Kurs rozszerzony języka Python

Inne języki

Marcin Młotkowski

29 stycznia 2020

Plan wykładu

Plan wykładu

Kanoniczna implementacja

CPython

Podstawowa implementacja języka Python w C.

PyPy

- jit compilation;
- wysoka zgodność z Pythonem 2.7 i 3.6;
- 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.

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 2.7.9, choć są niezgodności.

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.

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

Moduł z funkcją obliczającą średnią arytmetyczną elementów listy.

Dodanie do Pythona nowej funkcji

Zadanie

Moduł z funkcją obliczającą średnią arytmetyczną elementów listy.

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

```
extern PyObject * mean(PyObject *, PyObject *);

PyObject * mean(PyObject * self, PyObject * args)
{
    PyObject * res;
    PyObject * item;
    PyObject * lista;
    Py_ssize_t n;
    if (!PyArg_ParseTuple(args, "O", &lista)) return NULL;
    if (!PyList_Check(lista)) printf("To nie jest lista!\n");
    n = PyList_Size(lista);
```

Implementacja, cd.

cd. funkcji

```
    for (i = 0; i < n; i++) {
        item = PyList_GetItem(lista, i);
        if (!PyLong_Check(item)) continue;
        suma += PyInt_AsLong(item);
    }
    res = Py_BuildValue("i", suma/n);
    Py_INCREF(res);
    return res;
}
```

Opakowanie funkcji

```
#include <python3.5/Python.h>
```

```
extern PyObject * mean(PyObject *, PyObject *);

PyObject * mean(PyObject * self, PyObject * args)
{ ... }
```

Deklaracje modułu

```
static PyMethodDef funkcje[] = {
    { "mean", mean, METH_VARARGS, "Pierwsza funkcja" },
    { NULL, NULL, -1, NULL }
};

static PyModuleDef moduledef = {
    PyModuleDef_HEAD_INIT, "modulik", "Opis modułu",
    -1, funkcje, NULL, NULL, NULL, NULL,
};
PyMODINIT_FUNC PyInit_modulik(void) {
    return PyModule_Create(&moduledef);
}
```

Kompilacja i instalacja

setup.py

```
from distutils.core import setup, Extension
modul = Extension('modulik', sources = ['test.c'])
setup(name = "MyPackage",
        version = '0.1',
        description = 'Demonstracja C API',
        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)

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 -lpython3.5 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 środowsika pythonowego, którą można modyfikować niezależnie od głównej instalacji. Można mieć wiele takich wirtualnych środowisk.

Lokalne środowisko Pythonowe

virtualenv

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

```
$ virtualenv --system-site-packages $HOME/mojesrodowisko
$ cd $HOME/mojesrodowisko/
$ source bin/activate
```

Przykład

jupyter

Interaktywne środowisko do analizy danych i obliczeń naukowych, np. w pythonie.

Przykład Pawła Rychlikowskiego

Plan wykładu

Formaty

- egg: stary format;
- wheel: aktualny.

Formaty

- egg: stary format;
- wheel: aktualny.

Instalacja pakietów

pip

Dystrybucja programów

- Cyton: wygenerowanie kodu w C i kompilacja;
- Nuitka: generowanie kodu C++;

Dystrybucja programów

- Cyton: wygenerowanie kodu w C i kompilacja;
- Nuitka: generowanie kodu C++;
- inne, np. py2exe

A bez kompilacji

Skompresować pliki do zip'a!

A bez kompilacji

Skompresować pliki do zip'a!

1. sposób

Plik początkowy nazwać `__main__.py` i skompresować cały projekt.

A bez kompilacji

Skompresować pliki do zip'a!

1. sposób

Plik początkowy nazwać `__main__.py` i skompresować cały projekt.

2. sposób

```
$ python3 -m zipapp apka -m ''apka:startapp''
```

gdzie `apka` to katalog z plikami, a plik `apka/startapp.py` to początek programu.

Plan wykładu