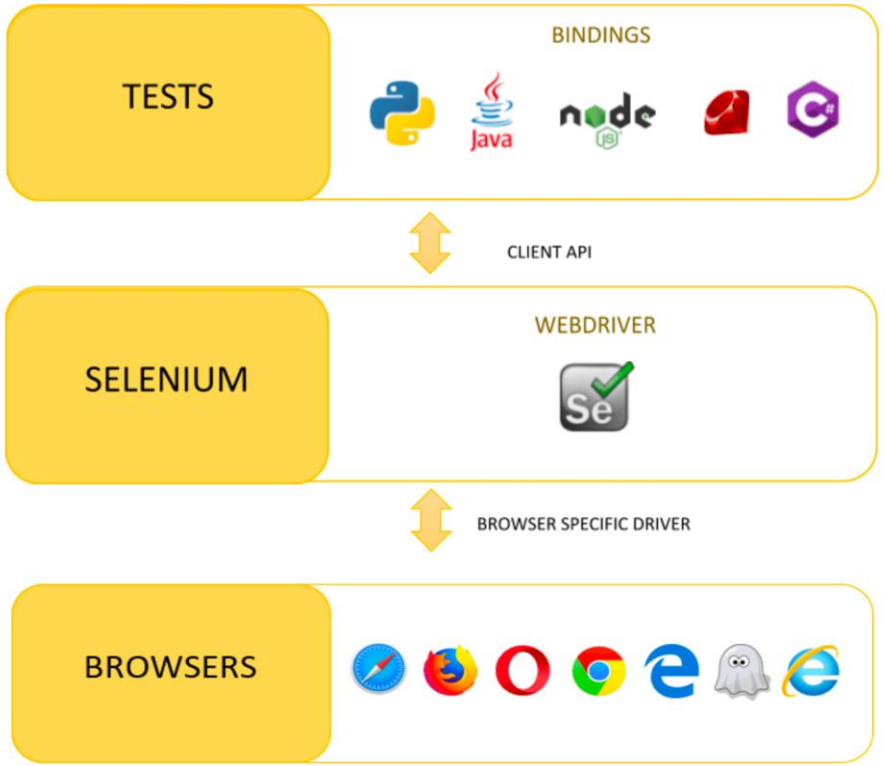
Wstęp do Selenium

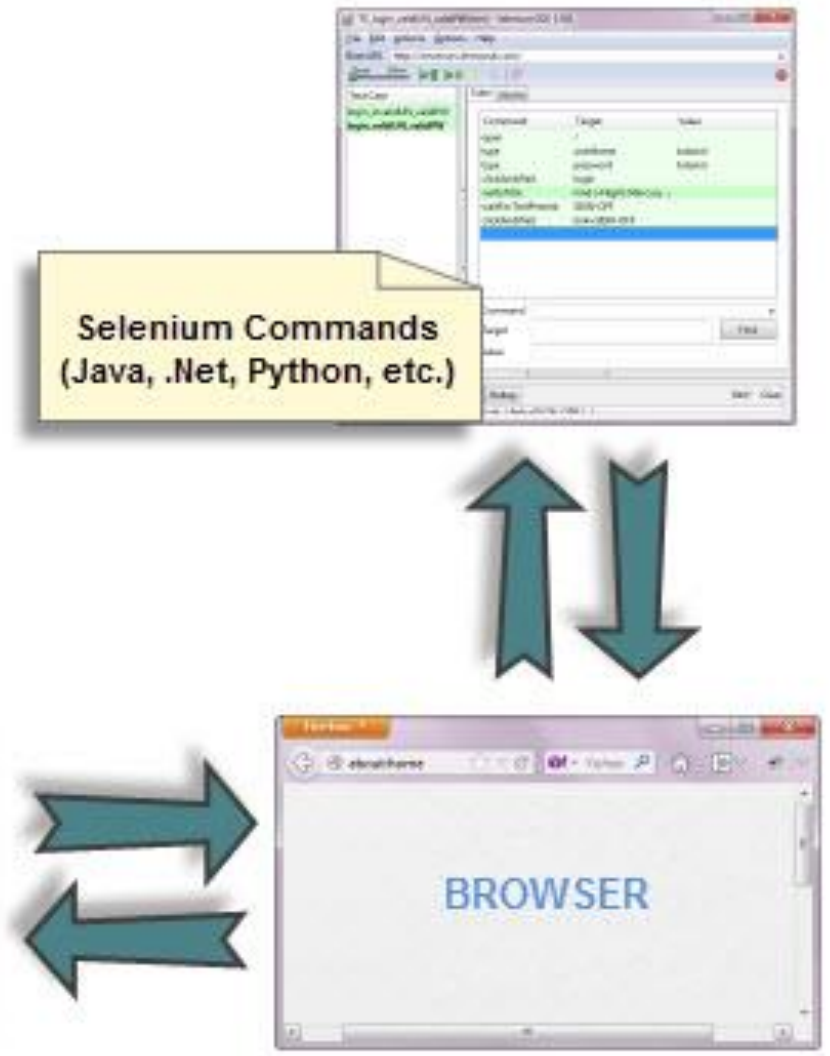
Selenium...

- Selenium jest niezależnym językowo (Python, Java, C#, etc...) frameworkiem do testowania i automatyzacji zadań webowych.
- Poza testowaniem, Selenium można z powodzeniem wykorzystać do napisania prostego bota (szybkie logowanie i wypełnianie arkusza? Dlaczego nie!)
- Łącznikiem pomiędzy kodem a wykonywanymi akcjami na stronach internetowych jest Webdriver – zestaw specyficznych sterowników dla najpopularniejszych przeglądarek (każda ma własny).



Webdriver

- WebDriver to narzędzie do testowania aplikacji internetowych w różnych przeglądarkach przy użyciu różnych języków programowania.
- Obsługuje głównie przeglądarki takie jak Firefox, Chrome, Safari i Internet Explorer.
- WebDriver mówi bezpośrednio do przeglądarki i używa jej własnego silnika do sterowania nią.
- Kontroluje przeglądarkę z poziomu systemu operacyjnego.
- Do działania potrzebujemy IDE do kodzenia z biblioteką Selenium oraz przeglądarkę.



Instalacja

- Pobierz sterowniki dla wybranej przez nas przeglądarki (uwaga: istotna jest również wersja browsera!) : https://www.selenium.dev/selenium/docs/api/py/index.html#drivers
- Umieść ścieżkę do drivera w zmiennych środowiskowych, tak by był do niego dostęp z każdego miejsca w systemie.
- Pobrać bibliotekę selenium do Pythona: **pip install selenium**.
- Celem upewnienia się, że wszystko przebiegło sprawnie, uruchom przeglądarkę za pomocą selenium (więcej na następnym slajdzie).

Pierwsze uruchomienie

- Wykonaj poniższy kod w terminalu, by sprawdzić czy wszystko działa:

```
C:\>python
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from selenium import webdriver
>>> driver = webdriver.Chrome()
```

- Uruchomić powinno się okno przeglądarki Google Chrome (musi być takowa wcześniej zainstalowana), z adnotacją, że jest ono sterowane poprzez nasz software.

- Aby zamknąć okno, należy wpisać:

```
DevTools listening on ws://127.0.0.1:57580/devtools/browser/23de1dfd-dd4f-411e-8203-93d0898074e2
>>> driver.close()
>>>
```

Powtórka z testowania

- Poziomy i typy testów.

Powtórka z testowania

- Poziomy i typy testów.
- Bibilioteka unittest.

Powtórka z testowania

- Poziomy i typy testów.
- Bibilioteka unittest.
- setUp i tearDown

Powtórka z testowania

- Poziomy i typy testów.
- Bibilioteka unittest.
- setUp i tearDown
- setUpClass i tearDownClass

Powtórka z testowania

- Poziomy i typy testów.
- Bibilioteka unittest.
- setUp i tearDown
- setUpClass i tearDownClass
- Asercje.

Powtórka z testowania

- Poziomy i typy testów.
- Bibilioteka unittest.
- setUp i tearDown
- setUpClass i tearDownClass
- Asercje.
- Pomijanie testów.

Selenium...

- Wykorzystując odpowiednie funkcjonalności selenium, jesteśmy w stanie wykonać w zasadzie każdą akcję na stronie internetowej w sposób automatyczny.
- Kod obok najpierw otwiera przeglądarkę i przechodzi na stronę python.org.
- W kolejnych linijkach element o nazwie „q” w HTML, zostaje zlokalizowany (jest to pole tekstowe).
- Za pomocą metody **clear()** na znalezionym elemencie czyścimy pole tekstowe, a następnie wpisujemy do niego tekst „pycon”.
- **Keys.RETURN** symbolizuje wciśnięcie klawisza ENTER.
- Po 5 sekundach następuje zamknięcie przeglądarki.

```
import time
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

driver = webdriver.Chrome()
driver.get("http://www.python.org")
print(driver.title)
elem = driver.find_element_by_name("q")
elem.clear()
elem.send_keys("pycon")
elem.send_keys(Keys.RETURN)
print(driver.page_source)
time.sleep(5)
driver.close()
```

Zadanie – google.com

Napisz test, sprawdzający działanie wyszukiwarki Google.

Uruchom automatycznie przeglądarkę.

Przejdź do strony www.google.com.

Zlokalizuj pole tekstowe i wpisz w nie słowo „Python”. Potwierdź wyszukanie.

Sprawdź czy widoczny jest link do wikipedii.

Lokalizowanie elementów

Pisanie testów selenium polega najczęściej na potokowej lokalizacji kolejnych obiektów na stronie i wykonywanie na nich przeróżnych akcji (kliknij przycisk, wypełnij pole tekstowe, zaznacz checkbox, sprawdź labelkę, kolor, itd.). Całość polega na operowaniu na znacznikach HTML i stylach CSS. Element możemy zlokalizować na różne sposoby, wykorzystując jedną z poniższych funkcji:

- *find_element_by_id*
- *find_element_by_name*
- *find_element_by_xpath*
- *find_element_by_link_text*
- *find_element_by_partial_link_text*
- *find_element_by_tag_name*
- *find_element_by_class_name*
- *find_element_by_css_selector*

ZADANIE – tworzenie nowego konta

Napisz test, który przetestuje możliwość utworzenia nowego konta w bazie danych. Wejdź na: **http://thedemosite.co.uk/addauser.php**, przeanalizuj kontent strony i utwórz użytkownika (różne długości username'a). Rozpatrz dwa przypadki. Skąd wiemy, że się udało?

ZADANIE – logowanie

Wykorzystując utworzone konta, spróbuj się zalogować automatycznie wchodząc na stronę: **http://thedemosite.co.uk/login.php**.

Sprawdź dwa przypadki: udane logowanie i nieudane logowanie – skąd wiemy, że się udało?

Co jeśli jakiś element wczytuje się długo?

Obecnie większość aplikacji internetowych korzysta z technik AJAX. Gdy strona jest ładowana przez przeglądarkę, elementy na tej stronie mogą ładować się w różnych odstępach czasu. Utrudnia to lokalizowanie elementów: jeśli element nie jest jeszcze obecny w DOM, funkcja **locate** zgłosi wyjątek **ElementNotVisibleException**. Korzystając z czekania, możemy rozwiązać ten problem.

Dwa rodzaje waitów:

- **explicit waits** – czeka na spełnienie określonego warunku (np. do momentu, kiedy będzie można kliknąć na jakiś przycisk, ale np. nie dłużej niż 5 sekund),
- **implicit waits** – czeka maksymalnie określoną ilość sekund przy każdej lokalizacji elementu.

O czym warto poczytać?

- xpath
- CSS Selectors
- wzorzec stron – page object model
- ActionChains
- Selenium IDE
- Selenium Remote
- Selenium Grid

Automatyzacja prostych zadań systemowych

subprocess

Moduł subprocess umożliwia tworzenie nowych procesów, łączenie się z ich potokami wejścia/wyjścia/błędu i uzyskiwanie ich kodów wynikowych.

subprocess implementuje tylko jedną klasę: **Popen**. Podstawowym zastosowaniem tej klasy jest tworzenie nowego procesu w systemie. Do konstruktora przyjmuje ona kilka argumentów (polecam dokumentację):

- *args* – string lub sekwencja argumentów do programu.
- *shell* – jeśli ustawione na True, komenda będzie uruchomiona przez powłokę systemową (jakby było odtwarzane z terminala)
- *cwd* – ustawia katalog, z którego polecenia ma zostać uruchomione.

```
>>> import subprocess
>>> task = subprocess.Popen("ping 127.0.0.1")
>>>
Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

subprocess

```
>>> task = subprocess.Popen(["ping", "127.0.0.1", "-n", "3"])
>>>
Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 3, Received = 3, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

- Możemy poprzednią komendę wywołać używając listy stringów zamiast napisu. Dzięki temu możemy podawać dodatkowe argumenty jako kolejne elementy w liście.
- Uwaga, nie zadziała to, jeśli będziemy chcieli zapisać całą komendę w pojedynczym elemencie listy.

```
>>> task = subprocess.Popen(["ping 127.0.0.1 -n 3"])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Users\Lukasz_Paluch\AppData\Local\Programs\Python\Python38-32\lib\subprocess.py", line 854, in __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
  File "C:\Users\Lukasz_Paluch\AppData\Local\Programs\Python\Python38-32\lib\subprocess.py", line 1307, in _execute_child
    hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
FileNotFoundError: [WinError 2] The system cannot find the file specified
```

subprocess

- Klasa **Popen** jest wykorzystywana do tworzenia bardziej zaawansowanych komend (dokumentacja!). Do prostych przypadków wystarczy jednak zwykła metoda **run** z modułu **subprocess**.

- Różnica polega na tym, że **run** uruchamia zadanie i czeka aż się ono zakończy zwracając obiekt **subprocess.CompletedProcess**, który przechowuje informację o zwróconym kodzie operacji (0 zazwyczaj oznacza, że wszystko skończyło się pomyślnie, bez błędu). **Popen** nie czeka, więc możemy w międzyczasie wykonywać inne instrukcje i odpytywać obiekt **Popen** o zakończenie obliczeń metodą **communicate**.

```
>>> subprocess.run("ping 127.0.0.1")

Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
CompletedProcess(args='ping 127.0.0.1', returncode=0)
```

os

Ten moduł zapewnia przenośny sposób korzystania z funkcji zależnych od systemu operacyjnego. Oto kilka przykładów:

- *os.name - nazwa zaimportowanego modułu zależnego od systemu operacyjnego.*
- *os.environ – obiekt mapujący zmienne środowiskowe na typ string.*
- *os.getenv(key, defautl=None) - zwróci wartość klucza zmiennej środowiskowej, jeśli istnieje, lub domyślną, jeśli nie.*
- *os.getcwd() – zwróci ścieżkę do aktualnego katalogu*
- *os.getlogin() - zwróci nazwę użytkownika zalogowanego na terminalu sterującym procesem*
- *os.get_terminal_size(fd=STDOUT_FILENO) – zwróci wymiary terminala*

os świetnie nadaje się do przeglądania grup użytkowników, przyznawanie i zmianę uprawnień i zarządzaniem plikami.

os – cd.

- *os.listdir(path='.') - zwraca listę zawierającą nazwy wpisów w katalogu podanym przez ścieżkę.*
- *os.mkdir(path, ...) – tworzy nowy katalog pod podaną ścieżką.*
- *os.chdir(path) – przechodzi do innego katalogu.*
- *os.remove(path, ...) – usuwa plik (przy katalogu zwraca wyjątek)*
- *os.rmdir(path) – usuwa katalog podany w ścieżce (musi być pusty!)*
- *os.walk(top, topdown=True, ...) - generuje nazwy plików w drzewie katalogów, chodząc po folderach z góry na dół lub z dołu do góry*

os

Oprócz głównego modułu, mamy również podmoduł *os.path*, który dodaje funkcjonalność operowania na ścieżkach do plików.

- *os.path.exists(path) – sprawdza czy ścieżka istnieje.*
- *os.path.dirname(path) – zwraca ścieżkę do katalogu, w którym znajduje się końcowy plik/folder.*
- *os.path.isdir(path) – sprawdza czy ścieżka wskazuje na katalog.*
- *os.path.isfile(path) – sprawdza czy ścieżka wskazuje na plik.*
- *os.path.join(path, *paths) – łączy części ścieżek w całość (stawiając odpowiednie slashe)*
- *os.path.split(path) - podzieli ścieżkę na parę (head, tail), gdzie tail jest ostatnim składnikiem ścieżki, a head jest wszystkim, co do tego prowadzi (z reguły plikiem).*
- *os.path.splitext(path) – podzieli ścieżkę na parę (root, ext), gdzie ext jest rozszerzeniem pliku lub pustym stringiem, jeśli ostatni element to katalog.*

sys

Moduł ten zapewnia dostęp do niektórych zmiennych używanych lub obsługiwanych przez tłumacza oraz do funkcji, które silnie oddziałują z interpreterem. Jest to, podobnie jak **subprocess** i **os**, biblioteka wbudowana. Oto przykłady funkcji i atrybutów:

- *sys.exc_info() - Ta funkcja zwraca krotkę zawierającą trzy wartości, które zawierają informacje o obecnie obsługiwanym wyjątku*
- *sys.exit() – wychodzi z Pythona (programu).*
- *sys.getrefcount(object) – zwraca ilość referencji do obiektu.*
- *sys.getrecursionlimit() | sys.setrecursionlimit() – funkcje pozwalające sprawdzić i ustawić*
- *sys.getsizeof(object) – zwraca rozmiar obiektu w bajtach.*
- *sys.path - Lista stringów określająca ścieżkę wyszukiwania modułów.*
- *sys.argv - Lista argumentów wiersza poleceń przekazanych do skryptu w języku Python*

sys

- Dzięki bibliotece **sys** możemy pozwolić użytkownikowi na podanie wartości z zewnątrz programu, podając je po spacji za nazwą pliku.

```
python argv.py arg1 arg2 12345
```

- sys.argv przechowuje listę argumentów przekazanych do pliku (pierwszy element jest nazwą pliku).
- wszystkie wartości z sys.argv są stringami, nawet jeśli podana zostanie liczba (trzeba pamiętać o rzutowaniu!)
- Można napisać prosty skrypt bashowy realizujący podstawienie odpowiednich argumentów do wywołania pliku wyciągając te wartości z wcześniej przygotowanego pliku konfiguracyjnego.

```python
import sys


def print_all_params(*params):
    print(*params)


if __name__ == '__main__':
    print(f'Nazwa pliku: {sys.argv[0]}')
    print_all_params(sys.argv[1:])
```

psutil

psutil to wieloplatformowa biblioteka Pythona używana do uzyskiwania dostępu do szczegółów systemu i narzędzi procesowych. Można monitorować wykorzystanie zasobów, takich jak procesor, pamięć, dyski, sieć, czujniki. Testy na wykorzystanie zasobów? Czemu nie! **psutil** nie jest biblioteką wbudowaną, trzeba go pobrać z PyPI komendą **pip install psutil**.

- *psutil.cpu_percent() - Ta funkcja oblicza bieżące wykorzystanie procesora w całym systemie jako wartość procentową.*
- *psutil.cpu_count() - Ta funkcja pokazuje liczbę logicznych procesorów w systemie.*
- *psutil.virtual_memory() - Ta funkcja podaje użycie pamięci systemowej w bajtach.*
- *psutil.disk_partitions() - Ta funkcja udostępnia szczegóły wszystkich zamontowanych partycji dyskowych w postaci listy krotek, w tym urządzenia czy zamontowane punkty*
- *psutil.sensors_battery() –zwraca informację o stanie baterii (w tym czy jest aktualnie ładowana)*
- *psutil.users() – zwraca listę użytkowników połączonych z systemem.*

Zadanie

Napisz skrypt, który będzie wymagał od użytkownika podania 3 argumentów w momencie wywoływania (z terminala):

- komendy systemowej (np. ping, timeout, shutdown)
- ścieżki do pliku
- nazwy zmiennej środowiskowej.

Skrypt powinien sprawdzić jakiego typu jest plik, do którego ścieżkę podano (tekstowy, pythonowy, etc). Jeśli jest to plik pythonowy, skrypt powinien go uruchomić. Jeśli tekstowy, powinien wypisać pierwszą linijkę tesktu. Jeśli plik nie istnieje, powinien zostać wychwycony wyjątek (sprawdź dokładnie jaki) oraz wypisane wszystkie informacje na jego temat. W przypadku udanego wykonania zadania, skrypt ustawi flage file_flag na True.

Następnie skrypt powinien sprawdzić czy istnieje zmienna środowiskowa o nazwie podanej przez użytkownika. Jeśli tak, to powinien pobrać jej wartość, jeśli nie – powinien ją utworzyć i przypisać jej wartość „DONE” i ustawi flagę env_flag na True.

Jeżeli obydwie flagi są poprawne, skrypt wyciągnie informację na temat nazwy systemu i użytkownika i poinformuje go, że za moment na tym systemi zostanie uruchomiona komenda, którą podał jako wstęp do skryptu (po czym – jeżeli komputer aktualnie się ładuje - uruchomi ją).