Kurs rozszerzony języka Python

Wykład 14.

Marcin Młotkowski

21 stycznia 2025

Plan wykładu

Plan wykładu

Najprościej

Mamy mnóstwo katalogów i podkatalogów

- tworzymy plik `__main__.py`, od którego powinno się zacząć uruchamianie aplikacji;
- pakujemy wszystko do zip'a (`aplikacja.zip`) i wysyłamy do miejsca docelowego;
- w miejscu docelowym:
  `$ python aplikacja.zip`.

Wady: na ogół działa; ale nie kontroluje wersji, modułów dodatkowych etc.

Formaty dystrybucji pakietów i programów pythonowych

Dystrybucja: archiwum zip z plikami źródłowymi + informacje dodatkowe + moduły binarne (tylko wheel).

- **egg**: starszy format, niepolecany;
- **wheel**: aktualny format.

Przykład

Implementacja obliczania n-tej liczby Fibonacciego algorytmem macierzowym.

Narzędzia

setuptools: instalacja `# pip3 install build`

Struktura projektu

```
pyproject.toml
src/
  pydemo/
    __init__.py
    pyimpl.py
```

Objaśnienie

`pyproject.toml`: opisuje wymagania do zbudowania projektu.
`__init__.py` oznacza, że budujemy pakiet.

Co dalej

```
$ python -m build
$ python -m pip install dist/<plik>.whl
```

Plan wykładu

Implementacja w C

Zaprogramować w C i udostępnić w Pythonie jako moduł.

Implementacja w C

Zaprogramować w C i udostępnić w Pythonie jako moduł.
Co jest potrzebne: C API

Problemy łączenia dwóch języków

Zagadnienia

- problemy z różnymi typami danych (listy, kolekcje, napisy);
- przekazywanie argumentów i zwracanie wartości;
- tworzenie nowych wartości;
- obsługa wyjątków;
- zarządzanie pamięcią.

Dodanie do Pythona nowej funkcji

Zadanie

Implementacja obliczania n-tej liczby Fibonacciego w C

Dodanie do Pythona nowej funkcji

Zadanie

Implementacja obliczania n-tej liczby Fibonacciego w C

Elementy implementacji:

- plik nagłówkowy <Python.h>;
- implementacja funkcji;
- odwzorowanie funkcji w C na nazwę udostępnioną w Pythonie;
- funkcja inicjalizująca o nazwie *initnazwa_modułu*.

Implementacja funkcji (1)

```
#include <python3.8/Python.h>
extern PyObject * fib(PyObject *, PyObject *);

PyObject * fib(PyObject * self, PyObject * args)
{

  PyObject * res;
  PyObject * wyraz;

  n = PyLong_AsLong(wyraz);

  if (n == 0)
  {
    res = Py_BuildValue("i", 0);
    Py_INCREF(res);
    return res;
  }
```

Implementacja funkcji (2)

```
n = PyLong_AsLong(wyraz);

if (n == 0)
{
  res = Py_BuildValue("i", 0);
  Py_INCREF(res);
  return res;
}
```

Implementacja funkcji (3)

```
    ...
    res = Py_BuildValue("i", w11);
    Py_INCREF(res);
    return res;
}
```

Deklaracje modułu

```
static PyMethodDef metody[] = {
  {"cfib", fib, METH_VARARGS,
       "n-ta liczba Fibonacciego", },
  { NULL, NULL, -1, NULL }
};

static PyModuleDef moduledef = {
  PyModuleDef_HEAD_INIT,
  "fastcomp",
  "Szybkie obliczenia",
  -1,
  metody,
  NULL, NULL, NULL, NULL,
};
```

Inicjowanie modułu

```
PyMODINIT_FUNC
PyInit_fastfibb(void)
{
  PyObject *m;
  m = PyModule_Create(&moduledef);

  return m;
}
```

Budowa pakietu wheel

pyproject.toml:

```
[build-system]
requires = ["setuptools"]
build-backend = "setuptools.build_meta"

[project]
name = "cmodule"
version = "1.1"
dependencies = ['importlib-metadata; python_version>"3.6"'
```

Kompilacja i instalacja

setup.py

```
from setuptools import Extension, setup

setup(
    ext_modules = [
        Extension(name = "cmodule", sources = ["fastfb.c"]
        ]
    )
```

Kompilacja

```
$ python -m build
$ python -m pip install <plik.whl>
```

Typy danych w Pythonie

Wszystko w Pythonie jest obiektem

Zarządzanie pamięcią

Mechanizm zarządzania pamięcią

- Każdy obiekt ma licznik odwołań zwiększany za każdym przypisaniem.
- Jeśli licznik jest równy zero obiekt jest usuwany z pamięci.
- W programach w C trzeba dbać o aktualizację licznika.

Zmiana licznika odwołań

Zwiększenie licznika

```
void Py_INCREF(PyObject *o)
```

Zmniejszenie licznika

```
void Py_DECREF(PyObject *o)
```

Trochę łatwiej

Biblioteka Boost:

+ łączenie Pythona z C++
+ łatwiejsza od C API
  - czasem nie da się ominąć C API (ale się rozwija)

Jak skorzystać

```
$ python -m pip install <plik.whl>
```

```
from cmodule import fib
```

Wykonanie programów Pythonowych

```
Py_Initialize();
PyRun_SimpleString("i = 2")
PyRun_SimpleString("i = i*i\nprint(i)")
Py_Finalize();
```

Wykonanie programów w pliku

```
Py_Initialize();
FILE * f = fopen("test.py", "r");
PyRun_SimpleFile(f, "test.py");
Py_Finalize();
```

Kompilacja

```
gcc -lpython3.8 test.c
```

Bezpośrednie wywoływanie funkcji Pythonowych

Deklaracja zmiennych

```
PyObject *pName, *pModule, *pArgs, *pFunc, *pValue;
```

Import modułu Pythonowego

```
Py_Initialize();
pName = PyString_FromString("modulik");
pModule = PyImport_Import(pName);
```

Pobranie funkcji z modułu

```
pFunc = PyObject_GetAttrString(pModule, "foo");
```

Wywołanie funkcji

```
pValue = PyObject_CallObject(pFunc, pArgs);
```

Pakiet, ale jeszcze inaczej

Ta sama implementacja, ale w Rust!

Program

```rust
use pyo3::prelude::*;

#[pyfunction]
fn fastfib (m: i64) -> i128 { ... }

#[pymodule]
fn rustmodule(m: &Bound<'_, PyModule>) -> PyResult<()> {
    // inicjowanie modułu
    m.add_function(wrap_pyfunction!(fastfib, m)?)
}
```

Struktura projektu

```
pyproject.toml
Cargo.toml
src/
    lib.rs
```

Konfiguracja pakietu Pythonowego

pypyroject.toml:

```
[build-system]
requires = ["maturin>=1,<2"]
build-backend = "maturin"

[project]
name = "rustmodule"
version = "0.4"
requires-python = ">=3.7"
classifiers = [
    "Programming Language :: Rust",
    "Programming Language :: Python :: Implementation :: CP
    "Programming Language :: Python :: Implementation :: Py
]
```

Konfiguracja pakietu Rust

Cargo.toml:

```
cargo-features = ["edition2024"]

[package]
name = "lib"
# these are good defaults:
version = "0.1.0"
edition = "2024"

[lib]
name = "rustmodule"
crate-type = ["cdylib"]

[dependencies]
pyo3 = { version = "0.22.6", features = ["extension-module"
```

Jak to razem zbudować

- instalacja pakietu potrzebnego do wywoływania funkcji rust'owych w pythonie:
  $ python -m pip install maturin
- instalacja narzędzia do budowy pakietów pythonowych:
  $ python -m pip install build
- kompilacja wszystkiego:
  $ python -m build
- instalacja:
  $ python -m pip install <plik.whl>

Plan wykładu

Kanoniczna implementacja

CPython

Podstawowa implementacja języka Python w C.

PyPy

Wydajność

4.8 razy szybszy niż CPython 3.7

- *jit compilation*;
- napisany w RPython (Restricted Python);
- wysoka zgodność z Pythonem 2.7.18 i 3.9.15;
- możliwość dołączania własnego odśmiecacza pamięci;
- wsparcie dla *greenletów* i stackless;
- nieco inne zarządzanie pamięcią.

Stackless Python

- interpreter oparty na mikrowątkach realizowanych przez interpreter, nie przez kernel;
- dostępny w CPythonie jako *greenlet*;
- *stackless* bo unika korzystania ze stosu wywołań C.

Cython

Inspirowany składnią C język podobny do pythona (nadzbiór Pythona).
Kod jest kompilowany do C/C++ i dostępny dla CPythona jako moduł.
Realizacje: pandas

Przykład prograu: fraktale

```
def create_fractal( double min_x,
                    double min_y,
                    double pixel_size,
                    int nb_iterations,
                    colours,
                    image):

    cdef int width, height
    cdef int x, y, start_y, end_y
    cdef int nb_colours, current_colour, new_colour
    cdef double real, imag

    nb_colours = len(colours)
    # image is an ndarray of size: w,h,3
    width = image.shape[0]
    height = image.shape[1]

    for x in range(width):
        real = min_x + x*pixel_size
```

Numba

Podzbiór Pythona i numpy kompilowany LLVM. Cechy:

- wektoryzacja kodu by wykorzystać wielordzeniowość;
- wykorzystanie GPU;
- "wystarczy" udekorować kod:

```
from numba import jit

@jit(nopython=True)
def go_fast(a):
    trace = 0.0
    for i in range(a.shape[0]):
        trace += np.tanh(a[i, i])
    return a + trace
```

Jython

Cechy Jythona

- implementacja Pythona na maszynę wirtualną Javy;
- kompilacja do plików `.class`;
- dostęp do bibliotek Javy;
- zgodny z Python 2.7.1.

IronPython

- Implementacja Pythona w środowisku Mono i .NET;
- zgodny z Pythonem 2.7 i 3.4–3.6.

Python for S60

Implementacja Nokii na telefony komórkowe z systemem Symbian 60

- implementacja Python wersji 2.2.2;
- dostęp do sprzętu (SMS'y, siła sygnału, nagrywanie video, wykonywanie i odbieranie połączeń);
- wsparcie dla GPRS i Bluetooth;
- dostęp do 2D API i OpenGL.