Type inference for dynamic languages

Tomáš Kulich, Ľubo Žák, Dominik Kapišinský
Seminár teoretickej informatiky, 10.5.2013

Outline

- Short introduction to Python, overview of its (scary?) dynamic features
- Why dynamic languages? Why Python?
- Problem definition
- Overview of existing solutions

Python

Python

- not a snake!
- written by Guido van Rossum (BDFL)

  as a Christmas passtime in '89
- Just one representant of wider family of languages (Python, Ruby, Groovy, Clojure, Javascript, etc..) (not Scala!)
- Chosen for its maturity
- Scripting language, but not only scripting language

What does it mean to be dynamic?

- **Object is just 'sack' full of objects and functions bound to their names**
- Attributes can be dynamically added
- Functions are first-class objects, e.g. attributes
- Methods are just functions
- Class is just 'template' for creating objects, it has no deeper meaning

Common Python outcries

I know JavaScript and it sucks

- JavaScript is connected to DOM, it is DOM that is broken
- No proper IDE
- JS community is full of lousy coders and lousy code
- If error, 'fail as late as possible' strategy (this is bad!)

"*Python is Javascript done right*" (one young CTO)

- Broken ==

```
"" == 0 //true

0 == "0" //true

"" == "0" //false
```

- Casting like mad

```
[]+[] === "" //true

{}+[] === 0 //true

isNaN({}+{}) //true

[1,2,3]+4 === '1,2,34' //true

(from Gary Bernhardt's WAT)
```

Indentation for specifying code blocks? You must be joking!

Just try it!

Code must be type safe

(or: if you do not have compile time type safety, the code must broke all the time)

“*What kind of type safety checks if you convert inches to centimeters correctly?*” (paraphrased from GvR's speech)

If you do not test, it'll get nasty in any language

Access modifiers are esential

(Access modifiers = private, public, protected, ..)

- Does Java really respect access modifiers?
  - reflection model
  - bytecode manipulation
- "*We don't lock the door, we take our tie and put it on the door and say that the room is occupied. If you come in, what you see and how you react is your own problem.*"

  (Raymond Hettinger, author of the itertools library)

Java evolution (skeptic version)

1. Restrict everything
2. Oops, those feature WERE usefull, let's get them back using bytecode manipulation!
3. Gazilion of frameworks / libraries are coded for this sole purpose
4. Almost every one is full of dark magic, almost none of these is compatible with any other one

Java EE is not solution, at least for now.

Python is SO slow

- WAY faster than PHP and who cares?
- PYPY
- Great C extensions (numpy, scipy,...)

“Premature optimization is root of all evil”
(common knowledge)

You can do dynamic.. So what?

- You can prototype faster
- Modularity, better code reuse
- Nicer architecture?
- You can unit test properly
  - Mocking is amazingly easy
- There is no DI framework in Python (ZOPE?)

Most serious outcry: Where is some fine IDE?

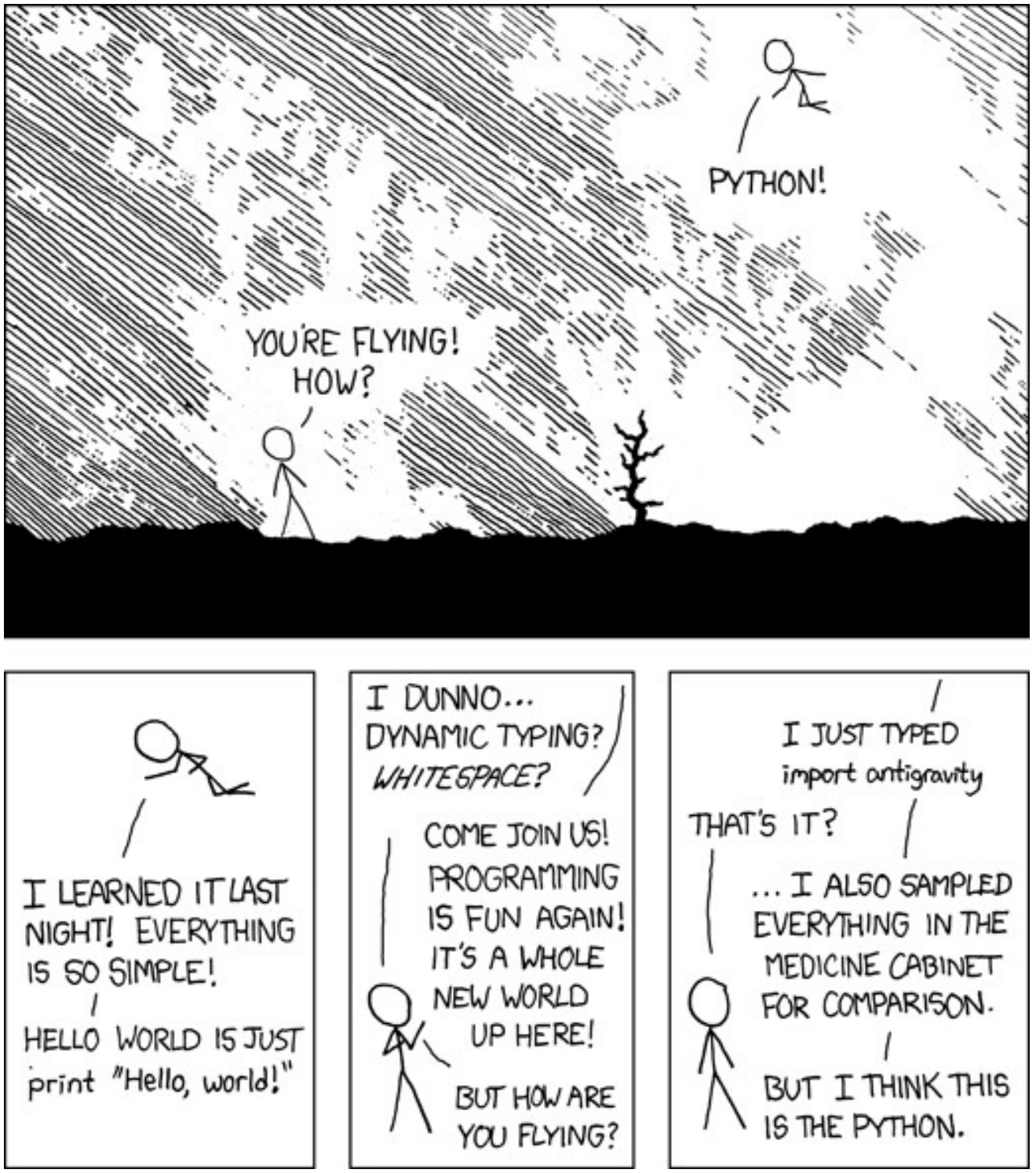
- You may not like Java, but Eclipse is great, IDEA is even better (?)
- Eclipse is great **for Java** but plugins for other languages are quite poor
- VIM is a great editor, but it is not fine IDE

What do we need from fine IDE

- Highlight syntax (done)
- Show as much errors and warnings during typing the code as possible
- Code completion
- Easier refactoring
- Quick navigation in the project (show definition of variable, function, object, class)
- Management of user's project (needed?)
- Managing running configurations (needed?)
- Support for testing (definitly not needed)
- Quality of code meassures (Pylint)

Pylint, Pyflakes

Can we do better?


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

How can we infer the type?

- static-type approach: ignore most of the dynamic features, make some inference rules

- another approach: run the code and see

  (does the object 'o' have the attribute 'www' on the line 42 ? Well let's see)

Can we do something in between?

Symbolic execution

- Do exactly the same as interpreter, but:
  - Instead of variables' values use their types
  - Substitute std library and native code by mock implementation
    - Mock implemntation must have no side-effects and mus return correct type
  - If you are not sure, what would interpreter do, guess

  We are trying to give valuable advice, but we are not proving anything

Typez

`a=1+2` is just `Typez(int)`

```
o=Empty()
o.a=5
o.b='ahoj'

o=Typez(
  'a': Typez(int),
  'b': Typez(str),
  '__class__':Typez(...),
)
```

Problematic?

EASY	DOABLE	HARD
function, classes	if	Inversion of controll
closures	try/catch	eval, exec?
inheritance		
in-time semi-accuracy		
builtins, std. library		
for, while		
list comprehension		

for, while

- Ostrich algorithm

builtins

```
class Num_(inf_object):

    def __add__(self,x):
        return 0

    def __div__(self,x):
        return 0

    def __mul__(self,x):
        return 0
```

•

try/catch

- trivialy reduceable to if

if

- Former strategy: if-else-merge
- New strategy: shoot and relax

collections

```
class list():
    def __init__(self):
        self.val=None

    def append(self, item):
         if undecidable:
             self.val=item

    def __setitem__(self, attr, item):
        if undecidable:
            self.val=item

    def __getitem__(self, attr):
        return self.val
```

Future work

Thank you for your attention

If Python is so great, why it is used so infrequently?

- "*There is a lot of friction in the system*" (Andy Tanenbaum)
- Speed (very important point for e.g. Google)
- Can I trust BDFL? Can I believe they are not going to screw it up?
- Great with POSIX systems, sharp edges on Windows