كلية العلوم والتقنيات بطنجة

Faculté des Sciences et Techniques de Tanger

Département Génie Informatique

Module : Programmation avancée avec python

Filière : Master Sciences de données et Intelligence artificielle (IASD)

Cours préparé et enseigné par :

- Pr. Sanae KHALI ISSA

Déroulement du cours

Introduction

Chapitre 1 : Les bases de la programmation python

- Instructions de base : lecture/écriture
- Les variables
- Les opérateurs
- Structure conditionnelle
- Structure itérative / boucle
- L'instruction : break
- L'instruction : continue

Chapitre 2 : Fonctions & Modules

- Modules
- Fonctions
- Packages

Chapitre 3 : Structures de données

- Les chaines de caractères
- Les types séquentiels : listes, tuples, ensembles
- Les types de correspondance : dictionnaire

Chapitre 4 : Gestion des tableaux avec la bibliothèque **Numpy**

Chapitre 5 : Gestion des fichiers avec la bibliothèque **Pandas**

Chapitre 6 : Gestion des graphiques avec le libraire **Matplotlib**

Projet de fin de module

Introduction

Introduction

Histoire du langage python

- **1991 :** Guido van Rossum travaille aux Pays-Bas sur le projet AMOEBA : un système d'exploitation distribué. Il conçoit Python à partir du langage ABC et publie la version 0.9.0 sur un forum Usenet.
- **1996 :** sortie de **Numerical Python**, ancêtre de **numpy**
- **2001 :** naissance de la PSF (**Python Software Fundation**)

Les versions se succèdent... Un grand choix de modules est disponible, des colloques annuels sont organisés, Python est enseigné dans plusieurs universités et est utilisé en entreprise...

- **2006 :** première sortie de **IPython**
- **Fin 2008 :** sorties simultanées de **Python 2.6** et de **Python 3.0**
- **2013 :** versions en cours des branches 2 et 3 : **v2.7.3 et v3.3.0**
- **…**
- **2 Octobre 2023 :** sortie de la dernière version stable de python : **v3.12.0**

Introduction

Points forts du langage python

Langage simple

- Syntaxe claire et cohérente
- Gestion automatique de la mémoire
- Typage dynamique : pas de déclaration des types de données

Open source

- Python est libre et gratuit même pour les usages commerciaux
- Importante communauté de développeurs
- Nombreux outils standards disponibles

Langage interprété

- Langage interprété rapide
- De nombreux modules disponibles à partir des bibliothèques optimisées (souvent écrites en C ou C++)

Orienté Objet

- Modèle objet puissant mais pas obligatoire
- Structuration multifichier aisée des applications : facilite les modifications et les extensions

Travail interactif

- Nombreux interpréteurs interactifs disponibles
- Importante documentation disponible en ligne
- Développement rapide et incrémentiel
- Tests et déboggage outillés

Ouverture au monde

- Langage de haut niveau
- Interfaçable avec C/C++/FORTRAN
- Langage de script de plusieurs applications importantes : data science, développement web, application de bureau, etc.
- Excellente portabilité
- Indépendant de la machine

Extensible

- Disponibilité des bibliothèques et des packages dans tous les domaines
- Possibilité d'ajouter d'autres fonctions et modules.

Introduction

Exécution d'un programme en python

- Afin d'exécuter un programme en python, il faut installer un **interpréteur python** téléchargeable depuis le site : https://www.python.org/downloads/
- Et utiliser un **éditeur de texte** pour rédiger les script python ou utiliser un **IDE : Integrated Development Environment** tels que :

Sublime Text

Visual Studio Code

Pycharm

Anaconda

Jupyter

- Un script python est un ficher avec l'extension **.py**

Chapitre 1 : Les bases de la programmation Python

Chapitre 1 : les bases de la programmation python

La fonction : print

C'est **une fonction** qui permet **d'afficher** les éléments passés comme arguments de la fonction

Syntaxe générale

```
print (nom_variable)
print (nom_variable1, nom_variable2, ...)
print (" texte ")
print ( " texte " , nom_variable)
print ( f " texte  { nom_variable } " )
print ( " texte {} " .format(nom_variable) )
```

Exemple

```
a=12
b=200
print(a)
print(a, b)
print("c'est un exemple pour la fonction print ")
print("a=",a)
print(f"a={a} et b={b}")
print("a={} et b={}".format(a,b))
```

Chapitre 1 : les bases de la programmation python

La fonction : input

- C'est une fonction qui permet à l'utilisateur d'un programme de saisir des valeurs à l'aide de son clavier.
- La fonction **input()** renvoie une valeur de type **chaine de caractères,** pour faire des calculs sur la valeur saisie, il faut la convertir à un entier avec la fonction **int (nom_variable)** ou un réel avec la fonction **float(nom_variable)**

Syntaxe générale

nom_variable = **input ()**
nom_variable = **input** (" texte ")

Exemple

```
a=input("entrer une première valeur : ")
b=input("entrer une deuxième valeur : ")
a=int(a)
b=int(b)
print(f"la somme de {a} et {b} est {a+b} ")
```

Chapitre 1 : les bases de la programmation python

Les variables

- Une variable est **un emplacement mémoire** dans lequel on peut mémoriser une valeur.
- Une variable est identifiée par **son nom** et **son adresse.**
- Le nom d'une variable doit :
  - Etre formé des lettres (A - Z) (a - z), des chiffres (1 - 9) et des lignes de soulignement (_)
  - Commencer par une lettre.
- En python, l**e typage des données est dynamique** : le type d'une variable est déterminée selon la valeur attribuée à cette variable.
  - Pour savoir le type d'une variable, il suffit d'utiliser la fonction **type (nom_variable)**

Ex:

```
a=3
print(type(a))
<class 'int'>
```

Chapitre 1 : les bases de la programmation python

Types natifs des variables

Types prédéfinis	Signification	Exemples
int	Nombre entier	a=1, b=-2
float	Nombre réel	a=1.3
complex	Nombre complexe	a=3+5j
bool	Nombre logique	a=true , b=false
str	Chaine de caractères	a="Bonjour", b="s"

Chapitre 1 : les bases de la programmation python

Opérateurs arithmétiques

x+y	Somme de **x** et **y**
x-y	Différence de **x** et **y**
x*y	Produit de **x** et **y**
x/y	Quotient de **x** et **y**
x//y	Quotient entier de **x** et **y**
x%y	reste de la division euclidienne de **x** et **y**
xy**	**x** à la puissance **y**

Opérateurs de comparaison

<	inférieur à
<=	inférieur ou égal à
>	supérieur à
>=	supérieur ou égal à
==	égal à
!=	différent

Opérateurs logiques

and	ET logique
or	OU logique
not	Négation logique

Chapitre 1 : les bases de la programmation python

Opérateurs d'affectation

- C'est l'opération qui permet d'attribuer à une variable une valeur simple ou résultante d'une expression arithmétique :
  - Identificateur_variable = valeur simple
  - Identificateur_variable = expression arithmétique
- Une **affectation multiple** c'est attribuer à plusieurs variables une seule valeur avec une seule affectation.
  - Identificateur_variable1=Identificateur_variable2= Valeur
- Une **affectation parallèle** c'est affecter des valeurs à plusieurs variables en **parallèle**.
  - Identificateur_variable1, Identificateur_variable2= Valeur1, Valeur 2

Exemple :

```
s=p*(r**2)
a=b=c=3
x,y,z=1,2,3
```

Chapitre 1 : les bases de la programmation python

Opérateurs d'assignation

op1+= op2	**op1=op1 +op2**	Additionne les deux valeurs **op1** et **op2, le résultat est stocké dans op1**
op1-= op2	**op1= op1 – op2**	Soustrait les deux valeurs **op1** et **op2, l**e résultat est stocké dans **op1**
op1*= op2	**op1= op1*op2**	Multiplie les deux valeurs **op1** et **op2**, le résultat est stocké dans **op1**
op1/= op2	**op1= op1/op2**	Divise les deux valeurs **op1** et **op2**, le résultat est stocké dans **op1**
op1//=op2	**op1=op1//op2**	Divise les deux valeurs **op1** et **op2**, la partie entière du résultat est stocké dans **op1**
op1=op2**	**op1=op1**op2**	Calculer **op1** à la puissance **op2** et mettre le résultat dans **op1**

Exemple :

```
A+=2 #Ajouter 2 à A puis stocker la nouvelle valeur dans A
B//=5 # Diviser B sur 5 puis stocker la partie entière du résultat dans B
C**=3 # calculer C à la puissance 3 puis stocker le résultat dans C
```

Chapitre 1 : les bases de la programmation Python

Expression logique simple

- C'est une **comparaison** de deux valeurs de même type en utilisant **un opérateur de comparaison**.
- La valeur d'une expression logique est de type **booléen** (vrai ou faux)

Exemple

A==B
A<=4

Expression logique complexe

- C'est une **combinaison** entre deux expressions logiques simples en utilisant un **opérateur logique**.
- La valeur d'une expression logique complexe est de type **booléen** (vrai ou faux)

Exemple

Opérateur de comparaison — Opérateur logique — Opérateur de comparaison

$$(A + 3 < B * 2) \ \textbf{and} \ (A \ \% \ B == 0)$$

Expression logique simple — Expression logique simple

Chapitre 1 : les bases de la programmation Python

La structure conditionnelle : if

if expression logique **:**

Instruction 1
Instruction 2
…
Instruction n

Instructions à exécuter si l'expression logique est vraie

Exemple

```
if  X % 2 == 0 :
      print ( " c'est un nombre pair " )
```

Chapitre 1 : les bases de la programmation Python

La structure conditionnelle : if … else

if expression logique **:**

Instruction 1
…
Instruction n

} à exécuter si l'expression logique est vraie

else :

Instruction 1
…
Instruction n

} à exécuter si l'expression logique est fausse

Exemple

```
if X % 2 == 0 :
    print (" c'est un nombre pair ")
else :
    print (" c'est un nombre impair " )
```

Chapitre 1 : les bases de la programmation Python

La structure conditionnelle : if … elif … else

if expression logique_1 :
    Instructions à exécuter si **l'expression logique_1** est vraie
elif expression logique_2 :
    Instructions à exécuter si **l'expression logique_2** est vraie
…

elif expression logique_n :
    Instructions à exécuter si **l'expression logique_n** est vraie
else :
    Instructions à exécuter si toutes **les expressions logiques citées précédemment sont** fausses

Chapitre 1 : les bases de la programmation Python

La structure conditionnelle : if … elif … else

Exemple

```
if  X>0 :
        print (″ C'est un nombre positif ″)
elif X < 0 :
        print (″ C'est un nombre négatif ″ )
else :
        print (″ C'est un nombre nul″ )
```

Chapitre 1 : les bases de la programmation Python

L'instruction : match … case

match IdentificateurVariable :

case Valeur1 :
Instructions à exécuter si **IdentificateurVariable = Valeur1**
…

case ValeurN :
Instructions à exécuter si **IdentificateurVariable = ValeurN**

case other :
Instructions à exécuter si **IdentificateurVariable** n'appartient pas à la liste des valeurs **{Valeur1, Valeur2, … , ValeurN}**

Chapitre 1 : les bases de la programmation Python

L'instruction : match ... case

Exemple

```
choix = input ("entrer votre choix : ")
match choix :
   case "Lundi" :
       print("1er jour de la semaine ")
   case "Mardi" :
      print("2ème jour de la semaine ")
   case "Mercredi" :
      print("3ème jour de la semaine ")
   case "Jeudi" :
      print("4ème jour de la semaine « )
  case other :
      print("choix introuvable ")
```

Chapitre 1 : structure conditionnelle

Exercice d'application

Ecrire un script python qui permet d'afficher les mentions suivantes selon la valeur de la moyenne générale choisie par l'utilisateur.

- Si **moyenne >=16**, la mention affichée est : **Très Bien**
- Si **14 <= moyenne <16**, la mention affichée est : **Bien**
- Si **12 <= moyenne <14**, la mention affichée est **Assez Bien**
- Si **10 <= moyenne <12**, la mention affichée est **Passable**
- Si **moyenne <10**, l'étudiant a échoué

Chapitre 1 : structure conditionnelle

Exercice d'application

```
moyenne = float (input (" Entrer la moyenne générale :  ")
if moyenne >= 16 :
   print (" Vous avez une mention très bien ")
elif moyenne >=14 :
        print (" Vous avez une mention bien " )
elif moyenne >=12 :
        print (" Vous avez une mention assez bien " )
elif moyenne >=10 :
        print (" Vous avez une mention passable " )
else :
        print (" Vous avez échoué" )
```

Chapitre 1 : les bases de la programmation Python

Les boucles

Une boucle est un ensemble d'instructions qui se ***répètent*** un certain ***nombre de fois***.

Deux boucles sont utilisées en python :

- La boucle **for**
- La boucle **while**

Chapitre 1 : les bases de la programmation Python

La boucle for

```
for compteur in liste_valeurs :
    Instruction 1
    Instruction 2
    ...
    Instruction n
```

} à exécuter pour chaque valeur du compteur

Exemple

```
for x in [1, 2, 3, 4, 5] :
    print (" x prend la valeur  ", x)
```

Chapitre 1 : les bases de la programmation Python

La boucle for

```
for i in  "Bonjour" :
        print (i, end = "  ") # le résultat d'exécution est l'affichage des caractères  B  o  n  j  o  u  r
for x in [4, 5, 6] :
        print (x, end = "  ") # le résultat d'exécution est l'affichage des valeurs  4  5  6
for y in ["B", 15, 3.6, "Et"] :
        print (y, end = "  ") # le résultat d'exécution est l'affichage des valeurs  B  15  3.6  Et
for j in range(5) :
        print (j, end = "  ") # le résultat d'exécution est l'affichage des valeurs  0  1  2  3  4
for k in range (1, 5) :
        print (k, end = "  ") # le résultat d'exécution est l'affichage des valeurs  1  2  3  4
for k in range (1, 10, 3) :
        print (k, end = "  ") # le résultat d'exécution est l'affichage des valeurs  1  4  7
```

Chapitre 1 : les bases de la programmation Python

La boucle while

while expression logique **:**

Instruction 1
Instruction 2
…
Instruction n

} à exécuter tant que l'expression logique est vraie

Exemple

```
x=1
while x <=5 :
        print (" x prend la valeur ", x)
        x=x+1
```

Chapitre 1 : les bases de la programmation Python

La boucle while

Exercice d'application

Ecrire un script python qui demande à l'utilisateur d'introduire un ensemble des valeurs numériques puis calculer et afficher leur carré.
Le script s'arrête une fois l'utilisateur introduit la valeur 0.

Solution

```
N = int(input (" Entrer une valeur :"))
while N != 0 :
    Carre = N*N
    print (f " le carré du nombre saisi est {Carre}")
    N = int(input (" Entrer une valeur :"))
```

Chapitre 1 : les bases de la programmation Python

L'instruction break

L'instruction **break** permet de « casser » l'exécution d'une boucle (**while** ou **for**). C'est à dire, elle fait sortir de la boucle et passer à l'instruction suivante.

Exemple avec la boucle for

```
for val in "Bonjour":
    if val == "j":
        break
    print(val)
```

Résultat d'exécution

```
B
o
n
```

Exemple avec la boucle while

```
i = 0
while True:
    print(i)
    i = i + 1
    if i >= 5:
        break
```

Résultat d'exécution

```
0
1
2
3
4
```

Chapitre 1 : les bases de la programmation Python

L'instruction continue

Le mot-clé **continue** est utilisé pour terminer l'itération en cours dans une boucle for (ou une boucle while), et passer à l'itération suivante.

Exemple avec la boucle for

```
for val in "Bon":
    if val == "o":
        continue
    print(val)
```

Résultat d'exécution

```
B
n
```

Exemple avec la boucle while

```
i = 7
while i>0:
    i = i - 1
    if i == 5:
        continue
    print(i)
```

Résultat d'exécution

```
6
5
4
3
2
1
0
```

Chapitre 2 : Fonctions & Modules

Chapitre 2 : Fonctions

Définition

- Une fonction est un **bloc d'instructions** regroupées sous un **même nom** (*le choix du nom de la fonction doit répondre aux mêmes contraintes pour choisir un identificateur d'une donnée*).
- Une fonction peut avoir des **paramètres/arguments**
- Une fonction peut **renvoyer une valeur** avec le mot clé **return**
- Pour exécuter le code d'une fonction, il suffit d'écrire **son nom** avec **ses paramètres** *(dans le cas d'une fonction ayant des paramètres)*

Syntaxe générale d'une fonction avec des paramètres et ayant une valeur de retour

```
def nom_fonction (liste des paramètres) :
        Instruction 1
        Instruction 2
        …
        Instruction N
        return (nom_variable)
```

Chapitre 2 : Fonctions

Exemple 1

```
def afficher_numeros () :
        n=int(input("entrer le nombre des valeurs à afficher : "))
        for i in range(n) :
                print(i)
afficher_numeros () # appel de la fonction afficher_numeros ()
```

Résultat d'exécution

```
Entrer le nombre des valeurs à afficher : 6
0
1
2
3
4
5
```

Chapitre 2 : Fonctions

Exemple 2

```
def afficher_numeros (n) :
        for i in range(n) :
            print(i)
N=int(input("entrer le nombre des valeurs à afficher : "))
afficher_numeros (N) # appel de la fonction avec un paramètre
```

Résultat d'exécution

```
Entrer le nombre des valeurs à afficher : 6
0
1
2
3
4
5
```

Chapitre 2 : Fonctions

Exemple 3

```
def afficher_numeros (n =3 ) :
        for i in range(n) :
                print(i)
N=4
afficher_numeros (N) # appel de la fonction avec un paramètre
afficher_numeros () # appel de la fonction sans paramètre permet de prendre la valeur par défaut
```

Résultat d'exécution

```
0
1
2
3
0
1
2
```

Chapitre 2 : Fonctions

Exemple 4

```
def somme (x, y) :
        s =x+y
        print ( "la somme des deux valeurs est : ", s )

a=int(input("entrer le premier nombre : "))
b=int(input("entrer le deuxième nombre : "))
somme (a, b) # appel de la fonction somme avec deux paramètres a et b
```

Résultat d'exécution

```
12
10
La somme des deux valeurs est : 22
```

Chapitre 2 : Fonctions

Exemple 5

```
def somme (x, y) :
        s =x+y
        return (s)

a=int(input("entrer le premier nombre : "))
b=int(input("entrer le deuxième nombre : "))
print ( "la somme des deux valeurs est : ", somme (a, b))  # appel de la fonction somme
```

Résultat d'exécution

```
12
10
La somme des deux valeurs est : 22
```

Chapitre 2 : Modules

Définition

- Un projet Python est généralement composé de **plusieurs fichiers sources** ayant l'extension **.py**,
- Un **module** est un **fichier script Python** permettant de définir des éléments de programme **réutilisables** dans d'autres scripts python. Ce mécanisme permet d'élaborer efficacement des bibliothèques de fonctions ou de classes.
- ***L'utilisation des modules peut avoir plusieurs avantages à savoir :***
  - La réutilisation du code ;
  - La possibilité d'intégrer la documentation et les tests au module ;
  - La réalisation de services ou de données partagés ;

Exemples des modules standards de python

- Module **math** : il fournit un ensemble de fonctions permettant de réaliser des calculs mathématiques complexes
- Module **random** : il implémente des générateurs de nombres pseudo-aléatoires pour différentes distributions
- Module **datetime** : il fournit des classes pour manipuler de façon simple ou plus complexe des dates et des heures
- Module **sys** : il fournit un accès à certaines variables système utilisées et maintenues par l'interpréteur, et à des fonctions interagissant fortement avec ce dernier

Chapitre 2 : Modules

Importation des modules

- Afin d'importer le contenu d'un module:
  - **import nom_module** *# permet d'importer tout le contenu du module*
  - **import nom_module as nom_alias** *# permet d'importer tout le contenu du module avec la création d'un alias vers le nom de module*
  - **from nom_module import *** *# permet d'importer tous les éléments du module (méthodes et variables)*
  - **from nom_module import nom_element** *# permet d'importer un élément précis du module*

Exemples

```
import math
x=int(input(" x= ")
print(" RC= " , math.sqrt(x))
```

```
import math as mt
x=int(input(" x= " )
print(" RC= " , mt.sqrt(x))
```

```
from math import *
x=int(input(" x= " )
y=int(input(" y= " )
print(" le pgcd des deux valeurs est :",
gcd(x, y)
```

Chapitre 2 : Modules

Exemple de création et d'importation d'un module

Module calcul.py

```
def addition (x, y) :
    return (x+y)

def soustraction (x, y) :
    return (x-y)

def multiplication (x, y) :
    return (x*y)
```

Script principal

```
from calcul import *
a=float(input(" Entrer une première valeur : " ))
b=float(input(" Entrer une deuxième valeur : " ))
s=addition (a, b)
print ("La somme des deux valeurs est ", s)
d=soustraction (a, b)
print ("La différence des deux valeurs est ", d)
p=multiplication (a, b)
print ("Le produit des deux valeurs est ", p)
```

Chapitre 2 : Fonctions & Modules

Exercices : Série N1