Kurs rozszerzony języka Python

Wykład 14.

Marcin Młotkowski

25 stycznia 2023

Plan wykładu

Plan wykładu

Kanoniczna implementacja

CPython

Podstawowa implementacja języka Python w C.

PyPy

- *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):
```

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 tefony 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.

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;
}
```

Kompilacja i instalacja

setup.py

```
from distutils.core import setup, Extension

modul = Extension('fastfibb',
                  sources = ['fastfb.c'])

setup (name = 'fastfibb',
       version = '0.1',
       description = 'This is a demo package',
       ext_modules = [modul])
```

Kompilacja i instalacja

```
$ python setup.py build
$ python setup.py install
```

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

Skopiować do lokalnego katalogu plik *.so

```
from fastfibb import cfib
```

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);
```

Plan wykładu

Lokalne środowisko Pythonowe

virtualenv

Tworzy w lokalnym katalogu pełną wersję środowiska pythonowego, którą można modyfikować niezależnie od głównej instalacji. Można mieć wiele takich wirtualnych środowisk.

Lokalne środowisko Pythonowe

virtualenv

Tworzy w lokalnym katalogu pełną wersję środowiska pythonowego, którą można modyfikować niezależnie od głównej instalacji. Można mieć wiele takich wirtualnych środowisk.

```
$ python3 -m venv --system-site-packages $HOME/mojesrodowis
$ cd $HOME/mojesrodowisko/
$ source bin/activate
```

Alternatywnie

```
virtualenv --system-site-packages $HOME/mojesrodowisko
```

Przykład