

Jakub Badowski

Instytut Nafty i Gazu – Państwowy Instytut Badawczy

Możliwości zastosowania języka Python oraz jego bibliotek do wytwarzania oprogramowania dla branży naftowo-gazowniczej

W artykule dokonano przeglądu możliwości wykorzystania języka Python, zyskującego coraz większą popularność w środowiskach naukowych. Autor wprowadza w podstawy języka, pokazuje, jak w prosty sposób można zainstalować środowisko programistyczne oraz rozpocząć korzystanie z biblioteki standardowej i olbrzymiej liczby bibliotek zewnętrznych, dostępnych na otwartej licencji. Artykuł zawiera także przykłady gotowych i działających skryptów, prezentujących sposób wykorzystania niektórych z tych bibliotek. Autor zwraca uwagę na dwa aspekty wykorzystania języka Python: jako platformy do tworzenia profesjonalnych rozwiązań dla branży naftowo-gazowniczej, a także jako darmowego, ale potężnego narzędzia, łatwego do wykorzystania przez środowisko naukowe, mogącego zastąpić wykorzystywane dotychczas programy komercyjne, często kosztowne.

Słowa kluczowe: Python, język programowania, program komputerowy, analiza danych.

The possibilities of using Python language and its libraries in developing software for the oil and gas industry

The article reviews the possibilities of using Python language, which is systematically gaining more popularity within the science community. The author introduces the readers to the essentials of Python programming, shows how easy development environment can be installed and how to start using the standard library and also a large number of external libraries, available on an open source license. The article also contains examples of ready-made and active scripts that present how to use some of these libraries. The author draws attention to two aspects of using the Python language: as a platform for creating professional solutions for the oil and gas industry, as well as a free but powerful tool, easy to use by the scientific community, which can replace the often expensive commercial programs, used so far.

Key words: Python, programming language, computer application, data analysis.

Wstęp

Wydaje się, że programowanie staje się w ostatnich latach umiejętnością bardzo pożądaną, jeśli już nie wymaganą, w profilu naukowca, niezależnie od branży, jaką się zajmuje. Nie jest tajemnicą, że taki trend będzie się utrzymywał w przyszłości. Rozumieją to rodzice, którzy posyłają swoje sześciolatnie dzieci na zajęcia dodatkowe z robotyki, na których programuje się proste układy. Twórcy specjalistycznego oprogramowania również dostrzegają ten trend i wprowadzają nowe możliwości uruchamiania autorskich skryptów w ramach oferowanego programu, z którego korzystają naukowcy. Widać to szczególnie wyraźnie w kontekście bardzo nośnych ostat-

nio tematów *big data*, *data mining*, *data science*. Konferencji i szkoleń na ten temat jest całe mnóstwo, a w opisie większości z nich pojawia się temat pisania skryptów.

Z obserwacji autora wynika, że w większości przypadków, jeśli pracownik naukowy miał do czynienia z jakimś językiem programowania, był to Visual Basic for Applications (VBA), używany głównie do automatyzacji pracy w arkuszach kalkulacyjnych. Zdarza się też wykorzystywanie języka Simulink z pakietu Matlab do budowania modeli symulacyjnych.

Kolejnym ważnym aspektem jest interdyscyplinarność w badaniach naukowych. Bardzo często jednym z ogniw tej współpracy

naukowej jest właśnie sektor informatyczny. Zdaniem autora ścisła kooperacja programistów ze specjalistami z branży może dać bardzo dobre rezultaty. Choć wydaje się to oczywiste, życie często pokazuje inaczej. Aby do takiej owocnej współpracy mogło dojść, środowisko branżowe musi zdawać sobie sprawę z dostępnych możliwości, jakie oferuje szeroko rozumiany sektor IT. Zdaniem autora potrzebna jest olbrzymia praca na tym polu, a ten artykuł stanowi próbę jej podjęcia.

W niniejszym artykule przybliżono opis języka Python, jego środowiska programistycznego oraz wybranych modułów z biblioteki standardowej. Jako że biblioteka standardowa języka Python jest bardzo rozbudowana, a artykuł ma cha-

rakter przeglądowy, opisano najważniejsze moduły z punktu widzenia rozpoczęcia pracy z językiem.

Mimo że język Python jest coraz popularniejszy, materiały na jego temat są bardzo rozproszone. Istnieje wprawdzie profesjonalna dokumentacja, ale informacje w niej zawarte są mało przystępne dla osób, które programowanie traktują jako dodatkowe narzędzie pracy (np. osoby ze środowiska naukowego, które nie są zawodowymi programistami). Głównym tego powodem jest fakt, że za językiem Python stoi organizacja Python Software Foundation, typu non profit (nienastawiona na zysk), a cały ciężar wsparcia bierze na siebie społeczność zgromadzona wokół języka.

Opis języka

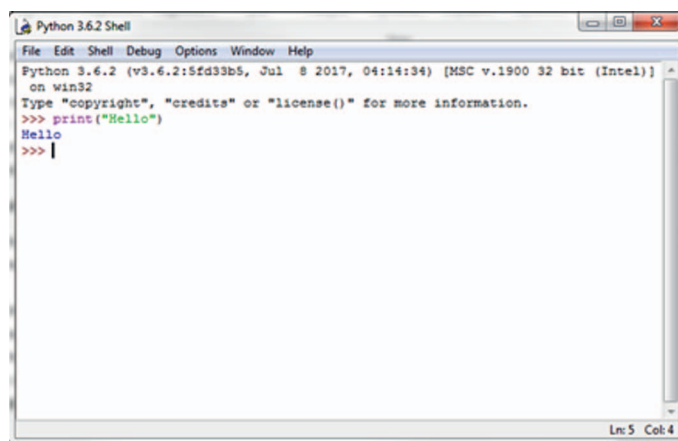
Python jest językiem dynamicznym, tzn. językiem programowania wysokiego poziomu, w którym wiele operacji, takich jak: dodawanie nowego kodu, rozszerzanie obiektów i definicji czy zmiana typów danych, można wykonywać podczas działania programu (w innych językach, np. w języku C++, operacje te przeprowadzane są na etapie kompilacji). Mówimy, że jest to język ogólnego przeznaczenia, co w praktyce oznacza bardzo duży zakres jego wykorzystania: poczynając od aplikacji desktopowych (programy instalowane lokalnie na komputerze), przez skrypty automatyzujące pracę w sieci internetowej, po aplikacje internetowe (dostępne przez przeglądarkę internetową) i programowanie mikrokontrolerów.

Do największych zalet języka należą:

- jego czytelna i prosta struktura,
- możliwość programowania na wielu platformach (Windows, Unix, Linux, Mac OS, mikrokontrolery),
- rozbudowana biblioteka standardowa oraz wielka liczba otwartych bibliotek zewnętrznych, gotowych do wykorzystania,
- prostota instalacji modułów za pomocą menadżera pakietów *pip*,
- możliwość łączenia z innymi językami programowania (część bibliotek wykorzystywanych w Pythonie jest napisana w języku C++),
- prostota pracy – wraz z kompilatorem instalowane jest domyślne środowisko programistyczne w postaci prostego edytora wraz z interaktywną konsolą (rysunek 1),
- możliwość pracy w trybie wiersza poleceń.

Główne cechy języka:

- wszystkie wartości są przekazywane przez referencję;
- do zarządzania pamięcią wykorzystywany jest mechanizm automatycznej dealokacji pamięci (ang. *garbage collection*),
- umożliwia programowanie w kilku paradygmatach (programowanie obiektowe, programowanie strukturalne oraz programowanie funkcyjne),



Rys. 1. Domyślne środowisko programistyczne języka Python

- w Pythonie wszystko jest obiektem (moduły, paczki, funkcje, każdy typ, zarówno podstawowy, jak i złożony – jak listy, krotki (patrz: podrozdział „Typy złożone”) czy słowniki).

Pierwszy działający program

Ponieważ każdy dobry wstęp do języka programowania rozpoczyna się od programu wypisującego na ekranie tekst *Hello World* (pol. *Witaj, świecie*), autor nie zamierza łamać tej tradycji (rysunek 2).

```
1 print("Hello World!")
```

Rys. 2. Pierwszy program

Niektórych może dziwić fakt, że jest to tylko jedna linia, nie ma żadnej funkcji głównej czy inicjującej, ale taki właśnie jest język Python. Ponadto, jak to uwidoczono na następnych przykładach, czytelnik nie zobaczy także klamer ani średników. Składnia języka opiera się na wcięciach oraz znakach nowej linii (*\n*), które zastępują klamry i średniki. Nowością

jest też fakt, że zmienne można deklarować w momencie użycia. Nie trzeba robić tego wcześniej, jak ma to miejsce np. w języku C. W dalszej części artykułu został opisany krok po kroku sposób instalacji środowiska Python, który pozwala na skompilowanie tego i innych przykładów.

Zmienne

Zmienne to konstrukcje programistyczne posiadające nazwę (identyfikator), miejsce przechowywania w pamięci (adres + długość przechowywanych danych) oraz wartość. W kodzie źródłowym można się do nich odwoływać za pomocą nazwy lub adresu. Wartość zmiennej może być odczytywana lub zastępowana nową wartością w czasie działania programu. Nazwa i miejsce przechowywania zmiennej w pamięci nie zmieniają się w trakcie jej istnienia. Zmienne posiada-

ją również swój typ, czyli informację określającą rodzaj przechowywanych danych (może to być liczba całkowita, napis, adres obiektu itp.). Pod tym względem języki programowania można podzielić na:

- języki z typowaniem statycznym – zmienne mają z góry określony typ, jaki mogą przechowywać,
- języki z typowaniem dynamicznym – typ nie jest atrybutem zmiennej, ale wartości w niej przechowywanej. W takim przypadku ta sama zmienna może w różnych momentach przechowywać dane innego typu.

Do zmiennych można przypisywać także funkcje. Jest to możliwe, ponieważ język Python wszystko traktuje jako obiekt. Obiekty mają swoje adresy i w takiej właśnie formie są przypisywane do zmiennych. Mówimy, że jest to przypisywanie przez referencję.

Typy zmiennych

Język Python posiada typowanie dynamiczne, co oznacza, że jego zmienne nie mają typów przypisanych statycznie, jak np. w języku C++, w którym typy zmiennych muszą być znane już na etapie kompilacji kodu. Typy danych są przypisywane do zmiennych w trakcie działania programu na podstawie przechowywanych w nich wartości. Typowanie dynamiczne ułatwia wykonywanie operacji na zmiennych, z drugiej zaś strony utrudnia kontrolę integralności programu. Ma to szczególne znaczenie przy implementowaniu wzorców projektowych (ang. *design patterns*), co wykracza poza ramy niniejszego artykułu.

W celu wprowadzenia pewnego porządku autor prezentuje uproszczony podział typów w języku Python:

- typy wbudowane
 - typy proste (podstawowe)

- *int* – liczba całkowita (ang. *integer*)
- *float* – liczba zmiennoprzecinkowa
- *complex* – liczba zespolona
- *str* – łańcuch znaków (ang. *string*)
- *bool* – typ logiczny (ang. *boolean*) [True|False]
- *NoneType* – typ pusty [None] – znany z innych języków jako „null”;
- typy złożone
 - kolekcje
 - lista (ang. *list*)
 - krotka (ang. *tuple*)
 - słownik (ang. *dictionary*)
 - zbiór (ang. *set*)
 - zbiór stały (ang. *frozenset*);
- typy zewnętrzne.

Typy złożone

Szczególną cechą Pythona, wyróżniającą go na tle innych języków programowania, jest wbudowany zasób specjalnych rodzajów typów zmiennych zwanych „kolekcjami”. W najprostszej konfiguracji kolekcje składają się ze specjalnie ułożonego zbioru typów prostych:

- lista (ang. *list*) – jest to znana z innych języków programowania tablica składająca się z elementów mogących przechowywać dane dowolnego typu. Istnieje także możliwość definiowania list wielowymiarowych;
- krotka (ang. *tuple*) – można ją określić mianem „stała lista”. Od listy różni się tym, że nie można zmieniać wartości, kolejności ani dodawać nowych elementów. Tak jak

przy listach elementy mogą być dowolnego typu. Krotki wykorzystuje się najczęściej do grupowania wartości parametrów funkcji oraz zwracania kilku wartości z funkcji;

- słownik (ang. *dictionary*) – to taki rodzaj tablicy, w której w miejsce indeksu liczbowego nadawanego automatycznie programista może wpisać dowolną wartość. Słownik to bardzo przydatna, choć rzadko spotykana w innych językach konstrukcja. Jej odpowiednikiem w języku skryptowym PHP jest tablica asocjacyjna;
- zbiór (ang. *set*) oraz zbiór stały (ang. *frozenset*) – są to typowo matematyczne zbiory danych. Można na nich wykonywać operacje na zbiorach, jak np. suma, iloczyn itp.

Zasięg zmiennych

W przeciwieństwie do innych języków programowania dostęp do zmiennych w Pythonie jest mało restrykcyjny. W zasadzie zawsze istnieje możliwość odczytania wartości zmiennej w dowolnym miejscu w programie. Jeżeli w ciele funkcji zachodzi konieczność modyfikacji zmien-

nej spoza zasięgu, wystarczy ustawić, aby zmienna była interpretowana jako globalna. Należy jednak pamiętać, że nazwy wymienione w instrukcji *global* nie mogą być używane w tym samym bloku kodu w części tekstowo poprzedzającej tę instrukcję.

Operacje na zmiennych

Oprócz trywialnych operacji na zmiennych, jak dodawanie, odejmowanie, mnożenie, dzielenie, potęgowanie, reszta z dzielenia, Python posiada możliwość wykonywania nieco bardziej zaawansowanych operacji, jak suma logiczna, iloczyn logiczny, przesunięcia bitowe, negacja bitowa czy konkatencja (łączenie) oraz mnożenie zmiennych typu *string*.

Podstawowe funkcje wbudowane

Funkcja *print()* to najczęściej wykorzystywana funkcja wbudowana. Służy przede wszystkim do wypisywania danych na wyjście standardowe, czyli w większości przypadków na ekran komputera. Ma ona bardzo przydatny tryb przeznaczony do wydajnego formatowania łańcuchów znaków, znany z innych języków programowania (PHP, C++) jako funkcja *printf()*. Funkcja *id()* zwraca unikalny identyfikator każdego zdefiniowanego elementu – zmiennej, funkcji czy obiektu (upraszczając, ponieważ zmienne i funkcje to w Pythonie także obiekty). Funkcja *input()* służy do interakcji użytkownika ze skryptem. W czasie wykonywania skryptu mogą zajść jakieś okoliczności wymagające interwencji użytkownika. Za pomocą tej funkcji programista może przechwycić komendę i zapisać ją do zmiennej w celu dalszego wykorzystania. Funkcja *dir()* pełni ważną rolę w procesie ręcznego debugowania kodu źródłowego. Zwraca ona listę dostępnych atry-

butów i metod badanego obiektu. Funkcje wbudowane są dostępne do użycia bez konieczności importowania jakiegokolwiek modułu. Można z nich korzystać, podając nazwę funkcji wprost (bez notacji z kropką, jak w przypadku modułów z biblioteki standardowej lub zewnętrznych).

Bloki kontrolne

Głównymi założeniami języka Python jest prostota i przejrzystość kodu. Z tego względu liczba instrukcji blokowych, takich jak instrukcje warunkowe i pętle, jest mocno zredukowana. Nie ogranicza to jednak w żaden sposób możliwości języka. Czytelnicy mający styczność z innymi językami programowania, jak C czy Visual Basic, z pewnością znają instrukcje: *switch*, *do ... while* czy *label ... goto*. Język Python ogranicza te konstrukcje tylko do jednej instrukcji warunkowej:

- *if ... else ...*

oraz dwóch rodzajów pętli:

- *for* oraz
- *while*.

W blokach kontrolnych nie stosuje się nawiasów ostrych {}, jak w przypadku języków programowania pochodzących od języka ALGOL. Bloki oznaczane są przez wcięcia (tabulator, dwie spacje lub cztery spacje) oraz znaki nowej linii. W wierszu poleceń każdy blok kontrolny musi być zakończony pustą linią.

Instrukcje warunkowe

Instrukcje warunkowe są nieodłącznym elementem każdego języka programowania. Składnia tej instrukcji w Pythonie jest intuicyjna i nie wymaga komentarza. Różni się natomiast w szczegółach implementacyjnych tworzenie warunków, choć i tak większość operatorów jest identyczna jak w innych językach programowania:

```
== równe,
> większe,
< mniejsze,
>= większe lub równe,
<= mniejsze lub równe,
!= różne.
```

Istnieje jednakże jedna, bardzo ważna różnica. W języku Python nie istnieje operator „*===*”, który oznacza tzw. dokładną równość. Pojęcie to sprowadza się do sprawdzenia oprócz zawartości zmiennej także jej typu. Aby „dokładna równość” dała w rezultacie logiczną prawdę, musi się zgadzać zarówno typ, jak i wartość zmiennej. W języku Python rolę „dokładnej różności” pełni słowo kluczowe *is*. Analogicznie w przypadku warunku „różny”, oznaczonego symbolem „*!=*”, Python posługuje się notacją *is not*. Po raz kolejny widać jak w Pythonie eksponowana jest czytelność kodu. Język Python oferuje także bardzo przydatny operator *in*, pozwalający sprawdzić, czy w kolekcji znajduje się element o podanej wartości.

Pętle

Poniższy skrypt zawiera przykłady użycia wszystkich rodzajów pętli dostępnych w języku Python (rysunek 3). Od linii 10 zaczyna się pętla nieskończona, przerwana w linii 12, w celach demonstracyjnych, instrukcją *break*. Pętle nieskończone są bardzo często wykorzystywane w programach do komunikacji sieciowej do nasłuchiwania przychodzących komunikatów. Dla przykładu, najprostsza implementacja serwera WWW to nic innego jak pętla nieskończona.

Warto zwrócić uwagę na działanie pętli „for” z wykorzystaniem funkcji wbudowanej *range()* – w skrypcie powyżej od linii 19. Funkcja *range()* jest tzw. generatorem, czyli taką funkcją, po której można iterować. Ma to tę zaletę, że parser nie tworzy w pamięci nowej listy, co zabierałoby zasoby, ale kolejny element jest „generowany” przy każdej iteracji. Analogiczne działanie uzyskuje się w innych językach (PHP, C++) za pomocą wzorca projektowego Iterator (ang. *Iterator design pattern*).

```

1  exampleList = [54, 25, 86, 35, 7, 3, 5]
2
3  # Pętla While
4  condition = 1
5  while condition < 10:
6      print(condition)
7      condition += 1
8
9  # Nieskończona pętla While
10 while True:
11     print("Testowy ciąg znaków")
12     break
13
14 # Pętla For
15 for x in exampleList:
16     print(x)
17
18 # Pętla For z wykorzystaniem funkcji range()
19 for x in range(1, 10):
20     print(x)
21

```

Rys. 3. Przykłady implementacji pętli

Instalacja środowiska

W celu instalacji środowiska Python należy pobrać aktualną paczkę instalacyjną ze strony <https://www.python.org/downloads/>, odpowiednią dla swojego systemu operacyjnego. W trakcie pisania tego artykułu autor korzystał z Pythona w wersji 3.6.2 dla systemu Windows 7. Proces instalacji jest bardzo intuicyjny i nie różni się niczym od instalacji innych programów. Dla początkujących użytkowników zaleca się pozostanie przy domyślnych ustawieniach konfiguracyjnych. Kolejnym krokiem jest dodanie dwóch katalogów do zmiennej środowiskowej (dla systemu Windows: %PATH%). Te katalogi to:

- C:\Users\username\AppData\Local\Programs\Python\Python36-32,
- C:\Users\username\AppData\Local\Programs\Python\Python36-32\Scripts.

Pierwszy odpowiada za możliwość korzystania z polecenia *python* w terminalu bez konieczności podawania pełnej ścieżki. W katalogu *Scripts* natomiast znajduje się menadżer pakietów *pip*, który także jest często wykorzystywany w pracy programisty.

Jeśli środowisko Python jest już zainstalowane, w celu weryfikacji kompilatora należy przejść do okna terminala (*cmd* w Windows, *bash* dla systemów z rodziny Linux) i wpisać polecenie: *python*. W komunikacie powitalnym pojawiają się dane dotyczące aktualnie zainstalowanej wersji kompilatora, w jakiej architekturze działa (32 bity lub 64 bity) oraz inne przydatne informacje. Następnie pojawia się konsola interaktywna, do której można bezpośrednio wpisywać kod w Pythonie i go kompilować.

Po instalacji kompilatora Python można właściwie od razu przystąpić do pracy we wbudowanym środowisku IDLE

(Python’s Integrated Development Environment), które do prostych zadań w zupełności wystarczy. Bardziej doświadczeni programiści mają często swoje ulubione edytory. Autor rekomenduje korzystanie z Sublime Text 3 – potężnego edytora dla programistów, który można pobrać ze strony <https://www.sublimetext.com/> (rysunek 4).

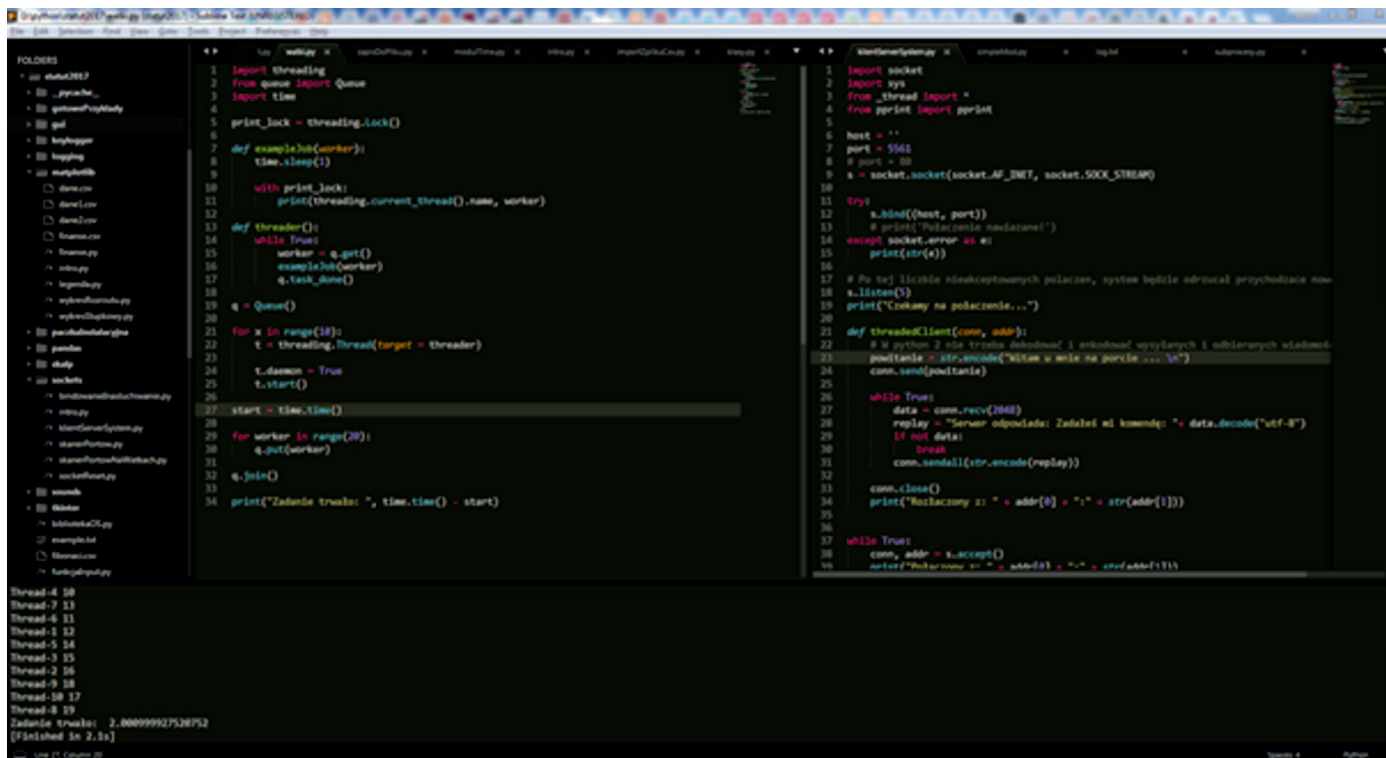
Spośród wielu zalet tego edytora warto wymienić choćby kilka:

- prostota i lekkość – podczas instalacji edytor nie jest obciążony wieloma dodatkami,
- modułowość – potrzebne dodatki, których jest ogromna liczba, można doinstalować w zależności od potrzeb,
- prostota instalacji dodatków – dodatki instaluje się bezpośrednio z edytora. Proces instalacji odbywa się w tle,
- wygodny sposób definiowania preferencji – preferencje użytkownika oraz skróty klawiszowe definiuje się w prostym pliku o formacie JSON,
- bardzo wygodne zarządzanie projektami – programista może wygodnie pracować przy kilku projektach, każdym w innym języku,
- możliwość kompilacji kodu bezpośrednio w edytorze.

Jako ciekawostkę warto podkreślić, że spora część edytora Sublime Text została napisana właśnie w języku Python.

Instalacja modułów zewnętrznych

Po instalacji środowiska programistycznego programista ma do dyspozycji wszystkie moduły z biblioteki standardowej. Zanim zaczniemy pisać własny kod, należy upewnić się, czy nie ma gotowego modułu, który rozwiąże problem lub ułatwi



Rys. 4. Okno edytora Sublime Text 3 w czasie pracy programisty

nam to zadanie. Pełna lista modułów jest dostępna na <https://docs.python.org/3/library/index.html>. Największą jednak zaletą języka Python są jego biblioteki zewnętrzne. Aktualizowana lista modułów posortowanych tematycznie znajduje się na stronie internetowej Python Software Foundation – pod adresem <https://wiki.python.org/moin/UsefulModules>.

Sposób instalacji modułów jest bardzo prosty niezależnie od używanego systemu operacyjnego – choć sam proces instalacji różni się w szczegółach. W systemach z rodziny Linux (w tym przypadku Debian) moduł zewnętrzny instalujemy np. poleceniem: `sudo apt-get install python-matplotlib`.

Polecenie to pobiera potrzebne pliki ze zdalnego serwera, którego adres znajduje się w repozytorium. Pobierane są również wszystkie potrzebne zależności, czyli inne biblioteki, z których dany moduł korzysta. Przykładowo, instalując `matplotlib`, zostają pobrane także `six`, `cycler`, `pytz`, `numpy`.

W systemach Windows (w tym przypadku Windows 7) istnieją przynajmniej trzy popularne sposoby instalacji modułu zewnętrznego:

1. Sposób pierwszy – pobieramy paczkę `tar.gz` i ją rozpakujemy. Klikamy prawym przyciskiem, trzymając klawisz Shift, na katalog z modułem i z menu kontekstowego wybieramy „Otwórz okno poleceń tutaj”. W oknie poleceń wpisujemy komendę: `setup.py install`.
2. Sposób drugi – instalacja pakietów i modułów za pomocą menadżera pakietów. `Pip` to bardzo wygodny system zarządzania zależnościami, napisany w Pythonie i w nowszych wersjach instalowany wraz z jego środowiskiem. Poniżej znajduje się kilka komend, za pomocą których można łatwo zarządzać zależnościami:
 - `pip install some-package-name` – instalacja pakietu,
 - `pip uninstall some-package-name` – usunięcie pakietu,
 - `pip install --upgrade matplotlib` – aktualizacja przykładowego pakietu.
3. Sposób trzeci – katalog bibliotek Python: <http://www.lfd.uci.edu/~gohlke/pythonlibs/>, z którego pobieramy plik z rozszerzeniem `.whl`, a następnie wpisujemy do konsoli: `pip install nazwa-paczki.whl`.

Wybrane biblioteki zewnętrzne

Biblioteka `matplotlib` to potężne narzędzie do generowania wykresów. Bardzo szybko można zacząć pracę z prostymi figurami. Wystarczą dwie linie kodu, aby wygenerować pierwszy wykres. Domyślnie wykresy renderowane są w nowym oknie systemowym, posiadającym możliwość zmiany „na żywo” najważniejszych parametrów oraz proste menu na-

wigacyjne. Sformatowany wykres można również zapisać do pliku graficznego. Wykres można formatować, określając parametry, tj. tytuł, podpis osi, dodawanie legendy, grubość siatki czy kolory wykresów. Wiele przykładów tworzenia wykresów za pomocą biblioteki `matplotlib` znajduje się na oficjalnej stronie internetowej projektu.

Natomiast pełna moc biblioteki objawia się dopiero w połączeniu z pakietem *pandas*, przeznaczonym do manipulowania danymi oraz ich analizy.

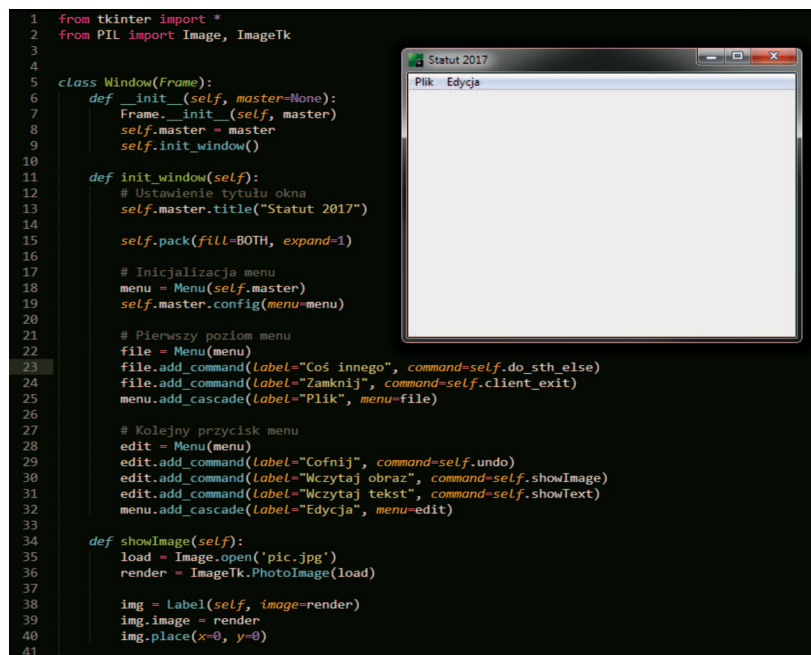
Za pomocą interfejsu *pandas* można bardzo łatwo wczytać dane z pliku, dokonywać pewnych operacji i z powrotem zapisywać, zmieniając przy tym np. format pliku. Obsługiwane formaty danych to: CSV, Excel, JSON, XML (<http://pandas.pydata.org/pandas-docs/version/0.20/io.html>).

Biblioteka *pandas* opakowuje słownik danych w obiekt o typie *DataFrame*. Nawiązując do analogii arkusza danych (arkusza kalkulacyjnego), poszczególne krotki słownika są kolumnami, które zawierają dane w kolejności poszczególnych wierszy arkusza. Wierszom nadawany jest unikatowy indeks, czyli kolejny numer zaczynający się od zera. Można sprawić, aby indeksem była jakaś inna kolumna danych. Wtedy dane w tej kolumnie muszą być unikatowe. Kolumna, która jest indeksem, jest wyświetlana, ale nie jest wliczana do liczby kolumn. Jeżeli za pomocą *set_index()* zmienimy przeznaczenie jakiejś kolumny na kolumnę indeksową, przestaje ona być wliczana do liczby kolumn.

Wczytując dane z pliku, można od razu ustawić, która kolumna ma być indeksem, za pomocą parametru *index_col* (numerujemy oczywiście od zera):

```
df = pd.read_csv('new1.csv',
names = ['Daty', 'Kursy'], index_col = 0)
```

Programowanie interfejsów użytkownika dla aplikacji okienkowych kojarzy się wielu osobom z trudnym zadaniem, wymagającym drogich narzędzi programistycznych o skomplikowanych licencjach. Poniższy skrypt pokazuje, jak za pomocą kilkudziesięciu linii kodu można stworzyć działające okno systemowe, w ramach którego możemy tworzyć dalszy interfejs (rysunek 5).



```
1 from tkinter import *
2 from PIL import Image, ImageTk
3
4
5 class Window(Frame):
6     def __init__(self, master=None):
7         Frame.__init__(self, master)
8         self.master = master
9         self.init_window()
10
11     def init_window(self):
12         # Ustawienie tytułu okna
13         self.master.title("Statut 2017")
14
15         self.pack(fill=BOTH, expand=1)
16
17         # Inicjalizacja menu
18         menu = Menu(self.master)
19         self.master.config(menu=menu)
20
21         # Pierwszy poziom menu
22         file = Menu(menu)
23         file.add_command(Label="Coś innego", command=self.do_something)
24         file.add_command(Label="Zamknij", command=self.client_exit)
25         menu.add_cascade(Label="Plik", menu=file)
26
27         # Kolejny przycisk menu
28         edit = Menu(menu)
29         edit.add_command(Label="Cofnij", command=self.undo)
30         edit.add_command(Label="Wczytaj obraz", command=self.showImage)
31         edit.add_command(Label="Wczytaj tekst", command=self.showText)
32         menu.add_cascade(Label="Edycja", menu=edit)
33
34     def showImage(self):
35         load = Image.open('pic.jpg')
36         render = ImageTk.PhotoImage(load)
37
38         img = Label(self, image=render)
39         img.image = render
40         img.place(x=0, y=0)
41
```

Rys. 5. Przykładowy program tworzący proste okno systemowe z tradycyjnym menu

Podsumowanie

Niniejszy artykuł zawiera opis najważniejszych konstrukcji języka Python, jego biblioteki standardowej, a także przegląd wybranych bibliotek zewnętrznych. Zaprezentowano zestaw narzędzi do tworzenia oprogramowania na potrzeby wewnętrzne i zewnętrzne.

Autor zwraca uwagę na dwa aspekty wykorzystania języka Python: jako platformy do tworzenia profesjonalnych rozwiązań dla branży naftowo-gazowniczej, a także jako darmowego, ale potężnego narzędzia, łatwego do wykorzystania przez środowisko naukowe, mogącego zastąpić komercyjne, często kosztowne używane dotychczas programy.

Zdaniem autora obecnie nie ma bariery technologicznej na poziomie programowania – jedynym ograniczeniem jest wiedza programisty. Wszystkie niezbędne narzędzia programistyczne są dostępne na licencjach otwartych (MIT, GNU/

GPL itp.). W kilka miesięcy, korzystając z wiedzy dostępnej w Internecie, można poznać nową technologię i zacząć w niej tworzyć kompleksowe i profesjonalne rozwiązania/programy komputerowe.

Ogromną zaletą wykorzystywania języka Python zamiast komercyjnych rozwiązań jest duża elastyczność pobierania danych oraz zwracania wyników w celu dalszego ich przetwarzania. Używając dynamicznego języka, programista może zaimplementować rozwiązanie, które pobiera dane w czasie rzeczywistym, np. z czujników, logów czy kamery termowizyjnej, analizuje je na bieżąco i natychmiast podejmuje decyzję co do dalszego postępowania. Takie oprogramowanie może działać również na mikrokontrolerze (w zależności od potrzebnej mocy obliczeniowej do przetwarzania tych danych), co daje nowe, ogromne możliwości.

Prosimy cytować jako: Nafta-Gaz 2018, nr 2, s. 113–120, DOI: 10.18668/NG.2018.02.05

Artykuł nadesłano do Redakcji 27.11.2017 r. Zatwierdzono do druku 9.02.2018 r.

Artykuł powstał na podstawie pracy statutowej pt.: *Możliwości zastosowania języka Python oraz jego bibliotek do wytwarzania*

oprogramowania dla branży naftowo-gazowniczej – praca INiG – PIB na zlecenie MNiSW; nr zlecenia: 19/SP/17, nr archiwalny: DK-4100-6/17.

Literatura

[1] Strona internetowa: <http://effbot.org/zone/python-with-statement.htm> (dostęp: 27.10.2017).

[2] Strona internetowa: <http://simeonfranklin.com/blog/2012/jul/1/python-decorators-in-12-steps/> (dostęp: 27.10.2017).

[3] Strona internetowa: <https://docs.python.org/3/> (dostęp: 27.10.2017).

[4] Strona internetowa: <https://docs.python.org/3/library/index.html> (dostęp: 27.10.2017).

[5] Strona internetowa: <https://jeffknupp.com/blog/2013/04/07/improve-your-python-yield-and-generators-explained/> (dostęp: 27.10.2017).

[6] Strona internetowa: <https://jeffknupp.com/blog/2016/03/07/python-with-context-managers/> (dostęp: 27.10.2017).

[7] Strona internetowa: <https://matplotlib.org/index.html> (dostęp: 27.10.2017).

[8] Strona internetowa: https://www.youtube.com/watch?v=o-Vp1vrfL_w4&list=PLQVvvaa0QuDe8XSftW-RAxdo6Oma-eL85M (dostęp: 27.10.2017).

[9] Strona internetowa: https://www.youtube.com/watch?v=YSe-9Tu_iNQQ&list=PLQVvvaa0QuDfju7ADVp5W1GF9jVhjbX_ (dostęp: 27.10.2017).



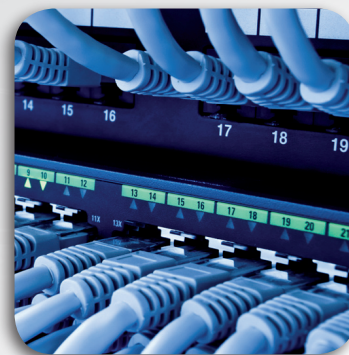
Mgr inż. Jakub BADOWSKI
 Asystent w Zakładzie Informatyki.
 Instytut Nafty i Gazu – Państwowy Instytut Badawczy
 ul. Lubicz 25 A
 31-503 Kraków
 E-mail: jakub.badowski@inig.pl

OFERTA

ZAKŁAD INFORMATYKI

Zakres działania:

- zastosowanie matematyki i technologii informatycznych do:
 - » konstrukcji dziedzinowych systemów eksperckich,
 - » budowy i eksploatacji baz danych i baz wiedzy,
 - » wykorzystania metod analiz ryzyka,
 - » konstrukcji Systemu Zarządzania Integralnością Gazociągów,
 - » budowy komputerowych systemów wspomaganie decyzji,
 - » analizy statystycznej wyników eksperymentów badawczych,
 - » tworzenia unikatowego oprogramowania,
- zarządzanie siecią komputerową i dostępem do internetu w ramach sieci korporacyjnej INiG – PIB.



Kierownik: mgr Andrzej Dietrich
Adres: ul. Bagrowa 1, 30-733 Kraków
Telefon: 12 653-25-12 w. 149
Fax: 12 650 77 50, 12 653 16 65
E-mail: andrzej.dietrich@inig.pl

