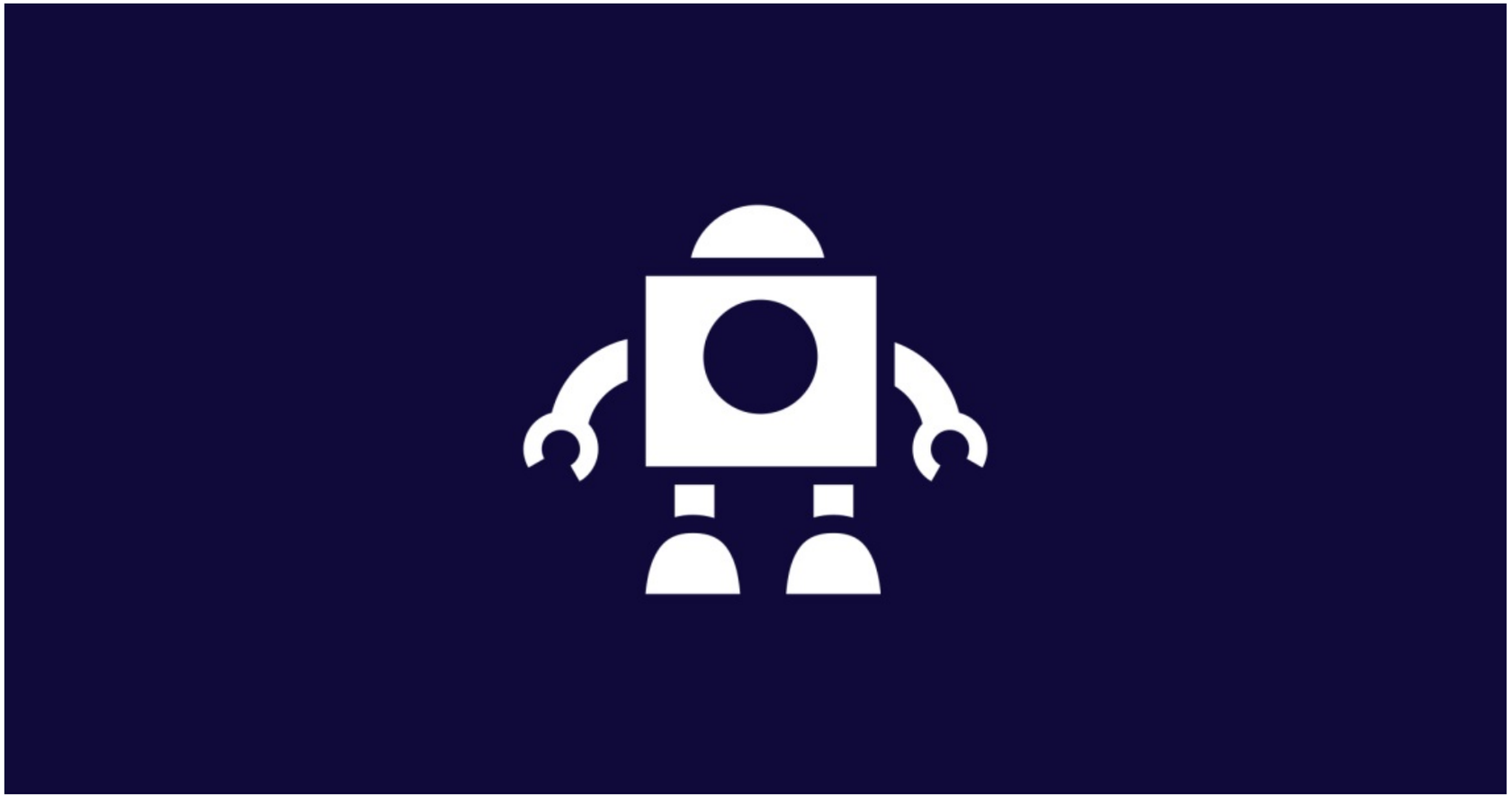

Jak oswoić Bota? – Clicker (sheadovas/poradniki/howto/jak-oswoic-bota-clicker/)

Lis 04, 2017 / howTo (sheadovas/category/poradniki/howto/)

Jak napisać własnego bota w Pythonie.

Witam Was pierwszym z dwóch artykułów dotyczących pisania botów, gdzie pokażę jak w łatwy sposób można napisać swojego pierwszego bota w Pythonie.

Uwaga na boku: Wpis ten powstał, ponieważ ostatnio widzę coraz większe zainteresowanie pisania botów, a że sam temat wbrew pozorom nie jest wcale trudny, lecz ciekawy i przyjemny to postanowiłem dodać kilka słów od siebie. Zapraszam do lektury.

A po co to komu?

Boty mogą służyć w wielu celach, najbardziej ogólnym jest szeroko rozumiana automatyzacja, która może przydać się zarówno w pisaniu testów jednostkowych (gdzie bez bota tester musiałby zweryfikować, czy po kliknięciu w przycisk aplikacji coś się działo, to tutaj może mu przyjść z pomocą własnoręcznie napisany bot), jak i taki bot może posłużyć do mniej szlachetnych zamiarów (np klikanie w grach, ...) ale my na nich nie będziemy się skupiać.

> Z racji, że ten sam kod można wykorzystać do „złych” celów, to autor nie wpisu nie ponosi odpowiedzialności, za wykorzystanie poniższego kodu do „złych” celów,

Dwie szkoły botów

Na wstępie chciałbym z kategoryzować boty na dwie główne „szkoły”, które charakteryzują się sposobem działania (wszystko w kontekście aplikacji webowych):

1. **Clicker**
2. **Requester**

> Nie traktujcie tych nazw jako jakieś specjalistyczne definicje – są one wymyślone przeze mnie na potrzeby tego artykułu.

Clicker

Clicker, a więc „klikacz” charakteryzuje się, tym że robi dokładnie to co mu powiemy: klika w ściśle określone miejsca, wciska odpowiednie klawisza w ściśle zdefiniowanej kolejności – nie zastanawia się (w prymitywnej wersji), czy kursor jest nad klikalnym obiektem, czy też może jednak należałoby przesunąć kursor o te 5px w lewo aby trafić w guzik. Jego działanie opiera się na otworzeniu wcześniej „nagranych” czynności.

Jego zaletą jest łatwość adaptacji do wielu sytuacji oraz łatwość stworzenia (prymitywna wersja tworzy się bardzo szybko), z kolei sporą wadą jest to, że operuje on na ekranie, tj. jeżeli aplikacja zajmuje cały ekran, to bot potrzebuje całego ekranu do działania. Nie jest też możliwe uruchomienie wielu instancji botów jednocześnie (nawet różnych), bo wszystkie one operują na tym samym sprzęcie (ekran, klawiatura, kursor, aktywne okno) i mogłoby dochodzić do konfliktów, ba! Użytkowanie w tym czasie sprzętu na którym uruchomiony jest bot nie jest pożądane (ten sam problem co z wieloma instancjami botów).

Requester

Requester jest odwrotnością Clickera: jego stworzenie wymaga więcej wysiłku, nie operujemy tutaj „na tym co widoczne”, ale używamy mechanizmów kryjących się pod maską webaplikacji, a więc zapytań (GET, POST). Tutaj bardziej emulujemy interakcje pochodzące od użytkownika (w wyniku np wciśnięcia klawisza), przez co takie rozwiązanie potrafi być dużo bardziej elastyczne i wydajne.

Sporą zaletą tego rozwiązania jest to, że taki bot może z powodzeniem działać w tle, nie wymaga dostępu do kontroli nad kursorem, klawiaturą i myszą. Może działać w większej ilości instancji. Wadą tak jak wspomniałem jest to, że musimy spędzić więcej czasu na analizie wysyłanych zapytań do serwera / aplikacji. Inną wadą jest to, że nie nadaje się do wszystkich aplikacji: jeżeli czegoś nie jesteśmy w stanie wysłać jako zapytania / emulować wewnątrz bota, to tutaj wygrywa Clicker.

Hybryda

Nikt też nie zabroni nam stworzyć hybrydy obydwu powyższych rozwiązań, łączącej zalety obydwu rozwiązań przez co eliminujemy część wad.

„Oswajamy” bota ;)

W łamach tej mikro-serii przedstawię dwa pierwsze typy botów.

Wstęp fabularny dla obydwu botów jest identyczny: chcemy napisać bota, który przetestuje działanie wyszukiwarki. Na pierwszy ogień pójdzie *Clicker*.

Clicker

Naszym pierwszym celem jest napisanie bota w Pythonie, który ma za zadanie zweryfikować czy wpisanie tekstu do wyszukiwarki daje jakiś wynik. Jego sposób działania opiera się na krokach:

1. Stworzeniu screenshota przed rozpoczęciem pracy
2. Wybraniu frazy, którą chcemy wyszukać
3. Znalezieniu pola do wyszukiwania
4. Aktywowaniu go

5. Wpisaniu frazy wyszukiwania
6. Wciśnięciu klawisza „Enter”, w celu rozpoczęcia wyszukiwania
7. Zrobieniu kolejnego zrzutu, już po wyszukaniu danej frazy.

Przygotowanie środowiska

Ja w swoim przykładzie będę używał Pythona 2.7 oraz biblioteki pyautogui (https://pyautogui.readthedocs.io/en/latest/index.html), która zawiera wszystko czego potrzebujemy do napisania bota. Jeżeli jesteś użytkownikiem Linuksa bazującym na Debianie, to proces instalacji wygląda następująco:

Instalacja PyAutogui (Linuks) — Shell

```
$ sudo pip install python-xlib
$ sudo apt-get install scrot python-tk python-dev -y
$ sudo pip install pyautogui
```

Jeżeli jesteś użytkownikiem innego systemu to odsyłam do Wiki (https://pyautogui.readthedocs.io/en/latest/install.html) pokazującego jak zainstalować poprawnie bibliotekę na swoim systemie.

Kolejnym krokiem jest stworzenie screenshota przedstawiającego pasek wyszukiwania, w który będziemy chcieli kliknąć i zapisaniu go do folderu projektu jako „search_bar.png”.

(https://i0.wp.com/szymonsiarkiewicz.pl/wp-content/uploads/2017/11/search_bar.png)
search_bar.png

Piszemy bota!

Najpierw importy, chcemy zaimportować *pyautogui*, ale chcemy także losować jedną z wybranych fraz do wyszukiwania (aby nie było nudno!). Na pewno chcemy też dać chwilę botowi na wykonanie akcji, więc potrzebujemy też sleep’a:

Importy i podstawowe zmienne — Python

```python
import pyautogui
from random import randrange
from time import sleep


def main():
    search_bar_img = "search_bar.png"
    words = ["pyautogui", "pasja-informatyki", "szymon siarkiewicz blog"]
    word = words[randrange(0, len(words))]
```

W tej chwili okaże się po co nam był obrazek wzorcowy paska wyszukiwania, otóż *pyautogui* posiada funkcję

`locateOnScreen`

, która zwraca pozycję obiektu przekazanego jako obrazek w argumencie:

Python

```python
    loc = pyautogui.locateOnScreen(search_bar_img)

    if loc is not None:
        print loc

        x = loc[0]
        y = loc[1]

        width = loc[2]
        height = loc[3]
```

W przypadku gdy może istnieć więcej niż jeden wzorcowy obrazek na ekranie, to warto rozważyć zastosowanie funkcji

`locateAllOnScreen`

, która zwraca listę wszystkich znalezionych obiektów na ekranie.

Chciałbym też zauważyć, że *pyautogui* nie zawsze potrafi odnaleźć przekazany obiekt jako obrazek, dlatego tutaj warto zastanowić się jednak, czy potrzebujemy auto-detekcji i czy nie lepiej przed rozpoczęciem jego pracy by mu nie nagrać odpowiednich pozycji na ekranie, które musiałby później odtworzyć (takie coś łatwo jest napisać samemu i zostawiam to jako ćwiczenie do samodzielnego wykonania).

Kolejnym krokiem jest kliknięcie w pasek wyszukiwania w celu jego (uwaga: staropolskie słowo) „sfokusowania”, a klikamy nie byle gdzie, bo na jego środek:

Python

```
        # click center of the search bar
        pyautogui.click(x + (width / 2), y + (height / 2))
        sleep(1)
```

Następnie „wpisujemy” to co chcemy wyszukać przy użyciu klawiatury i klikamy enter, aby rozpocząć proces wyszukania:

Python

```
        # enter what are we looking for
        pyautogui.typewrite(word)
        sleep(1)

        # press enter for search
        pyautogui.press('enter')
```

Ostatnim krokiem jest reakcja na sytuację, w której nie udało się odnaleźć elementu wzorcowego (niestety zdarza się to dość często):

Python

```
    else:
        print "Cannot find selected picture :("
```

Pewnie zauważyliście, że w powyższym kodzie brakuje weryfikacji testu poprzez robienie zrzutów ekranów, spokojnie – jest na to sposób – służy do tego funkcja

`screenshot()`

, a dodanie tej funkcjonalności pozostawiam Wam w ramach prostego ćwiczenia ;)

Pełny kod pozostawiam poniżej:

Python

```python
import pyautogui
from random import randrange
from time import sleep


def main():
    search_bar_img = "search_bar.png"
    words = ["pyautogui", "pasja-informatyki", "szymon siarkiewicz blog"]
    word = words[randrange(0, len(words))]
    loc = pyautogui.locateOnScreen(search_bar_img)

    if loc is not None:
        print loc

        x = loc[0]
        y = loc[1]

        width = loc[2]
        height = loc[3]

        # click center of the search bar
        pyautogui.click(x + (width / 2), y + (height / 2))
        sleep(1)

        # enter what are we looking for
        pyautogui.typewrite(word)
        sleep(1)

        # press enter for search
        pyautogui.press('enter')

    else:
        print "Cannot find selected picture :("
```

Upgrade! (czyli co można ulepszyć i w jaki sposób)

Jeżeli chwilę pobawicie się ze wspomnianą przeze mną biblioteką to zauważycie, że naszemu botowi przydałoby się kilka ulepszeń. Poniżej przedstawiam kilka feature'o-ćwiczeń do samodzielnego wykonania (jeżeli chcecie, to możecie podzielić się rozwiązaniami w postaciami linków do repo w komentarzach).

1. Lepsze wyszukiwanie obrazka:
   - Zamiast z `locateOnScreen` warto by było skorzystać z innych specjalizujących się w tym bibliotek,
   - Caching znalezionych pozycji – jeżeli raz znaleźliśmy pozycję danego obiektu, to zapiszmy ją sobie gdzieś i korzystajmy z niej przy przyszłych uruchomieniach.
2. Nagrywanie sekwencji kliknięć do pliku i ich odtwarzanie (to o czym wspomniałem w artykule).
3. Wywołanie przy użyciu opcji / switch'y z command-line'a (patrz: argparse (https://docs.python.org/3/library/argparse.html)).

Podsumowanie

Jeżeli chodzi o typ bota „Clicker", to tyle co chciałem Wam przekazać, zachęcam do wykonania ćwiczeń jeżeli interesujecie się tematyką botów.

Dajcie znać co sądzicie o tym artykule w komentarzach, nie zapomnijcie także śledzić bloga przez social-media (panel po prawej).

Zapraszam do kolejnego wpisu z tej mikro-serii, a także do lektury reszty bloga!

Code ON!