Python pour Développeurs Java — Cheatsheet

Adservio | Dr Olivier Vitrac

2026-02-03

Contents

1 Python pour Développeurs Java — Cheatsheet

1.1 Philosophie

Java	Python
Compilation explicite	Interprété, REPL
Types statiques stricts	Types dynamiques (+ hints optionnels)
Verbeux mais explicite	Concis, “batteries included”
`public static void main`	Script direct

1.2 Syntaxe de Base

1.2.1 Hello World

```java
// Java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

```python
# Python
print("Hello, World!")
```

1.2.2 Variables

```java
// Java
String name = "Alice";
int age = 30;
final double PI = 3.14159;
```

```python
# Python
name = "Alice"
age = 30
PI = 3.14159  # Convention MAJUSCULES, pas de const

# Type hints (optionnels, pour documentation)
name: str = "Alice"
age: int = 30
```

1.2.3 Conditions

```java
// Java
if (x > 0) {
    System.out.println("positif");
} else if (x < 0) {
    System.out.println("négatif");
} else {
    System.out.println("zéro");
}
```

```python
# Python — indentation obligatoire, pas de {}
if x > 0:
    print("positif")
elif x < 0:
    print("négatif")
else:
    print("zéro")
```

1.2.4 Boucles

```java
// Java - for classique
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}

// Java - for-each
for (String item : items) {
    System.out.println(item);
}
```

```python
# Python - range
for i in range(10):
    print(i)

# Python - itération directe
for item in items:
    print(item)

# Python - avec index
for i, item in enumerate(items):
    print(f"{i}: {item}")
```

1.3 Collections

1.3.1 Listes (≈ ArrayList)

```java
// Java
List<String> fruits = new ArrayList<>();
fruits.add("pomme");
fruits.add("banane");
String first = fruits.get(0);
```

```python
# Python
fruits = ["pomme", "banane"]
fruits.append("orange")
first = fruits[0]

# Slicing (puissant !)
last = fruits[-1]           # Dernier élément
subset = fruits[1:3]        # Index 1 et 2
reversed_list = fruits[::-1] # Inversion
```

1.3.2 Dictionnaires (≈ HashMap)

```java
// Java
Map<String, Integer> ages = new HashMap<>();
ages.put("Alice", 30);
ages.put("Bob", 25);
int age = ages.get("Alice");
```

```python
# Python
ages = {"Alice": 30, "Bob": 25}
ages["Charlie"] = 35
age = ages["Alice"]
age = ages.get("Alice", 0)  # Valeur par défaut si absent
```

1.3.3 Sets (≈ HashSet)

```java
// Java
Set<String> unique = new HashSet<>();
unique.add("a");
unique.add("b");
```

```python
# Python
unique = {"a", "b"}
unique.add("c")
```

1.4 Fonctions

1.4.1 Définition

```java
// Java
public static int add(int a, int b) {
    return a + b;
}
```

```python
# Python
def add(a, b):
    return a + b

# Avec type hints
def add(a: int, b: int) -> int:
    return a + b
```

1.4.2 Arguments par défaut

```java
// Java — surcharge nécessaire
public static void greet(String name) {
    greet(name, "Hello");
}
public static void greet(String name, String greeting) {
    System.out.println(greeting + ", " + name);
}
```

```python
# Python — valeur par défaut
def greet(name, greeting="Hello"):
    print(f"{greeting}, {name}")
```

```
greet("Alice")            # Hello, Alice
greet("Bob", "Bonjour")   # Bonjour, Bob
```

1.4.3 Lambda

```
// Java
Function<Integer, Integer> square = x -> x * x;
```

```
# Python
square = lambda x: x * x
```

1.5 Classes

1.5.1 Définition

```
// Java
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    @Override
    public String toString() {
        return "Person(" + name + ", " + age + ")";
    }
}
```

```
# Python
class Person:
    def __init__(self, name: str, age: int):
        self.name = name
        self.age = age

    def __str__(self):
        return f"Person({self.name}, {self.age})"

# Utilisation
p = Person("Alice", 30)
print(p.name)  # Pas de getter explicite
```

1.5.2 Dataclass (≈ Record Java)

```
from dataclasses import dataclass
```

```python
@dataclass
class Person:
    name: str
    age: int

# Génère automatiquement __init__, __repr__, __eq__
p = Person("Alice", 30)
```

1.5.3 Héritage

```java
// Java
public class Student extends Person {
    private String school;

    public Student(String name, int age, String school) {
        super(name, age);
        this.school = school;
    }
}
```

```python
# Python
class Student(Person):
    def __init__(self, name: str, age: int, school: str):
        super().__init__(name, age)
        self.school = school
```

1.6 Gestion d'Erreurs

```java
// Java
try {
    int result = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Division par zéro");
} finally {
    System.out.println("Cleanup");
}
```

```python
# Python
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Division par zéro")
except Exception as e:
    print(f"Erreur: {e}")
finally:
    print("Cleanup")
```

1.7 Fichiers

```java
// Java
try (BufferedReader reader = new BufferedReader(new FileReader("file.txt"))) {
```

```java
    String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
}
```

```python
# Python — context manager (équivalent try-with-resources)
with open("file.txt", "r") as f:
    for line in f:
        print(line.strip())

# Tout lire d'un coup
content = open("file.txt").read()

# Écriture
with open("output.txt", "w") as f:
    f.write("Hello\n")
```

1.8 JSON

```java
// Java (avec Jackson)
ObjectMapper mapper = new ObjectMapper();
MyClass obj = mapper.readValue(jsonString, MyClass.class);
String json = mapper.writeValueAsString(obj);
```

```python
# Python — intégré
import json

# Parsing
data = json.loads('{"name": "Alice", "age": 30}')
print(data["name"])

# Sérialisation
json_str = json.dumps({"name": "Bob"}, indent=2)

# Fichier
with open("data.json") as f:
    data = json.load(f)

with open("output.json", "w") as f:
    json.dump(data, f, indent=2)
```

1.9 Concurrence

```java
// Java — CompletableFuture
CompletableFuture.supplyAsync(() -> fetchData())
    .thenApply(data -> process(data))
    .join();
```

```python
# Python — asyncio
import asyncio
```

```python
async def main():
    data = await fetch_data()
    result = await process(data)
    return result

asyncio.run(main())
```

1.10 Tests

```java
// Java – JUnit
@Test
public void testAdd() {
    assertEquals(4, Calculator.add(2, 2));
}
```

```python
# Python – pytest
def test_add():
    assert add(2, 2) == 4

# Exécution
# pytest test_file.py -v
```

1.11 Idiomes Python Courants

1.11.1 List Comprehension

```python
# Équivalent de stream().map().collect()
squares = [x**2 for x in range(10)]

# Avec filtre
evens = [x for x in range(10) if x % 2 == 0]

# Dict comprehension
squares_dict = {x: x**2 for x in range(5)}
```

1.11.2 Unpacking

```python
# Déstructuration
a, b, c = [1, 2, 3]
first, *rest = [1, 2, 3, 4]  # first=1, rest=[2,3,4]

# Swap
a, b = b, a
```

1.11.3 F-strings

```python
name = "Alice"
age = 30
print(f"Je suis {name}, j'ai {age} ans")
print(f"Calcul: {2 + 2 = }")  # Affiche "Calcul: 2 + 2 = 4"
```

1.11.4 Walrus Operator (:=)

```python
# Assignation dans une expression
if (n := len(items)) > 10:
    print(f"Trop d'items: {n}")
```

1.12 Pièges pour Javistes

1.12.1 1. Indentation significative

```python
# ❌ Erreur de syntaxe
if True:
print("oui")

# ✅ Correct
if True:
    print("oui")
```

1.12.2 2. Mutable par défaut dans les arguments

```python
# ❌ Bug classique
def append_to(item, lst=[]):
    lst.append(item)
    return lst

append_to(1)  # [1]
append_to(2)  # [1, 2] ← Surprise !

# ✅ Correct
def append_to(item, lst=None):
    if lst is None:
        lst = []
    lst.append(item)
    return lst
```

1.12.3 3. == vs is

```python
a = [1, 2, 3]
b = [1, 2, 3]

a == b  # True (valeurs égales)
a is b  # False (objets différents)

# Pour None, toujours utiliser is
if x is None:
    pass
```

1.12.4 4. Pas de switch (avant 3.10)

```python
# Python 3.10+ : match/case
match status:
    case 200:
```

```
        print("OK")
    case 404:
        print("Not Found")
    case _:
        print("Unknown")

# Avant 3.10 : dict
actions = {
    200: lambda: print("OK"),
    404: lambda: print("Not Found"),
}
actions.get(status, lambda: print("Unknown"))()
```

1.13 Ressources

- Python Official Tutorial
- Real Python
- Python for Java Developers

Cheatsheet de référence — À garder sous la main pendant le workshop.