Advanced Topics in Programming Language - Duck Typing in Python

Tommaso Puccetti
Studente presso Universita degli studi di Firenze

April 16, 2019

Contents

List of Tables

List of Figures

1 Possible references

- You Tube video;
- Python duck typing (or automatic interfaces)- how change the depen-decy injection;
- Simple example;
- Something more technical;
- Ultimate guide to Python's type checking;
- Static vs Dynamic (Stack Overflow)
- Why Python is dynamic and strongly typed
- Maybe schematic overview on typechecking

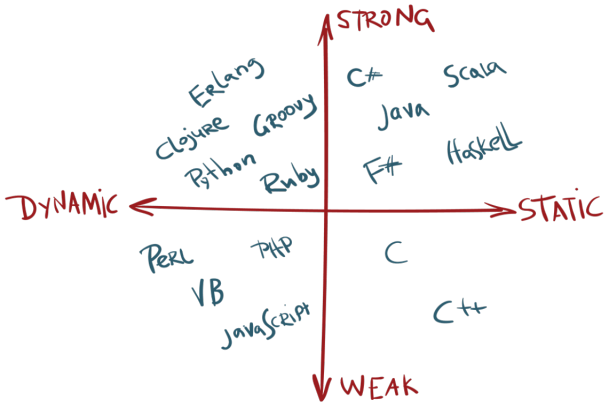
2 Questions

- Are language that implement type inference always static ?

3 Type checking: classification

Type checking is the process of verifying and enforces the typing rules of a language. In other words the **type checker** (the type checking algorithm of the language) is used to prove the **type safety** of a program.
Here the possible categories:

- Dynamic vs. Static;
- Explicit vs Implicit;
- Weakly vs Strongly

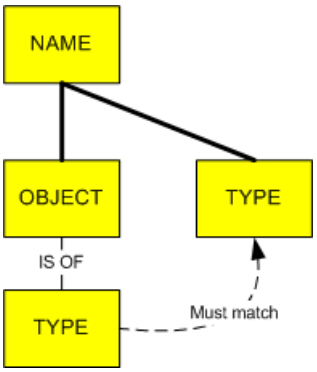


INSERIRE VANTAGGI E SVANTAGGI DA OVERVIEW SCHEMATICA PER OGNUNA DELLE SUBSUBSECTION

3.1 Static type checking

Is the process of verifying the type safety of a program based on the analysis of a program text. If a program passes a static type checker, then the program is guaranteed to satisfy some set of type safety properties for all possible inputs ***(Wikipedia)***.
A language is statically typed if the type of a variable is known at compile time. For some languages this means that you as the programmer must specify what type each variable is (e.g.: Java, C, C++); other languages offer some form of type inference, the capability of the type system to deduce the type of a variable ***(Stack Overflow)***;

A language is statically-typed if the type of a variable is known at compile-time instead of at run-time. Common examples of statically-typed languages include Java, C, C++, FORTRAN, Pascal and Scala. ***(Schematic overview)***;



3.1.1 Example

Here a java example:

```
int variable;
variable = 10;
variable = "ten";
```

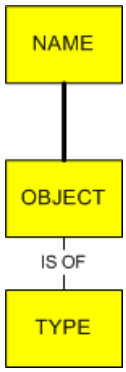
It causes a compilation error:

```
INSERIRE ERRORE DI COMPILAZIONE
```

3.2 Dynamic type checking

Is the process of verifying the type safety of a program at runtime. It may cause a program to fail at runtime ***(Wikipedia)***.
A language is dynamically typed if the type is associated with run-time values, and not named variables/fields/etc. This means that you as a programmer can write a little quicker because you do not have to specify types every time (unless using a statically-typed language with type inference) ***(Stack***

Overflow).
In Dynamically typed languages, variables are bound to objects at run-time by means of assignment statements, and it is possible to bind the same variables to objects of different types during the execution of the program. Dynamic type checking typically results in less optimized code than static type checking. It also includes the possibility of run time type errors and forces run time checks to occur for every execution of the program (instead of just at compile-time) (***Schematic overview***).



3.2.1 Example

```
variable = 10
variable = "ten"
```

CAPIRE SE SONO SOTTO CATEGORIE DI STATIC O DISTINZIONE PIU GENERALE

3.3 Explicitly typed

Each variables is annotaded in source code with type's information. In this case the *type check is simple but the language is more difficult* (from the programmers point of view).

3.4 Implicitly typed (inference)

The data types of source code are automatically detected. It is also rederred as **type inference**. The language *is easier but the type check algorithm is far more complex.*

3.5 Strongly typed

A strongly-typed language is one in which variables are bound to specific data types, and will result in type errors if types do not match up as expected in the expression regardless of when type checking occurs. (***Schematic overview***)
EXAMPLEEEEEEEEE

3.6 Weakly typed

A weakly-typed language on the other hand is a language in which variables are not bound to a specific data type; they still have a type, but type safety constraints are lower compared to strongly-typed languages.
EXAMPLEEEEEEEEE

3.7 Example: type inference vs. dynamic typing

These two kind of typings could be confused. Here an example to clarify the differences:

```
var1 = 10
var2 = "astring"
var3 = var1 + var2
```

1. In **dynamically typed** language this code run without errors: at runtime the *var1* is forced to be a string and the result is *"10astring"*;
2. By the other side, in **inferred type language** the compiler *throw an error.*