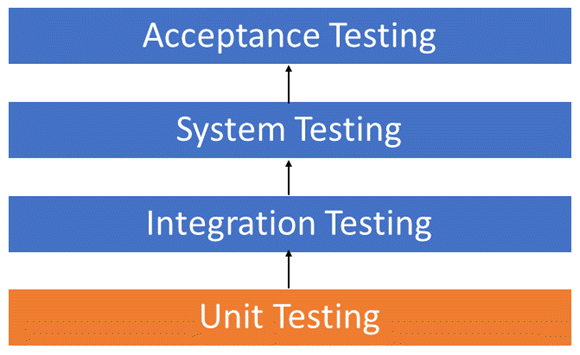
# **Tests**

On distingue différentes catégories de tests :

* + 1. **Tests unitaires :** testent une partie (une unité) d’un système afin de s’assurer qu’il fonctionne correctement (**build the system right**)
    2. **Tests d’acceptation :** testent le système afin de s’assurer qu’il est conforme aux besoins (**build the right system**)
    3. **Tests d’intégration :** testent le système sur une plate-forme proche de la plate-forme cible
    4. **Tests de sécurité :** testent que l’application ne contient pas de failles de sécurité connues (injection de code, attaque XSS, …)
    5. **Tests de robustesse :** testent le comportement de l’application aux limites des ressources disponibles (mémoire, CPU, …) sur la plate-forme



1. **Smoke test :**

* Imaginons, par exemple, que vous développez un site web. Votre **Smoke test** peut utiliser *curl* pour vérifier que le site est accessible et que la récupération (fetch) de la page produit un code d’état HTTP 200 (OK). Si la récupération (fetch) de la page d’accueil génère un autre code, tel que 404 (*Introuvable*) ou 500 (*Erreur interne du serveur*), vous savez que le site web ne fonctionne pas. Vous savez également que cela ne sert à rien d’exécuter d’autres tests. Au lieu de cela, vous diagnostiquez l’erreur, la corrigez et redémarrez vos tests.
* Les **Smoke tests** vérifient les fonctionnalités les plus basiques de votre application ou service. Ces tests sont souvent exécutés avant des tests plus complets et exhaustifs. Les **Smoke tests** doivent s’exécuter rapidement.

1. **Tests d’intégration :**

* Par exemple, un système d’e-commerce peut inclure un *site web*, une DB de produits et un système de paiement. Vous pouvez écrire un **test d’intégration** qui ajoute des éléments au panier d’achat, puis achète les articles. Le test vérifie que l’application web peut se connecter à la base de données de produits, puis traiter la commande.
* Les *tests d’intégration* vérifient que plusieurs composants logiciels fonctionnent ensemble pour constituer un système complet
* Vous pouvez combiner des tests unitaires et des tests d’intégration pour créer une stratégie de test en couches. Par exemple, vous pouvez exécuter des tests unitaires sur chacun de vos composants avant d’exécuter les tests d’intégration. Si tous les tests unitaires réussissent, vous pouvez passer aux tests d’intégration avec une plus grande confiance.

1. **Tests de régression :**

* Une ***régression*** se produit quand le comportement existant change ou s’arrête après l’ajout ou la modification d’une fonctionnalité. Les ***tests de régression*** permettent de déterminer si des modifications du code, de la configuration ou autres affectent le comportement général du logiciel.
* Les tests de régression sont importants, car une modification apportée à un composant peut affecter le comportement d’un autre composant.
* Imaginons par exemple que vous optimisez une base de données pour les performances d’écriture. Les performances de lecture de cette base de données, qui sont gérées par un autre composant, peuvent chuter de manière inattendue. La chute des performances de lecture est une régression.

1. **Tests de l’Interface Utilisateur (UI tests) :**

* Un **test unitaire** ou un **test d’intégration** peut vérifier que l’interface utilisateur *reçoit* correctement les données. Mais les **UI tests** aident à vérifier que l’UI s’affiche correctement et que le résultat fonctionne comme prévu pour l’utilisateur.
* Vous pouvez utiliser des **UI tests** pour vérifier que l’UI se comporte comme prévu sur les *différents navigateurs web*.
* Un **UI test** peut, par exemple, vérifier que l’animation correcte s’affiche en réponse à un clic sur un bouton. Un deuxième test peut vérifier que la même animation s’affiche correctement quand la fenêtre est redimensionnée.

# Install & Antisèches

* + - 1. [Pytest](https://docs.pytest.org/en/6.2.x/contents.html) :
         1. [Intro-Guru](https://www.guru99.com/pytest-tutorial.html), memento : [[CLI](https://cheatography.com/nanditha/cheat-sheets/pytest/) , [guide de survie](https://gist.github.com/kwmiebach/3fd49612ef7a52b5ce3a), [cheat sheet](https://cheatography.com/amicheletti/cheat-sheets/pytest-ing/)]
         2. Exemple : pkg Pandas avec les tests qui vont avec [[GitHub](https://github.com/pandas-dev/pandas/tree/v1.3.1/pandas), [TDD](https://pandas.pydata.org/docs/development/contributing_codebase.html#test-driven-development-code-writing), [Wiki Testing](https://github.com/pandas-dev/pandas/wiki/Testing)]
      2. Extensions VS Code (pour les tests) :
         1. *Extensions par défaut* ([Python](https://marketplace.visualstudio.com/items?itemName=ms-python.python) + [Pylance](https://marketplace.visualstudio.com/items?itemName=ms-python.vscode-pylance)) :

Guide d’utilisation pas à pas [[doc offic1](https://code.visualstudio.com/docs/python/testing#_enable-a-test-framework), article [DZone](https://dzone.com/articles/vs-code-setup-for-python-development-and-testing)],

Mon propre fichier de configuration [[**.vscode\***settings*.json](https://1drv.ms/u/s!AmJGbSlW18YGscdglSoMziNpsHgf5A?e=DPIayC)]

* + - * 1. [TestExplorerUI](https://marketplace.visualstudio.com/items?itemName=LittleFoxTeam.vscode-python-test-adapter) : *Extension qui fonctionne beaucoup mieux* (à installer)

Parmi ses avantages : indiquer le *msg d'erreur* dans un pop-up, *temps d'exécution*, etc.

* + - * 1. [TestFileMagic](https://marketplace.visualstudio.com/items?itemName=herbcaudill.test-file-magic) *ou encore* [TestSwitcher](https://marketplace.visualstudio.com/items?itemName=bmalehorn.test-switcher)(à tester ?)

Pratique pour les projets avec des centaines de tests à passer : *Switcher* facilemententre lefichier de test et le fichier source correspondant, *Créer* le fichier test s’il n’existe pas, etc.

* + - * 1. Extension [DocTests](https://marketplace.visualstudio.com/items?itemName=NoahSyn10.pydoctestbtn)
      1. PyCharm : [[Playlist vid offic JetBrains de 9 vid](https://www.youtube.com/watch?v=dAbpPklX7wo&list=PLCTHcU1KoD99Rim2tzg-IhYY2iu9FFvNo), doc offic [article](https://www.jetbrains.com/help/pycharm/pytest.html#pytest-fixtures)]
      2. Couverture du code : pkg [coverage](https://github.com/nedbat/coveragepy), [quick-start](https://coverage.readthedocs.io/en/coverage-5.5/#quick-start), [pytest-cov](https://pytest-cov.readthedocs.io/en/latest/index.html)
      3. Sites pour s’entraîner dessus : [uitestpractice](http://www.uitestpractice.com/)

# [Doctest](https://docs.python.org/3/library/doctest.html) :

* + - * 1. Cours : [[docstringtest.py](https://1drv.ms/u/s!AmJGbSlW18YGschK0FbpXrHIytBH6A?e=MWsWDp)], Ressources : [sametmax](https://sametmax.oprax.fr/les-docstrings/index.html), [pymotw](https://pymotw.com/3/doctest/), [Doc offic](https://docs.pytest.org/en/6.2.x/doctest.html) : [doctest + pytest ]
        2. **Q : *pytesting-doctests-in-visual-studio-code-debugger* ?**Puis-je lancer des doctests avec pytest en passant par VSCode ? R : [stackoverflow](https://stackoverflow.com/questions/51192937/pytesting-doctests-in-visual-studio-code-debugger)
        3. **Q : *how-to-let-pytest-discover-and-run-doctests-in-installed-modules*?** R : [stackoverflow](https://stackoverflow.com/questions/49254777/how-to-let-pytest-discover-and-run-doctests-in-installed-modules)



# Test-Rapport-Couverture

[test\_example.py](https://1drv.ms/u/s!AmJGbSlW18YGsdcOnXEnmsG8qQGzGg?e=2IygE3), [module\_cov.py](https://1drv.ms/u/s!AmJGbSlW18YGsdZ-Uq6F4imvW-SIXQ?e=aq0Ghh), [test\_cov.py](https://1drv.ms/u/s!AmJGbSlW18YGsdZ9Rcm3AzDaWpc7iQ?e=vZvUHX)

# Test de performance avec Locust

Folder de script : [cliquer ici](https://1drv.ms/u/s!AmJGbSlW18YGsddPPBVpfXxKNPhyDQ?e=z0DPtn)

# Tester une WebApp

**Ressources :**

* + - * 1. Série d’articles : [[les bases](https://medium.com/swlh/unit-testing-in-python-basics-21a9a57418a0), [How to Test Flask Applications](https://medium.com/analytics-vidhya/how-to-test-flask-applications-aef12ae5181c) *with Databases, Templates and Protected Pages*, …]
        2. Video : [[Playlist de 10 vid](https://www.youtube.com/watch?v=dj6Z1HBbDN8&list=PL_ppFIFzf4EFbnx9VvOjqAdjJRDh-HAxK)]
        3. **Quelles sont les différentes façons de lancer des tests en CLI :** [stackoverflow](https://stackoverflow.com/questions/36456920/is-there-a-way-to-specify-which-pytest-tests-to-run-from-a-file), [vid1](https://www.youtube.com/watch?v=VKY-0LEmrwk&list=PLS1QulWo1RIaNFUz4zrztWlCJgkpXht-H&index=2), [vid2](https://www.youtube.com/watch?v=FDGYhuCOgEY&list=PLL34mf651faNqwhZEM91zU3c-dcc4dLF0&index=3)

**Répondre aux questions suivantes En *ligne de commande* et à l’aide de l’*extension VSCode* pour s’assurer que :**

N.B : Vous n’avez pas besoin que le serveur web soit en marche pour réaliser les tests

* 1. La WebApp se lance correctement et la page *Home* s’affiche proprement ?

Indice : status\_code = 200, <form in response.data

[test\_route.py](https://1drv.ms/u/s!AmJGbSlW18YGscdws9oTrc024nW3xg?e=j7wMi4), se limiter au : test\_home\_page\_returns\_correct\_html\_naif

**Rappel Status Code : 200 (***succès de la requête***)**, **401 (***erreur d’authentification***)**, **403** (*accès refusé*), **404 (***page non trouvée***)**, **504 (***serveur ne répond pas***)**

* 1. La page *Home* accepte des requêtes POST (et que votre message personnalisé est affiché dans la page *bienvenue*) ?

Indice : tester.post('/', data={"text":"sayf"}, follow\_redirects=True)

[test\_route.py](https://1drv.ms/u/s!AmJGbSlW18YGscdws9oTrc024nW3xg?e=j7wMi4), se limiter au : test\_home\_page\_accepts\_post\_request

* 1. Tester que toute connexion à la page *admin* est impossible sans *user\_name* = ‘*admin*’ et *password* = ‘*admin*’ ? Vous pouvez par exemple tester la combinaison *user\_name* = ‘*wrong*’ et *password* = ‘*wrong*’. Ce type de test est valide pour le *signup* et *login*, …

Indice : tester.post('/', data={"text":"sayf"}, follow\_redirects=True)

[test\_login.py](https://1drv.ms/u/s!AmJGbSlW18YGscciEB_GXCTPFs1nxQ?e=FW5PS1)

* 1. Tester la combinaison de plusieurs paramètres (*avec un seul test*) ?

Indice : @pytest.mark.parametrize, [doc offic](https://docs.pytest.org/en/stable/parametrize.html#parametrizemark), [vid1](https://www.youtube.com/watch?v=7qMhuVGqGY4&list=PLS1QulWo1RIaNFUz4zrztWlCJgkpXht-H&index=3), [vid2](https://www.youtube.com/watch?v=OH4A-D5nAI4&list=PLL34mf651faNqwhZEM91zU3c-dcc4dLF0&index=7)

[test\_parameters.py](https://1drv.ms/u/s!AmJGbSlW18YGschInfliTbNfNGMsWQ)

* 1. Retester que toute connexion à la page *admin* est impossible en utilisant maintenant la notion @pytest.mark.parametrize pour la combinaison *user\_name*=(‘*admin*’/‘*wrong*’) et *password* = (‘*admin*’/‘*wrong*’) ?

[test\_login.py](https://1drv.ms/u/s!AmJGbSlW18YGscciEB_GXCTPFs1nxQ?e=FW5PS1), se limiter au : test\_login\_param

* 1. Tester que je ne peux pas accéder à la page bienvenue en cliquant sur ok (et ce sans avoir saisi dans le *text box*) ? Ou encore, l’*admin\_page* (sans avoir passé par la page *login*)
  2. Optimiser les derniers tests en utilisant la notion de fixture (appelé encore *test context*) ?

Pour vous aider :

* + - * 1. [test\_fixture\_example.py](https://1drv.ms/u/s!AmJGbSlW18YGsckC7GorYDsLSBdAKQ?e=Vb1t6d),
        2. Avec Selenium : Intro à Selenium [[Notebook](https://1drv.ms/u/s!AmJGbSlW18YGsdNNWY4e0VpqqQ1Dzg?e=zCCfdS)], [Article](https://www.seleniumeasy.com/python/using-fixtures-in-pytest), [test\_selenium\_demo.py](https://1drv.ms/u/s!AmJGbSlW18YGsdUjeuNJVigA9baZ-Q?e=boQ6xZ)
        3. [déf](https://fr.wikipedia.org/wiki/Test_fixture), [stackoverflow](https://stackoverflow.com/questions/12071344/what-are-fixtures-in-programming), [fixture\_vid](https://www.youtube.com/watch?v=6_ngzfZygGg&list=PLL34mf651faNqwhZEM91zU3c-dcc4dLF0&index=5)
        4. Lien vers la Doc offic : [fixture](https://docs.pytest.org/en/stable/fixture.html#what-fixtures-are), [fixture scopes](https://docs.pytest.org/en/6.2.x/fixture.html?highlight=conftest#fixture-scopes), [conftest.py](https://docs.pytest.org/en/6.2.x/fixture.html?highlight=conftest#scope-sharing-fixtures-across-classes-modules-packages-or-session), [auto\_use](https://docs.pytest.org/en/6.2.x/fixture.html?highlight=conftest#autouse-fixtures-fixtures-you-don-t-have-to-request)

*Un test automatisé est un programme qui se découpe en quatre étapes dites* ***AAAA*** *pour* **Arrange***,* **Act***,* **Assert** etune dernière dite **Cleanup**

* + - * 1. **Arrange (setUp) :** La mise en place de l’environnement : création et initialisation des objets nécessaires à l’exécution du test.
        2. **Act :** Le test proprement dit.
        3. **Assert :** La vérification des résultats obtenus par le test.
        4. **Cleanup (tearDown) :** Ladésactivation et le retour à l’état initial
  1. Comment exécuter des tests en parallèle ?

Indice : pip install pytest-xdist ; pytest -n 4 ; Exemple : [test\_speed.py](https://1drv.ms/u/s!AmJGbSlW18YGsdVcJiQI9MuOslzYrQ?e=XYKzgy)

* 1. Couverture du test ?

Indice : pip install pytest-xdist ; pytest -n 4 ; Exemple : [test\_speed.py](https://1drv.ms/u/s!AmJGbSlW18YGsdVcJiQI9MuOslzYrQ?e=XYKzgy)

* 1. Relancer le dernier test sur *Chrome*, *FireFox* et *Edge* pour être sûr que le site soit compatible avec ces principaux browsers (éventuellement *safari*) ?

1è approche : [test\_selenium\_fix\_class](https://1drv.ms/u/s!AmJGbSlW18YGsdVZgTp32xZvM5Wp3w?e=w402Pc), 2è approche plus élégante : [test\_selenium\_fixt\_params.py](https://1drv.ms/u/s!AmJGbSlW18YGsdVXsVmCBEptGZRYLg?e=BsEYXH)

Indice : <https://www.blazemeter.com/blog/improve-your-selenium-webdriver-tests-with-pytest>

* 1. **Approfondissement** : Comment afficher le rapport automatiquement juste après la fin du test ?

Indice : <https://stackoverflow.com/questions/52032885/pytest-how-to-display-generated-report-after-test-run/52034116#52034116>

* 1. Tests fonctionnels/**UI tests** avec **Selenium sur Azure**

**Site à tester** [SpaceGame](https://tailspin-spacegame-web.azurewebsites.net/), [Step 1](https://docs.microsoft.com/fr-fr/learn/modules/run-functional-tests-azure-pipelines/3-set-up-environment), [Step 2](https://docs.microsoft.com/fr-fr/learn/modules/run-functional-tests-azure-pipelines/4-plan-ui-tests),

Réaliser les tests suivants sur Windows, macOS et Linux en utilisant Chrome, FireFox et Edge pour être sûr que le site soit compatible avec tous les OS/browser des joueurs qui consultant le site

1. **Q :** **Automatiser les tests des fenêtres modales** :
   * + - 1. Quand je clique sur certains éléments, comme le bouton **Download the Game**, par exemple, je vérifie que la *fenêtre modale* s’affiche correctement. Ensuite, quand je clique à l’extérieur, je vérifie que la *fenêtre modale* disparaît et que la *fenêtre principale* est à nouveau active.
         2. Je vérifie ensuite le même comportement pour les quatre images miniatures qui permettent d’afficher les exemples d’écrans de jeu.
         3. Lorsque je clique sur le joueur en tête du classement, je vérifie que son profil s’affiche (fenêtre modale, même comportement …).
   1. Réaliser le test précédent en utilisant un conftest.py ? [conftest\_vid](https://www.youtube.com/watch?v=PIwyrzCLTCI&list=PLL34mf651faNqwhZEM91zU3c-dcc4dLF0&index=6)
   2. Dynamic fixture ? Indice : [article](https://algorithmicallysound.com/2020/06/30/pytest-dynamic-fixtures/)