

Développer avec Robot Framework

22-23 Septembre 2025

Nickel NANTES

RÉMI PICARD

Tour de table

- Expérience dev
- Expérience testeur
- Expérience avec Robot Framework
- Attente vis à vis de cette formation ?

Qui suis-je ?

- Rémi PICARD
- Dev Scala Cobalt
- 4 ans chez Nickel
- 4 ans d'expérience avec Robot Framework 🤖
- Passionné par les technos Web, Data et DevOps
- 13 ans d'expérience dans l'IT 💡
- Joueur d'échecs ♟



Jour 1

Découverte de Robot Framework

Découverte de Robot Framework

- Généralités
- Installer l'env de dev
- Rappels Python
- Comprendre le fonctionnement de Robot Framework
- Découvrir la ligne de commande robot
- Ecrire mes premiers tests (codelab)

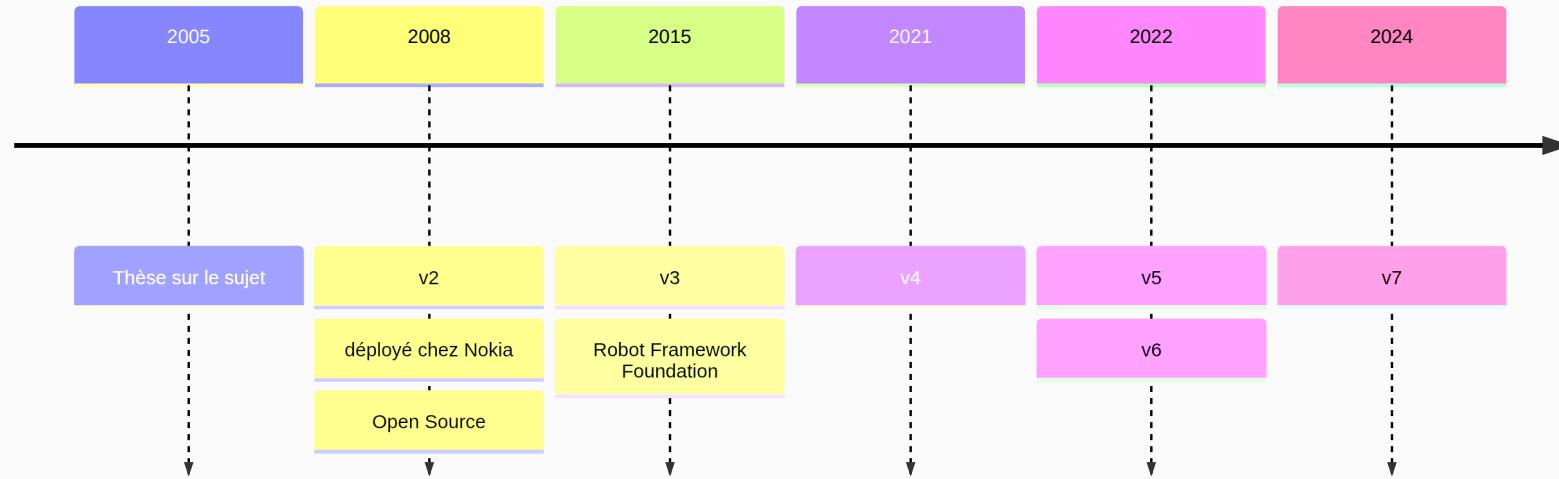
Présentation

- Outil d'automatisation
- Langage
- Open Source codé en Python
- Fonctionnalités clefs en main (assertions, rapport de tests...)
- Extensible via des librairies RobotFramework (HTTP, JSON, SQL, Kafka ...)
- Extensible via des librairies Python

Robot Framework => "Robot"
RF / RBF / RBT

Histoire

20 ans déjà !

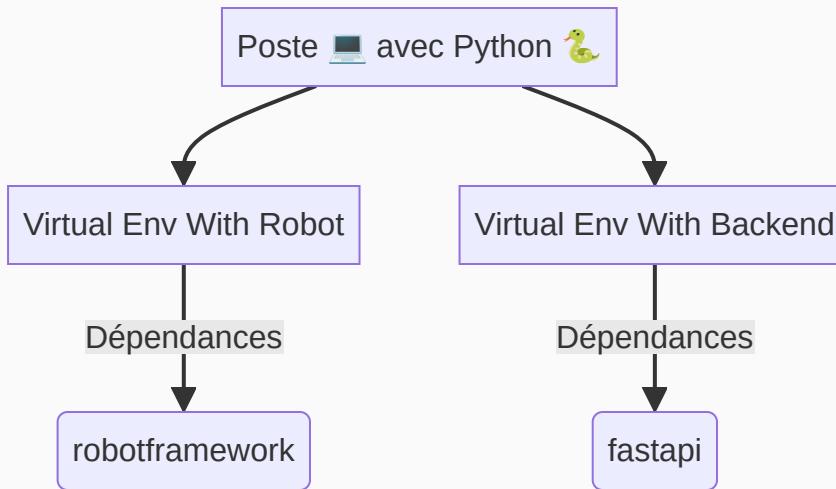


Communauté

- Slack
- RoboCon / RBCN : conférence annuelle à Helsinki 
- [Documentation](#)

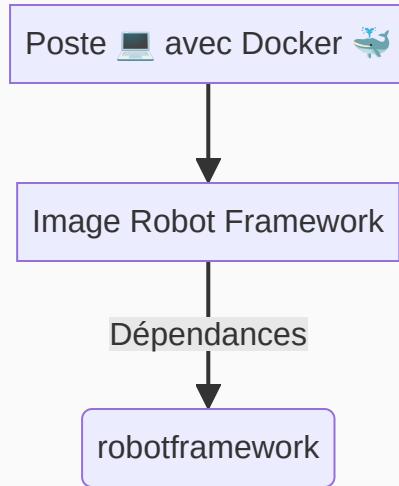
Env de dev 

Installation Python Virtual Environment



Python Virtual Environment

Installation Docker



Installation PyCharm / VSCode

Set up your IDE

Codelab



- [Installation env de dév](#)
- [Hello World](#)
- [Configuration IDE](#)

Développer avec Python 🐍

Rappels

Types natifs

Type	Mot-clé	Exemple
Entier	int	42, -17, 0
Flottant	float	3.14, 2.5e-3
Booléen	bool	True, False
Chaîne	str	"Hello", 'World'
Tableau	list	["Hello", "World"]
Map	dict	{"key": "value", "key2": "value2"}
Aucun	NoneType	None

[Documentation](#)

Méthodes

```
1 def ma_fonction(arg1: int, arg2: str, default_arg="default", *args, **kwargs) -> int:
2     # ...
3     return 42
4
5 ma_fonction(42, "Quarante-deux")
6 ma_fonction(42, "Quarante-deux", "override default value")
7 ma_fonction(42, "Quarante-deux", "default", 1, 2, 3)
8 ma_fonction(42, "Quarante-deux", "default", robot="Nono", john="Doe")
9 ma_fonction(42, "Quarante-deux", "default", 1, 2, 3, robot="Nono", john="Doe")
```

Développer avec Robot Framework

Tests, Variables, Keyword, Python ...

Tests

- fichier `*.robot`
- indentation comme en Python 
- ensemble de phrases 
- Section `*** Test Cases ***`
- Import librairies dans `*** Settings ***`

Tests

Déclaration

```
1 *** Settings ***
2 Library      String
3
4 *** Test Cases ***
5 Mon Premier Test
6     ${chaine}=    Generate Random String    10
7     Log      Hello ${chaine}
```

Variables

- Même types qu'en Python 
- Syntaxe `${...}` (comme en Bash)
- Création dans un Test ou Keyword
- Création en global dans la section `*** Variables ***`
- Import des variables Python possible

Variables

Déclaration

```
1 *** Test Cases ***
2 Creation Variable
3     ${ma_variable}    Set Variable    42
4     Log    ma_variable=${ma_variable}
```

Variables

Import

```
1 # resources/mes_variables_python.py  
2  
3 variable_python = 42
```

```
1 *** Settings ***  
2 Variables      resources/mes_variables_python.py  
3  
4 *** Test Cases ***  
5 Creer Variable  
6     Log      variable_python=${variable_python}
```

Variables

Section Variables

```
1  *** Variables ***
2  ${nombre}    42
3  ${chaine}    Ma chaîne de caractères
4  @{tab}        1  2  3
5  &{map}        clef1=valeur1  clef2=valeur2
6
7  *** Test Cases ***
8  Teste Variables
9      Log    nombre=${nombre}
10     Log    chaine=${chaine}
11     Log    tab=${tab}
12     Log    map=${map}
```

Injection Python

Evaluate

```
1  *** Test Cases ***
2  Teste Evaluate
3      ${nb}=    Evaluate    41 + 1
4      Log      nb=${nb}
5
6  Teste Evaluate Autre Syntaxe
7      ${nb}=    Set Variable   ${${41 + 1}}
8      Log      nb=${nb}
```

Keyword Concept

- Ensemble de mots clés (séparés par 1 espace)
- Forme une phrase 
- Représente une **action** 
- Déclaré dans la section `*** Keywords ***`
- Robot Framework traduit les phrases en appels Python 

Keyword

Syntaxe Robot Framework

```
1 *** Keywords ***
2 Mon Premier Keyword
3     Log    Hello World
4
5 Mon Premier Keyword Avec Argument
6     [Arguments]    ${name}
7     Log    Hello ${name}
8
9 Mon Premier Keyword Avec Argument Et Return
10    [Arguments]   ${name}
11    Log    Hello ${name}
12    RETURN    42
```

Keyword

Arg dans Keyword

```
1  *** Keywords ***
2  Keyword Avec ${arg1} Intégré
3      Log    Hello ${arg1}
4
5  Keyword Avec ${arg1} Intégré Et Arguments
6      [Arguments]    ${name}
7      Log    Hello ${arg1}, ${name}
8
9  *** Test Cases ***
10 Appel Keywords
11     ${arg1}    Set Variable    Ma Variable
12     Keyword Avec ${arg1} Intégré
13     Keyword Avec ${arg1} Intégré Et Arguments    Arg2
```

Keyword

List (args) / Dict (kwargs)

```
1  *** Keywords ***
2  Keyword Avec Args
3      [Arguments]    @{list}
4      FOR ${i}  IN  @{list}
5          Log    i=${i}
6      END
7
8  Keyword Avec Kwargs
9      [Arguments]    &{map}
10     FOR ${k}  ${v}  IN  &{map}
11         Log    key=${k}, value=${v}
12     END
13
14 *** Test Cases ***
15 Appel Keywords
16     ${list}  Create List  1  2  3
17     Keyword Avec Args    ${list}
18
19     ${map}  Create Dictionary  cle1=valeur1  cle2=valeur2
20     Keyword Avec Kwargs    &{map}
```

Keyword

Gestion des espaces

- **1 espace** entre chaque mot
- **2 espaces ou +** (ou **tabulation**) entre chaque argument

Variable de retour

Keyword

Arguments

`${retour}= Mon Premier Keyword Terminator 2ème arg`

The diagram illustrates the structure of a keyword definition. It shows the variable of return (`${retour}`), followed by an equals sign (`=`). Then, the keyword name (`Mon Premier Keyword`) is enclosed in a green dashed box, with two green arrows pointing to the spaces before and after the keyword name. After the keyword name, there is a space followed by the terminator (`Terminator`), which is enclosed in a purple dashed box, with a purple arrow pointing to the space before the terminator. Finally, there is another space followed by the argument (`2ème arg`), which is enclosed in a grey dashed box, with a green arrow pointing to the space before the argument.

Keyword

Gestion des chaînes

- inutile de mettre des " ou des ' autour des chaînes de caractère

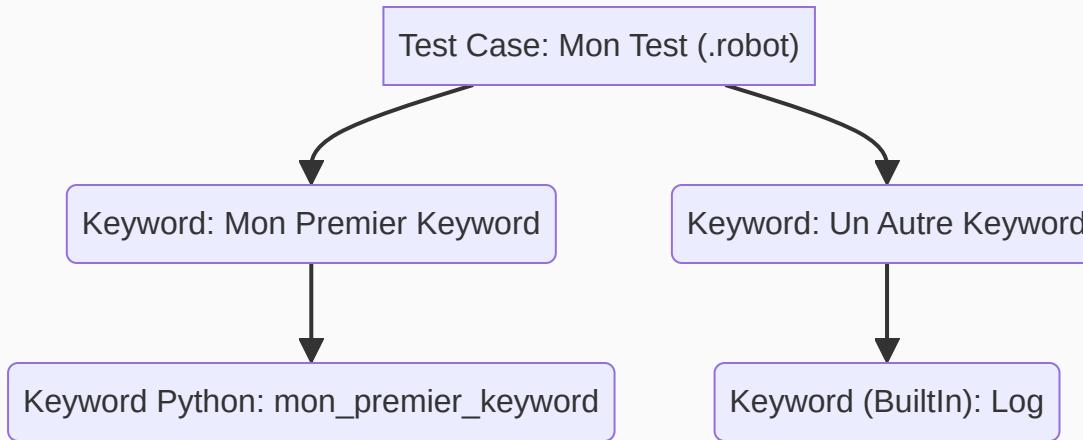
Keyword Appel

```
1 *** Test Cases ***
2 Mon Premier Test
3     Mon Premier Keyword
4         Mon Premier Keyword Avec Argument      Nono le petit robot
5             ${retour}=    Mon Premier Keyword Avec Argument Et Return    Terminator
6
7 *** Keywords ***
8 Mon Deuxième Keyword
9     Mon Premier Keyword
```

Keyword Syntaxe Python

```
1 # Mon Premier Keyword
2 def mon_premier_keyword():
3     print("Hello World")
4
5 # Mon Premier Keyword Avec Argument Et Return    ${name}
6 def mon_premier_keyword_avec_argument_et_return(name) -> int:
7     print(name)
8     return 42
```

Test -> Keyword -> Keyword



Section

- `*** Settings ***` : imports librairies / ressources / variables
- `*** Test Cases ***` : déclarations Tests
- `*** Keywords ***` : déclarations Keywords
- `*** Variables ***` : déclarations Variables

Ligne de commande robot

```
1  robot --help  
2  
3  # Lance tous les tests présents dans le dossier tests  
4  robot tests  
5  
6  # Lance le test "Mon Test"  
7  robot -t "Mon Test" tests
```

Résultats

- `report.html` : synthèse
- `log.html` : détails par tests et keywords
 -  ERROR => automatiquement affiché (avec focus sur le keyword en erreur)
 -  SUCCESS
- `output.xml` / `xunit.xml` : sortie technique pour intégration continue / outils

Résultats

log.html

TODO Capture d'écran

Quizz

TODO Klaxoon ?



Tester les syntaxes

- Keyword
- Test
- Variable

RequestsLibrary

- Tests d'API
- Wrapper de la lib Python requests
- [Documentation](#)

RequestsLibrary

```
1 *** Settings ***
2 Library           RequestsLibrary
3
4 *** Test Cases ***
5 Quick Get Request Test
6     ${response}=    GET  https://www.google.com
7
8 Quick Get Request With Parameters Test
9     ${response}=    GET  https://www.google.com/search  params=query=ciao  expected_status=200
10
11 Quick Get A JSON Body Test
12     ${response}=    GET  https://jsonplaceholder.typicode.com/posts/1
13     Should Be Equal As Strings  1  ${response.json()}[id]
```



API Booker

Jour 2

Bonnes pratiques et Industrialisation

Bonnes pratiques Robot Framework

- Organiser les tests et les ressources
- Utiliser les librairies standards de Robot Framework
- Découvrir les syntaxes avancées
- Ecrire des tests robustes
- Découvrir les outils autour de Robot Framework
- Mettre en place une intégration continue

Organiser les tests et les ressources

Structure standard d'un projet

- `tests/` : fichiers de tests (`.robot`).
- `resources/` : keywords partagés, variables et librairies maison (`.resource`, `.py`).

```
1 mon-projet/
2   └── tests/
3     └── cas_de_test.robot
4   └── resources/
5     └── keyword_comuns.resource
6     └── ma_librarie.py
```

RobotFramework - Project Structure

Ligne de commande robot

```
1 # Vérifie la syntaxe de tous les tests
2 robot --dryrun tests
3
4 # Ajoute le répertoire courant dans le PYTHON PATH
5 robot --pythonpath . tests
```

PYTHON PATH

Sans --pythonpath :

```
1 *** Settings ***
2 Resource    ./resources/ma_lib.resource
3
4 *** Test Cases ***
5 Tester Ma Lib
6     Mon Keyword
```

Avec --pythonpath . :

```
1 *** Settings ***
2 Resource    resources/ma_lib.resource
3
4 *** Test Cases ***
5 Tester Ma Lib
6     Mon Keyword
```

Librairies standards

Boîte à outils intégrée

- `BuiltIn` : incontournables (`Log`, `Set Variable`, `Run Keyword If`, `Sleep ...`)
- `String` : chaînes de caractères
- `Collections` : listes et dictionnaires

Écrire des tests robustes

Gérer l'asynchronisme

- **Problème** : Les éléments d'une page web n'apparaissent pas instantanément.
- **Anti-pattern** : Sleep -> attente fixe, fragile et qui ralentit les tests.
- **Solution** : Wait Until Keyword Succeeds -> boucle d'attente intelligente.

```
1 *** Test Cases ***
2 Attendre un élément
3    Wait Until Keyword Succeeds    10s    1s    Page Should Contain    Bienvenue
```

Outils autour de Robot Framework

- RequestsLibrary : Tests d'API REST.
- JSON

Intégration continue

- **Principe** : Exécuter les tests automatiquement à chaque `push` sur le dépôt Git
- **Exemple avec Gitlab CI** : Un fichier `.gitlab-ci.yml` qui lance la commande `robot`
- **Artefacts** : Publier les rapports HTML pour chaque pipeline

Questions & Réponses

- Merci pour votre attention.
- Bonne chance pour l'automatisation de vos tests !
- Amusez-vous bien !