Programowanie
przed 14 wykładem

Andrzej Giniewicz

21.06.2024

Dziś zajmiemy się makrami Excela w języku Python, korzystającymi z biblioteki xlwings.

1 Instalacja

Biblioteka xlwings instaluje się razem z dystrybucją Anaconda, która jest polecana do zajęć. Jeśli masz zainstalowaną Anacondę, jedyne co trzeba zrobić, to zarejestrować dodatek do Excela zgodny z Anacondą i powiadomić go, gdzie znajduje się Python oraz biblioteka. Robimy to komendą

```
xlwings addin install
```

Zwróćmy uwagę, by program Excel był zamknięty podczas instalowania, aktualizowania lub usuwania dodatku.

Jeśli terminal zgłosi, że nie komendy xlwings nie ma w ścieżce, oznacza to, że nie dodaliśmy Anacondy do ścieżki podczas instalacji. Istnieją dwa sposoby na obejście tego problemu. Pierwszy to uruchomienie komendy `conda activate` w celu włączenia wirtualnego środowiska Anacondy i tym samym ustawienia odpowiednich ścieżek. Druga opcja to uruchomienie `Anaconda Command Prompt` w menu systemu operacyjnego zamiast terminala z BASH'em, który ma od razu ustawione odpowiednie ścieżki. W obu wariantach ponawiamy komendę z wcześniej i powinna zadziałać.

Osoby, które nie zainstalowały Anacondy, tylko używają innych dystrybucji Pythona, na przykład ze sklepu Microsoftu lub ze strony python.org, muszą ręcznie zainstalować bibliotekę xlwings. Mogą to zrobić według instrukcji na stronie projektu[1].

Dodatek raz zainstalowany w Excelu nie będzie się aktualizował automatycznie. Dodatkowo jest kompatybilny tylko z wersją xlwings, dla której został dodany. Oznacza to, że musimy go ponownie zainstalować po aktualizacji Pythona lub samej biblioteki.

[1] Patrz `https://docs.xlwings.org/en/stable/installation.html`.

2 Podstawy składni

Interfejs programistyczny w bibliotece xlwings jest zbudowany na podstawie interfejsu z VBA, więc znając jeden z nich, mamy szansę rozeznać się w obu. Ponieważ sporo o VBA już było powiedziane, a składnię samego Pythona wykorzystujemy na tym i innych kursach, pokażę kilka przykładów, jak dostać się do funkcjonalności potrzebnej w pisaniu własnego makra. Do wszystkich przykładów musimy zaimportować bibliotekę xlwings.

```
import xlwings as xw
```

2.1 Obiekty Excela

Musimy mieć dostęp do aplikacji, skoroszytu i arkusza. Istnieje możliwość na dostanie się do aktywnego arkusza

```
arkusz = xw.sheets.active
```

aktywnego skoroszytu

```
skoroszyt = xw.books.active
```

lub aktywnej aplikacji

```
aplikacja = xw.apps.active
```

Możemy też dostać się do aktywnego skoroszytu w konkretnej aplikacji

```
skoroszyt = aplikacja.books.active
```

albo aktywnego arkusza konkretnego skoroszytu

```
arkusz = skoroszyt.sheets.active
```

Istnieje również możliwość pobrania konkretnego arkusza po numerze

```
arkusz = skoroszyt.sheets[0]
```

lub nazwie

```
arkusz = skoroszyt.sheets["Arkusz1"]
```

albo konkretnego skoroszytu po numerze

```
skoroszyt = aplikacja.books[0]
```

lub nazwie

```
skoroszyt = aplikacja.books["Dane_vba.xlsm"]
```

Aby pobrać konkretną aplikację, potrzebujemy jest numeru procesu (PID), który sprawdzimy komendą `xw.apps.keys()`

```
aplikacja = xw.apps[1337]
```

Jeśli makro uruchamiamy w konkretnym skoroszycie, najczęściej wybieramy obecnie aktywny skoroszyt w obecnie aktywnej aplikacji i z jego poziomu odnosimy się do arkuszy za pomocą nazw, czyli

```
arkusz = xw.books.active.sheets["Arkusz1"]
```

Dysponując arkuszem, możemy odwołać się do zakresów w tym arkuszu, na przykład za pomocą

```
arkusz["A1"] == arkusz[0, 0]
arkusz["A1:C3"] == arkusz[:3, :3]
```

Zwróćmy uwagę, że indeksowanie w Pythonie jest od zera, więc numery wierszy mogą być przesunięte. Jeśli potrzebujemy indeksowania za pomocą wartości zaczynających się od 1, xlwings pozwala na to, dostarczając wariantów z nawiasami okrągłymi do indeksowania, na przykład

```
skoroszyt.sheets(1) == skoroszyt.sheets[0]
arkusz.range((1,1)) == arkusz[0, 0]
arkusz.range((1,1), (3,3)) == arkusz[:3, :3]
```

Mimo to jest to rzadziej spotykane indeksowanie w xlwings.

Pomiędzy obiektami jest hierarchia, mamy aplikację, wewnątrz skoroszyt, wewnątrz arkusz, wewnątrz zakres. Każdy obiekt pamięta swojego rodzica w tej hierarchii.

```
aplikacja = xw.apps.active
zakres = aplikacja.books.active.sheets["Arkusz1"].range("A1")
zakres.sheet.book.app == aplikacja
```

2.2 Typy danych

Biblioteka xlwings tłumaczy typy danych Excela na typy Pythona. Dla pojedynczych komórek są to `float`, `str`, `datetime` oraz `None` dla pustych komórek. Podobnie jak w VBA, aby dostać się do wartości w komórce, używamy atrybutu `value`.

```
import datetime as dt
arkusz["A1"].value = 1
arkusz["A2"].value = "Hello"
arkusz["A3"].value = None
arkusz["A4"].value = dt.datetime(2024, 6, 21)
```

Konwersja odbywa się automatycznie w obie strony.

Wstawienie wartości w liście spowoduje ułożenie ich w wierszu, czyli

```
arkusz["A1"].value = [1, 2, 3]
```

jest równoważne do

```
arkusz["A1"].value = 1
arkusz["B1"].value = 2
arkusz["C1"].value = 3
```

Jeśli chcemy wpisać kolumnę w pionie, musimy skorzystać z listy list, czyli

```
arkusz["A1"].value = [[1], [2], [3]]
```

jest równoważne do

```
arkusz["A1"].value = 1
arkusz["A2"].value = 2
arkusz["A3"].value = 3
```

Istnieje też możliwość wstawiania list dwuwymiarowych pod całe zakresy, wtedy zewnętrzna lista jest listą wierszy, a każda z wewnętrznych list jest listą elementów w wierszu.

Alternatywą dla używania list jest używanie tablic NumPy. Wstawienie tablicy NumPy pod komórkę jest automatycznie konwertowane. Aby wyciągnąć dane z zakresu do tablicy NumPy, musimy użyć odpowiednich opcji. Na przykład poniższy kod

```
arkusz[:10, :10].options(np.array).value
```

wyciągnie pierwszych 10 wierszy i pierwszych 10 kolumn do tablicy NumPy.

Jeśli nie znamy rozmiaru wydobywanej tablicy, możemy użyć rozszerzenia zaznaczenia, działającego jak `CurrentRegion` w VBA, czyli poszerzającego zaznaczenie najbardziej jak się da, aby w wyniku nie było pustych wierszy ani pustych kolumn.

```
arkusz["A1"].expand()
```

lub równoważnie

```
arkusz["A1"].options(expand="table")
```

co można połączyć z wyciąganiem tablicy NumPy

```
arkusz["A1"].options(np.array, expand="table").value
```

Dane możemy też wstawiać oraz wyciągać jako tabele Pandas.

```
import pandas as pd

arkusz["A1"].options(pd.DataFrame, expand="table").value
```

istnieje możliwość sterowania tym, czy tabela posiada indeks oraz nagłówek

```
arkusz["A1"].options(pd.DataFrame, expand="table", index=False, header=False).value
```

2.3 Makra

Domyślnie xlwings uruchamia funkcję `main` z pliku Pythona o tej samej nazwie, co arkusz. Istnieje możliwość zmiany w opcjach na wstążce xlwings, gdzie możemy podać nazwę pliku, funkcję oraz ścieżkę do Pythona. Na nasze potrzeby uruchamianie makra z jednej funkcji jest wystarczające, więc jeśli ktoś będzie potrzebował więcej kontroli, może też użyć dostarczanej przez xlwings funkcji Visual Basic for Applications o nazwie `RunPython`. Przykładowo może wyglądać ona następująco

```
Sub HelloWorld()
    RunPython "import hello; hello.world()"
End Sub
```

Dzięki temu jesteśmy w stanie zintegrować funkcje Pythona z resztą arkusza, na przykład z przyciskami lub skrótami klawiaturowymi.

2.4 Formuły

Formuły rejestrujemy za pomocą dekoratorów. Piszemy

```
@xw.func
def dodaj(x, y):
    return x+y
```

po czym możemy użyć tej funkcji jako formuły

```
=dodaj(A1, A2)
```

Jeśli chcemy skorzystać z formuł działających na wektorach, powinniśmy poprosić xlwings o traktowanie danych zawsze jako obiekt dwuwymiarowy, aby działało tak samo dla zaznaczenie jednej komórki, wiersza oraz zakresu.

```
@xw.func
@xw.arg('dane', ndim=2)
def pierwiastek(dane):
    return [[komórka**0.5 for komórka in wiersz] for wiersz in dane]
```

Formuły również dają możliwość korzystania z tablic NumPy. Dzięki temu możemy łatwo zaimplementować mnożenie macierzy

```
@xw.func
@xw.arg('A', np.array, ndim=2)
@xw.arg('B', np.array, ndim=2)
def mat_mul(A, B):
    return A @ B
```

czego użyjemy na przykład następująco

```
=mat_mul(A1:C3, D1:F3)
```

2.5 Wykresy

Istnieje możliwość generowania wykresów. Każdy wykres wygenerowany w bibliotece Matplotlib może zostać dodany do arkusza jako wykres.

```
import matplotlib.pyplot as plt

figure = plt.figure()
plt.plot(...)  # dowolny wykres

arkusz.pictures.add(figure, name='Wykres', update=True)
```

Opcja `update` powoduje, że gdy dodamy nowy wykres o tej samej nazwie (`name='Wykres'`), to wykres zostanie zaktualizowany, czyli podmieniony z uwzględnieniem ewentualnych zmian w położeniu i skali, które zastosowaliśmy w arkuszu.

Jeśli chcemy, możemy zamknąć wykres w formule i użyć go w arkuszu, aby uzyskać modyfikowalny wykres zależny od pól w arkuszu. W tym przypadku potrzebujemy dodatkowej opcji `caller`, która jeśli jest dodana do makra, ma automatycznie wstawiony obiekt odpowiadający miejscu wywołania.

```
@xw.func
def rysuj(n, caller):
    figure = plt.figure()
    plt.plot(...)  # dowolny wykres, zależny od n

    caller.sheet.pictures.add(fig, name='Wykres', update=True)
    return f'Wykres z parametrem n={n}'
```

Makra używamy na przykład jako

```
=rysuj(A1)
```

Ponieważ biblioteka Matplotlib pozwala na wykonanie wielu różnych wizualizacji[2], dzięki xlwings wszystkie te możliwości mamy w Excelu.

3 Co dalej?

To tylko część możliwości darmowej wersji biblioteki xlwings. Jeśli ktoś jest zainteresowany, może rzucić okiem na dokumentację[3], gdzie znajdzie więcej funkcjonalności darmowej wersji oraz informacje o wersji PRO, często używanej przez różnego rodzaju firmy. Spróbuj przeanalizować kod makra z zadania[4], i porównaj go z kodem VBA. Czy uda Ci się go uruchomić? Sprawdź w dokumentacji znaczenie używanych metod i obiektów, jeśli któregoś nie rozumiesz.

[2] Patrz `https://matplotlib.org/stable/gallery/index.html`.
[3] Patrz `https://docs.xlwings.org/en/stable/index.html`.
[4] Patrz `http://prac.im.pwr.edu.pl/~giniew/doku.php?id=rok2324:letni:prog:excel`.

Od niedawna Microsoft pracuje nad swoją własną integracją Pythona i Excela[5], jednakże to rozwiązanie jest obecnie dostępne tylko dla testerów przyszłych wersji programu Excel w konkretnych wersjach oraz cechuje się pewnym ograniczeniem — kod programu uruchamiany jest w chmurze, nie jak w przypadku xlwings, na komputerze użytkownika. Ma to swoje wady i zalety. Z jednej strony nie wymaga instalacji Pythona na komputerze, z drugiej strony nasz kod oraz dane wędrują do chmury, co nie zawsze jest możliwe, na przykład w sytuacji przetwarzania danych osobowych w instytucjach finansowych.

4 Podsumowanie

Tym wykładem kończymy blok o arkuszach kalkulacyjnych. Zachęcam, by się z nimi porządnie zaprzyjaźnić, ponieważ często stanowią pierwsze narzędzie do kontaktu z klientem i niekiedy wystarczają, aby wykonać potrzebne analizy. Dodatkowo ucząc się VBA, mamy możliwość używania innych programów wykorzystujących go jako język makr. Nie ma ich dużo, ale możliwe, że na studiach traficie na jeszcze jeden — pakiet statystyczny Statistica, mający niezerową popularność wśród medyków. Samo VBA można wykorzystać w innych programach pakietu Office, ale też na przykład AutoCAD, SolidWorks lub ArcGIS. Równocześnie pamiętajmy, że choć jest to język mający swoje zastosowania, jest dość archaiczny, zatem jeśli mamy możliwość wyboru środowiska makr, w którym programujemy, powinniśmy przynajmniej rozważyć wybór pomiędzy VBA a xlwings, który daje nam pełny dostęp do całego ekosystemu pakietów Pythona.

[5] Patrz `https://support.microsoft.com/en-us/office/introduction-to-python-in-excel-55643c2e-ff56-4168-b1ce-9428c8308545`.