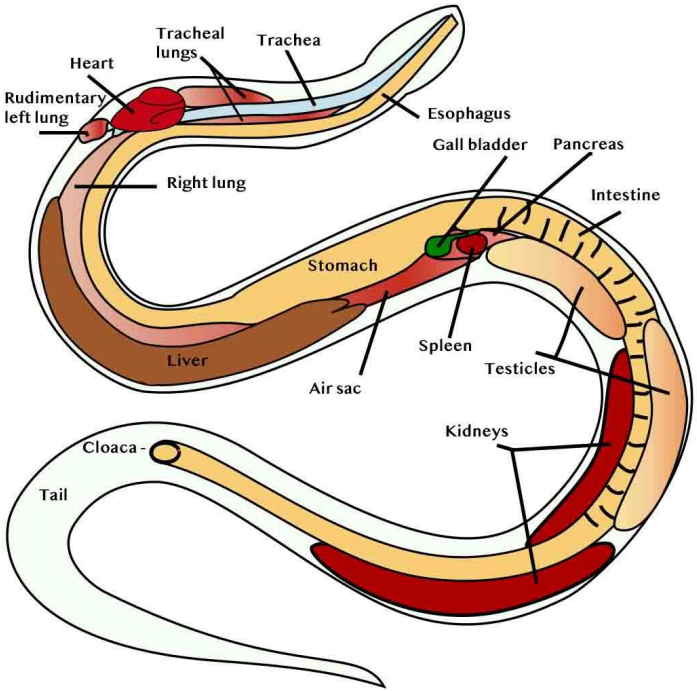
18. Python отвътре

12 декември 2023

Какво знаем за Python?



Python

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

Интерпретатор?

- Interpreter vs translator
  (симултантен превод / превод)
- Алтернативи на интерпретаторите?

Таксономия

- компилатори
- JIT компилатори
- интерпретатори

Компилирани езици

Популярни примери:

- C
- C++
- Rust

Компилатори (подготовка)

- синтактичен анализ
- оптимизация
- транслация към изпълним код
  - изпълними файлове (`.exe`, `a.out`)
  - библиотеки (`.dll`, `.so`)
  - формати (PE/ELF/COFF/Mach-O)

Компилатори (изпълнение)

- зарежда се изпълним файл
- зареждат се библиотеки
- обработват се секциите в тях (данни, код, ресурси)
- процесорът изпълнява машинния код

Какво е машинен код?

- assembler?
- нули и единици?
- opcodes

JIT компилирани езици

Популярни примери:

- C#
- Java

JIT компилатори (подготовка)

- синтактичен анализ
- транслация към байткод
  - `.jar` (Java)
  - `.dll` (C#; тези dll-и са малко по-различни)

JIT компилатори (изпълнение)

- зарежда се виртуална машина (runtime)
  - виртуалната машина обикновено е native code
  - “native code” е просто машинен код
- зарежда се байткод
- при първо извикване байткодът се компилира
- при нужда - profiling / optimization

Какво е байткод

- opcodes
- инструкции (като асемблер)
- нули и единици
- подобна идея като машинния код
- ... но за виртуален процесор

Интерпретирани езици

- Python
- Ruby
- JavaScript
- Perl

Интерпретатори (подготовка)

???

Интерпретатори (подготовка)

... кодът си остава просто текст

Интерпретатори (изпълнение)

- зарежда се интерпретатор
  - обикновено е native code
- зарежда се сорс код
- синтактичен анализ
- вътрешна (in-memory) репрезентация
  - AST
  - bytecode
- Вътрешната репрезентация се изпълнява

Конкретно за Python

- Има байткод
- стандартна библиотека (на Python)
  - `.py`
  - `.pyc`
- `python.dll`
  - builtin типове
  - builtin функции
- можем да си напишем наши builtins
  - `.pyd` (Windows)
  - `.so` (Linux)

Върху файловата система

- Във Windows
  `%PYTHONPATH%`
- Във Unix (и приятели)
  `/usr/lib/python*/**`
  `/usr/bin/python`

.py / .pyc

- `.py` е сорс код на Python
- `.pyc` е кеширан байткод

... нека разгледаме Python байткода в детайли!

Примерна функция

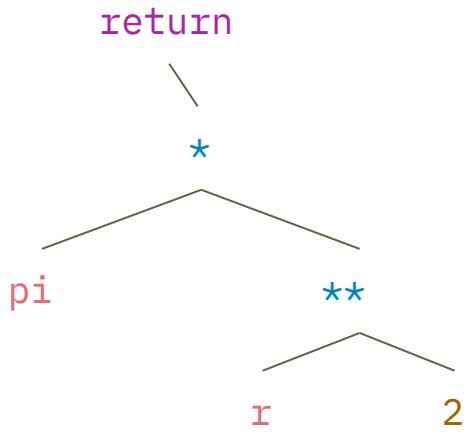
```
from math import pi

def circle_area(r):
    return pi * (r ** 2)

print(circle_area.__code__.co_code)
# b't\x00j\x01|\x00d\x01\x13\x00\x14\x00S\x00'
```

import dis

```
>>> import dis
>>> dis.dis(circle_area.__code__.co_code)
          0 LOAD_GLOBAL              0 (0)
          4 LOAD_FAST                0 (0)
          6 LOAD_CONST               1 (1)
          8 BINARY_POWER
         10 BINARY_MULTIPLY
         12 RETURN_VALUE
```

Абстрактно синтактично дърво (AST)

- Дървовидна репрезентация на програма
- Тялото на `circle_area` (грубо) изглежда така:



Инфиксен запис

- Обхождаме AST в дълбочина
- Обхождаме в ред ляво-корен-дясно
- Получаваме:
  ```
  (return (pi * (r ** 2)))
  ```
- **Забележка 1:** ако знаем приоритетите не са нужни скоби
- **Забележка 2:** като “нормален” език за програмиране

Префиксен (полски) запис

- Обхождаме AST в дълбочина
- Обхождаме в ред корен-ляво-дясно
- Получаваме:
  `(return (* pi (** r 2)))`
- **Забележка 1:** Това е валиден Lisp (Scheme) код:
  `(* pi (expt r 2))`
- **Забележка 2:** ако знаем арността не са нужни скоби

Суфиксен (обратен полски) запис

- Обхождаме AST в дълбочина
- Обхождаме в ред ляво-дясно-корен
- Получаваме:
  ```
  ((pi (r 2 **) *) return)
  ```
- **Забележка 1:** ако знаем арността не са нужни скоби
- **Забележка 2**: това е валиден PostScript код:
  ```
  PI r 2 exp mul
  ```

Байткод!

```
((pi (r 2 **) *) return) - със скоби
pi  r  2  **  *  return - без скоби
```

```
LOAD_GLOBAL 0          ⇔   pi
LOAD_FAST 0       ⇔   r
LOAD_CONST   1         ⇔   2
BINARY_POWER      ⇔   **
BINARY_MULTIPLY   ⇔   *
RETURN_VALUE      ⇔   return
```

Байткод (2)

- Python байкода е обратен полски запис на AST-то
- ... подобно на JVM (Java)
- ... подобно на .NET (C#)
- ... подобно на p-code (Pascal)

import ast

Модул, с който да сглобяваме AST/код

CPython

- Всичко, което видяхме са имплементационни детайли!
- Важат за CPython

Python имплементации

Python:

- CPython # интерпретиран
- IronPython # JIT върху .NET
- Jython # JIT върху JVM
- Pyston # JIT чрез LLVM
- Cython # компилиран (с уговорки)
- Unladen Swallow # JIT; dead

Език vs имплементация

Python не е специален!

JavaScript:

- V8 (JIT)
- Chakra
- JavaScriptCore
- SpiderMonkey
- Rhino (JIT върху Java)

Една имплементация

OpenJDK притежава:

- HotSpot (JIT компилатор)
- GraalVM (JIT компилатор/интерпретатор/AOT (ahead-of-time) компилатор)
- Project Zero (JIT компилатор)
- байткод интерпретатор

JIT компилатори/интерпретатори

- разграничението често е условно
- ... и зависи от имплементацията
- на теория: теория == практика
- на практика: теория != практика

Въпроси?