Type systems

The INFDEV team

Hogeschool Rotterdam
Rotterdam, Netherlands

Introduction

Introduction

Lecture topics

- Issues with Python
- Issues with Python and possible solutions
- Static typing

Issues with Python

Issues with Python

Lack of...

- Lack of constraints: how can we specify that a function only takes integers as input
- Lack of structure: how can we specify that a variable will certainly support some methods
- Lack of assurances: how can we guarantee that programs with evident errors are not run

Issues with Python

What is wrong with this?

```
1 def f(x):
2   return (x * 2)
3 f("nonsense")
```

Issues with Python

What is wrong with this?

```
1 def f(x):
2   return (x * 2)
3 f("nonsense")
```

The function clearly works with integers, but is given a string

Issues with Python

What is wrong with this?

```
1 x = input()
2 if (x > 100):
3   print("dumb")
4 else:
5   print("dumber")
```

Issues with Python

What is wrong with this?

```
1 x = input()
2 if (x > 100):
3   print("dumb")
4 else:
5   print("dumber")
```

The comparison is nonsensical if `x` is not a number

Issues with Python

What is wrong with this?

```
1 def g(car):
2   return car.drive(2)
3 g(-1)
```

Issues with Python

What is wrong with this?

```
1 def g(car):
2   return car.drive(2)
3 g(-1)
```

We expect something with a `drive` method, but get an integer instead

Possible solutions

Possible solutions

Testing?

- Testing the program should be enough

Possible solutions

Testing?

- Testing the program should be enough
- Right?

Possible solutions

Testing?

- Testing the program should be enough
- Right?
- No. The number of possible execution paths is immense (order of billions), and each test only takes one.
- Testing can only guarantee presence of bugs, but not their absence!

Possible solutions

How many times would we need to test to be sure there is no error?

```
1 if (randint(0,100000) > 99999):
2   g(-1)
3 else:
4   g(mercedesSL500)
```

Possible solutions

How many times would we need to test to be sure there is no error?

```
1 if (randint(0,100000) > 99999):
2   g(-1)
3 else:
4   g(mercedesSL500)
```

¿100000

Possible solutions

Testing?

- We want our programming languages to perform checks for us
- Clearly nonsensical programs should be rejected before we can even run them
- It is safer and easier to spend more time "talking" with the IDE than hoping to find all errors at runtime

Static typing

Static typing

Introduction

- The language verifies[a], before running code, that all variables are correctly used
- "Correctly used" means that they are guaranteed to support all operations used on them
- This is by far and large the most typical solution to increase safety and productivity

[a] By means of the **compiler**.

Static typing

What is static typing?

- When declaring a variable, we also specify what sort of data it will contain
- The **sort** of data contained is called **TYPE** of the variable
- Types can be either primitives (int, string, etc.), custom (classes), or compositions (functions, list of elements of a given type, etc.)

Static typing

What is static typing?

- Especially in mainstream languages, the specification of the type of a variable is done by hand by the programmer
- In other languages (mostly functional languages like F#, Haskell, etc.) the type of variables is automatically guessed by the compiler
- In our case our programs will become a bit more verbose but better specified
- Still, static typing is not necessarily connected with verbosity

Static typing

```
1 def f(x):
2   return (x * 2)
```

Becomes, typed:

What has improved and why?

Static typing

```
1 def f(x):
2   return (x * 2)
```

Becomes, typed:

What has improved and why?

The second definition encodes information about what goes in and what comes out of the function

Static typing

Is this still possible to write (as it was in Python)?

Static typing

Is this still possible to write (as it was in Python)?

No: we get a compiler error because a string cannot be used where a number is expected

Static typing

```
1 x = input()
2 if (x > 100):
3   print("dumb")
4 else:
5   print("dumber")
```

Becomes, typed:

What has improved and why?

Static typing

```
1 x = input()
2 if (x > 100):
3   print("dumb")
4 else:
5   print("dumber")
```

Becomes, typed:

What has improved and why?

The variable declaration specifies what is allowed (and what is not) inside the variable.

Static typing

```
1 def g(car):
2   return car.drive(2)
3 g(-1)
```

Becomes, typed:

What has improved and why?

Static typing

```
1 def g(car):
2   return car.drive(2)
3 g(-1)
```

Becomes, typed:

What has improved and why?

The function declaration specifies available methods of extttcar. We will thus get a compiler error.

Typing rules and semantic rules

Typing rules and semantic rules

How do we describe them?

- How do we describe such relations clearly?
- We use the so-called **typing rules**, which specify what may be done and what not
- Typing rules are quite intuitive: they state that if one or more premises are true, then the conclusion is true as well

Typing rules and semantic rules

$$\frac{A \wedge B}{C}$$

If A and B are true, then we can conclude C

Typing rules and semantic rules

$$\frac{\text{I wish to buy a pretty car} \wedge \text{ I have 120000 euros}}{\text{I buy a Mercedes SL500}}$$

How do we read this rule?

Typing rules and semantic rules

$$\frac{\text{I wish to buy a pretty car} \wedge \text{ I have 120000 euros}}{\text{I buy a Mercedes SL500}}$$

How do we read this rule?

If I have 120000 euros and I wish to buy a pretty car, then I buy a Mercedes SL500

Typing rules and semantic rules

$$\frac{\text{I have my umbrella with me} \wedge \text{ It is raining}}{\text{I open my umbrella}}$$

How do we read this rule?

Typing rules and semantic rules

$$\frac{\text{I have my umbrella with me} \wedge \text{ It is raining}}{\text{I open my umbrella}}$$

How do we read this rule?

If I have my umbrella with me, and it is raining, then I open my umbrella

Typing rules and semantic rules

Reading typing rules

Let us apply this machinery to programming languages

Typing rules and semantic rules

Reading typing rules

- Let us apply this machinery to programming languages
- We will effectively give the specification of a modern compiler
- This looks like a "broadly scoped" execution of the program, and it is indeed such
- Instead of having a coupling of the variables with values in the stack and the heap, we maintain a coupling of the variables with their declared type

Typing rules and semantic rules

- We want to specify this in the typing rule notation
- The typing rules manipulate a stack of declarations
- x : int

$$\frac{\texttt{c} : \texttt{Boolean} \wedge\ \texttt{a} : \texttt{void} \wedge\ \texttt{b} : \texttt{void}}{\texttt{if c then a else b : void}}$$

If a part of a program does not have a type derived through the typing rules (also void is fine), then the whole program cannot be run and we get a compiler error

This is it!

The best of luck, and thanks for the attention!