Les bases du langage Python

Guide complet pour apprendre les fondamentaux de Python : syntaxe, types de données, structures de contrôle, fonctions et programmation orientée objet.

Programmation **15 min de lecture** **Niveau Débutant**

Document généré le 11/07/2026 à 20h40 · nouvy.fr/wiki/python-bases-fondamentaux

Sommaire

62 section(s) · 15 min de lecture

↳ Prochaines étapes

↳ Points clés à retenir

Introduction à Python

Python est un **langage de programmation interprété, orienté objet et multi-paradigme**. Créé par Guido van Rossum en 1991, Python est devenu l'un des langages les plus populaires au monde grâce à sa simplicité et sa polyvalence.

□ Pourquoi apprendre Python ?

- □ **Syntaxe claire et lisible** : Code facile à comprendre et à maintenir
- □ **Polyvalent** : Web, data science, IA, automatisation, scripting...
- □ **Grande communauté** : Milliers de bibliothèques et ressources
- □ **Demandé sur le marché** : Très recherché par les entreprises
- □ **Gratuit et open source** : Libre d'utilisation

□ Domaines d'utilisation

Domaine	Frameworks/Bibliothèques	Exemples d'applications
Data Science / IA	NumPy, Pandas, Scikit-learn	Analyse de données, Machine Learning
Développement Web	Django, Flask, FastAPI	Sites web, APIs REST
Automatisation	Selenium, BeautifulSoup	Scripts, web scraping
Électronique / IoT	Raspberry Pi, MicroPython	Robotique, domotique
Cybersécurité	Scapy, Nmap	Pentest, analyse réseau

Installation de Python

Windows

1. Téléchargez Python depuis : https://www.python.org/downloads/
2. Lancez l'installateur
3. ⚠ **Important** : Cochez "Add Python to PATH"
4. Cliquez sur "Install Now"

macOS

```
# Avec Homebrew (recommandé)
brew install python3

# Vérifier l'installation
python3 --version
```

Copier

Linux (Ubuntu/Debian)

```
# Installer Python 3
sudo apt update
sudo apt install python3 python3-pip -y

# Vérifier l'installation
python3 --version
pip3 --version
```

Copier

Vérification de l'installation

```
python --version
# ou
python3 --version

# Résultat attendu : Python 3.x.x
```

Copier

Premier programme Python

Mode interactif (REPL)

```
python3
```

Copier

```
>>> print("Hello, World!")
Hello, World!
>>> 2 + 2
4
>>> exit()
```

Copier

Créer un fichier Python

Créez un fichier `hello.py` :

```
print("Hello, World!")
print("Bienvenue en Python!")
```

Copier

Exécutez-le :

```
python3 hello.py
```

Copier

Syntaxe de base

Variables

En Python, pas besoin de déclarer le type d'une variable :

```
# Déclaration de variables
x = 10
name = "Alice"
pi = 3.14
is_valid = True

# Affichage
print(x)          # 10
print(name)       # Alice
print(type(x))    # <class 'int'>
print(type(name)) # <class 'str'>
```

Copier

Règles de nommage

 Autorisé :

```
my_variable = 10
_private = 20
variable123 = 30
myVariable = 40  # CamelCase (moins courant en Python)
```

Copier

 Interdit :

```
123variable = 10  # Ne peut pas commencer par un chiffre
my-variable = 20  # Tirets non autorisés
class = 30        # Mot-clé réservé
```

Copier

Commentaires

```
# Commentaire sur une ligne

"""
Commentaire
sur plusieurs
lignes
"""

x = 10  # Commentaire en fin de ligne
```

Copier

Indentation

⚠ **L'indentation est obligatoire en Python** (généralement 4 espaces) :

```
if x > 5:
    print("x est supérieur à 5")  # Indenté
    print("Encore dans le if")    # Indenté
print("Hors du if")               # Non indenté
```

Copier

Entrées utilisateur

Python utilise la fonction `input()` pour récupérer les données saisies par l'utilisateur :

```
# Demander le nom
name = input("Quel est votre nom ? ")
print(f"Bonjour {name} !")

# Demander l'âge (conversion en entier)
age = int(input("Quel est votre âge ? "))
print(f"Vous avez {age} ans")

# Demander un nombre décimal
price = float(input("Entrez un prix : "))
print(f"Prix TTC : {price * 1.20}€")
```

Copier

Gestion des erreurs d'entrée

```
# Vérifier que l'entrée est valide
try:
    age = int(input("Votre âge : "))
    print(f"Vous avez {age} ans")
except ValueError:
    print("Erreur : veuillez entrer un nombre valide")

# Boucle jusqu'à obtenir une entrée valide
while True:
    try:
        number = int(input("Entrez un nombre : "))
        break  # Sortir si conversion réussie
    except ValueError:
        print("Ce n'est pas un nombre valide, réessayez")

print(f"Vous avez saisi : {number}")
```

Copier

Types de données

Types de base (Scalaires)

1. Entiers (int)

```
x = 10
y = -5
z = 1000000
big_number = 1_000_000  # Lisibilité avec underscores

print(x + y)  # 5
print(x * y)  # -50
print(x ** 2) # 100 (puissance)
```

Copier

2. Nombres à virgule flottante (float)

```
pi = 3.14
temperature = -5.7
scientific = 1.5e3  # 1500.0 (notation scientifique)

print(pi * 2)         # 6.28
print(round(pi, 1))   # 3.1
print(int(pi))        # 3 (conversion en entier)
```

Copier

3. Chaînes de caractères (str)

```
name = "Alice"
message = 'Hello'
multiline = """Texte
sur plusieurs
lignes"""

# Concaténation
full_name = "Alice" + " " + "Dupont"  # Alice Dupont

# Répétition
repeated = "Ha" * 3  # HaHaHa

# Longueur
print(len(name))  # 5

# Accès par index
print(name[0])    # A
print(name[-1])   # e (dernier caractère)
```

Copier

4. Booléens (bool)

```
is_valid = True
is_empty = False

print(is_valid and is_empty)  # False
print(is_valid or is_empty)   # True
print(not is_valid)           # False
```

Copier

Types de séquences

1. Listes (list)

Modifiable, ordonnée, indexée

```
# Création
fruits = ["pomme", "banane", "kiwi"]
numbers = [1, 2, 3, 4, 5]
mixed = [1, "texte", 3.14, True]  # Types mixtes

# Accès
print(fruits[0])   # pomme
print(fruits[-1])  # kiwi (dernier élément)

# Modification
fruits[1] = "orange"
print(fruits)  # ['pomme', 'orange', 'kiwi']

# Ajout d'éléments
fruits.append("mangue")      # Ajoute à la fin
fruits.insert(1, "fraise")   # Insère à l'index 1

# Suppression
fruits.remove("kiwi")        # Supprime par valeur
del fruits[0]                # Supprime par index
last = fruits.pop()          # Retire et retourne le dernier

# Autres opérations
print(len(fruits))           # Longueur
print("pomme" in fruits)     # Vérifier présence
fruits.sort()                # Trier
fruits.reverse()             # Inverser
```

Copier

2. Tuples (tuple)

Non modifiable (immuable), ordonnée, indexée

```
# Création
coord = (10, 20)
person = ("Alice", 25, "Paris")

# Accès
print(coord[0])  # 10
print(coord[1])  # 20

# Décomposition (unpacking)
x, y = coord
print(x)  # 10
print(y)  # 20

# □ Erreur : les tuples sont immuables
# coord[0] = 15  # TypeError

# Utilité : retours multiples de fonctions
def get_min_max(numbers):
    return min(numbers), max(numbers)

minimum, maximum = get_min_max([1, 5, 3, 9])
print(minimum, maximum)  # 1 9
```

Copier

3. Chaînes de caractères (str) - Opérations avancées

```
text = "Hello World"

# Slicing (découpage)
print(text[0:5])    # Hello
print(text[:5])     # Hello (début implicite)
print(text[6:])     # World (fin implicite)
print(text[::2])    # HloWrd (avec pas de 2)

# Méthodes de chaînes
print(text.upper())           # HELLO WORLD
print(text.lower())           # hello world
print(text.replace("o", "0")) # Hell0 W0rld
print(text.split())           # ['Hello', 'World']

# f-strings (formatage moderne)
name = "Alice"
age = 25
print(f"Je m'appelle {name} et j'ai {age} ans")
print(f"Dans 5 ans, j'aurai {age + 5} ans")

# Autres formatages
print("Valeur: {:.2f}".format(3.14159))  # 3.14
```

Copier

4. Ranges (range)

```
# Créer une séquence de nombres
r = range(5)        # 0, 1, 2, 3, 4
print(list(r))      # [0, 1, 2, 3, 4]

r = range(1, 6)     # 1, 2, 3, 4, 5
print(list(r))      # [1, 2, 3, 4, 5]

r = range(0, 10, 2) # 0, 2, 4, 6, 8 (pas de 2)
print(list(r))      # [0, 2, 4, 6, 8]
```

Copier

Autres types de collections

1. Ensembles (set)

Non ordonné, sans doublons, modifiable

```
# Création
fruits = {"pomme", "banane", "kiwi"}
unique = set([1, 2, 2, 3, 3, 3])  # {1, 2, 3}

# Opérations
fruits.add("orange")
fruits.remove("banane")

# Opérations ensemblistes
a = {1, 2, 3}
b = {3, 4, 5}
print(a | b)  # {1, 2, 3, 4, 5} (union)
print(a & b)  # {3} (intersection)
print(a - b)  # {1, 2} (différence)
```

Copier

2. Dictionnaires (dict)

Paires clé-valeur, modifiable, non ordonné (avant Python 3.7)

```
# Création
person = {
    "name": "Alice",
    "age": 25,
    "city": "Paris"
}

# Accès
print(person["name"])        # Alice
print(person.get("age"))     # 25
print(person.get("email", "Non défini"))  # Valeur par défaut

# Modification
person["age"] = 26
person["email"] = "alice@example.com"

# Suppression
del person["city"]

# Parcourir un dictionnaire
for key in person:
    print(f"{key}: {person[key]}")

for key, value in person.items():
    print(f"{key}: {value}")

# Méthodes utiles
print(person.keys())    # dict_keys(['name', 'age', 'email'])
print(person.values())  # dict_values(['Alice', 26, 'alice@example.com'])
print(len(person))      # 3
```

Copier

Opérateurs

Opérateurs arithmétiques

```
x = 10
y = 3

print(x + y)   # 13 (addition)
print(x - y)   # 7 (soustraction)
print(x * y)   # 30 (multiplication)
print(x / y)   # 3.333... (division)
print(x // y)  # 3 (division entière)
print(x % y)   # 1 (modulo - reste)
print(x ** y)  # 1000 (puissance)
```

Copier

Opérateurs de comparaison

```
x = 10
y = 5

print(x == y)  # False (égal)
print(x != y)  # True (différent)
print(x > y)   # True (supérieur)
print(x < y)   # False (inférieur)
print(x >= y)  # True (supérieur ou égal)
print(x <= y)  # False (inférieur ou égal)
```

Copier

Opérateurs logiques

```
a = True
b = False

print(a and b)  # False
print(a or b)   # True
print(not a)    # False

# Combinaison
x = 10
print(x > 5 and x < 15)  # True
print(x < 5 or x > 15)   # False
```

Copier

Opérateurs d'appartenance

```
fruits = ["pomme", "banane", "kiwi"]

print("pomme" in fruits)     # True
print("orange" not in fruits) # True
```

Copier

Structures de contrôle

Condition if / elif / else

```
age = 18

if age < 18:
    print("Mineur")
elif age == 18:
    print("Tout juste majeur")
else:
    print("Majeur")

# Forme condensée (ternaire)
status = "Majeur" if age >= 18 else "Mineur"
print(status)
```

Copier

Boucle for

```
# Parcourir une liste
fruits = ["pomme", "banane", "kiwi"]
for fruit in fruits:
    print(fruit)

# Parcourir un range
for i in range(5):
    print(i)  # 0, 1, 2, 3, 4

# Avec index et valeur
for index, fruit in enumerate(fruits):
    print(f"{index}: {fruit}")

# Parcourir un dictionnaire
person = {"name": "Alice", "age": 25}
for key, value in person.items():
    print(f"{key} = {value}")
```

Copier

Boucle while

```
count = 0
while count < 5:
    print(count)
    count += 1

# Boucle infinie (avec break)
while True:
    response = input("Continuer? (o/n): ")
    if response == "n":
        break
    print("On continue!")
```

Copier

Break, Continue, Pass

```
# break : sortir de la boucle
for i in range(10):
    if i == 5:
        break
    print(i)  # 0, 1, 2, 3, 4

# continue : passer à l'itération suivante
for i in range(5):
    if i == 2:
        continue
    print(i)  # 0, 1, 3, 4

# pass : ne rien faire (placeholder)
for i in range(5):
    if i == 2:
        pass  # TODO: à implémenter plus tard
    print(i)
```

Copier

Match Case (Python 3.10+)

Le `match case` est l'équivalent du switch/case dans d'autres langages :

```
# Syntaxe de base
def handle_status(status):
    match status:
        case "success":
            return "Opération réussie"
        case "error":
            return "Une erreur s'est produite"
        case "warning":
            return "Attention requise"
        case _:  # case par défaut
            return "Statut inconnu"

print(handle_status("success"))  # Opération réussie
print(handle_status("error"))   # Une erreur s'est produite
```

Copier

Patterns avancés

```
# Pattern avec plusieurs valeurs
def get_day_type(day):
    match day.lower():
        case "lundi" | "mardi" | "mercredi" | "jeudi" | "vendredi":
            return "Jour de semaine"
        case "samedi" | "dimanche":
            return "Weekend"
        case _:
            return "Jour invalide"

# Pattern avec conditions
def check_number(num):
    match num:
        case n if n < 0:
            return "Négatif"
        case 0:
            return "Zéro"
        case n if n > 0 and n < 10:
            return "Petit nombre positif"
        case n if n >= 10:
            return "Grand nombre positif"
```

Copier

Avec des classes

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

def analyze_point(point):
    match point:
        case Point(0, 0):
            return "Point à l'origine"
        case Point(x, 0):
            return f"Point sur l'axe X: {x}"
        case Point(0, y):
            return f"Point sur l'axe Y: {y}"
        case Point(x, y) if x == y:
            return f"Point diagonal: {x}"
        case Point(x, y):
            return f"Point général: ({x}, {y})"

p1 = Point(0, 0)
p2 = Point(3, 3)
p3 = Point(2, 5)

print(analyze_point(p1))  # Point à l'origine
print(analyze_point(p2))  # Point diagonal: 3
print(analyze_point(p3))  # Point général: (2, 5)
```

Copier

Fonctions

Définition et appel

```
# Fonction simple
def greet():
    print("Hello!")

greet()  # Hello!

# Fonction avec paramètres
def greet_person(name):
    print(f"Bonjour {name}!")

greet_person("Alice")  # Bonjour Alice!

# Fonction avec retour
def add(a, b):
    return a + b

result = add(5, 3)
print(result)  # 8
```

Copier

Paramètres par défaut

```
def greet(name, message="Bonjour"):
    print(f"{message} {name}!")

greet("Alice")              # Bonjour Alice!
greet("Bob", "Salut")      # Salut Bob!
```

Copier

Arguments nommés

```
def create_user(name, age, city):
    print(f"{name}, {age} ans, habite à {city}")

# Appel avec arguments nommés
create_user(name="Alice", age=25, city="Paris")
create_user(city="Lyon", name="Bob", age=30)
```

Copier

Arguments variables

```
# *args : nombre variable d'arguments positionnels
def sum_all(*numbers):
    total = 0
    for num in numbers:
        total += num
    return total

print(sum_all(1, 2, 3))        # 6
print(sum_all(1, 2, 3, 4, 5)) # 15

# **kwargs : nombre variable d'arguments nommés
def display_info(**info):
    for key, value in info.items():
        print(f"{key}: {value}")

display_info(name="Alice", age=25, city="Paris")
```

Copier

Fonctions lambda

```
# Fonction anonyme (sur une ligne)
add = lambda x, y: x + y
print(add(5, 3))  # 8

# Utilisation avec map, filter
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, numbers))
print(squared)  # [1, 4, 9, 16, 25]

even = list(filter(lambda x: x % 2 == 0, numbers))
print(even)  # [2, 4]
```

Copier

Programmation orientée objet (POO)

Classes et objets

```
# Définir une classe
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def greet(self):
        print(f"Bonjour, je suis {self.name}")
    def get_age(self):
        return self.age

# Créer des objets
alice = Person("Alice", 25)
bob = Person("Bob", 30)

# Utiliser les méthodes
alice.greet()  # Bonjour, je suis Alice
print(alice.get_age())  # 25
```

Copier

Héritage

```
# Classe parent
class Animal:
    def __init__(self, name):
        self.name = name
    def speak(self):
        pass

# Classes enfants
class Dog(Animal):
    def speak(self):
        return f"{self.name} dit Woof!"

class Cat(Animal):
    def speak(self):
        return f"{self.name} dit Meow!"

# Utilisation
dog = Dog("Rex")
cat = Cat("Minou")
print(dog.speak())  # Rex dit Woof!
print(cat.speak())  # Minou dit Meow!
```

Copier

Encapsulation

```
class BankAccount:
    def __init__(self, balance):
        self.__balance = balance  # Attribut privé (__)
    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount
        else:
            print("Solde insuffisant")
    def get_balance(self):
        return self.__balance

account = BankAccount(1000)
account.deposit(500)
print(account.get_balance())  # 1500
# print(account.__balance)    # Erreur : attribut privé
```

Copier

Gestion des exceptions

```
# Try / except
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Division par zéro impossible!")

# Plusieurs exceptions
try:
    number = int(input("Entrez un nombre: "))
    result = 10 / number
except ValueError:
    print("Ce n'est pas un nombre valide")
except ZeroDivisionError:
    print("Division par zéro impossible")

# Avec else et finally
try:
    file = open("data.txt", "r")
    content = file.read()
except FileNotFoundError:
    print("Fichier introuvable")
else:
    print("Fichier lu avec succès")
finally:
    file.close()  # Toujours exécuté
```

Copier

Modules et packages

Importer des modules

```
# Import complet
import math
print(math.pi)        # 3.141592653589793
print(math.sqrt(16))  # 4.0

# Import spécifique
from math import pi, sqrt
print(pi)      # 3.141592653589793
print(sqrt(16)) # 4.0

# Import avec alias
import numpy as np
import pandas as pd

# Import tout (déconseillé)
from math import *
```

Copier

Créer un module

Fichier `my_module.py` :

```
def greet(name):
    return f"Hello {name}!"

PI = 3.14159
```

Copier

Utilisation :

```
import my_module

print(my_module.greet("Alice"))
print(my_module.PI)
```

Copier

Modules standard utiles

```
# datetime : dates et heures
from datetime import datetime
now = datetime.now()
print(now.strftime("%Y-%m-%d %H:%M:%S"))

# random : nombres aléatoires
import random
print(random.randint(1, 10))
print(random.choice(['a', 'b', 'c']))

# os : système d'exploitation
import os
print(os.getcwd())  # Répertoire courant
os.listdir(".")     # Liste les fichiers

# json : manipulation JSON
import json
data = {"name": "Alice", "age": 25}
json_string = json.dumps(data)
print(json_string)  # {"name": "Alice", "age": 25}
```

Copier

Manipulation de fichiers

Lecture de fichiers

```
# Méthode 1 : lecture complète
with open("data.txt", "r") as file:
    content = file.read()
    print(content)

# Méthode 2 : lecture ligne par ligne
with open("data.txt", "r") as file:
    for line in file:
        print(line.strip())

# Méthode 3 : lecture de toutes les lignes
with open("data.txt", "r") as file:
    lines = file.readlines()
    print(lines)
```

Copier

Écriture dans des fichiers

```
# Écrire (écrase le contenu)
with open("output.txt", "w") as file:
    file.write("Hello World
")
    file.write("Deuxième ligne
")

# Ajouter à la fin
with open("output.txt", "a") as file:
    file.write("Ligne ajoutée
")
```

Copier

Compréhensions de listes

```
# Liste classique
numbers = [1, 2, 3, 4, 5]
squared = []
for num in numbers:
    squared.append(num ** 2)

# Compréhension de liste (plus pythonique)
squared = [num ** 2 for num in numbers]
print(squared)  # [1, 4, 9, 16, 25]

# Avec condition
even = [num for num in numbers if num % 2 == 0]
print(even)  # [2, 4]

# Compréhension de dictionnaire
squared_dict = {num: num**2 for num in numbers}
print(squared_dict)  # {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Copier

Bonnes pratiques

Convention de nommage (PEP 8)

```
# Variables et fonctions : snake_case
my_variable = 10
def calculate_total():
    pass

# Classes : PascalCase
class MyClass:
    pass

# Constantes : UPPER_CASE
MAX_SIZE = 100
API_KEY = "abc123"
```

Copier

Documentation

```
def calculate_area(width, height):
    """
    Calcule l'aire d'un rectangle.
    Args:
        width (float): Largeur du rectangle
        height (float): Hauteur du rectangle
    Returns:
        float: Aire du rectangle
    """
    return width * height
```

Copier

Gestion des dépendances

```
# Créer un environnement virtuel
python3 -m venv venv

# Activer l'environnement
# Linux/Mac :
source venv/bin/activate
# Windows :
venvScriptsactivate

# Installer des packages
pip install requests numpy pandas

# Sauvegarder les dépendances
pip freeze > requirements.txt

# Installer depuis requirements.txt
pip install -r requirements.txt
```

Copier

Ressources pour aller plus loin

Documentation officielle

- Python.org - Site officiel
- Documentation Python - Référence complète
- PEP 8 - Guide de style Python

Plateformes d'apprentissage

- Real Python - Tutoriels de qualité
- LeetCode - Exercices de programmation
- HackerRank - Défis Python

Bibliothèques populaires

- **NumPy** : Calcul scientifique
- **Pandas** : Analyse de données

- **Matplotlib** : Visualisation de données
- **Django / Flask** : Développement web
- **TensorFlow / PyTorch** : Machine Learning

Conclusion

Python est un langage puissant et accessible qui convient aussi bien aux débutants qu'aux développeurs expérimentés. Sa syntaxe claire, sa vaste bibliothèque standard et sa grande communauté en font un excellent choix pour de nombreux projets.

Prochaines étapes

1. **Pratiquer régulièrement** : Créez de petits projets
2. **Explorer les bibliothèques** : NumPy, Pandas, Django...
3. **Lire du code** : Étudiez des projets open source
4. **Contribuer** : Participez à la communauté

Points clés à retenir

- Python est **interprété**, pas besoin de compilation
- L'**indentation** est obligatoire et définit les blocs de code
- Tout est **objet** en Python
- Utilisez des **environnements virtuels** pour vos projets
- Suivez les conventions **PEP 8** pour un code propre

Bon apprentissage avec Python !