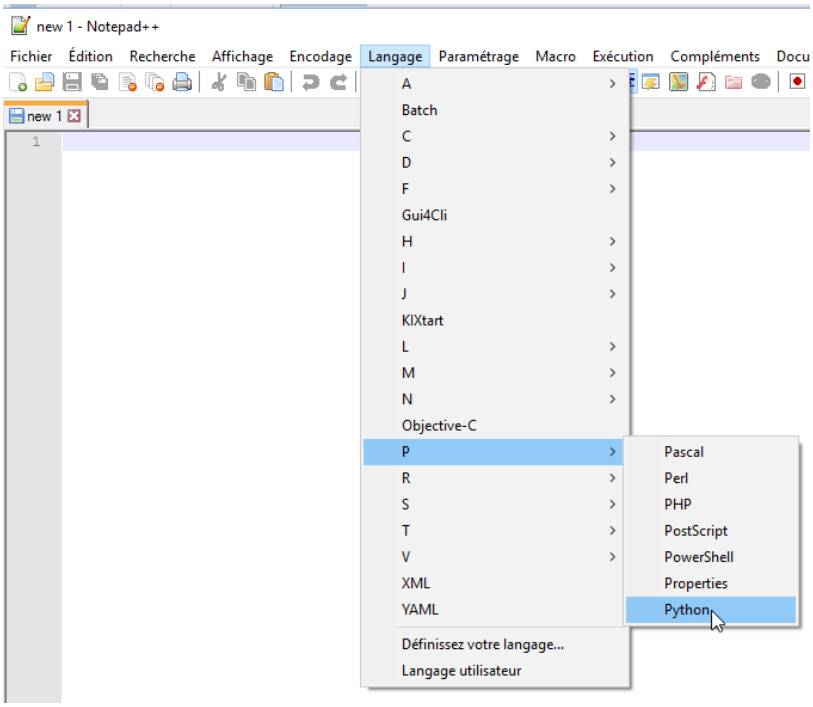
À la fin de ce chapitre, l'élève sera capable de :

Exécuter Notepad ++
Spécifier un langage
Créer et sauvegarder un programme avec une extension Python : .py
Ouvrir et lancer un programme créé

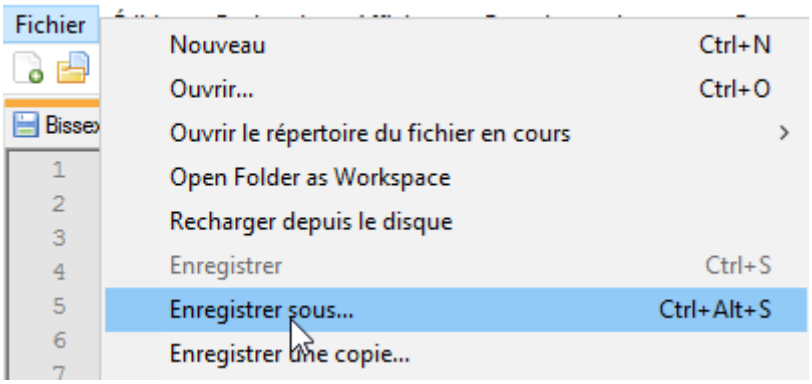
1. Utilisation de Notepad ++

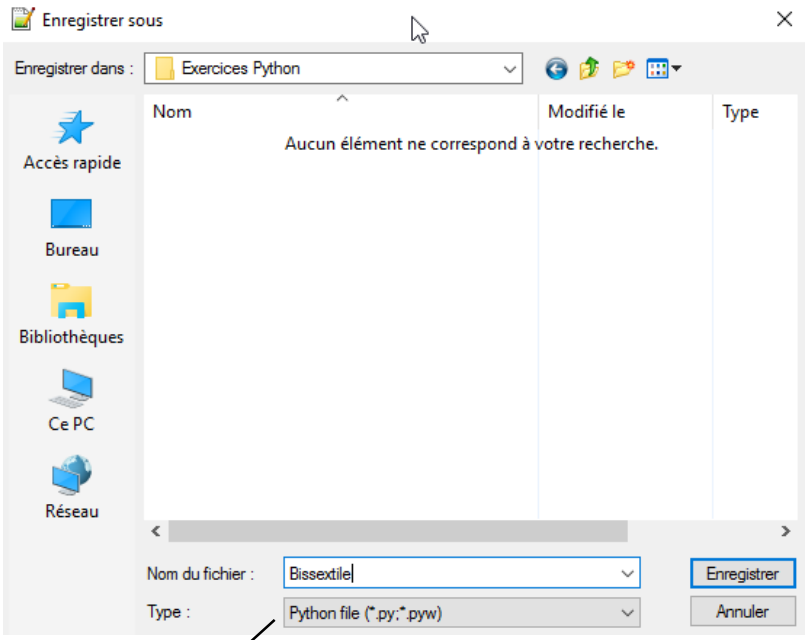
Exécuter Notepad ++

Spécifier que le langage utilisé sera en Python.



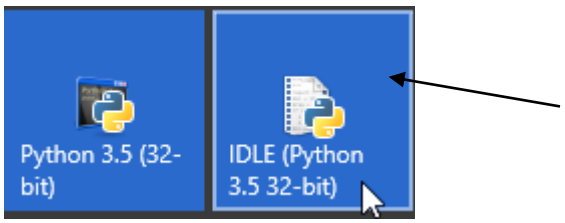
2. Sauvegarde du programme

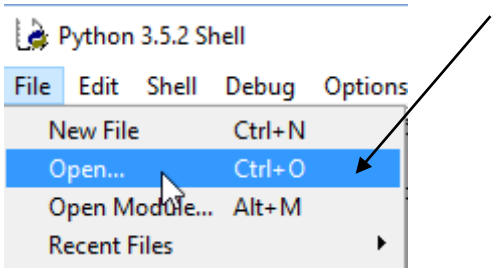


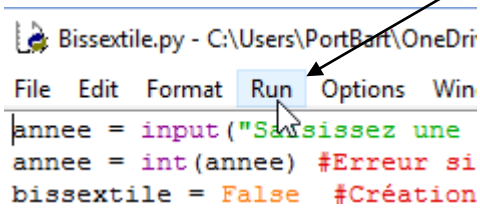


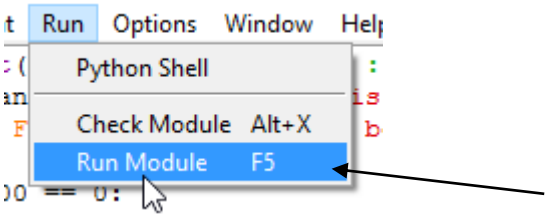
Extension Python

3. Exécuter le programme









Chapitre 3 : Le monde des variables

Objectif(s) des exercices de ce chapitre.

À la fin de ce chapitre, l'élève sera capable de :

Affecter une valeur à une variable
Connaître le type d'une variable : TYPE
Retourner un identifiant unique pour un objet : ID
Réaliser des opérations arithmétiques simples

Dans ce chapitre, tu vas apprendre à écrire des formules simples à l'aide d'opérateurs de Python et à te servir des parenthèses pour contrôler l'ordre des opérations. Tu vas ensuite créer des variables, que tu vas utiliser dans des calculs.

1. Variables et affectations

Une affectation se fait à l'aide du symbole "=". Elle est gardée en mémoire pour une utilisation ultérieure.

Taper ces expressions dans l'IDLE Python :

```
>>> a = 128
>>> a
128
>>> a, id(a), type(a)
(128, 137180768, <class 'int'>)
>>> 2 * a
256
```

a = 128 : On affecte ici à la variable a la valeur 128. Un ordre est à garder dans l'affectation. On ne peut pas écrire comme ligne de commande 128 = a qui aboutira à une erreur dans l'interpréteur.

id(a) : retourne un identifiant unique pour chaque objet.

type(a) : savoir de quel type est une variable. a appartient donc à la classe des entiers.

En résumé :

- Les variables permettent de conserver dans le temps des données de votre programme.
- Vous pouvez vous servir de ces variables pour différentes choses : les afficher, faire des calculs avec, etc.
- Pour affecter une valeur à une variable, on utilise la syntaxe `nom_de_variable = valeur`.
- Il existe différents types de variables, en fonction de l'information que vous désirez conserver : int, float, chaîne de caractères etc.
- Pour connaitre le type d'une variable, on utilise la fonction `type`
- Pour afficher une donnée, comme la valeur d'une variable par exemple, on utilise la fonction `print`.

2. Exercices de synthèse

Ex1. Quel est le type de la variable 3.4 :

Ex2. Quel est le type de la variable "Un essai" :

Ex3. Trouver les lignes de commandes pour :

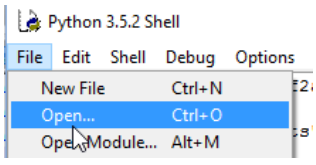
- Affecter la valeur 3 à la variable `a` ;
- Afficher la variable `a` ;
- Augmenter de 3 la valeur de la variable `a` ;
- Affecter à la variable `b` la valeur `a - 2` ;
- Afficher ceci : `a = 6 et b = 4`

Ex4. Introduit cet exercice avec Notepad++. N'oublie pas de changer le langage.

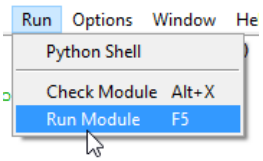
```
1  print("quatre plus quatre =")
2  print(4+4)
3  print("huit fois huit =")
4  print(8*8)
```

Enregistre-le sous le nom Opérations

Lance-le à l'aide de l'IDLE :



Une fois le fichier ouvert, tu dois l'exécuter :



Ex5. Écris-moi cette opération dans l'IDLE :

Python respecte t'il les priorités opératoires ?

```
>>> 20+10*365-3*52
3514
>>>
```

Ex6. Calcul l'opération suivante avec l'IDLE : ((20+10)*36-3)*52

Combien obtiens-tu ?

3. Solutions :

Ex1. <class 'float'>

Ex2. <class 'str'>

Ex3.

```
>>> a = 3
>>> print(a)
>>> a = a + 3
>>> b = a - 2
>>> print("a =", a, "et b =", b)
```

Chapitre 4 : Les chaînes, listes et Dictionnaires

Objectif(s) des exercices de ce chapitre.

À la fin de ce chapitre, l'élève sera capable de :

Utiliser des chaînes de caractères pour stocker du texte, des listes des dictionnaires
Manipuler des éléments multiples
Joindre une liste à une autre
Utiliser des dictionnaires

Dans ce chapitre, tu vas découvrir la manière dont Python utilise des chaînes pour stocker du texte, des listes et manipuler des éléments multiples. Tu vas apprendre que les éléments d'une liste peuvent être modifiés et que tu peux joindre une liste à une autre. Tu vas voir aussi comment utiliser des dictionnaires pour stocker des valeurs avec les clés qui les identifient.

1. Les chaînes

Tape-moi ces commandes dans l'IDLE. N'enregistre pas ces deux exercices.

1.

```
>>> fred = "Les gorilles ont de grosses narines, pourquoi ? Parce qu'ils ont de
gros doigts !"
>>> print(fred)
Les gorilles ont de grosses narines, pourquoi ? Parce qu'ils ont de gros doigts
!
```

print(fred) permet de voir ce que contient la variable fred.

2.

```
>>> monscore = 1000
>>> message = "Tu as obtenu %s points"
>>> print(message % monscore)
Tu as obtenu 1000 points
```

%s est un espace réservé pour le nombre de points.

% dit à Python de remplacer la valeur le %s du contenu du message par lz vzleur de la variable monscore.

2. Multiplier des chaînes

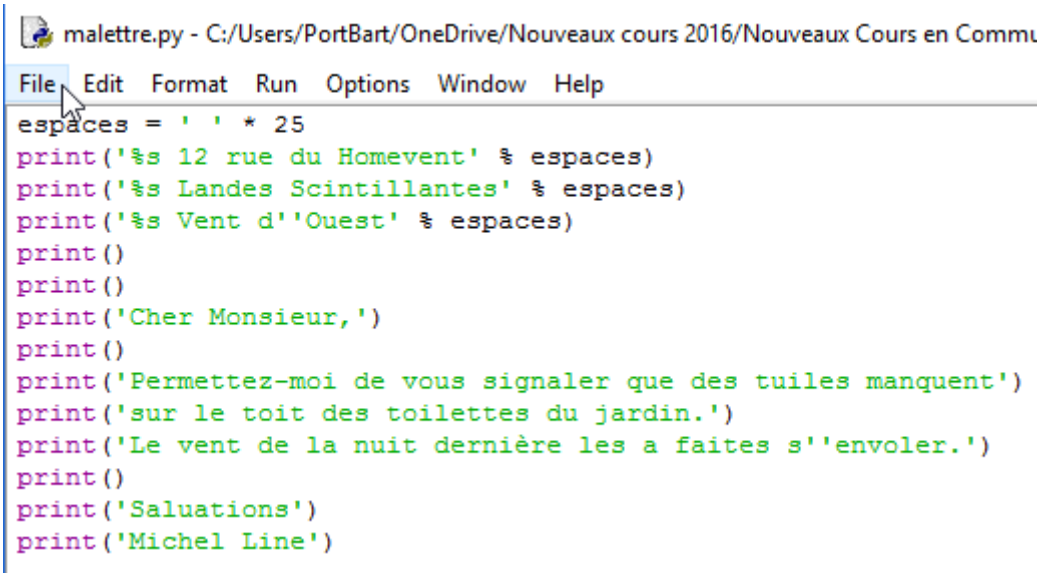
Teste cette commande dans l'IDLE :

```
>>> print(10* 'a')
aaaaaaaaaa
```

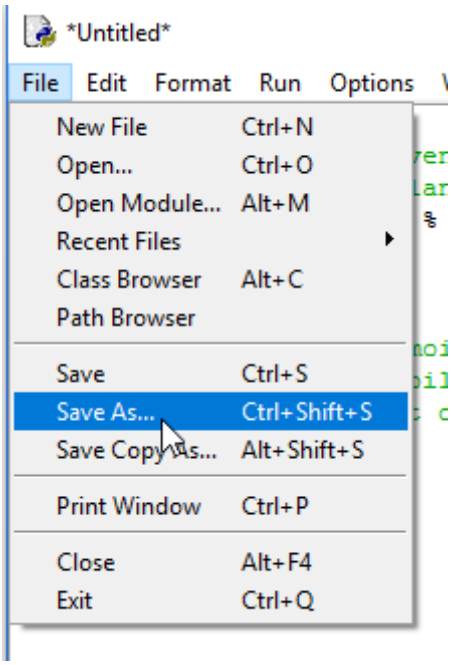
On peut donc multiplier des chaînes de caractères.

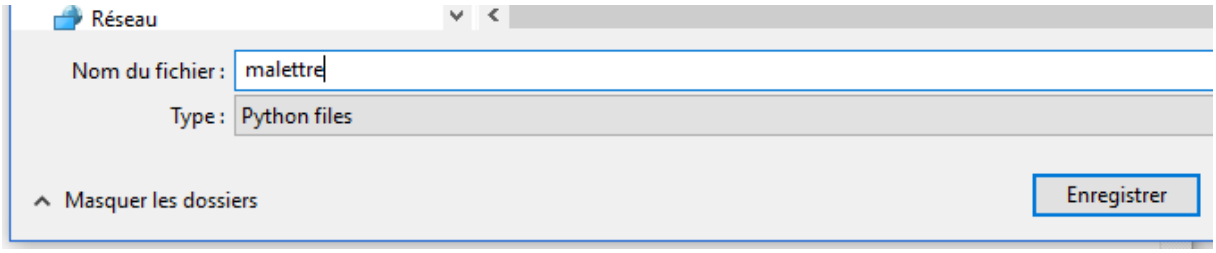
Alignons dans le programme suivant des chaînes de caractères avec un nombre déterminé d'espaces lors de l'affichage.

Créer un nouveau document (IDLE). Sélectionne le menu File > New File et saisis le code suivant :

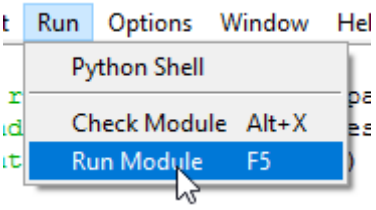


Dès que les lignes sont tapées, clique dans le menu File > save as et nomme le fichier malettre.py que tu enregistreras dans ton dossier personnel de travail.





Lance ton programme.



Et admire ton boulot.

3. Plus puissant que les chaînes : les listes

Tu vas pour cet exercice te transformer en sorcier ou sorcière.

Une liste d'ingrédient devra être créée.

Ferme avant tout ton programme précédent et introduit ces lignes de commandes dans l'IDLE Python.

```
>>> liste_sorcier = "patte d'araignée, ortieil de crapaud, oeil de triton, aile
de chauve-souris, beurre de limace, écailles de serpent"
>>> print(liste_sorcier)
patte d'araignée, ortieil de crapaud, oeil de triton, aile de chauve-souris, beu
rre de limace, écailles de serpent
```

Voici une autre manière de travailler plus intéressante car ici la liste pourra être manipulée.

Tape donc ces lignes de codes :

```
>>> liste_sorcier=["pattes d'araignée","orteil de crapaud","oeil de triton","ail
e de chauve-sourisé","beurre de limace","écailles de serpent"]
>>> print(liste_sorcier)
["pattes d'araignée", 'orteil de crapaud', 'oeil de triton', 'aile de chauve-sou
risé', 'beurre de limace', 'écailles de serpent']
```

Rien de fort différent mais maintenant tape ceci :

```
>>> print(liste_sorcier[3])
aile de chauve-sourisé
```

Tu peux aussi modifier un élément de la liste.

Modifions le 3ème élément qui est aile de chauve-souris en bave de crapaud gluant.

Tape donc ceci :

```
>>> liste_sorcier[3]="bave de crapaud gluant"
```

Maintenant réaffiche toute la liste :

```
>>> print(liste_sorcier)
["pattes d'araignée", 'orteil de crapaud', 'oeil de triton', 'bave de crapaud gl
uant', 'beurre de limace', 'écailles de serpent']
```

Faire attention que le premier élément de la liste porte le numéro 0.

Affichons maintenant les éléments de la liste allant du 2ème jusqu'au 5ème.

```
>>> print(liste_sorcier[2:5])
['oeil de triton', 'bave de crapaud gluant', 'beurre de limace']
```

Dans les listes ont peux y stocker des nombres et des chaînes.

4. Ajouter des éléments à une liste

Tape ce code à la suite des précédents.

```
>>> liste_sorcier.append("rot d'ours")
>>> print(liste_sorcier)
["pattes d'araignée", 'orteil de crapaud', 'oeil de triton', 'bave de crapaud g
uant', 'beurre de limace', 'écailles de serpent', "rot d'ours"]
```

5. Supprimer des éléments de la liste

Tape maintenant ce code.

```
>>> del liste_sorcier[5]
>>> print(liste_sorcier)
["pattes d'araignée", 'orteil de crapaud', 'oeil de triton', 'bave de crapaud gl
uant', 'beurre de limace', "rot d'ours"]
```

Écailles de serpent sera supprimé de la liste car il est le 5ème. En considérant que le 1er possède le n° 0.

6. Additionner ou regrouper des listes

Créons deux nouvelles listes.

Tape ce code :

```
>>> liste1 = [1,2,3,4]
>>> liste2 = ["J'ai","trébuché","et","je","suis","tombé"]
>>> print(liste1 + liste2)
[1, 2, 3, 4, "J'ai", 'trébuché', 'et', 'je', 'suis', 'tombé']
```

Additionne les deux listes dans une 3ème.

```
>>> liste3=liste1 + liste2
```

Affiche le résultat :

```
>>> print(liste3)
[1, 2, 3, 4, "J'ai", 'trébuché', 'et', 'je', 'suis', 'tombé']
```

7. Créer un dictionnaire

Principale particularité est que chacun de ses éléments possède une clé et une valeur correspondante.

Tape ces lignes de codes :

```
>>> sports_favoris = {"Ralph Williams" : "Football", "Michael Tippett" : "Basket
-ball", "Edward Elgar" : "Base-ball", "Rebecca Clarke" : "Voley-ball", "Ethel Sm
yth" : "Badminton", "Frank Bridge" : "Rugby"}
>>> print(sports_favoris["Rebecca Clarke"])
Voley-ball
```

Chaque élément (un sportif) est donc en lien avec une valeur (un sport).

8. Supprimer une valeur d'un dictionnaire

Tu vas tout d'abord afficher la liste complète et par après supprimer un élément.

```
>>> print(sports_favoris)
{'Frank Bridge': 'Rugby', 'Ralph Williams': 'Football', 'Ethel Smyth': 'Badminto
n', 'Michael Tippett': 'Basket-ball', 'Rebecca Clarke': 'Voley-ball', 'Edward El
gar': 'Base-ball'}
>>> del sports_favoris["Rebecca Clarke"]
>>> print(sports_favoris)
{'Frank Bridge': 'Rugby', 'Ralph Williams': 'Football', 'Ethel Smyth': 'Badminto
n', 'Michael Tippett': 'Basket-ball', 'Edward Elgar': 'Base-ball'}
```

9. Remplacer une valeur d'un dictionnaire

Tape ce code dans l'IDLE.

```
>>> sports_favoris["Ralph Williams"] = "Hockey sur glace"
>>> print(sports_favoris)
{'Frank Bridge': 'Rugby', 'Ralph Williams': 'Hockey sur glace', 'Ethel Smyth': '
Badminton', 'Michael Tippett': 'Basket-ball', 'Edward Elgar': 'Base-ball'}
```

Le sport de ce jeune homme a bien changé.

10. Puzzle de programmation

Crée pour t'exercer ces trois exercices.

1. Favoris

Crée une liste de tes loisirs favoris et donne-lui le nom de variable jeux. Crée ensuite une liste de tes nourritures préférées et nomme la variable nourritures. Joins les deux listes et appelle le résultat "favoris". Enfin, affiche le contenu de la variable.

2. Compter les combattants

Supposons que tu aies 3 bâtiments avec 25 ninjas cachés sur le toit de chaque bâtiment, ainsi que 2 tunnels avec 40 samouraïs cachés dans chaque tunnel. Combien de ninjas et de samouraïs sont prêts à se battre ? Une seule formule suffit dans le shell Python (IDLE) pour calculer cela.

3. Salutations

Crée deux variables : l'une pointe vers ton prénom et l'autre vers ton nom de famille. Réalise ensuite une chaîne qui utilise des espaces réservés pour afficher ton nom dans un message avec ces deux variables, du genre : Bonjour, Kévin Costume !

Chapitre 5 : Les structures conditionnelles

Objectif(s) des exercices de ce chapitre.

À la fin de ce chapitre, l'élève sera capable de :

Comprendre et savoir utiliser une condition IF et un bloc d'instructions
Comprendre et savoir utiliser dans une condition les mots-clés (if, elif et else)
Savoir utiliser les opérateurs de comparaison
Comprendre la notion d'indentation
Comprendre et savoir utiliser le type booléen
Comprendre et savoir utiliser les mots-clés (and, or et not)
Comprendre et savoir utiliser le mot-clé (is)

Dans ce chapitre tu vas apprendre à créer des blocs de code qui ne s'exécutent que lorsque des conditions particulières sont vraies.

Pour exécuter des portions différentes de code en réponse à des conditions différentes de code à l'aide de elif.

Si aucune des conditions n'est vérifiée un mot-clé else pourra exécuter une autre partie du code.

1. Les structures conditionnelles

Les instructions ne sont plus ici linéaires : dans le chapitre précédent l'interpréteur exécutait au fur et à mesure le code que vous saisissiez dans la console.

La structure conditionnelle que nous allons inclure dans les programmes va nous permettre de réaliser des tests et d'aller plus loin dans la programmation.

Les conditions permettent d'exécuter une ou plusieurs instructions dans un cas et d'autres instructions dans un autre cas.

2. La première condition et bloc d'instructions : la forme minimale en IF

Les conditions vont vous permettre de faire une action précise si une variable est par exemple positive et une autre action si celle-ci est négative ou une troisième action si la variable est nulle.

Ex5 : Lancer la console Python et taper ce premier exemple :

```
>>> # Premier exemple de condition
>>> a=5
>>> if a >0 : print (a," est plus grand que 0")

5  est plus grand que 0
```

Taper deux fois sur la touche Enter pour que le programme se lance.

Un bloc d'instruction = série d'instructions.

Donc, on attribue à la variable a la valeur de 5.

Et on test : Si la valeur de a est strictement plus grande que 0 alors j'affiche à l'écran a est plus grand que 0 sinon on écrit rien.

Le Si = mot clé IF.

Ex6 : Tester le même programme mais en attribuant à la variable a la valeur de -2.

Votre programme écrit quoi ?

Ex7 : Taper ce second exemple comportant un bloc d'instructions :

```
>>> #Deuxième exemple avec un bloc d'instructi
>>> a = 5
>>> b = 8
>>> if a > 0:
        # On incrémente la valeur de b
        b += 1
        # On ajoute les valeurs des variables
        print("a=",a," et b=",b)


a= 5  et b= 9
```

Notion importante : l'indentation.

On entend par indentation un certain décalage vers la droite, obtenu par un (ou plusieurs) espaces ou tabulations.

C'est un moyen pour l'interpréteur de savoir où se trouvent le début et la fin d'un bloc.

3. La forme complète (if, elif et else)

La première forme de condition est assez incomplète.

Considérons, par exemple, une variable a de type entier. On souhaite faire une action si cette variable est positive et une action différente si elle est négative.

L'instruction else (Ex8) :

Le mot-clé else, qui signifie « sinon » en anglais, permet de définir une première forme de complément à notre instruction if.).

```
age=21
if age >=18:
        print("Vous êtes majeur.")
else:
        print("Vous êtes mineur.")
```

If et Else : même niveau d'indentation. Et si a valait 0 ?

L'instruction elif (sinon-si) (Ex9) :

Le mot clé elif est une contraction de « else if », que l'on peut traduire par « sinon si ». Dans l'exemple que nous venons juste de voir, l'idéal serait d'écrire :

- si a est strictement supérieur à 0, on dit qu'il est positif ;
- sinon si a est strictement inférieur à 0, on dit qu'il est négatif ;
- sinon a ne peut qu'être égal à 0, on dit alors que a est nul.

```
1  a=0
2  if a >0:
3      print("a est positif.")
4  elif a<0:
5      print("a est négatif.")
6  else :
7      print("a est nul")
```

Tu peux ajouter autant de elif que tu veux.

4. Autres opérateurs de comparaison

Opérateur	Signification littérale
<	Strictement inférieur à
>	Strictement supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à
==	Égal à
!=	Différent de

La condition entre le if et le : cela se nomme un prédicat.

Tester les comparaisons suivantes directement en ligne de commande à partir du programme Python 3.5.

```
>>> a = 0
>>> a == 5
False
>>> a > -8
True
>>> a != 85.54
True
```

5. Le type Booléen (bool)

Au point précédent, vous pouvez remarquer que l'interpréteur renvoie un `True` ou un `False` (Vrai ou faux).

Ce sont les 2 valeurs de type booléen.

Le `T` et le `F` majuscule sont obligatoires pour que Python les comprenne.

Voici un exemple (Ex10) :

```
1  age = 21
2  majeur = False
3  if age >=18:
4      majeur = True
```

Que vaut la valeur booléenne de la variable majeur ?

Et comment afficher sa valeur en ligne de commande ?

```
>>> print(majeur)
True
>>>
```

6. Les mots-clés and, or et not

Utilité : tester plusieurs prédicats dans une condition.

Voici un exemple d'une condition situant un entier entre deux intervalles sans les mots-clés (Ex11) :

```
1  a = 5
2  if a >= 2:
3      if a <= 8:
4          print("a est dans l'intervalle.")
5      else :
6          print("a n'est pas dans l'intervalle.")
7  else :
8      print("a n'est pas dans l'intervalle.")
```

Assez lourd comme code !

Nous allons utiliser maintenant le mot clé and (et) et transformer l'exemple précédent (Ex12).

```
a=5
if a>=2 and a <=8:
    print("a est dans l'intervalle.")
else:
    print("a n'est pas dans l'intervalle.")
```

Plus simple et plus compréhensible.

Exercice Ex13 : Essayer de transformer le programme précédent en utilisant le mot clé or (ou) (la réponse finale doit-être identique).

Voici la solution :

```
>>> if a < 2 or a > 8:
...     print("a n'est pas dans l'intervalle.")
... else:
...     print("a est dans l'intervalle.")
...
a est dans l'intervalle.
```

Enfin il existe le mot-clé qui "inverse" un prédicat.

Tester ce programme (Ex14) :

```
majeur = False
if majeur is not True:
    print("Vous n'êtes pas encore majeur.")
```

Ajout à la liste d'un nouveau mot-clé : is.

Ce mot clé teste l'égalité non pas des valeurs de deux variables, mais de leurs références.

Chapitre 6 : Votre premier programme

Objectif(s) des exercices de ce chapitre.

À la fin de ce chapitre, l'élève sera capable de :

Comprendre et savoir utiliser la fonction Input()
Tester si un nombre est multiple d'un autre
Comprendre et savoir utiliser la matière vue dans les chapitres précédents

1. La fonction Input()

Input() : Attente d'une valeur saisie par l'utilisateur. Cette valeur est attribuée à une variable.

Cette fonction accepte un paramètre entre ses parenthèses : message affiché à l'utilisateur.

Ce message est encadré de guillemets " ".

Taper cet exemple :

```
1    annee = input("Saisissez une année : ")
```

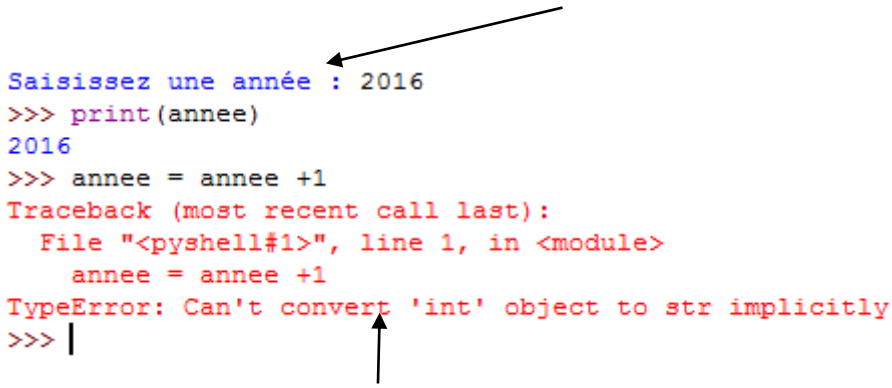
Lors de l'exécution du programme avec l'IDLE, introduisez la valeur : 2016.

Comment afficher la valeur introduite par l'utilisateur à partir du Shell Python ?

```
Saisissez une année : 2016
>>> print(annee)
2016
>>>
```

Un petit problème est quand-même présent. Le type de l'année est une chaîne de caractère et nous voulions travailler sur un entier.

Une vérification peut-être réalisée en tapant ceci :



Et vous apercevrez ce message d'erreur.

Il faudra donc convertir cette variable à l'aide de la fonction int() qui prend en paramètre la variable d'origine.

Ajouter ces lignes de commandes à la suite (toujours dans le Shell Python) :

```
>>> annee=int(annee)
>>> type(annee)
<class 'int'>
```

Type indique le type de la variable qui est bien maintenant un entier.

Réaliser un Quizz simple (Ex16)

Introduit ce code :

```
1  a=input("Quelle est la capitale de l'Angleterre ? : ")
2  if a=="Londres":
3      print("Correct")
```

La commande Si vérifie si une information est vraie. La ligne suivante s'exécute si c'est le cas.

2. Comment tester des multiples

Comment tester si un nombre a est multiple d'un nombre b ?

Il suffit de tester le reste de la division entière de b par a. Si ce reste est nul, alors a est un multiple de b.

Taper ces deux commandes (Shell Python :

```
>>> 5%2 # Le reste = 1 donc 5 n'est pas un multiple de 2
1
>>> 8%2 # Le reste = 0 donc 8 est un multiple de 2
0
```

À vous de jouer

Tous les éléments nécessaires pour réussir les exercices sont à vos dispositions. Si vous avez du mal passez à la correction et étudiez-la soigneusement. Il n'y a pas qu'une seule solution. Bonne chance !

Note : Avant de commencer à travailler, sachez qu'il vous sera plus simple de taper votre programme dans un éditeur (notepad ++) et de le sauvegarder. Celui-ci pourra être exécuté par la suite à l'aide du programme Idle (voir les explications dans le chapitre suivant).

Exercice : Ex15 : Déterminer si une année saisie par l'utilisateur est bissextile.

Règle : Une année est dite bissextile si c'est un multiple de 4 sauf si c'est en même temps un multiple de 100. Mais elle est quand même considérée comme bissextile si c'est un multiple de 400.

Démarche à suivre :

- Si une année n'est pas multiple de 4, le programme s'arrête et on affiche : L'année introduite n'est pas bissextile ;
- Maintenant si elle est multiple de 4, on regarde si elle est multiple de 100 ;
    - Si c'est le cas, on regarde si elle est multiple de 400 ;
        - Si c'est le cas, on affiche l'année est bissextile ;
        - Sinon, on affiche elle n'est pas bissextile ;
    - Sinon, on affiche l'année est bissextile.

Exercice Ex16 : Crée un Quizz en Python avec 5 questions sur les capitales – Pays.

3. Solutions

Exercice 15 :

*Bissextile.py - C:\Users\PortBart\OneDrive\Nouveaux cours 2016\Nouveaux cours\Exercices Python\...

File Edit Format Run Options Window Help

```
annee = input("Saisissez une année :") #Année saisie par l'utilisateur
annee = int(annee) #Erreur si jamais l'utilisateur n'a pas saisi un nombre
bissextile = False  #Création d'un booléen qui vaut vrai ou faux
                                        # selon que l'année est bissextile ou non
if annee % 400 == 0:
        bissextile = True
elif annee % 100 == 0:
        bissextile = False
elif annee % 4 == 0:
        bissextile = True
else:
        bissextile = False

if bissextile: #Si l'année est bissextile on affiche
        print("L'année saisie est bissextile.")
else : #Sinon on affiche
        print("L'année saisie n'est pas bissextile.")
```

Chapitre 7 : Les boucles

Objectif(s) des exercices de ce chapitre.

À la fin de ce chapitre, l'élève sera capable de :

Comprendre et savoir utiliser la boucle for
Définir un nombre de départ et le nombre que l'on veut atteindre
Comprendre et savoir utiliser la boucle while
Utiliser le mot-clé break pour quitter une boucle

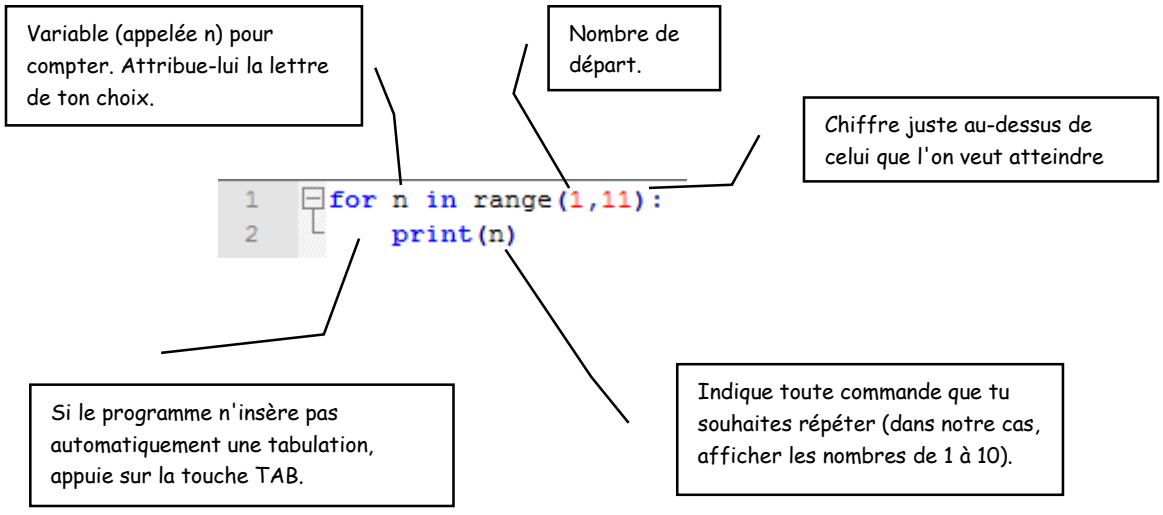
Dans ce chapitre, tu vas utiliser des boucles pour effectuer des tâches répétitives. Tu vas indiquer à Python ce que tu veux répéter en écrivant ces tâches à l'intérieur de blocs de code placés dans des boucles.

Pour quitter une boucle tu apprendras à utiliser le mot clé break.

1. À quoi ça sert le boucle for ?

Afficher de manière répétée des choses à l'écran.

Une boucle est une série de commande répétée un certain nombre de fois.



Comment compter alors jusque 100 (Ex17) ?

```
for n in range(1,101):
    print(n)
```

2. À ton tour ?

Teste toutes ces boucles et enregistre tes travaux :

Ex18 :

```
for n in range(1,21):
    print(n)
```

Ex19 :

```
for n in range(1,51):
    print(n)
```

Ex20 :

```
for a in range(1,201):
    print(a)
```

Ex 21 :

```
for b in range(1,101):
    print(b*10)
```

Ex22 :

```
for c in range(1,101):
    print(c*100)
```

3. Et la boucle While ?

Voici un programme à taper dans Notepad++.

Nomme se programme : while1

```
x = 45
y = 80
while x < 50 and y < 100:
    x = x + 1
    y = y + 1
    print(x, y)
```

Nous créons ici une variable x de valeur 45 et y de valeur 80.

La boucle vérifie deux conditions : d'abord que x est plus petit que 50, puis que y est plus petit que 100. Tant que ces deux condtions sont vraies, les lignes

suivantes sont exécutées et ajoutent 1 à chacune des deux variables, puis les affichent.

Résultat :

```
46 81
47 82
48 83
49 84
50 85
```

Chapitre 8 : Formes graphiques

Objectif(s) des exercices de ce chapitre.

À la fin de ce chapitre, l'élève sera capable de :

Importer et utiliser une bibliothèque : turtle
Utiliser une boucle pour dessiner une forme
Utiliser des pivotements et des fonctions forward et backward
Arrêter la tortue de dessiner et la faire recommencer

Tu vas apprendre dans ce chapitre à utiliser le module turtle de Python. Tu vas tracer quelques traits simples à l'aide des pivotements left (à gauche) et right (à droite), ainsi que des fonctions forward (en-avant) et backward (en arrière). Tu vas voir comment faire cesser la tortue de dessiner avec up et recommencer à dessiner avec down. Tu vas également découvrir que la tortue pivote en degrés.

Sauf si indiqué, il est plus facile de créer les programmes avec Notepad++ et ensuite de les exécuter avec l'IDLE Python.

1. Utiliser le module turtle de Python

Un module en Python est une manière de fournir du code de programmation pour l'utiliser dans un autre programme car il contient des fonctions que l'on peut réutiliser.

Le module turtle sert à programmer des graphismes vectoriels basés sur des lignes, des points et des courbes.

Tape ceci dans l'IDLE :

```
>>> import turtle
```

L'importation d'un module indique à Python que tu veux l'utiliser.

2. Tortue tortueuse

Crée ce programme avec Notepad ++, enregistre-le et exécute le à partir de l'IDLE Python (Ex23).

```
1  from turtle import *
2  forward(200)
```

Ce programme dessine une flèche vers la gauche.

Ajoute ces lignes à ce programme.

```
1  from turtle import *
2  forward(200)
3  right(90)
4  forward(200)
5  right(90)
```

Après exécution du programme, analyse les modifications à l'écran et essaie maintenant de dessiner un carré complet.

```
1  from turtle import *
2  forward(200)
3  right(90)
4  forward(200)
5  right(90)
6  forward(200)
7  right(90)
8  forward(200)
9  right(90)
```

3. Boucles et formes

Introduisons maintenant une boucle pour dessiner une forme (Ex24).

```
1  from turtle import *
2  for n in range(0,4):
3      forward(200)
4      right(90)
```

La forme dessinée à l'aide du programme Ex23 et Ex24 est identique. Mais tu peux remarquer qu'en utilisant une boucle, le programme devient plus court.

4. Exercices

Teste toutes ces boucles et enregistre tes travaux :

Ex25 :

```
from turtle import *
for n in range(0,6):
    forward(200)
    right(60)
```

Ex26 :

```
from turtle import *
for n in range(0,8):
    forward(100)
    right(45)
```

Ex27 :

```
from turtle import *
for n in range(0,5):
    forward(200)
    right(72)
```

Ex 28 :

```
from turtle import *
for n in range(0,5):
    forward(200)
    right(144)
```

Chapitre 9 : Mode aléatoire

Objectif(s) des exercices de ce chapitre.

À la fin de ce chapitre, l'élève sera capable de :

Comprendre et savoir utiliser la commande Random (nombre aléatoire)

1. Nombre au hasard

Pour obtenir un résultat au hasard par exemple pour un lancer de dé, on utilise en informatique le mode aléatoire à l'aide de la commande Random.

```
1   from random import *
```

random : recherche d'un nombre au hasard
import * : chercher des commandes dans la bibliothèque.

Affiche maintenant un nombre aléatoire entre 1 et 6 à l'aide de ce programme (Ex29) :

```
1   from random import *
2   print(randint(1,6))
```

2. Pile ou face

Tirage du pile ou face à l'aide de ce programme (Ex30) :

```
1   from random import *
2   pièce=["pile","face"]
3   print(choice(pièce))
```

choice : permet à Python de choisir un mot au hasard dans une liste "pièce"

3. Faire un sandwich

Dans cet exercice, il faut créer deux listes d'ingrédients et ensuite faire choisir au hasard un ingrédient à partir de chaque liste (Ex31).

```
1  from random import *
2  f1=["fromage","oeuf","confiture"]
3  f2=["carotte","salade","oignon"]
4  for s in range(0,10):
5      print("Vos 10 sandwichs seront à base de ",choice(f1)," et de",choice(f2))
```

Chapitre 10 : Les fonctions

Objectif(s) des exercices de ce chapitre.

À la fin de ce chapitre, l'élève sera capable de :

Rendre utilisable des extraits de fonctions
Utiliser des fonctions situées dans des modules
Contrôler la visibilité intérieure et extérieure des fonctions
Comprendre et savoir utiliser le mot-clé def
Importer des modules pour pouvoir en utiliser le contenu.

Ce que tu vas apprendre dans ce chapitre. Rendre utilisable des extraits des fonctions et utiliser des fonctions présentes dans des modules. Tu apprendras aussi que la portée des variables contrôle leur visibilité à l'intérieur et à l'extérieur des fonctions. Tu apprendras que la création des fonctions se repose sur le mot-clé def. Tu vas apprendre à importer des modules pour pouvoir en utiliser le contenu.

1. Utiliser des fonctions range et list

Ces fonctions servent à compter des éléments.

Écris-moi ce code dans l'IDLE Python.

```
>>> list(range(0,5))
[0, 1, 2, 3, 4]
```

Explication : Créer une liste de chiffres allant de 0 à 4. Il y a 5 chiffres dans la liste.

Autre code à taper dans l'IDLE :

```
>>> list(range(0,1000))
```

Les fonctions sont des extraits de code qui indiquent à Python d'effectuer quelque chose. Utilise des fonctions dans tes programmes le plus souvent que tu peux.

2. Qu'est-ce qu'une fonction ?

Une fonction possède trois parties : un nom, des paramètres et un corps.

Voici un exemple que tu peux essayer (IDLE):

Nom de la fonction foncttest. Elle possède un seul paramètre, mon_nom.

Son corps est constitué du bloc de code qui suit la ligne commençant par def.

Un paramètre est une variable qui n'existe que tant que la fonction est utilisée.

Pour utiliser la fonction, il suffit d'appeler son nom, avec des parenthèses autour de la valeur du paramètre.

Voici le code à introduire :

```
>>> def foncttest(mon_nom):
        print('Bonjour %s' % mon_nom)


>>> foncttest('Blaise')
Bonjour Blaise
```

Une fonction peut accepter 2, 3 ou n'importe quel nombre de paramètres.

Tape-moi ce code dans l'IDLE :

```
>>> def foncttest(pnom, nom):
        print('Bonjour %s %s' % (pnom, nom))


>>> foncttest('Blaise','Pascal')
Bonjour Blaise Pascal
```

Nous pouvons créer des variables pour les utiliser comme paramètres dans l'appel de fonction :

```
>>> prenom = 'Gustave'
>>> nomfam = 'Flaubert'
>>> foncttest (prenom, nomfam)
Bonjour Gustave Flaubert
```

Une fonction peut servir à calculer :

```
>>> def epargne(argent_poche, petits_boulots, dépenses):
        return argent_poche + petits_boulots - dépenses

>>> print(epargne(10,10,5))
15
```

Cette fonction prend trois paramètres, additionne les deux premiers et soustrait le troisième de la réponse.

3. Variables et portée

Une variable de ce type n'existe qu'à l'intérieur de la fonction.

Dans le monde de ma programmation, cela s'appelle la **portée** de la variable.

Tape cet exemple dans l'IDLE :

```
>>> def test_variable():
        premiere_variable = 10
        seconde_variable = 20
        return premiere_variable * seconde_variable

>>> print(test_variable())
200
```

Vous ne pouvez pas afficher la valeur des variables en dehors du bloc de code.

Créons une fonction pour montrer le nombre de canettes aplaties au fur et à mesure des semaines pendant un an. Elle prend en paramètre le nombre de canettes déjà aplaties :

Essaie ce code :

```
>>> def construction_vaisseau(canettes):
        total_canettes = 0
        for semaine in range(1,53):
                total_canettes = total_canettes + canettes
                print('Semaine %s = %s canettes' % (semaine, total_canettes))


>>> construction_vaisseau(2)
Semaine 1 = 2 canettes
Semaine 2 = 4 canettes
Semaine 3 = 6 canettes
Semaine 4 = 8 canettes
Semaine 5 = 10 canettes
Semaine 6 = 12 canettes
Semaine 7 = 14 canettes
Semaine 8 = 16 canettes
Semaine 9 = 18 canettes
Semaine 10 = 20 canettes
Semaine 11 = 22 canettes
```

Et ainsi de suite jusqu'à la semaine 52.

A la première ligne du corps de la fonction, nous créons une variable total_canettes initialisée à 0. Nous débutons ensuite une boucle pour les semaines de l'année, puis nous additionnons le nombre de canettes aplaties chaque semaine, pour l'afficher. Ce bloc de code forme le corps de la fonction, mais il y a un autre bloc de code dans cette fonction : les deux dernières lignes forment le bloc de la boucle for.

4. Utiliser des modules

Un module sert à regrouper des fonctions, des variables et d'autres choses dans des programmes plus vastes et plus puissants.

Calcul de la date et l'heure actuelles à l'aide d'un module intégré time :

Écris cet exemple dans l'IDLE :

```
>>> import time
>>> print(time.asctime())
Tue Dec 13 16:59:55 2016
```

La commande Import indique à Python que nous voulons utiliser la module time.

Asctime renvoie la date et l'heure actuelle en anglais.

Module utilisé pour interagir avec le système Python.

Tape ce code :

```
>>> import sys
>>> print(sys.stdin.readline())
```

Stdin est un objet spécial fournissant une fonction lisant le contenu d'une ligne de texte saisie eu clavier.

Tape maintenant ce code. Tu vas donner la valeur à la variable. Elle n'est plus fixée dans le programme.

Introduis ces lignes :

```
>>> def age_blague_idiote(age):
        if age >= 10 and age <= 13:
                print('Que donnent 13 + 49 + 84 + 155 + 97 ? La migraine !')
        else:
                print('Pardon ?')


>>> age_blague_idiote(9)
Pardon ?
>>> age_blague_idiote(12)
Que donnent 13 + 49 + 84 + 155 + 97 ? La migraine !
```

Ça fonctionne. La fonction va demander dans l'exercice suivant l'âge de la personne. Pour l'instant, elle était introduite entre ().

Et pour essayer appelle la fonction sans paramètre.

```
>>> def age_blague_idiote():
        print('Quel âge as-tu ?')
        age = int(sys.stdin.readline())
        if age >= 10 and age <= 13:
                print('Que donnent 13 + 49 + 84 + 155 + 97 ? La migraine !')
        else:
                print('Pardon ?')


>>> age_blague_idiote()
Quel âge as-tu ?
14
Pardon ?
```

Chapitre 11 : Classes et objets

Objectif(s) des exercices de ce chapitre.

À la fin de ce chapitre, l'élève sera capable de :

Apprendre à utiliser des classes
Créer des catégories
Construire des objets ou instances
Exécuter des fonctions d'un objet
Comprendre que les variables d'objet stockent des valeurs dans ces mêmes objets
Utiliser le paramètre self dans des fonctions (faire référence à d'autres fonctions et variables)

Dans ce chapitre tu vas apprendre à utiliser des classes pour créer des catégories de choses et construire des objets ou instances de ces classes.

Les enfants des classes héritent les fonctions de leurs parents et ne sont pas des clones. Par exemple un objet girafe peut posséder son propre nombre de taches.

Tu vas apprendre à exécuter des fonctions d'un objet et tu vas comprendre que les variables d'objet permettent de stocker des valeurs dans ces mêmes objets.

Tu vas utiliser le paramètre self dans des fonctions pour faire référence à d'autres fonctions et variables.

1. Organiser les choses en classe

En Python, les objets sont définis par des classes, qui servent à les ordonner dans des groupes.

Les choses	
Les inanimés	Les animés
Les trottoirs "par exemple"	Les animaux
	Les mammifères
	Les girafes

Définissons en premier lieu la classe Choses dans l'IDLE Python

```
>>> class Choses:
        pass
```

La class est nommée Choses.

Pass indique à Python que nous n'ajoutons pas d'autre information à son propos.

Appuyer 2 fois sur Enter entre chaque création de classes.

2. Enfants et parents

Si une classe A fait partie d'une classe B, on dit que A est un enfant de B et que B est un parent de A.

Dans le tableau ci-dessus, on dit que la classe au-dessus d'une autre est son parent, tandis que celle en-dessous est son enfant.

Attention respecter les majuscules et minuscules et le "é" est transformé en "e". Je n'insère pas ici le déterminant "les"

```
>>> class Inanimes(Choses):
        pass

>>> class Animes(Choses):
        pass
```

Ce que tu vois entre parenthèse est son parent.

Donc Choses est le parent de Inanimes et Animes.

On continue …

```
>>> class Trottoirs(Inanimes):
        pass
```

Essayer sans regarder les lignes de codes ci-après de trouver les classes Animaux, Mammifères, Girafes avec leurs parents respectifs.

```
>>> class Animaux(Animes):
        pass

>>> class Mammifères(Animaux):
        pass

>>> class Girafes(Animaux):
        pass
```

3. Ajouter des objets aux classes

Nous avons par exemple une girafe nommée Régine qui appartient à la classe Girafes. Régine sera un objet de la classe Girafes ou une instance.

Voici le code en Python à introduire :

```
>>> regine = Girafes()
```

Pour l'instant aucun paramètre entre les parenthèses.

Pour l'instant l'objet regine ne sert pas à grand-chose. Nous verrons plus tard que nous pouvons lui donner des fonctions qu'ils pourront utiliser.

4. Définir des fonctions de classes

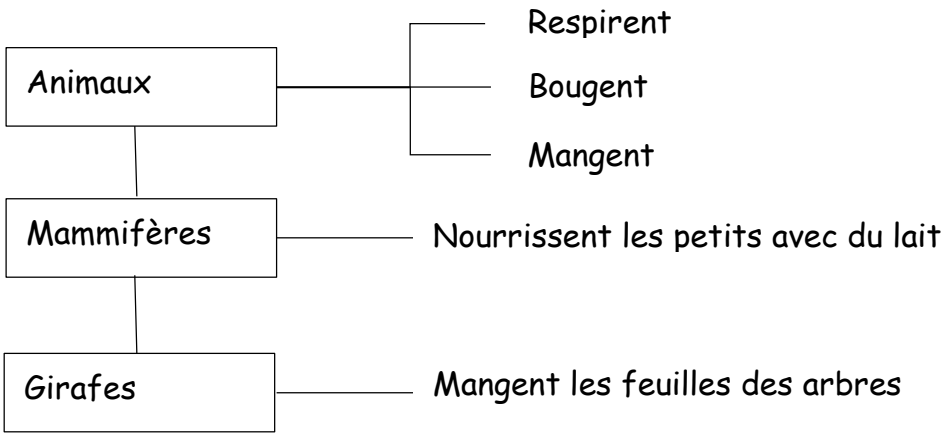
Voici un exemple d'une définition d'une fonction associée à une classe :

```
>>> class JeSuisUneClasse:
        def voici_une_fonction_de_Classe():
                print("Je suis une fonction de classe.")
```

Examine la syntaxe mais tu ne dois pas introduire ce code dans l'IDLE.

5. Ajouter des caractéristiques à une classe avec des fonctions

Nous allons ajouter des caractéristiques aux classes filles de la classe Animes définies précédemment. Nous allons décrire ce qu'elles sont et ce qu'elles savent faire.



Ces caractéristiques sont des actions ou fonctions.

Pour ajouter une fonction à une classe, nous utilisons le mot-clé `def`.

Comme ceci :

```
>>> class Animaux(Animes):
        def respirer(self):
                pass
        def bouger(self):
                pass
        def manger(self):
                pass
```

Le paramètre `self` permet qu'une fonction d'une classe puisse en appeler une autre.

Il est possible d'ajouter des fonctions aux deux autres classes : Mammifères et Girafes.

Essaie d'ajouter les fonctions sans regarder la solution à la page suivante.

Les espaces doivent être remplacés par des underscores _.

```
>>> class Mammifères(Animaux):
        def nourrir_petits_avec_du_lait(self):
                pass


>>> class Girafes(Mammifères):
        def manger_feuilles_des_arbres(self):
                pass
```

6. Mais pourquoi utiliser des classes et des objets

Nous allons pour bien comprendre reprendre l'exemple de régine la girafe.

Comme régine est un objet, nous pouvons appeler les fonctions fournies par sa classe (Girafes) mais aussi par ses classes parentes.

Il faut pour cela utiliser l'opérateur point (.) et le nom de la fonction.

Comme exemple nous allons dire à régine la girafe de manger et de bouger.

Voici et écris ce code :

```
>>> regine = Girafes()
>>> regine.bouger()
>>> regine.manger_feuilles_des_arbres()
```

Régine a trouvé un copain nommé Jules que nous allons créer.

```
>>> jules=Girafes()
```

Et il va bouger :

```
>>> jules.bouger()
```

Modifions un peu les classes pour que cela devienne plus évident. Ajoutons des instructions print à chaque fonction, à la pace de pass.

```
>>> class Animaux(Animes):
        def respirer(self):
                print("respire")
        def bouger(self):
                print("bouge")
        def manger(self):
                print("mange")

>>> class Mammiferes(Animaux):
        def nourrir_petits_avec_du_lait(self):
                print("nourrir les petits avec du lait")


>>> class Girafes(Mammiferes):
        def manger_feuilles_des_arbres(self):
                print("mange des feuilles des arbres")
```

Maintenant quand nous créons nos deux objets regine et jules, puis appelons des fonctions au départ d'eux, quelque chose se passe :

```
>>> regine = Girafes()
>>> jules = Girafes()
>>> jules.bouger()
bouge
>>> regine.manger_feuilles_des_arbres()
mange des feuilles des arbres
```

Aux deux premières lignes, nous établissons les variables regine et jules, objets de la classe Girafes. Ensuite, nous appelons la fonction bouger à partir de l'objet jules, et Python affiche bouge à la ligne suivante.

De la même manière, nous appelons la fonction manger_feuilles_des_arbres de regine, et Python affiche mange des feuilles des arbres.

Si c'étaient de vraies girafes et non de simples objets dans un ordinateur, l'une d'elles marcherait et l'autre mangerait des feuilles en haut d'un arbre.

7. Objets et classes en images

Approche graphique des objets et des classes ?

Reprenons le module turtle.

Lorsque nous écrivons turtle.Pen(), Python crée un objet de la classe Pen, fourni par le module turtle.

Nous allons établir deux objets tortues que nous allons nommer Aline et Karine (comme pour les 2 girafes).

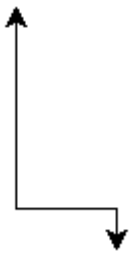
Introduisez ce code dans l'IDLE afin de créer deux objets (tortues) :

```
>>> import turtle
>>> aline = turtle.Pen()
>>> karine = turtle.Pen()
```

Appelons maintenant des fonctions sur chacun d'eux, pour les faire dessiner indépendamment.

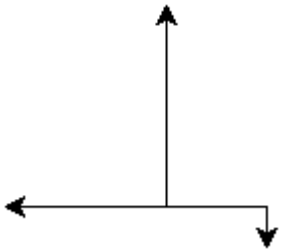
Introduis ce code :

```
>>> aline.forward(50)
>>> aline.right(90)
>>> aline.forward(20)
>>> karine.left(90)
>>> karine.forward(100)
```



Ajoutons une autre tortue : jacques. Et donnent lui un mouvement.

```
>>> jacques = turtle.Pen()
>>> jacques.left(180)
>>> jacques.forward(80)
```



Retiens que chaque fois que tu appelles turtle.Pen(), tu crées un nouvel objet indépendant des autres.

8. Fonctions appelant d'autres fonctions

Pour qu'une fonction de la classe `Girafes` appelle `bouger`, qui appartient à cette même classe, il faut se servir du paramètre `self` qui sert à appeler une autre fonction de la même classe (ou de ses parents).

Ajoutons une fonction nommée `trouver_nourriture` à la classe Girafes.

```
>>> class Girafes(Mammiferes):
        def trouver_nourriture(self):
                self.bouger()
                print("J'ai trouvé de la nourriture !")
                self.manger()
```

Tu viens de créer une fonction qui en combine deux autres. On emploie cela souvent en programmation.

Tape-moi maintenant ce code :

Tu vas en quelque sorte appeler une fonction unique qui effectue plusieurs choses. Ainsi que nous appelons `danser_une gigue`, notre girafe bouge 4 fois.

```
>>> class Girafes(Mammiferes):
        def trouver_nourriture(self):
                self.bouger()
                print("J'ai trouvé de la nourriture !")
                self.manger()
        def manger_feuilles_des_arbres(self):
                self.manger()
        def danser_une_gigue(self):
                self.bouger()
                self.bouger()
                self.bouger()
                self.bouger()
```

Que verras-tu si tu lances ensuite cette ligne de code ?

```
>>> regine.danser_une_gigue()
```

9. Initialiser un objet

Initialiser un objet revient à le préparer pour qu'il soit prêt à l'emploi.

Exemple :

Définition du nombre de taches pour chaque objet girafe au moment de son initialisation.

Il faut créer une fonction qui servira à définir les propriétés d'un objet au moment de sa toute première création.

Attention : deux caractères de soulignements de chaque côté de la fonction init.

Voici comment elle s'écrit :

```
>>> class Girafes:
        def __init__(self, taches):
                self.taches_girafe = taches
```

Ces lignes peuvent se lire comme suit : prendre la valeur du paramètre taches et la stocker pour un usage ultérieur dans la variable objet taches_girafe.

Établissons maintenant deux objets girafes : Oscar et Gertrude et affichons leur nombre de taches.

```
>>> oscar = Girafes(100)
>>> gertrude = Girafes(150)
>>> print(oscar.taches_girafe)
100
>>> print(gertrude.taches_girafe)
150
```

Nous créons une instance de la classe Girafes avec la valeur de paramètre 100. La fonction __init__ de la classe est appelée automatiquement et la valeur 100 est utilisée comme paramètre. Idem pour la seconde girafe mais avec comme valeur 150.

Nous affichons ensuite les résultats.

10. Puzzle de programmation

Moulin de Girafe

Fourche de tortue

Chapitre 12 : Fonctions intégrées de Python

Objectif(s) des exercices de ce chapitre.

À la fin de ce chapitre, l'élève sera capable de :

-
-
-
-
-
-

jhky

1.

En Python, les objets sont

Bibliographie

- Apprendre à coder niveau 3 – Vigot
- Tangente Éducation : Spécial programmation n°15
- Python pour les Kids - Eyrolles