

Tests automatisés avec Playwright, Appium et Pytest

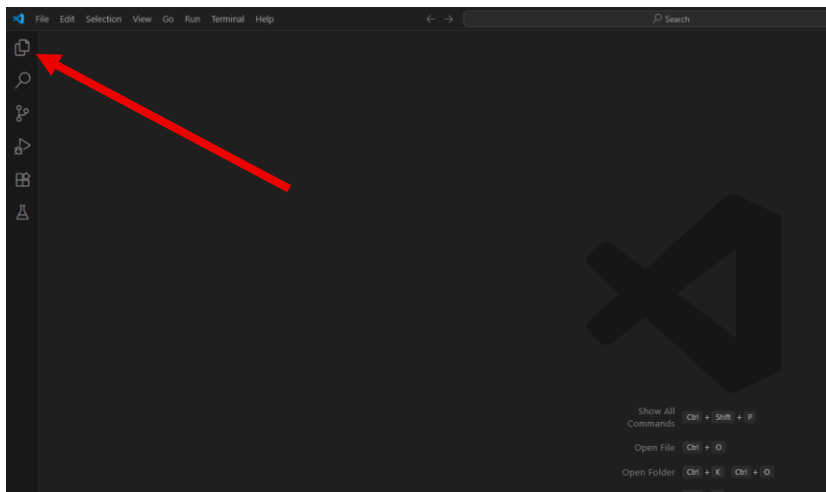
Pytest est une bibliothèque Python permettant de créer des tests automatisés pour tout genre d'application (sites web ou applications mobiles par exemple).

Pour ce tutoriel il est recommandé d'avoir une petite base de connaissance de la programmation avec python mais le code sera expliqué tout au long en considérant que vous débutez.

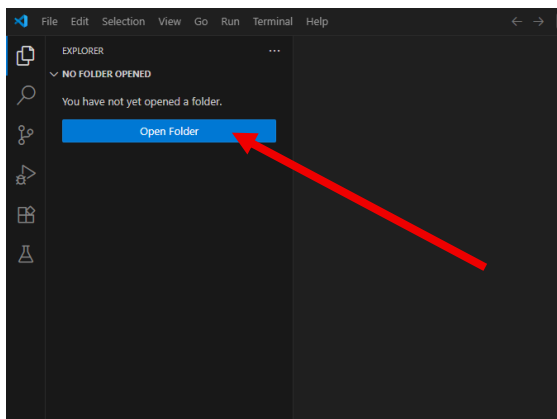
Installation de Pytest :

Ce tutoriel a été réalisé avec l'éditeur de code Visual Studio Code, c'est un éditeur gratuit très répandu. Téléchargement sur <https://code.visualstudio.com/download>

Une fois Visual Studio Code installé ouvrez le et créez un nouveau dossier :



Cliquez sur l'icône explorer et ouvrez un dossier



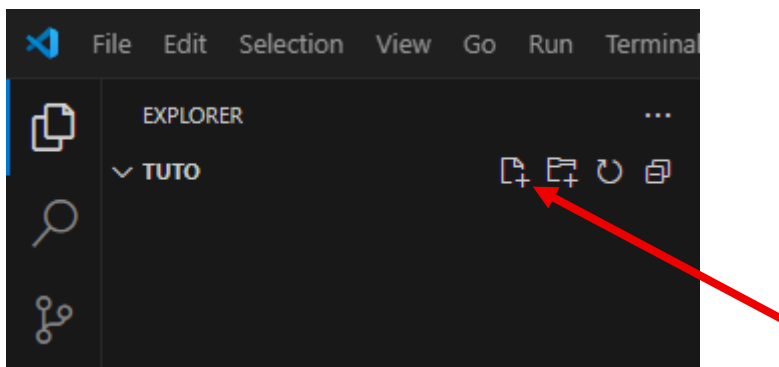
Tout au long nous allons coder avec le langage de programmation Python, c'est un des langages informatiques le plus utiliser car il facile à apprendre pour un débutant et l'on peut l'utiliser pour divers choses (applications de bureau, Data Science, IA, jeux vidéo...)

Le téléchargement est possible via le Microsoft Store :

<https://apps.microsoft.com/detail/9PNRBTZXMB4Z?hl=en-us&gl=FR&ocid=pdpshare>

Ou sur le site web officiel : <https://www.python.org/downloads/>

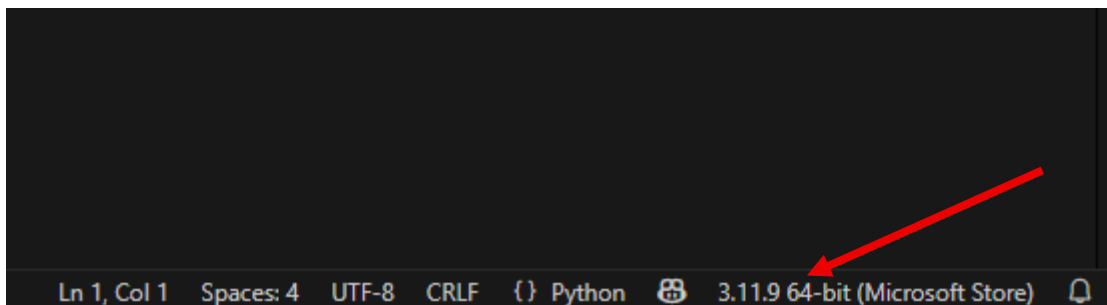
Créez un nouveau fichier python en cliquant sur cette icône :



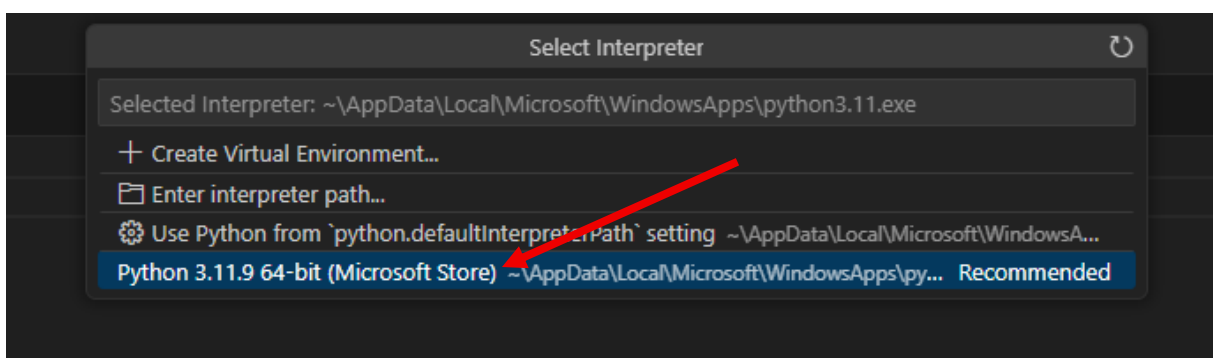
Appelrez ce fichier « test_playwright.py ».

Il faut maintenant configurer l'éditeur pour qu'il exécute python :

En bas à droite de la fenêtre cliquez sur le menu suivant :



Puis en haut, sélectionnez la version de python que vous aviez installé :

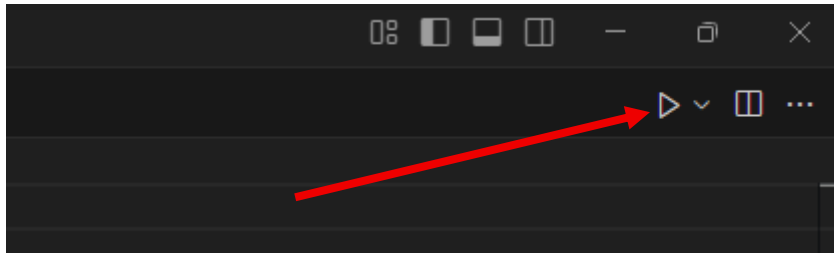


Afin de tester si python fonctionne bien, codez cette ligne, la fonction print sert à afficher du texte, le texte que l'on veut écrire est mis entre les parenthèses :

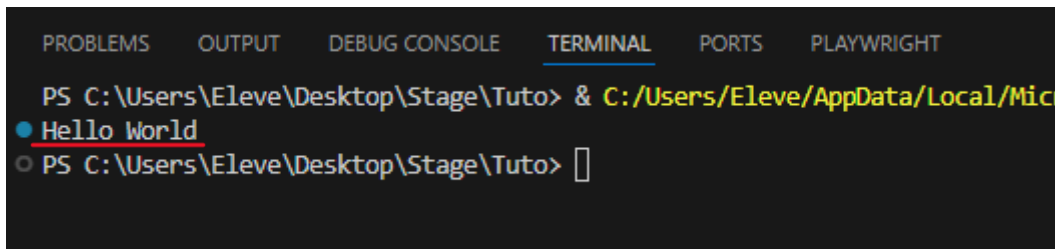
Attention, en programmation, le texte (chaîne de caractère ou "string") est toujours entre guillemets ' ou double guillemets ''.

```
print("Hello World")
```

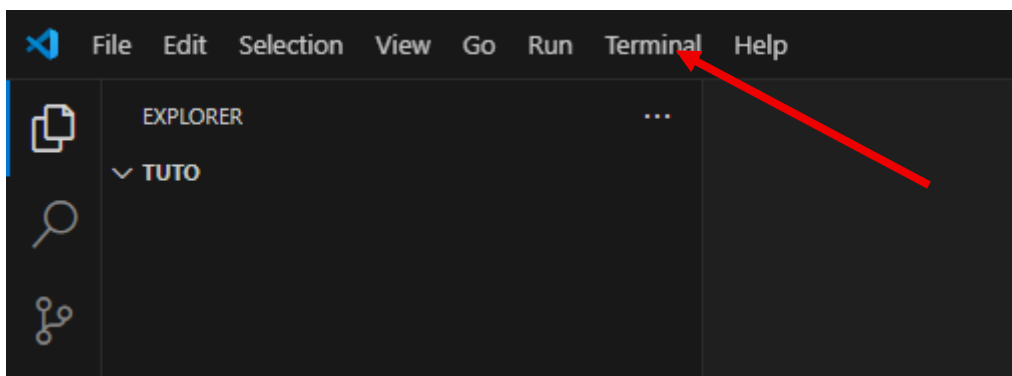
Maintenant le programme via cette icône en haut à droite :

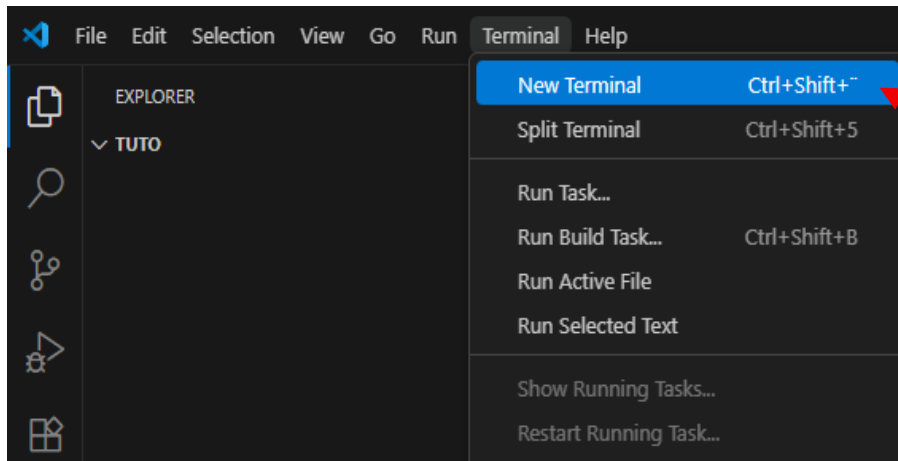


Le texte s'affiche correctement dans le terminal :

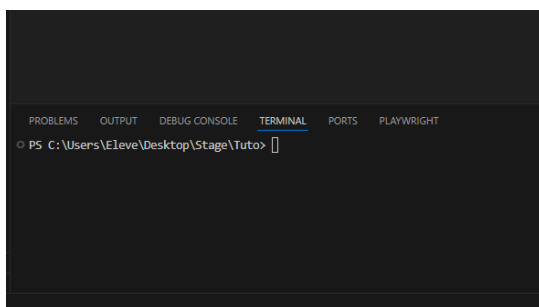


Avant d'aller plus loin, il va falloir créer ce que l'on appelle un environnement, ouvrez un nouveau terminal via le menu Terminal ou le raccourci Ctrl + Shift + `





Le terminal s'ouvre dans cette fenêtre :



Entrez la commande :

python -m venv venv

Cette commande signifie que l'on veut créer un environnement virtuel (venv) et qu'on lui donne venv comme nom.

Pour utiliser la venv il suffit d'entrer cette commande :

venv\Scripts\Activate

Pour l'ensemble de nos tests on va utiliser la bibliothèque Pytest, installable via cette commande :

pip install pytest

Et voilà, Pytest est installé, dans la prochaine partie nous allons faire nos premiers tests web.

Tests web automatisés :

Installation de Playwright :

Avant de pouvoir utiliser Playwright, il faut l'installer et télécharger les navigateurs nécessaires. Utilisez les commandes :

```
pip install playwright
```

```
playwright install
```

```
pip install pytest-playwright
```

Playwright permet d'automatiser des sites web comme un vrai utilisateur : cliquer sur des boutons, remplir des champs, charger des images, etc...

Premier test :

Notre premier test se déroulera comme suivant :

- 1) Aller sur le site de Wikipédia.
- 2) Rechercher un mot (pour ce tuto on va rechercher "Python").
- 3) Vérifier que la page contient le bon titre.

Dans votre fichier `_test.py` :

Entrez ou copiez le code suivant :

```
from playwright.sync_api import Page, expect
```

```
def test_recherche_wikipedia(page: Page):
```

```
    page.goto("https://fr.wikipedia.org")
```

```
    expect(page.get_by_role("searchbox", name="Rechercher sur Wikipédia")).to_be_visible()
```

```
    expect(page.get_by_role("button", name="Rechercher")).to_be_visible()
```

```
    page.get_by_role("searchbox", name="Rechercher sur Wikipédia").click()
```

```
    page.get_by_role("searchbox", name="Rechercher sur Wikipédia").fill("Python")
```

```
    page.get_by_role("button", name="Rechercher").click()
```

```
    expect(page.locator("#firstHeading")).to_contain_text("Python")
```

Explication du code :

```
from playwright.sync_api import Page, expect
```

Cette ligne permet d'importer (**import**) les commandes **Page** et **expect** depuis (**from**) la bibliothèque playwright (**playwright.sync_api**).

```
def test_recherche_wikipedia(page: Page):
```

Cette ligne définit (**def**) une fonction de test nommée **test_recherche_wikipedia**, qui prend en paramètre **page** (de type **Page**), c'est-à-dire une instance de navigateur ouverte automatiquement par Playwright pour faire le test.

```
page.goto("https://fr.wikipedia.org")
```

Cette ligne permet d'ouvrir la page (**goto**) du site Wikipédia en français dans le navigateur.

```
expect(page.get_by_role("searchbox", name="Rechercher sur Wikipédia")).to_be_visible()
```

Cette ligne attend (**expect**) que le champ de recherche (repéré par rôle (**.get_by_role**), du nom **name="Rechercher sur Wikipédia"**) soit visible sur la page (**.to_be_visible()**) avant de continuer. S'il ne trouve pas l'élément, une erreur se produira, faisant échouer le test.

```
expect(page.get_by_role("button", name="Rechercher")).to_be_visible()
```

Cette ligne attend (**expect**) que le bouton (repéré par rôle (**.get_by_role**), du nom **name="Rechercher"**) soit visible sur la page (**.to_be_visible()**) avant de continuer. S'il ne trouve pas l'élément, une erreur se produira, faisant échouer le test.

```
page.get_by_role("searchbox", name="Rechercher sur Wikipédia").click()
```

Ici, on clique (**.click()**) dans la barre de recherche. Cela place le curseur dans le champ, comme si on cliquait avec la souris.

```
page.get_by_role("searchbox", name="Rechercher sur Wikipédia").fill("Python")
```

Cette ligne remplit (**fill**) la barre de recherche avec le mot Python.

```
page.get_by_role("button", name="Rechercher").click()
```

Cette ligne simule un clic sur le bouton Rechercher, ce qui lance la recherche de l'article Python.

```
expect(page.locator("#firstHeading")).to_contain_text("Python")
```

Cette ligne vérifie (**expect**) que le titre principal de la page (repéré par l'id (=identifiant) **#firstHeading**) contient bien le texte (**.to_contain_text**) "Python". Cela permet de confirmer que Wikipédia a bien affiché la bonne page.

Attention, l'indentation en Python est très importante, elle sert à indiquer dans quel "morceau" du code on se situe, l'indentation est souvent faite avec une tabulation (touche Tab du clavier) mais des espaces fonctionnent aussi, la règle est que tout le code indenté ensemble doit être indenté au même niveau.

Une fois le "morceau" de code terminé, retirez l'indentation.



```
1 from playwright.sync_api import Page, expect
2
3 def test_recherche_wikipedia(page: Page):
4     page.goto("https://fr.wikipedia.org")
5
6     expect(page.get_by_role("searchbox", name="Rechercher sur Wikipédia")).to_be_visible()
7     expect(page.get_by_role("button", name="Rechercher")).to_be_visible()
8
9     page.get_by_role("searchbox", name="Rechercher sur Wikipédia").click()
10    page.get_by_role("searchbox", name="Rechercher sur Wikipédia").fill("Python")
11
12    page.get_by_role("button", name="Rechercher").click()
13
14    expect(page.locator("#firstHeading")).to_contain_text("Python")
```

Exemple : tout le code situé dans la fonction a une tabulation, tout le code est aligné au même niveau

Dans visual studio code, l'indentation est automatique mais il faut quand même - faire attention, souvenez-vous qu'après une ligne se terminant par un double point ":" il y a toujours une indentation après.

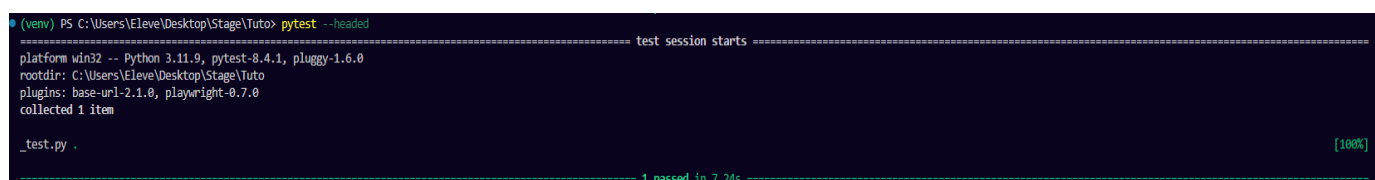
Pour lancer le test, utilisez la commande :

pytest

Et pour lancer le test avec l'interface :

pytest --headed

Si le texte a fonctionné, vous devriez voir dans la console quelque chose comme ceci :



```
(venv) PS C:\Users\Eleve\Desktop\Stage\Tuto> pytest --headed
===== test session starts =====
platform win32 -- Python 3.11.9, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\Eleve\Desktop\Stage\Tuto
plugins: base-url-2.1.0, playwright-0.7.0
collected 1 item

_test.py .

===== 1 passed in 7.24s =====
```

Il existe bien d'autres commandes pour créer des tests et il n'est aussi pas super simple de rechercher chaque élément de la page pour écrire le code, surtout si le test comprend beaucoup d'éléments.

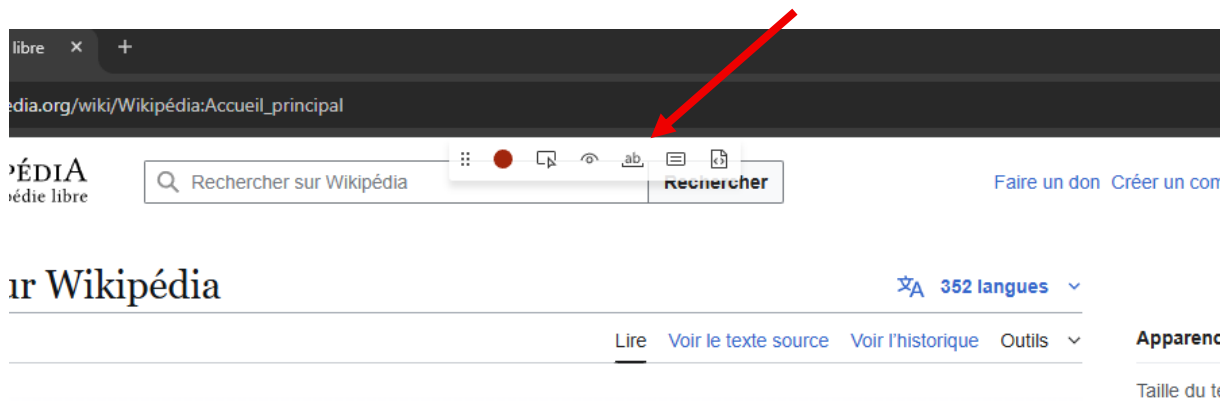
Une façon plus simple pour réaliser le code des tests est d'utiliser le **recorder** intégré à Playwright.






Pour l'utiliser, tapez cette commande dans le terminal :

`playwright codegen "URL_du_site"`

Exemple : **`playwright codegen https://fr.wikipedia.org`**

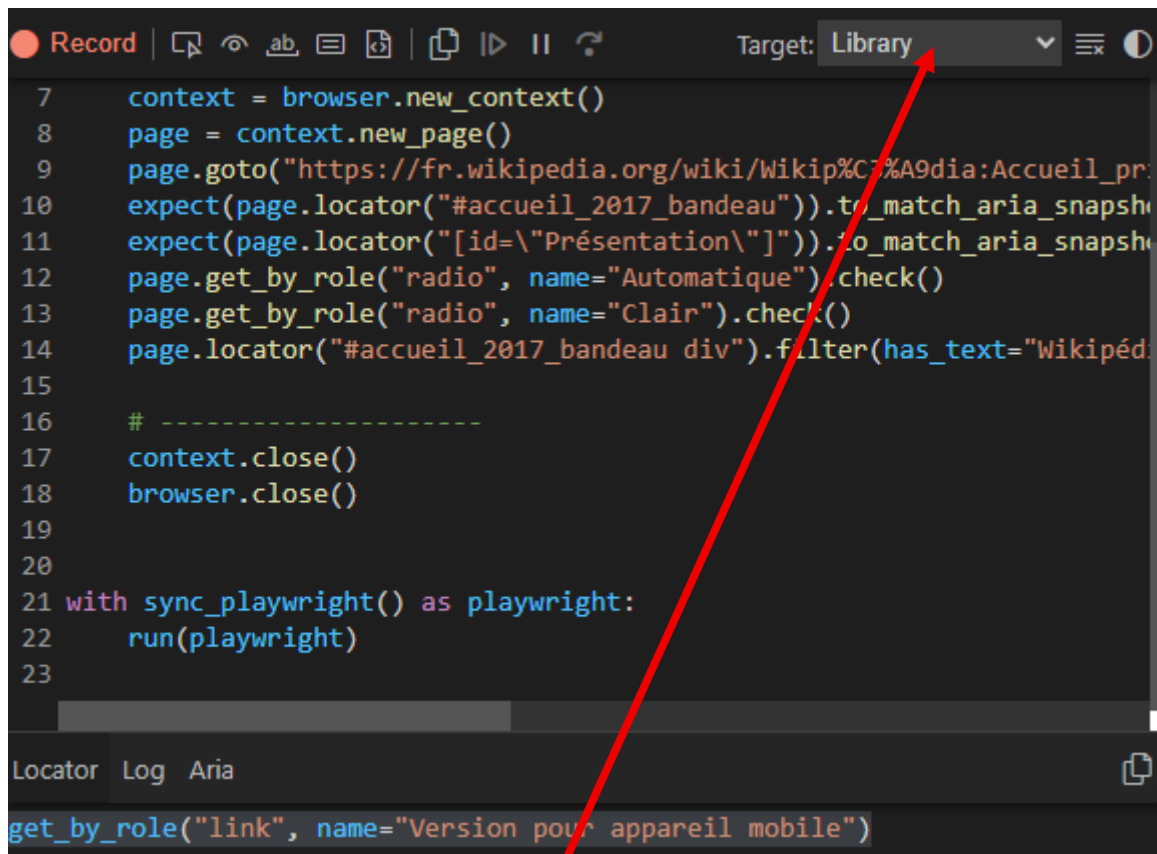
Deux fenêtres s'ouvrent, la première est le navigateur avec le site indiqué dans la commande. En haut de cette fenêtre se trouve une barre d'outils.



-  Permet d'activer ou de désactiver l'enregistrement.
-  Permet d'obtenir ce qui permet d'identifier un élément.
-  Permet de vérifier la visibilité d'un élément
-  Permet de vérifier la présence d'un texte précis dans un élément.
-  Permet de vérifier la présence d'une valeur précise dans un élément.

Lorsque vous cliquez sur des éléments ou que vous rentrez des valeurs, tout est enregistré automatiquement.

La deuxième fenêtre est le générateur de code, toutes les actions que vous faites dans le navigateur y est transformé en code, dans de nombreux langages.



```
7 context = browser.new_context()
8 page = context.new_page()
9 page.goto("https://fr.wikipedia.org/wiki/Wikip%C3%A9dia:Accueil_principal")
10 expect(page.locator("#accueil_2017_bandeau")).to_match_aria_snapshot()
11 expect(page.locator("[id='Présentation']")).to_match_aria_snapshot()
12 page.get_by_role("radio", name="Automatique").check()
13 page.get_by_role("radio", name="Clair").check()
14 page.locator("#accueil_2017_bandeau div").filter(has_text="Wikipédia")
15
16 # -----
17 context.close()
18 browser.close()
19
20
21 with sync_playwright() as playwright:
22     run(playwright)
23
```

Locator Log Aria

get_by_role("link", name="Version pour appareil mobile")

Sélectionner Pytest dans le menu indiqué pour obtenir le code fonctionnant avec ce que nous avons mis en place.

Il suffit maintenant de copier le code et lorsque vous l'exécuterez, toutes les actions que vous aviez faites auparavant sur la page sera reproduit.

Pour plus d'infos sur Playwright et Pytest :

Documentation Playwright : <https://playwright.dev/python/docs/input>

Documentation Pytest : <https://docs.pytest.org/en/stable/>

Tests mobiles automatisés :

Installation d'Appium :

Appium permet de contrôler un téléphone (Android ou IOS), combiné avec la bibliothèque de tests python Pytest que l'on a utilisé précédemment on peut faire des tests automatisés sur mobile.

Pour ce tutoriel, on va utiliser un téléphone physique utilisant Android.

Avant d'installer Appium, il faut avoir :

- Python (déjà installé plus tôt)
- Java JDK (installation sur <https://www.oracle.com/java/technologies/downloads/>)
- Node.js (installation sur <https://nodejs.org/fr/download>)

Maintenant installons Appium, ouvrez un nouveau dossier sur visual studio code et créez un fichier test_appium.py. Ensuite créez un environnement virtuel et tapez ou copiez cette commande dans votre terminal :

```
npm install -g appium
```

Puis vérifions l'installation avec :

```
appium -v
```

Et installons les drivers Appium :

```
appium driver install uiautomator2
```

Vérifiez avec :

```
appium driver list --installed
```

Installons aussi un plugin (= extension) qui nous permettra d'afficher l'écran sur une page web et d'inspecter les éléments afin de récupérer de quoi les identifier (un peu à la manière de Playwright inspector) :

```
appium plugin install --source=npm appium-inspector-plugin
```

Appium a également besoin d'autres outils pour fonctionner.

On pourrait utiliser Android Studio pour aussi faire des tests via émulateur mais dans ce tuto on va s'en passer).

Il faut installer le SDK pour Android, plus précisément les "platform-tools" :

Téléchargez sur :

<https://developer.android.com/tools/releases/platform-tools?hl=fr>

Décompressez l'archive et placez le dossier "platform-tools" dans un dossier suivant cette adresse en créant les dossiers nécessaires : "**C:\Program Files\Android\cmdline-tools\latest\platform-tools**"

Il faut maintenant que nous créons les variables d'environnement, lancer PowerShell **en mode Administrateur** via le menu démarrer et entrez les commandes suivantes :

Java JDK :

```
[System.Environment]::SetEnvironmentVariable("JAVA_HOME", "C:\Program Files\Java\jdk-24", "Machine")
```

Si le chemin des programmes sur votre machine est différent, adaptez la commande.

Android :

```
[System.Environment]::SetEnvironmentVariable("ANDROID_HOME", "C:\Program files\Android", "Machine")
```

PATH Android :

```
$oldPath = [System.Environment]::GetEnvironmentVariable("Path", "Machine")
```

```
$newPath = ";C:\Program Files\Android\cmdline-tools\latest\platform-tools"
```

```
[System.Environment]::SetEnvironmentVariable("Path", "$oldPath$newPath", "Machine")
```

```
[System.Environment]::SetEnvironmentVariable("Path", "$oldPath$newPath", "Machine")
```

Vérification :

```
echo $env:JAVA_HOME
```

```
echo $env:ANDROID_HOME
```

```
echo $env:Path
```

Redémarrez votre machine ou forcez la prise en compte immédiate avec :

```
Stop-Process -Name explorer -Force
```

```
Start-Process explorer
```

Enfin, installez les bibliothèques Python Appium :

```
pip install Appium-Python-Client
```

```
pip install selenium
```

Premier test avec Appium :

Dans ce tutoriel, nous allons créer un test, son but sera : d'ouvrir l'application calculatrice, de faire un calcul simple (2+3) et de vérifier si le bon résultat s'affiche.

Branchez le téléphone à votre machine via USB, si proposé, activez le débogage USB, sinon rendez-vous dans les paramètres, allez dans "À propos du téléphone" puis tapez 7 fois sur "Numéro de build" pour activer les options développeur. Et enfin dans les options développeur, activez le débogage USB.

Tapez cette commande pour afficher les appareils connectés :

```
adb devices
```

Si tout est bon, on peut continuer.

Comme nous allons faire un test sur l'app calculatrice, nous devons indiquer à Appium l'activité correspondant à la calculatrice, tapez dans votre terminal :

```
adb shell dumpsys window | findstr mCurrentFocus
```

Vous devrez obtenir :

```
mCurrentFocus=Window{7e07f22 u0  
com.sec.android.app.popupcalculator/com.sec.android.app.popupcalculator.Calculator}
```

com.sec.android.app.popupcalculator = Le package de l'app

com.sec.android.app.popupcalculator.Calculator = L'activité de l'app

La sortie dépend du téléphone mais le format reste le même, adaptez donc le tutoriel pour vous.

Ensuite, lancez Appium avec le plugin et adb activé avec une activation du mode non sécurisé avec cette commande :

```
appium --use-plugins=inspector --allow-cors --allow-insecure adb_shell
```

Une fois Appium démarré, allez dans votre navigateur et rendez-vous sur :

<http://localhost:4723/inspector>

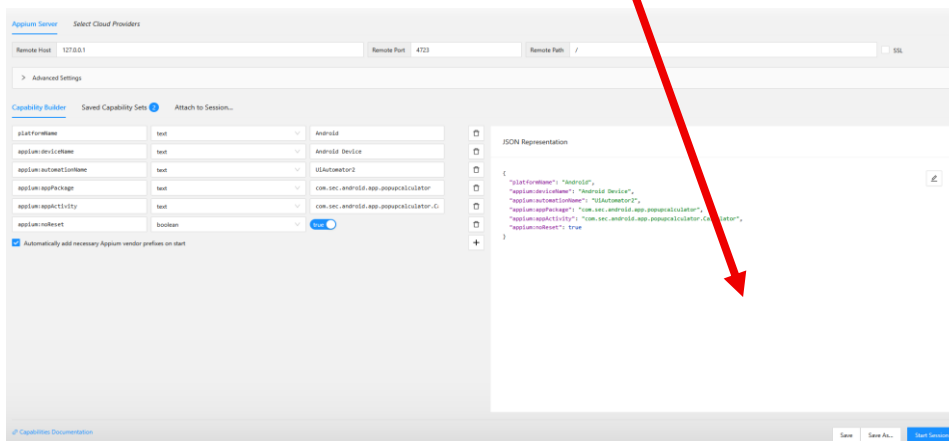
Laissez les paramètres par défaut, tout ce que vous devez ajouter est la configuration, elle est écrite en format JSON, cliquez sur l'icône modifier à droite et entrez :

```

{
  "platformName": "Android",
  "appium:deviceName": "Android Device",
  "appium:automationName": "UiAutomator2",
  "appium:appPackage": "com.sec.android.app.popupcalculator ",
  "appium:appActivity": "com.sec.android.app.popupcalculator.Calculator ",
  "appium:noReset": true
}

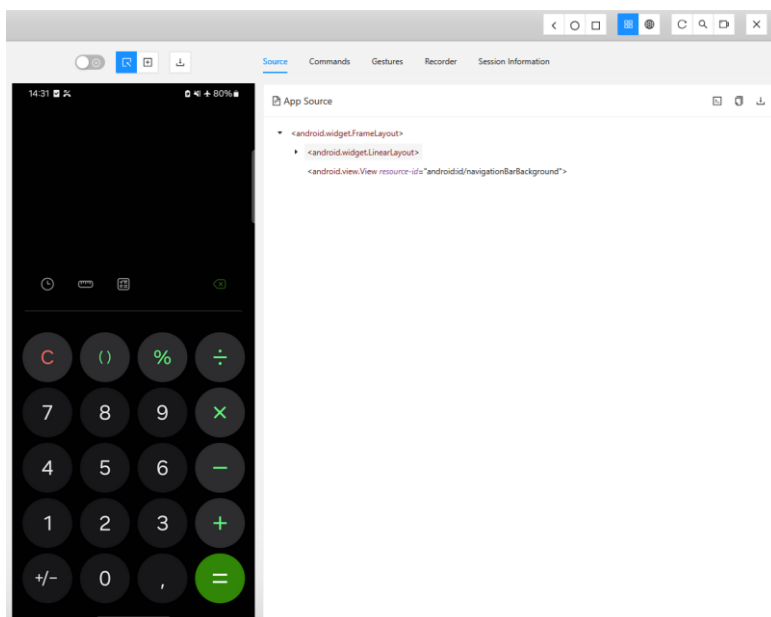
```

Faites bien adapter la ligne appPackage et appActivity avec les paramètres que l'on a vu précédemment.



Enfin, cliquez sur le bouton en bas à droite "Start Session".

Après un chargement, la page devrait montrer l'écran de votre téléphone :



Pour obtenir les caractéristiques d'un élément, cliquez simplement dessus et tout sera affiché à droite de votre écran :

Find By	Selector
accessibility id	2
id	com.sec.android.app.popupcalculatorid/calc_keypad_btn_02
-android uiautomator (docs)	new UiSelector().resourceId("com.sec.android.app.popupcalculatorid/calc_keypad_btn_02")
xpath	//android.widget.Button[@content-desc="2"]
Attribute	Value
elementId	
index	13
package	com.sec.android.app.popupcalculator
class	android.widget.Button
text	2
content-desc	2
resource-id	com.sec.android.app.popupcalculatorid/calc_keypad_btn_02
checkable	false
checked	false
clickable	true
enabled	true

Maintenant, rédigeons le code de notre test, retournez dans visual studio code, notre test peut être réalisé grâce à ce code :

Les commentaires (commençant par un #) expliqueront le code.

Importation des bibliothèques nécessaires

```
from appium import webdriver
```

```
from appium.options.android import UiAutomator2Options
```

```
from appium.webdriver.common.appiumby import AppiumBy
```

```
import time
```

Configuration du test

```
def setup_driver():
```

```
    # Paramètres (vous retrouvez le package et l'activité donc adaptez-les comme vous l'aviez fait avant dans l'inspecteur)
```

```
    options = UiAutomator2Options()
```

```
    options.platform_name = "Android"
```

```
    options.device_name = "Android Device"
```

```
    options.app_package = "com.sec.android.app.popupcalculator"
```

```
options.app_activity = "com.sec.android.app.popupcalculator.Calculator"
```

```
options.no_reset = True
```

```
driver = webdriver.Remote("http://localhost:4723", options=options)
```

```
return driver
```

Définition du test

```
def test_addition():
```

```
    # Initialisation du test
```

```
    driver = setup_driver()
```

```
    # Try = essayer un bout de code
```

```
    try:
```

```
        # Pour désigner un élément on part toujours de "driver", expliquant le code "driver"
```

```
        # Trouver l'élément de driver avec le XPATH puis cliquer dessus
```

```
        # Bouton "2"
```

```
        driver.find_element(AppiumBy.XPATH, '//android.widget.Button[@content-desc="2"]').click()
```

```
        # Bouton "+"
```

```
        driver.find_element(AppiumBy.XPATH, '//android.widget.Button[@content-desc="Plus"]').click()
```

```
        # Bouton "3"
```

```
        driver.find_element(AppiumBy.XPATH, '//android.widget.Button[@content-desc="3"]').click()
```

```
        # Bouton "="
```

```
        driver.find_element(AppiumBy.XPATH, '//android.widget.Button[@content-desc="Calcul"]').click()
```

```
        # Attendre 1 seconde pour être sûr que le résultat a eu le temps de s'afficher afin de ne pas provoquer une erreur pour rien (nécessite d'importer "time")
```

```
        time.sleep(1)
```

```
        # Trouver l'élément de driver avec le XPATH correspondant à l'affichage du résultat
```

```
        result_element = driver.find_element(AppiumBy.XPATH, '//android.widget.EditText[@resource-id="com.sec.android.app.popupcalculator:id/calc_edt_formula"]')
```

```
        # Récupérer le texte de l'élément
```

```
result = result_element.text.strip()
```

```
# Vérifier si 5 est affiché dans l'élément
```

```
assert "5" in result
```

```
# Afficher "Test réussi !" (Si la ligne au dessus provoque une erreur, le test se terminera avec une erreur et le texte ne sera pas affiché)
```

```
print("Test réussi !")
```

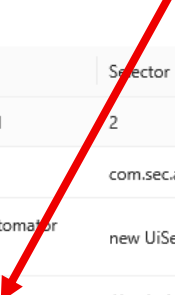
```
# Finally permet d'exécuter du code après le "try"
```

```
finally:
```

```
# Mettre fin au test
```

```
driver.quit()
```

Pour trouver le XPATH d'un élément il faut cliquer sur cet élément dans l'inspecteur et il est affiché ici :



Find By	Selector
accessibility id	2
id	com.sec.android.app.popupcalculator:id/calc_keypad_btn_02
-android uiautomator (docs)	new UiSelector().resourceId("com.sec.android.app.popupcalculator:id/calc_keypad_btn_02")
xpath	//android.widget.Button[@content-desc="2"]

Attribute	Value
elementId	
index	13
package	com.sec.android.app.popupcalculator
class	android.widget.Button
text	2
content-desc	2
resource-id	com.sec.android.app.popupcalculator:id/calc_keypad_btn_02
checkable	false
checked	false
clickable	true
enabled	true

Tapez ou copiez-collez ce code et exécutez-le avec :

Pytest

L'application devrait se lancer et faire le calcul tout seul !

Si 5 est bien affiché (car 2+3 égalent 5), le test est un succès et c'est vert :

```
===== test session starts =====
platform win32 -- Python 3.11.9, pytest-8.4.1, pluggy-1.6.0
rootdir: C:\Users\Eleve\Desktop\Stage\Tuto\Ex_code\Appium
plugins: base-url-2.1.0, playwright-0.7.0
collected 1 item

test_appium.py . [100%]

===== 1 passed in 36.45s =====
```

Conseil : Pensez à fermer l'application testée avant chaque test, une erreur pourrait se produire, ex : un ancien résultat est là, ou un bouton a été modifié... ce qui ferait planter le test.

Pour plus d'infos sur Appium et Pytest :

Documentation Appium : <https://appium.github.io/python-client-sphinx/>

Documentation Pytest : <https://docs.pytest.org/en/stable/>

Mon projet pour passer un appel téléphonique et en répondre à un autre automatiquement : https://github.com/Superdiamant7/automatic_call_appium