

# Automatisation avec Selenium

Selenium WebDriver: les bases



**A4Q**

Selenium Tester  
Foundation

# Automatisation avec Selenium

Selenium Tester Foundation

## Introduction et Objectifs

# La Formation

- **2 jours de formation alternant théorie et pratique**
  - Horaire 9h – 17h30
  - 1h30 de pause déjeuner
  - 2 pauses de 15 min (matin et après-midi)
- **Passage de la certification le troisième jour (matin)**
  - 1heure
  - QCM de 40 questions
  - 65% minimum requis

# Le matériel de formation

- Les diapositives en support du syllabus – Selenium Tester Fondation
- Les termes employés viennent du Glossaire ISTQB le plus récent ou sont définis dans l'Annexe A
- Un examen blanc est inclus pour évaluer votre préparation à l'examen Selenium Tester Fondation
- Chaque section contient des questions types afin d'évaluer vos connaissances au fur et à mesure
- Des exercices sont prévus pour les techniques principales

# Principaux domaines couverts

- 1) Bases du test automatisé
- 2) Technologies Internet pour l'automatisation des tests des applications web
- 3) Usage du WebDriver de Selenium
- 4) Préparer des scripts de test maintenables

# Acronymes (1)

<b>AKA:</b>	Also Known As (aussi connu sous le nom de)
<b>API:</b>	Application Programming Interface
<b>CI:</b>	Continuous Integration (intégration continue)
<b>CSS:</b>	Cascading Style Sheets
<b>DOM:</b>	Document Object Model
<b>GUI:</b>	Graphical User Interface
<b>HTTP:</b>	Hyper Text Transfer Protocol
<b>ISTQB:</b>	International Software Testing Qualifications Board
<b>KDT:</b>	Keyword Driven Testing (test dirigé par les données / mots-clefs)
<b>REST:</b>	Representational State Transfer
<b>ROI:</b>	Return on Investment (retour sur investissement)

# Acronymes (2)

<b>SDLC:</b>	Software Development Life Cycle (cycle de vie logiciel)
<b>SOAP:</b>	Simple Object Access Protocol
<b>SUT:</b>	System/Software Under Test (système/application sous test)
<b>TAA:</b>	Test Automation Architecture (architecture d'automatisation des tests)
<b>TAE:</b>	Test Automation Engineer (Ingénieur de test automatique)
<b>TAS:</b>	Test Automation Solution (Solution d'automatisation des tests)
<b>TCP:</b>	Transmission Control Protocol
<b>UI:</b>	User Interface (interface utilisateur)
<b>W3C:</b>	World Wide Web Consortium

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 1

### Bases de l'automatisation des tests



# Termes à mémoriser

architecture

capture/playback

comparateur

Tests exploratoires

Attaque de faute

Framework / plateforme

Crochet (de test) / test  
hook

Paradoxe du pesticide

Dette technique

testabilité

Harnais de test

Oracle de test

testware

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 1

Section 1 :

Aperçu de l'automatisation des tests

# Objectif d'apprentissage

STF-1.1 (K2) Expliquer les objectifs, avantages, désavantages et limitations de l'automatisation des tests

# Qu'est ce que l'automatisation des tests?

- L'automatisation simule habituellement une exécution de test manuel par un humain
- En dépit des aspects liés à l'automatisation, l'analyse, le design et l'implémentation restent en général manuels
- L'automatisation nécessite un support considérable incluant
  - Les environnements où sont exécutés les tests
  - Un ou plusieurs outils et harnais de test spécialisés
  - Des bibliothèques de code pour les fonctionnalités
  - Des scripts à exécuter
  - Une plateforme de suivi et de reporting afin d'évaluer les résultats

# Objectifs de l'automatisation des tests

- Améliorer l'efficacité du testing en réduisant le coût de chaque exécution
- Tester plus de choses que ne le permettrait le test manuel seul
- Réduire le temps d'exécution des tests
- Augmenter la fréquence d'exécution des tests
- Pouvoir anticiper les tests au sein du cycle de vie logiciel pour réduire la présence des défauts dans le code (tester plus tôt)

# Avantages de l'automatisation

## **L'exécution de tests automatisés peut se révéler plus efficace que des tests manuels**

- Exécuter des tests fastidieux, générateurs d'erreur humaines
- Réduire le temps d'exécution / exécuter plus de tests par build
- Augmenter la fréquence d'exécution des tests
- Permettre aux testeurs de se consacrer à des tests manuels plus complexes, plus intéressants
- Réduire les erreurs dues à l'ennui ou à la distraction des testeurs
- Anticiper les tests / tester plus tôt dans le cycle de vie logiciel (p.ex. En intégration continue, automatisation des tests unitaires, de composant ou d'intégration)
- Donner un feedback plus rapide sur la qualité du système et réduire le temps de présence des défauts dans le système
- Exécuter les tests en dehors des heures ouvrées
- Augmenter la confiance dans le build

# Désavantages de l'automatisation (1)

- Coûts supplémentaires (incluant un investissement initial élevé)
- Nouvelles technologies devant être maîtrisées par les testeurs qui n'ont pas forcément l'expertise nécessaire
- Compétences requises en développement ou service à fournir à l'équipe de test
- Maintenance importante des outils, environnements et autres éléments de test
- Augmentation de la dette technique
- L'automatisation requiert les mêmes processus, la même discipline que le développement
- Perte de vue de l'objectif initial : en se concentrant sur l'automatisation les testeurs peuvent oublier la gestion des risques du projet

# Désavantages de l'automatisation (2)

- Augmentation du *paradoxe du pesticide* avec l'automatisation (les mêmes tests sont exécutés chaque fois)
- Les défaillances trouvées sont souvent liées à l'automatisation elle-même (faux positifs) et non au système sous test
- Les outils ont tendance à être *mono-focus* en se concentrant sur un résultat unique face à un événement donné, quand il y en a souvent plus d'un
- Programmation intelligente requise: les outils sont bornés et stupides, à l'inverse des testeurs



# Les risques liés au projet d'automatisation

- Attentes irréalistes du management qui pousse souvent l'automatisation dans la mauvaise direction
- Vision court-termiste destructrice; seule une vision à long terme est profitable
- Fausse impression de sécurité quand de nombreux tests ont été exécutés et qu'ils ont trouvé peu de défauts
- Problèmes techniques possibles quand le projet s'appuie sur des outils trop à la pointe
- Coopération avec le développement pouvant amener à des problèmes organisationnels

# Les limites de l'automatisation

- Nécessité d'avoir une maturité suffisante dans l'organisation; de mauvais processus de tests une fois automatisés ne donnent que de mauvais résultats plus rapidement (voire même plus lentement parfois)
- Tous les tests ne peuvent, ni ne doivent être automatisés
- Les tests manuels sont toujours nécessaires (tests exploratoires...)
- L'analyse, le design ou l'implémentation des tests resteront probablement manuels
- Les humains trouvent la plupart des défauts, l'automatisation ne trouve que ce qu'elle a été programmée à trouver

# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 1

Bases de l'automatisation des tests

Section 2 :

Tests manuels vs. tests automatisés

# Objectifs d'apprentissage

STF-1.2 (K2) Comprendre la relation entre tests manuels et tests automatisés

# Problèmes lié à l'automatisation

- Les versions anciennes d'outils d'automatisation ont un fort taux d'échec
- Leur problème principal: une modélisation incorrecte des tests
  - L'activité de test était largement sous-estimé (tests trop simplistes)
  - Chaque test était exécuté à l'identique à chaque fois (paradoxe du pesticide)
  - Les scripts n'étaient pas maintenables : chaque changement d'IHM impactait l'exécution
- En réalité, les tests manuels impliquent de nombreuses décisions de la part du testeur; donc beaucoup d'intelligence
- Les scripts de tests comportent 3 colonnes, dans leur plus simple expression
  - Une version réduite de la tâche à accomplir
  - Les données liées à la tâche
  - Le comportement attendu

# Contexte et caractère raisonnable

- En soi, un script de test manuel n'a que peu de valeur
- C'est la compréhension du testeur (manuel) qui apporte sa valeur au test
- Le testeur apporte :
  - 1) Un contexte: une compréhension du système, que faire, où, quand, et comment...
  - 2) Une interprétation raisonnable des résultats selon des circonstances données.
- Les outils et les scripts sont idiots: peu de contexte et aucune capacité d'interprétation.
- Toutefois, les outils d'automatisation reposent tous sur un langage de programmation
- C'est à l'automaticien d'utiliser le langage de programmation pour apporter contexte et sens à son script

# Nécessité de savoir programmer

- Un script automatisé est un programme écrit pour tester un autre programme
- L'automatisation fonctionnelle ne peut réussir que si l'intelligence du testeur est ajoutée au script au moyen de la programmation
- Cela demande :
  - L'analyse de ce qui se passe dans le test
  - Le design pour résoudre les problèmes rencontrés
  - L'implémentation d'outils et de librairies
  - La documentation
  - La maintenance lorsque des changements se produisent



# Tests manuels vs. automatisés

- Tous les tests ne peuvent, ni ne doivent, être automatisés
- Un business case est nécessaire avant d'automatiser un test :
  - Doit-il être exécuté souvent pour apporter de la valeur?
  - A quelle cadence doit-il être maintenu?
  - Comprendons-nous le contexte et le degré d'interprétation raisonnable requis?
  - Est-ce qu'un raisonnement dynamique (testeur) est requis pour ce test?
  - Le résultat est-il hautement automatisable?
  - Faut-il une intervention manuelle au cours du test?
- Les tests manuels ne sont jamais obsolètes; certains seront encore et toujours nécessaires

# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 1

Bases de l'automatisation des tests

Section 3 :

Facteurs de succès

# Objectif d'apprentissage

STF-1.3 (K2) Identifier les facteurs techniques de succès pour un projet d'automatisation des tests

# Comment réussir

- L'automatisation ne réussit pas **par accident**
- L'organisation nécessite les mêmes facteurs de succès que pour tout autre projet de développement
  - Une vue à long terme alignée sur les besoins métier
  - Un bon management et une attention redoublée pour les détails
  - Des processus matures
  - Une architecture et une plateforme d'automatisation formalisées
  - Une formation adéquate tant en matière de test que de développement
  - Pouvoir concevoir et utiliser une bonne documentation
- Rien ne peut garantir le succès, mais l'absence des critères ci-dessus précipite certainement le projet vers sa chute

# Testabilité

- L'automatisation peut rarement réussir de manière isolée
- Le soutien actif et l'approbation du développement sont précieux
- L'application doit être conçue pour être testable (ex : id sur les éléments html)
- Il y a différents niveaux d'interfaces pour automatiser
  - Le niveau GUI (souvent fragile et propice aux erreurs )
  - Application programming interface (API) au niveau public et protégé
  - Les interfaces API d'usage privé (hooks)
  - Le niveau des protocoles (HTTP, TCP, UD, etc.)
  - Le niveau des services (SOAP, REST, etc.)
- Plus on est bas dans cette liste et plus robustes / moins fragiles seront les tests

# Facteurs de succès, pour conclure...

- Management averti qui sait ce qu'il est possible (ou non) de faire
- Collaboration des développeurs prêts à s'investir et à travailler avec les testeurs
- Application sous test conçue pour être testable
- Développer un business case pour les court, moyen et long termes
- Le choix d'outils appropriés à l'environnement et au système sous test
- Une bonne formation aux tâches de test et de développement
- Avoir une stratégie d'automatisation des tests bien documentée, acceptée et soutenue par le management
- Un plan de maintenance de l'automatisation formel et bien documenté
- Automatiser au bon niveau d'interface en fonction du contexte de test requis

# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin



# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 1

Bases de l'automatisation des tests

Section 4:

Risques et bénéfices du WebDriver de Selenium

# Objectif d'apprentissage

STF-1.4 (K2): Comprendre les risques et bénéfices du WebDriver de Selenium

# Le WebDriver de Selenium

- Une interface de programmation pour développer des scripts Selenium
- Le WebDriver supporte les langages suivants
  - C#
  - Haskell
  - Java
  - JavaScript
  - Objective C
  - Perl
  - PHP
  - Python
  - R
  - Ruby
- Selenium WebDriver fournit une API unifiée ainsi que des implémentations pour un ensemble de navigateurs



# Avantages du WebDriver de Selenium

- L'exécution des tests est cohérente et reproductible
- Bien taillé pour les tests de régression
- Il peut détecter des problèmes de niveau UI que les tests unitaires ou de niveau API ne verraient pas
- Open Source (investissement initial limité)
- Permet un test rapide de compatibilité des navigateurs
- Connections avec différents langages – communauté de testeurs plus large

# Risques du WebDriver de Selenium

- Passer trop de temps à tester au niveau UI (niveau navigateur) et pas assez au niveau unitaire
- Alourdir le temps total d'exécution des tests de régression dans une approche d'intégration continue (CI)
- Tentation d'ignorer les tests plus difficiles à automatiser
- Perte de rentabilité par rapport au test manuel quand l'application web ne requiert pas beaucoup de changements

# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 1

Bases de l'automatisation des tests

Section 5:

Le WebDriver de Selenium dans l'architecture  
d'automatisation des tests (TAA)

# Objectif d'apprentissage

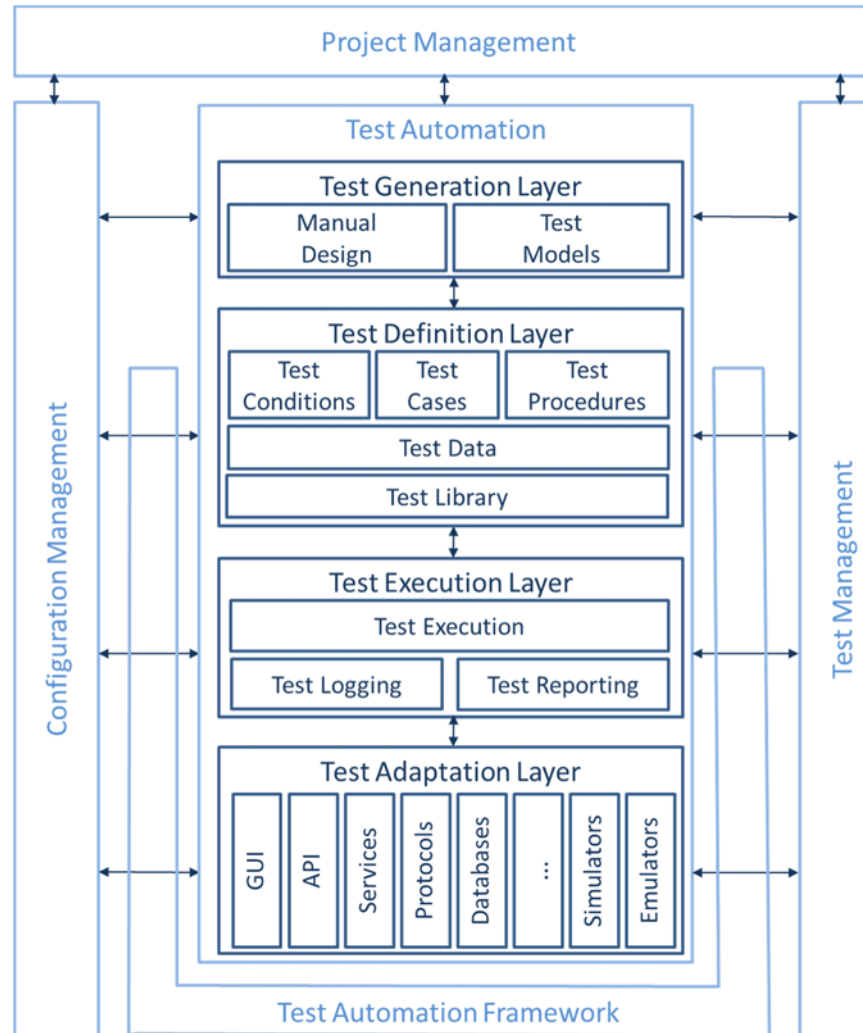
STF-1.5 (K2) Expliquer la place du WebDriver de Selenium dans la TAA



# TAA – test automation architecture

- L'architecture d'automatisation des tests (TAA) est un ensemble de couches, services et interfaces constituant une large part de la solution d'automatisation des tests (TAS)
- Elle consiste en 4 couches venant en support de l'automatisation
  - Couche de génération (des tests): design manuel et/ou automatisé des cas de test
  - Couche de définition (des tests): définition et implémentation des cas et suites de test
  - Couche d'exécution (des tests): exécution des tests automatisés et suivi / reporting des résultats
  - Couche d'adaptation (des tests): fournit objets et code pour interfacer l'application sous test à différents niveaux

# Schéma du TAA



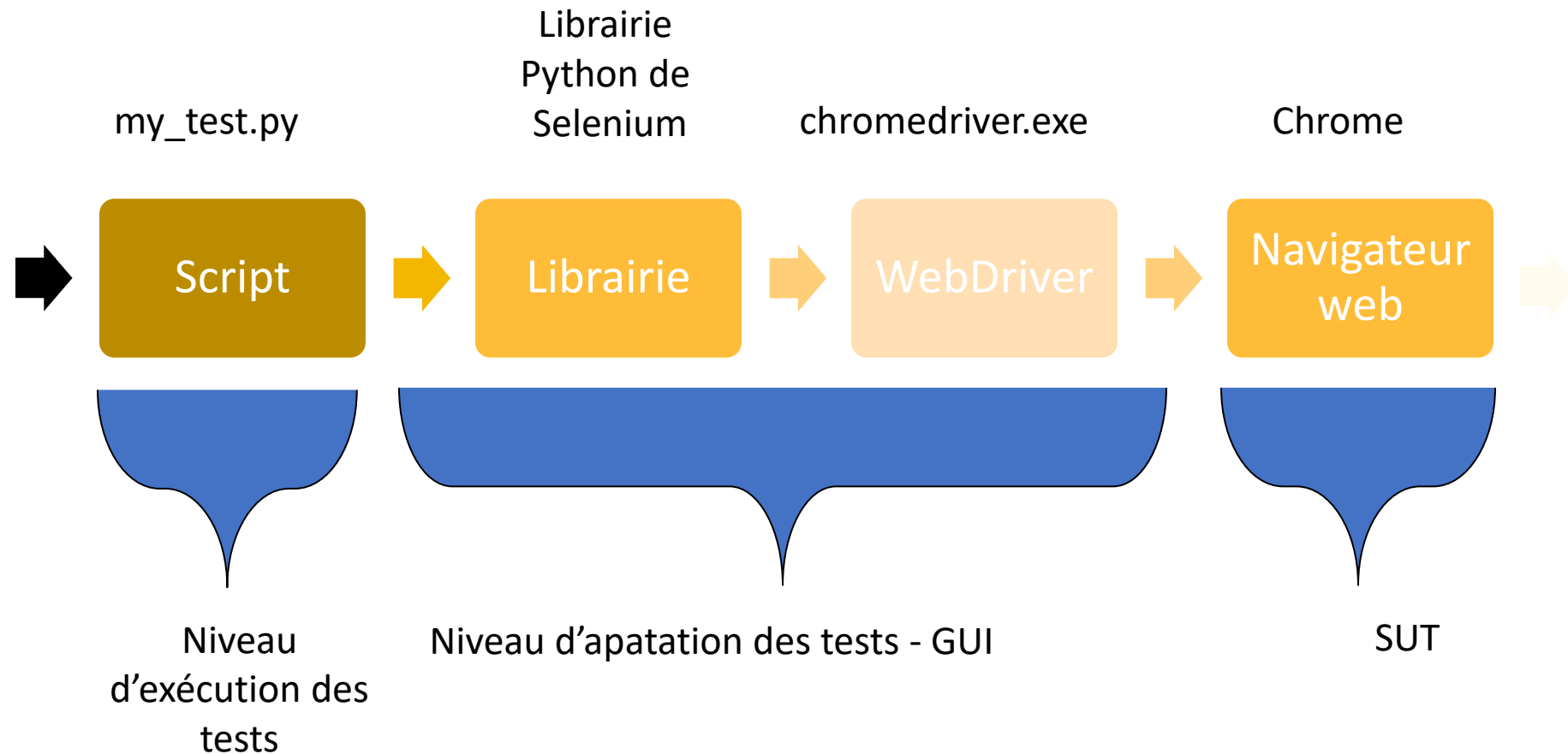
# Les avantages de l'abstraction

- Le WebDriver de Selenium est un composant de la couche d'adaptation
- La couche d'adaptation layer permet une séparation entre l'application sous test et les tests eux-mêmes
- Ceci permet aux cas de test d'être plus abstraits, séparés de l'application
- Quand l'application change, la fonctionnalité tend à rester la même
- Les cas de tests peuvent rester inchangés (seules de nouvelles fonctionnalités requièrent de nouveaux cas de test)
- Les scripts automatisés peuvent toutefois devoir encore changer pour exécuter l'application sous test

# Le WebDriver

- Le WebDriver, au niveau de la couche d'adaptation, fournit une interface de programmation pour l'application sous test
- Le WebDriver facilite la communication avec le SUT
  - Permet la visibilité des valeurs courantes dans le SUT
  - Permet de contrôler les objets du SUT
- Le script automatisé utilise l'API pour traduire comme suit:
  - Le script appelle une API dans le WebDriver
  - Cette API interagit avec l'objet correspondant du SUT
  - Le SUT répond à l'appel
  - Le succès ou l'échec est communiqué en retour au script
  - En cas de succès l'étape suivante du cas de test est appelée dans le script

# Exemple – Fragment de TAA avec le WebDriver de Selenium



# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 1

Bases de l'automatisation des tests

Section 6 :

Raisons et buts pour collecter des métriques

# Objectifs d'apprentissage

STF-1.6 (K2) Expliquer les raisons et les buts de collecter des métriques dans l'automatisation



# Pourquoi collecter des métriques ?

- L'automatisation coûte cher en effort, en temps et en ressources
- La plupart des organisations veulent un *business case* démontré avant de se lancer dans l'automatisation, ce qui nécessite des métriques (résultat d'un POC)
- Différentes métriques peuvent être utiles
  - Utilisabilité de la TAS\* par des analystes de test ou des automaticiens
  - Maintenabilité de la TAS
  - Performance de la suite automatisée
  - Fiabilité de la plateforme automatisée
- Malheureusement, de nombreuses métriques collectées au sein des projets d'automatisation sont loin d'être utiles

*\*Test automation solution*

# Problèmes de collecte des métriques

- L'automatisation est un investissement qui n'apporte pas de valeur à court terme. Celle-ci n'apparaît que sur le long terme
- Tenter de prouver la valeur au démarrage requiert habituellement une comptabilité créative
- Le management insiste souvent pour que l'on prouve très tôt la valeur, avec pour conséquence que les automaticiens (TAE\*) développent un grand nombre des tests inutiles et les exécutent tout aussi inutilement
- Essayer de collecter des métriques pertinentes peut être un sujet de distraction pour une équipe qui débute
- La projection de coûts et de valeur liés à l'exécution future d'un test donné est hautement aléatoire

*\*Test automation ingeneers*

# Pommes vs. oranges

Les coûts des **tests manuels** peuvent être fixés assez aisément

- Parce que la cas de test est abstrait, une fois développé, il ne doit être modifié que lorsqu'un changement majeur du SUT apparaît
- La plupart des coûts futurs sont liés au temps d'exécution nécessaire
- Peu de faux positifs avec un testeur manuel expérimenté

Le coût des **tests automatisés** présente beaucoup plus de variables

- Parce que le script de test est concret, il doit être modifié à presque chaque changement mineur du SUT
- Les coûts futurs peuvent être élevés en cas de changement du SUT
- Identifier et corriger un faux-positif peut revenir cher

La plupart des coûts d'automatisation ne peuvent être qu'estimés

Ces estimations tendent à être manipulées pour prouver ce que l'on veut au lieu d'être réaliste

# Métriques utiles

- Effort de test de régression évité grâce à l'automatisation.
- Coût de l'effort de maintenance, de support et de développement de l'automatisation par l'équipe d'automaticiens.
- Nombre d'exécutions réussies entre 2 défaillances
- Motifs de défaillances dans l'automatisation (points communs, tendances lourdes en suivant l'analyse des causes racines).
- Nombre des défaillances de l'automatisation en comparaison du nombre de défaillances du SUT trouvées grâce à l'automatisation.

# Pour conclure

- Essayez de trouver des métriques justifiant l'importance et la pertinence du projet
- Expliquez au management pourquoi il est difficile d'enregistrer des métriques utiles (et vraies)
- Eviter de leurrer le monde en cherchant à démontrer la valeur immédiate de l'automatisation
  - Tant qu'elle n'a pas acquis une certaine maturité, l'automatisation ne délivrera que peu de valeur.
  - En trichant sur la vraie valeur de l'automatisation, vous risquez de vous faire prendre la main dans le sac et/ou de tuer le projet.

# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 1

Bases de l'automatisation des tests

Section 7 :

La boîte à outils de Selenium

# Objectifs d'apprentissage

STF-1.7 (K2) Comprendre et pouvoir comparer les objectifs d'utilisation de la boîte à outils de Selenium (WebDriver, Selenium Server, Selenium Grid)



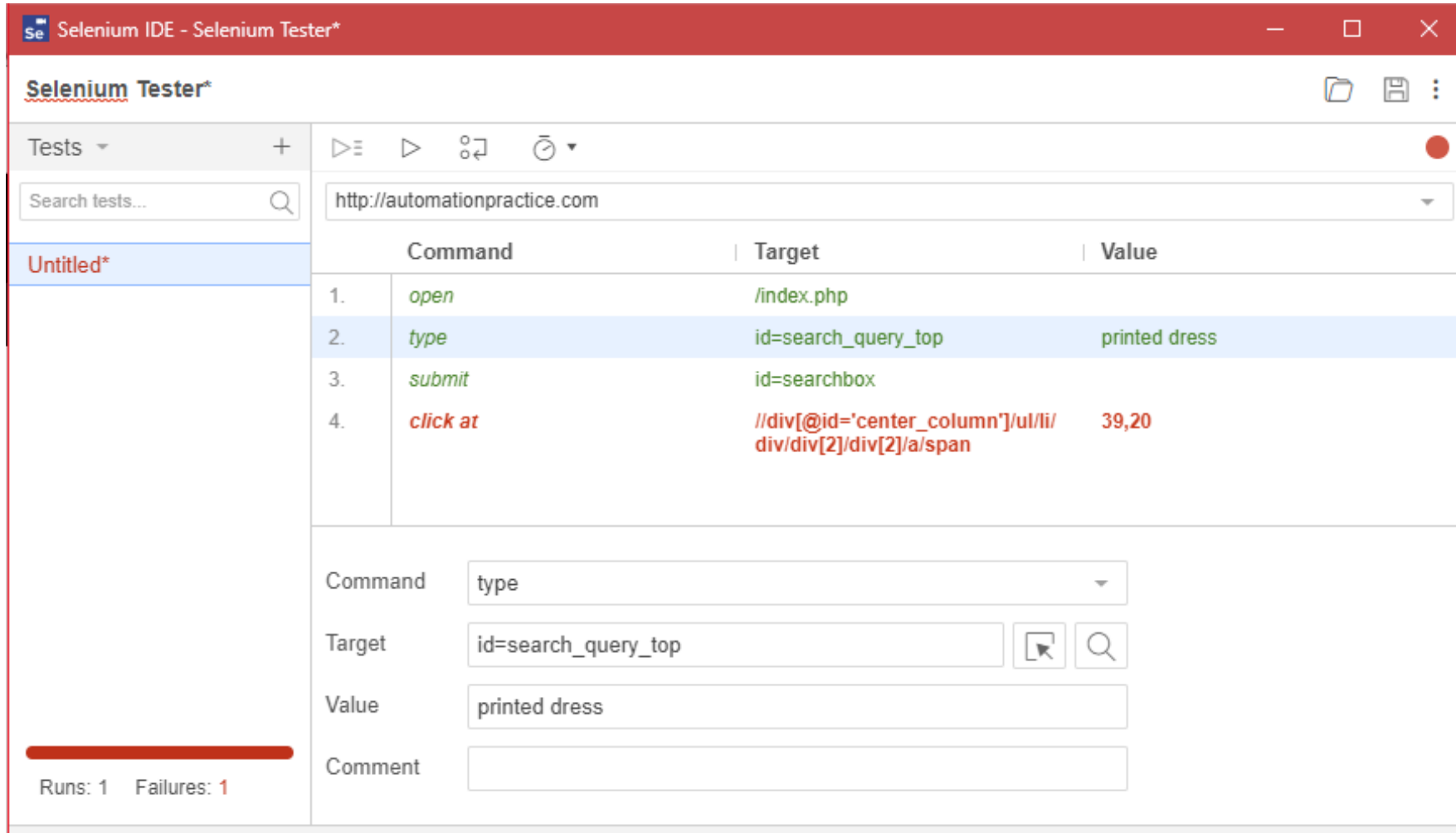
# Selenium

- Introduit pour la première fois en 2004
- A l'origine un outil interne à ThoughtWorks, rendu OpenSource la même année
- Consiste actuellement en 4 outils
  - Selenium IDE
  - Selenium WebDriver
  - Selenium Grid
  - Selenium Standalone Server

# Selenium IDE

- Add-on à Chrome et Firefox
- Facile d'usage
- Enregistre les actions de l'utilisateur sur les pages web
- Rejoue les scripts enregistrés
- Permet l'insertion de points de validation
- Ne dispose ni de variables, ni de fonctions ou de structure de contrôle de flot
- Trouve des localisateurs logiques (*sensible locators*)

# Selenium IDE: Example



The screenshot displays the Selenium IDE interface. The top bar shows the title 'Selenium IDE - Selenium Tester\*'. Below it, the 'Selenium Tester\*' window contains a 'Tests' tab with a search bar and a dropdown menu set to 'http://automationpractice.com'. The main area shows a list of test steps for an 'Untitled\*' test suite:

	Command	Target	Value
1.	open	/index.php	
2.	type	id=search_query_top	printed dress
3.	submit	id=searchbox	
4.	click at	//div[@id='center_column']/ul/li/div/div[2]/div[2]/a/span	39,20

The 'type' command in step 2 is selected. The bottom panel shows the configuration for this command:

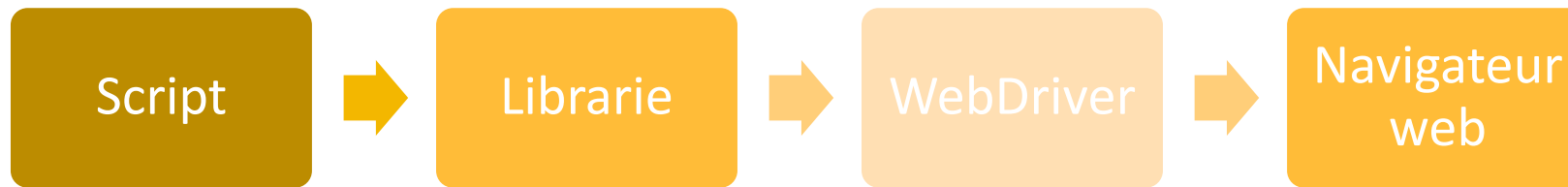
- Command: type
- Target: id=search\_query\_top
- Value: printed dress
- Comment: (empty)

At the bottom left, a status bar indicates 'Runs: 1' and 'Failures: 1'.

# Le WebDriver de Selenium

- WebDriver est une plateforme permettant à des scripts de contrôler les navigateurs
- Il dispose d'APIs pour de nombreux langages de programmation (p.ex. Python, Java, C#, Ruby)
- Peut requérir des pilotes spécifiques pour certains navigateurs (téléchargés séparément)
- Dans cette formation nous emploierons Python pour Selenium WebDriver

# Selenium WebDriver: Architecture



- Quand un script crée une instance pour un objet d'une librairie de WebDriver, la librairie démarre un processus WebDriver qui ouvre le navigateur
- Le script appelle les commandes de la librairie et cette dernière les envoie au navigateur au travers de WebDriver,

# Selenium WebDriver: Exemple

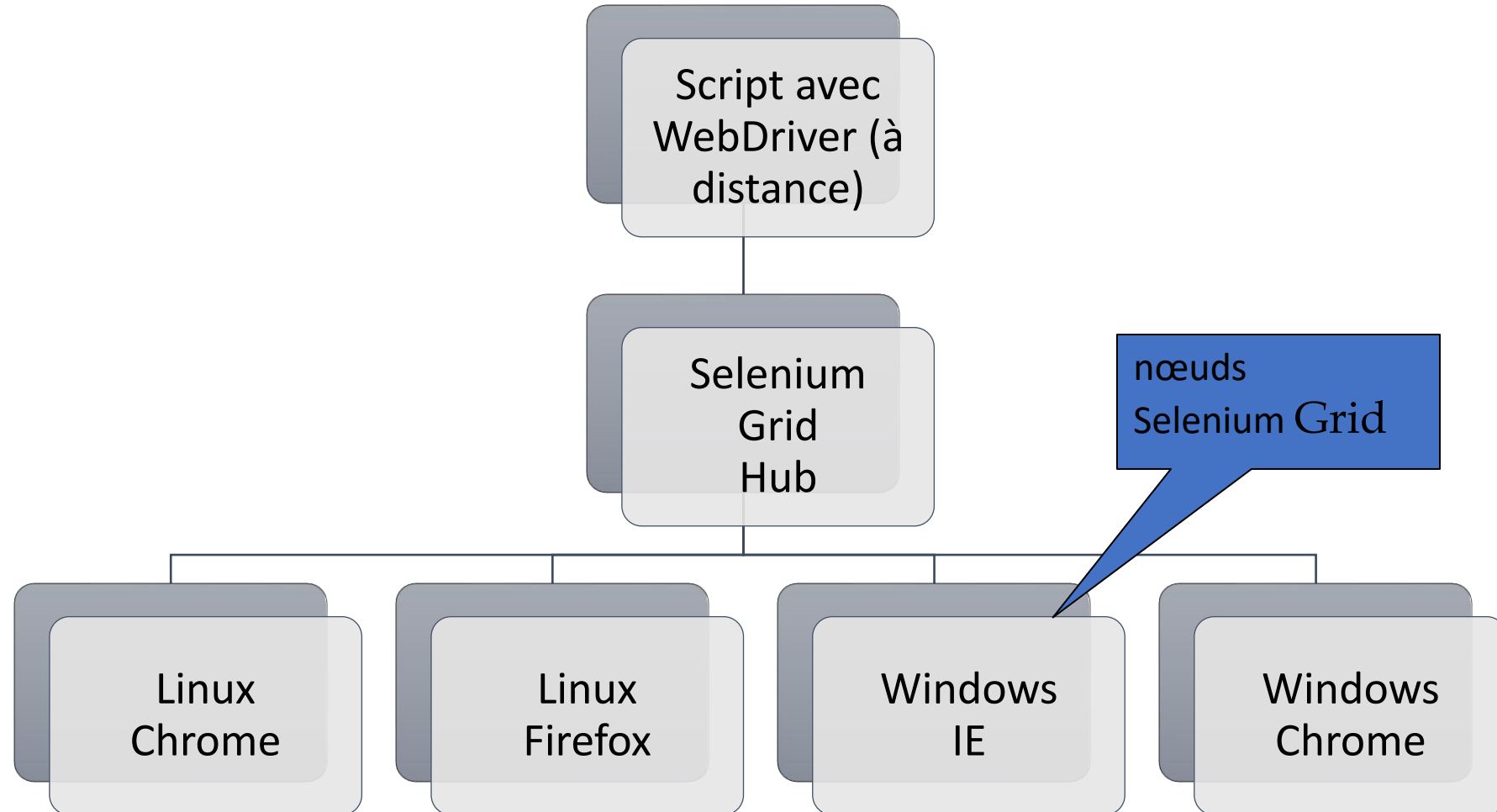
- 🔗 Le code ci-dessous utilise une instance déjà ouverte de navigateur et:
  - 🔗 Entre l'URL, "RBCS-US.com"
  - 🔗 Clique sur le bouton "search", qui fait apparaître la boîte de recherche
  - 🔗 Vide le champ de recherche de tout texte
  - 🔗 Insère "automation" dans la boîte de recherche
  - 🔗 Clique sur le bouton "search" pour lancer la recherche

```
def testUntitledTestCase():  
    driver.get("https://rbc-us.com/")  
    driver.find_element_by_xpath("//body[@id='home']").click()  
    driver.find_element_by_name("q").clear()  
    driver.find_element_by_name("q").send_keys("automation")  
    driver.find_element_by_xpath("//button[@type='submit']").click()
```

# Selenium Grid

- Permet d'exécuter des scripts de test sur différentes machines et configurations
- Permet une exécution distribuée et simultanée
- Selenium Grid est très flexible
- Selenium Grid consiste en un hub et différents nœuds
- Le hub contrôle quel test est exécuté sur quel nœud
- Les nœuds d'une même grille peuvent avoir différents OSes, différents types de navigateurs et différentes versions

# Exemple – Selenium Grid

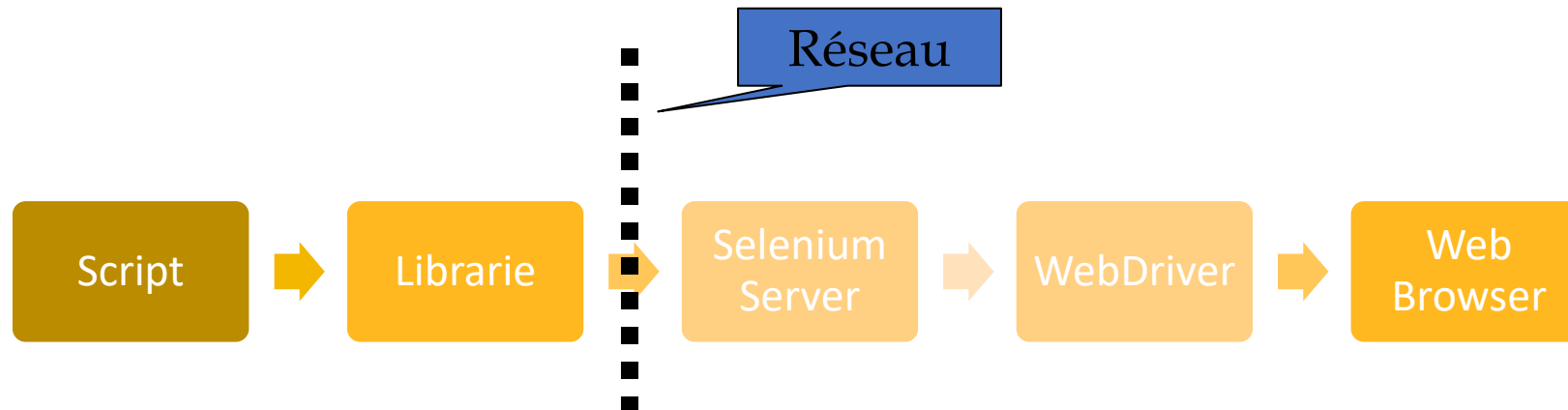




# Selenium Standalone Server

- Implémente des hubs et des nœuds pour Selenium Grid
- Permet une exécution des tests à distance, même sans Selenium Grid
- Ecrit en Java, distribué sous forme de .jar
- Doit être exécuté séparément et configuré correctement (en dehors des scripts de test)

# Selenium Standalone Server : où ?



Selenium Standalone Server permet d'exécuter des tests sur un navigateur tournant sur une machine différente

# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 2

Technologies Internet pour  
automatiser les tests  
d'applications Web

# Termes à mémoriser

CSS selector / Sélecteur CSS

HTML

Tag / balise

XML

XPath

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 2

Technologies Internet pour automatiser les tests d'applications Web

Section 1 :

Comprendre HTML et XML

# Objectif d'apprentissage

STF-2.1 (K3) comprendre et pouvoir écrire des documents HTML et XML

# HTML

- HyperText Markup Language
  - Ecrit de manière textuelle avec des balises
  - Les balises indiquent au navigateur comment afficher le document
- En 2014, HTML 5 a été mis en œuvre pour remplacer 4.01

```
<!DOCTYPE html>
<html>
  <head>
    <title> Page Title</title>
  </head>
  <body>
    <h1>This is a Heading</h1>
    <p>This is a paragraph</p>
  </body>
</html>
```

**Le code ci-dessus donnera:**

**This is a Heading**

This is a paragraph



# Balises HTML

- Les éléments de HTML disposent souvent d'une balise de début et d'une balise de fin
- Les balises ouvrantes s'écrivent entre crochets <>
- La balise de fin comporte un "/"

```
<p>Paragraph text</p>
```

- Les éléments vides peuvent adopter la forme auto-fermante

```
<br />
```

- Un navigateur plus "laxiste" accepterait aussi

```
<br>
```

# Balises HTML de base

Tag	Used for
<code>&lt;html&gt; ... &lt;/html&gt;</code>	Racine du document HTML
<code>&lt;!DOCTYPE&gt;</code>	Définit le type de document (pas nécessaire avec HTML 5)
<code>&lt;head&gt; ... &lt;/head&gt;</code>	Définition et méta donnée du document
<code>&lt;body&gt; ... &lt;/body&gt;</code>	Contenu principal du document
<code>&lt;h1&gt;... &lt;/h1&gt;</code>	Titre (défini de <code>&lt;h1&gt;</code> à <code>&lt;h6&gt;</code> )
<code>&lt;p&gt; ... &lt;/p&gt;</code>	Paragraphe
<code>&lt;br /&gt;</code>	Simple retour de ligne
<code>&lt;div&gt; ... &lt;/div&gt;</code>	Section dans le document
<code>&lt;-- ... --&gt;</code>	Commentaire (peut aller sur plusieurs lignes)
<code>&lt;a href="URL"&gt;link text&lt;/a&gt;</code>	Hyperlien (cliquable)
<code>&lt;img src="X.jpg" alt="XYZ" /&gt;</code>	Image à placer dans le document

# Listes et Tables

Tag	Used for
<code>&lt;ul&gt; ... &lt;/ul&gt;</code>	Liste non ordonnée (puces)
<code>&lt;ol&gt; ... &lt;/ol&gt;</code>	Liste ordonnée (numérotée)
<code>&lt;li&gt; ... &lt;/li&gt;</code>	Élément d'une liste (pour <code>&lt;ul&gt;</code> ou <code>&lt;ol&gt;</code> )
<code>&lt;table&gt; ... &lt;/table&gt;</code>	Tableau HTML
<code>&lt;tr&gt; ... &lt;/tr&gt;</code>	Ligne du tableau
<code>&lt;th&gt; ... &lt;/th&gt;</code>	Titre de colonne
<code>&lt;td&gt; ... &lt;/td&gt;</code>	Donnée dans une cellule du tableau
<code>&lt;tbody&gt; ... &lt;/tbody&gt;</code>	Regroupement de tableaux
<code>&lt;thead&gt; ... &lt;/thead&gt;</code>	En tête de tableau
<code>&lt;tfoot&gt; ... &lt;/tfoot&gt;</code>	Pied de page de tableau
<code>&lt;colgroup&gt; ... &lt;/colgroup&gt;</code>	Groupe de colonnes pour le formatage

# Formulaires HTML

Tag	Used for
<code>&lt;form&gt; ... &lt;/form&gt;</code>	Formulaire HTML pour entrée de données
<code>&lt;input&gt; ... &lt;/input&gt;</code>	Champ de saisie de données
<code>&lt;text area&gt; ... &lt;/textarea&gt;</code>	Saisie de texte sur plusieurs lignes
<code>&lt;button&gt; ... &lt;/button&gt;</code>	Bouton cliquable
<code>&lt;select&gt; ... &lt;/select&gt;</code>	Liste déroulante
<code>&lt;option&gt; ... &lt;/option&gt;</code>	Option dans une liste déroulante
<code>&lt;optgroup&gt; ... &lt;/optgroup&gt;</code>	Groupement d'options dans une liste déroulante
<code>&lt;fieldset&gt; ... &lt;/fieldset&gt;</code>	Groupement d'éléments dans un formulaire
<code>&lt;label&gt; ... &lt;/label&gt;</code>	Nom d'un élément de saisie

# XML

- eXtensible Markup Language
- Langage de balise définissant un ensemble de règles permettant d'encoder des documents dans un format lisible à la fois par des hommes ou des machines
- Requiert une structure rigoureuse (chaque balise doit être impérativement fermée)
- Le document adopte toujours une structure arborescente

```
<?xml version="1.0" ?>
<note>
  <date>2018-06-12</date>
  <hour>10:30</hour>
  <to>Francis</to>
  <from>Morrow</from>
  <body>Please pick me up this weekend!</body>
</note>
```

# Attributs XML

- Balises formatées comme pour HTML, mais créées selon les besoins
- XML supporte des attributs qui peuvent être ajoutés aux balises pour décrire les éléments

```
<person gender="female">
```

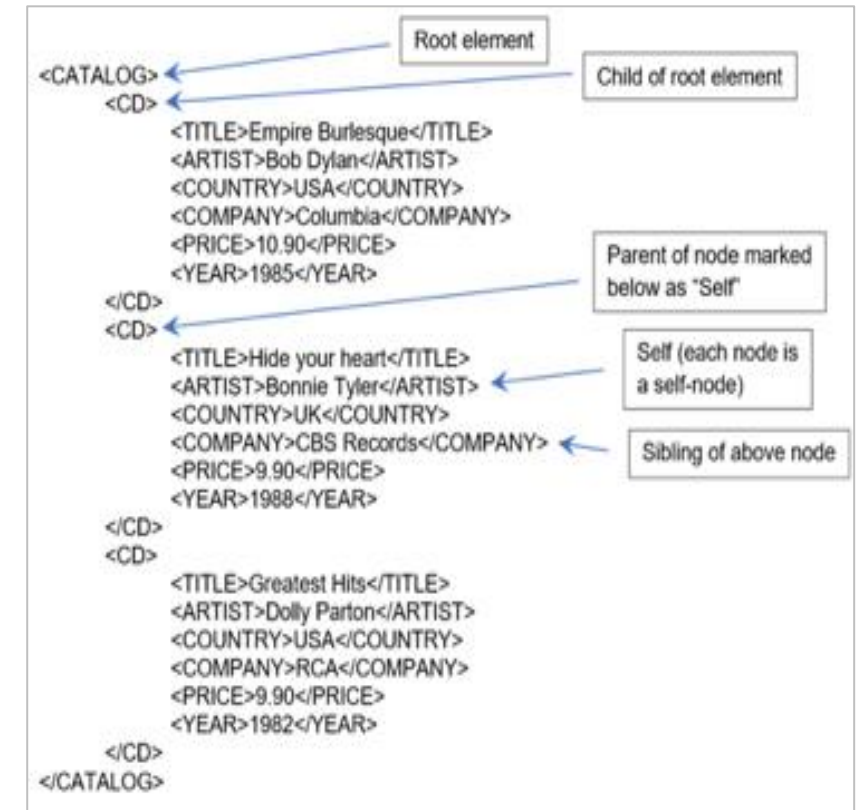
- La même information peut aussi être utilisée comme élément et non plus comme attribut
- Les attributs sont moins flexibles que les éléments
  - Ils ne peuvent contenir plusieurs éléments
  - Ils ne peuvent contenir de structure arborescente
  - Ils sont moins aisément extensibles

```
<person gender="female">  
  <firstname>Anna</firstname>  
  <lastname>Smith</lastname>  
</person>
```

```
<person>  
  <gender>female</gender>  
  <firstname>Anna</firstname>  
  <lastname>Smith</lastname>  
</person>
```

# XML et la structure arborescente

- Le nœud **courant (current)** est un nœud choisi arbitrairement.
  - Toutes les autres relations dérivent du nœud courant
  - Dans l'exemple ci-contre "Bonnie Tyler" a été choisi comme nœud courant
  - → Toute relation se rapporte à ce nœud
- Chaque nœud peut s'identifier lui-même en tant que **self**
- Un nœud **parent** est situé un niveau au-dessus du nœud courant
- Chaque élément ou attribut de nœud a exactement un parent
- La racine n'a pas de parent
- Un nœud fils (**child**) est toujours situé un niveau sous son parent dans la hiérarchie
- Un nœud frère (**sibling**) est au même niveau que le nœud courant
- Un nœud frère à le même parent que le nœud courant
- Un nœud ancêtre (**ancestor**) est situé en ligne directe au-dessus du nœud courant (parent, grand parent, arrière grand parent...)
- Un nœud descendant (**descendent**) est situé en ligne directe en-dessous du nœud courant (fils, petit fils...)



# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin



# Exercice : un document HTML

1. Ouvrir la page web ***https://www.python.org***
2. Dans le code source de la page, trouver un bouton de ***recherche / search***  
Naviguer dans l'arborescence de la page HTML jusqu'à la balise ***div*** ou ***form*** contenant ce bouton
3. Trouver la balise ancêtre la plus proche de l'élément trouvé à l'étape 2 et possédant un attribut ***ID***

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 2

Technologies Internet pour automatiser les tests d'applications Web

Section 2:

XPath et recherche dans les documents XML

# Objectif d'apprentissage

STF-2.2 (K3) Appliquer XPath pour chercher dans les documents XML

# Expressions XPath

- ✚ Les expressions de navigation (Path) permettent à l'automaticien de trouver les nœuds spécifiques où agir
- ✚ Les expressions sont similaires à celles permettant de naviguer dans la structure de répertoire d'un ordinateur

Expression	Description
<b>TheNode</b>	Sélectionne tous les éléments portant le nom "TheNode"
<b>/</b>	Sélectionne à partir de la racine
<b>//</b>	Renvoie les descendants de l'élément courant
<b>.</b>	Renvoie l'élément courant
<b>..</b>	Sélectionne le parent de l'élément courant
<b>@</b>	Sélectionne un attribut de l'élément courant

# Sélectionner les nœuds

- Des nœuds spécifiques peuvent être sélectionnés au moyen de ces expressions

```
<?xml version="1.0" encoding="UTF-8"?>
<Magazines>
  <Magazine>
    <title lang="en">Time</title>
    <price>9.99</price>
  </Magazine>
  <Magazine>
    <title lang="en">Newsweek</title>
    <price>8.95</price>
  </Magazine>
</Magazines>
```

Expression	Résultat
Magazines	Sélectionne tous les nœuds de nom "Magazines"
/Magazines	Sélectionne l'élément racine "Magazines"
Magazines/Magazine	Sélectionne tous les éléments de "Magazine" dans "Magazines"
//Magazine	Sélectionne tous les éléments "Magazine" dans le document
Magazines//Magazine	Sélectionne tous les éléments "Magazine" descendants de "Magazines"
//@lang	Sélectionne tous les attributs "lang"

# Wildcards (jokers) dans XPath

- Des Wildcards peuvent sélectionner des nœuds XML sur base de critères spécifiques

Wildcard	Description
*	Correspond a tout élément de nœud
@*	Correspond à tout élément d'attribut
<b>Node()</b>	Correspond à tout type de nœud

# Prédicats

- Les prédicats sont des conditions que les éléments sélectionnés doivent rencontrer
- Ils sont entourés de crochets [] et suivent le nom de l'élément

```
<?xml version="1.0" encoding="UTF-8"?>
<Magazines>
  <Magazine>
    <title lang="en">Time</title>
    <price>9.99</price>
  </Magazine>
  <Magazine>
    <title lang="en">Newsweek</title>
    <price>8.95</price>
  </Magazine>
</Magazines>
```

Expression	Résultat
<code>/Magazines/Magazine[1]</code>	Sélectionne le premier élément de Magazine
<code>/Magazines/Magazine[last()]</code>	Sélectionne le dernier élément fils de Magazine
<code>/Magazines/Magazine[last()-1]</code>	Sélectionne l'avant-dernier élément de Magazine
<code>//title[@lang]</code>	Sélectionne tous les éléments de titre avec l'attribut "lang"
<code>//title[@lang='en']</code>	Sélectionne tous les éléments de titre avec l'attribut lang=en
<code>*[@price &gt; 35]</code>	Elément avec attribut <i>price</i> plus grand que 35

# Operateurs dans les prédicats

Operateur	Description	Exemple
	Sélection de chemins multiples	//Magazine   //CD
+	Addition	2 + 2
-	Soustraction	5 – 3
*	Multiplication	8 * 8
div	Division	14 div 2
mod	Modulus (reste de la division)	7 mod 3
=	Egal	price=4.35
!=	Pas égal / différent	price!=4.35
<	Plus petit que	price<4.35
<=	Plus petit ou égal à	price<=4.35
>	Plus grand que	price>4.35
>=	Plus grand ou égal à	price>=4.35
or	Ou	price>3.00 or lang="en"
and	Et	price>3.00 and lang="en"
not	Inverse	not lang="en"
	Concaténation de chaîne de caractères	"en"    "glish"



# Fonctions de chaînes utiles

Nom	Description
<b>string(arg)</b>	Retourne la valeur de chaîne de l'argument
<b>substring(str, start, len)</b>	Retourne une partie de la chaîne de longueur "len" en commençant à "start"
<b>string-length(str)</b>	Retourne la longueur de chaîne. En l'absence d'argument, retourne la longueur du nœud courant
<b>compare(str1, str2)</b>	Retourne -1 si str1 < str2, 0 si les chaînes sont égales, +1 si str1 > str2
<b>concat(str1, str2, ...)</b>	Retourne une concaténation de toutes les chaînes introduites
<b>upper-case(str)</b>	Convertit l'argument de la chaîne en majuscules
<b>lower-case(str)</b>	Convertit l'argument de la chaîne en minuscules
<b>contains(str1, str2)</b>	Retourne TRUE si str1 contient str2
<b>starts-with(str1, str2)</b>	Retourne TRUE si str1 commence par str2
<b>ends-with(str1, str2)</b>	Retourne TRUE si str1 finit par str2

Ex : `//Magazine/title[contains(@lang, 'e')]` → tous les **title**, fils de **Magazine** dont la valeur d'attribut **lang** contient un 'e'

# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Exercice sur XPath

- Ouvrir un navigateur Chrome
- Aller à la page:  
*<http://automationpractice.com>*
- Ecrire des localisateurs XPath afin de trouver les objets suivants sur la page (et uniquement ceux-ci):
  - Boîte de saisie Search
  - Lien “Sign in”
  - Le bouton “Add to cart” pour un produit indiqué par le formateur

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 2

Technologies Internet pour automatiser les tests d'applications Web

Section 3:

Localisateurs CSS

# Objectifs d'apprentissage

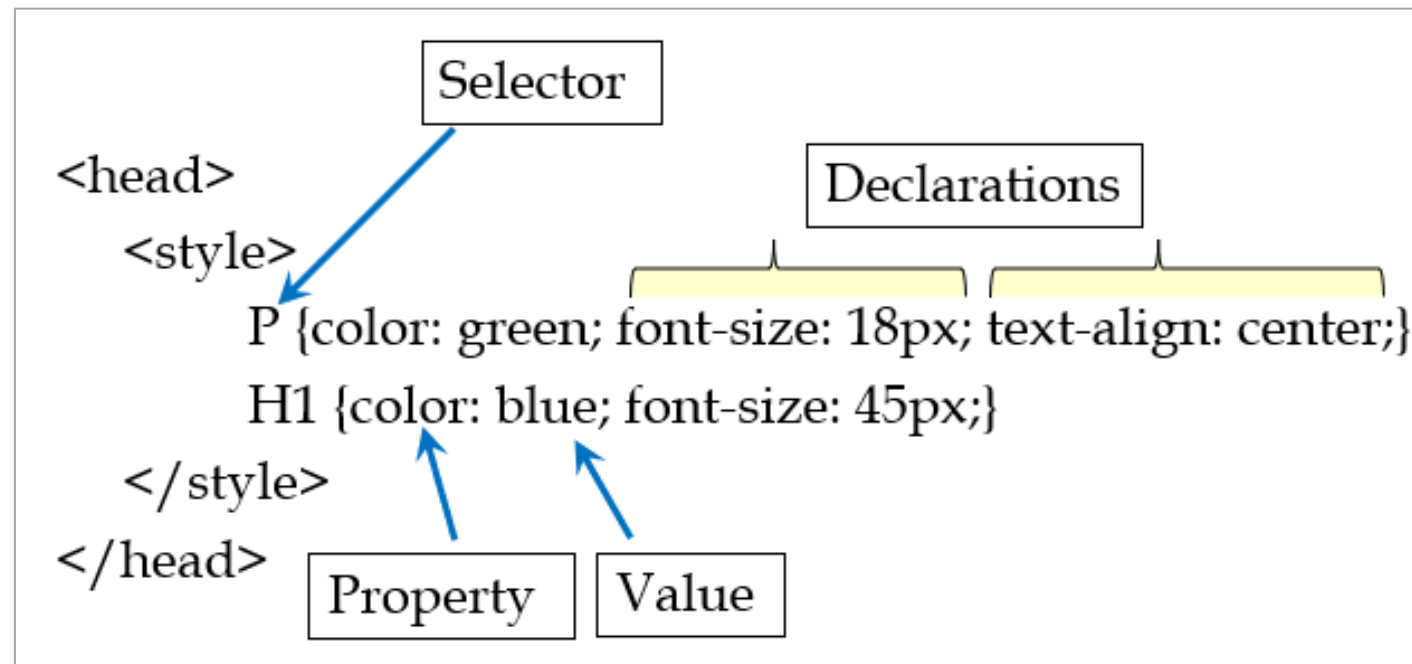
STF-3.3 (K3) Appliquer les localisateurs CSS pour trouver des éléments dans des documents HTML

# Cascading Style Sheets

- CSS est utilisé pour définir comment les éléments HTML doivent être affichés sur un écran, une page ou un autre média
- CSS peut être utilisé de 3 manières pour les documents HTML
  - Une feuille de style externe, séparée des pages HTML
  - Une feuille de style interne, au sein du <head> du fichier HTML
  - “en ligne”, ajoutée directement en tant qu’attribut d’un élément
- Quand plusieurs fichiers CSS sont présents, seul le dernier est appliqué
- CSS n’est pas considéré comme un langage de programmation, c’est plutôt un langage de feuille de styles
- Un jeu de règles CSS consiste en un ensemble de sélecteurs et de blocs de déclaration (voir diapo suivante)

# Un ensemble de règles CSS

- ✚ Chaque sélecteur pointe vers un élément HTML dont on veut définir le style
- ✚ Un bloc de déclaration contient une ou plusieurs déclarations se terminant par “;”



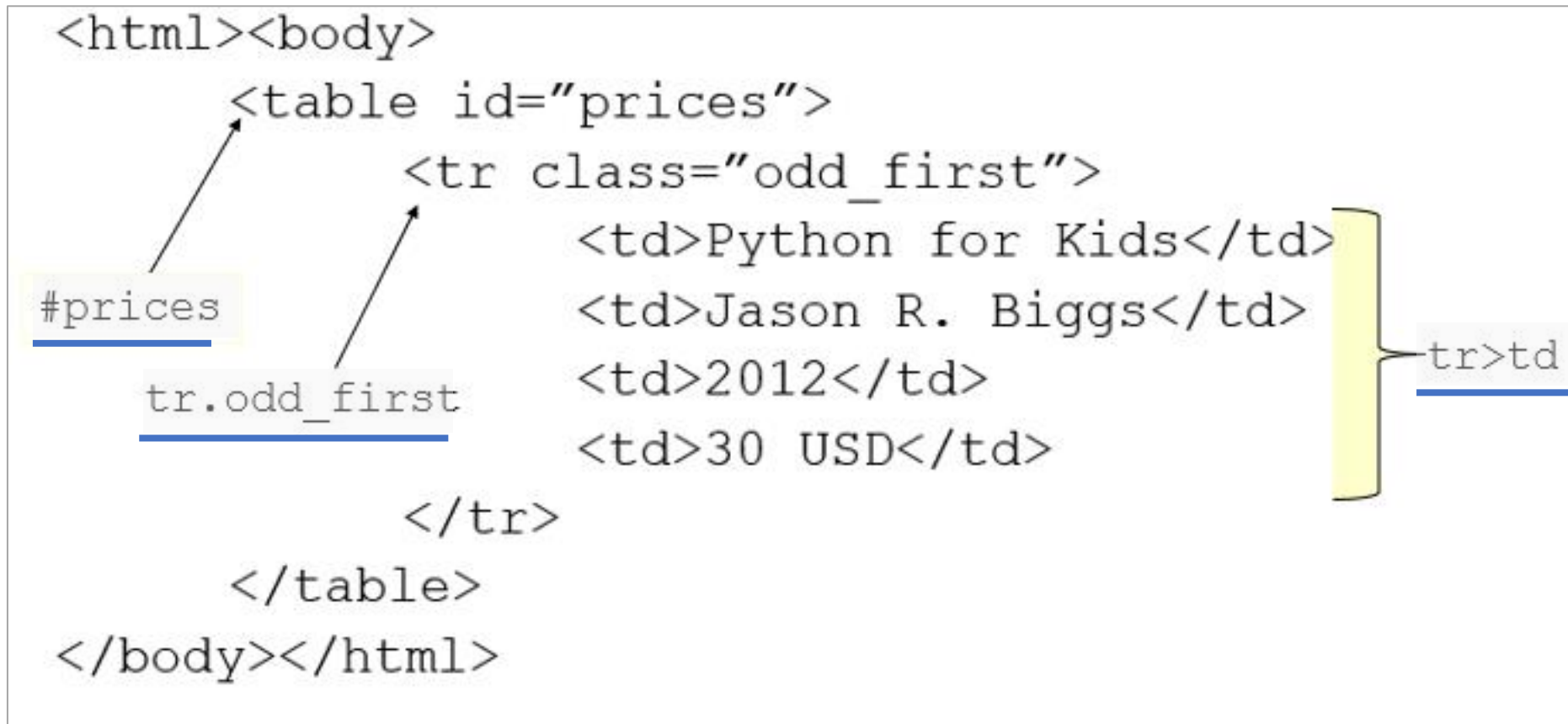
# CSS vs. SélecteursXPath

- 🔗 XPath et CSS peuvent tous deux être utilisés pour sélectionner des éléments

CSS	XPath	Résultat
<b>div.even</b>	<code>//div[@class="even"]</code>	Éléments div avec une classe d'attribut ="even"
<b>#login</b>	<code>//*[@id="login"]</code>	Un élément avec id="login"
<b>*</b>	<code>//*</code>	Tous les éléments
<b>input</b>	<code>//input</code>	Tous les éléments input
<b>p,div</b>	<code>//p //div</code>	Tous les éléments p et div
<b>div input</b>	<code>//div//input</code>	Tous les éléments input contenus dans l'ensemble des éléments div
<b>div &gt; input</b>	<code>//div/input</code>	Tous les éléments input qui ont l'élément div comme parent
<b>br + p</b>		Sélectionne tous les éléments p placés directement après l'élément br
<b>p ~ br</b>		Sélectionne tous les éléments p placés directement avant l'élément br



# Sélecteur CSS: Exemple



# Sélecteurs CSS – Utilisation des Attributs

CSS	Résultat
<b>[lang]</b>	Tous les éléments avec l'attribut lang
<b>[lang=en]</b>	Tous les éléments avec l'attribut lang ayant exactement la chaîne 'en'
<b>[lang^=en]</b>	Tous les éléments avec l'attribut lang commençant par la chaîne en
<b>[lang   =en]</b>	Tous les éléments avec l'attribut lang égal à la chaîne 'en' ou commençant par la chaîne 'en', et suivie d'un tiret
<b>[lang\$=en]</b>	Tous les éléments avec l'attribut lang se terminant par la chaîne 'en'
<b>[lang~=en]</b>	Tous les éléments avec l'attribut lang dont la valeur est une liste de mots séparés par des blancs et dont l'un est exactement la chaîne 'en'
<b>[lang*=en]</b>	Tous les éléments avec l'attribut lang contenant la chaîne 'en'

# Sélecteurs CSS avec attributs: Exemple

- Les nœuds peuvent être sélectionnés de diverses manières en utilisant les attributs

```
<html><body>
  <table id="price-list">
    [id=price-list] <tr class="odd_first">
    [id|=price]      <td>Python for Kids</td>
                    <td>Jason R. Biggs</td>
                    <td>2012</td>
                    <td>30 USD</td>
                    </tr>
  </table>
</body></html>
```

[class~=odd\_first]  
[class\*=rs]  
[class^=od]  
[class\$=st]

# Sélecteurs CSS pour éléments de formulaires

CSS	Résultat
<b>:checked</b>	Sélectionne tous les éléments choisis (checked) (dans les check boxes, boutons radio, et options activées (état on))
<b>:default</b>	Sélectionne un élément de formulaire qui correspond à la valeur par défaut dans un groupe d'éléments associés
<b>:defined</b>	Sélectionne tous les éléments qui ont été définis
<b>:disabled</b>	Sélectionne les éléments inactifs
<b>:enabled</b>	Sélectionne les éléments actifs
<b>:focus</b>	Sélectionne les éléments qui ont le focus
<b>:invalid</b>	Sélectionne tout élément qui n'a pu être validé
<b>:optional</b>	Sélectionne les éléments qui n'ont pas l'attribut obligatoire
<b>:out-of-range</b>	Sélectionne tout élément entré dont la valeur actuelle se trouve hors des attributs min et max fixés
<b>:read-only</b>	Sélectionne les éléments non éditables par l'utilisateur
<b>:read-write</b>	Sélectionne les éléments éditables par l'utilisateur
<b>:required</b>	Sélectionne les éléments qui ont l'attribut obligatoire /requis
<b>:valid</b>	Sélectionne tout élément qui a pu être validé
<b>:visited</b>	Sélectionne un lien déjà visité par l'utilisateur

# Sélecteurs CSS utiles pour l'automatisation

CSS	Résultat
<b>:not(&lt;selector&gt;)</b>	Sélectionne les éléments qui ne correspondent pas au sélecteur spécifié
<b>:first-child</b>	Sélectionne tous les éléments qui sont les premiers enfants de leurs éléments parents
<b>:last-child</b>	Sélectionne tous les éléments qui sont les derniers enfants de leurs éléments parents
<b>:nth-child(&lt;n&gt;)</b>	Sélectionne tous les éléments qui sont les n <sup>ières</sup> enfants de leurs éléments parents
<b>:nth-last-child(&lt;n&gt;)</b>	Sélectionne tous les éléments qui sont les n <sup>ières</sup> enfants de leurs éléments parents en comptant à partir du dernier enfant
<b>div:nth-of-type(&lt;n&gt;)</b>	Sélectionne tous les éléments qui sont les n <sup>ières</sup> div enfants de leurs éléments parents

# Exercice: les sélecteurs CSS

- Ouvrir un navigateur Chrome
- Aller à la page:

*<http://automationpractice.com>*

- Ecrire des sélecteurs CSS afin de trouver les objets suivants sur la page (et uniquement ceux-ci):
  - Boîte de saisie Search
  - Lien “Sign in”
  - Le bouton “Add to cart” d’un produit

# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 3

Utiliser le WebDriver de  
Selenium



# Terminologie à retenir

class attribute

DOM (Document Object Model)

fonction

ID

iframe

Dialogue modal (modal dialog)

pytest

WebDriver

wrapper

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 3

Utiliser le WebDriver de Selenium

Section 1:

Mécanismes de suivi et de Reporting

# Objectif d'apprentissage

**STF-3.1 (K3) Utiliser les mécanismes appropriés de suivi et de Reporting**

# Exécution de scripts

**L'automatisation des tests simule un testeur humain exécutant un test via le clavier et la souris**

- Un testeur exécutant un test a une assez bonne idée de ce qui s'est passé
- Un script automatisé repose sur le suivi (logging) pour comprendre ce qui s'est passé
  - Il peut être programmé « from scratch »
  - Alternativement, les cadres existants peuvent être utilisés pour exécuter des tests et gérer le log (ex : “unittest” ou “pytest”)

Pour ce syllabus, nous avons choisi de travailler avec pytest parce qu'il fonctionne très bien avec le WebDriver de Selenium

# Exécution de tests avec "pytest"

- "pytest" exécute tous les cas de test dans le répertoire en cours et les sous-répertoires enfants en dessous (VOIR LE PPT « INSTALLER PYTHON »)
- Il recherche les fichiers ***test\_\*.py*** ou ***\*\_test.py***
- Le comportement de l'exécution dépend des « flags »
  - ***pytest*** : (no flag) exécute tous les tests
  - ***pytest -v*** : mode « verbose » affichant les noms complets
  - ***pytest -q*** : version silencieuse (quiet), affiche moins d'informations
  - ***pytest -s*** : pour voir les print en sortie
  - ***pytest --html=report.html*** : exécute le test avec un rapport sous forme de fichier html (si le module ***pytest-html*** est installé)

# Marques

Les marques peuvent être utilisées pour appliquer des métadonnées aux fonctions de test  
→ (*import pytest* en début de script)

- Pour toujours ignorer un test: Utiliser ***skip***

***@pytest.mark.skip***

- Pour sauter si une certaine condition est remplie: utilisez ***skipif***

***@pytest.mark.skipif(condition)***

- Pour produire un échec prévu, utilisez ***xfail***

***@pytest.mark.xfail***

- Pour effectuer plusieurs appels, utilisez ***parameterize***

***@pytest.mark.parameterize(argnames)***

# Informations générales de suivi (log)

- L'automatisation de GUI tend à échouer plus souvent que le test manuel
- Lorsqu'un script échoue, l'analyste doit déterminer:
  - S'il s'agit d'un faux positif.
  - ... ou s'il s'agit d'une véritable défaillance du SUT qui doit faire l'objet d'une enquête?
- Sans un bon suivi (log), la cause d'un échec peut être délicate à trouver.
- Différentes informations peuvent être enregistrées
  - Toutes les étapes du test (utile pour le débogage)
  - Avertissements lorsque des résultats inattendus se produisent (p. ex. temporisation lente)
  - Échecs réels

# Bibliothèque de suivi python

Chaque message de suivi de Python dans la bibliothèque possède un niveau de sévérité (du plus bas au plus haut)

- **debug**: pour diagnostiquer des problèmes
- **info**: pour vérifier que les choses fonctionnent
- **warning**: quelque chose d'inattendu s'est produit, problème potentiel survenu
- **error**: un grave problème s'est produit
- **critical**: un problème critique s'est produit

```
import logging
logging.basicConfig(level=logging.WARNING)
# default logging level is WARNING

logging.info("Hello world.")
logging.warning("Title: %d Dalmatians" % 101)
logging.debug("Title: %s" % "101 Dalmatians")
```

## Exemple de code utilisé pour enregistrer un événement

*Notez qu'en raison du niveau de journalisation par défaut, les messages **info** et **debug** ne s'affichent pas dans le journal*



# Assertions

- Lors de l'exécution d'un test, les résultats attendus doivent être vérifiés
- WebDriver peut utiliser des assertions pour vérifier les résultats
- Si l'assertion est évaluée comme fausse au moment de l'exécution, le script lève une exception

```
assert variable == 13, "Variable should equal 13"
```

# Une étape automatisée

Une étape dans un cas de test automatisé imite les actions d'un testeur manuel

- Avec WebDriver, le découpage classique d'un pas de test serait :
  1. Localiser un élément web sur la page
  2. Interagir avec l'élément
  3. S'assurer du résultat attendu
- Notez qu'un échec peut se produire à tout instant
  1. Si l'élément est introuvable ou est dans un état inattendu, une exception sera levée
  2. Si la tentative d'agir sur l'élément échoue, une exception est levée
  3. On peut utiliser l'assertion pour vérifier que l'action attendue s'est produite

# Reporting

- Le reporting n'est pas la même chose que le suivi (logging)
  - Le **Logging** fournit des informations à l'analyste de test qui a exécuté un script de test quant à ce qui s'est produit pendant l'exécution
  - Le **Reporting** fournit cette information — et d'autres — aux intervenants du projet
- Les automaticiens doivent comprendre
  - Qui sont les parties prenantes
  - Quelle information ces derniers veulent avoir (besoin)
  - Comment ils veulent que les informations leurs soient fournies
- Les différents intervenants voudront des informations différentes
- Dans la mesure du possible, les rapports doivent être groupés et livrés automatiquement

# Contenu du rapport

- Résumé de ce qui a été testé
- Environnements et configurations testés
- Ce qui s'est produit lors de la récente exécution
- Les tendances qui sont significatives
- Faux positifs et autres questions
- D'autres métriques comme convenu avec chaque intervenant

# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Exercice en utilisant "pytest"

Écrivez un script Python qui:

- Sera exécuté avec pytest Framework et produira un rapport HTML
- Se compose d'un cas de test
- Enregistre l'heure actuelle \*
- Évalue si l'heure actuelle est avant midi et échoue si elle est postérieure

\* Utiliser la fonction *time.localtime()*

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 3

Utiliser le WebDriver de Selenium

Section 2:

Naviguer vers différentes URLs

# Objectif d'apprentissage







STF-3.2 (K3) Accédez à différentes URLs à l'aide des commandes de WebDriver



# Différents pilotes de navigateur

- Différents navigateurs peuvent avoir des comportements différents
- La plupart des sites Web doivent être testés avec différents navigateurs
- Chaque navigateur est susceptible d'avoir besoin d'un pilote (driver) spécial
- Chaque pilote doit être installé sur la station de test pour permettre aux scripts de test d'accéder aux navigateurs

## Exemples:

-  Firefox (geckodriver.exe)
-  Internet Explorer (IEDriverServer.exe)
-  Safari (safaridriver)
-  Edge (MicrosoftWebDriver.msi)
-  Chrome (chromedriver.exe)
-  HtmlUnit (HtmlUnit driver)

# L'objet pilote

- Le navigateur Web est l'objet principal à manipuler pour le test
- Le pilote crée l'interface de programmation entre le script et le navigateur Web
- Une variété de fonctions et de propriétés deviennent disponibles une fois que l'objet pilote est instancié

# Ouverture d'un écran de navigateur

- Instanciez d'abord un pilote de navigateur

```
from selenium import webdriver  
driver = webdriver.Chrome()
```

- Entrez l'URL souhaitée

```
driver.get('https://www.python.org')
```

- On peut s'assurer que la page correcte a été ouverte

```
assert driver.current_url == 'https://www.python.org/', ErrMsg  
assert driver.title == 'Welcome to Python.org', ErrMsg
```

# Naviguer avec le navigateur

- Retour arrière dans l'historique

```
driver.back()  
driver.execute_script("window.history.go(-1) ")
```

- Avancer dans l'historique

```
driver.forward()  
driver.execute_script("window.history.go(+1) ")
```

- Actualiser la page

```
driver.refresh()
```

# Fermeture du navigateur

- Fermer la fenêtre active (TAB)

```
driver.close()
```

*(Lorsque la dernière fenêtre est fermée, le navigateur se ferme)*

- Fermer le navigateur et le pilote en même temps

```
driver.quit()
```

# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Exercice: navigation

- Concevoir un cas de test qui vérifie la disponibilité en ligne de la documentation Python
- N'oubliez pas d'inclure les assertions appropriées pour vérifier les résultats attendus du test.

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 3

Utiliser le WebDriver de Selenium

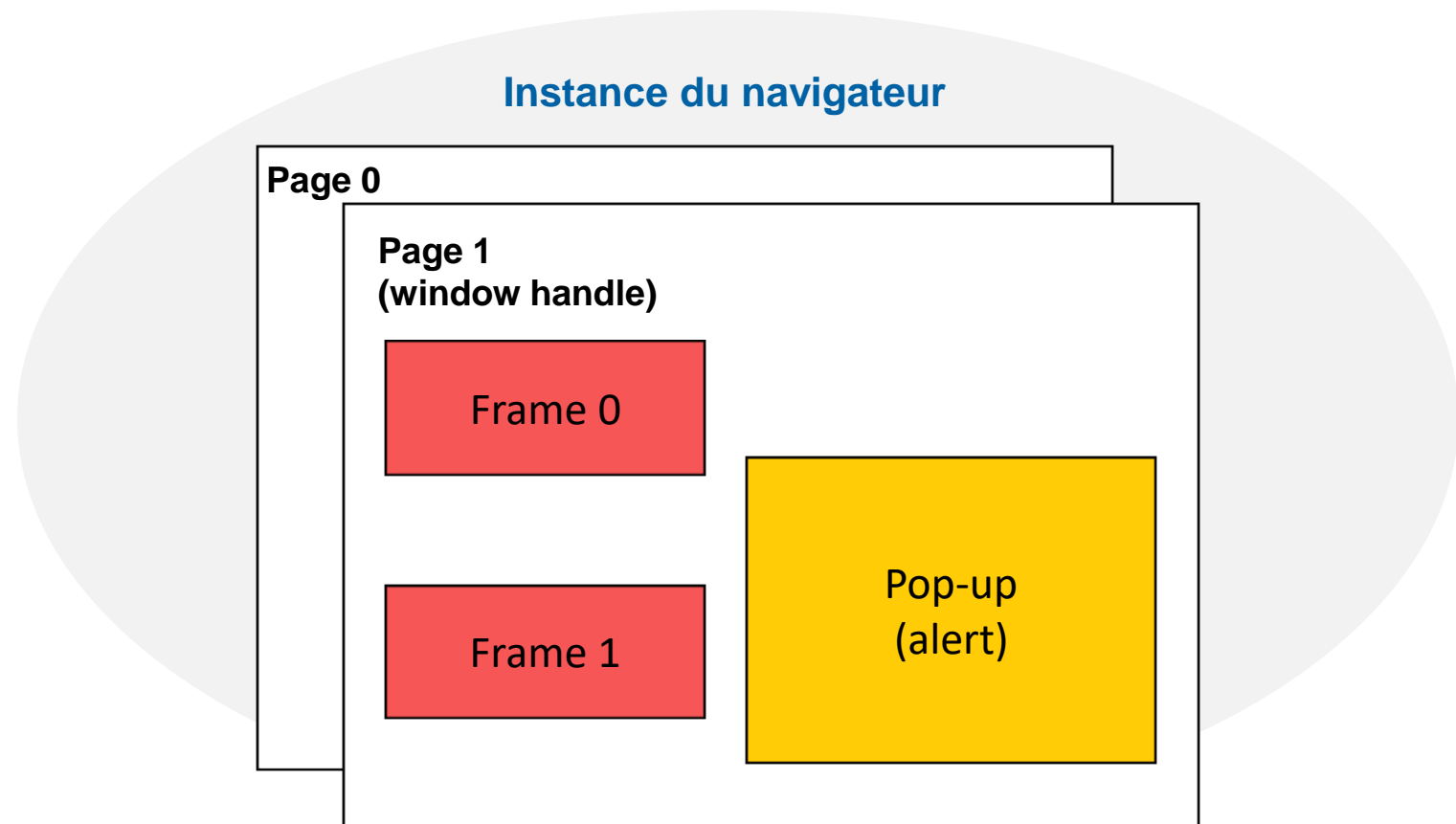
Section 3:

Modifier le contexte de la fenêtre



# Objectif d'apprentissage

STF-3.3 (K3) Modifier le contexte de la fenêtre dans les navigateurs web à l'aide des commandes de WebDriver



# Modification des contextes

- La modification du contexte de script peut se faire de deux façons:
  - Modification des navigateurs
  - Modification des fenêtres (onglets) dans un navigateur
- Le navigateur web n'a pas besoin d'avoir le focus pour être en mesure d'exécuter les commandes du WebDriver de Selenium
- Cela permet d'exécuter plusieurs tests simultanément

## 1- Ouvrir le premier navigateur

```
dr1 = webdriver.Chrome()  
dr1.get("https://www.python.org")
```

## 2- Ouvrir le deuxième navigateur

```
dr2 = webdriver.Chrome()  
dr2.get("https://www.ruby-lang.org")
```

## 3- Enregistrer les titres (pour prouver qu'ils sont ouverts)

```
log.info("Python site: " + dr1.title)  
log.info("Ruby site: " + dr2.title)
```

## 4- Fermez les deux navigateurs et détruisez les objets WebDriver

```
dr1.quit()  
dr2.quit()
```

# Ouverture de plusieurs onglets dans le navigateur

- L'ouverture de plusieurs onglets dans un navigateur est possible
- Différents systèmes d'exploitation peuvent avoir différentes techniques pour l'ouverture de plusieurs onglets
- Une couverture complète dépasse le cadre de ce cours
- Une façon de faire est d'exécuter du code JavaScript

```
driver.execute_script("$ (window.open(' ')) ")
```

# Basculement entre les onglets d'un navigateur

- Tout d'abord, identifiez les "handles" de tous les onglets ouverts
- Cette propriété contient un tableau de « handles » pour ouvrir des onglets

```
driver.window_handles
```

## Exemple de retour

```
['CDwindow-2628131AAB86DE6B665B3181A8DC5075',  
'CDwindow-7EDD0B7A27F703DA019AA9D7A4C0C409',  
'CDwindow-A200AB483F40B1F2D4101508F6C86830']
```

*Les « handles » d'un navigateur web sont en fait des chaînes*

- Changez l'onglet actif en un seul, spécifique

```
driver.switch_to.window(driver.window_handles[1])
```

# Modification des cadres (frames)

- Trouver un cadre et passer à ce dernier

```
fr = driver.find_element_by_id('foo')  
driver.switch_to.frame(fr)
```

- Passer à un cadre don't l'ID est déjà connu

```
driver.switch_to.frame('foo')
```

- Passer à un cadre parent

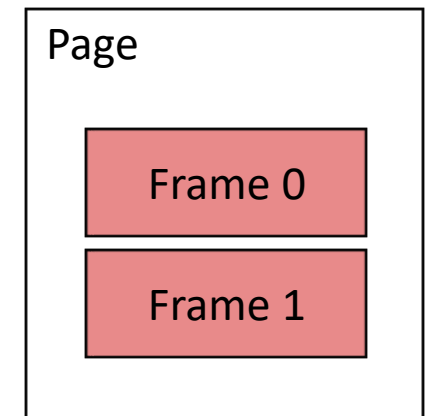
```
driver.switch_to.parent_frame()
```

- Retourner à la page entière (tab)

```
driver.switch_to.default_content()
```



Attention: Le ***switch\_to.frame*** prend comme base le contexte dans lequel il est appelé. Si la frame appelée n'existe pas dans le contexte une exception est lancée.



Depuis le contexte de la Frame 0 on ne pourrait pas appeler la Frame 1

# Modification de la taille du navigateur

- Agrandir la fenêtre du navigateur

```
driver.maximize_window()
```

- Minimiser la fenêtre du navigateur

```
driver.minimize_window()
```

- Afficher le navigateur en plein écran

```
driver.fullscreen_window()
```

# Exercices iframe

## Changer de cadres

- Pour l'URL suivante, chargez le navigateur, basculez vers l'iframe, puis revenez à la page par défaut

*[https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref\\_alert](https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_alert)*

## agrandir la fenêtre

- ajouter une étape au script : après l'ouverture du navigateur qui maximise la fenêtre du navigateur

# Exercice pour deux navigateurs

## Instancier deux navigateur BR1 et BR2

1. BR1: ouvrir python.org
2. BR2: ouvrir <https://www.w3.org/>
3. BR1: faire une vérification sur le titre de la page
4. BR2: faire une vérification sur le titre de la page



# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 3

Utiliser le WebDriver de Selenium

Section 4:

Prendre des captures d'écran de pages Web

# Objectif d'apprentissage

STF-3.4 (K3) Prendre des captures d'écran de pages web à l'aide des commandes de WebDriver

# La valeur des captures d'écran

- Les testeurs manuels interagissent visuellement avec les objets GUI sur l'écran
- Quand un test échoue, ils peuvent voir exactement le problème
- Les testeurs automatisés ne peuvent généralement pas voir la série de tests
- Parfois, il est utile de voir l'état de l'écran
  - Lorsque le test automatisé détecte une défaillance
  - Lors de l'exécution d'un test très visuel où le résultat de réussite/échec dépend du testeur analysant l'écran visuellement
  - Lors de tests de configuration sur plusieurs systèmes

**Une image d'un échec dans le journal peut réduire le temps nécessaire pour résoudre le problème**

# Traitement des captures d'écran

Les automaticiens doivent soigneusement planifier les captures d'écran

Ils doivent éviter d'écraser les captures d'écran précédentes

- Besoin d'un dossier unique pour les stocker
- Besoin d'un algorithme pour les nommer

Deux objectifs différents de captures d'écran

- Prise d'une capture de l'intégralité de l'écran du navigateur
- Prise d'une capture d'un élément unique sur un écran

# Différentes captures d'écran

- Une capture d'écran de l'intégralité de la fenêtre du navigateur

```
driver.get_screenshot_as_file('C:\\temp\\screenshot.png')
```

- Une capture d'un élément individuel de l'écran

```
ele = driver.find_element_by_id('btnLogin')  
ele.screenshot('c:\\temp\\element_screenshot.png')
```

# Autres options de capture d'écran

- Plutôt que de créer un fichier, vous voudrez peut-être créer une chaîne codée en base64 d'une image
- Plus sûr pour streamer (p.ex. vers un enregistrement de base de données)

```
img_b64 = driver.get_screenshot_as_base64()  
img_b64 = element.screenshot_as_base64
```

- Vous pouvez également vouloir simplement créer une représentation binaire d'une image sans la stocker dans un fichier

```
png_str = driver.get_screenshot_as_png()  
png_str = element.screenshot_as_png()
```

# Exercice: captures d'écran

- Ajouter au script du dernier exercice un capture d'écran de chaque page de chaque navigateur



# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 3

Utiliser le WebDriver de Selenium

Section 5:

Localiser les éléments GUI

# Objectif d'apprentissage

STF-3.5 (K4) Localiser les éléments GUI en utilisant diverses stratégies

# Localisation des éléments GUI

- L'utilisation d'un élément GUI à l'aide de WebDriver nécessite que vous le trouviez tout d'abord sur l'écran
- Une variété de fonctions sont disponibles: deux options principales:
  - Pour rechercher un seul objet: *find\_element\_by\_xx*
  - Pour rechercher plusieurs objets: *find\_elements\_by\_xx*
- Localisateurs HTML : trouver un élément...
  - Au moyen de l'ID: *by\_id(id\_)*
  - Au moyen de la classe: *by\_class\_name(name)*
  - Au moyen du nom de la balise : *by\_tag\_name(name)*
  - Au moyen d'un localisateurs XPath: *by\_xpath(xpath)*
  - Au moyen d'un sélecteurs CSS: *by\_css\_selector(css\_selector)*

# Localiser les éléments UI par leur ID

Soit l'élément html

```
<element id="unique_id">
```

Exemple de localisation avec Python WebDriver :

```
element_found = driver.find_element_by_id('unique_id')
```

## Avantages

- Efficace
- Chaque ID doit être unique dans le document HTML
- Les testeurs peuvent (théoriquement) ajouter des ID au SUT

## Inconvénients

- Les ID peuvent changer dynamiquement lorsqu'ils sont générés automatiquement
- ID non approprié pour le code utilisé dans plusieurs endroits
- Les testeurs peuvent ne pas être autorisés à modifier le SUT

# Exercice: localiser avec l'ID

- Utiliser WebDriver pour ouvrir la page:

***<http://automationpractice.com/index.php>***

- Localisez la boîte de recherche par ID et imprimez le résultat pour montrer s'il a été trouvé ou non

## *Tips*

dans Chrome, vous pouvez voir le HTML pour un élément UI en effectuant un clic droit et en sélectionnant ***Inspect***

# Localiser les éléments UI avec le nom de classe

- Faire référence à l'attribut HTML **class**

```
<element class="class-name1">
```

- Exemple de localisation avec Python WebDriver :

```
element_found = driver.find_element_by_class_name('class-name1')
```

## Avantages

- Les noms de classes peuvent être utilisés à plusieurs endroits dans le DOM
- Les testeurs peuvent ajouter des noms de classe au SUT

## Inconvénients

- Il faut veiller à ne pas localiser le mauvais élément
- Les testeurs peuvent ne pas être autorisés à modifier le SUT

# Exercice: localiser avec le nom de classe

- Utiliser WebDriver pour ouvrir la page:

***<http://automationpractice.com/index.php>***

- Localisez le lien « Sign In » par nom de classe et imprimez le résultat de savoir s'il a été trouvé ou non



# Localiser les éléments UI avec le nom de balise

- Faire référence à l'attribut HTML **tag name**

`<h2>`

- Exemple de localisation avec Python WebDriver :

```
heading2_found = driver.find_element_by_tag_name('h2')
```

## Avantages

- si un nom de balise est unique pour la page, il est plus facile à trouver

## Inconvénients

- si le nom de la balise n'est pas unique pour la page, vous pouvez localiser le mauvais élément

# Exercice: localiser avec le nom de balise

- Utiliser WebDriver pour ouvrir la page:

***<http://automationpractice.com/index.php>***

- Localiser l'élément UI **<h1>** au moyen de son nom de balise et imprimer le résultat pour montrer s'il a été trouvé ou non

# Localiser les éléments UI avec le Texte du Lien

- Faire au texte placé dans un lien cliquable

```
<a href="next.html">Next Page</a>
```

- Peut être trouvé par le texte de lien complet ou partiel

```
element = driver.find_element_by_link_text('Next Page')
```

```
element = driver.find_element_by_partial_link_text('Next Pa')
```

## Avantages

- Peut trouver l'élément s'il est unique sur la page
- Le texte du lien est visible pour l'utilisateur (dans la plupart des cas), il est donc facile de savoir ce que le code de test recherche
- Le texte de lien partiel a légèrement moins de chance de changer que le texte de lien complet

## Inconvénients

- Plus susceptibles de changer que le nom d'ID ou de classe
- Le texte partiel peut créer plus de correspondances

# Exercice: localiser avec le texte du lien

- Utiliser WebDriver pour ouvrir la page:

***<http://automationpractice.com/index.php>***

- Localisez le lien «Contact us» par le texte du lien et imprimez le résultat pour montre qu'il a été trouvé
- Localisez le lien «Contact us» par le texte partiel du lien et imprimez le résultat pour montre qu'il a été trouvé

# Localiser les éléments UI avec Xpath

- Soit le code HTML suivant

```
<html>
<body>
  <form id="sample_form1">
    <input name="text1" type="text" />
    <input name="text2" type="text" />
    <input name="submit_button" type="submit" value="Enter" />
  </form>
</body>
</html>
```

- Le Xpath peut s'écrire en chemin absolu ou relatif

```
element = driver.find_element_by_xpath('/html/body/form[2]')
```

```
element = driver.find_element_by_xpath("//form[@id='sample_form1']/input[2]")
```

## Avantages

- Peut trouver des éléments sans attributs uniques (ID, classe, nom, etc.)
- Peut utiliser XPath dans des localisateurs génériques, en utilisant les différentes stratégies x\_by\_ (par ID, par classe, etc.) au besoin

## Inconvénients

- Le code XPath absolu est fragile; il peut se rompre lors des plus petits changements dans la structure HTML. **Il FAUT** utiliser les méthode relative
- XPath peut être implémenté différemment sur différents navigateurs ou non pris en charge nativement (par exemple, IE); nécessitera un effort supplémentaire

# Exemple de Generic Locator pour Xpath

- Un localisateur XPath générique

```
def find_by_xpath(driver, path_string):  
    element = driver.find_element_by_xpath('path_string')  
    return element
```

- Peut être appelé pour différents attributs en changeant la chaîne qui est passée

```
path_string = "//*[@id = '%s']" % 'id_to_find'  
path_string = "//*[@class = '%s']" % 'class_to_find'
```

- Appel de la fonction générique

```
element_found = find_by_xpath(driver, path_string)
```

# Exercice: localiser avec XPath

- Utiliser WebDriver pour ouvrir la page:

***<http://automationpractice.com/index.php>***

- Localisez l'élément UI de la zone de recherche avec XPath et imprimez le résultat pour montrer qu'il a été trouvé

Remarque: une approche consiste à rechercher d'abord une balise, puis un nom d'ID ou de classe dans le XPath

# Localiser les éléments UI avec le sélecteur CSS

- Soit le code HTML suivant

```
<html>
  <body>
    <p class="paragraph">Some Latin nonsense.</p>
  </body>
</html>
```

- Vous pouvez écrire

```
element = driver.find_element_by_css_selector('p.paragraph')
```

## Avantages

- Peut trouver des éléments sans attributs uniques (ID, classe, nom, etc.)
- Comme avec Xpath, vous pouvez affiner votre recherche

## Inconvénients

- Si un élément n'est pas unique à une page, vous pouvez localiser le mauvais élément



# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 3

Utiliser le WebDriver de Selenium

Section 6:

Obtenir l'état des éléments GUI

# Objectif d'apprentissage

STF-3.6 (K3) Obtenir l'état des éléments GUI à l'aide des commandes de WebDriver

# Obtenir l'état des éléments UI

- Après avoir localisé un WebElement spécifique, nous avons souvent besoin d'obtenir des informations à partir de celui-ci
- L'information peut inclure:
  - Visible/non visible
  - Activé/non activé
  - Sélectionné/non sélectionné
- Raisons d'un automaticien pour déterminer cette info :
  - S'assurer que l'état est comme prévu à un moment donné du test
  - S'assurer qu'un contrôle est dans un état qui peut être manipulé selon les besoins du cas de test (par exemple, activé)
  - S'assurer qu'une fois que le contrôle a été manipulé, il est dans l'état attendu
  - S'assurer que les comportements attendus sont corrects

# Propriétés et accesseurs communs

Propriété/Méthode	Arguments	Retourne	Description
<b>get_attribute()</b>	propriété à récupérer	Valeur Propriété, attribut ou aucun	Obtient la valeur de la propriété. Si aucune propriété de ce nom, obtient la valeur de l'attribut de ce nom. Sinon, retourne None
<b>get_property()</b>	propriété à récupérer	Valeur Propriété	Obtient la valeur de la propriété.
<b>is_displayed()</b>		Boolean	Retourne true si visible pour l'utilisateur
<b>is_enabled()</b>		Boolean	retourne true si l'élément est activé
<b>is_selected()</b>		Boolean	Retourne true si CheckBox ou radio est sélectionné
<b>location</b>		X,Y emplacement	Retourne l'emplacement X, Y sur le canevas de rendu
<b>size</b>		Hauteur, largeur	Retourne la hauteur et la largeur de l'élément
<b>tag_name</b>		La propriété tag_name	Retourne le tag_name de l'élément
<b>text</b>		Texte de l'élément	Retourne le texte associé à l'élément

# Obtenir une valeur de texte

- Localisez l'élément `WebElement` désiré

```
element = driver.find_element_by_id('element_id')
```

- Utilisez la propriété **text** pour accéder au texte

```
element_text = element.text
```

# Exercice: obtenir la valeur du texte

- Utiliser WebDriver pour ouvrir la page:

***<http://automationpractice.com/index.php>***

- Obtenez la valeur de texte pour le lien ***Sign in***
- Imprimer le texte qui a été retourné

## *Tips*

Une façon d'imprimer le résultat :

```
link_text = <code to get the link text>
print ('Sign in link text is: ' + link_text)
```

# Obtenir une valeur d'attribut

- Localisez le WebElement souhaité

```
element = driver.find_element_by_id('element_id')
```

- Utilisez la méthode ***get\_attribute()*** pour accéder à la valeur désirée

```
value = element.get_attribute('attribute_name')
```

## REMARQUE

- Notez que cette méthode essaiera d'abord de récupérer la propriété de l'élément
- Si cela échoue, elle va essayer de récupérer l'attribut
- Si cela échoue, retourne None



# Exercice: Récupérer la valeur d'un attribut

- Utiliser WebDriver pour ouvrir la page:

***<http://automationpractice.com/index.php>***

- Utiliser la valeur de l'attribut href pour l'élément UI ***Sign in***
- Imprimer

*Note: cela devrait ressembler à du html exécutable*

# Obtenir l'état sélectionné

- Localisez le WebElement souhaité
- Utilisez la méthode ***is\_selected()*** pour obtenir l'état actuel de l'élément
- Retourne true si l'élément est sélectionné, sinon retourne false

```
element = driver.find_element_by_id('checkbox_id')  
checkbox_state = element.is_selected()
```

# Exercice: obtenir l'état sélectionné

- Utiliser WebDriver pour ouvrir la page:

***[http://automationpractice.com/index.php?id\\_category=8&controller=category#/categories-casual\\_dresses/compositions-cotton](http://automationpractice.com/index.php?id_category=8&controller=category#/categories-casual_dresses/compositions-cotton)***

- Obtenir l'état sélectionné pour la case à cocher Taille S
- Imprimer
- En utilisant les commandes sleep(), suspendre le script assez longtemps pour vérifier la boîte manuellement
- Obtenir l'état sélectionné pour la case à cocher Taille S
- Imprimer

# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 3

Utiliser le WebDriver de Selenium

Section 7:

Interagir avec les éléments GUI

# Objectif d'apprentissage

STF-3.7 (K3) Interagir avec les éléments GUI à l'aide des commandes de WebDriver

# Interagir avec les éléments UI

- L'interaction avec un élément Web implique :
  - Localiser l'élément (section 3,5)
  - S'assurer que l'élément est dans l'état correct (section 3,6)
  - Contrôler l'élément pour changer son état comme souhaité
- En IHM, chaque WebElement se manipule selon sa nature
  - Tapez dans le champ de texte
  - Cliquez sur un bouton, un lien, une image, etc.
  - Sélectionner un élément dans une liste
  - Soumettre le formulaire de traitement
- Il peut être nécessaire de prendre en compte les besoins de synchronisation

# Interagir avec les champs de texte

- Localiser le champ de texte
- Assurez-vous qu'il est en bon état (visible et activé)
- Effacer le texte existant hors de lui
- Simuler la frappe en utilisant le clavier

```
# First clear the control  
element.clear()  
  
# Now type into the control  
string_to_type = 'XYZ'  
element.send_keys(string_to_type)
```



# Interagir avec les éléments UI cliquables

- Localiser le WebElement
- Assurez-vous qu'il est activé
- Déterminez si 'cliquable' pour attendre la synchronisation (utiliser les ***expected\_condition()*** méthode examinée au chapitre 4)

```
Driver.support.expected_conditions.element_to_be_clickable(locator)
```

- Cliquez sur le WebElement

```
element.click()
```

- Si bouton radio, vérifier qu'il est maintenant sélectionné

```
element.is_selected()
```

# Exercice: cliquer sur WebElement

- Utiliser WebDriver pour ouvrir la page

***<http://automationpractice.com/index.php>***

- Localiser le lien Sign In, cliquez dessus, et vérifiez que la page de connexion (Login) est affichée

## ***Remarque***

*une façon de vérifier que la page de connexion est affichée est de rechercher la présence du titre de la page*

# Interagir avec les éléments CheckBox

- Une CheckBox est un élément cliquable
- Toutefois, son état final dépend de l'état de départ
  - Si elle est cochée, un clic la décoche
  - Si elle est décochée, un clic la coche
- On ne peut pas aveuglément cliquer dessus; on doit connaître l'état final désiré
- Une bonne occasion d'ajouter une certaine intelligence par la construction d'une fonction d'agrégation

L'algorithme sera du type :

**IF** Si nous voulons qu'elle soit cochée et qu'elle ne l'est pas, cliquons dessus

**Else IF** Si nous la voulons décochée et qu'elle est cochée, cliquons dessus

# Exemple de fonction pour CheckBox

Définir une fonction avec pour arguments :

- CheckBox à utiliser: *'element'*
- État désiré (Boolean): *'want\_checked'*

```
def set_check_box(element, want_checked):  
    if want_checked and not element.is_selected():  
        element.click()  
    elif element.is_selected() and not want_checked:  
        element.click()
```

Appeler la fonction :

```
set_checkbox (checkbox, True)
```

# Interagir avec les listes déroulantes (Select)

**Les contrôles Select (listes déroulantes) autorisent l'utilisateur à sélectionner un ou plusieurs éléments d'une liste**

1. Importer la librairie Select :

```
from selenium.webdriver.support.ui import Select
```

2. Instancier l'élément (liste déroulante) en tant qu'objet Select

```
select = Select(driver.find_element_by_id(element_id))
```

3. Manipuler la liste. Ex : *select*. ...

- Sélectionner par une valeur d'élément (***select\_by\_value(value)***)
- Sélectionner tous les éléments qui affichent le texte correspondant (***select\_by\_visible\_text(text)***)
- Sélectionner un élément par son index (***select\_by\_index(index)***)
- Désélectionner tous les éléments (***deselect\_all()***)
- Désélectionner au moyen de l'index (***deselect\_by\_index(index)***)
- Désélectionner au moyen de la valeur (***deselect\_by\_value(value)***)
- Désélectionner au moyen du texte visible (***deselect\_by\_visible\_text(text)***)
- Renvoyer une liste de tous les éléments sélectionnés (***all\_selected\_options***)
- Retourner le premier sélectionné (ou seulement si un seul contrôle Select) (***first\_selected\_option***)
- Voir la liste de toutes les options de la liste (***options***)

# Cliquer sur une option de liste déroulante

- Il peut être plus aisé de manipuler un Select comme n'importe quel WebElement et définir une fonction générale de manipulation
- Définir une fonction avec pour arguments:
  - La référence de l'objet WebDriver: *'driver'*
  - L'**ID** de l'objet dropdown: *'dropdown\_id'*
  - L'ID de l'option souhaitée: *'option\_id'*

```
def click_dropdown_option_by_id_and_id(driver, dropdown_id, option_id):  
    dropdown_element = driver.find_element_by_id('dropdown_id')  
    dropdown_element.click()  
    option_element = driver.find_element_by_id('option_id')  
    option_element.click()
```

# Exercice: utilisation d'une liste Dropdown

- Utiliser WebDriver pour ouvrir la page:

***<http://automationpractice.com/index.php?controller=prices-drop>***

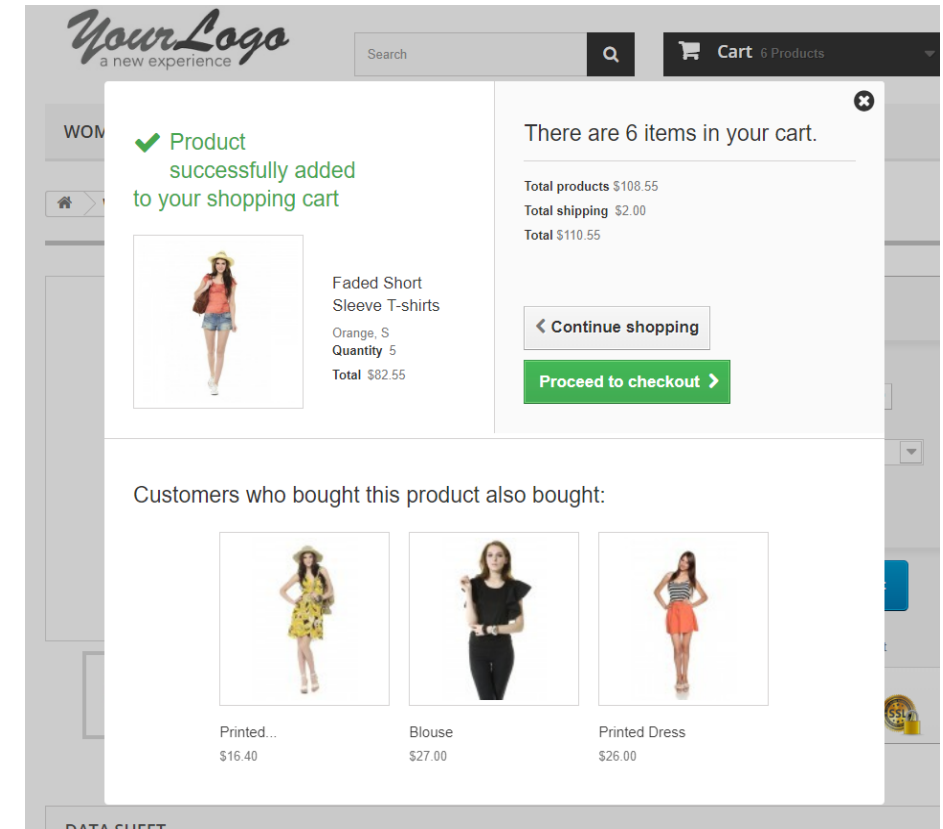
- Sélectionnez l'option 'In Stock' depuis le menu déroulant 'Sort By'

## ***Remarque***

*une façon de faire est de cliquer sur le menu déroulant, puis de cliquer sur l'option*

# Interagir avec les dialogues modaux

- Une boîte de dialogue modale apparaît sur un écran Web et empêche l'interaction avec l'écran Web jusqu'à sa fermeture
- Les boîtes de dialogue modales sont utilisées
  - Pour obtenir des informations spécifiques de l'utilisateur
  - pour que l'utilisateur exécute certaines tâches
- Exemple (image ci-contre)
  - Après l'ajout de l'élément au panier sur un site e-commerce, permet de choisir entre continuer les achats ou Check out
  - Montrer plus d'options d'achat basées sur le choix précédent





# Traitement des dialogues modaux (1)

- Tout le code de la boîte de dialogue modale est en fait placé dans le HTML de l'écran d'appel
- L'utilisation de la boîte de dialogue modale dépend de l'élément modal trouvé dans le code html
- Dans ce site e-commerce, l'ID de l'élément modal est **layer\_cart**
- Vous souhaitez obtenir une référence à cet élément

```
modal = driver.find_element_by_id('layer_cart')
```

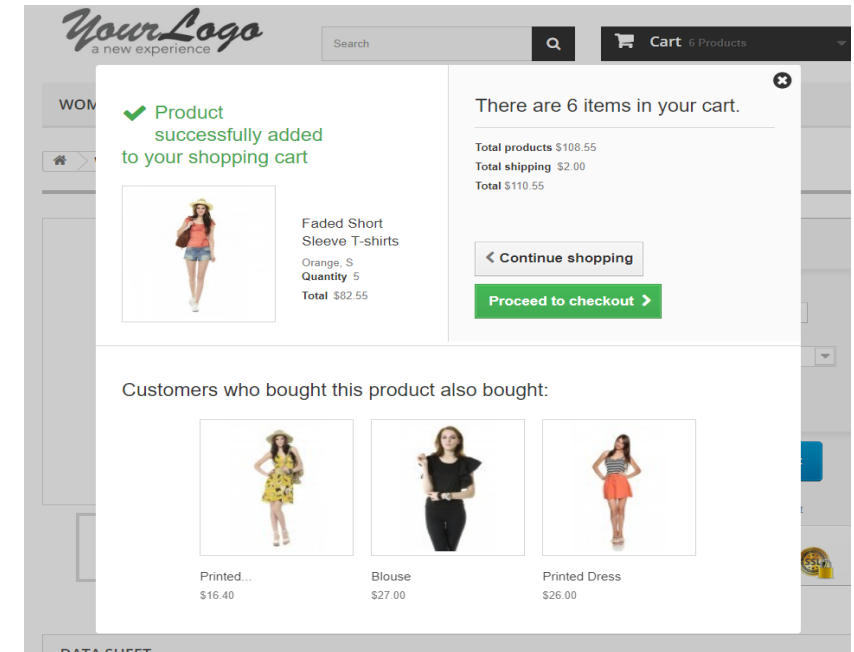
# Traitement des dialogues modaux (2)

- La tâche suivante consiste à déterminer ce que vous voulez faire dans la boîte de dialogue modale
- Dans notre cas, cliquer sur le bouton *Proceed to Checkout*

```
proceed_button = modal.find_element_by_class_name('button-medium')
```

- Cliquer sur le bouton pour revenir à l'écran web principal

```
proceed_button.click()
```



# Exercice: dialogues modaux

- Utiliser WebDriver, pour aller sur la page:  
***<http://automationpractice.com>***
- Cliquez sur le premier élément (Faded Short Sleeve T-shirts)
- Cliquez sur le bouton Add to cart
- Trouvez le modal et cliquez sur le bouton Proceed to checkout
- Vérifiez que la page du panier d'achat est affichée

# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Automatisation avec Selenium

## Selenium Tester Foundation

### Chapitre 3

#### Utiliser le WebDriver de Selenium

##### Section 8:

Interagir avec les invites utilisateur (User Prompts)  
dans les navigateurs Web

# Objectif d'apprentissage

STF-3.8 (K3) Interagir avec les invites utilisateur dans les navigateurs Web à l'aide des commandes de WebDriver

# Boîtes de dialogue d'invite d'utilisateur

- Une invite d'utilisateur (pop-up) est une boîte de dialogue modale générée par une page Web
- L'invite utilisateur nécessite une manipulation spéciale car elle ne fait pas partie de la page du navigateur
- Le W3C définit trois invites d'utilisateur similaires:
  - Alert
  - Confirm
  - Prompt
- Étant donné que les trois invites utilisateur sont très similaires et ont sensiblement les mêmes comportements, nous parleront « d'alerte »

# Interagir avec une alerte

## Voici quelques méthodes de manipulation d'alerte

- Capturer le texte d'une alerte afin de pouvoir vérifier le message et le comparer à un message attendu

```
alert = driver.switch_to.alert
msg_text = alert.text
expected_text = 'XYZ'
assert expected_text in msg_text, "The expected text not found"
```

- Il existe de deux façons de fermer la boîte de dialogue d'alerte

```
alert = driver.switch_to.alert
alert.accept()
alert.dismiss()
```



# Exercice: utilisation des alertes

- Utiliser WebDriver pour ouvrir la page:

***[https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref\\_alert](https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_alert)***

- Cliquez sur le bouton “Try it” (attention, il y a une iframe) Cela affichera un message d'alerte
- Vérifiez que le texte attendu est dans l'alerte, puis acceptez l'alerte

# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Automatisation avec Selenium

Selenium Tester Foundation

## Chapitre 4

Préparation de scripts de test  
maintenables

# Termes à mémoriser

fixture

persona

wrapper

# Automatisation avec Selenium

## Selenium Tester Foundation

### Chapitre 4

Préparation de scripts de test maintenables

Section 1 :

Maintenabilité des scripts de test

# Objectif d'apprentissage

STF-4.1 (K2) Comprendre quels facteurs soutiennent et affectent la maintenabilité des scripts de test

# Automatisation intelligente

- Les testeurs manuels sauront probablement toujours mieux tester que l'automatisation
- Mais l'automatisation peut simuler les processus de réflexion des testeurs manuels
- Quand un testeur manuel utilise un contrôle, il pense
  - Le contrôle existe-t-il?
  - Le contrôle est-il visible?
  - Le contrôle est-il activé?
  - Si tout ce qui précède est vrai, ils utilisent le contrôle
  - Il vérifie ensuite que le contrôle a fait ce qu'il attendait
- Il s'agit d'un algorithme utile pour ajouter de l'intelligence à la fonctionnalité d'un outil en créant des *wrappers* autour de la fonctionnalité d'outil

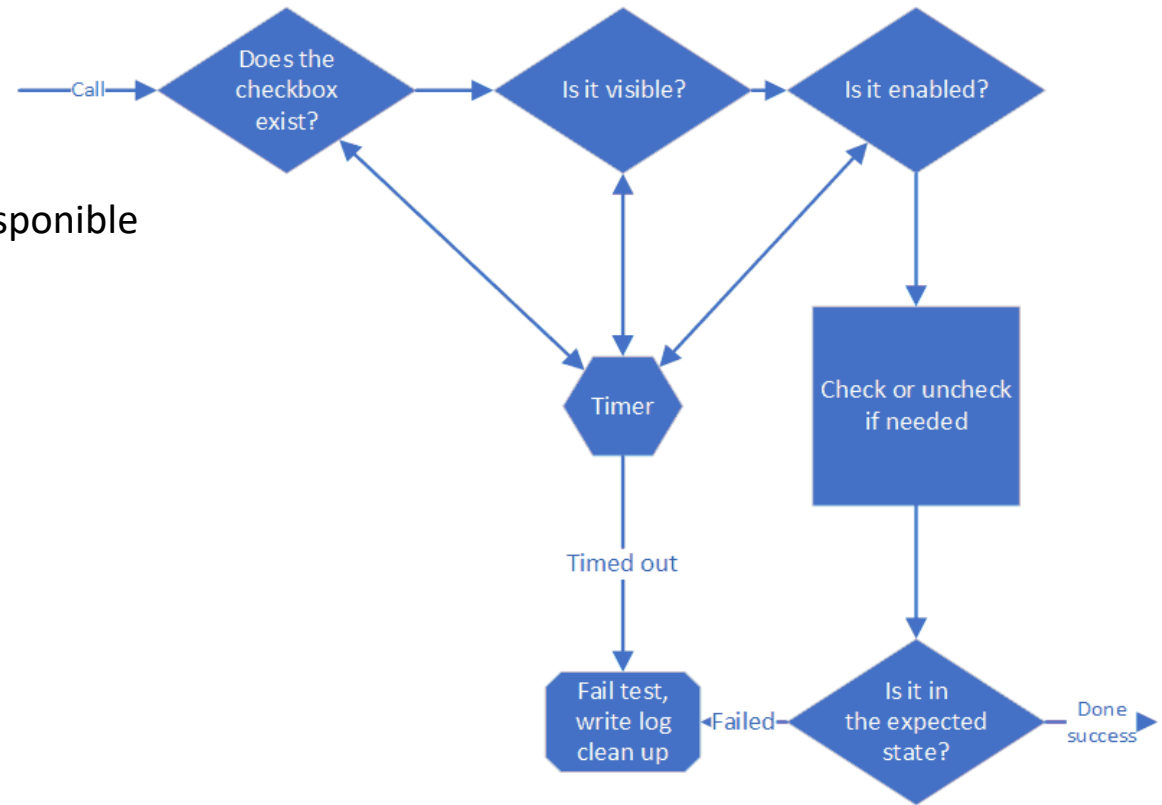
# Construire des fonctions Wrappers

## Exemple d'une fonction callable pour une case à cocher

Arguments à passer :

- Le contrôle CheckBox à utiliser
- L'état final désiré (coché/non coché)
- La temps à attendre pour que le contrôle devienne disponible

Il faut anticiper des ressources pour construire ces wrappers, mais ils peuvent réduire les faux positifs et améliorer le suivi





# Créer d'autres fonctions

- Le code utilisé à plusieurs reprises **doit** être mis dans des fonctions
- Cette approche est indépendante de l'Orienté Objet (écrire des fonctions n'est pas synonyme d'orienté objet !)

Ex de construction simple :

- Les fonctions de **localisateur** peuvent être placées dans un fichier appelé `locators.py`
- Les **actions fréquemment exécutées** (et la gestion des erreurs) peuvent être placées dans un fichier appelé `helpers.py`

# Utiliser des chemins d'accès relatifs

- Les éléments Web peuvent être localisés en utilisant des chemins d'accès absolus ou des chemins relatifs
- Les chemins absolus peuvent se rompre si l'interface utilisateur de la page Web est modifiée
- Les chemins **d'accès relatifs sont plus robustes**, surtout lorsque le chemin commence à proximité de l'objet Web désiré
- Travailler avec les développeurs pour s'assurer que le code HTML est bien écrit et les changements aussi rares que possible

# Nommage et commentaires

- **UTILISER LES VARIABLES** : Les noms globaux (variables et constantes) rendent le code de test plus lisible
- **ECRIRE DES COMMENTAIRES** : Les commentaires devraient expliquer pourquoi vous faites quelque chose: n'importe quel automaticien peut donc voir ce que vous avez fait!
- Les noms des fonctions / méthodes / classes doivent être cohérents
- Il devrait y avoir une approche cohérente et logique de nommage, par exemple, deux fonctions connexes ne devraient pas être appelées

```
get_user_permission_state()
```

Et

```
get_permiss_state_all_users()
```

# Noms de fichiers cohérents

- Utiliser des noms de fichiers cohérents qui véhiculent leur place dans la hiérarchie fonctionnelle

```
test_admin_users_create.py  
test_admin_users_edit.py
```

- Plutôt que :

```
test_admin_create_users.py
```

- Utilisez :

```
test_admin_users_create.py
```

# Créer des comptes de test et des fixtures

- Les comptes de messagerie (servant au JDD ou à recevoir le résultat des exécutions) ne doivent pas être liés à un utilisateur spécifique. Un compte générique, p. ex. ***qauser@example.com*** est préférable
- Le partage du compte de messagerie et des comptes SUT avec d'autres utilisateurs peut introduire l'incertitude (y compris pour les testeurs manuels)
- Les comptes de messagerie, les comptes SUT et les fixtures ne doivent être utilisés que par le groupe d'automaticiens
- Rendre ces comptes réalistes pour simuler des utilisateurs réels (y compris les *personas*)

# Log de suivi d'exécution (progrès)

- Le module de journalisation python (**logging**) facilite les activités de diagnostic et d'audit
- Plutôt que d'utiliser les déclarations **print()**, vous pouvez diriger les sorties enregistrées vers un fichier journal et/ou la console
- Le fichier journal peut vous donner une vue claire de la navigation effectuée et de l'état du logiciel pendant l'exécution

## *Exemple de journalisation*

- Lorsque vous naviguez vers une page de connexion :

```
logging.info('login page found')
```

- Lorsque vous trouvez le Create User modal:

```
logging.info('Create User modal found')
```

- Lorsque vous ne trouvez pas de texte attendu dans une alerte:

```
logging.error('ERROR: Unable to find text in alert')
```

# Automatisation avec Selenium

## Selenium Tester Foundation

### Chapitre 4

Préparation de scripts de test maintenables

Section 2 :

Mécanismes d'attente  
(synchronisation)

# Objectif d'apprentissage

STF-4.1 (K3) Utiliser les mécanismes d'attente appropriés



# Pourquoi les attentes sont-elles nécessaires?

- Les testeurs manuels n'ont pas besoin « d'attendre »
- Ils peuvent voir quand une action se produit
  - Si elle se déroule dans un laps de temps raisonnable, ils passent à l'étape suivante
  - Si quelque chose a pris trop de temps, ils écrivent simplement un rapport d'incident si c'est répétable
- Les outils d'automatisation fournissent peu de contexte ou de notion du raisonnable
- C'est à l'automaticien de définir ce qui est raisonnable lorsque le script s'exécute

# Problèmes avec les attentes dans l'automatisation

- Les outils d'automatisation sont d'esprit littéral
- Si on leur dit d'attendre 3 secondes, à 3,001 secondes ce sera l'échec (pas de caractère raisonnable)
- Certains automaticiens utilisent les pires choix:

```
time.sleep(5)
```

- La fonction ***sleep()*** constitue toujours le pire cas possible et ne doit pas être utilisée
- Le WebDriver avec Python a deux attentes différentes:
  - Attente implicite
  - Attentes explicites

# Attentes implicite

- Définit une attente dans l'objet WebDriver
- Une fois réglé, reste actif jusqu'à ce que l'objet WebDriver soit « détruit »
- Chaque fois qu'un WebElement est adressé, WebDriver interroge DOM jusqu'à ce que l'objet soit trouvé ou que le temps s'épuise (timeout)

```
driver = webdriver.Chrome()  
driver.implicitly_wait(10)
```

# Attentes explicites

- ***Sleep(n)*** fait partie des attente explicite
- Mais d'autres attentes explicites sont plus appropriée à l'automatisation de test
- Python définit un certain nombre de ***ExpectedConditions*** qui simplifient le réglage des attentes explicites
- ***ExpectedConditions*** est utilisable à partir d'une méthode du WebDriver appelée ***WebDriverWait()***

*Le code ci-dessous attend qu'un élément soit cliquable pour l'affecter à une variable « element » :*

*Les imports de librairies nécessaires*

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By
```

*Le code*

```
wait = WebDriverWait(driver, 10)
element = wait.until(EC.element_to_be_clickable((By.ID, 'someID')))
```

# Méthodes ExpectedCondition définies

- ✚ title\_contains
- ✚ presence\_of\_element\_located
- ✚ visibility\_of\_element\_located
- ✚ presence\_of\_all\_elements\_located
- ✚ text\_to\_be\_present\_in\_element
- ✚ text\_to\_be\_present\_in\_element\_value
- ✚ frame\_to\_be\_available\_and\_switch\_to\_it
- ✚ invisibility\_of\_element\_located
- ✚ element\_located\_to\_be\_selected
- ✚ element\_selection\_state\_to\_be
- ✚ element\_located\_selection\_state\_to\_be
- ✚ title\_is
- ✚ visibility\_of
- ✚ element\_to\_be\_clickable
- ✚ staleness\_of
- ✚ element\_to\_be\_selected
- ✚ alert\_is\_present

# Exemple d'attente explicite avec gestion de timeout

```
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException
```

```
def click_by_xpath(driver, path):
    try:
        element = wait(driver, 5).until(EC.element_to_be_clickable((By.XPATH, path)))
        element.click()
    except TimeoutException:
        element = None
    return element
```

# Exercice: travailler avec des attentes

- Dans un navigateur Web, ouvrez
- **<https://www.python.org/>**
- Rechercher la boîte de recherche
- Effectuez une recherche avec n'importe quel terme en cliquant sur le bouton « Go »"
- Vérifiez que la recherche a été exécutée en utilisant les attentes implicites appropriées
- Exécuter le même test à l'aide d'attentes explicites

# Automatisation avec Selenium

## Selenium Tester Foundation

### Chapitre 4

Préparation de scripts de test maintenables

Section 3 :

Le Page Object Pattern



# Objectif d'apprentissage

STF-4.3 (K4) Analyser le GUI de SUT et utiliser des Page Object pour faire ses abstractions

# Utiliser Page Object

- Un Page Object est une classe, un module ou un autre ensemble de fonctions qui contient l'interface à la forme ou à la page testée.
- Déplacer la complexité des scripts vers le TAA\*
- Diviser la solution d'automatisation de test en couches
  - La couche d'abstraction de test (Test Abstraction Layer) doit être une partie distincte de TAS (habituellement dans TAA)
  - Un Page Object est une partie de la couche d'adaptation de test (Test Adaptation Layer)
  - Le script se trouve dans la couche d'exécution de test (Test Execution Layer)

*\*Test Automation Architecture (architecture d'automatisation des tests)*

# Raisons d'utilisation du Page Object

- Crée un code réutilisable qui peut être partagé par plusieurs scripts de test
- Réduit la quantité de code dupliqué
- Encapsule toutes les opérations sur l'interface graphique du SUT en une seule couche
- Lorsque le changement se produit inévitablement, il constitue un point unique pour réparer tous les scripts affectés
- Réduit les coûts et les efforts de maintenance pour les scripts d'automatisation de test
- Donne une séparation claire entre les aspects métier et techniques lors de la conception des tests automatisés

# Exemple de code n'utilisant pas Page Objects

```
first_name = self.browser.find_element_by_css_selector("#id_first_name")
first_name.send_keys("Clem")
last_name = self.browser.find_element_by_css_selector("#id_last_name")
last_name.send_keys("Kaddidlehopper")
password = self.browser.find_element_by_css_selector("#id_password")
password.send_keys("QWERTY")
email = self.browser.find_element_by_css_selector("#id_email")
email.send_keys("test+43@example.com")
product = self.browser.find_element_by_css_selector("#id_the_product")
product.send_keys("test")
self.browser.find_element_by_css_selector('#create_account_form button').click()
self.browser.find_element_by_css_selector('#next-step').click()
self.browser.find_element_by_css_selector('#next-step').click()
self.browser.find_element_by_css_selector('#next-step').click()
```

# Même code en utilisant Page Objects

```
signup_form = homepage.getSignupForm()
signup_form.setName("Clem", "Kaddidlehopper")
signup_form.setPassword("QWERTY")
signup_form.setEmail('test+43@example.com')
signup_form.setProductName('test')
onboarding_1 = signup_form.submit()
onboarding_2 = onboarding_1.next()
onboarding_3 = onboarding_2.next()
items_page = onboarding_3.next()
```

# Exemple de Page Object

```
class BasePage(object):  
  
    def __init__(self, driver):  
        self.driver = driver
```

```
class SignupPage(BasePage):  
    url = "http://localhost:8000/account/create/"  
  
    def setName(self, first, last):  
        self.fill_form_by_id("id_first_name", first)  
        self.fill_form_by_id("id_last_name", last)  
  
    def setEmail(self, email):  
        self.fill_form_by_id("id_email", email)  
  
    def setPassword(self, password):  
        self.fill_form_by_id("id_password", password)  
        self.fill_form_by_id("id_password_confirmation", password)  
  
    def setProductName(self, name):  
        self.fill_form_by_id("id_first_product_name", name)  
  
    def submit(self):  
        self.driver.find('#create_account_form button').click()  
        return OnboardingInvitePage(self.driver)
```

# Règles de construction de Page Object

- Les Page Objects ne doivent pas contenir d'affirmations métier ni de points de vérification
- Les Page Objects peut contenir des vérifications techniques (par exemple, si une page a été chargée avec succès) et des attentes
- Un Page Object n'a pas besoin de couvrir la page entière, il peut s'appliquer à seulement une section d'un formulaire
- Seul le Page Object devrait contenir des fonctions de Selenium

# Page Object Example sur une page RBCS

The screenshot shows the RBCS website with several annotations in yellow boxes:

- Header**: Points to the top navigation bar containing the RBCS logo, links for Training, Consulting, Resources, and Store, and a right-side menu with CONTACT, ABOUT, and SEARCH.
- Choose category**: Points to a dropdown menu in the section 'Get software testing training resources:' which currently shows 'Best Practices' and a 'Go' button.
- Rest of the page**: A large box encompassing the main content area below the header.

The main content area features a large hero image of a man on a phone. Below it, there is a video player for 'RBCS, Inc.' and a text block titled 'Why choose us for software testing training and consulting' with a 'Learn more about RBCS' link.



# Page Objects Example

- Nous pouvons extraire 3 Page Objects de la partie visible de la page de RBCS:
  - En-tête
  - Sélecteur de catégorie
  - Reste de la page
- Nous pouvons exporter et implémenter plusieurs opérations de la partie d'en-tête, par exemple:
  - Recherche
  - Cliquez sur le logo
  - Afficher la page « about »

# Exemple

## Extrait de code d'une méthode d'un Page Object

```
def doSearch(self, search_text):  
    driver.find_element_by_xpath("//body[@id='home']/div/div/div/header/div/div/button").click()  
    driver.find_element_by_name("q").clear()  
    driver.find_element_by_name("q").send_keys(search_text)  
    driver.find_element_by_xpath("//button[@type='submit']").click()
```

# Exercice: travailler avec Page Object

- Analyser la page ***<http://automationpractice.com>***
- Divisez-la en page objects logiques
- Automatisez ces étapes:
  - Ouvrir la page
  - Recherche de "chiffon"
  - Ajouter le premier vêtement trouvé au panier
- Écrivez le code à l'aide du modèle des Page Object (diviser le code en fonctions de cas de tests et de fonctions page objects séparées)

# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Automatisation avec Selenium

## Selenium Tester Foundation

### Chapitre 4

Préparation de scripts de test maintenables

Section 4 :

Test piloté par Mots clés  
(keyword-driven testing)

# Objectif d'apprentissage

STF-4.4 (K4) Analyser les scripts de test et appliquer des principes de test pilotés par mot clé pour concevoir des scripts de test

# Keyword Driven Testing (KDT)

## **Une extension naturelle du modèle Page Object :**

- Les mots clés sont abstraits, les termes liés au métier décrivant une tâche fonctionnelle que le SUT est censée remplir
- Le 'Quoi' (=qu'est-ce que le SUT fait) plutôt que le 'Comment'
- KDT modélise la façon dont un test manuel fonctionne
- Les analystes de tests non techniques définissent les mots-clés dont ils ont besoin en fonction des tâches à tester
- Les analystes techniques de test (automaticiens) créent la fonctionnalité derrière les mots clés et l'infrastructure pour exécuter les tests

### ***Définition de l'ISTQB***

Une technique de scripting qui utilise des fichiers de données pour contenir non seulement les données de test et les résultats attendus, mais également les mots clés liés à l'application sous test. Les mots clés sont interprétés par des scripts spéciaux de prise en charge, appelés par le script de contrôle du test.

# Mot-clé/mot action

- Un mot-clé (action) est une action métier abstraite ou une étape d'un cas de test
- En modélisant une étape de test manuelle, un script par mot-clé contiendra généralement trois colonnes:
  - La tâche à faire (le mot-clé)
  - Les données pour le faire (avec peut-être besoin de plusieurs colonnes)
  - Le résultat attendu s'il fonctionne correctement
- Le mot-clé est abstrait ; il ne contient pas d'actions physiques spécifiques sur le SUT
- L'implémentation des mots-clés (par exemple, dans un Page Object) contient les actions spécifiques sur le SUT



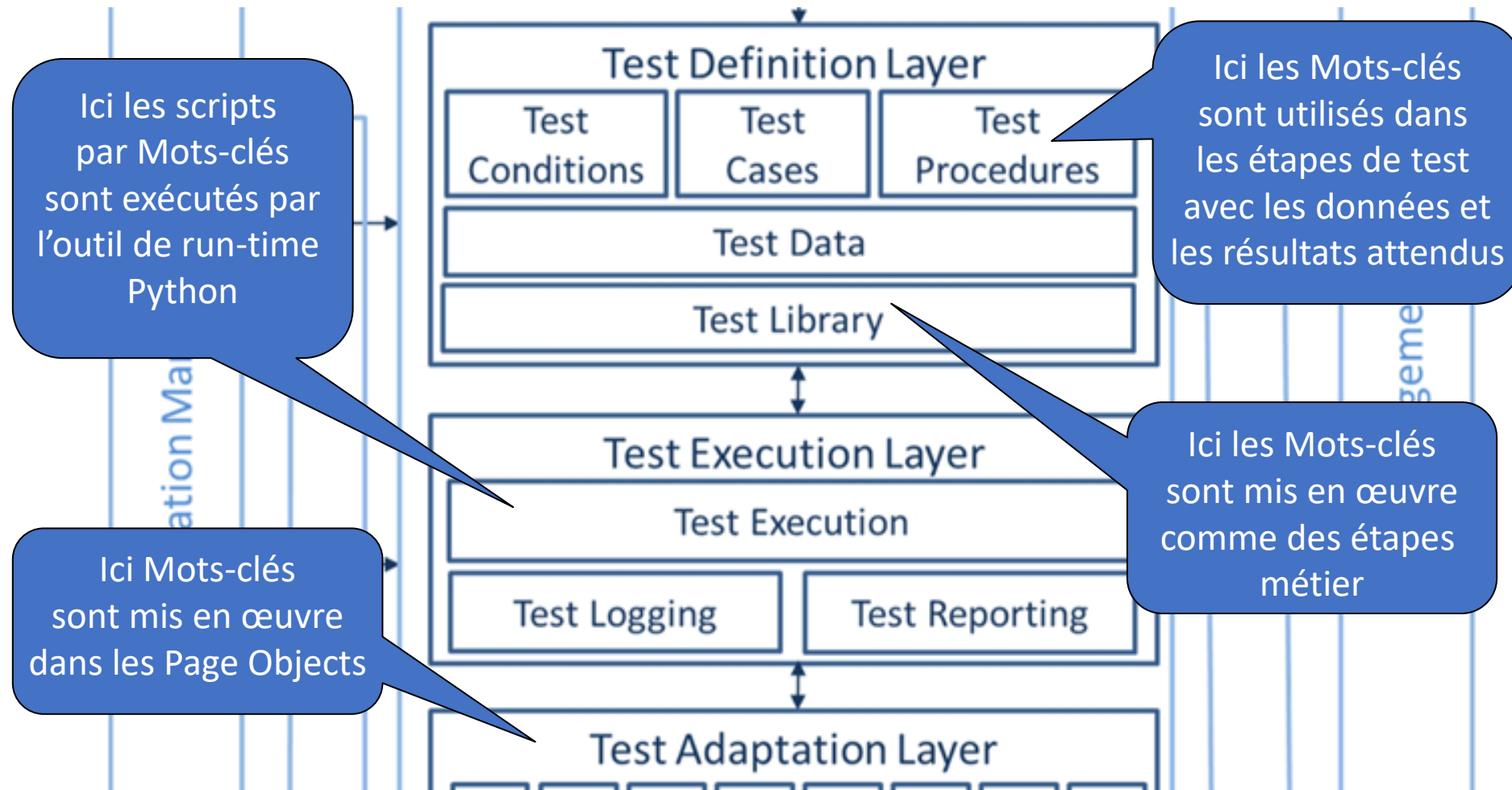
# Avantages du KDT (1)

- La conception de cas de test est découplée de l'interface du SUT
- Une distinction claire est faite entre les couches d'Execution, d'Abstraction, et de Definition des Tests.
- Une division presque complète du travail:
  - Les analystes conçoivent des cas de test et écrivent des scripts en utilisant des mots clés, des données et des résultats attendus
  - Les analystes techniques de test (automaticiens) implémentent les mots clés et l'infrastructure d'exécution nécessaire pour exécuter les tests
- Réutilisation des mots-clés dans différents cas de test
- Amélioration de la lisibilité des cas de test (ils ressemblent aux cas de test manuel)
- Moins de redondance (code maintenu à un seul emplacement)

# Avantages du KDT (2)

- Réduction des coûts d'entretien et de l'effort
- Si l'automatisation est hors ligne, un testeur manuel peut exécuter le test manuellement directement à partir du script automatisé
- Parce que les tests de mots clés sont abstraits, les scripts utilisant des mots-clefs peuvent être écrits bien avant que le SUT soit disponible pour le test (tout comme les tests manuels)
- Étant donné que les scripts sont complètement séparés du niveau de mise en œuvre, différents outils d'exécution peuvent être utilisés indifféremment

# Keyword Driven Testing dans TAA



# Choses à considérer

- Les mots-clés plus précis permettent des scénarios plus spécifiques, mais au ajoutent à la complexité de la maintenance des scripts

=> Les mots-clés agrégés peuvent simplifier le développement initial mais compliquer la maintenance

# Mise en œuvre de KDT

## Top-down

- Écrivez des étapes de cas de test comme vous écririez des tests manuels, puis implémentez-les dans votre outil
- Ces étapes peuvent être écrites par les analystes de test
- Plus rentable lorsque de nombreux mots-clés ont déjà été mis en œuvre ou lorsque les scripts de tests sont mis à jour

## Bottom-up

- Enregistrer un script, puis le réadapter pour rencontrer l'architecture KDT
- Habituellement utilisé au début de l'automatisation des tests

# Outils pour KDT

- Cucumber
- Robot Framework
- Katalon Studio
- Concordion
- HPE Unified Functional Testing

**De plus, l'architecture pilotée par Mots clés peut être implémentée dans de nombreux langages de programmation**

# Exemple d'un outil KDT

## Shop

▸ Settings

1	Search for	printed dress			
2	Add to cart	1			
3	Proceed to checkout				
4	Sign in	john.doe@mail.com	Password123		
5	Enter shipping address	777 One Way	90210	Beverly Hills	CA
6	Choose payment				
7	Confirm order				
8					
9					
10					
11					
12					

EditText EditRun

Apply ChangesSearch

```
1 *** Settings ***
2 Library Selenium2Library
3
4 *** Test Cases ***
5 Shop
6     Search for    printed dress
7     Add to cart   1
8     Proceed to checkout
9     Sign in       john.doe@mail.com    Password123
10    Enter shipping address    777 One Way    90210    Beverly
11    Choose payment
12    Confirm order
```

# Exemple d'un test KWD

```
my_shopping_address = ("777", "One Way",  
                        "90210", "Beverly Hills",  
                        "CA", "USA")  
  
def test_shop():  
    search_for("printed dress")  
    add_to_cart(1)  
    proceed_to_checkout()  
    sign_in("john.doe@mail.com", "Password123")  
    enter_shopping_address(my_shopping_address)  
    choose_payment()  
    sign_out()
```



# Exercice : de l'approche Top-Down à KDT (option)

- Ecrire une fonction de cas de test en Python
- Écrire des étapes de test en tant qu'appels de fonction dans cette fonction (à l'aide de mots clés)
- Implémenter les mots-clés à partir du cas de test
- Étapes de test:
  - Ouvrir <http://www.automationpractice.com>
  - Recherchez "printed dress"
  - Ajouter la première robe trouvée au panier
- Vous pouvez éventuellement implémenter tout le processus d'achat

# Questions types d'examen

- Evaluons vos connaissances
- Le formateur va choisir une question type extraite des notes
- Répondez à la question
- Le formateur discutera alors les bonnes (ou mauvaises) réponses possibles au besoin

# Annexe A (1)

**class attribute:** Attribut HTML qui pointe vers une classe dans une feuille de style CSS. Il peut également être utilisé par un JavaScript pour apporter des modifications aux éléments HTML avec une classe spécifiée

**comparateur:** Un outil pour automatiser la comparaison des résultats attendus par rapport aux résultats réels

**CSS selector:** Les sélecteurs sont des modèles qui ciblent les éléments HTML que vous souhaitez styler

**Document Object Model (DOM):** Interface de programmation d'application qui traite un document HTML ou XML en tant que structure arborescente dans laquelle chaque nœud est un objet représentant une partie du document

# Annexe A (2)

**fixture:** une fixture de test est un objet fictif ou un environnement utilisé pour tester de façon constante un élément, un périphérique ou un morceau de logiciel

**framework:** Fournit un environnement pour exécuter des scripts de test automatisés; y compris les outils, les bibliothèques et les fixtures

**fonction:** Une fonction Python est un groupe d'instructions réutilisables qui effectuent une tâche spécifique.

**hook:** Une interface qui est introduite dans un système qui est créé principalement pour fournir une meilleure testabilité à ce système

**HTML (HyperText Markup Language):** Le langage de balises standard pour la création de pages Web et d'applications Web

**ID:** Attribut qui spécifie une chaîne d'identification unique pour un élément HTML. La valeur doit être unique dans le document HTML

**iframe:** Un cadre en ligne HTML, utilisé pour incorporer un autre document dans un document HTML

# Annexe A (3)

**dialogue modal:** Un écran ou une boîte qui force l'utilisateur à interagir avec elle avant qu'ils puissent accéder à l'écran sous-jacent

**Page Object Pattern:** Un modèle d'automatisation de test qui exige que la logique technique et la logique métier soient traitées à différents niveaux

**paradoxe des pesticides:** Un phénomène dans lequel répéter le même test plusieurs fois amène à trouver moins de défauts

**persona:** A user profile created to represent a user type that interacts with a system in a common way

**pytest:** A Python testing framework

**tag:** Les éléments HTML sont délimités par des balises, écrites à l'aide de crochets <>

**dette technique:** Implique le coût supplémentaire de la refonte causée par le choix d'ignorer les mauvaises conceptions ou les implémentations à court terme

# Annexe A (4)

**WebDriver:** L'interface dans laquelle les tests de Selenium sont écrits. Différents navigateurs peuvent être contrôlés via différentes classes Java, par exemple, ChromeDriver, FirefoxDriver, etc.

**wrapper:** Une fonction dans une bibliothèque logicielle dont le but principal est d'appeler une autre fonction, souvent en ajoutant ou en améliorant la fonctionnalité tout en masquant la complexité

**XML (eXtensible Markup Language):** Un langage de balises qui définit un ensemble de règles pour coder des documents dans un format à la fois lisible par l'homme et la machine

**XPath (XML Path Language):** Un langage de requête pour sélectionner des nœuds à partir d'un document XML