



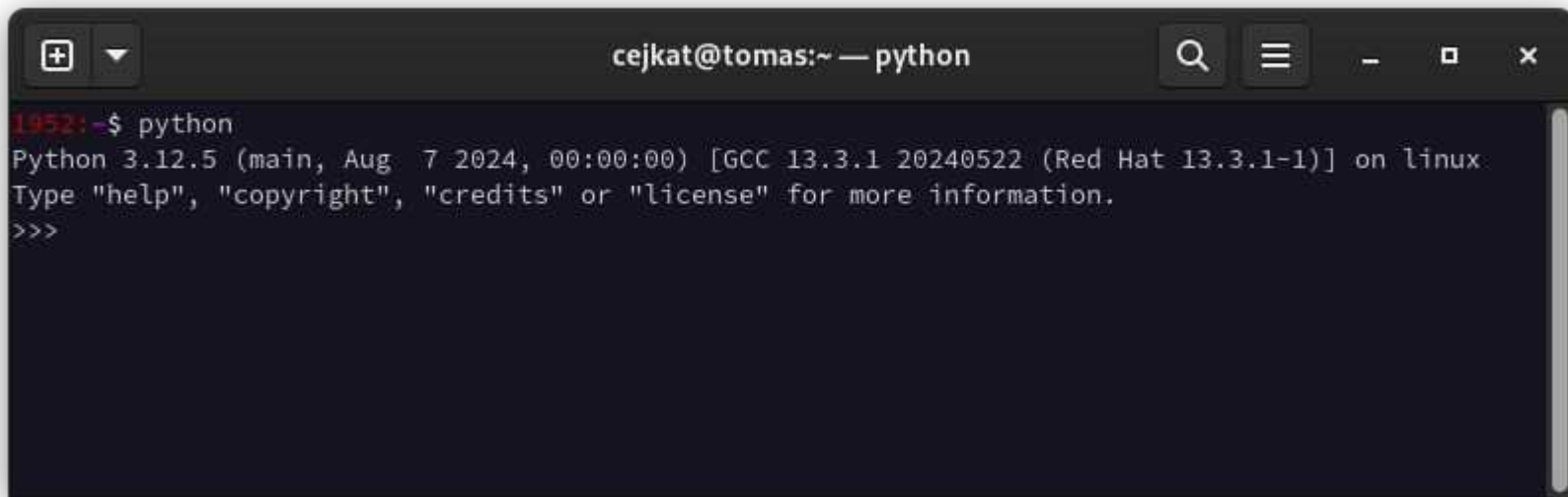
# Základy Pythonu, ale pro srdcaře (tedy v C)

doc. Ing. Tomáš Čejka, Ph.D.  
CESNET, zájmové sdružení právnických osob  
+ FIT ČVUT v Praze

[cejkat@cesnet.cz](mailto:cejkat@cesnet.cz)

Praha, 13.10.2024

- Python — skriptovací jazyk
- CPython — originální implementace Python interpretu vytvořena v C
  - obsahuje tzv. c-api — programovací rozhraní, struktury, makra, prototypy funkcí

A terminal window with a dark background. The title bar shows 'cejkat@tomas:~ — python'. The terminal content shows a shell prompt '1952:~\$' followed by 'python'. The output is 'Python 3.12.5 (main, Aug 7 2024, 00:00:00) [GCC 13.3.1 20240522 (Red Hat 13.3.1-1)] on linux' followed by instructions to type 'help', 'copyright', 'credits', or 'license' for more information, and a prompt '>>>' on the next line.

```
1952:~$ python
Python 3.12.5 (main, Aug 7 2024, 00:00:00) [GCC 13.3.1 20240522 (Red Hat 13.3.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

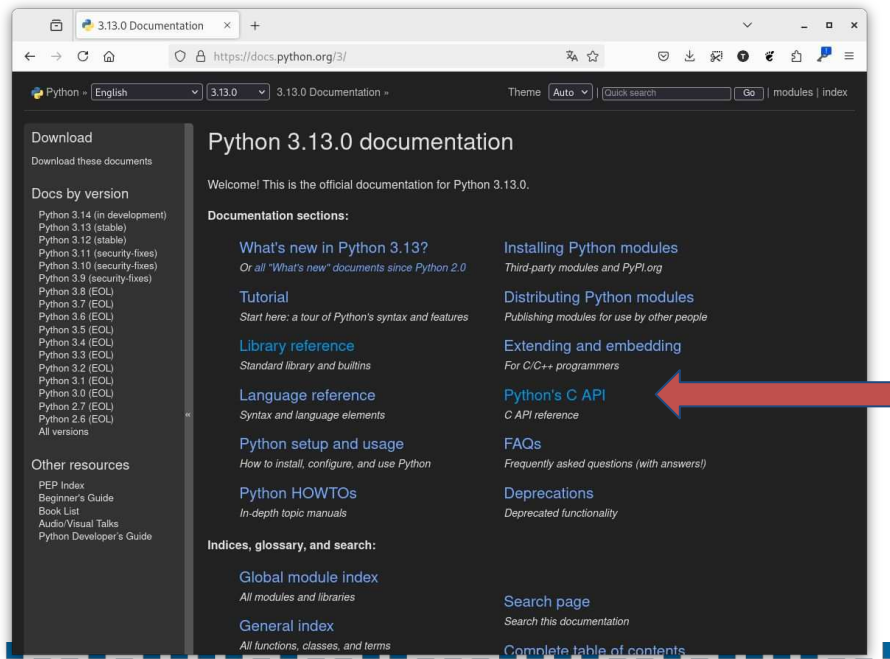
- Python — skriptovací jazyk
- CPython — originální implementace Python interpreteru vytvořena v C
  - obsahuje tzv. c-api — programovací rozhraní, struktury, makra, prototypy funkcí

## Dokumentace:

<https://docs.python.org/3/>

<https://docs.python.org/3/c-api/>

Hint: file:///usr/share/doc/python3-docs/html/c-api/index.html



- **Network Measurements Analysis (NEMEA)...**

- první části a moduly byly psány v jazyce **C / C++** → kompilované, výkonné, ...
- ... ale vývoj byl velmi náročný, náchylný na chyby, složitější na diagnostiku a opravy...

**Jak použít Python na “rychlejší” prototypování?**



- Python umožňuje volat funkce z binárních knihoven (shared object — .so)
  - modul `ctypes` <https://docs.python.org/3/library/ctypes.html>
- Mapování C struktur v bytes a Python hodnot
  - modul `struct` <https://docs.python.org/3/library/struct.html>

```
# ***** Load libtrap library *****

lib = CDLL("libtrap.so")

lib.trap_init.argtypes = (POINTER(ModuleInfo), IfcSpec)
lib.trap_init.restype = errorCodeChecker

def init(module_info, ifc_spec):
    """Initialize libtrap.
    module_info - structure created by CreateModuleInfo()
    ifc_spec - structure created by parseParams()
    """
    lib.trap_init(module_info, ifc_spec)
    terminate = lib.trap_terminate
    finalize = lib.trap_finalize
```

```
struct.unpack("="Q", bytes)[0]
```

#### Fundamental data types

[ctypes](#) defines a number of primitive C compatible data types:

ctypes type	C type	Python type
<a href="#">c_bool</a>	<code>_Bool</code>	bool (1)
<a href="#">c_char</a>	<code>char</code>	1-character bytes object
<a href="#">c_wchar</a>	<code>wchar_t</code>	1-character string
<a href="#">c_byte</a>	<code>char</code>	int
<a href="#">c_ubyte</a>	<code>unsigned char</code>	int
<a href="#">c_short</a>	<code>short</code>	int
<a href="#">c_ushort</a>	<code>unsigned short</code>	int
<a href="#">c_int</a>	<code>int</code>	int

Řešení postavené nad `ctypes` a `struct` (tzv. Python wrapper) pomohlo!

(<https://github.com/CESNET/Nemea-Framework/tree/edb4c4d7c15f0c2b69a481d1af6ee4feaf1365b4/python>,

cca rok 2015/2016)

Ale bohužel to nebylo dostatečně rychlé řešení...

(práce s daty byla v Pythonu stále pomalá)





## Home

scoder edited this page on May 7, 2020 · 10 revisions

### Cython: C-Extensions for Python, Wiki

This is a wiki for anything related to the [Cython](#) or [Pyrex](#) projects.

[Cython](#) is a language that makes writing C extensions for the Python language as easy as Python itself.

**Cython** is based on the well-known [Pyrex](#), but supports more cutting edge functionality and optimizations.

Development of Cython is partly motivated by the needs of [SAGE](#).

See our proposals for the [Google Summer of Code 2011](#).

<https://github.com/cython/cython/wiki>

[https://cython.readthedocs.io/en/latest/src/tutorial/cython\\_tutorial.html](https://cython.readthedocs.io/en/latest/src/tutorial/cython_tutorial.html)

- Python / Cython syntaxe, `cdef`, `python setup.py build_ext --inplace ...`





# CPython — C-API

- **Možnosti použití:**

- v C/C++ mohou používat/volat Python
- v C/C++ mohou napsat rozšíření (modul), který se dá volat z Pythonu
- v C/C++ programu mohou spouštět-interpretovat skripty

---

**Materiály k této přednášce:**

<https://github.com/cejkato2/linuxdays24>

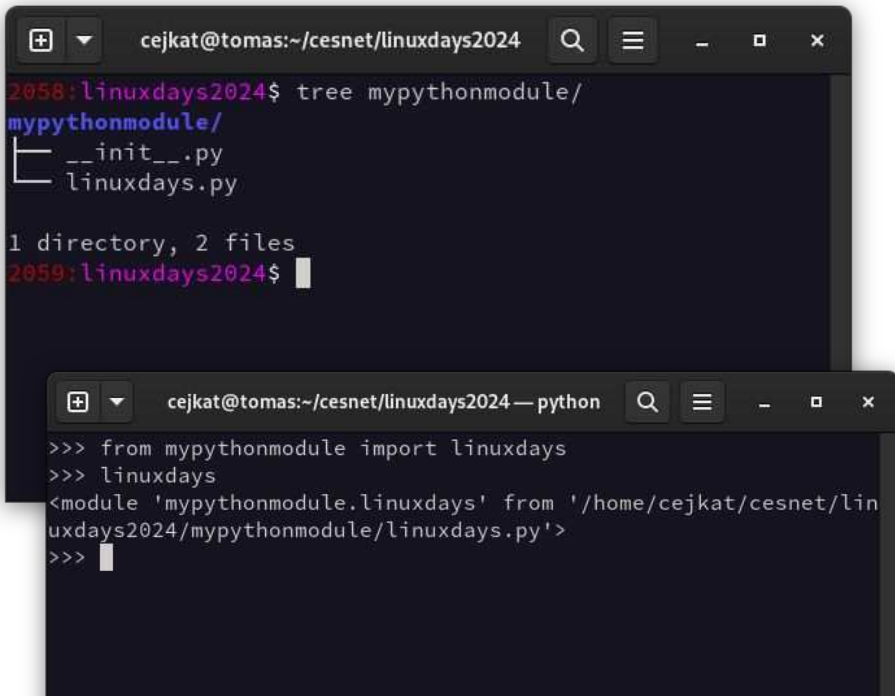
(větve gitu)



cesnet  
■■■■■

Modul

## Python



The image shows two terminal windows. The top window displays the command `tree mypythonmodule/` and its output, showing a directory structure with `__init__.py` and `linuxdays.py`. The bottom window shows a Python REPL session where the module is imported and the `linuxdays` variable is accessed, returning a module object.

```
cejkat@tomas:~/cesnet/linuxdays2024
2058:linuxdays2024$ tree mypythonmodule/
mypythonmodule/
├── __init__.py
└── linuxdays.py

1 directory, 2 files
2059:linuxdays2024$

cejkat@tomas:~/cesnet/linuxdays2024 — python
>>> from mypythonmodule import linuxdays
>>> linuxdays
<module 'mypythonmodule.linuxdays' from '/home/cejkat/cesnet/linuxdays2024/mypythonmodule/linuxdays.py'>
>>>
```

## Python

```
cejkat@tomas:~/cesnet/linuxdays2024
2058:linuxdays2024$ tree mypythonmodule/
mypythonmodule/
├── __init__.py
└── linuxdays.py

1 directory, 2 files
2059:linuxdays2024$

cejkat@tomas:~/cesnet/linuxdays2024 — python
>>> from mypythonmodule import linuxdays
>>> linuxdays
<module 'mypythonmodule.linuxdays' from '/home/cejkat/cesnet/linuxdays2024/mypythonmodule/linuxdays.py'>
>>>
```

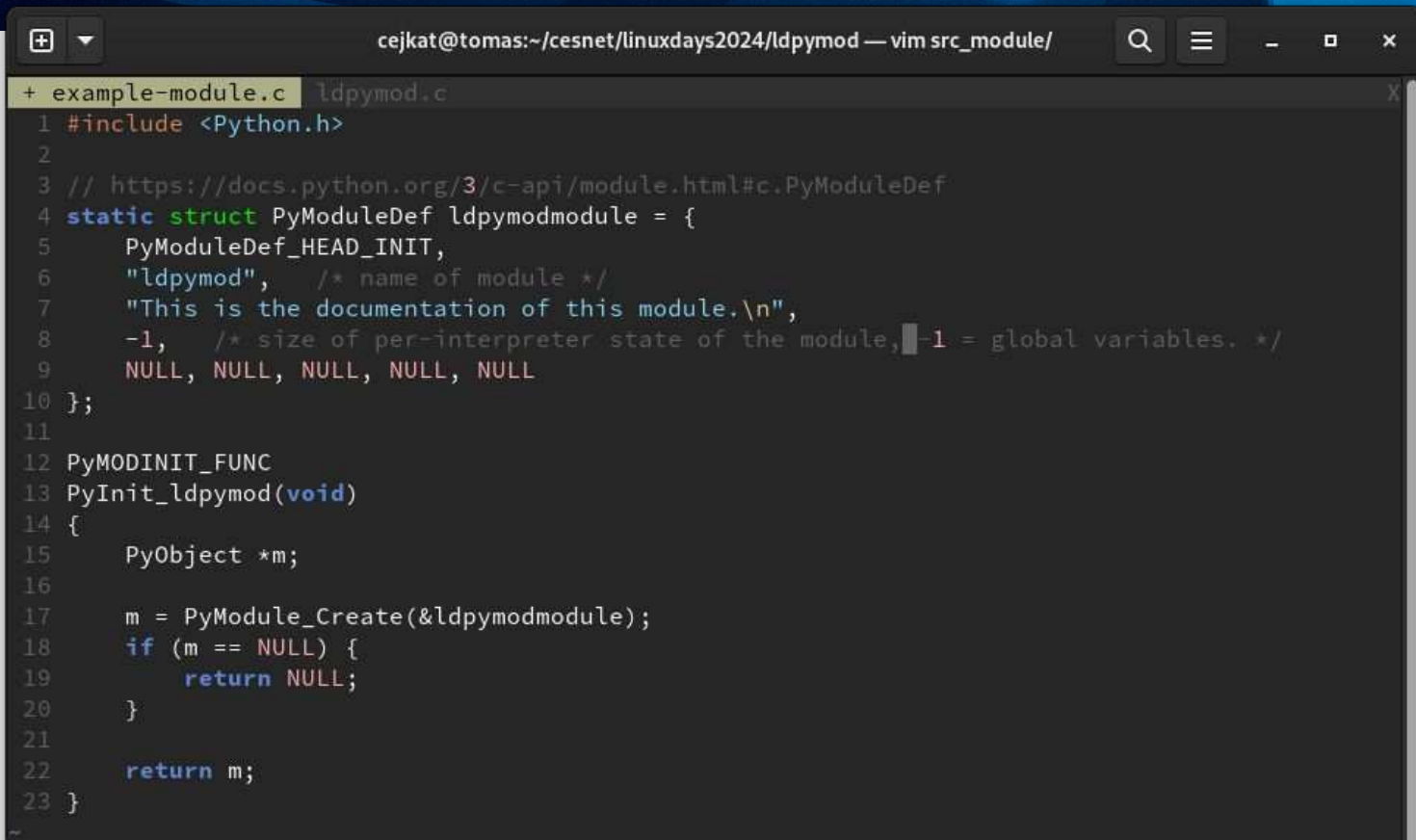
## C-API

```
2101:linuxdays2024$ tree ldpymod/
ldpymod/
├── pyproject.toml
├── README
├── src
│   └── ldpymod.c
└── tests
    └── test_ldpymod.py

2104:(master) ldpymod$ python3 -m build
...
Successfully built ldpymod-0.1.tar.gz and
ldpymod-0.1-cp312-cp312-linux_x86_64.whl

2331:(02_function) ldpymod$ tox
===== 1 passed in 0.02s =====
type: OK (3.31=setup[3.12]+cmd[0.19] seconds)
congratulations :) (3.38 seconds)
```





The screenshot shows a Vim editor window with the title bar "cejkat@tomas:~/cesnet/linuxdays2024/ldpymod — vim src\_module/". The editor is displaying the file "example-module.c" (highlighted in the tab bar) which contains the following C code:

```
1 #include <Python.h>
2
3 // https://docs.python.org/3/c-api/module.html#c.PyModuleDef
4 static struct PyModuleDef ldpymodmodule = {
5     PyModuleDef_HEAD_INIT,
6     "ldpymod", /* name of module */
7     "This is the documentation of this module.\n",
8     -1, /* size of per-interpreter state of the module, -1 = global variables. */
9     NULL, NULL, NULL, NULL, NULL
10 };
11
12 PyMODINIT_FUNC
13 PyInit_ldpymod(void)
14 {
15     PyObject *m;
16
17     m = PyModule_Create(&ldpymodmodule);
18     if (m == NULL) {
19         return NULL;
20     }
21
22     return m;
23 }
```

- pyproject.toml — extension, zdrojové soubory

```
python3 -m build          # sestavení balíku/modulu
```

```
tox                       # automatické testy
```

hint: PYTHONPATH — nastavení cesty k Python modulům

```
2335:(02_function) ldpymod$ PYTHONPATH=.tox/type/lib64/python3.12/site-packages/ python -c
'import ldpymod; print(dir())'

['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__',
'__spec__', 'ldpymod']
```

# Funkce v modulu

## Pole struktur s definicí metody PyMethodDef:

(<https://docs.python.org/3/c-api/structures.html#c.PyMethodDef>)

```
static PyMethodDef ldpymod_methods[] = {
    {"hello", (PyCFunction) ldpymod_hello, METH_NOARGS,
     "Get tuple with string and number.\n\n"
     "Returns:\n tuple(str, int)\n\n"},
    {NULL, NULL, 0, NULL}
};
```

## Přidání seznamu metod do objektu:

```
static struct PyModuleDef ldpymodmodule = {...
    ldpymod_methods,
    ...
};
```



# Konstanty v modulu

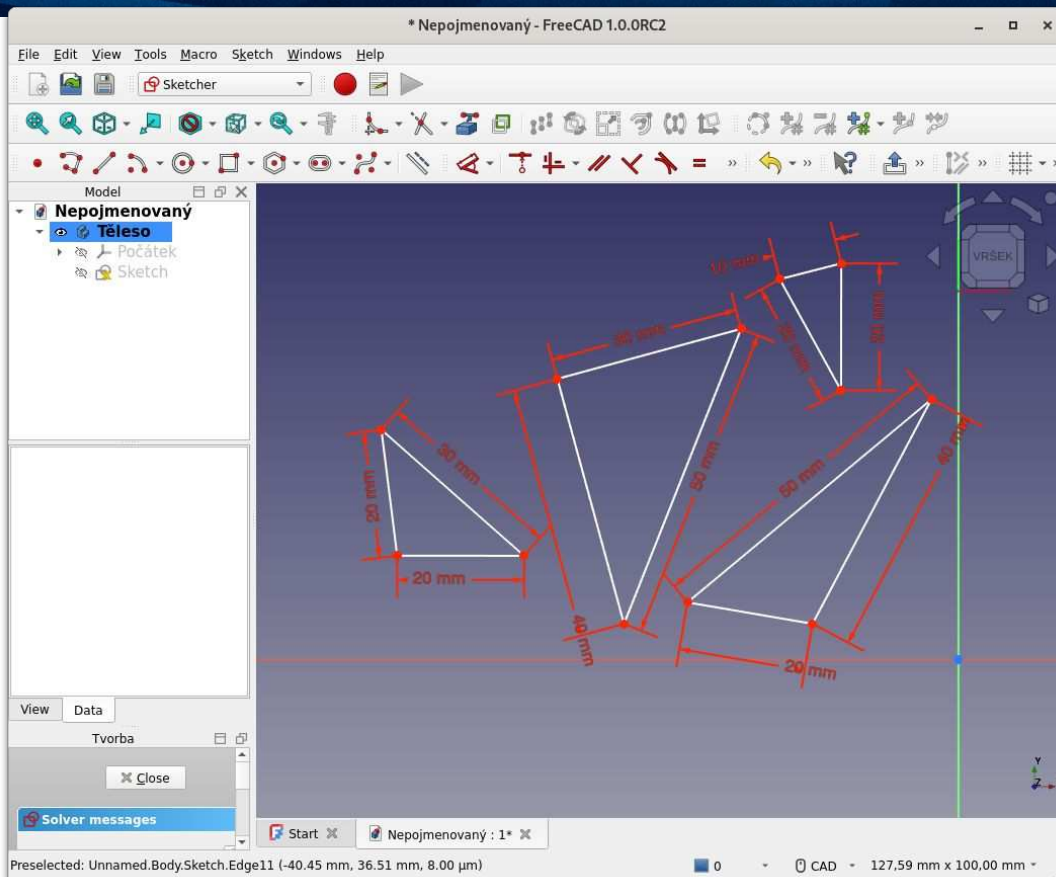


Přidání konstanty k modulu, příklad:

```
PyModule_AddIntConstant(m, "FMT_JSON", 2);
```

# “Praktický” příklad (Python vs C-API Python)

# “Praktický” příklad — celkový obsah trojúhelníků



Vstupy:

- Seznam trojúhelníků zadaných rozměry a, b, c
- Příklad: `seznam = [(2, 2, 3), (3, 3, 4)]`
- "Koeficienty pro škálování" `coef_a, coef_b, coef_c`

Chceme funkci:

```
l = [(2, 2, 3), (3, 3, 4)]  
spocti_plochu(l, 1, 1, 1)
```



# Výjimky



## Definice ukazatele:

```
PyObject *GeneralError;
```

## Přidání výjimek do modulu:

```
/* Add Exceptions into the module */  
GeneralError = PyErr_NewException("ldpymod.GeneralError", NULL, NULL);
```

```
Py_INCREF(GeneralError);
```



**Počítání referencí!**

```
PyModule_AddObject(m, "GeneralError", GeneralError);
```

cesnet  
". . . . ."

Objekt

## Definice struktury PyTypeObject:

```
PyTypeObject LinuxDaysObj = {  
    PyVarObject_HEAD_INIT(NULL, 0)  
    "ldpymod.LinuxDaysObj", /* tp_name */  
    sizeof(LinuxDaysObj_s), /* tp_basicsize */  
    ... };
```

## Definice struktury pro instance objektu:

```
typedef struct {  
    PyObject_HEAD  
    /* user-defined data: */  
    uint32_t counter;  
} LinuxDaysObj_s;
```

## Přidání objektu do modulu:

```
if (PyType_Ready(&LinuxDaysObj) < 0) {  
    return NULL;  
}  
Py_INCREF(&LinuxDaysObj);
```

```
PyModule_AddObject(m, "LinuxDaysObj", (PyObject *) &LinuxDaysObj);
```



# Metody objektu



## Pole struktur s definicí metody PyMethodDef:

```
static PyMethodDef lddobject_methods[] = {  
    {"area", (PyCFunction) lddobject_totalarea, METH_VARARGS |  
    METH_KEYWORDS, "PYDOC"},  
    {NULL, NULL, 0, NULL}  
};
```

## Přidání seznamu metod do objektu:

```
PyTypeObject LinuxDaysObj = {...  
    lddobject_methods, /* tp_methods */  
};
```



# Reference Counter

Každý objekt v Pythonu má čítač referencí, v C jej musíme hlídat při používání!

```
Py_INCREF(...);
```

```
Py_DECREF(...);      # - sníží počet referencí + uvolní paměť při 0
```

```
Py_XDECREF(...);     # - kontroluje zda argument není NULL
```

- Pokud neinkrementujeme → paměť může být smazána a my budeme přistupovat k nevalidním datům!
- Pokud nedekrementujeme → memory leaks!



Výsledek?

## Python implementace (Heronův vzorec)

```
def python_area(triangles, coef_a=1, coef_b=1,
                coef_c=1):
    def compute_area(a, b, c):
        s = (a + b + c) / 2
        return math.sqrt(s * (s - a) * (s - b) *
                           (s - c))
    sum_area = 0
    for t in triangles:
        a, b, c = t
        a *= coef_a
        b *= coef_b
        c *= coef_c
        sum_area += compute_area(a, b, c)
    return sum_area

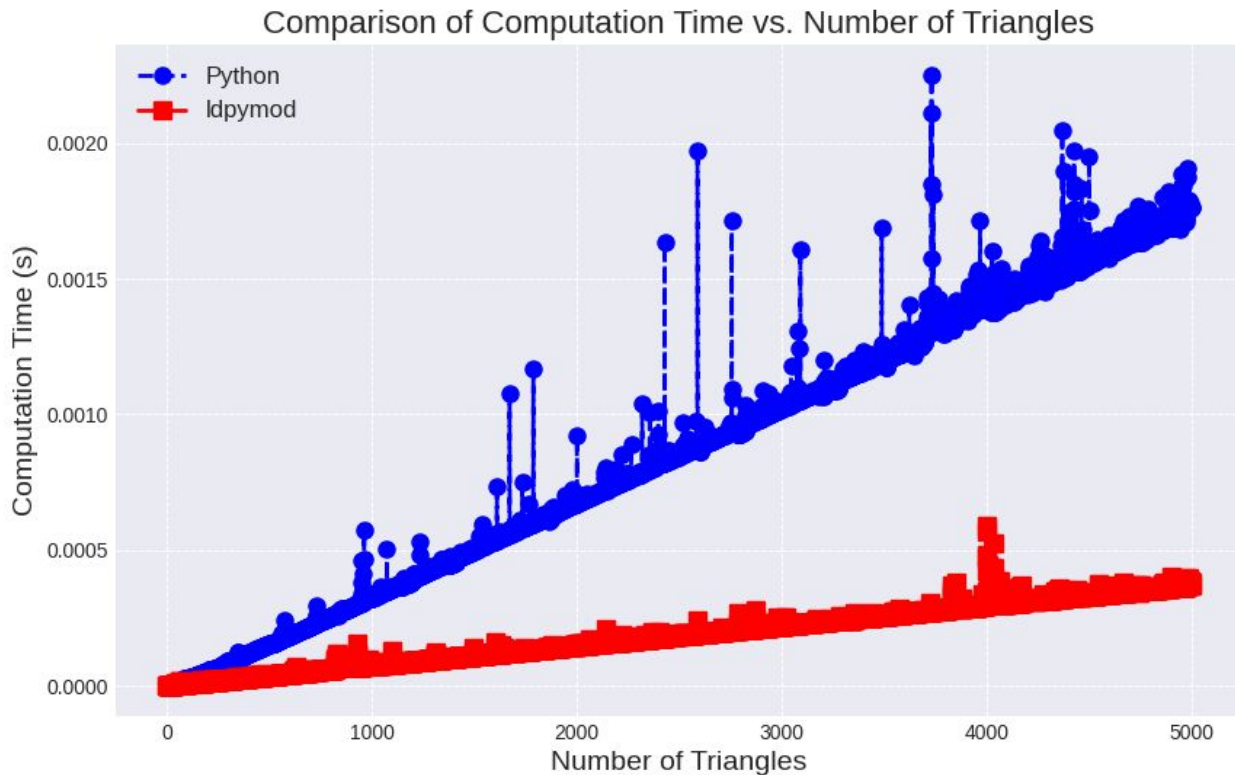
vysledek = python_area(triangles, coef_a,
                        coef_b,
```

```
                        coef_c)
```

## Použití našeho modulu:

```
import ldpymod

obj = ldpymod.LinuxDaysObj()
result = obj.area(triangles, coef_a, coef_b,
                  coef_c)
```







# Děkuji za pozornost

Tomáš Čejka

cejkat@cesnet.cz

cejkato2@fit.cvut.cz

<https://www.linkedin.com/in/cejkatomas/>

<https://fosstodon.org/@tomcejka>

<https://x.com/tomcejka>