

Cours Python

Partie 1 : les bases

Bases de Python

I Pour démarrer

```
# Un premier programme
nom = input("Entrez votre nom: ")
print("Bonjour", nom)
```

Ce qui suit le caractère # est un commentaire du programme.

la fonction `input()` permet de demander à l'utilisateur d'entrer une chaîne de caractères.

La fonction `print()` permet d'afficher du contenu.

A Opérateurs mathématiques

Addition	soustraction	multiplication	division	puissance	reste	quotient entier
+	-	*	/	**	%	//

Python respecte les priorités opératoires

Exemple

Mathématiques	Python
$3^2 - 7 \times 8$	<code>3 ** 2 - 7 * 8</code>
$\frac{3 - 7}{1 - 5^2}$	<code>(3 - 7) / (1 - 5 ** 2)</code>
Le reste de 13 divisé par 5	<code>13 % 5</code>
Le quotient de 13 divisé par 5	<code>13 // 5</code>

Par convention, pour plus de lisibilité, on laisse un espace autour des opérateurs.

B Types de base

- **Les entiers signés** `int` pour "integer" en anglais. Par exemple : 2, - 24, -3455.
- **Les nombres à virgule flottantes** `float` que l'on appellera les "flottants" pour "float" en anglais. C'est une partie des nombres réels. Le séparateur décimal est le point. Par exemple 1.2, -4.56e5, -4.0. Remarques : 3.1e4 = 31000.0 -4 est de type `int` et -4.0 est de type `float`.
- **Les chaînes de caractères** `str` pour "string" en anglais. Elles contiennent du texte délimité par des guillemets simples ou doubles. Par exemple : "Bonjour", 'Test'. On peut utiliser des caractères échappements 'Aujourd\'hui' permet de mettre une apostrophe dans une chaîne délimitée par une simple cote. \n permet de mettre un saut de ligne.

```
>>> print('un \n deux')
un
deux
```

- **Les booléens** 'bool' pour boolean en anglais prennent la valeur True ou False . Par exemple

```
>>> 2 > -6
True
>>> 4 < 4
False
```

Attention à ne pas oublier la majuscule à True et False.

C Variables

Pour stocker des valeurs on utilise des variables. En Python, une variable est un nom auquel Python associe un objet pour y stocker, entre autre, une ou des valeurs attribuées a cette variable et son type.

Les caractères autorisés pour un nom de variable sont les lettres, les chiffres et le tiret bas : "_" . De plus, un nom de variable ne peut ni commencer par un chiffre ni être un mot réservé de Python (comme print). Python est sensible à la casse, il différencie les majuscules des minuscules. nombre , Nombre et NOMBRE sont des variables différentes.

Exemple

- Noms de variables valides : joueur1, compteurDeVie, compteur_de_vie
- Noms de variables non valides :
 - 1ierJoueur (commence par un chiffre)
 - compteur-de-vie (contient un tiret)
 - continue (mot réservé Python)

Affecter une variable Pour affecter une valeur à une variable, on utilise le symbole = .

Exemple :

```
age = 15
nom = "Jules"
```

En pseudo code on écrit :

```
age ← 15
nom ← "Jules"
```

Une *expression* est une combinaison de valeurs, variables, opérateurs, et de fonctions qui sont évaluées par le langage. Elle fournit une valeur et son type qui peut être stocké dans une variable.

Par exemple :

```
>>> a = 2 + 5 # Calcule 2 + 5 puis l'affecte à la variable a avec le type
↪ int.
>>> a
7
>>> b = 5
>>> b = b * 2 # Calcule 5 * 2 puis affecte 10 à la variable b avec le type
↪ int.
>>> b
10
```

- La fonction `type` prend en argument une expression et retourne son type
- La fonction `isinstance` prend en argument une expression et un type retourne `True` si l'expression est du type donnée et `False` sinon

Exemple :

```
>>> type(2 + 2)
<class 'int'>
>>> type(2.0 + 1)
<class 'float'>
>>> a = "3"
>>> type(a)
<class 'str'>
>>> isinstance(a, int)
False
>>> isinstance(a, str)
True
```

Transtypage

Il existe des fonctions pour convertir une expression, lorsque c'est possible dans un autre type :

- `int` : convertit en entier
- `float` : convertit en flottant
- `str` : convertit en chaîne de caractères

Exemple

```
>>> a = '42'
>>> int(a)
42
>>> type(int(a))
<class 'int'>
>>> int(4.2)
```

```
4
>>> int('a')
ValueError: invalid literal for int() with base 10: 'a'
```

Attention, `int` tronque la partie décimale, ce n'est pas la partie entière au sens mathématique (floor) : pour un nombre négatif ce n'est pas la même chose. Par exemple `int(-4.7)` vaut -4 et non -5.

II Les fonctions

Vous avez déjà appelé des fonctions en utilisant les fonctions intégrées `print`, `input`, `round`, `type`, `isinstance`. Écrire une fonction permet de ne pas avoir à réécrire du code dont on va se resservir.

A Définir une fonction

Pour définir une fonction en Python, on utilise le mot-clé `def`. La syntaxe est la suivante ;

```
def nom_fonction(listeparametres): # Ne pas oublier les :
    bloc d'instructions # Indentation obligatoire
    return résultats
# On arrête l'indentation pour finir.
```

Remarques :

- Les paramètres sont séparés par une virgule
- La liste des paramètres peut être vide
- L'indentation correspond à 4 caractères espace.
- l'instruction `return` arrête l'exécution de la fonction et renvoie une ou plusieurs valeurs que l'on peut alors récupérer.
- S'il n'y a pas l'instruction `return` alors `None` est renvoyé et l'on parle de procédure.

Exemple

La fonction qui convertit un temps donné en heures et minutes, en minutes, s'écrit en pseudo code :

```
fonction conversion_minutes(heures, minutes)
    minutes_totales <- heures * 60 + minutes
    renvoyer minutes_totales
```

En Python :

```
def conversion_minutes(heures, minutes):
    minutes_totales = heures * 60 + minutes
    return minutes_totales
```

B Appel d'une fonction

Pour utiliser une fonction, il faut l'appeler ("call" en anglais) et lui passer la valeur des arguments dans l'ordre de sa définition et dans le type attendu.

Exemple

Pour connaître à combien de minutes correspond 2h30, il suffit de faire l'appel

`conversion_minutes(2,30)` qui renvoie 150. On dit que l'on a fait l'appel de la fonction `conversion_minutes` pour les valeurs 2 et 30.

C docstring

Une docstring pour "documentation string" (chaîne de caractères de documentation) permet de mettre la documentation d'une fonction. Elle est indentée et encadrée par des triples guillemets.

Exemple

```
def prix_ttc(ht, tva):  
    """  
    Calcul un prix TTC à partir du prix HT et du montant de la TVA.  
    Entrées :  
    - ht un nombre  
    - tva un nombre  
    Sortie:  
    - float le prix TTC  
    """  
    ttc = ht * ( 1 + tva / 100)  
    return ttc
```

Après exécution de ce code, on peut récupérer ces informations à l'aide de la fonction `help`

Exemple

```
>>> help(prix_ttc)  
Help on function prix_ttc in module __main__:  
  
prix_ttc(ht, tva)  
    Calcul un prix TTC à partir du prix HT et du montant de la TVA.  
    Entrées :  
    - ht un nombre  
    - tva un nombre  
    Sortie:  
    - float le prix TTC
```

III Opérateurs de comparaison et opérateurs logiques

A Opérateurs de comparaison

Ces opérateurs comparent deux expressions et retournent le booléen True si l'évaluation des expressions vérifie la relation et False sinon.

Égalité	Différent de	Supérieur	Supérieur ou égal	inférieur	inférieur ou égal
==	!=	>	>=	<	<=

Remarque

Attention à ne pas confondre l'affectation = et la comparaison ==.

Exemples

```
>>> 2 * 5 == 15 - 5
True
>>> 2 + 2 == 5
False
>>> 2 != 5
True
>>> 2 + 2 != 4
False
>>> 2 > 5
False
>>> 9 - 2 > 4
True
>>> 5 >= 5
True
>>> 9 >= 4
True
>>> 2 < 5
True
>>> 9 - 2 < 4
False
>>> 2 <= 5
True
>>> 4 <= 4
True
```

B Opérateurs logiques

Les principaux opérateurs logiques sont :

- and : “et” retourne True uniquement si les deux expressions sont vraies.
- or : “ou inclusif” retourne True si l’une des expressions est vraie et faux sinon.

- not : négation. Il retourne True si l'expression est évaluée à False et False sinon.

Pour résumer leur fonctionnement dans une table de vérité. Une table de vérité comprend tous les cas possibles.

La table de vérité de l'opérateur and

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

La table de vérité de l'opérateur or

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

La table de vérité de l'opérateur not

a	not(a)
False	True
True	False

Exemples

```
>>> (2 < 3) and (4 < 5)
True
>>> (4 == 5) and ( 2 + 2 == 4)
False
>>> (2 == 1 + 1) or (3 != 3)
True
>>> (5 != 5) or (3 != 3)
False
>>> not (5 == 4)
True
```

IV Préconditions, jeu de tests et variables locales/globales

A Préconditions

Lorsque l'on écrit une fonction, il faut veiller à ce que les arguments donnés soient cohérents. Par exemple, il faut être certain que l'on ne donne pas un nombre négatif pour une donnée qui représente une longueur et doit donc être positive. C'est ce qu'on appelle une précondition.

Pour cela on utilise le mot-clé `assert` dans le corps de la fonction. Sa syntaxe est :

```
assert condition
```

Si la condition retourne `True` le programme continue, sinon le programme s'arrête et une exception est levée, un message `AssertionError` est retournée.

On peut aussi utiliser la syntaxe :

```
assert condition , chaineExplicitantLErreur
```

Exemple

La précondition pour utiliser la fonction `est_pair` est que la valeur donnée en paramètre est un entier.

```
def est_pair(n ):
    """
    Retourne True si n est pair et False sinon
    entrée: n un entier
    sortie : un booléen
    """
    assert isinstance(n,int) , "n doit être un entier"
    return n % 2 == 0
```

Après exécution de ce code, dans la console, on aura :

```
>>> est_pair(4)
True
>>> est_pair('4')
AssertionError: n doit être un entier
```

Cela permet d'être certain que lors de son appel la fonction les paramètres soient ceux attendus

B Jeu de tests

En programmation, lorsque cela est possible, pour chaque fonction, on fournit un jeu de tests qui appelle la fonction et vérifie qu'elle retourne ce qui est attendu.

Cela permet d'éviter des erreurs par la suite et lire les tests ou les créer permet de mieux comprendre ce que cette fonction fait.

Pour cela on utilise le mot-clé `assert`

Exemple

```
def minutes(h, m):  
    nb_minutes = h*60 + m  
    return nb_minutes  
### Jeu de tests  
assert minutes(3, 17) == 197  
assert minutes(10, 32) == 632  
assert minutes(0, 41) == 41  
assert not minutes(1,0) == 1
```

Lors de l'exécution du programme, si l'un des test échoue une erreur est levée, sinon c'est que les tests ont réussi.

C Variables locales et globales

Une **variable locale** est une variable définie à l'intérieur d'une fonction. Elle n'est accessible qu'à l'intérieur de cette fonction.

Une **variable globale** est une variable accessible dans tout le programme. Dans une fonction, si elle n'est pas en paramètre elle n'est accessible qu'en lecture et ne peut être modifiée.

Exemples

```
def test():  
    a = 0  
    return a  
print(a)
```

La variable `a` est locale et n'est pas accessible en dehors de la fonction. Ce programme retourne :

NameError: name 'a' is not defined

```
a = 3  
def test():  
    return a + 1  
print(test())
```

La variable `a` est globale et donc accessible en lecture dans la fonction.

Ce programme retourne 4.

```
a = 3
def test():
    a = a + 1
    return a
test()
```

La variable a est globale mais ne peut être modifiée dans la fonction. Le programme retourne le message d'erreur :

UnboundLocalError: cannot access local variable 'a' where it is not associated with a value

L'erreur montre que Python attendais qu'il y ait une variable locale a dans la fonction.

```
a = 3
def test(a):
    a = a + 1
    return a
print(test(a),a)
```

Ce programme affiche 4 3 la variable a globale n'est pas changée. Lorsqu'elle est passée en paramètre. C'est une autre variable locale nommée a qui est affectée de la valeur 3 puis incrémenté de 1.

```
a = 0
def test():
    global a
    a = a + 1
    return a
print(test(),a)
```

Ce programme affiche 1 1 Le mot clé `global` permet de modifier la valeur de la variable globale a à l'intérieur de la fonction. Modifier une variable globale dans une fonction est une pratique risquée et sauf exception vous ne devrez pas utiliser le mot clé `global` .

V Opérateurs sur les chaînes de caractères

opérande1	opérateur	opérande2	nom de l'opération	exemple	résultat
str	+	str	concaténation	"bon" + "jour"	"bonjour"
str	*	int	répétition	"Aie" * 3	"AieAieAie"
int	*	str	répétition	3 * "Aie"	"AieAieAie"
str	in	str	est dans	"a" in "blabla"	True
str	in	str	est dans	"e" in "blabla"	False
str	not in	str	n'est pas dans	"a" not in "blabla"	False
str	not in	str	n'est pas dans	"e" not in "blabla"	True

VI Structures d'embranchement

Les instructions d'embranchement permettent de **sélectionner le bloc d'instructions à exécuter** selon la valeur d'une **expression booléenne**.

A Embranchement simple

La structure est la suivante :

```
if condition vraie: # ne pas oublier les :  
    bloc d'instructions # indentation obligatoire  
# pour terminer, on stoppe l'indentation
```

Exemple

Si le prix d'entrée dans un musée dépend de l'âge on pourrait avoir la règle :

si l'âge est supérieur à 16 ans **alors** le prix du billet est 13 €.

```
def prix_musee(x):  
    if x >= 16:  
        return 13
```

- Lors de l'appel `prix_musee(27)` :
 - Le paramètre `x` prend la valeur 27 ;
 - Le test `x >= 16` retourne `True` ;
 - On entre dans le "alors" qui est le bloc indenté et la fonction retourne 13
- Lors de l'appel `prix_musee(15)` :
 - Le paramètre `x` prend la valeur 15 ;
 - Le test `x >= 16` retourne `False`
 - On sort de la structure "si ...alors ..." et la fonction retourne `None`.

B Embranchement simple avec alternative.

La structure est la suivante :

```
if condition vraie: # ne pas oublier les :  
    bloc d'instructions 1 # indentation obligatoire  
else: # ne pas oublier les :  
    bloc d'instructions 2 # indentation obligatoire  
# pour terminer, on stoppe l'indentation
```

Exemple

Si le prix d'entrée dans un musée dépend de l'âge on pourrait avoir la règle :

si l'âge est supérieur à 16 ans **alors** le prix du billet est 13 € **sinon** c'est gratuit.

```
def prix_musee(x: int) -> int:
    if x >= 16:
        return 13
    else:
        return 0
```

- Lors de l'appel `prix_musee(27)` :
 - Le paramètre `x` prend la valeur 27 ;
 - Le test `x >= 16` retourne `True` ;
 - On entre dans le "alors" qui est le bloc indenté et la fonction retourne 13
- Lors de l'appel `prix_musee(15)` :
 - Le paramètre `x` prend la valeur 15 ;
 - Le test `x >= 16` retourne `False` ;
 - On entre dans le "sinon" et la fonction retourne 0.

C Embranchements multiples

La syntaxe est la suivante :

```
if condition 1 vraie: # ne pas oublier les :
    bloc d'instructions 1 # indentation obligatoire
elif condition 2 vraie: # ne pas oublier les :
    bloc d'instructions 2 # indentation obligatoire
else: # ne pas oublier les :
    bloc d'instructions 3 # indentation obligatoire
# pour terminer, on stoppe l'indentation
```

On peut mettre plusieurs `elif` de suite (contraction de `else if` en anglais)

Exemple

si l'âge est supérieur à 18 ans **alors** le prix du billet est 13 €.

sinon si l'âge est supérieur à 16 ans **alors** le prix du billet est 9 €.

sinon le billet est gratuit

On programme une fonction `prix_billet` qui va calculer le prix du billet en fonction de l'âge.

L'âge doit être positif. On obtient :

```
def prix_billet(age):
    assert age >= 0, "L'age doit être positif"
    if age >= 18:
        prix = 13
    elif age >= 16:
```

```
prix = 9
else:
    prix = 0
return prix
```

Que l'on peut tester avec le jeu de tests suivant :

```
assert prix_billet(3) == 0
assert prix_billet(17) == 9
assert prix_billet(18) == 13
assert prix_billet(34) == 13
assert prix_billet(56) == 13
```

VII Boucles bornées (itératives)

En informatique comme dans la vie courante, il faut souvent répéter la même succession d'instructions.

Définition

Une boucle itérative permet de répéter un nombre fixé de fois une suite d'instructions.

La syntaxe en Python :

```
for compteur in range(debut, fin + 1): # Ne pas oublier les :
    bloc d'instructions
# Pour terminer on stop l'indentation
```

Remarques :

En Python, l'instruction `range(valeur initiale, valeur finale + 1)` définit une plage de valeurs entières consécutives commençant à `debut` et se terminant `fin + 1`.

Par exemple :

l'instruction `range(1,6)` correspond à la plage de valeurs 1, 2, 3, 4, 5.

l'instruction `range(0,3)` correspond à la plage de valeurs 0, 1, 2.

Dans le cas où `valeur initiale = 0`, on peut simplement écrire `range(valeur finale + 1)` au lieu de `range(0, valeur finale + 1)`.

Ainsi, `range(0,3)` est la même instruction que `range(3)`

Exemples

```
1. for i in range(3):
    print("Bonjour")
    print('Au revoir')
```

Affiche :

```
Bonjour  
Bonjour  
Bonjour  
Au revoir
```

2. L'exécution de ce programme :

```
for i in range(1,4):  
    print(i," fois 7 est égal à ",i * 7)
```

Affiche :

```
1 fois 7 est égal à 7  
2 fois 7 est égal à 14  
3 fois 7 est égal à 21
```

Précision sur la fonction range

On peut ajouter un troisième paramètre à la fonction range : le pas.

C'est un nombre entier qui donne l'intervalle entre les valeurs du compteur.

Ainsi range(1,10,2) correspond à la plage de valeurs : 1, 3, 5, 7, 9.

et range(10, -1, -3) correspond à la plage de valeurs 10, 7, 4, 1.

VIII Chaînes de caractères

Complément sur les chaînes de caractères.

Connaitre la longueur d'une chaîne de caractères len

La longueur d'une chaîne de caractères, c'est-à-dire le nombre de caractères qu'elle contient, s'obtient grâce à la fonction Python len de l'anglais length (longueur).

Exemple

```
>>> a = "test"  
>>> len(a)  
4
```

La longueur de la chaîne associée à a est 4

Indexation des caractères

Les éléments d'une chaîne de caractères sont ordonnés et associés à un indice qui commence à 0. Les indices vont donc de 0 à la longueur de la chaîne moins 1.

L'élément d'indice *i* d'une chaîne de caractères *ch* est défini par l'instruction *ch[i]*.

Remarque : Tenter d'accéder à un indice qui n'existe pas une erreur de type est levée :

`IndexError: string index out of range`

Exemple

```
>>> ch = "Bonjour"
>>> ch[0] # Retourne le premier élément
'B'
>>> ch[3] # Retourne le quatrième élément
'j'
>>> ch[len(ch)-1] # Retourne le dernier élément
'r'
>>> ch[7] # ch n'a que 7 caractères (indices 0 à 6) : erreur
IndexError: string index out of range
```

On peut aussi accéder aux éléments d'une chaîne *ch* en utilisant des indices négatifs : -1 correspond alors au dernier caractère, -2 à l'avant dernier caractère, jusqu'au premier élément d'indice `-len(ch)`

Exemple

```
>>> ch = "azerty"
>>> ch[-1] # Retourne le dernier élément
'y'
>>> ch[-3] # Retourne l'avant-avant dernier élément.
'r'
```

Le type chaîne de caractères est *immutable*, c'est-à-dire qu'il n'est pas modifiable. Ainsi, tenter de modifier un élément d'une chaîne lève une erreur de type.

Exemple

```
>>> ch = "tris"
>>> ch[0] = "c"
TypeError: 'str' object does not support item assignment
```

IX Boucles conditionnelles

Définition

Une boucle conditionnelle permet de répéter une séquence d'instructions tant qu'une condition est vérifiée.

La structure en Python est la suivante :

```
while condition vraie: # Ne pas oublier les :
    bloc d'instructions # Indentation obligatoire
# Pour terminer, on stoppe l'indentation
```

Remarques

- La boucle s'arrête lorsque la condition qui doit être vérifiée devient fausse ;
- Si la condition est toujours vraie, on a une *boucle infini* ce qui peut planter le programme.

Exemple

seuilPuiss2 est une fonction prenant en argument un entier n qui retourne le plus petit entier k tel que $2^k \geq n$.

Par exemple `seuilPuiss2(10)` doit retourner 4 car 2^4 est la plus petite puissance de 2 plus grande que 10 et que $2^3 = 8$ est plus petit que 10.

L'algorithme en pseudo code :

```
fonction seuilPuiss2(n:entier)
    k <- 0
    Tant que 2 ** k <= n
        k = k + 1
    Fin tant que
    Retourner k
```

En Python

```
def seuilPuiss2(n) :
    k = 0 # ICI
    while 2 ** k <= n: #LA
        k = k + 1
    return k
```

Trace de l'appel `seuilPuiss2(10)`.

Commentaire	k	2 ** k	2 ** k <= n
#ICI	0	—	—
#LA	0	1	True
#LA	1	2	True
#LA	2	4	True
#LA	3	8	True
#LA	4	16	False

La fonction retourne la dernière valeur associée à k soit 4 comme attendue.

L'instruction break

Elle permet une sortie anticipée d'une boucle, ce qui peut être utile dans certain cas. On peut aussi utiliser cette instruction dans une boucle for.

```
while True:
    a = input()

    if a == "q":
        print("Fin")
        break
    else:
        print("echo",a)
```

Ce programme demande à l'utilisateur de saisir une chaîne de caractères au clavier. Si cette chaîne est q le programme affiche Fin et s'arrête, sinon il affiche echo suivi de la chaîne saisie puis attend à nouveau une saisie par l'utilisateur.

Appendice A - Les modules

Les modules appelés bibliothèques ou librairies dans d'autres langages, permettent des extensions du langage en lui ajoutant des fonctions et des constantes.

Il y a :

- math pour les fonctions et constantes mathématiques ;
- random pour le hasard ;
- matplotlib.pyplot pour la création de graphique.

Il en existe de nombreux autres qui permettent par exemple le traitement d'images, le traitement de données, l'intelligence artificielle, aller chercher des informations sur le web, etc.

Pour utiliser un module il faut d'abord l'importer dans Python à l'aide du mot clé import suivi du nom du module.

Exemple

On importe la fonction racine carrée sqrt pour "square root" en anglais et affiche une valeur approchée de $\sqrt{2}$ et du nombre π .

```
import math
print("Racine carrée de 2 :",math.sqrt(2))
print('pi :',math.pi)
```

Pour ne pas écrire le nom du module en entier on peut utiliser un alias. En utilisant le mot-clé `as`.

```
import math as m
print("Racine carrée de 2 :",m.sqrt(2))
print('pi :',m.pi)
```

Lorsqu'on n'a besoin que de quelques éléments d'un module on peut décider de n'importer que ces éléments. On utilise la structure `from nom-module import liste des éléments` séparés par des virgules

```
from math import sqrt, pi
print("Racine carrée de 2 :",sqrt(2))
print('pi :',pi)
```

Exemple - module random

Le module `random` permet d'utiliser des fonctions qui génèrent du pseudo-hasard.

```
import random as rd
print(rd.randint(1,6)) # Affiche un nombre entier choisi au hasard
↳ compris entre 1 et 6
print(rd.random()) # Affiche un flottant dans l'intervalle [0; 1[
print(rd.choice([2,3,5,7,11,13])) # Affiche un nombre au hasard parmi les
↳ nombres entre crochets.
```

Table des matières

Bases de Python	1
I Pour démarrer	1
A Opérateurs mathématiques	1
B Types de base	1
C Variables	2
II Les fonctions	4
A Définir une fonction	4
B Appel d'une fonction	5
C docstring	5
III Opérateurs de comparaison et opérateurs logiques	6
A Opérateurs de comparaison	6
B Opérateurs logiques	6
IV Préconditions, jeu de tests et variables locales/globales	8
A Préconditions	8
B Jeu de tests	8
C Variables locales et globales	9
V Opérateurs sur les chaînes de caractères	10
VI Structures d'embranchement	11
A Embranchement simple	11
B Embranchement simple avec alternative.	11
C Embranchements multiples	12
VII Boucles bornées (itératives)	13
Précision sur la fonction range	14
VIII Chaînes de caractères	14
Connaitre la longueur d'une chaîne de caractères len	14
Indexation des caractères	14
IX Boucles conditionnelles	15
L'instruction break	17
Appendice A - Les modules	17
Exemple	17
Exemple - module random	18