23. Python отвътре

27 май 2026

Първо - обещахме ви малко за предното контролно!

- Но само ако колегата, който си го поиска е тук...

Регулярни изрази

- **Въпросът:**

13. Кой от низовете **няма** да бъде match-нат от следния регулярен израз - r"^\w+(-\d+)?(\.\w+)*$"?

А: "file_.pdf"
Б: "dickpick.jpg"
В: "document-21-01.doc"
Г: "nothing"
Д: "__private__.prv.file"

- Процент верни отговори: **38.9%**
- Този просто беше гаден...
- Верният отговор е **В**
- Защо?

Референции

- **Въпросът:**

- Процент верни отговори: **33.33%**
- Ще се възползвам отново да ви порисувам по дъската

9. Кое ще изведе True?

```
food = ['spam', 'eggs', 'more spam']
food[1] = food[:]
food[2] = food
```

А: food == food[1]
Б: food is food[1]
В: food[1][2] is food[2]
Г: food[2][1] is food[2]
Д: food[2][2] is food[2]

Римляни

- **Въпросът:**

10. Та какво са направили римляните за нас?

А: Нищо! Само ни отнеха всичко, което сме имали, и всичко, което са имали бащите ни, и бащите на бащите ни, и бащите на бащите на…
Б: акведукта, канализацията, пътищата, напоителната система, медицина, образование, мир, вино, обществени бани
В: ROMANES EUNT DOMUS!!!

- Процент верни отговори:
  30.6%
- Тц, тц тц…

Romani Ite Domum?

Рейнж

- **Въпросът:**

1. Какво е range?

А: Функция, която връща списък от цели числа в интервал
Б: Тип, чиито обекти могат да се индексират и обхождат интервал от цели числа
В: Бавен начин да правим цикли
Г: Специален comprehension
Д: Нито едно от посочените

- Процент верни отговори:
  27.8%
- Йоан отбеляза, че тук има един много верен отговор и един средно верен отговор
- Ако сте казали **Б** - имате точка
- Ако сте казали *В* - нашето уважение...
- Но не и точка

Рекурсия

- **Въпросът:**

- Процент верни отговори: **11.11%**
- Този и аз го объркаx, спокойно
- Не, не е рекурсия...
- Защото не е функция

26. Какво ще изведе следният код:

```
while type:
    type = type(type)
print(type)
```

А: False
Б: None
В: <class 'type'>
Г: RecursionError: maximum recursion depth exceeded
Д: Нищо

Name mangling

- **Въпросът:**

- Процент верни отговори: **5.5%**
- Да помислим - кои можем да изключим веднага?

8. Кое ще изведе True?

```
class Parent:
    __protected = 1

class Child(Parent):
    __protected = 1

    def get(self):
        return self.__protected
```

А: Parent.__protected == 1
Б: Child._Parent__protected is 1
В: Parent._Child__protected > 0
Г: Parent._Parent__protected is not Child._Child__protected
Д: Child.get() == 1
Е: Child().get == 1

Сега за следващото и оставям Йоан да ви говори

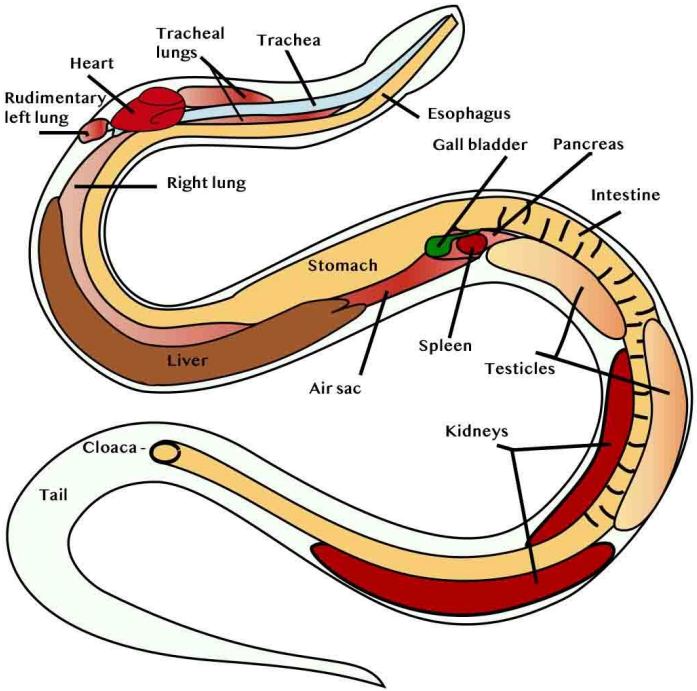
- Четвъртък, 11.06, от 19:00 спечели с един глас над сряда
- Но...
- Бях забравил да цъкна това:

- И двама души бяха гласували по два пъти... За четвъртък, 11.06 от 19:00
- С други думи!

Финално!

- Следващото контролно ще се проведе в **сряда, 10.06 от 19:00**
- Курсът реши!
- Ще запазим зала и ще пуснем новина, надяваме се да можем да се възползваме от 200, както обикновено

Какво знаем за Python?



Преговор - модули

Какво може да бъде модул в python?

- Файл с разширение `.py`
- Директория съдържаща файл с име `__init__.py`
- Можем да ги видим с `__file__` атрибута

```
>>> import functools
>>> functools.__file__
'C:\\dev\\Python311\\Lib\\functools.py'
```

Ho!

```
>>> import pyexpat
>>> pyexpat.__file__
'C:\\dev\\Python311\\DLLs\\pyexpat.pyd'
```

Преговор - байткод

Как може да видим кода на функция в Python?

- През `__code__` атрибута:

```
>>> (lambda: 42).__code__
<code object <lambda> at 0x7fc717a0dea0, file "<stdin>", line 1>
```

Ho!

```
>>> print.__code__
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'builtin_function_or_method' object has no attribute
'__code__'

>>> print
<built-in function print>
>>> print.__class__
<class 'builtin_function_or_method'>
```

Защо?

- Повечето Пайтън е написан на Пайтън
- ... например `datetime`, `functools`, etc.
- Но и доста е написано на C
- ... например `print`
- Понякога се прави за скорост
- Понякога се прави, защото няма друг начин

Python vs CPython

- Python
  - Абстрактната идея за езика
- CPython
  - Една конкретна имплементация
  - ... оригиналната имплементация
  - ... референтната имплементация
  - ... де факто стандартът на езика

CPython

- интерпретиран
- байткод-базиран
- динамичен
- ядро написано на C
- стандартна библиотека на Python
- ... и малко C

Къде е кода?

- `.py` е сорс код на Python
- `.pyc` е кеширан байткод
- Можем да ги намерим в Питон-дистрибуцията
  - Във Windows
    `%PYTHONPATH%`
  - Във Unix (и приятели)
    `/usr/lib/python*/**`
    `/usr/bin/python`

builtins

- Горното важи за функции написани на Python
- Както споменахме – не всичко в Python е написано на Python

код?

код!

Специални функции в C кода

Забелязахте ли pattern-а в имената на функциите в C кода?

- `PyObject_print`, `PyObject_Str`, `PyErr_format`, …
- pattern-а е `Py(Type)_(Function)`

Не се отнася само за функции

... Цели типове са имплементирани на C

```
>>> int
<class 'int'>
>>> int.x = 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't set attributes of built-in/extension type 'int'
```

Още код!

Вградени типове

- `int, float, str, list, dict, set`
- `map` и `filter` също са вградени типове, а не фунцкии
- има още неща в `__builtins__`

Още примери за C функции

- `PyDict_New`, `PyDict_Contains`, `PyDict_SetItem`
- `PyDict_SetItemString`, `PyDict_GetItem`
- `PyDict_GetItemString`, `PyDict_Size`, `PyDict_Merge`, `PyDict_Update`
- `PyList_New`, `PyList_Size`, `PyList_GetItem`
- `PyList_SetItem`, `PyList_Insert`, `PyList_Append`
- `PyList_GetSlice`, `PyList_Sort`, `PyList_Reverse`
- и т.н.

С кодът зад Python

- Чрез тези функции е имплементиран Пайтън
- Това е т.нар. Python C API
- ... ние можем също да ги използваме
- built-in (вграден) е малко подвеждащо име
- ... ние също можем да си дефинираме “built-in” функции и класове

Какво означава “вграден”

- Всичко вградено ли се намира в `__builtins__`?

```
import _sqlite3

_sqlite3.connect
# <built-in function connect>
```

- Малко повече за този модул...

```
_sqlite3.__file__
# '/usr/lib/python3.11/lib-dynload/_sqlite3.cpython-311-x86_64-linux-gnu.so'
# ... или под Windows:
# 'C:/python3.11/DLLs/_sqlite3.pyd'
```

Модули - втори път

Всъщност модул в python може да бъде:

- Файл с разширение `.py`
- Директория съдържаща файл с име `__init__.py`
- Файл с разширение `.pyd/.so`

Native модули

- `.pyd` е преименуван `.dll` файл (dynamic link library)
- `.so` си е `.so` (shared object)
- shared object/dynamic link library са два термина за едно и също нещо
- представляват C библиотека
- ... обикновено в комбинация с `.h` (header) файла

Какво съдържа един .dll/.so

- Съдържа изпълним код, import таблица, export таблица
- Последната съдържа имената на функциите
- Не се съдържат, обаче, дефиниции на типове
- Не се съдържат аргументите на функциите
- Как тогава можем да компилираме код, който използва функции от такива DLL-и?
- С header файлове. Те съдържат дефинициите на типове, декларации на функции и т.н.

Демо - C стандартна библиотека

- Под Windows:
  ```
  dumpbin /exports C:\Windows\System32\msvcrt.dll
  ```
- Под Linux:
  ```
  nm -gD nm -gD /usr/lib/libc.so.6
  ```

Демо - Python C API

- Под Windows:
  ```
  dumpbin /exports C:\Python3.11\DLLs\python3.dll
  ```
- Под Linux:
  ```
  nm -gD nm -gD /usr/lib/x86_64-linux-gnu/libpython3.11.so.1.0
  ```

Python C API - хедъри

В общия случай се намират:

- Под Linux: `/usr/include/python{version}/`
- Под Windows: `C:\python{version}\include`

Python.h

- Декларации на функции
- Разни макроси, които трябват на Python

C vs Python

- `int` в C не е като `int` в Пайтън
- `string` в C не е като `string` в Пайтън
- Когато искаме да подаваме стойности от едното място към другото, се налага да правим някакво преобръщане
- ... то се нарича **marshalling**

Marshalling към Python

```
Py_BuildValue("s", "spam") -> 'spam'
Py_BuildValue("i", 42) -> 42
Py_BuildValue("(sii)", 42, "hi", 8) -> (42, 'hi', 8)
Py_BuildValue("{is,is}", 1, "one", 2, "two") -> {1: 'one', 2: 'two'}
Py_BuildValue("") -> None
```

Marshalling към C

```
const char* str;
int number;
PyArg_ParseTuple(args, "si:string_peek", &str, &number);
```

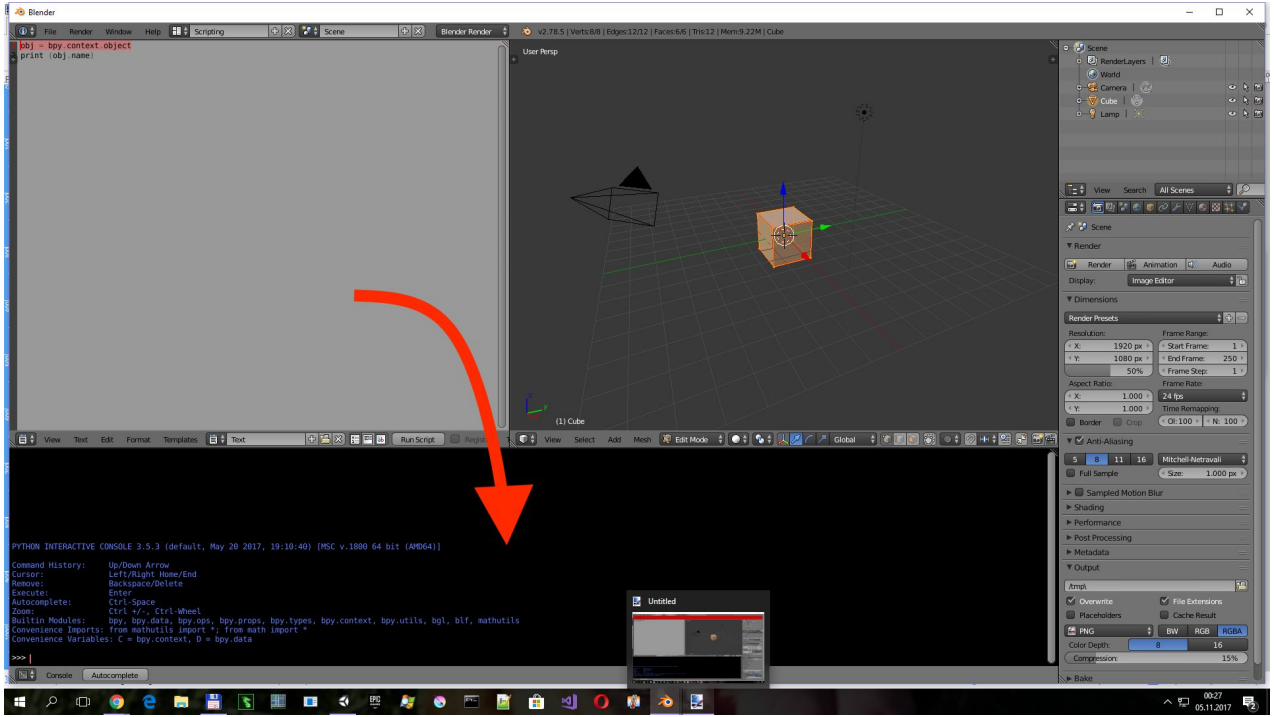
Защо C API модули?

- Пайтън е бавен! (Numpy, Pandas)
- Преизползване на примитиви на OS (posix, win32, Cocoa + pyobjc)
- Преизползване на готов код (MySQL, PostgrSQL, Qt, PyGTK+,)
- Достъп до хардуер (Tensorflow, PyTorch, PyOpenGL, vulkan)
- Искам си указателите!

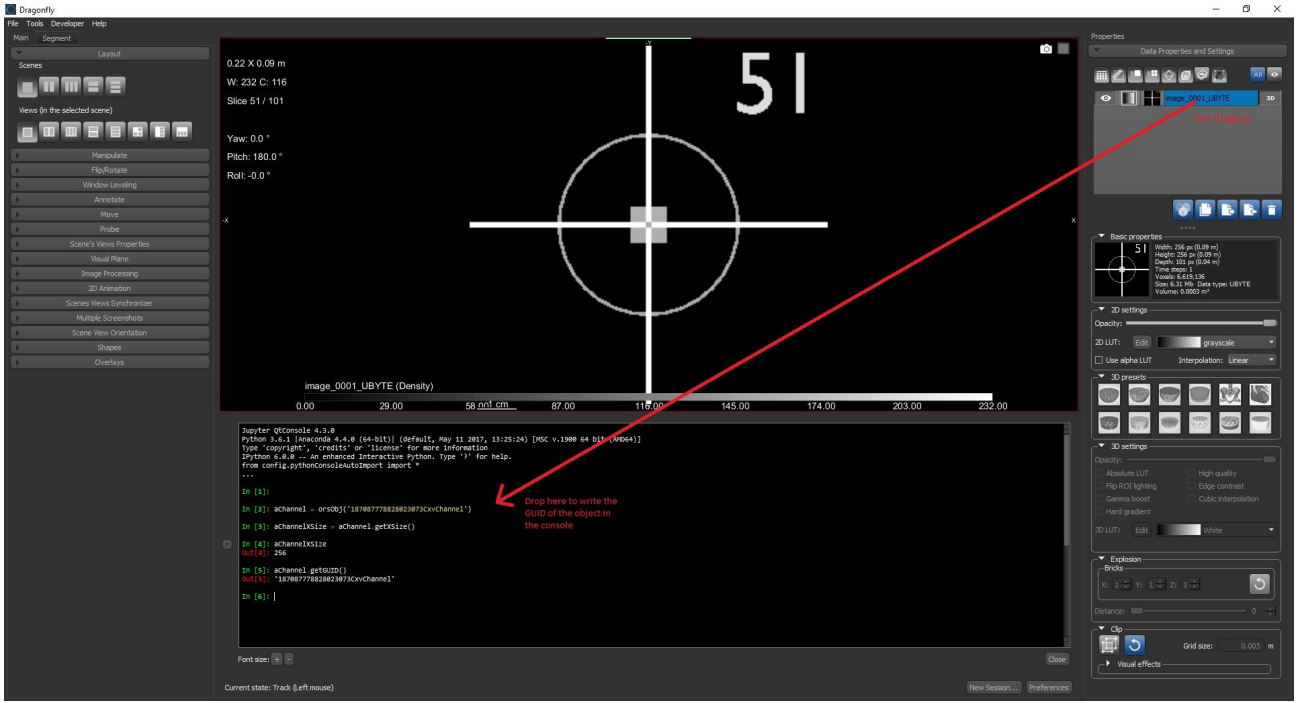
C API отвъд модулите

- python.exe
  - Просто програма, която използва C API (в Programs/python.c)
  - Създава интерпретатор
  - Вика PyRun_InteractiveLoop (дефинирана в Python/pythonrun.c)
- Можем да направим наш интерпретатор
  - Проста конзолна програма
  - ... или нещо графично и шарено (например IDLE)
- Можем да вградим интерпретатор в друг софтуер
  - Чрез C API даваме предоставяме Python-ски интерфейс
  - ... и нашият софтуер може да бъде използван чрез код вместо чрез GUI
  - ... или може да бъде разширяван чрез Python plugin-и

Blender


Blender
File Render Window Help Scripting Scene Blender Render
obj = bpy.context.object
print (obj.name)
User Persp
(1) Cube
View Text Edit Format Templates Text Run Script
View Select Add Mesh Edit Mode Global
PYTHON INTERACTIVE CONSOLE 3.5.3 (default, May 20 2017, 19:10:40) [MSC v.1800 64 bit (AMD64)]
Command History: Up/Down Arrow
Cursor: Left/Right Home/End
Remove: Backspace/Delete
Execute: Enter
Autocomplete: Ctrl-Space
Zoom: Ctrl +/-, Ctrl-Wheel
Builtin Modules: bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, blf, mathutils
Convenience Imports: from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data
>>>
Console Autocomplete
Untitled
Scene
RenderLayers
World
Camera
Cube
Lamp
Render
Dimensions
Anti-Aliasing
Sampled Motion Blur
Shading
Performance
Post Processing
Metadata
Output
Bake
00:27
05.11.2017

AutoCad


Dragonfly
File Tools Developer Help
Main Segment
Layout
Scenes
Views (in the selected scene)
Manipulate
Flip/Rotate
Window Leveling
Annotate
Move
Probe
Scene's Views Properties
Visual Plane
Image Processing
2D Animation
Scenes Views Synchronizer
Multiple Screenshots
Scene View Orientation
Shapes
Overlays
0.22 X 0.09 m
W: 232 C: 116
Slice 51 / 101
Yaw: 0.0 °
Pitch: 180.0 °
Roll: -0.0 °
51
image_0001_UBYTE (Density)
0.00 29.00 58.00 87.00 116.00 145.00 174.00 203.00 232.00
1 cm
Jupyter QtConsole 4.3.0
Python 3.6.1 |Anaconda 4.4.0 (64-bit)| (default, May 11 2017, 13:25:24) [MSC v.1900 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.0.0 -- An enhanced Interactive Python. Type '?' for help.
from config.pythonConsoleAutoImport import *
In [1]:
In [2]: aChannel = orsObj('1870877788280230 73CxvChannel1')
In [3]: aChannelXSize = aChannel.getXSize()
In [4]: aChannelXSize
Out[4]: 256
In [5]: aChannel.getGUID()
Out[5]: '1870877788280230 73CxvChannel1'
In [6]:
Drop here to write the GUID of the object in the console
Font size:
Close
Current state: Track (Left mouse)
New Session...
Preferences
Properties
Data Properties and Settings
Basic properties
2D settings
Opacity:
2D LUT:
Edit
grayscale
Use alpha LUT
Interpolation: Linear
3D presets
3D settings
Explosion
Bricks
Distance:
Clip
Grid size:
Visual effects

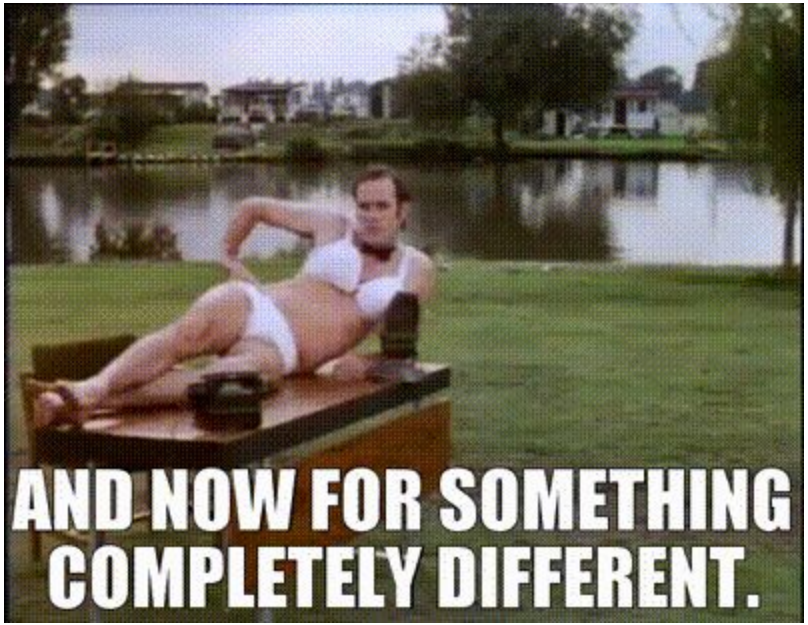
Много други

- Maya
- IDA Pro
- Ghidra
- GIMP
- Inkscape
- FreeCAD
- Sublime Text
- Houdini
- Nuke
- QGIS
- Scribus

… и разни игри

- EVE Online
- World of Tanks
- Battlefield 2
- Civilization IV
- Temple of Elemental Evil
- Vampire the Masquerade: Bloodlines
- Godot Engine

- Повечето – по-стари
- Напоследък за игри се предпочита Lua / JavaScript
- … заради по-лесния sandboxing


AND NOW FOR SOMETHING
COMPLETELY DIFFERENT.

Таксономия

- Interpreter vs translator (в контекста на естествените езици)
  - симултантен превод / превод
- Алтернативи на интерпретаторите?
  - компилатори
  - JIT компилатори
- Алтернативи на CPython?

Интерпретирани питони

- CPython
  - референтната имплементация
- Stackless Python
  - call frame-овете са в heap-а вместо на системния стек
  - зелени нишки!

Интерпретирани питони (2)

- Разни други за микроконтролери
  - MicroPython
  - CircuitPython
  - Tinypy
- Разни други за browser
  - Brython
  - Skulpt
  - Transcrypt

JIT-компилирани питони

- PyPy
  - Поддържа подмножество на Python
- Pyston
  - LLVM-базиран
- IronPython
  - .NET
  - Python 3.4
- Jython
  - JVM
  - Слаба (и неофициална) Python 3 поддръжка
- GraalPython
  - GraalVM (алтернативна JVM имплементация)

Компилирани питони

- Cython
  - Надмножество на Python
  - Транспилира до C или C++
  - Може да се ползва за писане CPython модули
  - Съвместим с всякакви имплементации, поддържащи C API
- Mojo
  - Надмножество на Python
  - Частично closed source
  - В публична beta
  - Много hype; малко употреба в реалния свят
- Nuitka
  - Използва чист Python синтаксис
- mypyc
  - Използва типови анотации при компилация

Алтернативни имплементации

- Може да са няколко версии назад със синтаксиса
- Може да липсват части от стандартната библиотека
- Често може да нямат C API поддръжка
- ... или да имат ограничена C API поддръжка
- Горните липси често са умишлен избор, а не дефект
- ... за сметка на комуникацията с други езици
  - GraalVM/Jython могат лесно да импортират и викат Java библиотеки
  - IronPython може да импортира и вика .NET/C# библиотеки
  - Brython/Skulpt/Transcrypt могат да викат/ползват JavaScript/DOM
- Или други предимства

Алтернативни имплементцтации (2)

Често не са толкова прости за класификация

- CPython 3.11+ има и JIT
  - ... но е експериментален и изключен по подразбиране
- GraalPython и Jython поддържат интерпретиране
  - ... защото JVM може да работи в режим на интерпретиране
- GraalPython и IronPython поддържат частична компилация
  - ... защото GraalVM и .NET поддържат ahead-of-time компилацията
- Cython се очаква да бъде смесван със CPython
  - Части от Python кода са компилирани; части - интерпретирани
  - Нека разгледаме пример...

Cython - алтернатива на C API

- Позволява ни да пишем [почти] Python
- Сорс файлове с разширение `.pyx`
- Произвежда native (`.pyd`/`.so`) модул
- Вика Python C API без да пишем C код

Cython пример

```
# fib.pyx

def fib(n):
    """Print the Fibonacci series up to n."""
    a, b = 0, 1
    while b < n:
        print(b)
        a, b = b, a + b
```

Cython компилация като пакет

- `$ pip install cython`
- обикновено го слагаме в пакет (който качваме pip)
- Във файл с име `setup.py`:

```
from setuptools import setup
from Cython.Build import cythonize

setup(
    ext_modules=cythonize("fib.pyx"),
)
```

- `$ python setup.py build_ext --inplace`

Cython - компилация в работна среда

- Има алтернативен вариант (удобен при честа промяна на кода)
- В същата директория като `fib.pyx` правим `fib.pyxbld`:

```
from distutils.extension import Extension

def make_ext(modname, pyxfilename):
    return Extension(name=modname, sources=[pyxfilename], language='c')
```

- В python:

```
>>> import pyximport
>>> pyximport.install(build_dir = '/some/path')
>>> import fib
>>> fib
<module 'fib' from '[...]/fib.cpython-38-x86_64-linux-gnu.so'>
```

Cython - компилация в работна среда (2)

- Работи тривиално в *nix
- Може да се окаже по-пипкаво за подкарване в Window
- Работи, защото import системата в python е pluggable
- Можем да си регистрираме свои разширения за import
  (в случая `.pyxbld`)
- За сравнение – Hy – Lisp диалект работещ върху Python
  https://docs.hylang.org/en/stable/interop.html#using-hy-from-python

Cython - компилация в работна среда (3)

```
>>> fib.fib
<cyfunction fib at 0x7f368b750380>
>>> fib.fib(7)
0
1
1
2
3
5
```

Cython – output

- `.pyd/.so` файл (основно)
- код на C (междинен файл)

Последният да затвори вратата

```
static PyObject *__pyx_f_3fib_fib(PyObject *__pyx_v_n, CYTHON_UNUSED int __pyx_skip_dispatch) {
  PyObject *__pyx_v_a = NULL;
  PyObject *__pyx_v_b = NULL;
  PyObject *__pyx_r = NULL;
  __Pyx_RefNannyDeclarations
  PyObject *__pyx_t_1 = NULL;
  PyObject *__pyx_t_2 = NULL;
  int __pyx_t_3;
  int __pyx_lineno = 0;
  const char *__pyx_filename = NULL;
  int __pyx_clineno = 0;
  __Pyx_RefNannySetupContext("fib", 1);

  __pyx_t_1 = __pyx_int_0;
  __Pyx_INCREF(__pyx_t_1);
  __pyx_t_2 = __pyx_int_1;
  __Pyx_INCREF(__pyx_t_2);
  __pyx_v_a = __pyx_t_1;
  __pyx_t_1 = 0;
  __pyx_v_b = __pyx_t_2;
  __pyx_t_2 = 0;

  while (1) {
    __pyx_t_2 = PyObject_RichCompare(__pyx_v_b, __pyx_v_n, Py_LT); __Pyx_XGOTREF(__pyx_t_2); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 7, __pyx_L1_error)
    __pyx_t_3 = __Pyx_PyObject_IsTrue(__pyx_t_2); if (unlikely((__pyx_t_3 < 0))) __PYX_ERR(0, 7, __pyx_L1_error)
    __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
    if (!__pyx_t_3) break;
    __pyx_t_2 = __Pyx_PyObject_CallOneArg(__pyx_builtin_print, __pyx_v_b); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 8, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_2);
    __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;

    __pyx_t_2 = __pyx_v_b;
    __Pyx_INCREF(__pyx_t_2);
    __pyx_t_1 = PyNumber_Add(__pyx_v_a, __pyx_v_b); if (unlikely(!__pyx_t_1)) __PYX_ERR(0, 9, __pyx_L1_error)
    __Pyx_GOTREF(__pyx_t_1);
    __Pyx_DECREF_SET(__pyx_v_a, __pyx_t_2);
    __pyx_t_2 = 0;
    __Pyx_DECREF_SET(__pyx_v_b, __pyx_t_1);
    __pyx_t_1 = 0;
  }
```

Въпроси?