ORSYS formation

Learning for success

VOTRE SUPPORT DE COURS

FORMATION PRESENTIELLE

Python, automatisez vos tâches bureautiques

27 - 29 Mai 2024

Bienvenue **chez ORSYS**

CE QUE VOUS DEVEZ SAVOIR

Vos deux espaces en ligne accessibles depuis un ordinateur, une tablette, un smartphone

https://docadmin.orsys.fr

- Déclarer votre présence en cours (vous devez signer chaque demi-journée).
- Valider vos acquis.
- Évaluer la formation.

Le mot de passe pour accéder à cet espace vous sera donné par le formateur.

L'évaluation est à compléter juste avant la fin de la session.

La validation des acquis est à remplir en 2 étapes, au début et à la fin de la session.

https://myorsys.orsys.fr

- Récupérer la version digitale de votre support de cours et des autres ressources pédagogiques.
- Retrouver l'historique de vos formations.
- Garder contact avec vos formateurs et les autres participants.

INFORMATIONS GÉNÉRALES

Horaires

Les sessions commencent à 9h15 le premier jour et à 9h les jours suivants. Elles se terminent à 17h30 sauf le dernier jour où elles prennent fin à 17h (excepté pour les sessions de quatre ou cinq jours où elles se terminent lors de la pause de l'après-midi).

Pour les formations en présentiel

- Nos centres de formation ouvrent à partir de 8h45.
- Les déjeuners et pauses-café vous sont offerts.
- Vous trouverez les plans d'évacuation affichés dans chaque centre.

L'accessibilité

Que vous suiviez une formation en présentiel ou à distance, n'hésitez pas à contacter notre référent handicap : psh-acceuil@orsys.fr

L'environnement

Les éco-gestes en présentiel et en distanciel.

Éteindre l'ensemble des périphériques

Utiliser nos poubelles de tri sélectif

Éteindre les lumières lors de votre départ

Garder votre gobelet pour vous hydrater

BONNE FORMATION !

Osez explorer le champ des possibles avec ORSYS FORMATION

Plus de 2000 formations dans les domaines du management, développement personnel, technologies numériques et principaux métiers de l'entreprise.

Retrouvez tous nos programmes de formation sur www.orsys.com

L'ADN ORSYS

Notre mission

Accompagner les individus et les organisations dans leur montée en compétence.

Nos valeurs

- LA QUALITÉ DE SERVICE
- L'ENGAGEMENT ET LA PERFORMANCE
- LA BIENVEILLANCE ET LA RECONNAISSANCE

160 000 PERSONNES FORMÉES

96,5 % DE TAUX DE SATISFACTION

22 000 SESSIONS ORGANISÉES

47 ANS D'EXPÉRIENCE

UN RÉSEAU DE **2 200** FORMATEURS EXPERTS DE TERRAIN

DONNÉES 2022

3 critères qualité **qui font la différence**

1. La qualité de l'offre

→ UNE OFFRE RICHE ET MULTIMODALE

40 domaines d'expertise, plus de **2 000** formations proposées en inter et intra, **600** parcours digitaux et des solutions à la carte.

→ DES INTERVENANTS EXPERTS

Professionnels de terrain, nos formateurs transmettent leur expérience métier aux participants.
Formés à notre pédagogie, ils sont rigoureusement validés par nos équipes.

→ UNE PÉDAGOGIE IMMERSIVE ET INNOVANTE

Nos ingénieurs pédagogiques et formateurs élaborent des méthodes spécifiques pour favoriser la transmission des connaissances en privilégiant la mise en pratique directe.

2. La qualité de service

→ UN INTERLOCUTEUR DÉDIÉ

Un consultant formation proche de chez vous et dédié à votre entreprise est là pour vous accompagner : étude de vos besoins, proposition de solutions adaptées en termes de pédagogie, d'organisation et de financement.

→ UN ENVIRONNEMENT OPTIMAL EN PRÉSENTIEL ET À DISTANCE

+ de 35 centres en France, en Belgique, en Suisse et au Luxembourg

- Facilités d'accès et salles de cours spacieuses, lumineuses et adaptées aux personnes en situation de handicap.
- Pour le distanciel, un accès à votre environnement adapté, simple et fiable : une connexion Internet suffit !

3. La qualité de notre organisation

→ LA QUALITÉ DE NOS PROCESSUS

ORSYS est certifié par des référentiels officiels nationaux pour la qualité de ses processus.

→ NOS ENGAGEMENTS

→ NOS CHARTES ÉTHIQUES

ORSYS formation

Learning for success

VOTRE SUPPORT DE COURS

FORMATIONS ORSYS

Toutes nos solutions

ORSYS *vous accompagne dans le développement des compétences*

→ L'INTER → L'INTRA → LE SUR-MESURE

ORSYS formation

Python tâches bureautiques

Python, automatisez vos tâches bureautiques
Dépasser les limites d'Excel, fichiers CSV avec Python

ORSYS formation

Sommaire

Cours

Annexes

ORSYS formation

Module 1: Les principes d’un programme

- Qu’est ce qu’un programme ?
- Écriture d’un programme: syntaxe et instructions
- Qu'est-ce qu’une fonction ?
- Qu’est-ce qu’une librairie ?
- Présentation du langage Python
- Installation

Qu’est ce qu’un programme ?

La programmation est le processus de création d'instructions pour un ordinateur afin qu'il effectue une tâche spécifique.

Vous donnez des instructions à une machine pour que celle-ci fasse ce que vous voulez. Ces instructions c’est ce qu’on appelle un programme.

Les programmes peuvent avoir plusieurs raisons d’être

- Automatisation des tâches.
- Résolution de problèmes.
- Manipulation de données.
- Contrôles d’appareils.

Ecriture d'un programme

Pour l'écriture d'un programme informatique, vous pouvez utiliser un logiciel spécifique qui va vous faciliter la tâche.

Cet éditeur spécifique est souvent désigné par le terme IDE (Integrated Development Environment) c'est à dire un environnement de développement.
Cet environnement nous aide dans l'écriture des programmes à l'aide des fonctionnalités suivantes

- Coloration syntaxique.
- Détection des erreurs.
- Auto complétion.
- Importation automatique des modules.
- Exécution pas à pas du programme (débogage - debugging).
- Gestion des points d'arrêt.
- Exécution du programme.

Ecriture d'un programme

Votre programme se compose généralement de plusieurs fichiers textes respectants une syntaxe particulière pour la description des instructions à effectuer.

En Python, les programmes peuvent être découpés en plusieurs fichiers. Chaque fichier représente un module.

Chaque module peut être à son tour subdivisé en plusieurs unités de traitement appelées fonctions.

Qu’est-ce qu’une fonction ?

En programmation, une fonction est une séquence d'instructions qui effectue une tâche spécifique et peut être utilisée à plusieurs endroits dans un programme.

Les fonctions sont utilisées pour organiser le code en le divisant en morceaux plus petits et autonomes.

Les fonctions peuvent prendre des données en entrée (appelées arguments) et retourner des données en sortie.

Qu'est-ce qu'une fonction ?

Elles ressemblent à des fonctions mathématiques

Exemple

f(x) = x+2

Ici f(5) = 7

5 → f(x) → 7

On peut utiliser une fonction avec d'autres arguments (exemple 10)

f(10) = 12

10 → f(x) → 12

Qu'est-ce qu’une librairie ?

Une librairie (ou bibliothèque) est un ensemble de fonctions et de modules pré-écrits qui peuvent être utilisés dans un programme.

Les librairies sont créées pour fournir des fonctionnalités spécifiques et éviter à chaque développeur de réinventer la roue.

En programmation, l'utilisation de librairies permet d'accélérer le développement en exploitant des solutions déjà mises en place pour des problèmes courants.

Ces librairies peuvent couvrir divers domaines tels que la manipulation de fichiers, la gestion des bases de données, le traitement d'images, la création d'interfaces graphiques, etc.

Présentation du langage Python

Python est un langage de programmation de haut niveau qui peut s'utiliser dans de nombreux contextes et s'adapter à tout type d'utilisation grâce à des bibliothèques spécialisées.

Python se caractérise par sa syntaxe claire et concise qui favorise la lisibilité du code. Ce qui en fait un langage facile à apprendre pour les débutants.

Voici quelques caractéristiques importantes de Python

- Lisibilité du code.
- Interprété.
- Polyvalence.
- Communauté active.
- Portabilité.
- Open source.

Présentation du langage Python

Richesse des bibliothèques

Python offre une multitude de bibliothèques prêtes à l'emploi qui étendent les fonctionnalités du langage de base.

Exemples

- `math` pour les opérations mathématiques.
- `requests` pour les requêtes web.
- `numpy` pour le traitement numérique (fonctions mathématiques).

Communauté active

La communauté Python est très active, ce qui signifie qu'il y a une abondance de ressources en ligne, de tutoriels et de forums pour obtenir de l'aide.

Installation

Suivez le document “Installation de Python”
Installation de Python

Pour

- L’installation du langage de Programmation Python.
- L’installation de l’IDE Visual Studio Code.

Ecriture d’un premier programme

Pour écrire votre premier programme, vous pouvez utiliser `Visual Studio Code`.

Créez un répertoire qui contiendra votre programme.

Ouvrez ce répertoire dans Visual Studio Code.

Créez un fichier Python d’extension *.py (exemple: `ex_001.py`) dans Visual Studio Code qui contiendra votre programme.

Le fichier Python doit contenir le code suivant

```
print("Hello Python!")
```

Ecriture d'un premier programme

Le code source est disponible ici

https://github.com/ZorroLeVrai/Python.Bureautique/blob/main/Exemples/ex_001.py

Pour exécuter le code précédent, vous pouvez utiliser cette ligne de commande
python nom_du_fichier.py

Exemple
python ex_001.py

Module 2: Les fondamentaux des langages et Python

- Qu’est-ce qu’une variable ?
- Les types de variables.
- L’opérateur d’assignation
- Les opérateurs arithmétiques
- Les opérateurs booléens
- Les opérateurs relationnels
- Récupération de valeurs d’entrée
- Cast de valeur
- Formatage
- Les structures conditionnelles
- Les structures itératives
- Les structures de données avancées
- Les fonctions
- Les modules
- Les librairies
- Les classes

ORSYS formation

Qu’est-ce qu’une variable ?

Les variables sont des symboles qui associent un nom (un identifiant) à une valeur.

Les variables ont pour but de stocker des informations dans la mémoire vive de l’ordinateur. On peut les voir comme des boîtes contenant des valeurs. Ces valeurs sont stockées pour une utilisation ultérieure.

Exemples de programme utilisant une variable (ex_002.py)

```
nom = "World"

print("Hello", nom)      #Hello World
```

Les types de variables

Les variables peuvent être de différents types.

Les types les plus fréquents sont

- Les entiers `**int**`, ils servent à stocker des nombres entiers.
- Les décimaux `**float**`, ils servent à stocker des nombres décimaux.
- Les chaînes de caractères `**str**`, ils servent à stocker du texte.
- Les booléens `**bool**`, ils servent à stocker des valeurs binaires (vraie: True ou fausse: False).
- Les vides `**None**`, c'est un type qui ne représente aucune valeur (rien), c'est ni un un 0, ni une False, ni une chaîne de caractère vide.

Les types de variables

Exemple utilisant différents types de variables (ex_003.py)

```
var_int = 1
var_float = 3.14
var_string = "Bonjour"
var_bool = True
var_vide = None

#Affichage du contenu des variables
print("var_int:", var_int)          #var_int: 1
print("var_float:", var_float)      #var_float: 3.14
print("var_string:", var_string)    #var_string: Bonjour
print("var_bool:", var_bool)        #var_bool: True
print("var_vide:", var_vide)        #var_vide: None
```

Opérateur d'assignation

L'opérateur '=' en python est l'opérateur d'assignation / affectation, il permet d'affecter à une variable une valeur donnée.

Exemple: ma_variable = 3
Permet d'assigner la valeur entière 3 à la variable `ma_variable`.

Lorsque l'on met `ma_variable` dans une instruction, Python récupére la valeur contenue dans cette variable.

Exemple
ma_variable2 = ma_variable + 1
Permet d'assigner la valeur entière 4 à la variable `ma_variable2`.

Les opérateurs arithmétiques

Il existe plusieurs types d'opérateurs : unaires, binaires, arithmétiques, logiques, d'affectation, relationnels.

Exemples d'opérateurs arithmétiques (ex_004.py)

```
print("Addition:", 1 + 2)           #Addition: 3
print("Soustraction:", 2 - 1)      #Soustraction: 1
print("Multiplication:", 2 * 3)     #Multiplication: 6
print("Division flotante:", 3 / 2)   #Division flotante: 1.5
print("Division entière:", 3 // 2)    #Division entière: 1
print("Reste de division (modulo):", 4 % 3) #Reste de division (modulo): 1
print("Puissance:", 3**2)           #Puissance: 9
```

Les opérateurs booléens

Exemples d'opérateurs booléens (ex_005.py)

```
print("Opérateur ET:", True and True)      #Opérateur ET: True
print("Opérateur OU:", True or False)      #Opérateur OU: True
print("Opérateur de négation:", not False)   #Opérateur de négation: True
```

Les opérateurs relationnels

Exemples d'opérateurs relationnels (de comparaison) (ex_006.py)

```
print("Opérateur supérieur:", 4 > 3)            #Opérateur supérieur: True
print("Opérateur inférieur:", 4 < 3)            #Opérateur inférieur: False
print("Opérateur d'égalité:", 4 == 3)           #Opérateur d'égalité: False
print("Opérateur de non égalité:", 4 != 3)      #Opérateur de non égalité: True
print("Opérateur supérieur ou égal:", 4 >= 3)   #Opérateur supérieur ou égal: True
print("Opérateur inférieur ou égal:", 4 <= 3)   #Opérateur inférieur ou égal: False
```

Récupération de valeurs d'entrée

Pour qu'il y ait un dialogue possible entre l'utilisateur et l'ordinateur (application console), on a recours à ce qui s'appelle des affichages et des récupérations de valeurs.

L'affichage se fait par la fonction print(), qui affichera les valeurs passées en paramètre sur le terminal.

La récupération se fait par la fonction input(), elle récupère une saisie de l'utilisateur sous forme de chaîne de caractères `str`. Il est possible d'y ajouter une chaîne à afficher en paramètre.

Exemple (ex_007.py)

```
nom = input("Entrez votre nom: ")
#Affiche `Bonjour` + le nom saisie précédemment
print("Bonjour", nom)
```

Récupération de valeurs d'entrée

Les fonctions d'entrée input() et d'affichage print() amènent 2 composantes essentiels à la programmation console, qui sont

- Pour la récupération: le casting des variables
- Pour l'affichage: le formatage des chaînes de caractères.

Cast de valeur

Quelques fois on a besoin de passer d'un type de variable vers un autre.

Pour ce faire, on se sert de ce qui s'appelle le cast (en français transtypage).

Pour réaliser un transtypage, il faut utiliser la fonction de cast qui porte le nom du type vers lequel on veut passer.

Exemple (ex_008.py)

```
ma_string = "123.45"
mon_prix = float(ma_string)
#La variable ma_string vaut 123.45 et est de type <class 'str'>
print(f"La variable ma_string vaut {ma_string} et est de type {type(ma_string)}")
#La variable mon_prix vaut 123.45 et est de type <class 'float'>
print(f"La variable mon_prix vaut {mon_prix} et est de type {type(mon_prix)}")
```

Formatage

Si l'on veut présenter de façon claire des informations à l'utilisateur, il faut souvent se servir du formatage.
Il existe plusieurs méthodes pour formater du texte

La méthode .format()

Cette méthode est liée au type string. Pour s'en servir il suffit de placer deux accolades {} dans une chaine de caractère qui serviront d'emplacement prévus pour l'affichage des variables

Ces marqueurs peuvent être numérotés (en commençant par 0) pour indiquer quel paramètre de la méthode sera affiché.

Les chaînes formatées

Les f-strings sont des chaînes de caractères pour formater directement le texte en y incluant entre accolades les variables

Formatage

Exemple (ex_009.py)

```
nombre_pi = 3.141592653589793
#nombre PI = 3.14
print("nombre PI = {0:0.2f}".format(nombre_pi))
print("nombre PI = {nombre_pi:0.2f}")
```

On peut également formater plusieurs paramètres (ex_010.py)

```
#Valeurs:
# 0.123
#  25
a = 0.1234567
b = 25
print(f"Valeurs:\n{a:6.3}\n{b:4}")
```

Les structures conditionnelles

Si l'on veut permettre à notre programme de prendre des chemins différents de manière conditionnelle, on a recours aux structures conditionnelles.

Les clauses if, elif et else

Pour réaliser une structure conditionnelle, on se sert des clauses/mots clés suivants

- **if**: initialise la structure de contrôle. On donne une condition et on exécute le bloc d'instruction si elle est vraie.
- **elif**: ajoute une autre condition dans le cas où la précédente n'aura pas été validée. On peut enchaîner ainsi autant de structures elif que l'on souhaite.
- **else**: toujours la dernière partie d'une structure de contrôle, son bloc est exécuté dans le cas où aucune des conditions précédentes n'a été validée.

Les structures conditionnelles

Exemple (ex_011.py)

```
temperature = float(input("Entrez la température: "))
if temperature < 0:
    print("Très froid")
elif temperature < 15:
    print("Froid")
elif temperature < 30:
    print("Tiède")
else:
    print("Chaud")
```

Le cast vers le type `float` (type décimal) à la première ligne est nécessaire car la fonction input() retourne une chaîne de caractères.

Les structures conditionnelles

La clauses match / case

Lorsque l'on travaille avec une structure conditionnelle, il est fréquent que l'on ait beaucoup de `elif` qui utilisent la même variable (ex_012.py)

```
numero = float(input("Entrez le numéro: "))
if numero == 1:
    print("Un")
elif numero == 2:
    print("Deux")
else:
    print("Supérieur à deux")
```

Les structures conditionnelles

Ce type de structure peut être écrit à l'aide de la clause match / case (ex_013.py)

```
numero = float(input("Entrez le numéro: "))
match numero:
    case 1:
        print("Un")
    case 2:
        print("Deux")
    case _:
        print("Supérieur à deux")
```

Les structures conditionnelles

L'expression ternaire

Il est possible d'utiliser une expression ternaire pour gérer les conditions.
Il s'agit d'une expression comportant une condition. On peut la comparer à une structure conditionnelle if.

Syntaxe

variable = <Valeur si vrai> if <condition> else <valeur si faux>

Exemple (ex_014.py)

```
temperature = float(input("Entrez la température de l'eau: "))
etat = "solide" if temperature < 0 else ("liquide" if temperature <= 100 else "gazeux")
print(etat)
```

Les structures itératives

Les boucles for

La boucle for…in qui sera exécutée pour chaque élément d'un ensemble de type conteneur ou interval. Elle mettra chaque élément un à un dans une variable.

Exemple (ex_015.py)

```
for element in [0, 1, 2, 3, 4]:
    print(element)

for item in range(0, 5):
    print("Itération:", item)
```

Les structures itératives

Les boucles while

La boucle `while` (tant que) qui sera exécutée tant que la condition spécifiée est vraie.

Exemple (ex_016.py)

```
# Using the while loop
iteration = 0
while iteration < 5:
    print("Loop:", iteration)
    iteration += 1

#Loop: 0
#Loop: 1
#Loop: 2
#Loop: 3
#Loop: 4
```

Les structures de données avancées

Les listes

Les listes sont les types de conteneur les plus utilisés. Elles permettent de manipuler un ensemble de données via l'utilisation de ses méthodes.

Exemple (ex_017.py)

```
liste_vide = []
ma_liste = [2,1,3]
print(ma_liste)   #[2, 1, 3]
ma_liste.sort()
print(ma_liste)   #[1, 2, 3]
ma_liste.append(4)
print(ma_liste)   #[1, 2, 3, 4]
ma_liste.extend([5,6])
print(ma_liste)   #[1, 2, 3, 4, 5, 6]
ma_liste.remove(4)
print(ma_liste)   #[1, 2, 3, 5, 6]
ma_liste.pop()
print(ma_liste)   #[1, 2, 3, 5]
```

Les structures de données avancées

Les tuples

Le tuple permet de regrouper des données, on appelle ça du packing ou construction.

Les données sont non-modifiables et identifiées par leurs indices ou index.

Exemple

```
tuple_vide = ()
mon_tuple = (1,2,3)
Mon_tuple2 = 1,2,3
```

On peut assigner plusieurs variables en une seule ligne avec une opération de unpacking

```
var1, var2 = (1, 2)
```

Les structures de données avancées

Les sets

Un set est un ensemble d'éléments uniques, les doublons sont impossibles, les valeurs doivent donc être immuable.

Lors de l'ajout ou du retrait d'un élément d'un set, le conteneur se voit ainsi automatiquement réordonner.

Lorsque l'on cast une liste en set, on obtient une série d'éléments sans doublons qui ne peuvent plus être modifiés via leur index.

Exemple (ex_018.py)

```
mon_set = {1, 2, 3, 3, 5}
print(mon_set)  #{1, 2, 3, 5}
mon_set.add(4)
print(mon_set)  #{1, 2, 3, 4, 5}
mon_set.pop()
print(mon_set)  #{2, 3, 4, 5}
```

Les structures de données avancées

Les dictionnaires

Un dictionnaire est un conteneur se servant d'une association de clés et de valeur.

Il est possible d'accéder aux valeurs qui le constituent via l'utilisation la clé associée entre crochets.

A l'aide du mot clé `del`, on peut supprimer une entrée.

Exemple (ex_019.py)

```
mon_dico = {1:'Un', 2:'Deux', 3:'Trois', 4:'Quatre'}
print(mon_dico)     #{1:'Un', 2:'Deux', 3: 'Trois', 4: 'Quatre'}
print(mon_dico[2])  #Deux
```

Les structures de données avancées

Certaines méthodes facilitent la manipulation du dictionnaire.

- .values() : récupère les valeurs
- .keys() : récupère les clés
- .items() : récupère des tuples (clés, valeur)

Exemple (ex_020_1.py)

```
mon_dico = {1:'Un', 2:'Deux', 3:'Trois', 4:'Quatre'}
print(mon_dico.values())  #dict_values(['Un', 'Deux', 'Trois', 'Quatre'])
print(mon_dico.keys())  #dict_keys([1, 2, 3, 4])
print(mon_dico.items())  #dict_items([(1, 'Un'), (2, 'Deux'), (3, 'Trois'), (4, 'Quatre')])
```

Les structures de données avancées

Vous pouvez itérer sur les valeurs d'un dictionnaire à l'aide d'une boucle `for`.

Exemple (ex_020_2.py)

```
mon_dico = {1: 'Un', 2: 'Deux', 3: 'Trois', 4: 'Quatre'}

for cle, valeur in mon_dico.items():
    print(f"{cle} -> {valeur}")

# 1 -> Un
# 2 -> Deux
# 3 -> Trois
# 4 -> Quatre
```

Les fonctions

Une fonction est un bloc de code réutilisable qui accomplit une tâche spécifique.
Elle prend généralement des données en entrée, effectue des opérations sur ces données, et renvoie un résultat.

La structure générale d'une fonction en Python est la suivante

```
def nom_de_la_fonction(parametre1, parametre2, ...):
    # Bloc de code à exécuter
    # …
    return resultat
```

- `**def**` est le mot-clé pour définir une fonction.
- `parametre1`, `parametre2`, … sont les paramètres d'entrée que la fonction peut prendre.
- Le bloc de code indenté sous la déclaration de la fonction est ce que la fonction fait.
- `**return**` est utilisé pour renvoyer un résultat à partir de la fonction.

Les fonctions

Exemple d'utilisation d'une fonction sous Python (ex_021.py)

```
def additionner(a, b):
    resultat = a + b
    return resultat

# Appel de la fonction
somme = additionner(3, 5)
print(somme)  # Affiche 8
```

La fonction s'appelle `additionner`.
Elle prend deux paramètres `a` et `b`.
À l'intérieur de la fonction, elle additionne `a` et `b` pour obtenir le résultat retourné. Elle renvoie le résultat à l'aide de la commande `return`.

Ici la fonction `additionner` prend en argument les valeurs 3 et 5.
Le résultat de la fonction est stocké dans la variable `somme`.

Les modules

Un module est un fichier contenant du code Python qui peut être réutilisé dans d'autres programmes.

Un module peut contenir des variables, des fonctions et des classes.

L'utilisation de modules permet d'organiser le code de manière modulaire, en regroupant des fonctionnalités connexes dans des fichiers séparés.

Les modules

Création d'un module

Créez un fichier Python avec l'extension `.py`. Par exemple, `mon_module.py`.

Exemple de module `mon_module.py` (ex_022_module.py)

```
def dire_bonjour(nom):
    print(f"Bonjour, {nom}!")

PI = 3.14159
```

Les modules

Utilisation d'un module

Importez le module dans votre script principal en utilisant l'instruction `import`.

Faites référence aux fonctions, variables ou classes du module en utilisant la notation dot `.`

Exemple d'utilisation dans un script principal (ex_022_script.py)

```
import ex_022_module

# Utilisation de la fonction dire_bonjour
ex_022_module.dire_bonjour("Alice")

# Utilisation de la variable PI
print(ex_022_module.PI)  # Affiche 3.14159
```

Les librairies

Une librairie (ou bibliothèque) est un ensemble de modules ou de fonctions pré-écrits qui étendent les fonctionnalités de base du langage.

Pour utiliser une librairie, vous devez l'importer dans votre script Python. Cela se fait avec l'instruction import.

Une fois la librairie importée, vous pouvez utiliser ses fonctions et modules en faisant référence à leur nom.

Exemple (ex_023.py)

```
import math
# Utilisation de la fonction sqrt (racine carrée)
resultat = math.sqrt(25)
print(resultat)  # 5.0
```

Les librairies

Certaines librairies ne sont pas incluses dans l'installation de base de Python. Vous pouvez les installer à l'aide de l'outil `pip` (le gestionnaire de paquets de Python).

Exemple pour installer la librairie `requests` qui est utilisée pour effectuer des requêtes web, vous devez taper la commande suivante dans un terminal

```
pip install requests
```

Ensuite, dans votre script Python, vous pouvez l'importer comme suit

```
import requests
```

Vous pouvez voir les fonctions qui sont définies dans une librairie à l'aide de la fonction `dir`

Exemple pour voir les fonctions définies dans la librairie `math`

```
print(dir(math))
```

Les classes

Une classe est une structure qui permet de définir et de regrouper des données et des fonctions qui sont liées entre elles.

Une classe peut être vue comme un modèle ou un plan qui décrit les caractéristiques et les comportements d'un type d'objet.

Nous pouvons par exemple créer une classe `Eleve` qui peut contenir les données d'un élève qui sont associées à différentes méthodes.

Les classes

Exemple (ex_024.py)

```
class Eleve:
    def __init__(self, nom):
        self.nom = nom
        self.notes = dict()

    def ajouter_note(self, matiere, note):
        self.notes[matiere] = note

    def supprimer_note(self, matiere):
        del self.notes[matiere]

    def voir_notes(self):
        print(self.notes)
```

Les classes

Nous pouvons créer une instance de la classe précédente à l'aide du code suivant

```
john = Eleve("John")
```

Voici un exemple d'utilisation de cette instance

```
john.ajouter_note("Math", 12)
john.ajouter_note("Français", 14)
john.voir_notes()  # {'Math': 12, 'Français': 14}

john.supprimer_note("Français")
john.voir_notes()  # {'Math': 12}
```

Module 3: Python et le traitement des données

- Qu’est-ce qu’une distribution Python ?
- Utiliser une distribution Python
- Les librairies pour un projet d’analyse de données
- Utiliser les notebooks Jupyter
- Importer des données
- Exporter des données
- Manipuler des données avec Pandas

Qu’est-ce qu’une distribution Python ?

Une distribution Python est un ensemble préconfiguré d'outils, de bibliothèques et parfois même d'environnements de développement, destiné à simplifier l'installation et la gestion des composants nécessaires pour le développement en Python.

Une des distributions Python les plus populaires est Anaconda, qui est souvent utilisée dans les domaines de la Data Science.

Utiliser une distribution Python

Lien pour installer la distribution Anaconda
https://www.anaconda.com/download

Création d'un environnement virtuel (facultatif)

Anaconda propose la gestion d'environnements virtuels. Un environnement virtuel permet d'isoler les dépendances de différents projets Python.

Vous pouvez créer un nouvel environnement virtuel en utilisant la commande `conda create`.

Exemple de création d'un environnement utilisant Python 3.8 et le package `sqlite`.
conda create --name mon_environnement python=3.8 sqlite

Vous pouvez activer l'environnement précédemment créé à l'aide de la commande `conda activate`
conda activate mon_environnement

Utiliser une distribution Python

Gestion des packets

Anaconda utilise `conda`, un gestionnaire de paquets, pour installer, mettre à jour et gérer les bibliothèques Python.

Par exemple pour installer la librairie `numpy`, vous pouvez utiliser la commande suivante
conda install numpy

Les librairies pour un projet d'analyse de données

Pour démarrer un projet de data science en Python, vous avez besoin de certaines bibliothèques populaires qui vous aideront à

- Manipuler les données
- Effectuer des analyses statistiques
- Créer des visualisations
- Mettre en œuvre des modèles d'apprentissage automatique.

Voici quelques-unes des principales bibliothèques utilisées dans le domaine de la data science, que vous pouvez installer avec l'aide de gestionnaires de paquets tels que `pip` ou `conda`.

Les librairies pour un projet d'analyse de données

NumPy

Bibliothéque utilisée pour effectuer des calculs mathématiques et scientifiques. Elle fournit également un support pour les tableaux multidimensionnels et des fonctions mathématiques pour travailler efficacement avec ces tableaux.

Pandas

Pandas offre des structures de données flexibles pour manipuler et analyser des données, notamment les DataFrames qui sont utiles pour manipuler des ensembles de données tabulaires.

Les librairies pour un projet d'analyse de données

Matplotlib

Matplotlib est une bibliothèque de visualisation qui permet de créer des graphiques statiques, des diagrammes en barres, des graphiques en nuage de points, etc

Seaborn

Seaborn est une bibliothèque de visualisation construite sur Matplotlib, offrant une interface de plus haut niveau pour créer des graphiques attrayants.

Les librairies pour un projet d'analyse de données

Scikit-learn

Scikit-learn est une bibliothèque d'apprentissage automatique (machine learning) qui offre des outils pour la classification, la régression, le clustering, la prétraitement des données, etc.

Jupyter

Jupyter est une application web qui vous permet de créer et de partager des documents qui contiennent du code, des visualisations et du texte narratif.

Vous pouvez également utiliser Jupyter Notebooks pour expérimenter et documenter vos analyses.

Utilisez les notebooks Jupyter

Anaconda inclut Jupyter Notebooks, un environnement interactif pour le développement Python.

Largement utilisés dans le domaine de la data science. Ils vous permettent de combiner du code, du texte, des images et des visualisations dans un seul document.

Installation de Jupyter

Pour installer Jupyter, vous pouvez

- soit installer la distribution Anaconda qui inclut Python et Jupyter.
- soit utiliser la commande suivante dans votre terminal
  pip install jupyter

Utilisez les notebooks Jupyter

Démarrer un Notebook

Ouvrez votre terminal ou votre invite de commandes.

Naviguez vers le répertoire où vous souhaitez créer votre notebook.

Exécutez la commande suivante pour démarrer le serveur Jupyter et ouvrir le tableau de bord dans votre navigateur
jupyter notebook

Créer un nouveau Notebook

Dans le tableau de bord Jupyter, cliquez sur le bouton "New" et choisissez "Python 3".
Cela ouvrira un nouvel onglet avec un notebook vide.

Utilisez les notebooks Jupyter

Utiliser le Notebook

Un notebook Jupyter est composé de cellules. Vous pouvez ajouter du code, du texte ou des visualisations dans ces cellules.

Pour écrire du code Python, cliquez sur une cellule, tapez votre code, puis appuyez sur Shift + Enter pour l'exécuter.

Exemple

```
print("Bonjour, Jupyter!")
```

Pour ajouter du texte formaté, choisissez le type "Markdown" dans la liste déroulante du menu. Vous pouvez utiliser la syntaxe Markdown pour formater le texte.

```
## Titre de section
Ceci est un paragraphe *en italique*.
```

Utilisez les notebooks Jupyter

Vous pouvez exécuter les cellules dans n'importe quel ordre.

Pour modifier une cellule, double-cliquez dessus.

Pour interrompre l'exécution d'un notebook, cliquez sur le bouton "Interrupt The Kernel" (le carré noir) dans la barre d'outils.

Pour réinitialiser complètement le notebook, vous pouvez utiliser le bouton "Restart the Kernel" (la flèche circulaire).

Sauvegarder et exporter

Pour sauvegarder votre notebook, cliquez sur "File" puis "Save and Checkpoint" (ou Ctrl + S).

Vous pouvez exporter le notebook dans différents formats tels que HTML, PDF ou Notebook en utilisant l'option "Download as" dans le menu "File".

Importer des données

Lecture d'un fichier texte (importation_données.ipynb)

```
# Lecture d'un fichier texte
with open('input.txt', 'r') as file:
    contenu = file.read()
    print(contenu)
```

Lecture d'un fichier JSON

```
import json

# Lecture d'un fichier JSON
with open('input.json', 'r') as file:
    donnees_json = json.load(file)

# Affichage des données
pretty_json = json.dumps(donnees_json, indent=2)
print(pretty_json)
```

Importer des données

Lecture d'un fichier CSV

```
import pandas as pd

# Lecture d'un fichier CSV
donnees_csv = pd.read_csv('input.csv')

# Affichage des premières lignes
donnees_csv.head()
```

Importer des données

Lecture d'un fichier Excel

```
import pandas as pd

# Lecture d'un fichier Excel
donnees_excel = pd.read_excel('input.xlsx')

# Affichage des premières lignes
donnees_excel.head()
```

Exporter des données

Ecriture d'un fichier texte (exportation_données.ipynb)

```
# Écriture dans un fichier texte
with open('output.txt', 'w') as file:
    file.write("Contenu à écrire dans le fichier.")
```

Ecriture d'un fichier json

```
import json
#Ecriture dans un fichier json
data = {'FirstName': 'John', 'LastName': 'Doe', 'Colonne1': [1, 2, 3]}
with open('output.json', 'w') as f:
    json.dump(data, f, indent=2)
```

Exporter des données

Ecriture d'un fichier CSV

```
import pandas as pd

# Création d'un DataFrame
data = {'Colonne1': [1, 2, 3], 'Colonne2': ['A', 'B', 'C']}
df = pd.DataFrame(data)

# Exportation vers un fichier CSV
df.to_csv('output.csv', index=False)
```

Exporter des données

Ecriture d'un fichier Excel

```
import pandas as pd

# Création d'un DataFrame
data = {'Colonne1': [1, 2, 3], 'Colonne2': ['A', 'B', 'C']}
df = pd.DataFrame(data)

# Exportation vers un fichier Excel
df.to_excel('output.xlsx', index=False)
```

Manipuler des données avec Pandas

Importation des données dans un dataframe (utilisation_pandas.ipynb)

```
#importations
import pandas as pd

#initialisation des variables
path = "./Data/eleves.csv"

#récupération des dataframes
df = pd.read_csv(path, encoding='latin1', dtype={'id': 'int', 'prenom': 'string', 'nom': 'string',
'sexe': 'string', 'classe': 'int'}, parse_dates=['date_naissance'])
```

Manipuler des données avec Pandas

Affichage du contenu des dataframes

```
#affichage des premières lignes du dataframe
df.head()


#informations sur le dataframe
df.info()

#informations sur les types des colonnes du dataframe
df.dtypes


#affichage des données statistiques du dataframe
df.describe()
```

Manipuler des données avec Pandas

Sélection des données

```
#sélection de colonnes spécifiques
df[['prenom', 'nom']]


#Affichage du nom des colonnes du dataframe
df.columns

#Filtrage de données
donnees_filtrees = df[df['sexe'] == 'Fille']
Donnees_filtrees

#Filtrage de données
donnees_filtrees2 = df[(df['date_naissance'].dt.year > 2007) & (df['sexe'] == 'Fille')]
donnees_filtrees2
```

Manipuler des données avec Pandas

Manipulation des données

```
#Ajout d'une nouvelle colonne basée sur des calculs
df['nom_prenom'] = df['prenom'] + " " + df['nom']
df

#Renommer une colonne
df.rename(columns={'prenom': 'first name'}, inplace=True)
df

#Suppression de colonnes
df.drop(['nom_prenom'], axis=1)
```

Manipuler des données avec Pandas

Transformation des données

```
#Tri des données
donnees_triees = df.sort_values(by='date_naissance')
donnees_triees

#Calculs statistiques
moyenne = df['date_naissance'].mean()
mediane = df['date_naissance'].median()
f"moyenne: {moyenne}; mediane: {mediane}"

#Appliquer une fonction personnalisée à une colonne
df['class_plus_un'] = df['classe'].apply(lambda x: x+1)
df
```

Visualisation des données avec Python

- Tracés de courbes
- Tracés d'histogrammes
- Tracés d'un Pie Chart
- Visualiser ses données sur une carte interactive
- Graphiques interactifs

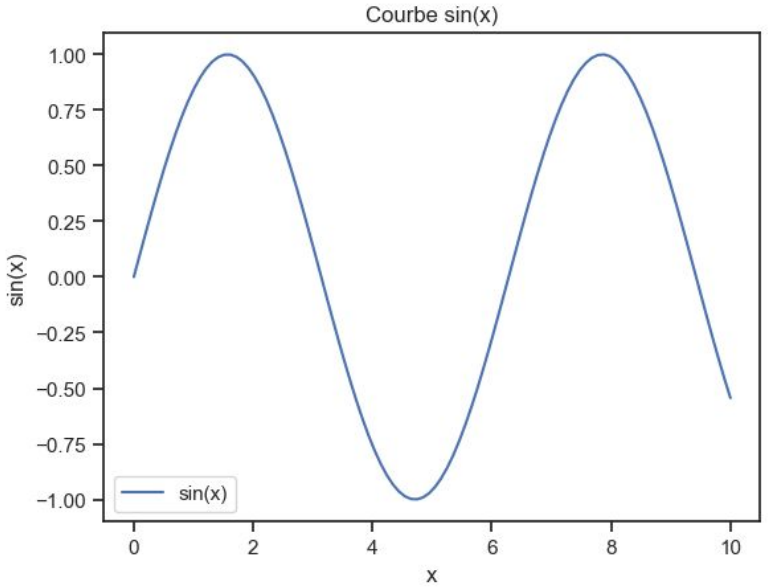
Tracés de courbes

Importation des librairies pour utiliser numpy, matplotlib et seaborn (Visualisation_de_donnees_matplotlib.ipynb)

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Tracé d'une courbe avec matplotlib

```
# Données
x = np.linspace(0, 10, 100)
y = np.sin(x)
# Tracer la courbe
plt.plot(x, y, label='sin(x)')
plt.title('Courbe sin(x)')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.legend()
plt.show()
```



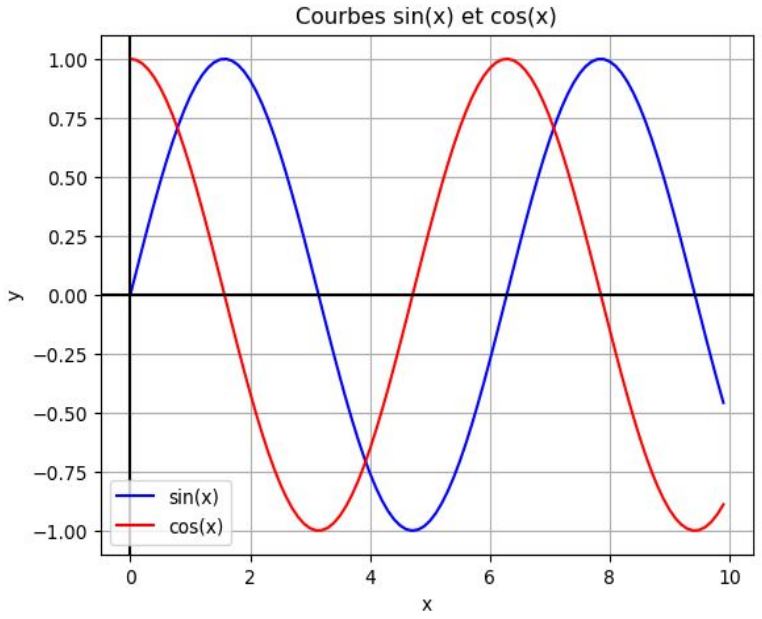
ORSYS formation

Tracés de courbes

Tracé de 2 courbes avec matplotlib

```
x = np.arange(0, 10, 0.1)
# Plusieurs courbes sur le même graphique
plt.plot(x, np.sin(x), label='sin(x)', color='blue')
plt.plot(x, np.cos(x), label='cos(x)', color='#FF0000')
plt.title('Courbes sin(x) et cos(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True, which='both')
#Affichage de l'axe
plt.axhline(y=0, color='k')
plt.axvline(x=0, color='k')

plt.legend()
plt.show()
```

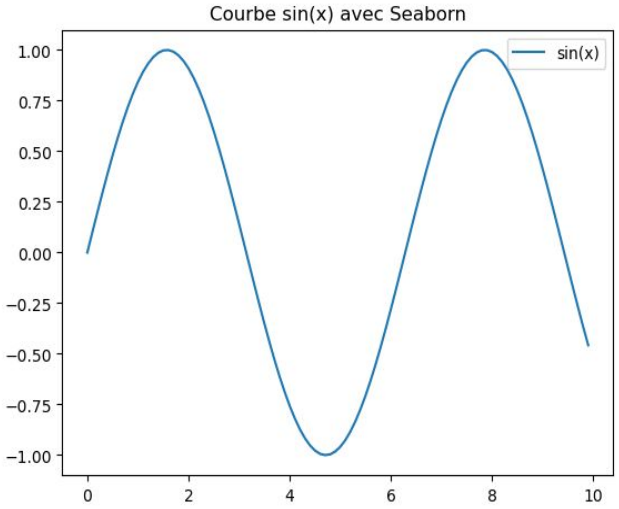


Tracés de courbes

Tracé d'une courbe avec Seaborn (Visualisation_de_donnees_seaborn.ipynb)

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Utilisation de Seaborn pour tracer une courbe
x = np.arange(0, 10, 0.1)
sns.lineplot(x=x, y=np.sin(x), label='sin(x)')
plt.title('Courbe sin(x) avec Seaborn')
plt.legend()
plt.show()
```

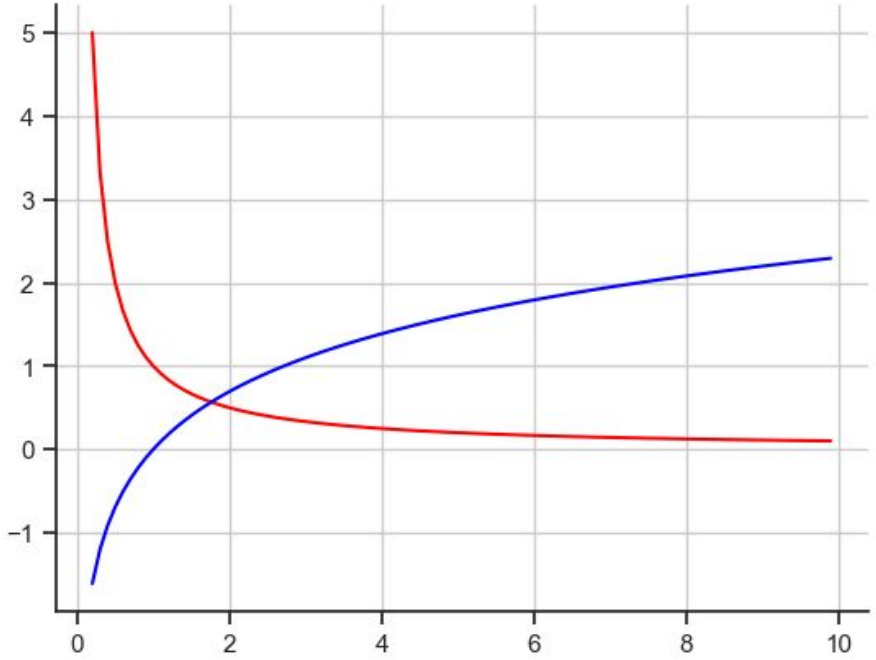


ORSYS formation

Tracés de courbes

Tracé de 2 courbes avec Seaborn

```
x = np.arange(0.2,10,0.1)
fig, ax = plt.subplots()
ax.plot(x, 1/x, color='#FF0000')
ax.plot(x, np.log(x), color='#0000FF')
ax.grid(True, which='both')
sns.despine(ax=ax, offset=0)
```



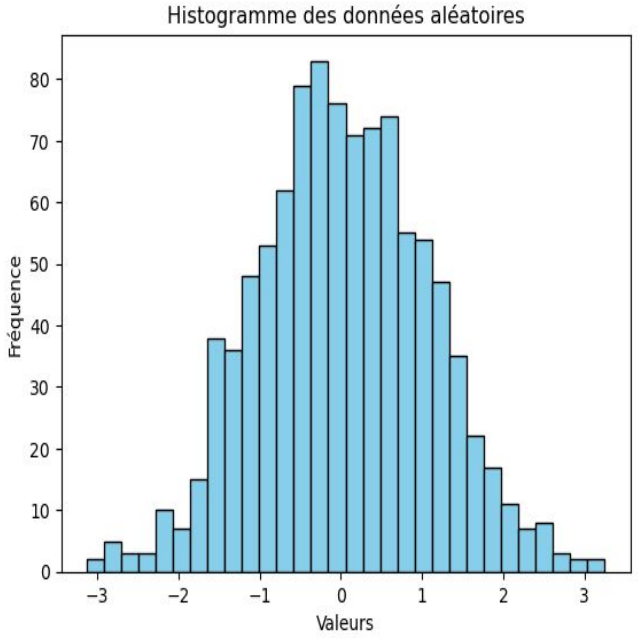
Tracés d’histogrammes

Tracé d’histogrammes avec Matplotlib

```
import numpy as np
import matplotlib.pyplot as plt

# Données
donnees = np.random.randn(1000)

# Tracer l'histogramme
plt.hist(donnees, bins=30, color='skyblue', edgecolor='black')
plt.title('Histogramme des données aléatoires')
plt.xlabel('Valeurs')
plt.ylabel('Fréquence')
plt.show()
```



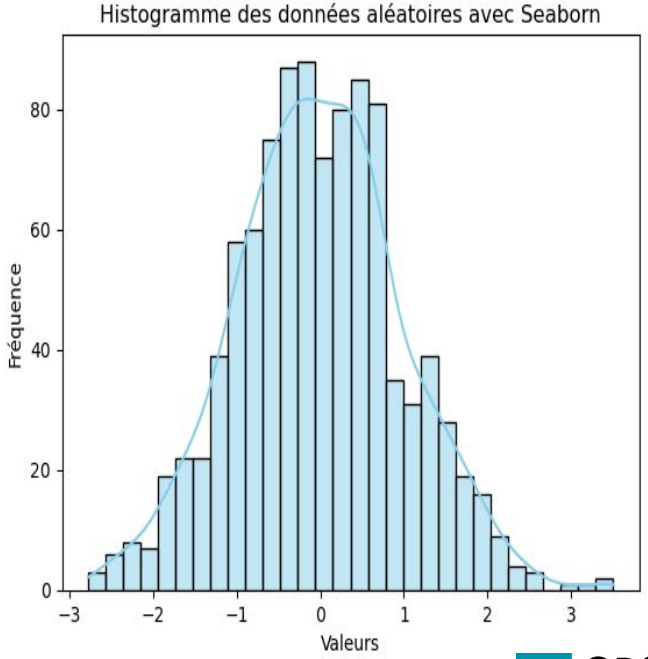
Tracés d'histogrammes

Tracé d'histogramme avec Seaborn

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#données
donnees = np.random.randn(1000)

sns.histplot(donnees, bins=30, kde=True, color='skyblue')
plt.title('Histogramme des données aléatoires avec Seaborn')
plt.xlabel('Valeurs')
plt.ylabel('Fréquence')
plt.show()
```



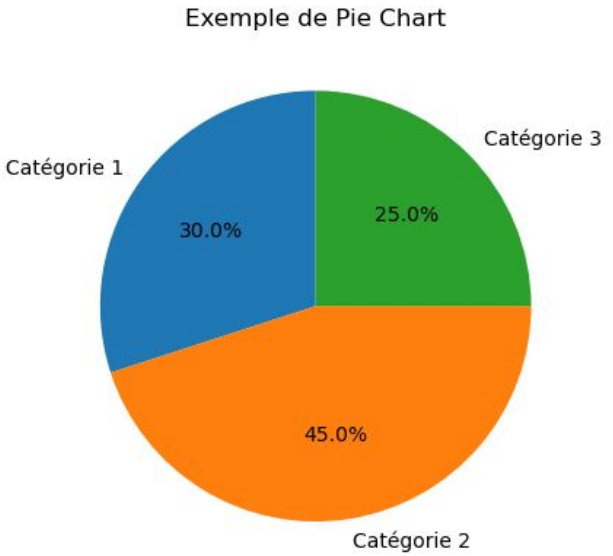
ORSYS formation

Tracés d'un Pie Chart

Tracé d'un Pie Chart avec Matplotlib

```
# Données pour le Pie Chart
labels = ['Catégorie 1', 'Catégorie 2', 'Catégorie 3']
sizes = [30, 45, 25]

# Tracer le Pie Chart
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title('Exemple de Pie Chart')
plt.show()
```



Visualiser ses données sur une carte interactive

Pour visualiser vos données sur une carte interactive, vous devez installer la librairie `folium`.

Pour installer `folium`, exécutez la commande suivante dans un terminal (en mode administrateur)
conda install folium

Une fois la librairie `folium` installée, vous devez importer la librairie dans votre Notebook Jupyter.

```
import folium
```

Visualiser ses données sur une carte interactive

Pour visualiser une carte, vous pouvez utiliser le code suivant (Cartes_interactives.ipynb)

```
latitude = 48 + 51 / 60 + 24 / 3600
longitude = 2 + 21 / 60 + 8 / 3600

# Création d'une carte centrée sur une position spécifique
carte = folium.Map(location=[latitude, longitude], zoom_start=12)

# Affichage de la carte
carte
```

Visualiser ses données sur une carte interactive

Vous pouvez ajouter des marqueurs de position à l'aide du code suivant

```
# Ajout d'un marqueur à une position spécifique
folium.Marker(location=[48+51/60+29.6/3600, 2+17/60+40.2/3600], popup='Tour Eiffel').add_to(carte)

# Ajout d'un autre marqueur
folium.Marker(location=[48+52/60+25.6/3600, 2+17/60+42.1/3600], popup='Arc de
Triomphe').add_to(carte)

# Affichage de la carte
carte
```

Graphiques interactifs

Panel est une bibliothèque Python qui facilite la création de tableaux de bord interactifs et de graphiques.

Vous pouvez importer panel à l'aide de la ligne suivante

```
import panel as pn
```

Grâce à la librairie Panel, nous pouvons associer un ou plusieurs widgets à des variables.

Ces variables ayant une incidence sur le graphique affiché, nous pouvons voir l'impact de la modification d'une variable en temps réel.

Exemples de graphiques interactifs (graphiques_interactifs.ipynb)

Graphiques interactifs

```
import numpy as np
import panel as pn
import matplotlib.pyplot as plt

#chargement de l'extension Panel
pn.extension()
x = np.linspace(0, 10, 201)
# Création du widget pour la puissance
puissance_widget = pn.widgets.FloatSlider(name='Puissance', start=0.1, end=10.0, step=0.1, value=1.0)

@pn.depends(puissance_widget.param.value)
def update_plot(puissance):
    fig, ax = plt.subplots()
    ax.plot(x, x**puissance)
    plt.close(fig)
    return fig

# Création du panneau interactif
panel_matplotlib = pn.Column(puissance_widget, update_plot)
# Affichage du panneau interactif
panel_matplotlib
```

Installation de Python

Lien à utiliser pour l'installation de Python

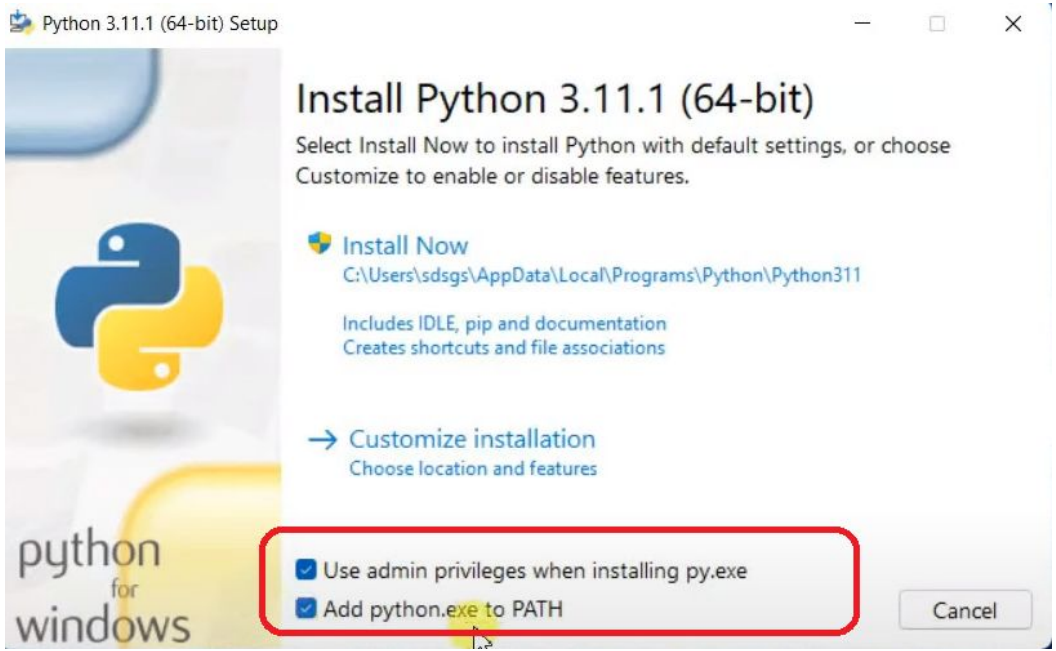
https://www.python.org/downloads/

Téléchargez la dernière version de Python.

Une fois le fichier téléchargé, exécutez le programme d'installation.

Lors de l'installation, n'oubliez de cocher la case `Ajouter Python à la variable PATH`.

Installation de Python



Installation de Python

Pour vous assurez que Python est bien installé, tapez la commande suivante dans un terminal

```
python --version
```

Vous devriez voir la version de Python qui s'affiche à l'écran.

Installation de Visual Studio Code

Lien à utiliser pour l'installation de Visual Studio Code

https://code.visualstudio.com/

Suivez les instructions pour l'installation de Visual Studio Code.

Installez les extensions suivantes pour Visual Studio Code

- Python.
- Code Runner.

Lignes de commandes Python

Compilation et exécution d'un programme Python (ici le programme `mon_programme.py`)
python mon_programme.py

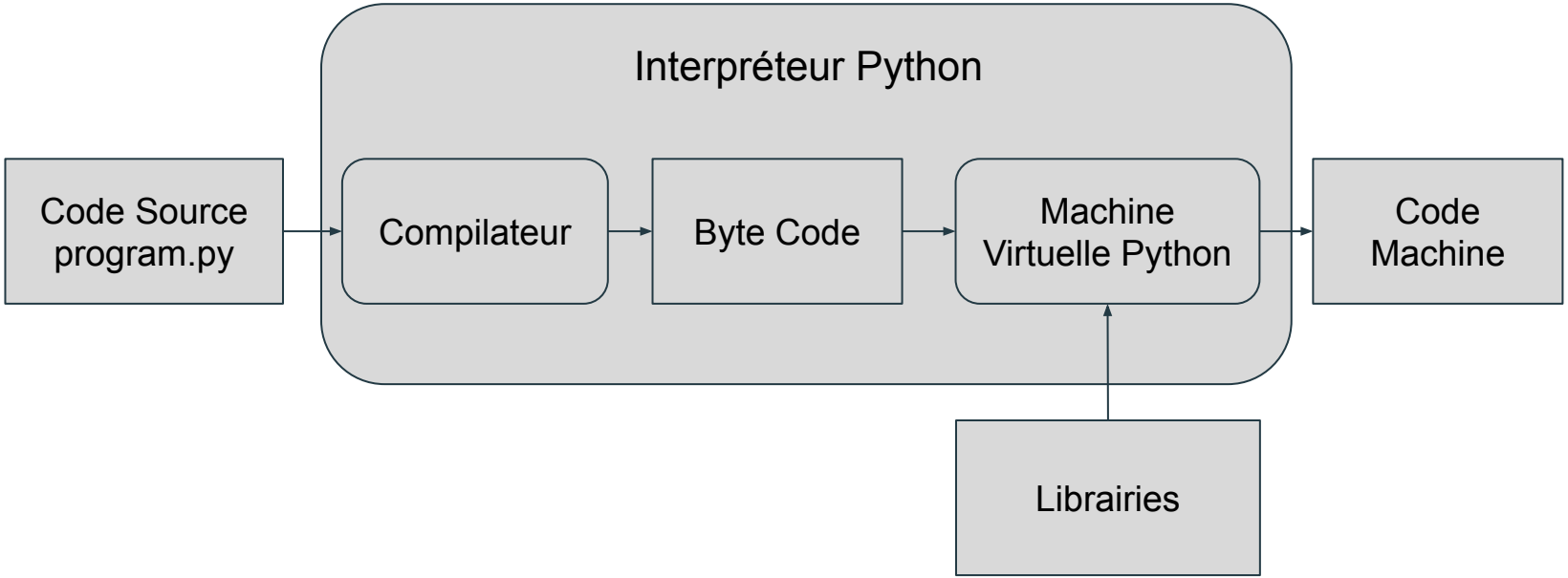
Compilation d'un programme Python en Byte Code (fichier *.pyc)
python -m py_compile mon_programme.pyc

Compilation de tous les programmes en byte code
python -m compileall .

Exécution d'un byte code
python mon_programme.cpython-312.pyc

Le principal avantage du bytecode est sa portabilité.
Comme le bytecode n'est pas lié à une architecture machine spécifique, il peut être exécuté sur n'importe quel appareil doté d'une machine virtuelle appropriée.

Schéma de compilation Python



Exercices

Exercices pour Python

Les fondamentaux - exercice 1

Ecrire un programme, qui à partir de la saisie d'un nom et prénom, affiche le message suivant:

Bonjour M. Ou Mme « Prénom » « NOM ».

Vous pouvez utiliser les méthodes

- upper(): pour mettre toute la chaîne de caractères en majuscule.
- capitalize(): pour mettre la première lettre en majuscule et le reste en minuscule.

Les fondamentaux - exercice 2

Écrire un programme qui, à partir de la saisie d'un rayon et d'une hauteur, calcule le volume d'un cône droit.

Voici la formule pour calculer le volume d'un cône

$$V = \frac{1}{3}B \times h$$

V: le volume du cône.
B: la surface de la base du cône ($\pi * r^2$)
H: la hauteur du cône

Les fondamentaux - exercice 3

Les structures conditionnelles

Écrire un programme qui, à partir de la saisie de l'âge d'une personne, affiche True si la personne est majeur et False dans le cas contraire.

Les fondamentaux - exercice 4

Les structures conditionnelles

Ecrire un programme qui prend en entrée une température et qui renvoie l'état de l'eau à cette température c'est à dire "SOLIDE", "LIQUIDE" ou "GAZEUX".

On prendra comme conditions les suivantes

- Si la température est strictement négative alors l'eau est à l'état solide.
- Si la température est entre 0 et 100 (inclus) l'eau est à l'état liquide.
- Si la température est strictement supérieure à 100 l'eau est à l'état gazeux.

Il est possible de réaliser cet exercice sans if imbriqués grâce au elif

Les fondamentaux - exercice 5

Les structures itératives

- Écrire un programme en python qui affiche les tables de multiplications de 1 à N.

N : est un entier supérieur à zéro saisie par l'utilisateur.

- Gérer l'affichage en ajoutant des espaces et en retournant à la ligne (voir ci-contre).

- Une table va de 1 à 10.

```
   1   2   3   4   5   6   7   8   9  10
--------------------------------------------------
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
   4   8  12  16  20  24  28  32  36  40
   5  10  15  20  25  30  35  40  45  50
   6  12  18  24  30  36  42  48  54  60
   7  14  21  28  35  42  49  56  63  70
   8  16  24  32  40  48  56  64  72  80
   9  18  27  36  45  54  63  72  81  90
  10  20  30  40  50  60  70  80  90 100
```

Les fondamentaux - exercice 6

Les structures itératives

On dispose d'une feuille de papier d'épaisseur 0.1 mm.

Combien de fois doit-on la plier au minimum pour que l'épaisseur dépasse 400 m ?

Écrire un programme en Python pour résoudre ce problème.

Les fondamentaux - exercice 7

Les structures itératives

Réalisez un programme permettant à l'utilisateur d'entrer comme données

- Une population initiale.
- Un taux d'accroissement.
- Une population visée.

Ce programme permettra à l'utilisateur de savoir en combien de temps la population visée sera atteinte.

Les fondamentaux - exercice 8

Les fonctions

Ecrire une fonction qui prend en paramètre

- Prenom
- Nom

Elle retournera une chaîne avec le prénom et le nom séparé d'un espace, exemple: "John Doe".

Vous afficherez le résultat de cette fonction à l'aide la fonction print().

Les fondamentaux - exercice 9

Les fonctions

Ecrire une fonction `compter_lettre_a`.

Cette fonction prendra en paramètre une chaîne de caractères.

Créer une boucle qui parcourt les lettres de la chaîne et compte le nombre de lettres égales à 'a'.

La fonction renverra un entier

- Exemple: compter_lettre_a("abba")
  résultat : 2
- Exemple : compter_lettre_a("mixer")
  résultat : 0

Traitement de données - exercice 10

Ecrire un programme permettant à un utilisateur de sauvegarder un texte secret dans un fichier.

Si le fichier n'existe pas, il devra être créé avec un nouveau secret.

L'utilisateur pourra

- Voir le secret.
- Modifier le secret.
- Quitter le programme (cette action sauvegardera le fichier).

Pour éviter tout problème, il est conseillé de ne lire et écrire le fichier qu'une seule fois à l'entrée et à la sortie du programme.

Traitement de données - exercice 11

Via l'utilisation d'une variable de type liste, vous devrez réaliser un logiciel permettant à l'utilisateur d'entrer une série de notes.

Les notes doivent être entre 0 et 20.
La saisie d'une note négative, stoppera la saisie des notes.

Une fois la saisie des notes terminée, on affichera les informations suivantes

- La note maximale.
- La note minimale.
- La moyenne des notes.

Traitement de données - exercice 12

Ecrire un script qui demande les informations d'un produit

- Titre.
- Prix.
- Stock.

Il les ajoute ensuite dans un fichier produits.csv

Traitement de données - exercice 13

Dans cet exercice nous allons créer un programme de gestion des numéros de téléphones.

Nous allons sauvegarder les données téléphoniques dans un fichier JSON.

Au démarrage de l'application si le fichier `Contacts.json` existe, nous allons charger les données de ce fichier dans un dictionnaire.

A la fermeture de l'application, les données sont sauvegardées dans le fichier `Contacts.json`.

Traitement de données - exercice 13

Lors du démarrage du programme, le menu principale suivant sera affiché

=== MENU PRINCIPAL ===

1. Voir l'annuaire

2. Ajouter un contact

3. Editer un contact

4. Supprimer un contact

5. Rechercher un numéro de téléphone

0. Quitter le programme

Le choix de l'option 1, affiche tout le contenu de l'annuaire.

Le choix de l'option 2, demande la saisie du nom et du numéro du contact.

Traitement de données - exercice 14

Calcul de prêt immobilier

Dans cet exercice vous allez

1. Calculer la mensualité d'un prêt immobilier de 100 000 € pour une période de 10 ans (120 mois) avec un taux de 3%.
2. Calculer la capacité de crédit pour un prêt d'une période de 20 ans (240 mois) avec un taux de 3% et pour une capacité de remboursement de 1200 € / mois.

Visualisation des données - exercice 15

Le but de cet exercice est d'obtenir des statistiques sur les passagers du Titanic.

A partir des fichiers `passengers.csv` et `survival.csv` disponibles ici
https://github.com/ZorroLeVrai/Python.Bureautique/tree/main/Notebooks/Exercices/Data/Titanic
créez 2 dataframes `df_passagers` et `df_survie`.

1. Joindre les 2 dataframes `df_passagers` et `df_survie` à partir de la colonne `PassengerId` pour ne former qu'un seul dataframe qui contiendra toutes les données.
2. Combien y a t-il de passagers dans notre dataframe ?
3. Quelle est la moyenne du prix du trajet ?
4. Quels passagers ont payé les billets les plus chers ?
5. Quelles personnes n'ont pas payé leurs billets de 1ère classe ?
6. Quelle personne a obtenu le billet de première classe le moins cher ?
7. Quelle est la personne la plus jeune ? et la personne la plus âgée à bord du Titanic ?
8. Quel est le prix médian des billets de 1ère, 2ème ou 3ème classe ?

Visualisation des données - exercice 15

9. Quel est le pourcentage de femmes dans la liste des passagers ?
10. Quelle est la proportion des survivants ?
11. Quelle est la proportion des survivants par classe de transport ?