Commandes Python

mardi 6 mai 2025

15:02

|  |  |  |  |
| --- | --- | --- | --- |
| Commande | Description | Exemple | Particularité |
| python3 | Permet de lancer des scripts Python | python3 mon\_script.py  python3 -m venv env | Pas besoin de le spécifier si le chemin d'accès est dans le script et qu'on place la ligne de commande suivante en tête de script -m pour spécifier qu'on souhaite utiliser un module de Python et non un fichier script |
| #!/usr/bin/env python | Ligne de commande à entrer au début de chaque script pour permettre son lancement depuis le terminal si le chemin d'accès est précisé dans le PATH | #!/usr/bin/env/python |  |
| #  """ commentaire """ | # permet de commenter le code  """ docstrings = permet de commenter sur plusieurs lignes et surtout de commenter de manière à pouvoir faire apparaitre un tooltip notamment pour les fonctions et classes | # ceci est un commentaire """ Ceci est une chaine de commentaires, utilisée pour les fonctions par exemple, et ça tient sur plusieurs lignes. """ |  |
| import from | Permet d'importer un module (fichier contenant du code Python) Il peut contenir des fonctions, des classes, des variables etc Permet d'organiser le code de manière logique et réutilisable | import mon\_module  resultat = mon\_module.ma\_fonction()  from mon\_module import ma\_fonction resultat = ma\_fonction() | Penser à spécifier le nom du module avant d'appeler la fonction.  On peut aussi importer des éléments spécifiques d'un module  Si on importe des éléments spécifiques il n'est plus nécessaire d'appeler la fonction depuis le module, voir exemple  Pour importer plusieurs éléments spécifiques, séparer ces éléments par des virgules    as pour mettre un alias de manière à rendre le code moins lourd    Ne pas mettre le .py quand on appelle l'import  import \* pour importer tout le contenu d'un package, à éviter néanmoins    **A placer en début de code** |
| as | Permet d'importer des packages avec un alias pour rendre le code plus lisible | import numpy as np |  |
| \_\_init\_\_.py | Fichier à créer pour que l'interpréteur Python comprenne que le répertoire est un package Peut contenir du code d'initialisation | from calculator.operations import multiplication    if \_\_name\_\_ == '\_\_main\_\_':  resultat = multiplication(10, 30)  print(resultat) | On peut utiliser les packages importés avec import et from pour importer des éléments spécifiques du package, voir exemple |
| pip install <nom\_du\_package> | Gestionnaire de paquets inclus dans Python pour installer des packages | pip install numpy    pip install -r requirements.txt | A utiliser dans le terminal classique, pas le python  On peut installer des versions spécifiques de package avec "==2.1.0" par exemple  ==2.1.0 pour l'exacte version 2.1.0  ~=2.2 pour toute version au dessus de 2.2 mais pas de 3.x ou utlérieure  Idem avec ~=2.1.0  >2.5.0 installera la version la plus élevée disponible au dessus de 2.5.0  "requests>2.4.0,<2.6.0" installera la version la plus élevé disponible entre les versions utilisées  versions prises au hasard pour exemple On peut installer plusieurs packages en les séparant par des espaces    -r pour spécifier un chemin requirements.txt de manière à installer tous les scripts contenus à l'intérieur du .txt (le remplir à l'aide d'un pip freeze > en amont) |
| pip uninstall <package> | Permet de désinstaller un package | pip uninstall numpy |  |
| pip freeze  pip list | Permet de voir quels packages sont déjà installés  freeze est plus pratique pour la lecture machine et liste la lecture humaine |  | A utiliser dans le terminal classique, pas le python |
| pip show | Permet de voir les infos de packages spécifiques | pip show request numpy pandas | On sépare les noms des packages avec un espace |
| print("") | Équivalent d'echo ou de printf, permet d'afficher du texte | print("Hello world!")  print(livre) | à noter que les variables sont à appeler sans ""    à noter que les strings sont à appeler avec ""  Penser à bien mettre la virgule après une string pour appeler une variable |
| input() | Interrompt le déroulement du programme afin de permettre à l'utilisateur d'entrer une donnée L'information tapée par l'utilisateur est renvoyée sous forme de string Peut être utilisée avec ou sans paramètres | prenom = input() prenom = input("Entrez votre prénom : ") print(Bonjour",prenom)  operation = int(input("Entrez une opération :")) Permet de transformer l'entrée en int directement | Penser à bien mettre la virgule après une string pour appeler une variable    **Attention :** input() renvoie toujours une string, il sera parfois nécessaire de convertir cette string avec int() pour des integer ou float() pour des virgules flottantes Voir exemple pour transformer en int dans la même commande |
| python3 xx.py | Permet de lancer un programme python depuis le terminal | python3 pystart.py | py ou python pour MS python3 pour MacOS/Linux python pour les vm |
| 1234 | On peut faire des calculs directement dans le terminal | (100+81)\*11 | Fonctionne aussi avec des espaces entre chaque symboles et valeurs |
| var = "testVariable" | Permet de déclarer une variable | livre = "Gatsby le Magnifique" livre2 = "Beloved" livre = livre2  print(livre)  Beloved | "" à utiliser pour les strings  Les int ne doivent pas être entre "" sans quoi ils seront considérés comme des strings    Fonctionne aussi sans espaces, mais rend le code moins aéré  Il suffit de faire la même manipulation en changeant juste la valeur pour modifier une variable  On peut attribuer une valeur de variable à une autre variable |
| variable | Permet d'appeler une variable | livre  Gatsby le Magnifique | On peut aussi appeler les variables sans print pour les afficher, comme ceci  On préfère utiliser print cependant |
| f-string{} | Permet d'afficher et de mélanger variables et chaines de caractères, chaine de caractères précédée d'un f (min ou maj) et contenant des expressions entre accolades | nom = "Dupont" prenom = "Jean"  age = 30    print(f"Bonjour, je m'appelle {prenom} {nom} et j'ai {age} ans.")  print(f"Le résultat de l'opération est: {round(resultat, 2)}") | Penser à bien mettre les accolades    à noter qu'on peut mélanger strings, variables, fonctions etc   A noter qu'on peut s'en servir dans quasiment tout ce qui comporte des strings (input, print, etc) |
| + | Permet de concaténer variables et strings | nom = "Dupont" prenom = "Jean"  age = 30    print(f"Bonjour, je m'appelle " + nom prenom + "et j'ai " + str(age) + " ans.") | Peu utilisé dans le cas de figure en exemple, on préfèrera toujours la f-string |
| += | Permet d'assigner une valeur à une variable | x = 5    x += 3    print(x)    8 | Équivaut à x = x + 3 |
| type | Permet de renvoyer le type d'une variable | nombre = 2 phrase = "Python c'est cool" booleen = True  type(nombre)  class'int' type(phrase) class'str' type(booleen) class'bool' |  |
| [liste] | Fonctionne un peu comme l'array (et les variables), permet de faire des listes d'objets Les listes acceptent n'importe quel type de données et même des combinaisons de type de données   On utilise l'indice pour accéder aux éléments d'une liste   A noter que les indices fonctionnent aussi avec les strings. Dans ce cas, chaque caractères correspond à un indice.   **Pour modifier une case d'une liste**, il suffit d'utiliser l'indice de la case que l'on souhaite modifier avec la nouvelle valeur et l'opérateur =  [] vides pour créer une liste vide | ["pomme", "orange", "poire"]  fruits = ["pomme", "orange", "poire"]  fruits[0] "pomme" fruits[-1] "poire"  langage = "PYTHON" langage[2] "T"  fruits[0] = "fraise"  liste\_entiers = [] | Comme toujours, les indices commencent à zéro On peut utiliser les nombres négatifs pour accéder aux derniers éléments de la liste.  On peut utiliser les valeurs des cases pour accéder aux éléments d'une liste, **mais privilégier l'utilisation de l'indice**, les valeurs contenues dans les listes sont trop changeantes  On peut créer des listes vides qu'on incrémente ensuite, voir exemple |
| append() | Méthode de liste pour ajouter une entrée à la fin d'une liste existante | fruits.append("mangue") |  |
| remove() | Méthode de liste pour retirer une entrée d'une liste existante | fruits.remove("orange") |  |
| del  pop | Pour supprimer une entrée dans une liste vis l'index  Permet aussi de supprimer une paire clé-valeur dans un dictionnaire | del fruits[1] |  |
| len() | Méthode de liste pour connaître la longueur d'une liste | len(fruits)  print(f"La liste contient {lenf(fruits)} éléments.") | Doit être contenu dans un print pour être affiché , voir exemple |
| sort() | Méthode de liste pour trier les éléments d'une liste | fruits.sort() | A noter que le tri se fait alphabétiquement pour les strings et numériquement pour les listes de nombre |
| extend() | Permet d'ajouter plusieurs élément à la fin de la liste | fruits.extend("mangue", "pêche") |  |
| insert() | Permet d'insérer un élément à une position donnée dans la liste, se construit avec deux paramètres : l'index souhaité et l'élément à insérer | fruits.insert(1, "abricot") |  |
| pop() | Permet de supprimer un élément à une position donnée de la liste, ou le dernier élément si aucun indice n'est spécifié Si on l'inclus dans une variable et qu'on appelle cette dernière, on obtient en renvoi l'élément concerné | fruits = ["pomme", "banane", "cerise"]    x = fruits.pop(1)    print(x)  banane |  |
| index() | Permet de renvoyer l'index de la première occurrence de l'élément spécifié dans la liste | fruits = ['apple', 'banana', 'cherry']    x = fruits.index("cherry")    print(x)  2 |  |
| count() | Permet de renvoyer le nombre d'occurrences de l'élément spécifié dans la liste | fruits = ["apple", "banana", "cherry", "cherry", "cherry"]    x = fruits.count("cherry")    print(x)  3 |  |
| reverse() | Inverse l'ordre des éléments de la liste | fruits = ['apple', 'banana', 'cherry']    fruits.reverse()    print(fruits)  ["cherry", "banana", "apple"] |  |
| tuple() | Permet d'enregistrer des données à l'instar des listes (et des variables) mais une fois crées les tuples sont immuables. | fruits\_tuple = ("pomme", "banane", "cerise")  legumes\_tuple = ("poireau", "brocoli", "carotte")  fruits\_et\_legumes\_tuple = fruits\_tuple + legumes\_tuple  print(fruits\_et\_legumes\_tuple) | On peut cependant concaténer deux tuples, voir exemple |
| in | Opérateur permettant de savoir si un élément est présent dans une liste ou un tuple.   Marche aussi avec not pour les négations. | nombres = [1, 2, 3, 4, 5]  5 in nombres True  8 in nombres False  if operation not in ["+", "-", "\*", "/"]:  print("Erreur: le symbole d'opération doit être '+', '-', '\*' ou '/'.")  raise SystemExit("Fin du programme") | True si présent False si manquant |
| dictionnaire={} | Permet d'enregistrer des données plus complexes que des nombres, strings ou listes.  Fonctionne avec des **paires clés-valeurs**  Ils sont indiqués par des accolades {} au début et à la fin, Les paires clés-valeurs comprennent un deux-points : placé entre la clé et la valeur et une virgule à la fin.  Chaque dictionnaire doit être composé de clés uniques | new\_campaign = { "responsable\_campagne": "Jeanne d'Arc", "nom\_camapgne": "Campagne nous aimons les chiens", "date\_debut": "01/01/2026", "influenceurs\_vip": ["@monAmourDeChien", "@meilleuresFriandisesDoggo"] } | A noter qu'il n'y a pas de virgule après la dernière paire clés-valeurs.  Des listes, truples, etc peuvent être incrémentés dans les dictionnaires (pensez aux crochets) |
| dictionnaire[]= valeur | Permet d'ajouter une paire clé-valeur dans un dictionnaire.  Permet aussi la modification des clés si on réécrit une valeur sur une clé déjà existante. | fruits["kiwi"]="vert"  fruits["kiwi"]="jaune" |  |
| dict() | Fonction intégrée pour créer un objet de type dictionnaire Utilise une autre syntaxe que précédemment (avec =), permet d'être un peu plus flexible | dictionnaire = dict(a=1, b=2)  Équivaut à ("a": 1, "b": 2) |  |
| dictionnaire[] | Permet d'accéder à une valeur dans un dictionnaire | new\_campaign["responsable\_campagne"] "Jeanne d'Arc" |  |
| key() | Permet d'obtenir une vue sur les clés du dictionnaire | car = {  "brand": "Ford",  "model": "Mustang",  "year": 1964  }    x = car.keys()    print(x)  dict\_keys(['brand', 'model', 'year']) |  |
| values() | Permet d'obtenir une vue sur les valeurs du dictionnaire | car = {  "brand": "Ford",  "model": "Mustang",  "year": 1964  }    x = car.values()    print(x)  dict\_values(['Ford', 'Mustang', 1964]) |  |
| items() | Permet d'obtenir une vue sur les paires clés-valeurs du dictionnaire | car = {  "brand": "Ford",  "model": "Mustang",  "year": 1964  }    x = car.items()    print(x)  dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)]) |  |
| get(clé) | Retourne la valeur associée à la clé spécifiée. Si la clé est non présente, retourne la valeur "none" | car = {  "brand": "Ford",  "model": "Mustang",  "year": 1964  }    x = car.get("model")    print(x)  Mustang |  |
| pop(clé) | Supprime la clé spécifiée et retourne la valeur associée. Si clé non présente, retourne la valeur "none" | car = {  "brand": "Ford",  "model": "Mustang",  "year": 1964  }    car.pop("model")    print(car)  {'brand': 'Ford', 'year': 1964} |  |
| clear() | Supprime tous les éléments du dictionnaire | car.clear() |  |
| in | Permet de vérifier si une clé spécifique existe dans un dictionnaire | "brand" in car | True si présent False si manquant |
| venv | Module permettant de créer et gérer des environnements virtuels | python3 -m venv --help  python3 -m venv <environment\_name> | Privilégier les noms usuels, env par exemple  **Il faut exécuter source environment\_name/bin/activate pour activer l'environnement et travailler à l'intérieur** |
| deactivate | Permet de quitter l'environnement virtuel en cours | deactivate |  |
| class | Permet déclarer une classe, la créer Une classe est composée de déclaration d'affectation de variables et des déclarations de fonctions | class Rectangle:  width = 3  height = 2    def calculate\_area(self):  return self.width \* self.height | On utilise le CapitalCase pour le nommage Indentation de 4 espaces pour délimiter le code de la classe Les méthodes d'instance incluent toujours "self" en premier paramètre |
| super() | Permet dans une classe enfant d'accéder au code des méthodes des classes parents | class Drink:  """Une boisson."""    def \_\_init\_\_(self, price):  """Initialise un prix."""  self.price = price    def drink(self):  """Boire la boisson."""  print("Je ne sais pas ce que c'est, mais je le bois.")      class Coffee(Drink):  """Café."""    prices = {"simple": 1, "serré": 1, "allongé": 1.5}    def \_\_init\_\_(self, type):  """Initialise le type du café."""  self.type = type  super().\_\_init\_\_(price=self.prices.get(type, 1))      def drink(self):  """Boire le café."""  print("Un bon café pour me réveiller !") |  |

Conditions Python

Penser à indenter correctement.

lundi 12 mai 2025

15:46

|  |  |  |  |
| --- | --- | --- | --- |
| Condition | Description | Exemple | Particularité |
| if | Condition qui permet d'exécuter certaines lignes de code uniquement sur la condition est vraie( True). Si la condition est fausse (False), le code ne s'exécutera pas. | if ensoleille:  print("On va à la plage !") | **On peut utiliser des opérateurs logiques pour toutes les conditions ci-dessous :**  and : vérifie si deux conditions sont toutes les deux vraies. or : vérifie si au moins une condition est vrai.  not : vérifie si une condition n'est pas vrai.  Voir exemple. |
| if/else | Permet d'exécuter du code sous certaines conditions.  Si la condition est True, le code s'arrête avant le else Si la condition est False, le code s'arrête après le else | ensoleille = True  if ensoleille:  print("On va à la plage !")  else:  print("On reste à la maison !") |  |
| if/elif/else | Permet de définir des conditions multiples.  le mot-clé elif permet d'ajouter autant de conditions que nécessaire.  On doit ensuite terminer avec une instruction else. | ensoleille = False neige = True  if ensoleille:  print("On va à la plage !")  elif neige:  print("On fait un bonhomme de neige !")  else:  print("On reste à la maison !")  Exemple avec opérateurs logique :  avec\_soleil = True  en\_semaine = False  if avec\_soleil and not en\_semaine:  print("on va à la plage !")  elif avec\_soleil and en\_semaine:  print("on va au travail !")  else:  print("on reste à la maison !") | Il est important de garder en tête l'ordre d'évaluation des conditions dans une expression conditionnelle, surtout lorsque des opérateurs "or" et "and" sont utilisés. L'opérateur "or" s'arrêtera dès qu'il trouvera une condition vraie, tandis que l'opérateur "and" s'arrêtera dès qu'il rencontrera une condition fausse. |
| expressions comparatives  Égal à : a == b    Non égal à : a != b    Moins que : a < b    Moins que ou égal à : a <= b    Plus que : a > b    Plus que ou égal à : a >= b | Permet de comparer différentes expressions entre elles et d'évaluer si une expression est vraie ou fausse. | nombre\_de\_sieges = 30  nombre\_dinvites = 25    if nombre\_dinvites < nombre\_de\_sieges:  # autoriser plus d’invités  else:  # ne pas autoriser plus d’invités | Penser à bien mettre "==" pour une égalité. Un "=" c'est pour assigner une variable. |
| + - / \* | Opérateur arithmétiques de base, sers de mille façons | if operation == "+":  resultat = nombre1 + nombre2  elif operation == "-":  resultat = nombre1 - nombre2  elif operation == "\*":  resultat = nombre1 \* nombre2  elif operation == "/":  # Vérifie si la variable `nombre2` n'est pas nulle pour la division  if nombre2 == 0:  print("Erreur: impossible de diviser par zéro.")  raise SystemExit("Fin du programme")    resultat = round(nombre1 / nombre2, 2) | \*\* pour l'exponentiation  // ou % pour le calcul de modulo    utiliser les simples quotes '+' quand c'est utilisé dans un string plutôt que de doubler les doubles quotes "+" |
| match/case | Permet de faciliter la comparaison de valeurs à l'aide de motifs.  Pour simplifier la syntaxe de certaines structures if, elif et else.  Plus concis et lisible. | fruit = "pomme"  match fruit:  case "pomme":  print("J'aime les pommes !")  case "banane":  print("Je n'aime pas les bananes.")  case "orange":  print("Les oranges sont bonnes pour la santé.")  case \_:  print("Je ne connais pas ce fruit.") | \_ pour définir une action à effectuer si aucune condition ne correspond |
| raise | Permet de faire remonter une exception ou une erreur si les conditions sont réunies, ou non justement | x = -1    if x < 0:  raise Exception("Désolé, pas de nombre en dessous de zéro.")  x = "hello"  if not type(x) is int:  raise TypeError("Seul des integer sont autorisés.") | Faire bien attention à la syntaxe, outil puissant Utilisation de type, des expressions comparatives, etc possibles  Exception pour remonter une exception TypeError pour remonter une erreur de type SystemExit pour arrêter le programme |

Boucles Python

mercredi 14 mai 2025

08:56

|  |  |  |  |
| --- | --- | --- | --- |
| Boucle | Description | Exemple | Particularité |
| for | Boucle utilisée pour itérer sur une séquence. Utilisable sur liste, tuple, dictionnaire ou même strings.  Permet d'exécuter le même code sur chaque élément d'une séquence | races\_de\_chien = ["golden retriever", "chihuahua", "terrier", "carlin"]  for chien in races\_de\_chien:  print(chien)  for x in range(5):  print(x)  0 1 2 3 4 | Dans ce cas, chien est un nom de variable qui se met à jour pour être l'élément suivant chaque fois que la boucle se répète. Garder la pertinence des noms de variable est important. |
| while | Boucle qui continue de s'exécuter jusqu'à qu'une certaine condition soit remplie | capacite\_maximale = 10  capacite\_actuelle = 3  while capacite\_actuelle < capacite\_maximale:  capacite\_actuelle += 1 | Attention à éviter les boucles infinies |
| break | Permet de sortir d'une boucle prématurément Souvent utilisée quand une condition est rencontrée et qu'on veut arrêter la boucle avant sa fin normale | for i in range(10):  if i == 5:  break  print(i) 0 1 2 3 4 |  |
| continue | Permet de passer à la prochaine itération de la boucle **sans exécuter le restant du code présent dans la boucle pour l'itération en cours (on ignore le code qui se trouve en dessous)** Souvent utilisée quand une condition est rencontrée et qu'on veut continuer la boucle sans exécuter le reste du code | liste = [1, 2, 3, 4, 5]  # Boucle for sur la liste  for element in liste:  if element == 3:  continue  print(element)  1 2 4 5 |  |
| try/except | Bloc qui permet de tester un morceau de code et de retourner une Exception si celui-ci ne fonctionne pas | while True:  try:  x = int(input("Entrez un nombre entier : "))  break  except ValueError:  print("Oops ! Ce n'est pas un nombre entier. Essayez encore...") | Penser à bien faire un exept par erreur potentiellement rencontrée, entrer dans le détail |
| with | Instruction permettant d'ouvrir un fichier et de le fermer automatiquement à la fin du bloc | with open("file.txt") as fichier:  for ligne in fichier:  # faire quelque chose avec une ligne  print(ligne) |  |

Fonctions Python

mercredi 14 mai 2025

12:51

|  |  |  |  |
| --- | --- | --- | --- |
| Fonction | Description | Exemple | Particularité |
| --- | Fonctions intégrées | --- | --- |
| help() | Permet d'afficher la docstring d'une fonction | help(isnumeric)  help(fonctionperso1) |  |
| isnumeric() | Permet de vérifier si le contenu d'une variable ou d'un morceau de code est numérique | nombre1 = 15    print(str.isnumeric(nombre1))  True  nombre1 = input("Entrez un nombre entier: ")  nombre2 = input("Entrez un nombre entier: ")    if not nombre1.isnumeric() or not nombre2.isnumeric():  print("Erreur: les deux nombres doivent être des nombres entiers") | True si toute la variable est composé de chiffres False si il y a le moindre caractère dans la variable ou le code S'utilise avec "str" en début pour les strings mais peut aussi prendre directement les variables et autres, voir exemple **Ne pas confondre numeric et integer** |
| int() | Permet de convertir une string ou autre en integer | chaine = "42"  nombre = int(chaine) print(nombre) 42  nombre1 = int(nombre1) | **On peut appeler en variable la même variable qui est converti** |
| float() | Permet de convertir une string ou autre en float | chaine = "42.5"  nombre = float(chaine) print(nombre) 42.5  nombre1 = float(nombre1) | **On peut appeler en variable la même variable qui est converti** |
| range | Permet de renvoyer une séquence de nombres allant de 0 à ce nombre moins un. | for x in range(5):  print(x)  0 1 2 3 4  for x in range(100):  print(f"{x} bouteilles de bières au mur !")  0 bouteilles de bière au mur !    1 bouteilles de bière au mur !    …    99 bouteilles de bière au mur ! | Par défaut réglée sur 0 pour la valeur de début, on peut la modifier en ajoutant un autre nombre entier (par exemple range(4, 10), cette plage renverra les valeur de 4 à 9) |
| round(,) | Permet d'arrondir le résultat d'une division Fonctionne avec le nombre à arrondir avant la virgule et le nombre de décimales à utiliser pour arrondir après la virgule.  Par défaut à zéro, donnant l'integer le plus proche | x = round(5.76543, 2)  print(x)    5.77 |  |
| split() | Permet de séparer une string en liste dans laquelle chaque mot est un objet de liste | txt = "welcome to the jungle"    x = txt.split()    print(x)    ['welcome', 'to', 'the', 'jungle']   nombres = 1,2,3,4  liste = nombres.split(",")  print(liste)  ['1', '2', '3', '4'] | On peut spécifier le separator et le maxsplit separator : spécifie le séparateur utilisé quand le split s'effectue. (espace par défaut, pratique dans le cas où les virgules sont présentes) maxsplit : spécifie combien il y a de split à faire (toutes les occurrences par défaut) |
| open("chemin du fichier", mode d'ouverture) | Fonction permettant de lire et écrire sur un fichier Fonctionne en intégrant open à une variable | fichier = open("hello.txt", "w")  fichier.write("Hello, world!")  fichier.close()    def extract(filename="input.csv"):  data = []  with open(filename, "r") as file:  csv\_reader = csv.DictReader(file) | chemin d'accès et mode d'ouverture en paramètres r pour lecture w pour écriture  a pour ajout, append, écris à la suite  r+ pour lecture et écriture Le mode d'ouverture par défaut est lecture  On peut spécifier le nom du fichier avec filename="" si on veut l'inclure dans une variable par la suite |
| .write() | Permet d'écrire dans un fichier précédemment ouvert | f = open("demofile2.txt", "a")  f.write("See you soon!")  f.close() | Le fichier doit être ouvert comme précédemment |
| .zip() | Permet de regrouper ensemble plusieurs itérables (listes, tuples etc) élément par élément en créant un nouvel itérable.  Le nouvel itérable est constitué de tuples, chacun étant constitué des éléments des itérables d'origine selon leur index respectifs | liste1 = [1, 2, 3]  liste2 = ['a', 'b', 'c']  resultat = zip(liste1, liste2)  print(list(resultat)) | Ok po |
| --- | Fonctions personnalisées définies par le développeur | --- | --- |
| def + nomfonction + (paramètres): | Permet la création d'une fonction personnalisée  Seul def et le nom de la fonction est obligatoire Ensuite on entre la fonction en dessous | Def afficher\_message():  print("Bonjour comment ça va ?")    message() | Les paramètres sont facultatifs mais très souvent utilisés    **Chaque fonction doit être responsable d’une seule et unique fonctionnalité** |
| (param1,param2) | Paramètres servant à transmettre des valeurs à la fonction | def afficher\_nom\_prenom(nom, prenom):  print("Nom :", nom)  print("Prénom :", prenom)    afficher\_nom\_prenom("Dupont", "Jean")  Nom : Dupont Prénom : Jean | **ATTENTION : les paramètres ne sont pas considérés comme des variables la virgule est obligatoire !** Néanmoins, nous pouvons intégrer les paramètres dans des variables |
| return | Mot clé permettant le renvoi d'une valeur à la fin d'une fonction pour l'utiliser par la suite dans le code | def addition(a, b):  resultat = a + b  return resultat, a, b  param1 = 20 param2 = 50  retour = addition(param1, param2) print(retour)  (70, 20, 50)  exemple complet :  def salaire\_mensuel(salaire\_annuel):  return salaire\_annuel / 12      def salaire\_hebdomadaire(salaire\_mensuel):  return salaire\_mensuel / 4      def salaire\_horaire(salaire\_hebdomadaire, heures\_travaillees):  return salaire\_hebdomadaire / heures\_travaillees      salaire\_annuel = float(input("Entrez votre salaire annuel : "))    heures\_travaillees = float(  input("Entrez le nombre d'heures travaillées par semaine : "))    mensuel = salaire\_mensuel(salaire\_annuel)  hebdomadaire = salaire\_hebdomadaire(mensuel)  horaire = salaire\_horaire(hebdomadaire, heures\_travaillees)    print("Votre salaire horaire est de", horaire, "euros.") | Si la valeur de retour est invalide, retourne un none  On peut retourner plusieurs valeurs ou variables (à séparer par des virgules), dans ce cas elles sont stockées dans un tuple |
| nomfonction() | Permet d'appeler une fonction | afficher\_message() | Si parenthèses vides lors de la création de la fonction, pas besoin de paramètres en entrée |
|  |  |  |  |

Packages Python

Privilégier les alias avec "as"

"==1.0.0" pour installer une version spécifique

**On privilégie toujours d'installer les packages nécessaires dans des environnements virtuels propres et non de manière globale**

lundi 19 mai 2025

09:12

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Package | Description | Commande associée | Description commande | Exemple |
| Os | Permet d'interagir avec le système d'exploitation | .chdir()  .mkdir()  .rmdir()  .remove()  .system() | .chdir() pour effectuer un cd  .mkdir() pour créer un nouveau dossier  .rmdir() pour supprimer un dossier  .remove() pour supprimer un fichier  .system() pour exécuter une commande system (ls, lsblk, etc)    On privilégiera os.subprocess() à os.system pour des questions de sécurité et de contrôle    .getcwd() pour afficher le dossier dans lequel on est | os.chdir("chemin/vers/repertoire")  os.mkdir("new\_directory") os.rmdir("new\_directory")  os.remove("newFile.txt")  os.system("ls") |
| Requests | Utilisé pour extraire des données à partir d'un site web | .get()  .content()  .status\_code() | .get() permet de récupérer le code html d'un site  .content() permet d'afficher le code html brut d'un site  .status\_code() permet de retourner une valeur pour définir le statut d'une URL (200 ok, 404 is not found) | import requests    url = "<https://www.gov.uk/search/news-and-communications>"  page = requests.get(url)    # Voir le code html source  print(page.content)    import requests  r = requests.get("http://www.example.com")  print(r.status\_code) |
| Beautiful Soup | Permet de parser les éléments exacts voulus dans un code HTML | with open("index.html", 'r') as file:  ····soup = BeautifulSoup(file, 'html.parser')    soup.title soup.title.string  soup.find\_all('h2') soup.find(id="titre")  soup.find\_all("li")  soup.select("li.product") | Permet d'extraire des informations du fichier "index.html", base pour le fonctionnement de BS  à noter que les url fonctionnent.    Permet d'extraire précisément les différentes entrées d'un fichier html, selon les balises utilisées | print(soup.title) |
| Csv | Permet de lire et écrire dans les fichiers CSV | import csv | .reader() prends tout le texte dans un CSV le parse ligne par ligne et convertit chaque ligne dans une liste de chaines.  a deux arguments : le fichier en question et le séparateur à utiliser  Le séparateur le plus commun est la virgule   DictReader() est sensiblement pareil que reader, sauf qu'il sait que la première ligne est un en-tête et il sauvegarde les autres lignes en tant que dictionnaires. Chaque clé est un mot de colonne et la valeur est la valeur de la colonne.    .writer() permet de créer un objet pour intéragir avec un fichier CSV  writer.writerow() permet d'écrire une ligne (liste ou itérable) dans le fichier CSV | import csv    with open('couleurs\_preferees.csv') as fichier\_csv:  reader = csv.reader(fichier\_csv, delimiter=',')  for ligne in reader:  print(ligne)  import csv    with open('couleurs\_preferees.csv') as fichier\_csv:  reader = csv.DictReader(fichier\_csv, delimiter=',')  for ligne in reader:  print(ligne['nom'] + " travaille en tant que " + ligne['metier'] + " et sa couleur préférée est " + ligne['couleur\_preferee'])  en\_tete = ["titre", "description"]    with open('data.csv', 'w') as fichier\_csv:  writer = csv.writer(fichier\_csv, delimiter=',')  writer.writerow(en\_tete)  for titre, description in zip(titres, descriptions):  ligne = [titre, description]  writer.writerow(ligne) |
| Matplotlib | Package permettant de dessiner des graphiques | .figure() .plot()  .show() | .figure() permet de choisir quel graphique utiliser .plot() type de graphique, on met en argument les valeurs à passer en graphique  .show() pour afficher le graphique, égal à print()  .title() pour modifier le titre du graphique  .scatter() pour afficher sous forme de points de couleur | import matplotlib.pyplot as plt      plt.figure()    plt.plot([1, 2, 3, 4, 5])    plt.show() |
| Numpy | Permet d'effectuer des calculs numériques avec Python | .ceil() .floor() | .ceil() pour arrondir à l'interger supérieur  .floor() pour arrondir à l'integer inférieur | import numpy      print(numpy.ceil(1.2))    print(numpy.ceil(1.933))    print(numpy.floor(1.2))    2.0  2.0  1.0 |
|  |  |  |  |  |

Classes Python

jeudi 22 mai 2025

16:36

|  |  |  |  |
| --- | --- | --- | --- |
| Commande | Description | Exemple | Particularité |
| class | Permet déclarer une classe, la créer Une classe est composée de déclaration d'affectation de variables et des déclarations de fonctions | class Rectangle:  def \_\_init\_\_(self, length, width, color="red"):  self.length = length  self.width = width  self.color = color    rectangle = Rectangle(5, 3)    rectangle = Rectangle(length=2, width=3, color="white")    rect1 = Rectangle(4, 2, "blue")  rect2 = Rectangle(3, 1, color="pink")    print(rectangle.length)    area = rectangle.calculate\_area()  print(area) | On utilise le CapitalCase pour le nommage Indentation de 4 espaces pour délimiter le code de la classe Les méthodes d'instance incluent toujours "self" en premier paramètre    Pour créer un objet, il suffit d'appeler le constructeur.  Veiller à bien renseigner tous les paramètres demandés lors de la création du constructeur  La seconde écriture pour la création fonctionne aussi, peut aider à la lecture du code.    On peut modifier le paramètre optionnel comme ci-contre    Le . signifie simplement que l'on accède à un attribut |
| \_\_init\_\_ | Constructeur permettant de déclarer des variables dynamiques, ce sont des attributs d'instance | class Rectangle:  def \_\_init\_\_(self, length, width, color="red"):  self.length = length  self.width = width  self.color = color |  |
| @classmethod | Pour les attributs de classe, mettre cls (à la place de self) en premier paramètre au sein dSe la méthode | @classmethod  def find\_bird(cls, position):  """Retrouve un oiseau selon la position donnée."""  if position in cls.positions:  return f"On a trouvé un {cls.positions[position]} !" | On privilégie les attributs d'instance sauf exception |
| @staticmethod | Pour les attributs statiques | class Bird:  """Un oiseau. 🐦"""    @staticmethod  def get\_definition():  """Donne la définition d'un oiseau."""  return (  "Animal (vertébré à sang chaud) au corps recouvert de plumes, "  "dont les membres antérieurs sont des ailes et qui a un bec."  )    print(Bird.get\_definition()) | Peu utilisé, on privilégiera toujours les attributs d'instance |
| class ClasseFille(ClasseParent) | Permet de créer une sous classe, la classe parent se trouve entre parenthèses  Sert à étendre une classe de base en lui ajoutant des attributs | class FilmCassette(Film):  """ Un film en cassette !"""" | On peut faire de l'héritage multiple en séparant les classes parents par des virgules |
| def methode(arg1) def methode(arg1) | On peut surcharger une méthode de classe en quand une classe enfant crée une nouvelle implémentation d'une méthode héritée. | from abc import ABC # permet de définir des classes de base    class Shape(ABC):  def area(self):  return 0      class Square(Shape):  def \_\_init\_\_(self, length):  self.length = length    def area(self):  return length \* length | ABC signifie Abstract Base Class ou classe de base abstraite.  C'est une classe qui ne peut être instanciée, ne peut être utilisée que via une sous classe |
| super() | Permet dans une classe enfant d'accéder au code des méthodes des classes parents | class Drink:  """Une boisson."""    def \_\_init\_\_(self, price):  """Initialise un prix."""  self.price = price    def drink(self):  """Boire la boisson."""  print("Je ne sais pas ce que c'est, mais je le bois.")      class Coffee(Drink):  """Café."""    prices = {"simple": 1, "serré": 1, "allongé": 1.5}    def \_\_init\_\_(self, type):  """Initialise le type du café."""  self.type = type  super().\_\_init\_\_(price=self.prices.get(type, 1))      def drink(self):  """Boire le café."""  print("Un bon café pour me réveiller !") | Plus besoin de définir self.name par exemple |
|  |  |  |  |

Notes

mardi 6 mai 2025

17:09

Variables :

Constituées de 3 éléments :

* Nom
* Type
* Valeur

On peut appeler le variables sans print

Les types de données les plus simples, ou *primitifs*, utilisés dans Python, sont :

* les entiers (*Integers*, en anglais) ;
* les virgules flottantes (*Floats*, en anglais) ;
* les chaînes de caractères (*Strings*, en anglais) ;
* les booléens.

Voici quelques exemples d’opérateurs arithmétiques que vous pouvez utiliser dans votre code :

* x + y  : la somme de x et y (x plus y).
* x - y  : la différence entre x et y (x moins y).
* x \* y  : le produit de x et y (x fois y).
* x / y  : le quotient de x et y (x divisé par y).
* x % y  : le reste de x divisé par y.

Pense-bête

mercredi 7 mai 2025

10:13

Texte de remplacement généré par une machine :
Nommez une variable 
Un nom de variable doit refléter son contenu, comme le nom sur un carton. Voici quelques 
recommandations générales pour choisir un nom : 
• Utilisez des noms descriptifs dans votre code. 
Vous avez déjà retrouvé un vieux classeur dans un carton avec le nom « Trucs importants » sur la 
couverture ? C'était frustrant, n'est-ce pas ? Les noms de variables descriptifs et spécifiques 
simplifient la vie, et facilitent la lecture et la modification de votre code. Au lieu de 
), ajoutez des détails : 
(ou pire, qte 
quantite_en_stock 
solde actuel 
• Utilisez des mots complets. 
quantite 
, etc. 
Évitez d'abréger ou de raccourcir les mots autant que possible, même si une alternative plus 
courte paraît évidente. Par exemple, revenu _ annuel est plus clair que rev_annuel 
• Suivez une convention d'appellation commune. Il est recommandé de suivre la convention 
d'appellation du snake case en Python, conformément à la PEP8 qui est le standard de la 
syntaxe en Python : des noms composés de plusieurs mots séparés par des tirets bas ( 
comme nombre de chats 
reponse_finale 
1e_mei11 
, etc. 
• Commencez avec une lettre ou le tiret bas. 
Un nom de variable ne peut pas commencer par un nombre. 
• Utilisez uniquement les lettres, les chiffres et le tiret du bas... et surtout pas d'accents ! 
Par exemple, écrivez bonjour_l mais pas 
• N'oubliez pas que les noms de variables sont sensibles à la casse. 
age, Age et AGE sont trois variables différentes. 

Texte de remplacement généré par une machine :
Élément n01 (une paire) 
Élément n03 
Clé + Valeur 
Clé + Valeur 
myDictionary={"Rouge" : "#FFOOOO" " 
, Vert" : "#OOFFOO" , "Bleu" : "#0000FF") 
Clé + Valeur 
Élément n02 

Texte de remplacement généré par une machine :
BOUCLE FOR 
Pour chaque élément 
en séquence 
Dernier 
Vrai 
atteint? 
Faux 
Instructions 
Fin de la 
boucle For 
Tant que le dernier élément n'est pas atteint la boucle For renvoie des instructions 

Texte de remplacement généré par une machine :
BOUCLE WHILE 
Entrer dans la boucle while 
Tester 
Faux 
l'expression 
Vrai 
Instructions 
Fonctionnement de la boucle While 
Fin de la 
boucle While 

Texte de remplacement généré par une machine :
Définissez une fonction 
Les fonctions sont des blocs de code réutilisables qui permettent d'organiser et de structurer le code, 
ainsi que de faciliter sa maintenance. En Python, la création d'une fonction se fait à l'aide du mot-clé 
def , suivi du nom de la fonction et des éventuels paramètres entre parenthèses. 
Entrée 
Fonction 
Fonction avec une entrée et une sortie 
Sortie 
Nous allons maintenant explorer trois types de fonctions différents pour découvrir toutes les 
possibilités de cette fonctionnalité 
. Les fonctions sans paramètres. 
2. Les fonctions avec paramètres. 
3. Les fonctions avec une valeur de retour. 

Texte de remplacement généré par une machine :
Une fonction avec une valeur de retour est une fonction qui peut prendre des paramètres et effectuer 
des opérations, mais qui renvoie également une valeur à la fin. Cette valeur peut être utilisée à d'autres 
endroits du programme. Par exemple, si nous avons une fonction qui calcule la somme de deux 
nombres, nous pouvons stocker le résultat dans une variable pour l'utiliser plus tard dans notre 
programme. 
Somme 
La fonction "somme" additionne les deux 
paramètres en entrée 
Découvrez la fonction "somme" 

Texte de remplacement généré par une machine :
taux de conversionl 
taux de conversion2 
taux de conversion3 
'visiteurs ' 
'visiteurs ' 
o thon 
' conversions ' ] 
' conversions ' ] 
/ ' conversions'] 
Si vous commencez à écrire du code comme ça, la première chose à faire est de le transformer en 
fonction. Vous pouvez écrire une fonction appelée , et tout faire en 
une fois. 
I def (campagne) : 
O thon 
campagne[ ' conversions ' 
taux 
taux de 
taux de 
taux de 
de conversion 
campagne[ 'visiteurs' 
taux de conversion 
conversionl 
conversion2 
conversion3 
c a I c u Ier _ t a ion ( ampagne ) 
cal culer_t a ion (seconde_c ampagne) 
cal cul version (t roi si eme_c e) 
Les deux extraits de code ci-dessus font la même chose, mais il y en a un mieux structuré que l'autre, 
et qui suit le principe DRY ! 

Texte de remplacement généré par une machine :
html* 
(IkheadSJ 
HTML 
BODY 
HEAD 
TITLE 
Le HTML est conçu avec une structure en arborescence appelée DOM. 

Texte de remplacement généré par une machine :
Récupération du titre de la page HTML: 
soup. title- 
<tit1e>Exercice extraction (un élément HTML) 
Récupération de la chaîne de caractères du titre HTML 
soup. title. string- 
"Exercice extraction HTML " 
Trouver tous les éléments avec la balise <h2> 
soup. ' h2' ) - 
renvoie une liste d 'éléments HTML 
Trouver le premier élément avec 1' id -titre- 
soup. " titre" 
>Bienvenue sur notre site web</hl> 
Trouver tous les éléments < Ii>: 
soup. find_a11 ( " li " 
Trouver tous les éléments < avec la classe 
soup. select ( " li. product" ) - 
renvoie une liste d ' éléments HTML 
"product" 
en utilisant un sélecteur CSS 

Texte de remplacement généré par une machine :
Comme pour les documents texte, les paquets Python sont "versionnés" pour faciliter le suivi des 
versions de chacun. Les versions des paquets Python, toutefois, sont gérées selon un format strict : 
thon 
I major.minar.patch 
„ Où major 
minor et patch sont tous des nombres entiers. 
Par exemple, à l'heure actuelle, la dernière version du paquet requests 
est 
2.24.e 

Texte de remplacement généré par une machine :
Chaque fois qu'un changement est apporté à un paquet Python, le numéro de version change (quelle 
que soit l'importance de la modification). Les modifications sont réparties en trois catégories 
différentes : 
• Si la modification consiste en une correction de bug rétrocompatible, le numéro de patch est 
incrémenté. 
• Si elle consiste en un élément de nouvelle fonctionnalité rétrocompatible, le numéro minor est 
incrémenté. 
• Si elle n'est pas rétrocompatible , alors major est incrémenté. Il s'agit alors d'un « breaking 
, ce qui signifie que le changement de version peut "casser" un script Python donné, 
change » 
et il sera nécessaire de réécrire ce dernier en tenant compte des changements apparus dans la 
nouvelle version du paquet. 

Texte de remplacement généré par une machine :
Pour garantir que tous les développeurs travaillant sur un projet utilisent le même environnement 
virtuel, nous utilisons un fichier requirements .txt . Il s'agit de la liste des paquets Python dont 
l'installation est requise dans un environnement virtuel pour que l'application s'exécute correctement. 
Voici un exemple de fichier requirements . txt 
o thon 
matplotlib 
n urnpy 
requests 
Si le fichier requirements.txt ci-dessus est stocké dans un projet, il indique à tout nouvel 
utilisateur : "pour commencer à travailler sur ce projet, créez et activez un environnement virtuel, puis 
installez ces paquets". Ainsi, le développeur peut commencer à travailler immédiatement sur le projet, 
en étant sûr d'avoir le même environnement de développement local que les autres développeurs du 
projet. 
Vous avez peut-être déjà constaté que, dans cet exemple, les versions des paquets ne sont pas 
indiquées. Il est préférable de spécifier une version exacte ou une plage de versions de paquets dans 
votre fichier requirements . txt . Voici un autre exemple 
2 _hon 
matpIotIib=-3 
numpy>l .12 
B requests 

Texte de remplacement généré par une machine :
La programmation orientée objet (POO) est une façon différente de penser la programmation, par 
rapport à la programmation procédurale que vous avez connue précédemment. Elle permet de 
structurer son code autrement, tout en ajoutant une couche d'abstraction supplémentaire : en 
définissant un monde d'objets proche de notre monde à nous. 
Techniquement, il s'agit de manipuler des sortes de conteneurs (les classes), qui stockent à la fois des 
valeurs brutes (appelées « propriétés ») et des fonctions (appelées « méthodes »). 
Dans le cas suivant, « Client » constitue la classe. Un Client peut avoir un nom, un numéro de 
téléphone et un compte en banque - soit un état - et vous voudrez peut-être pouvoir facturer ou 
rembourser ce client — soit un comportement. 
Client 
État (Variables) 
name 
phone_number 
bank_account 
Comportement (Méthodes) 
bill_customer(cost) 
refund_customer(amount) 

Texte de remplacement généré par une machine :
• Les classes sont des modèles pour les objets. 
• Les classes allient état (les données ou variables) et comportement (les méthodes) s'appliquant 
à ces données. 
• Un objet peut être créé et utilisé dans un programme. 

Texte de remplacement généré par une machine :
Note : en programmation orientée objet, il existe trois types d'attributs : 
• les attributs d'instance (propres aux instances créées), 
• les attributs de classe (propres à la classe, et partagés entre les instances), 
• et les attributs statiques (qui sont presque indépendants de la classe). 
Si chaque type d'attribut possède une utilité propre, essayez autant que possible de privilégier 
les attributs d'instance, qui permettent d'utiliser la programmation orientée objet à son plein 
potentiel. 

Texte de remplacement généré par une machine :
En résumé 
• Créez une classe en utilisant le mot-clé class 
• La méthode magique 
init 
est connue sous le nom de constructeur. 
• Les constructeurs sont appelés lorsqu'un objet est créé. 

Texte de remplacement généré par une machine :
Le « point » signifie simplement que vous accédez à un attribut. rectangle. length peut se lire 
comme « length of rectangle » longueur du rectangle »). 

Texte de remplacement généré par une machine :
Filrn 
Attributs 
- name 
- watch() 
FilmCassette est 
une sous-classe de Film 
Filmcassette 
Attributs 
- name 
- magnetic_tape 
- watch() 
- rewind() 
Blade Runner 
est un film 
2001 : l'Odyssée de 
l'espace est un 
FilmCassette 
Attributs 
- name : Blade Runner 
- watch() 
Attributs 
- name : '2001 : l'Odyssée 
de l'espace' 
- magnetic_tape 
- watch() 
- rewind() 
FimCassette est une sous-classe/classe enfant de Film, et Film et une superclasse/classe 
parente de FilmCassette_ 

Texte de remplacement généré par une machine :
FilmCassette partout où vous pourriez utiliser un Film dans 
Vous pouvez utiliser un 
votre code. Un FilmCassette reste avant tout un Film . Nous reviendrons là-dessus - 
ainsi que sur les types - au chapitre 5. 
A l'inverse, un Film n'est pas un FilmCassette . Que pourrait-il se passer si vous essayiez 
de mettre un objet Film dans une méthode qui est censée utiliser un 
? Un 
FilmCassette 
Film n'a pas de méthode rewind() , nous obtiendrions donc une erreur ! Plus d'informations à 
ce sujet au chapitre 5. 