Commissariat Général à l'Energie Atomique
Centre Régional d'Etudes Nucléaires de Kinshasa
C.G.E.A./C.R.E.N-K.

Premiers pas en Python

Prof. Dr Saint-Jean DJUNGU

Janvier 2024

Tél : +243 818899608 *email :saintjean.djungu.ahuka@cgea-rdc.org*

Chapitre 1 : Généralités sur Python

1.1. Introduction

1.1.1. Pourquoi Python ?

Python est un langage programmation de haut niveau basé sur le langage C. C'est un langage de programmation interprété, interactif et orienté objet. Il a été créé vers la fin des années 1980 par Guido van Rossum, aux Pays-Bas et a mis du temps à gagner ses adeptes.

Ces dernières années ont pourtant vu un boom dans le classement de Python, et cela en grande partie est due grâce à ses possibilités dans le Machine Learning, la science des données, et dans l'IA (Intelligence Artificielle).

Le nom Python vient d'un hommage à la série télévisée *Monty Python's Flying Circus* dont Guido van Rossum est fan.

Langage polyvalent de haut niveau, Python trouve des utilisations dans la création d'applications web, l'analyse de données et le développement d'algorithmes. Il possède une syntaxe facile à utiliser et met l'accent sur la lisibilité et la simplicité du code : cela en fait un candidat idéal pour les développeurs débutants, en particulier ceux qui souhaitent faire carrière dans l'IA.

Python a la réputation d'être très fiable et efficace, ce qui le rend populaire tout autant auprès des grandes entreprises que des start-ups. Plusieurs *bibliothèques Python* sont à la disposition des développeurs : ces bibliothèques permettent d'accélérer diverses tâches dans le domaine de la construction d'algorithmes d'IA et de la liaison aux API. De plus, il peut être utilisé pour automatiser certaines activités, notamment certaines tâches quotidiennes chronophages.

Enfin, Python peut également être utilisé pour le développement web. En effet, il est souvent utilisé pour le *web scraping* : quelque chose qui pourrait prendre des heures à coder en PHP ne prendra que quelques minutes avec Python.

La communauté très active de Python est la première responsable de la croissance de ce langage. Elle offre des conseils sur les bonnes pratiques et des solutions de dépannage, aux débutants comme aux experts.

Pour cette raison, Python est connu comme le « langage de programmation majeur à la croissance la plus rapide ». Avec des applications dans certaines des technologies en pleine croissance et parmi les plus passionnantes, les développeurs qui connaissent Python trouveront rapidement des débouchés dans le Big Data, l'IA, la robotique ou la cybersécurité. Ces technologies devenant omniprésentes, il peut valoir la peine d'acquérir des compétences en langage Python dès maintenant !

Google décrit Python comme son langage dynamique le plus important et a même développé son propre « *Python Style Guide* ». Même la NASA utilise Python : sur son site web *code.nasa.gov*, elle répertorie déjà 63 projets développés avec Python.

1.1.2. Caractéristiques de Python

Le langage Python offre plusieurs caractéristiques. C'est un langage facile à apprendre, facile à lire, facile à entretenir, portable et extensible. Il offre la possibilité d'interagir avec les bases de données, programmer des interfaces graphiques et est couramment utilisé pour la production de contenu HTML sur les sites Web.

Idéal pour les fichiers texte et utilise des types intégrés utiles (listes, dictionnaires). C'est un langage Multi-usager (Web, GUI, Scripting, etc.), fortement typé et typé dynamiquement, axé sur la lisibilité et la productivité.

Pas de délimiteurs de blocs de code d'instructions et assure une gestion automatique de la mémoire : Utilise le ramasse miette pour la gestion des objets en mémoire. Python est un langage interopérable (avec C *Cython*, Java *Jython*, C++, Fortran *F2Py*, …). Il possède un système d'exceptions.

Toutes ces caractéristiques font que Python est désormais enseigné dans de nombreuses formations, depuis l'enseignement secondaire jusqu'à l'enseignement supérieur.

1.2. Utilisation du Python

Une fois installé, Python peut être utilisé en deux modes : le mode interactif ou le mode script.

1.2.1. Mode interactif

Les instructions tapées dans l'interpréteur de la ligne de commande python (Python Shell) sont exécutées directement.

Vous pouvez le faire depuis Unix, DOS, ou tout autre système qui vous fournit une fenêtre de l'interpréteur de ligne de commande ou un shell.

>>>

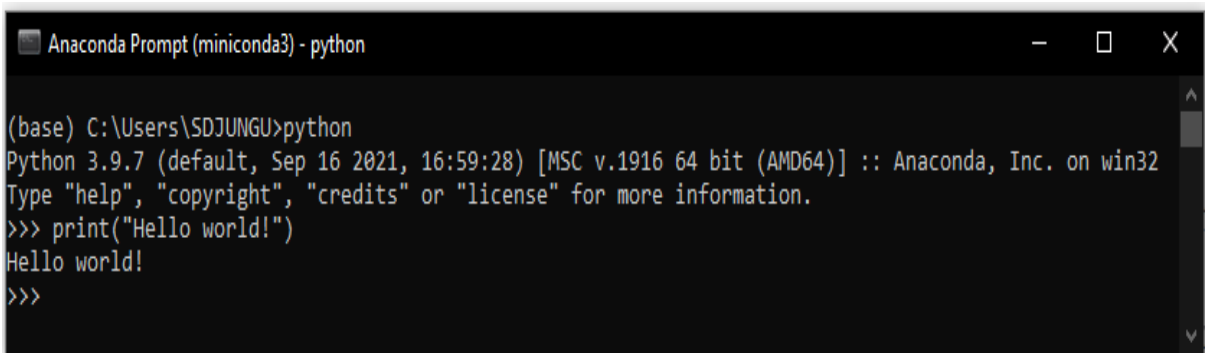
Le codage avec le shell est parfait pour l'apprentissage du langage et pour tester les modules.

Exemple : Ouvrez un shell puis lancez la commande : *python*

>>>print("Hello world!")

Hello world!

>>>



Documentation

Sous ce mode interactif, il est possible d'obtenir de la documentation en appelant la fonction *help()*, puis en entrant l'identifiant de la fonction ou de la méthode. La documentation complète du langage est disponible sur le réseau à *http://docs.python.org*.

1.2.2. Mode script

Un script Python peut être saisie dans un fichier d'extension *.py* et exécuté avec l'interpréteur python (*Python Shell*). Pour ce faire :

- Editez votre script avec n'importe quel éditeur de texte
- Ouvrez le python shell
- Ecrivez le nom de votre script
- Exécutez le script

Exemple : Ouvrez Notepad++ et entrez le code suivant :

print("Hello world!")

Enregistrez votre fichier sous le nom *test.py*, puis quittez l'éditeur de texte. Pour exécuter votre script, ouvrez un shell et entrez la commande :

python test.py

1.3. EDI de Python

Vous pouvez exécuter Python à partir d'une interface utilisateur graphique (GUI). Tout ce que vous avez besoin est une application GUI sur votre système qui prend en charge Python.

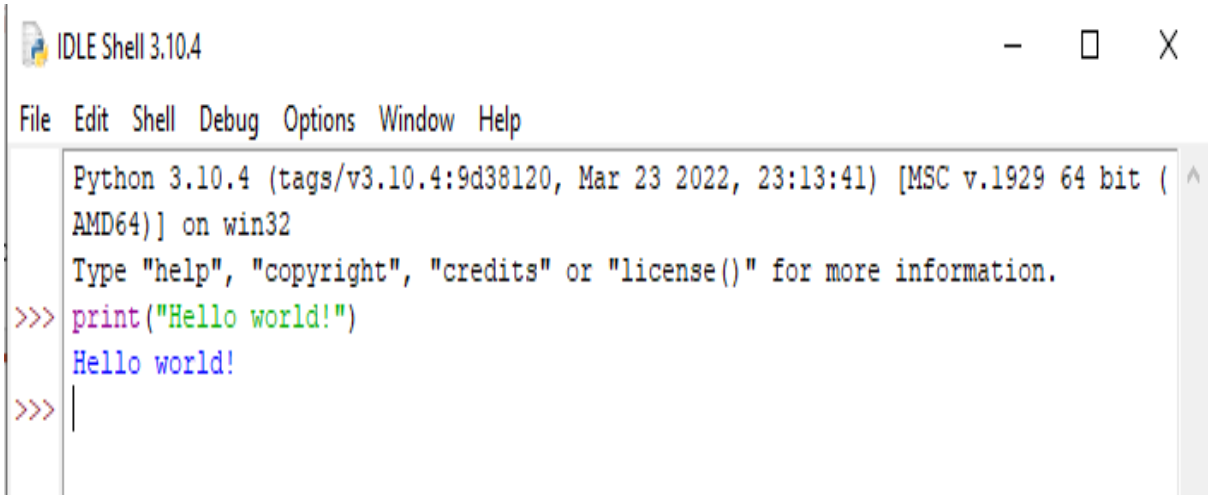
1.3.1. IDLE

IDLE est un environnement de développement intégré pour le langage Python. Intégralement écrit avec Python et la bibliothèque graphique *Tkinter* (*Tool kit interface*).

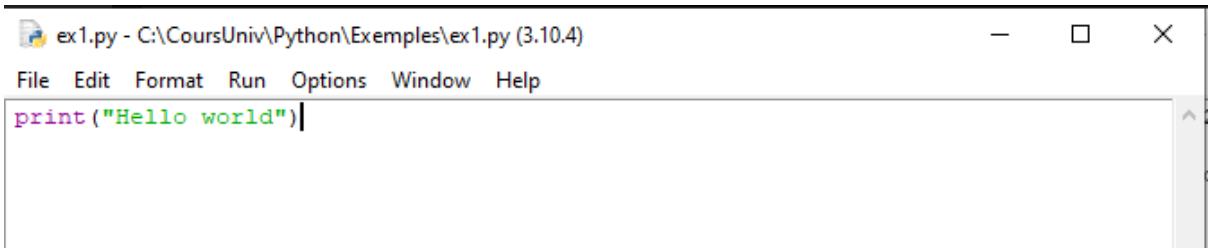
1.3.2. Fonctionnalités de IDLE

IDLE n'est rien d'autre qu'un éditeur de texte avec coloration syntaxique, l'auto complétion, et l'indentation (pour la création de fichiers python : scripts). C'est aussi un interpréteur (pour exécuter les scripts) et un débogueur intégré avec avancement par étape, point d'arrêts persistants et pile d'appels (pour tester les scripts).

Exemple



Nota : A partir du menu *File* de *IDLE Shell*, vous pouvez accéder à un éditeur de texte (*File* + *New File*) et saisir les instructions d'un programme et ensuite l'exécuter (*Run*).



1.4. Structure d'un programme Python

Un programme Python est structuré comme suit :

- Chaque instruction s'écrit sur une ligne. Si une ligne est trop grande, le caractère \ permet de passer à la ligne suivante.
- Les blocs d'instruction sont matérialisés par des indentations (plus de { et }). Le caractère : (deux points) sert à introduire les blocs.
- Les commentaires vont du caractère # jusqu'à la fin de la ligne. On peut utiliser les commentaires en bloc comme en C (/* ... */).
- Le code *# coding= latin1*, permet d'utiliser des accents dans le code Python.

Exemple d'un programme Python

```
#!/usr/bin/python
# coding= latin1
# Les modules utilises
import sys, socket
# Le code utilisateur
if (a == 1) :
        # sous bloc
        # indente (1 ou 4 espaces)
```

1.5. Exécution d'un programme Python

1.5.1. Premier contact : La console

La *console* est une fenêtre où l'on peut exécuter des commandes les unes après les autres. Les valeurs des variables vont être conservées jusqu'à la fermeture de celle-ci. L'interpréteur fonctionne sur le modèle :

```
>>> instruction python
résultat
```

1.5.2. Communication Script/User

Pour communiquer avec votre programme, vous pouvez utiliser les commandes *input* et *print.*

Exemple :

input("Quel âge avez-vous ? ")

print("Je suis fier de vous")

Chapitre 2 : Variables et instructions

2.1. Variables

2.1.1. Définition d'une variable

Pour pouvoir accéder aux données, le programme d'ordinateur (quel que soit le langage dans lequel il est écrit) fait abondamment usage d'un grand nombre de variables de différents types.

Une *variable* est une zone de la mémoire dans laquelle une valeur est stockée. Aux yeux du programmeur, cette variable est définie par un nom, alors que pour l'ordinateur, il s'agit en fait d'une adresse (i.e. une zone particulière de la mémoire). En Python, la déclaration d'une variable et son initialisation (c'est-à-dire la première valeur que l'on va stocker dedans) se font en même temps. Pour vous en convaincre, testez les instructions suivantes après avoir lancé l'interpréteur :

```
>>> x = 5
>>> x
5
```

Dans cet exemple, nous avons déclaré, puis initialisé la variable *x* avec la valeur 5. Notez bien qu'en réalité, il s'est passé plusieurs choses :

- Python a deviné que la variable était un entier. On dit que Python est un langage au typage dynamique.
- Python a alloué (i.e. réservé) l'espace en mémoire pour y accueillir un entier (chaque type de variable prend plus ou moins d'espace en mémoire), et a fait en sorte qu'on puisse retrouver la variable sous le nom *x*
- Python a assigné la valeur *5* à la variable *x*.

2.1.2. Types de variables

Le type d'une variable correspond à la nature de celle-ci. Les trois types principaux dont nous aurons besoin dans un premier temps sont les entiers (*integer* ou *int*), les réels (*float*) et les chaînes de caractères (*string* ou *str*). Bien sûr, il existe de nombreux autres types (par exemple, les nombres complexes), c'est d'ailleurs un des gros avantages de Python.

Dans l'exemple précédent, nous avons stocké un nombre entier (*int*) dans la variable *x*, mais il est tout à fait possible de stocker des nombres réels (*float*), des chaînes de caractères (*string* ou *str*) ou plein d'autres types de variables que nous verrons par la suite :

```
>>> y = 3.14
>>> y
3.14
>>> a = "Bonjour"
>>> a
'Bonjour'
>>> b = 'Salut chef'
>>> b
'Salut chef'
>>> c = '"Alpha"'
>>> c
'Alpha'
```

Vous remarquez que Python reconnaît certains types de variable automatiquement (entier, réel). Par contre, pour une chaîne de caractères, il faut l'entourer de guillemets (simples, doubles voire trois guillemets successifs simples ou doubles) afin d'indiquer à Python le début et la fin de la chaîne.

2.1.3. Noms de variables et mots réservés

Les noms de variables sont des noms que vous choisissez vous-même assez librement. Efforcez-vous cependant de bien les choisir : de préférence assez courts, mais aussi explicites que possible, de manière à exprimer clairement ce que la variable est censée contenir. Par exemple, des noms de variables tels que *altitude*, *altit* ou *alt* conviennent mieux que *x* pour exprimer une altitude.

Un bon programmeur doit veiller à ce que ses lignes d'instructions soient faciles à lire.

Sous Python, les noms de variables doivent en outre obéir à quelques règles simples :

- Un nom de variable est une séquence de lettres (a → z, A → Z) et de chiffres (0 → 9), qui doit toujours commencer par une lettre.

- Seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que $, #, @, etc. sont interdits, à l'exception du caractère _ (souligné).
- La casse est significative (les caractères majuscules et minuscules sont distingués).

Attention : Maximum, maximum, MAXIMUM sont donc des variables différentes.

Soyez attentifs ! Prenez l'habitude d'écrire l'essentiel des noms de variables en caractères minuscules (y compris la première lettre). Il s'agit d'une simple convention, mais elle est largement respectée. N'utilisez les majuscules qu'à l'intérieur même du nom, pour en augmenter éventuellement la lisibilité, comme dans *tableDesMatieres*.

En plus de ces règles, il faut encore ajouter que vous ne pouvez pas utiliser comme nom de variables les 33 « mots réservés » ci-dessous (ils sont utilisés par le langage lui-même) :

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

2.2. Affectation ou assignation

Nous savons désormais comment choisir judicieusement un nom de variable. Voyons à présent comment nous pouvons définir une variable et lui affecter une valeur.

Les termes « *affecter une valeur* » ou « *assigner une valeur* » à une variable sont équivalents. Ils désignent l'opération par laquelle on établit un lien entre le nom de la variable et sa valeur (son contenu).

En Python comme dans de nombreux autres langages, l'opération d'affectation est représentée par le signe égale :

```
>>> n = 6                    # définir la variable n et lui donner la valeur 6
>>> msg = "Quoi de neuf ?"   # affecter la valeur "Quoi de neuf ?" à msg
>>> pi = 3.14159             # assigner sa valeur à la variable pi
```

Les exemples ci-dessus illustrent des instructions d'affectation Python tout à fait classiques. Après qu'on les ait exécutées, il existe dans la mémoire de l'ordinateur, à des endroits différents :

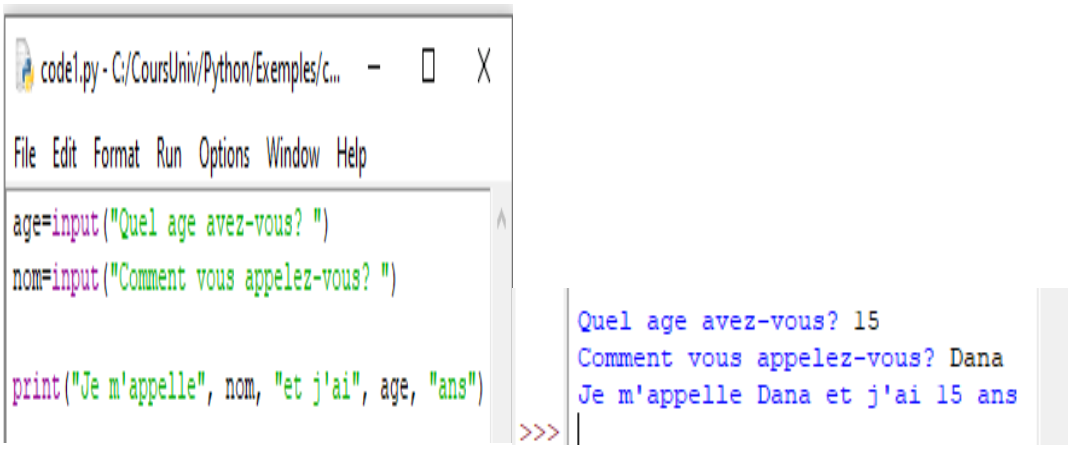
- trois noms de variables, à savoir *n*, *msg* et *pi* ;
- trois séquences d'octets, où sont encodées le nombre entier 6, la chaîne de caractères *Quoi de neuf ?* et le nombre réel *3,14159*.

Les trois instructions d'affectation ci-dessus ont eu pour effet chacune de réaliser plusieurs opérations dans la mémoire de l'ordinateur :

- créer et mémoriser un nom de variable ;
- lui attribuer un type bien déterminé ;
- créer et mémoriser une valeur particulière ;
- établir un lien (par un système interne de pointeurs) entre le nom de la variable et l'emplacement mémoire de la valeur correspondante.

Les trois noms de variables sont des références, mémorisées dans une zone particulière de la mémoire que l'on appelle *espace de noms*, alors que les valeurs correspondantes sont situées ailleurs, dans des emplacements parfois fort éloignés les uns des autres.

Exemple



Nota : Sous Python, il n'est pas nécessaire d'écrire des lignes de programme spécifiques pour définir le type des variables avant de pouvoir les utiliser.

Affectations multiples

Sous Python, on peut assigner une valeur à plusieurs variables simultanément.

Exemple :

```
>>> x = y = 7
>>> x
7
>>> y
7
```

Affectations parallèles

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

```
>>> a, b = 4, 8.33
>>> a
4
>>> b
8.33
```

Dans cet exemple, les variables a et b prennent simultanément les nouvelles valeurs 4 et 8,33.

2.3. Opérateurs et expressions

2.3.1. Opérateurs

On manipule les valeurs et les variables qui les référencent en les combinant avec des opérateurs pour former des expressions.

Exemple : a, b = 7.3, 12

```
y = 3*a + b/5
```

Dans cet exemple, nous commençons par affecter aux variables *a* et *b* les valeurs *7,3* et *12*. Comme déjà expliqué précédemment, Python assigne automatiquement le type « réel » à la variable *a*, et le type « entier » à la variable *b*.

La seconde ligne de l'exemple consiste à affecter à une nouvelle variable *y* le résultat d'une expression qui combine les opérateurs * , + et / avec les opérandes *a, b, 3* et *5*. Les opérateurs sont les symboles spéciaux utilisés pour représenter des opérations mathématiques

simples, telles l'addition ou la multiplication. Les opérandes sont les valeurs combinées à l'aide des opérateurs.

Python évalue chaque expression qu'on lui soumet, aussi compliquée soit-elle, et le résultat de cette évaluation est toujours lui-même une valeur. À cette valeur, il attribue automatiquement un type, lequel dépend de ce qu'il y a dans l'expression.

Dans l'exemple ci-dessus, *y* sera du type réel, parce que l'expression évaluée pour déterminer sa valeur contient elle-même au moins un réel.

Les opérateurs Python ne sont pas seulement les quatre opérateurs mathématiques de base. Il y a l'existence de l'opérateur de *division entière* //. Il faut encore ajouter l'opérateur ** pour l'*exponentiation*, ainsi qu'un certain nombre d'opérateurs logiques, des opérateurs agissant sur les chaînes de caractères, des opérateurs effectuant des tests d'identité ou d'appartenance, etc.

Signalons au passage la disponibilité de l'opérateur *modulo*, représenté par le caractère typographique %. Cet opérateur fournit le reste de la division entière d'un nombre par un autre. Essayez par exemple :

>>> 10 % 3 # (et prenez note de ce qui se passe !)

>>> 10 % 5

Exemple 1: Nombres

```
a=19
b=3

print(a + b)  #Addition                    22
print(a - b)  #Soustraction                16
print(a * b)  #Multiplication              57
print(a / b)  #Division                    6.333333333333333
print(a // b)  #Division entiere           6
print(a % b)  #Reste de la division entière 1
```

Exemple 2: Nombres complexes

```
a=19 + 5j
b=3 + 3j

print(a + b)  #Addition                  (22+8j)
print(a - b)  #Soustraction              (16+2j)
print(a * b)  #Multiplication            (42+72j)
print(a / b)  #Division                  (4-2.3333333333333335j)
print(a.real) #Renvoie la partie réelle 19.0
print(a.imag) #Renvoie la partie         5.0
```

2.3.2. Opérations

a. Opérations sur les types numériques

Les quatre opérations de base se font de manière simple sur les types numériques (nombres entiers et réels) :

```
>>> x = 45
>>> x + 2
47
>>> y = 2.5
>>> x + y
47.5
>>> (x * 10) / y
180.0
```

Remarquez toutefois que si vous mélangez les types entiers et réels, le résultat est renvoyé comme un réel (car ce type est plus général).

b. Opérations sur les chaînes de caractères

Pour les chaînes de caractères, deux opérations sont possibles, l'addition et la multiplication :

```
>>> chaine = "Salut"
```

```
>>> chaine
'Salut'
>>> chaine + " Python"
'Salut Python'
>>> chaine * 3
'SalutSalutSalut'
```

L'opérateur d'addition + permet de concaténer (assembler) deux chaînes de caractères et l'opérateur de multiplication * permet de dupliquer plusieurs fois une chaîne.

Attention : Vous voyez que les opérateurs + et * se comportent différemment selon qu'il s'agit d'entiers ou de chaînes de caractères : 2 + 2 est une addition, '2' + '2' est une concaténation. On appelle ce comportement *surcharge des opérateurs*.

Nota : Il est possible d'ajouter des variables dans une chaine de caractères. Pour cela, il faut ajouter les balises suivantes :

- « %s » : Pour ajouter un string
- « %i » : Pour ajouter un nombre entier
- « %.5f » : Pour ajouter un nombre à virgule (ici, 5 correspond au nombre de décimales voulues)

Exemple

```
>>>pi=3.145896614
>>>nom="Pi"
>>>sortie="La valeur de %s est de  %f mais on peut l'arrondir à  %.2f "%(nom, pi, pi)
>>>print(sortie)
La valeur de Pi est de  3.145896614 mais on peut l'arrondir à  3.14
```

c. Opérations illicites

Attention à ne pas faire d'opération illicite car vous obtiendriez un message d'erreur :

```
>>> 'toto' + 2
Traceback (most recent call last):
```

File "", line 1, in ?

TypeError: cannot concatenate 'str' and 'int' objects

Notez que Python vous donne le maximum d'information dans son message d'erreur. Dans l'exemple précédent, il vous indique que vous ne pouvez pas mélanger des objets de type *str* (*string*, donc des chaînes de caractères) avec des objets de type *int* (donc des entiers), ce qui est assez logique.

2.3.3. Priorité des opérations

Lorsqu'il y a plus d'un opérateur dans une expression, l'ordre dans lequel les opérations doivent être effectuées dépend de règles de priorité.

Sous Python, les règles de priorité sont les mêmes que celles qui vous ont été enseignées au cours de mathématique. Vous pouvez les mémoriser aisément à l'aide d'un « truc » mnémotechnique, l'acronyme PEMDAS:

- *P* pour parenthèses. Ce sont elles qui ont la plus haute priorité. Elles vous permettent donc de « forcer » l'évaluation d'une expression dans l'ordre que vous voulez. Ainsi 2*(3-1) = 4 , et (1+1)**(5-2) = 8.
- *E* pour exposants. Les exposants sont évalués ensuite, avant les autres opérations. Ainsi 2**1+1 = 3 (et non 4), et 3*1**10 = 3 (et non 59049 !).
- *M* et *D* pour multiplication et division, qui ont la même priorité. Elles sont évaluées avant l'addition *A* et la soustraction S, lesquelles sont donc effectuées en dernier lieu. Ainsi 2*3-1 = 5 (plutôt que 4), et 2/3-1 = -0.3333... (plutôt que 1.0).
- Si deux opérateurs ont la même priorité, l'évaluation est effectuée de gauche à droite. Ainsi dans l'expression 59*100//60, la multiplication est effectuée en premier, et la machine doit donc ensuite effectuer 5900//60, ce qui donne 98. Si la division était effectuée en premier, le résultat serait 59 (rappelez-vous ici que l'opérateur // effectue une division entière, et vérifiez en effectuant 59*(100//60).

2.4. Composition

Jusqu'ici nous avons examiné les différents éléments d'un langage de programmation, à savoir : les variables, les expressions et les instructions, mais sans traiter de la manière dont nous pouvons les combiner les unes avec les autres.

Or l'une des grandes forces d'un langage de programmation de haut niveau est qu'il permet de construire des instructions complexes par assemblage de fragments divers. Ainsi par exemple, si vous savez comment additionner deux nombres et comment afficher une valeur, vous pouvez combiner ces deux instructions en une seule :

```
>>> print(17 + 3)
>>> 20
```

Cela n'a l'air de rien, mais cette fonctionnalité qui paraît si évidente va vous permettre de programmer des algorithmes complexes de façon claire et concise.

Exemple :

```
>>>h, m, s = 15, 27, 34
>>>print("nombre de secondes écoulées depuis minuit = ", h*3600 + m*60 + s)
nombre de secondes écoulées depuis minuit =  55654
```

Attention, cependant : il y a une limite à ce que vous pouvez combiner ainsi :

Dans une expression, ce que vous placez à la gauche du signe égale doit toujours être une variable, et non une expression. Cela provient du fait que le signe égale n'a pas ici la même signification qu'en mathématique : comme nous l'avons déjà signalé, il s'agit d'un symbole d'affectation (nous plaçons un certain contenu dans une variable) et non un symbole d'égalité. Le symbole d'égalité (dans un test conditionnel, par exemple) sera évoqué un peu plus loin.

Ainsi par exemple, l'instruction $m + 1 = b$ est tout à fait illégale.

Par contre, écrire $x = x + 1$ est inacceptable en mathématique, alors que cette forme d'écriture est très fréquente en programmation. L'instruction $x = x + 1$ signifie en l'occurrence « augmenter la valeur de la variable x d'une unité » (ou encore : « incrémenter x »).

Permutation de deux variables :

```
>>> a=2
>>> b=5
>>> a,b=b,a
>>> print(a,b)
(5, 2)
```

2.5. Fonction type()

Si vous ne vous souvenez plus du type d'une variable, utilisez la fonction *type()* qui vous le rappellera.

```
>>> x = 2
>>> type(x)
<class 'int'>
>>> y = 2.0
>>> type(y)
<class 'float'>
>>> z = '2'
>>> type(z)
<class 'str'>
```

Faites bien attention, car pour Python, la valeur 2 (nombre entier) est différente de 2.0 (nombre réel), de même que 2 (nombre entier) est différent de '2' (chaîne de caractères). Nous verrons plus tard ce que signifie le mot *class*.

2.6. Conversion de types

Dans tout langage de programmation, on est souvent amené à convertir les types, c'est-à-dire passer d'un type numérique à une chaîne de caractères ou vice-versa. En Python, rien de plus simple avec les fonctions *int(), float()* et *str()*. Pour vous en convaincre, regardez ces exemples :

```
>>> i = 3
>>> str(i)
'3'
>>> i = '456'
>>> int(i)
456
>>> float(i)
456.0
>>> i = '3.1416'
>>> float(i)
```

3.1416

On verra au chapitre sur les fichiers que ces conversions sont essentielles. En effet, lorsqu'on lit ou écrit des nombres dans un fichier, ils sont considérés comme du texte. Toute conversion d'une variable d'un type en un autre est appelé *casting* en anglais, il se peut que vous croisiez ce terme si vous allez consulter d'autres ressources.

2.7. Gestion des erreurs

2.7.1. Nécessité de détecter les erreurs

Même lorsqu'un programme est au point, certaines circonstances exceptionnelles peuvent compromettre la poursuite de son exécution. Il peut s'agir par exemple de données incorrectes ou de la rencontre d'une fin de fichier prématurée (alors qu'on a besoin d'informations supplémentaires pour continuer le traitement).

Bien entendu, on peut toujours essayer d'examiner toutes les situations possibles au sein du programme et prendre les décisions qui s'imposent. Mais outre le fait que le concepteur du programme risque d'omettre certaines situations, la démarche peut devenir très vite fastidieuse et les codes quelque peu complexes. Le programme peut être rendu quasiment illisible si sa tâche principale est masquée par de nombreuses instructions de traitement de circonstances exceptionnelles.

Par ailleurs, dans des programmes relativement importants, il est fréquent que le traitement d'une anomalie ne puisse pas être fait par la méthode l'ayant détectée, mais seulement par une méthode ayant provoqué son appel. Cette dissociation entre la détection d'une anomalie et son traitement peut obliger le concepteur à utiliser des valeurs de retour de méthode servant de "compte rendu". Là encore, le programme peut très vite devenir complexe; de plus, la démarche ne peut pas s'appliquer à des méthodes sans valeur de retour donc, en particulier, aux constructeurs.

La situation peut encore empirer lorsque l'on développe des classes réutilisables destinées à être exploitées par de nombreux programmes.

2.7.2. Notion d'exception

Le langage Python dispose d'un mécanisme très souple nommé *gestion d'exception*, qui permet à la fois :

- de dissocier la détection d'une anomalie de son traitement,
- de séparer la gestion des anomalies du reste du code, donc de contribuer à la lisibilité des programmes.

Exemple : exceptionEr.py

```
nombre = input("Entrer valeur :  ")
try :
        nombre = float(nombre)
        resultat = 20.0 / nombre
except ValueError :
        print("Vous devez entrer un nombre")
except ZeroDivisionError :
        print("Essai de division par zéro")
else:
        print("%.3f / %.3f = %.3f" % ( 20.0, nombre, resultat))
```

2.8. Exercices

Exercice 2.1

Ecrire un programme Python permettant d'échanger les valeurs de deux variables a et b, et ce quel que soit leur contenu préalable.

Exercice 2.2

Soient trois variables a, b et c (supposées du même type). Écrire les instructions permutant leurs valeurs, de sorte que la valeur de b passe dans a, celle de c dans b et celle de a dans c.

Exercice 2.3

En supposant que les variables a, b et c sont de type entier et qu'elles contiennent respectivement les valeurs 5, 8 et 19, déterminer les valeurs des expressions suivantes :

a + b / c

a + c / b

(a + c) / b

a + b / a + b

(a + b) / (a + b)

Exercice 2.4

Ecrire un programme Python qui demande un nombre à l'utilisateur, puis qui calcule et affiche le carré de ce nombre.

Exercice 2.5

Écrire un programme Python qui demande deux nombres entiers et qui fournit leur somme et leur produit.

Exercice 2.6

Ecrire un programme Python qui demande à l'utilisateur de taper 5 entiers et qui affiche leur moyenne.

Exercice 2.7

Ecrire un programme Python qui permet d'introduire à l'ordinateur son prénom et nom son à partir du clavier. Demander ensuite à l'ordinateur de les afficher à l'écran.

Exercice 2.8

Ecrire un programme Python qui permet d'introduire à l'ordinateur les cotes sur 10 obtenues respectivement à l'examen et aux travaux pratiques dans un cours. Demander ensuite à l'ordinateur de calculer la cote globale obtenue sur 20 points dans ce cours et de l'afficher à l'écran.

Exercice 2.9

Ecrire un programme Python qui permet de convertir de francs en dollars.

Exercice 2.10

Ecrire un programme Python qui prend une somme en dollars et la décompose en billets de 10, 5 et 1 dollar.

Exercice 2.11

Ecrire un programme Python qui demande à l'utilisateur de taper le prix prixHT d'un kilo de tomates, le nombre de kilos de tomates achetés, le taux de tva (Exemple 10%, 16%, ...).

L'algorithme affiche alors le prix total prixTTC des marchandises. Faire en sorte que des libellés apparaissent clairement.

Exercice 2.12

Ecrire un programme Python qui demande à l'utilisateur de saisir la largeur et la longueur d'un champ et qui en affiche le périmètre et la surface.

Exercice 2.13

Ecrire un programme Python qui permet de demander à l'ordinateur de calculer le périmètre et la surface d'un cercle et de l'afficher à l'écran, sur base de la valeur du rayon de ce cercle introduite à partir du clavier.

Exercice 2.14

Ecrire un programme Python qui calcule la surface et le volume d'une sphère.

Exercice 2.15

A la naissance de Donel, son grand-père Yan, lui ouvre un compte bancaire. Ensuite, à chaque anniversaire, le grand père de Donel verse sur son compte 100 dollars, auxquels il ajoute le double de l'âge de Donel. Par exemple, lorsqu'elle a deux ans, il lui verse 104 dollars. Ecrire un programme Python qui permette de déterminer quelle somme aura Donel lors de son $n^{ième}$ anniversaire.

Chapitre 3 : Structures de contrôle

3.1. Opérateurs de comparaison

Python est capable d'effectuer toute une série de comparaisons entre le contenu de deux variables, telles que :

Opérateur	Signification
==	égal à
!=	différent de
>	supérieur à
>=	supérieur ou égal à
<	inférieur à
<=	inférieur ou égal à

Observez l'exemple suivant avec des nombres entiers.

```
>>> x = 5
>>> x == 5
True
>>> x > 10
False
>>> x < 10
True
```

Python renvoie la valeur *True* si la comparaison est vraie et *False* si elle est fausse. *True* et *False* sont des booléens.

Faites bien attention à ne pas confondre l'opérateur d'affectation = qui affecte une valeur à une variable et l'opérateur de comparaison == qui compare les valeurs de deux variables.

Vous pouvez également effectuer des comparaisons sur des chaînes de caractères.

```
>>> animal = "tigre"
>>> animal == "tig"
```

False

>>> animal != "tig"

True

>>> animal == 'tigre'

True

Dans le cas des chaînes de caractères, a priori seuls les tests == et != ont un sens. En fait, on peut aussi utiliser les opérateurs, <= et >=. Dans ce cas l'ordre alphabétique est pris en compte, par exemple :

>>> "a" < "b"

True

"a" est inférieur à "b" car il est situé avant dans l'ordre alphabétique. En fait, c'est l'ordre ASCII des caractères qui est pris en compte (i.e. chaque caractère est affecté à un code numérique). On peut donc comparer aussi des caractères spéciaux (comme # ou ~) entre eux. On peut aussi comparer des chaînes à plus d'un caractère.

>>> "ali" < "alo"

True

>>> "abb" < "ada"

True

Dans ce cas, Python compare caractère par caractère de la gauche vers la droite (le premier avec le premier, le deuxième avec le deuxième, etc). Dès qu'un caractère est différent entre l'une et l'autre des chaînes, il considère que la chaîne la plus petite est celle qui présente le caractère ayant le plus petit code ASCII (les caractères suivants de la chaîne sont ignorés dans la comparaison), comme dans l'exemple "abb" < "ada" ci-dessus.

Nota : Il est possible de se servir des opérateurs logiques ci-dessus pour faire la combinaison des conditions.

Opérateur	Utilisation et signification
AND	[Condition 1] and [Condition 2] Si les deux conditions sont vraies => True
OR	[Condition 1] or [Condition 2] Si une des conditions est vraie => True
NOT	not [Condition] Si la condition est fausse => True

3.2. Structure alternative

Elle détermine si le bloc d'instructions suivant est exécuté ou non. La condition est une expression booléenne dont la valeur détermine le bloc d'instructions à exécuter.

3.2.1. Structure à un niveau de test

Syntaxe

```
if condition:
    bloc_instruction1
else :
    bloc_instruction2
```

Indentation de bloc

- Un bloc est défini par une indentation obtenue en décalant le début des instructions vers la droite grâce à des espaces en début de ligne (habituellement 4 espaces mais ce n'est pas obligatoire).
- Toutes les instructions d'un même bloc doivent être indentées exactement au même niveau (c'est-à-dire décalées à droite d'un même nombre d'espaces). L'irrespect de l'indentation génère une erreur.

Exemple : condition1.py

```
chaine = input("Note sur 20 : ")
note = float(chaine)
if note>=10.0:
    print("J'ai la moyenne")
else:
    print("C'est en dessous de la moyenne")
```

```
print("Fin du programme")

>>>
Note sur 20 : 15
J'ai la moyenne
Fin du programme
```

3.2.2. Structure à deux niveaux de test

```
if condition1:
    if condition1_1:
        bloc_instruction1
    else:
        bloc_instruction2
else:
    bloc_instruction3
```

Exemple : condition2.py

```
note = float(input("Note sur 20 : "))
if note>=10.0:
    if note>=13:
        print("mention Bien")
    else:
        print("mention Passable")
else:
    print("C'est en dessous de la moyenne")
print("Fin du programme")

>>>
Note sur 20 : 15
Mention Bien
Fin du programme
```

3.3. Structure généralisée

a. Syntaxe en utilisant else if

```
if condition_1:
        bloc_instruction_1_1
else :
        if condition_2:
                bloc_instruction_2_1
        else :
                bloc_instruction_2_2
else:
        bloc_instruction _1_2
```

Exemple : condition3.py

```
note = float(input("Note sur 20 : "))
if note>20.0 or note<0.0:
        print("Note invalide!")
 else :
        if note>=10.0:
                print("J'ai la moyenne")
        if note==20.0:
                print("C'est même excellent !")
        else:
                print("C'est en dessous de la moyenne")
                if note==0.0:
                        print(" lamentable !")
print("Fin du programme")
>>>
Note sur 20 : 20
J'ai la moyenne
C'est même excellent !
Fin du programme
>>>
Note sur 20 : 3
```

C'est en dessous de la moyenne

Fin du programme

>>>

Note sur 20 : 0

C'est en dessous de la moyenne

lamentable !

Fin du programme

>>>

Note sur 20 : 30

Note invalide!

Fin du programme

b. Syntaxe en utilisant elif

```
if expression 1:
        bloc d'instructions 1
elif expression 2:
        bloc d'instructions 2
elif expression 3:
        bloc d'instructions 3
else:
        bloc d'instructions 4
```

Exemple : condition4.py

```
note = float(input("Note sur 20 : "))
if note==0.0:
        print("C'est en dessous de la moyenne")
        print("... lamentable !")
elif note==20.0:
        print("J'ai la moyenne")
        print("C'est même excellent !")
elif note<10.0 and note>0.0:
```

```
# ou bien : elif 0.0 < note < 10.0:
        print("C'est en dessous de la moyenne")
elif note>=10.0 and note<=20.0:
# ou bien : elif 10.0 <= note < 20.0:
         print("J'ai la moyenne")
else:
        print("Note invalide !")
print("Fin du programme")

>>>
Note sur 20 : 20
J'ai la moyenne
C'est même excellent !
Fin du programme

>>>
Note sur 20 : 0
C'est en dessous de la moyenne
... lamentable !
Fin du programme

>>>
Note sur 20 : 8
C'est en dessous de la moyenne
Fin du programme

>>>
Note sur 20 : 15
J'ai la moyenne
Fin du programme
```

3.4. Exercices

Exercice 3.1

Ecrire un programme Python qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif (on laisse de côté le cas où le nombre vaut zéro).

Exercice 3.2

Ecrire un programme Python qui demande un nombre entier à l'utilisateur et qui précise s'il est ou non compris entre 10 (exclus) et 20 (inclus).

Exercice 3.3

Ecrire un programme Python qui demande deux nombres à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif (on laisse de côté le cas où le produit est nul). Attention toutefois : on ne doit pas calculer le produit des deux nombres.

Exercice 3.4

Ecrire un programme Python qui demande trois noms à l'utilisateur et l'informe ensuite s'ils sont rangés ou non dans l'ordre croissant strict.

Exercice 3.5

Coel va dans un magasin d'électro-ménager. Il regarde dans sa poche et constate qu'il a x dollars dans sa poche (x étant un entier naturel). Il voudrait s'acheter une clé USB à 5 $. Combien restera-t-il dans sa poche après passage à la caisse ?

Exercice 3.6

Donel va à la fête foraine et joue à un jeu de fléchettes. La cible circulaire a 10 cases numérotées de 1 à 10. Si elle fait plus de 8 (8 inclus), elle gagne 10 $ sinon elle perd sa mise de départ de 2 $.

Exercice 3.7

Pour organiser des rencontres sportives, un moniteur doit connaître l'âge des enfants, puis constituer des équipes homogènes. Parmi les moins de 16 ans et les plus de 6 ans, les catégories sont : "Poussin" de 6 à 7 ans, "Pupille" de 8 à 9, "Minime" de 10 à 11 et "Cadet" après 12 ans. Ecrire un programme Python qui classe l'enfant dans la catégorie de son âge. Peut-on concevoir plusieurs algorithmes équivalents menant à ce résultat ?

Exercice 3.8

Au baccalauréat, la mention associée à une note sur 20 est « très bien » pour les notes supérieures ou égales à 16, « bien » pour les notes comprises entre 14 inclus et 16 exclu, « assez bien » pour les notes comprises entre 12 inclus et 14 exclu, « passable » pour les notes comprises entre 10 inclus et 12 exclu et « insuffisant » pour les notes strictement inférieures à 10.

Exercice 3.9

Ecrire un programme Python qui demande 5 notes, calcule la moyenne et attribue la mention correspondante :

- Si moyenne >= 16, mention "Très bien"
- Si moyenne >= 14, mention "Bien"
- Si moyenne >= 12, mention "Assez bien"
- Si moyenne >= 10, mention "Passable"
- Si moyenne >= 8, "Admis oral du deuxième groupe"
- Sinon "Recalé"

Exercice 3.10

Écrire un programme Python qui lit un prix hors taxe et qui calcule et affiche le prix TTC (Toutes Taxes Comprises) correspondant (avec un taux de TVA de 16%). Il établit ensuite une remise dont le taux est le suivant :

- 0% pour un montant TTC inférieur à 100 dollars,
- 1% pour un montant TTC supérieur ou égal à 100 dollars et inférieur à 200 dollars,
- 2% pour un montant TTC supérieur ou égal à 200 dollars et inférieur à 500 dollars,
- 5% pour un montant TTC supérieur ou égal à 500 dollars.

On affichera la remise obtenue et le nouveau montant TTC.

Exercice 3.11

Ecrire un programme Python qui permet de calculer une équation du second degré.

Exercice 3.12

Ecrire un programme Python destiné à prédire l'avenir, et il doit être infaillible. Il lira au clavier l'heure et les minutes, et il affichera l'heure qu'il sera une minute plus tard. Par

exemple, si l'utilisateur tape 7 puis 14, l'algorithme doit répondre : "Dans une minute, il sera 7 heure(s) 15". On suppose que l'utilisateur entre une heure valide et donc pas besoin donc de la vérifier.

Exercice 3.13

Un magasin de reprographie facture 0,10 $ les dix premières photocopies, 0,09 $ les vingt suivantes et 0,08 $ au-delà. Ecrire un algorithme qui demande à l'utilisateur le nombre de photocopies effectuées et qui affiche la facture correspondante.

Exercice 3.14

Les habitants du Sankuru paient l'impôt selon les règles suivantes :

- les hommes de plus de 20 ans paient l'impôt
- les femmes paient l'impôt si elles ont entre 18 et 35 ans
- les autres ne paient pas d'impôt

Le programme Python demandera donc l'âge et le sexe du Sankurois, et se prononcera donc ensuite sur le fait que l'habitant est imposable.

Exercice 3.15

Une assurance propose trois tarifs (Vert, Orange et Rouge) selon l'âge et le nombre d'accidents des automobilistes.

	Moins de 25 ans	25 ans et plus
0 accident	Orange	Vert
1 ou 2 accidents	Rouge	Orange
3 à 6 accidents	Pas assuré	Rouge
7 accidents ou plus	Pas assuré	Pas assuré

Ecrire un programme Python qui affiche le tarif après avoir saisi l'âge et le nombre d'accidents d'un automobiliste.

Exercice 3.16

Une compagnie d'assurance automobile propose 4 familles de tarifs du moins cher au plus onéreux : A, B, C et D. Le tarif dépend de la situation du conducteur.

- Un conducteur de moins de 25 ans et titulaire du permis depuis moins de deux ans, se voit attribuer le tarif D s'il n'a jamais été responsable d'accident. Sinon, la compagnie refuse de l'assurer.

- Un conducteur de moins de 25 ans et titulaire du permis depuis plus de deux ans, ou de plus de 25 ans mais titulaire du permis depuis moins de deux ans a le droit au tarif C s'il n'a jamais provoqué d'accident, au tarif D pour un accident, sinon il est refusé.
- Un conducteur de plus de 25 ans titulaire du permis depuis plus de deux ans bénéficie du tarif B s'il n'est à l'origine d'aucun accident et du tarif C pour un accident, du tarif D pour deux accidents, et refusé sinon.

Par ailleurs, pour encourager la fidélité de ses clients, la compagnie propose un contrat au tarif immédiatement inférieur s'il est assuré depuis plus d'un an.

Ecrire un programme Python qui propose un tarif d'assurance selon les caractéristiques d'un client potentiel.

Exercice 3.17

Vous désirez comparer deux offres d'abonnement téléphonique. La facture est calculée avec un fixe (somme à payer obligatoirement tous les mois) et une partie proportionnelle au temps passé à téléphoner (indiqué en minutes).

Offre	**Fixe**	**Prix à la minute**
Telecom 1	10 $	0.50 $
Telecom 2	15 $	0.42 $

Ecrire un programme Python qui indique l'opérateur le plus intéressant après avoir saisi la consommation moyenne mensuelle (en minutes).

Exercice 3.18

Dans un hôtel, sur les prix affichés pour deux chambres ayant les mêmes caractéristiques, on peut lire :

Chambre A : prix fixe de la chambre 150 dollars + 10 dollars de connexion internet par jour ;

Chambre B : prix fixe de la chambre 170 dollars + 5 dollars de connexion internet par jour ;

On souhaite concevoir un programme Python permettant de trouver le prix le plus avantageux entre les deux chambres en fonction du nombre de jours de location.

Exercice 3.19

Les élections législatives d'un pays X obéissent à la règle suivante :

- lorsque l'un des candidats obtient plus de 50% des suffrages, il est élu dès le premier tour.

- en cas de deuxième tour, peuvent participer uniquement les candidats ayant obtenu au moins 12,5% des voix au premier tour.

Ecrire un programme Python qui permet de saisir les scores de quatre candidats au premier tour. Cet algorithme traitera ensuite le candidat numéro 1 (et uniquement lui) : il dira s'il est élu, battu, s'il se trouve en ballottage favorable (il participe au second tour en étant arrivé en tête à l'issue du premier tour) ou défavorable (il participe au second tour sans avoir été en tête au premier tour).

Exercice 3.20

Ecrire un programme Python qui permet de calculer le lendemain d'une date selon les critères suivants :

- La date est décomposée dans trois variables année, mois et jour.
- Il faut gérer : les changements de mois
    - le nombre de jours dans le mois
    - le changement d'année
    - les années bissextiles pour le mois de février
- Pour information, une année est bissextile (contient 366 jours) si elle est multiple de 4, sauf les années de début de siècle (qui se terminent par 00) qui ne sont bissextiles que si elles sont divisibles par 400.

    Exemples :

    - 1980 et 1996 sont bissextiles car elles sont divisibles par 4

    - 2000 est une année bissextile car elle est divisible par 400

    - 2100 et 3000 ne sont pas bissextiles car elles ne sont pas divisibles par 400.

Chapitre 4 : Structures itératives

L'une des tâches que les machines font le mieux est la répétition sans erreur de tâches identiques. Dès lors, il existe bien des méthodes pour programmer ces tâches répétitives. Pour ce faire, nous allons commencer par l'une des plus fondamentales : la boucle de répétition construite autour de l'instruction *while*.

4.1. Boucle while

La boucle de type "*while*" permet la répétition d'un bloc d'instructions tant que la condition testée est vérifiée, donc vraie.

Syntaxe

while expression :

bloc instructions

Exemple : while1.py

```
i = 1
while i <= 4:
        print(i)
        i = i + 1
>>>
1
2
3
4
```

Remarquez qu'il est encore une fois nécessaire d'indenter le bloc d'instructions correspondant au corps de la boucle (ici 2 instructions).

Une boucle *while* nécessite généralement trois éléments pour fonctionner correctement:

- l'initialisation de la *variable de test* avant la boucle ;
- le *test de la variable* associé à l'instruction *while* ;
- la *mise à jour de la variable de test* dans le corps de la boucle.

Faites bien attention aux tests et à l'incrémentation que vous utilisez car une erreur mène souvent à des boucles infinies, c'est-à-dire qui ne s'arrêtent jamais. Vous pouvez

néanmoins toujours stopper l'exécution d'un script Python à l'aide de la combinaison de touches *Ctrl-C*. Par exemple :

```
i = 0
while i < 10:
        print("Le python c'est cool !")
```

Ici nous avons omis de mettre à jour la variable *i*, ainsi la boucle ne s'arrêtera jamais (sauf en pressant Ctrl-C) puisque la condition *i < 10* sera toujours vraie.

Ruptures de contrôle

- *continue* continue directement à la prochaine itération de la boucle
- *break* sort de la boucle courante (la plus imbriquée)

Exemple 2: while2.py

```
somme=0
while True:
        n=int(input("Entrez un nombre 0 pour arrêter"))
        if n==0:
                break
        somme=somme+n
print("la somme des nombes est", somme)
```

```
>>>
Entrez un nombre 0 pour arrêter 5
Entrez un nombre 0 pour arrêter 8
Entrez un nombre 0 pour arrêter 0
La somme des nombres est  13
```

Dans cet exemple, l'expression *True* est toujours vraie : on a une boucle sans fin. L'instruction break est donc le seul moyen de sortir de la boucle.

Exemple 3 : **while3.py**

```
i = 1
while i <= 5:
   if i==3:
        i = i + 1
        continue
```

```
    print(i)
    i = i + 1

>>>
1
2
4
5
```

La boucle a sauté la valeur 3.

4.2. Boucle for

La boucle *for* est utilisée très fréquemment en programmation Python pour réitérer une exécution un nombre de fois connu à l'avance.

Syntaxe

```
for élément in itérable :
        instructions
```

Grâce à la fonction *range([début], fin, [pas])* la boucle *for* peut parcourir un intervalle de *début* à *fin-1* ou en définissant la borne initiale. Elle peut également parcourir d'autres objets altérables (listes, chaines, tuples, ensembles, dictionnaires, tableaux…).

Exemple 1 : for1.py

```
for i in range(5):
        print(i)
print("Fin de la boucle")

>>>
0
1
2
3
4
Fin de la boucle
```

Exemple 2 : for2.py

```
for compteur in range(1,6):
        print(compteur, '* 9 =', compteur*9)
print("Fin de la boucle")
```

```
>>>
1 * 9 = 9
2 * 9 = 18
3 * 9 = 27
4 * 9 = 36
5 * 9 = 45
Fin de la boucle
```

Exemple 3 : **for3.py**

```
for i in range(1,8,2):
        print(i**2)
print("Fin de la boucle")

>>>
1
9
25
49
Fin de la boucle
```

4.3. Boucles while – else et for- else

Les boucles *while* et *for* peuvent posséder une clause *else* qui ne s'exécute que si la boucle se termine normalement, c'est-à-dire sans interruption.

Exemple

```
y = int(input("Entrez un entier positif : "))
while not(y > 0) :
        y = int(input('Entrez un entier positif, S.V.P. : '))
x = y // 2
while x > 1:
        if y % x == 0:
                print(x, "a pour facteur", y)
                break # voici l'interruption !
        x -= 1
else :
        print(y, "est premier.")

>>>
Entrez un entier positif : 0
```

Entrez un entier positif, S.V.P. : 9
3 a pour facteur 9

>>>
Entrez un entier positif : 7
7 est premier.

4.4. Exercices

Exercice 4.1

Écrire un programme qui demande à l'utilisateur de lui fournir un nombre entier positif et inférieur à 10 et ceci jusqu'à ce que la réponse soit satisfaisante.

Exercice 4.2

Ecrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : « Plus petit ! », et inversement, « Plus grand ! » si le nombre est inférieur à 10.

Exercice 4.3

Réécrire l'algorithme précédent en utilisant une autre forme de boucle.

Exercice 4.4

Ecrire un algorithme qui demande un nombre de départ, et qui ensuite affiche les vingt nombres suivants.

Exercice 4.5

Écrire un algorithme qui demande un nombre et qui calcule la somme des entiers consécutifs de 1 jusqu'à ce nombre.

Exercice 4.6

Lire deux nombres entiers dans les variables *nd* et *nf* et écrire les doubles des nombres compris entre ces deux limites (incluses).

Exercice 4.7

Écrire un algorithme qui demande un nombre et qui calcule la somme des entiers inversés consécutifs de 1 jusqu'à ce nombre.

Exercice 4.8

Ecrire un algorithme qui reçoit en entrée un nombre entier de 1 à 10 et affiche en sortie la table de multiplication de ce nombre.

Exercice 4.9

A la naissance de Donel, son grand-père Yan, lui ouvre un compte bancaire. Ensuite, à chaque anniversaire, le grand père de Donel verse sur son compte 100 dollars, auxquels il ajoute le double de l'âge de Donel. Par exemple, lorsqu'elle a deux ans, il lui verse 104 dollars. Ecrire un algorithme qui permette de déterminer quelle somme aura Donel lors de son $n^{ième}$ anniversaire.

Exercice 4.10

La population de Beni est de 10 000 000 d'habitants et elle augmente de 500000 habitants par an. Celle du Sankuru est de 5 000 000 habitants et elle augmente de 3% par an. Ecrire un algorithme permettant de déterminer dans combien d'années la population du Sankuru dépassera celle de Beni.

Exercice 4.11

Ecrire un algorithme qui demande successivement n nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces n nombres.

Exercice 4.12

Réécrire l'algorithme précédent, mais cette fois-ci on ne connaît pas d'avance combien l'utilisateur souhaite saisir de nombres. La saisie des nombres s'arrête lorsque l'utilisateur entre un zéro.

Exercice 4.13

Pour une promotion donnée, écrire un algorithme qui permet d'introduire le nom de chaque étudiant et les notes obtenues dans les cinq cours inscrits au programme, calcule la moyenne et affiche ensuite quel était le meilleur parmi ces étudiants.

Exercice 4.14

Lire la suite des prix (en dollars entiers et terminée par zéro) des achats d'un client. Calculer la somme qu'il doit, lire la somme qu'il paye, et simuler la remise de la monnaie en

affichant les textes "10 dollars", "5 dollars" et "1 dollar" autant de fois qu'il y a de coupures de chaque sorte à rendre.

Exercice 4.15

Votre tante fortunée vous envoie désormais un montant chaque mois. A la fin de l'année vous souhaitez faire un bilan de vos richesses. Ecrire un algorithme qui affiche la somme mensuelle moyenne reçue, le montant minimal reçu sur l'année et le montant maximal reçu sur l'année.

Exercice 4.16

Ecrire un algorithme qui demande à l'utilisateur de lui fournir la valeur d'un capital qu'il souhaite placer, ainsi que le taux (annuel) auquel sera effectué le placement. Il affiche l'évolution annuelle de ce capital jusqu'à ce qu'il ait atteint ou dépassé le double du capital initial.

Exercice 4.17

Modifier le programme de doublement de capital, de manière qu'il affiche, outre le capital obtenu chaque année, un numéro d'année, comme suit :

donnez le capital à placer et le taux : 10000 0.12

capital, à l'année 1 : 11200.00

capital, à l'année 2 : 12544.00

.....

capital, à l'année 6 : 19738.23

capital, à l'année 7 : 22106.82

Exercice 4.18

Un institut de sondage veut faire une enquête sur les intentions de vote à un référendum. Il y a trois intentions possibles :

- voter oui
- voter non
- voter blanc ou s'abstenir

L'institut veut distinguer les intentions de vote des hommes et des femmes.

Exercice 4.19

Yan est un fermier qui dispose d'un couple de shadoks capables de se reproduire à vitesse phénoménale. Un couple de shadocks met deux mois pour grandir ; à partir du troisième mois, le couple de shadocks engendre une paire de nouveaux shadocks (qui mettront deux mois pour grandir et donc trois mois pour engendrer une nouvelle paire, etc.). Et surtout, les shadoks ne meurent jamais !

D'après cet exercice le nombre de couples de shadoks f_n à chaque mois *n* obéit à la loi:

- $f_0 = 1$
- $f_1 = 1$
- $f_n = f_{n-1} + f_{n-2}$ pour tout $n > 1$

Développer un algorithme permettant de construire la suite des couples depuis le premier jusqu'au $20^{ème}$ mois.

Exercice 4.20

Ecrire un algorithme permettant d'évaluer vos chances de gagner dans l'ordre ou dans le désordre au tiercé, quarté ou quinté. De manière formelle, le problème est le suivant :

- Données : un nombre *n* de chevaux partants et un nombre $p \in \{3,4,5\}$ de chevaux joués
- Résultat : la probabilité x de gagner au jeu dans l'ordre, et la probabilité y de gagner au jeu dans le désordre

x et y nous sont donnés par la formule suivante, si n est le nombre de chevaux partants et p le nombre de chevaux joués:

$x = n ! / (n - p) !$ $y = n ! / (p ! * (n – p) !)$

Table des matières