

LES BASES DE L'ALGORITHMIQUE



Partie 2:
Langage Python

IDRISSI KHAMLIHI SAFAA

NTIC 2023-2024

Chapitre 1:

Introduction à Python et à la programmation informatique

Introduction

Cette partie a pour but de vous montrer ce qu'est le langage Python et à quoi il sert. Commençons par les bases absolues.

Un programme rend un ordinateur utilisable. Sans programme, un ordinateur, même le plus puissant, n'est rien d'autre qu'un objet. De même, sans joueur, un piano n'est rien d'autre qu'une boîte en bois.

Les ordinateurs contemporains ne peuvent qu'évaluer les résultats d'opérations très fondamentales, comme l'addition ou la division, mais ils peuvent le faire très rapidement, et peuvent répéter ces actions pratiquement autant de fois que nécessaire.

Introduction

Imaginez que vous vouliez connaître la vitesse moyenne que vous avez atteinte au cours d'un long voyage. Vous connaissez la distance, vous connaissez le temps, vous avez besoin de la vitesse.

Naturellement, l'ordinateur sera capable de calculer cela, mais l'ordinateur n'est pas conscient de choses telles que la distance, la vitesse ou le temps. Il est donc nécessaire de donner des instructions à l'ordinateur pour qu'il le fasse :

- ❑ Accepter un nombre représentant la distance ;
- ❑ Accepter un nombre représentant la durée du voyage ;
- ❑ Diviser la première valeur par la seconde et stocker le résultat dans la mémoire ;
- ❑ Afficher le résultat (représentant la vitesse moyenne) dans un format lisible.

Introduction

Ces quatre actions simples forment un **programme**. Bien sûr, ces exemples ne sont pas formalisés et sont très loin de ce que l'ordinateur peut comprendre, mais ils sont suffisamment bons pour être traduits dans une langue que l'ordinateur peut accepter.

Le **langage** est le mot clé.

Introduction

Supposons que vous ayez écrit un programme avec succès. Comment persuader l'ordinateur de l'exécuter ? Vous devez traduire votre programme en **langage machine**. Heureusement, la traduction peut être effectuée par l'ordinateur lui-même, ce qui rend le processus rapide et efficace.

Il existe deux façons différentes de **transformer un programme d'un langage de programmation de haut niveau en langage machine** :

Introduction



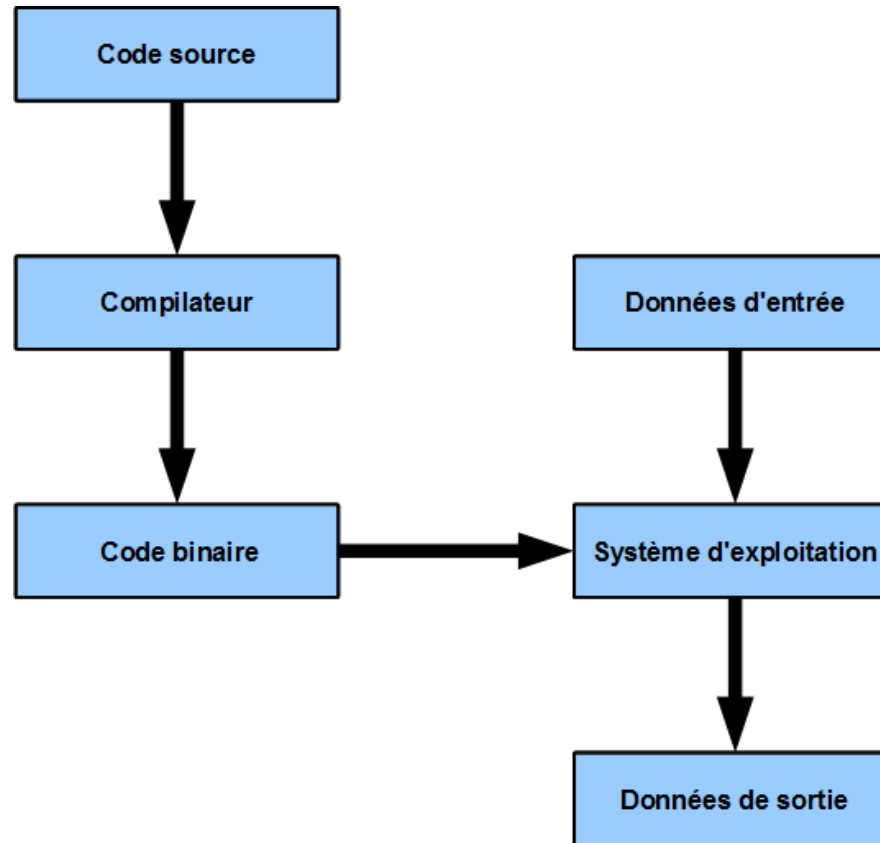
Introduction

1- LA COMPILATION

Le programme source est traduit une fois (cependant, cet acte doit être répété chaque fois que vous modifiez le code source) en obtenant un fichier (par exemple, un fichier .exe si le code est destiné à être exécuté sous MS Windows) contenant le code machine ; vous pouvez maintenant distribuer le fichier dans le monde entier ; le programme qui effectue cette traduction est appelé **un compilateur** ou un traducteur ;

les langages compilés vont être traduits une fois pour toutes par le compilateur afin de générer un nouveau fichier qui sera autonome (C, C++)

Introduction



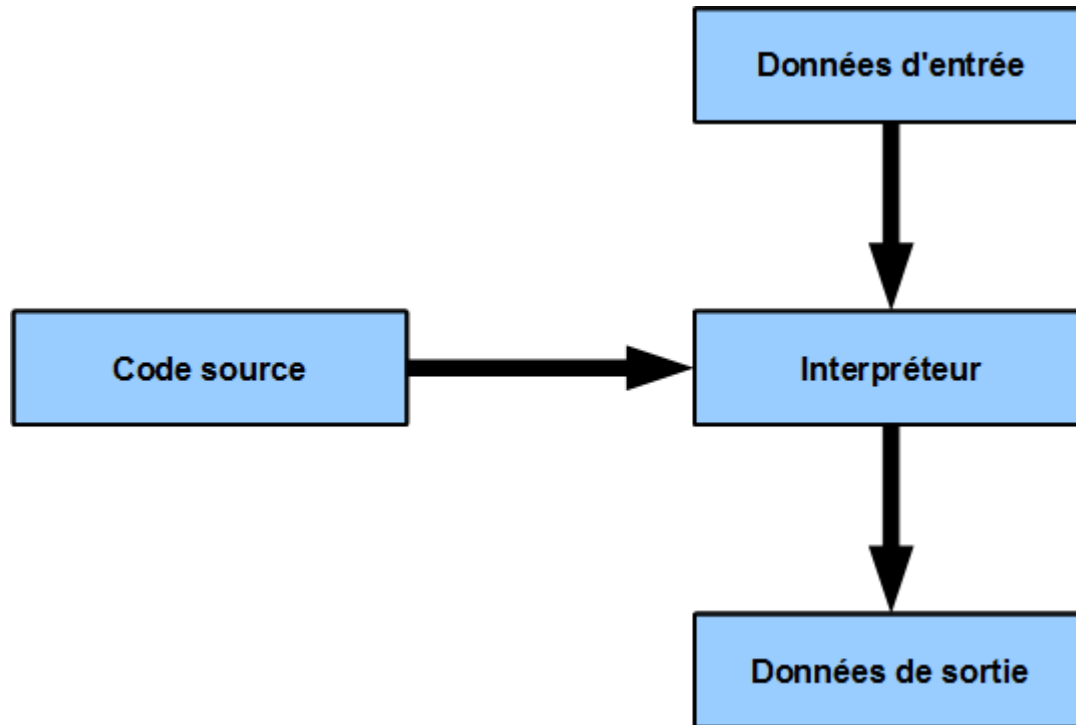
Introduction

2- L'INTERPRETATION

Vous (ou tout utilisateur du code) pouvez traduire le programme source chaque fois qu'il doit être exécuté ; le programme effectuant ce type de transformation est appelé un **interprète**, car il interprète le code **chaque fois qu'il doit être exécuté** ; cela signifie également que vous ne pouvez pas simplement distribuer le code source tel quel, car l'utilisateur final a également besoin de l'interprète pour l'exécuter.

les langages interprétés ont besoin d'un auxiliaire pour traduire au fur et à mesure les instructions (JAVA,Python ,HTML,Matlab)

Introduction



Python

Python est un langage de programmation de haut niveau, largement utilisé, interprété et orienté objet, avec une sémantique dynamique, utilisé pour la programmation générale.

Python a été créé par **Guido van Rossum**(né en 1956 à Haarlem, aux Pays-Bas) en 1991.

Python est un langage puissant, à la fois facile à apprendre et riche en possibilités.

Python

Préparer l'environnement du travail

Python

Pour commencer votre travail, vous avez besoin des outils suivants :

- Un **éditeur** qui vous aidera à écrire le code (il doit présenter certaines caractéristiques particulières, non disponibles dans les outils simples) ; cet éditeur dédié vous donnera plus que l'équipement standard du système d'exploitation (bloc-notes ou autres...).
- Une **console** dans laquelle vous pouvez lancer votre code nouvellement écrit et l'arrêter de force lorsqu'il devient incontrôlable.
- Un outil appelé **débogueur**, capable de lancer votre code étape par étape et vous permettant de l'inspecter à chaque instant de son exécution.

Editeur python

Qu'est-ce qu'un IDE ?

Un IDE (*Integrated Development Environment*) est un **regroupement d'outils** utiles pour le développement d'applications (éditeur de code, debugger, builder, indexation du code pour recherches « intelligentes » dans les projets...), rassemblés dans un logiciel unique.

il est déconseillé d'utiliser de simples éditeurs de texte tels que Edit , Notepad ou WordPad, car ils ne sont pas ANSI "intelligent", ni munis de la fonction de coloration syntaxique du code source pour Python, qui aide à relire rapidement et à éviter les fautes de syntaxe.

Par conséquent, il vaut mieux en utiliser des spécialisés en programmation, appelés environnement de développement (EDI). **Par exemple :**

- **PyCharm, Visual Studio Code, SciTE, DrPython, Eclipse, NetBeans, Nedit, PythonWin, Python, scripter**

Editeur PyCharm



PyCharm est (un IDE) un environnement de développement intégré utilisé pour programmer en Python.

Installation

Développé par l'entreprise JetBrains, PyCharm est un logiciel multi-plateforme qui fonctionne sous Windows, Mac OS X et Linux.

Il est décliné en édition professionnelle, diffusé sous licence propriétaire, et en édition communautaire diffusée sous licence Apache (donc 100 % gratuite), téléchargeable depuis le site officiel.

PyCharm

Ce logiciel existe sous deux formes, la version professionnelle et la version communautaire. C'est cette dernière que nous utilisons (en tant qu'étudiant, vous pouvez utiliser la version professionnelle gratuitement mais elle n'est vraiment utile que si vous faites du développement poussé en python ou que vous faites du développement web en python).

PyCharm

Choix d'un projet

Si c'est la première fois que vous utilisez PyCharm, vous n'avez pas encore de projet actif

Vos projets sont (s'il y en a) listés à gauche de cette fenêtre (là, il n'y en a pas).

Cliquez sur **Create new project**

Nom et emplacement du projet

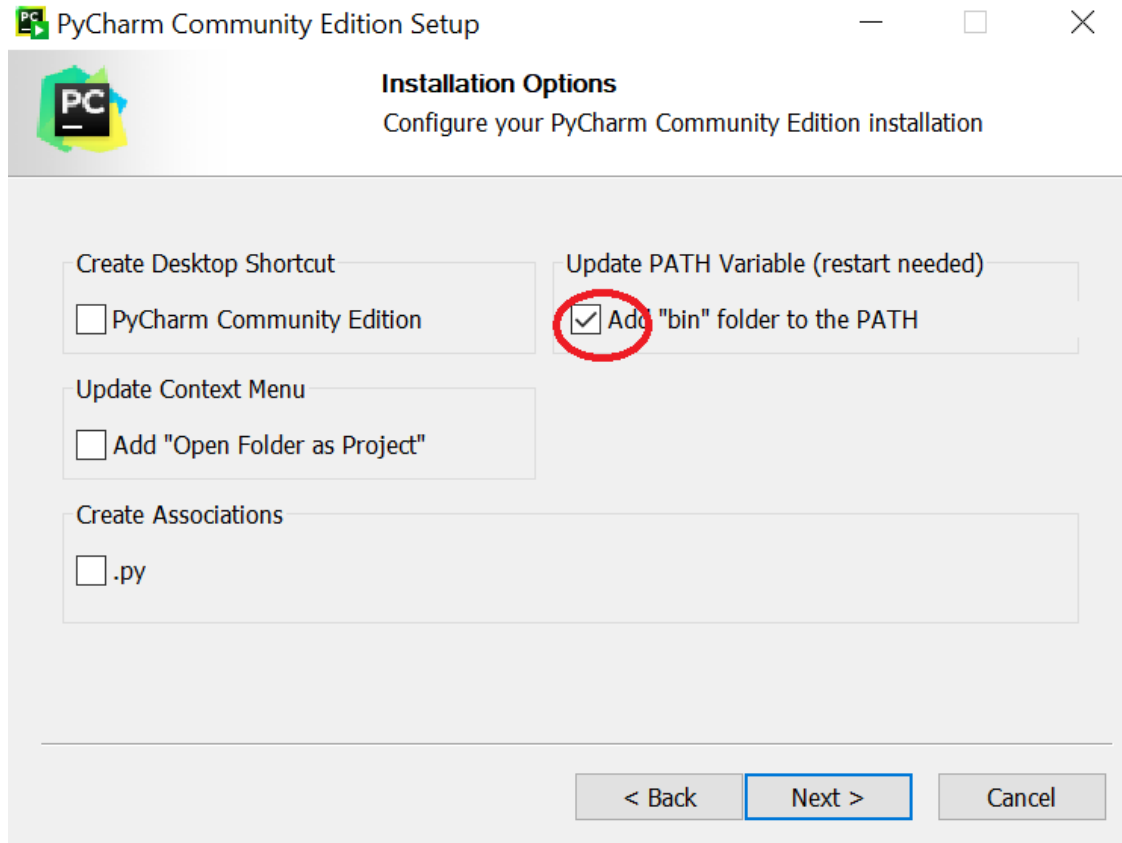
Essayez d'avoir un peu d'ordre dans vos projets :

un dossier principal où tous vos projets seront placés : ici **PycharmProjects**

Interpréteur

C'est la version de python que vous utiliserez pour ce projet. Nous utiliserons **TOUJOURS python3**. Choisissez le donc.

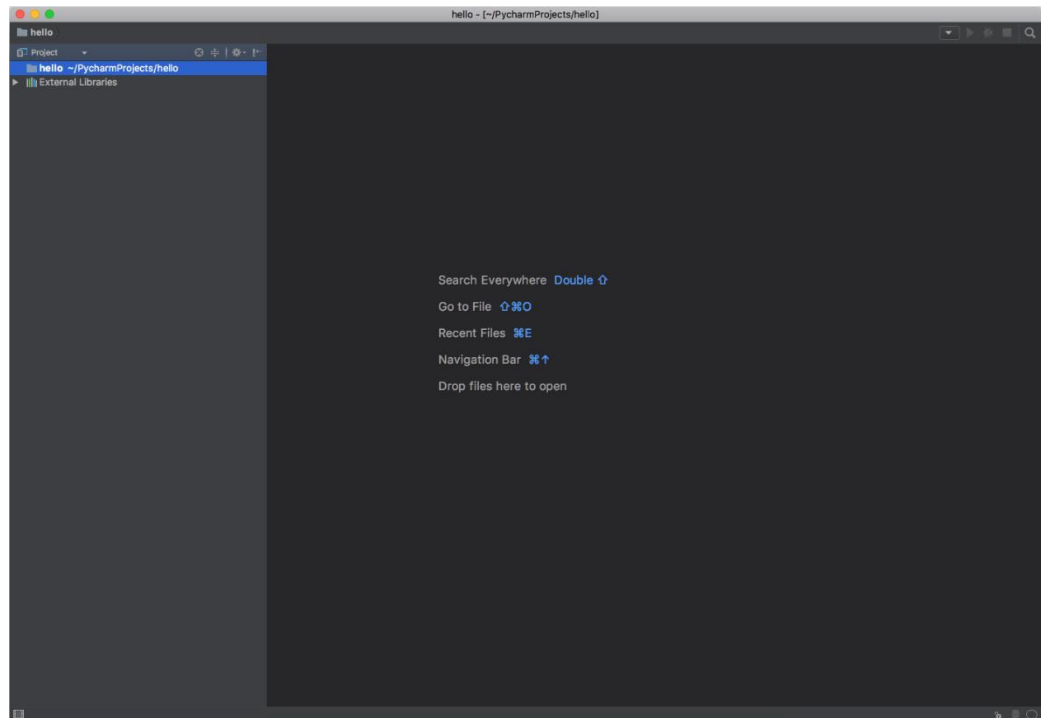
PyCharm



PyCharm

Créer et exécuter un fichier python

Une fois un nouveau projet créé, vous devez avoir une fenêtre du type ci-dessous qui apparaît :



PyCharm

Créer

Pour créer un nouveau fichier, il faut que vous ayez sélectionné le dossier dans lequel votre fichier va être créé dans la partie Project de la fenêtre. À partir de là vous pouvez créer un nouveau fichier dans le menu **File > New...** puis en choisissant un Python File . Le fichier est créé dans le dossier sélectionné.

On peut l'afficher dans la fenêtre principale en double cliquant sur son nom dans la partie Project .

Par exemple un fichier hello.py dont le code est :

```
print("hello world!")
```

PyCharm

Exécuter

Exécuter un fichier se fait via le menu Run . Il y a deux commandes Run dans ce menu : **Run nom_fichier** et **Run...** . Le second permet de choisir le fichier à exécuter. Une fois choisi, son nom apparaîtra en haut à droite de la fenêtre, à côté du triangle vert permettant de l'exécuter.

Chapitre 2:

Types de données, variables,
opérations d'entrée-sortie de base,
opérateurs de base

La fonction print()

Syntaxe :

print(argument1, argument2,...).

Plus généralement **function_name(argument).**

C'est une fonction **intégrée** très importante de Python, elle permet **d'afficher, au niveau de la console, les valeurs fournies en arguments** (càd entre parenthèses et séparés par des virgules « , »), et à l'affichage ces valeurs sont séparées par défaut par des **espaces " "** et à la fin, la fonction termine le travail par un **saut à la ligne**.

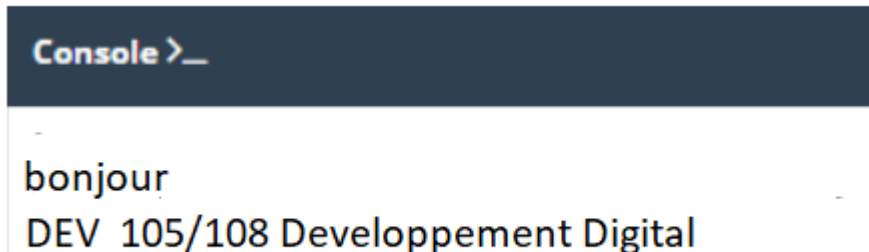
La fonction print()

Exemple :

Nous allons afficher que des chaînes de caractères pour le moment, nous verrons plus tard comment afficher des variables.

```
print("bonjour")
```

```
print("DEV", " 105/108", "Developpement Digital")
```

A screenshot of a console window with a dark blue header bar containing the text "Console >_". Below the header, the output of the code is displayed on two lines: "bonjour" and "DEV 105/108 Developpement Digital".

```
Console >_  
-  
bonjour  
DEV 105/108 Developpement Digital
```

La fonction print()

Comme on vient de le voir, à l'affichage des arguments de la fonction **print()**, ces derniers sont séparés par des espaces, on peut alors remplacer l'espace par n'importe quel autre caractère grâce à **l'option « sep »**.

Exemple :

```
print("DEV"," 105/108","Developpement Digital",sep="-")
```



The screenshot shows a dark-themed console window. The title bar at the top reads "Console >_". Below the title bar, the output of the print function is displayed on two lines: "bonjour" on the first line and "DEV - 105/108 - Developpement Digital" on the second line. The separator "-" is visible between the arguments in the second line of output.

La fonction print()

La fonction print() provoque par défaut un retour à la ligne après affichage des arguments, mais on peut modifier cela grâce **à l'option « end »**.

Exemples :

```
print("Bonjour", end=",")
```

```
print("Rahmani","Abdelhak","Formateur animateur", sep="-*-")
```



Console >_

Bonjour,Rahmani-*-*Abdelhak-*-*Formateur animateur

On peut coupler les 2 options « sep » et « end » dans une même fonction print() :

```
print("Mon", "nom", "est", sep="_", end=" : ")
```

```
print("Rahmani", "Abdelhak", sep="*", end=".")
```



Console >_

Mon_nom_est : Rahmani*Abdelhak.

La fonction `print()`

Remarque:

La fonction **`print()`** peut n'avoir aucun argument, c  d que l'instruction **`print()`** est correcte syntaxiquement et elle provoque naturellement un saut de ligne.

Il est tout    fait possible d'ins  rer des retours    la ligne au niveau des arguments d'une m  me fonction `print()`    l'aide de `\n` (le caract  re « `\` » s'appelle le caract  re d'  chappement ou « escape » qui signifie, si j'ose m'exprimer ainsi, on s'  chappe un petit moment pour faire autre chose (ici retour    la ligne) et on revient...).

La fonction print()

Si on souhaite maintenant, pour une raison ou une autre afficher le caractère d'échappement « \ » lui-même, alors il suffit de le doubler :

Exemple :

```
print("Affiche du caractère \\")
```



Console >_

Affiche du caractère \

La fonction print()

Exercice

Ecrire une fonction print() utilisant « sep » et « end » permettant d'afficher :

Programming***Essentials***in...Python

La fonction print()

Les données littérales

Soit l'exemple suivant :

```
print("2")  
print(2)
```

On constate qu'au niveau de l'affichage en console les 2 print() affichent le même résultat « 2 », mais en fait les manières de stocker en mémoire par la machine et manipuler ces 2 données sont complètement différentes.

Dans le 1er cas, il s'agit du caractère « 2 », tout comme les lettres « a » ou « z ».

Dans le second cas, il s'agit du nombre « 2 » qui a une toute autre signification.

Les Variables

L'ordinateur n'a pas un autre moyen de stocker les nombres et les traiter autrement qu'avec le système binaire composé uniquement des 2 représentations « 0 » et « 1 ». En effet, l'ordinateur est un assemblage complexe de composants électroniques dans lesquels passe du courant électrique et donc ils ne savent traiter les données que sous forme d'impulsions électriques qui ne peuvent être en vérité que sous 2 états : « 0 » courant passe, « 1 » courant ne passe pas, si bien que tous les transits d'information dans la machine se font quasiment avec ce système binaire.

Les Variables

Avec Python on est capable d'encoder des valeurs littérales contenant des valeurs numériques et textuelles sur lesquelles on peut effectuer des opérations arithmétiques.

Mais il est tout à fait normal de se demander comment stocker les résultats de ces opérations, afin de les utiliser dans d'autres opérations...Comment sauvegarder les résultats intermédiaires, et les utiliser à nouveau pour en produire d'autres ? La solution se trouve dans la notion de variable.

Les Variables

Déclaration et Type de Variables

Il existe plusieurs types de base de variables:

- Numbers (float, int).
- Strings.
- Boolean.

Pas besoin de déclaration explicite

Mutabilité des variables : une variable peut prendre plusieurs types de valeurs.

Les Variables

Une variable est un espace mémoire (une sorte de boîte) dans lequel il est possible de mettre une valeur. Par exemple, $x = 1$, la variable x prend la valeur 1, on parle alors de l'affectation ou l'assignation de la valeur 1 à la variable x .

En Python une variable doit avoir :

- Un nom ;
- Une valeur (le contenu du conteneur).

Les Variables

Exercice

Quelles sont les déclarations de variables non permises parmi les suivantes ?

- **my_var**
- **m**
- **101**
- **averylongvariablename**
- **m101**
- **m 101**
- **Del**
- **del**

Les Variables

Affectation

- **A= 1**
- **C="test" ou C='test'**
- **R=3.2**

Exemple

Il n'est pas nécessaire de déclarer une variable pour l'utiliser :

```
var = 1
```

```
print(var)
```

```
val= 1000.0
```

```
nom= 'John Doe'
```

```
print(var, val, nom)
```

Exemple

```
var = "3.7.1"
```

```
print("Python version: " + var)
```

```
print("Python version:",var)
```

On constate qu'on peut afficher en concaténant des « string » à l'aide de l'opérateur « + » ou simplement en séparant les arguments du print() avec « , ».



Console >_

```
Python version: 3.7.1
```

```
Python version: 3.7.1
```

Opérateurs

Opérateurs arithmétiques

$a + b$	Addition	Somme de a et b.
$a - b$	Soustraction	Différence de a et b.
$a * b$	Multiplication	Produit de a et b.
a / b	Division	Quotient de a et b.
$a // b$	Division entière	
$a \% b$	Modulo	Reste de a divisé par b.
$a ** b$	Exponentielle	Résultat de l'élévation de a à la puissance b.

Python comme calculateur

```
print(2 + 2)
```

```
print(2 - 2)
```

```
print(2 * 2)
```

```
print(2 / 2)
```

```
print(2 ** 3)
```

```
print(2 ** 3.)
```

```
print(2. ** 3)
```

```
print(2. ** 3.)
```

```
print(2 * 3)
```

```
print(2 * 3.)
```

```
print(2. * 3)
```

```
print(2. * 3.)
```

```
print(6 / 3)
```

```
print(6 / 3.)
```

```
print(6. / 3)
```

```
print(6. / 3.)
```

```
print(6 // 4)
```

```
print(6. // 4)
```

```
print(-6 // 4)
```

```
print(6. // -4)
```

```
print(14 % 4)
```

```
print(12 % 4.5)
```

```
print(-4 + 4)
```

```
print(-4. + 8)
```

```
print(-4 - 4)
```

```
print(4. - 8)
```

```
print(-1.1)
```

```
print(+2)
```

Opérateurs

Priorités des opérateurs

- parenthèses ()
- **$**$, $*$, $/$, $//$, $\%$**
- **$+$ et $-$**
- Exemple
- **$2 + 3 * 5 = ?$**

$$2 + 3 * 5 ** 2 = ?$$



Opérateurs

A priorité égale, on effectue généralement les calculs de gauche à droite pour tous les opérateurs sauf l'exponentielle « ** ».

Exemple

```
print(9 % 6 % 2)
```

Pour l'opérateur d'exponentielle, c'est différent :

```
print(2 ** 2 ** 3)
```



Opérateurs de raccourcis

Ces opérateurs sont pratiques et ils facilitent la vie au développeur. La règle est la suivante :

- variable = variable **op** expression

S'écrit de manière simplifiée :

- variable **op=** expression

$x = x * 2 \Leftrightarrow x *= 2$

$x = x + 1 \Leftrightarrow x += 1$

$i = i + 2 * j \Leftrightarrow i += 2 * j$

$i = i / 2 \Leftrightarrow i /= 2$

$i = i \% 10 \Leftrightarrow i \% = 10$

$i = i - (j * 2) \Leftrightarrow i -= (j * 2)$

$i = i ** 2 \Leftrightarrow i ** = 2$

Exercice

Exercice 1

Quel est le résultat de l'exécution de l'instruction suivante ?

```
print((2 ** 4), (2 * 4.), (2 * 4))
```

Exercice 2

Quel est le résultat de l'exécution de l'instruction suivante ?

```
print((-2 / 4), (2 / 4), (2 // 4), (-2 // 4))
```

Exercice 3

Quel est le résultat de l'exécution de l'instruction suivante ?

```
print((2 % -4), (2 % 4), (2 ** 3 ** 2))
```

Commentaires

Il suffit pour cela de les faire précéder par « # ».

Chaque ligne de commentaire doit être précédée par « # »

Exemple :

```
# Ce programme calcule l'hypoténuse c d'un triangle rectangle.
```

```
# a et b sont les mesures des autres côtés du triangle.
```

```
a = 3.0
```

```
b = 4.0
```

```
c = (a**2 + b**2)**0.5  # On utilise ** au lieu de la racine carrée.
```

```
print("c =", c)
```

Commentaires

Pour commenter ou dé-commenter un code, il suffit sélectionner la ou les lignes à commenter et utiliser le raccourci clavier : CTRL + « / ».

Pour insérer des commentaires sur plusieurs lignes, soit on place le « # » devant chaque ligne de commentaire, soit on utilise 3 fois les guillemets « """ """ » :

La fonction input()

La fonction `input()` est le reflet miroir de la fonction `printf()`, en effet, si cette dernière renvoie des données à la console, la première permet des données entrées par l'utilisateur, ce qui va permettre de l'interactivité et le dialogue entre l'utilisateur et la machine.

Exemple :

```
Print("Dites-moi quelque chose...")  
var= input()  
print("Hmm...", var, "... Vraiment?")
```

Console >_

Dites-moi quelque chose...

Bonjour

Hmm... Bonjour ... Vraiment?

La fonction input()

La fonction input() peut avoir un argument, ce qui évitera de la faire précéder par un print() :

Exemple :

```
var= input("Dites-moi quelque chose...")  
print("Hmm...", var, "... Vraiment?")
```

Console >_

```
Dites-moi quelque chose...Bonjour  
Hmm... Bonjour ... Vraiment?
```



La fonction input()

Remarque importante :

Le résultat de la fonction input() est une chaîne contenant tous les caractères que l'utilisateur entre depuis le clavier. Ce n'est pas un entier ou un réel ! en ce sens qu'on ne peut pas l'utiliser comme argument d'opérations arithmétiques. En revanche, on peut toujours utiliser le signe « + » de concaténation des chaînes ou même de la duplication à l'aide du « * » :

```
var= input("Dites-moi quelque chose...")  
print("Hmm...", var* 2, "... Vraiment?")
```

La fonction input()

la solution à ce genre de situation, lorsqu'on souhaite saisir des nombres au clavier est donnée par les fonctions **int()** et **float()** qui permettent de convertir des string (passés en argument entre parenthèses) en respectivement des entiers ou des réels :

Conversion de type

```
n = int(input("Saisir un nombre entier : "))
```

```
p = n ** 3
```

```
print(n, "à la puissance 3 est", p)
```

```
r = float(input("Entrer un nombre réel : "))
```

```
pr = r ** 3.0
```

```
print(r, "à la puissance 3 est", pr)
```



Opérations sur les chaînes de caractères

Concaténation

L'opérateur « + » appliqué à des chaînes de caractères permet de les concaténer.

Exemple :

#Concaténation des chaînes de caractères

```
prenom = input("Saisir votre prénom, please : ")
```

```
nom = input("Saisir votre nom, please : ")
```

```
print("Merci.")
```

```
print("\nVous vous appelez : " + prenom + " " + nom + ".")
```



Opérations sur les chaînes de caractères

Duplication

L'opérateur « * » appliqué à une chaîne de caractères et un nombre entier permet de la dupliquer de la manière suivante : `string * nombre` ou `nombre * string`.

Exemple :

#Duplication de chaînes

```
print("Ali" * 3)
```

```
print(3 * "Mohamed")
```

```
print(5 * "2")
```

#A ne pas confondre avec :

```
print(5 * 2)
```

Console >_

```
AliAliAli
```

```
MohamedMohamedMohamed
```

```
22222
```

```
10
```



Opérations sur les chaînes de caractères

Conversion de types nombres → chaînes de caractères

Nous avons vu que les fonctions `int()` et `float()` convertissent des string en nombres. De la même manière, la fonction `str()` fait le travail inverse en convertissant des nombres en chaînes.

Exemple :

```
a = 5
```

```
b = 3
```

```
print("produit de deux nombres :", a * b)
```

```
c = str(b)
```

```
print(("produit d'un nombre et une chaine:", a * c)
```

Console >_

15

33333



Opérations sur les chaînes de caractères

- Les string sont parcourues avec [start:stop:step]
- Step est par défaut 1

ch='merci'

ch[0] → 'm'

ch[0:2] → 'me' #stop-1

ch[1:] → 'erci'

ch[-1] → 'i'

ch[-3:-1] → 'rc' #-1 ne s'affiche pas

Exemple

1- Quel est le résultat de l'extrait de code suivant ?

```
x = int(input("Entrer un nombre : ")) # L'utilisateur entre 2  
print(x * "5")
```

2- Quel est le résultat de l'extrait de code suivant ?

```
x = input("Enter a number: ") # L'utilisateur entre 2  
print(type(x))
```


Operateurs de Comparaison

>	Strictement supérieur
>=	Supérieur ou égal
<	Strictement inférieur
<=	inférieur ou égal
==	égal
!=	différent
or	ou
and	et

Exemple

```
a = 2  
b = 2
```

```
print(a == b)
```

```
a = 2  
b = 2.0
```

```
print(a == b)
```

```
a = "Bonjour"  
b = "Bonsoir"
```

```
print(a == b)
```

Operateurs de Comparaison

Utiliser les réponses

Que pouvez-vous faire avec la réponse (c.-à-d. Le résultat d'une opération de comparaison) que vous obtenez de l'ordinateur?

Il y a au moins deux possibilités: premièrement, vous pouvez le mémoriser (le stocker dans une variable) et l'utiliser plus tard.

Comment tu fais ça? Eh bien, vous utiliseriez une variable arbitraire comme celle-ci:

```
answer = number1 >= number2
```

Operateurs de Comparaison

La deuxième possibilité est plus pratique et beaucoup plus courante : vous pouvez utiliser la réponse que vous obtenez pour prendre une décision concernant l'avenir du programme .

Vous avez besoin d'une instruction spéciale à cet effet, et nous en discuterons très bientôt. (les tests)

EXERCICES

- TP 4 : (série 4)
Notions de base Python

Chapitre 3:

Conditions et exécution conditionnelle en Python

Conditions et exécution conditionnelle

Python propose une instruction spéciale pour faire les tests. En raison de sa nature et de son application, elle est appelée **instruction conditionnelle** (ou instruction conditionnelle).

La première forme d'une déclaration conditionnelle:

```
if condition :  
    do_this_if_true
```

Conditions et exécution conditionnelle

L'indentation (un retrait) peut être obtenue de deux manières - en insérant un nombre particulier d'espaces (la recommandation est d'utiliser **quatre espaces** d'indentation), ou en utilisant le caractère de **tabulation**; note: s'il y a plus d'une instruction dans la partie en retrait, l'indentation doit être la même sur toutes les lignes; même s'il peut avoir la même apparence si vous utilisez des tabulations mélangées à des espaces, il est important que toutes les indentations soient identiques - Python 3 ne permet pas de mélanger les espaces et les tabulations pour l'indentation.

Conditions et exécution conditionnelle

l'instruction if

```
if condition :  
    instruction1  
instruction
```

Conditions et exécution conditionnelle

l'instruction if-else

```
if condition :  
    instruction1  
else:  
    instruction2
```

```
if condition :  
    instruction1  
else:  
    instruction2  
instruction
```

Exemple

Nous commencerons par le cas le plus simple - comment identifier le plus grand des deux nombres :

```
# read two numbers
```

```
number1 = int(input("Enter the first number: "))
```

```
number2 = int(input("Enter the second number: "))
```

```
# choose the larger number
```

```
    if number1 > number2 :
```

```
        larger_number = number1
```

```
    else:
```

```
        larger_number = number2
```

```
    print("The larger number is:", larger_number)
```

Conditions et exécution conditionnelle

l'instruction elif

```
if condition1 :  
    instruction1  
elif condition 2:  
    instruction2  
else :  
    Instruction3
```

Conditions et exécution conditionnelle

Déclarations emboîtées (tests imbriqués)

Considérons d'abord le cas où l'instruction placée après la if est une autre if

if the_weather_is_good:

 if nice_restaurant_is_found:

 have_lunch()

 else:

 eat_a_sandwich()

else:

 if tickets_are_available:

 go_to_the_theater()

 else:

 go_shopping()



Informations supplémentaires

Python est souvent livré avec de nombreuses fonctions intégrées qui feront le travail pour vous. Par exemple, pour trouver le plus grand nombre de tous, vous pouvez utiliser une fonction intégrée Python appelée **max()**. Vous pouvez l'utiliser avec plusieurs arguments.

Analysez le code ci-dessous:



Example

```
# read three numbers
number1 = int(input("Enter the first number: "))
number2 = int(input("Enter the second number: "))
number3 = int(input("Enter the third number: "))
# check which one of the numbers is the greatest
# and pass it to the largest_number variable
largest_number = max(number1, number2, number3)
# print the result
print("The largest number is:", largest_number)
```

De la même manière, vous pouvez utiliser la **min()** fonction pour renvoyer le nombre le plus bas

EXERCICES

- TP 5 : (série 5)

Les Tests et les Conditions

Chapitre 4:

Structures répétitives en Python

Structures répétitives

Les boucles s'utilisent pour répéter plusieurs fois l'exécution d'une partie du programme.

Boucle bornée

Quand on sait combien de fois doit avoir lieu la répétition, on utilise généralement une boucle **for**.

Boucle non bornée

Si on ne connaît pas à l'avance le nombre de répétitions, on choisit une boucle **while**.

Structures répétitives : boucle while

En général, en Python, une boucle peut être représentée comme suit:

while conditional_expression:

instruction

while répète l'exécution tant que la condition est évaluée True .

Remarque: toutes les règles concernant l' **indentation pour les tests** sont également applicables ici.

Structures répétitives

while conditional_expression:

 instruction_one

 instruction_two

 instruction_three

 :

 :

 instruction_n

Instruction_hors_while

Exemple

Afficher le maximum d'une suite de noombre saisi par l'utilisateur,
Le programme s'arrete si l'utilistaeur saisi -1

Reponse:

```
number = int(input("saisir un nombre ou taper -1 pour s'arreter: "))
max= number
while number != -1:
    if number > max:
        max= number
    number = int(input(" saisir un nombre ou taper -1 pour s'arreter :"))

print("The largest number is:", max)
```

Exercice d'application(while)

Afficher "Bonjour mes amis" le nombre de fois que l'utilisateur a saisi

Reponse:

```
number = int(input("saisir un nombre : "))
```

```
Compteur=0
```

```
while compteur!= number :  
    print(" Bonjour mes amis:")  
    compteur++
```

Structures répétitives

Une boucle infinie

Une boucle infinie, également appelée **boucle sans fin** , est une séquence d'instructions dans un programme qui se répète indéfiniment (boucle sans fin.)

Voici un exemple de boucle qui n'est pas en mesure de terminer son exécution :

while True:

print("I'm stuck inside a loop.")

Structures répétitives

Remarque

Essayez de vous rappeler comment Python interprète la vérité d'une condition et notez que ces deux formes sont équivalentes:

while number != 0: et **while number:**

La condition qui vérifie si un nombre est impair peut également être codée sous ces formes équivalentes :

if number % 2 == 1: et **if number % 2:**

Exercice d'application(while)

1. Ecrire un programme python qui saisi une valeur de référence valdébut, et puis saisit une suite de nombres et s'arrête dès que l'on saisit une valeur qui soit supérieure à valdébut. Le programme doit afficher la valeur qui a provoqué l'arrêt, ainsi que son rang.

2. Modifiez votre programme pour qu'il saisi deux entiers référence valdébut et valfin délimitant un intervalle donné (valdébut est plus petit que valfin), et qu'il arrête la boucle de saisie dès que l'on saisit une valeur qui soit dans cet intervalle.

Le programme doit afficher la valeur qui a provoqué l'arrêt ainsi que son rang.

Structures répétitives : boucle for

Un autre type de boucle disponible en Python vient de l'observation qu'il est parfois plus important de **compter les "tours" de la boucle** que de vérifier les conditions.

la boucle for est conçue pour effectuer des tâches plus compliquées - **elle peut «parcourir» de grandes collections de données élément par élément** . Nous allons vous montrer comment faire cela bientôt, mais pour le moment, nous allons présenter une variante plus simple de son application.

Structures répétitives

```
for i in range(100):
```

```
    # do_something()
```

- **La variable** après for est la **variable** de **contrôle** de la boucle; il compte les tours de boucle et le fait automatiquement;
- le mot clé **in** introduit un élément de syntaxe décrivant la plage de valeurs possibles affectées à la variable de contrôle;
- la fonction **range()** (c'est une fonction très spéciale) est chargée de générer toutes les valeurs souhaitées de la variable de contrôle; dans notre exemple: 0, 1, 2 .. 97, 98, 99; note: dans ce cas, la fonction range() démarre son travail à partir de 0 et le termine une étape (un nombre entier) avant la valeur de son argument;

Exemple

```
for i in range(10):
```

```
    print("la valeur de i est ", i)
```

- la boucle a été exécutée dix fois (c'est l'argument de la fonction range())
- la dernière valeur de la variable de contrôle est 9(pas 10, car **elle commence à partir 0** , pas à partir de 1)

Structures répétitives

La fonction `range()`:

Peut être équipée de **deux arguments** `rang(min,max)`, pas d'un seul:

```
for i in range(2, 8):  
    print(" la valeur de i est ", i)
```

- Dans ce cas, le premier argument détermine la (première) valeur initiale de la variable de contrôle.
- Le dernier argument montre la première valeur à laquelle la variable de contrôle ne sera pas affectée.

Remarque:

la `range()` fonction **accepte uniquement des entiers comme arguments** et génère des séquences d'entiers.

Structures répétitives

La `range()` fonction peut également accepter **trois arguments** `rang(min,max,pas)`.

```
for i in range(2, 8, 3):
```

```
    print("The value of i is currently", i)
```

Le troisième argument est un incrément - c'est une valeur ajoutée pour contrôler la variable à chaque tour de boucle (comme vous vous en doutez, la valeur par défaut de l'incrément est 1).

Exercice d'application(for)

Exercice1

Afficher les n premiers nombres, n saisi par l'utilisateur

Exercice 2

Afficher les n premiers nombres pairs , n saisi par l'utilisateur

Exercice 3

Calculer la somme des n nombres premiers, n saisi par l'utilisateur.

Les déclarations break and continue

Jusqu'à présent, nous avons traité le corps de la boucle comme une séquence d'instructions indivisible et inséparable qui sont exécutées complètement à chaque tour de la boucle. Cependant, en tant que développeur, vous pourriez être confronté aux choix suivants:

- il semble qu'il ne soit pas nécessaire de continuer la boucle dans son ensemble; vous devez vous abstenir de poursuivre l'exécution du corps de la boucle et aller plus loin;
- il semble que vous devez commencer le tour suivant de la boucle sans terminer l'exécution du tour en cours.

Les déclarations `break` and `continue`

Python fournit deux instructions spéciales pour l'implémentation de ces deux tâches.

- `break`- quitte la boucle immédiatement et met fin inconditionnellement au fonctionnement de la boucle; le programme commence à exécuter l'instruction la plus proche après le corps de la boucle;
- `continue`- se comporte comme si le programme avait soudainement atteint la fin du corps; le tour suivant est commencé et l'expression de la condition est testée immédiatement

Exemple: break

```
print("The break instruction:")  
  
for i in range(1, 6):  
    if i == 3:  
        break  
    print("a l'interieure de la boucle.", i)  
print("a l'exterieure de la boucle.")
```

Exemple:continue

```
print("\nThe continue instruction:")  
for i in range(1, 6):  
    if i == 3:  
        continue  
    print("a l'interieure de la boucle.", i)  
print("a l'exterieure de la boucle.")
```

Les boucles et la branche else

Les deux boucles `while` et `for` ont une caractéristique intéressante (et rarement utilisée).

```
i = 1
while i < 5:
    print(i)
    i += 1
else:
    print("else:", i)
```

La branche `else` de la boucle est **toujours exécutée une fois, que la boucle soit entrée dans son corps ou non** .

Les boucles et la branche else

Les boucles `for` se comportent un peu différemment.

```
for i in range(5):
```

```
    print(i)
```

```
else:
```

```
    print("else:", i)
```

La sortie peut être un peu surprenante.

La variable `i` conserve sa dernière valeur.

EXERCICES

- TP 6 : (série 6)

Structures répétitives, Les
boucles