Premis 00 Typing basics

Our typi

Our typical stack

nnotations and ORM

Enforcing the contrac

◆□ > ◆□ > ◆豆 > ◆豆 > ̄豆 = のへで

Summary 000

# Full Stack Type Safety

Szymon Pyżalski

Egnyte Inc.

Europython 2020

Premise 00 basics

Our typical stack

nnotations and ORM

Enforcing the contra

Summary 000



#### Premise

Typing basics

Our typical stack

Annotations and ORM

Enforcing the contract

Summary

▲□▶ ▲□▶ ▲ 三▶ ▲ 三▶ - 三 - のへで

Premise •0 Typing basics

Our typical stack

Annotations and ORM

Enforcing the contract 000

Summary 000







• Catch typing errors ASAP (not later than in CI)





▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @

- Catch typing errors ASAP (not later than in CI)
- Catch typing errors that span layers of stack

Premise	-
0.	

Typing basics C

Our typical stack

Annotations and ORM

Enforcing the contract 000

Summary 000







• Type annotation system in Python is new and immature





- Type annotation system in Python is new and immature
- Various layers of stack feature different typing paradigms

▲ロト ▲周ト ▲ヨト ▲ヨト ヨー のくで



#### Problems

- Type annotation system in Python is new and immature
- Various layers of stack feature different typing paradigms

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @

• We tend to test layers in separation

Pr	rem	is	e .
0	С		

Our typical stack

Annotations and ORM

Enforcing the contrac

Summary 000

#### Weak vs strong





# Weak typing A value can be misinterpreted unless we care about the type by ourselves.

▲ロト ▲周ト ▲ヨト ▲ヨト ヨー のくで



#### Weak vs strong

Weak typing A value can be misinterpreted unless we care about the type by ourselves. Strong typing We are protected from misinterpretations by the type system.

▲ロト ▲周ト ▲ヨト ▲ヨト ヨー のくで



```
Weak typing
```

```
#include < stdio.h>
short int fun(int* x) {
        short int y = *(short int*)x;
        return y + 1;
}
int main(int argc, char** argv) {
        int a = -10:
        int b = 777777;
        printf("%u\n", a); // prints: 4294967286
        printf("%d\n", fun(&b)); // prints: -8654
}
```

Premise 00 Typing basics

Our typical stack

Annotations and ORM 000 Enforcing the contrac

◆□ > ◆□ > ◆豆 > ◆豆 > ̄豆 = のへで

Summary 000

#### Static vs dynamic



# Static typing The types of objects can be determined during compile time.

▲ロト ▲周ト ▲ヨト ▲ヨト ヨー のくで



Enforcing the contrac

▲ロ▶ ▲周▶ ▲ヨ▶ ▲ヨ▶ ヨ のなべ

Summary 000

#### Static vs dynamic

#### Static typing The types of objects can be determined during compile time. Dynamic typing The types of objects are determined during runtime.



#### Dynamic typing in python

```
def sum(xs, init):
    result = init
    for x in xs:
        result += x
    return result
```

```
print(sum([1, 2, 3], 0)) # prints 6
print(sum({'a': 'b', 'c': 'd'}, 'Keys: ')) # prints: Keys: ac
```

▲ロ▶ ▲周▶ ▲ヨ▶ ▲ヨ▶ ヨ のなべ

Typing basics 

▲ロ▶ ▲周▶ ▲ヨ▶ ▲ヨ▶ ヨ のなべ

### Static typing with inference

```
package main
import "fmt";
func fact(n int) int {
        result := 1
        for i := 1; i <=n; i++ {</pre>
                result *= i
        }
        return result
}
func main() {
        x := 10
        y := fact(5)
        fmt.Println(x)
        fmt.Println(y)
```

}

Premise	Ту
00	00

ping basics 0

Our typical stack

Annotations and ORM 000 Enforcing the contract 000

Summary 000

#### Strict vs loose

▲□▶ ▲□▶ ▲ 三▶ ▲ 三▶ 三三 - のへぐ



# Strict typing Type conversions must be explicit. Type mismatch exceptions.

◆□ > ◆□ > ◆豆 > ◆豆 > ̄豆 = のへで



#### Strict vs loose

# Strict typing Type conversions must be explicit. Type mismatch exceptions.

▲ロ▶ ▲周▶ ▲ヨ▶ ▲ヨ▶ ヨ のなべ

Loose typing Type conversions can be implicit.

```
Stricter than Python
```

import Data.String.Utils (join)

```
list2Str :: [[Char]] -> [Char]
list2Str xs = if xs then "No elements" else (join "," xs)
```

▲ロ▶ ▲周▶ ▲ヨ▶ ▲ヨ▶ ヨ のなべ

main = do
 putStrLn \$ list2Str []
 putStrLn 10 -- Error

Typing basics

00000000000000

#### Looser than Python

1 + 'a'  $\{\} + 2$ 'abc' + ['d', 'e', 'f'] // "abcd, e, f"  $\{\} + 'z'$  $\{\} + \{\}$  $\{\} + []$  $[] + {}$ 

// '1a' 110 // NaN // NaN 110 // "[object Object]"

▲ロ▶ ▲周▶ ▲ヨ▶ ▲ヨ▶ ヨ のなべ

Premise	Турі
00	000

ng basics Ou

Our typical stack

Annotations and ORN 000 Enforcing the contract 000

▲□▶ ▲□▶ ▲ 三▶ ▲ 三▶ 三三 - のへぐ

Summary 000

#### Duck vs ???



Duck typing Interfaces Protocols are implemented implicitly. Object is compatible with a protocol if it implements required methods.

▲ロ▶ ▲周▶ ▲ヨ▶ ▲ヨ▶ ヨ のなべ



#### Duck vs ???

Duck typing Interfaces Protocols are implemented implicitly. Object is compatible with a protocol if it implements required methods.

??? Classes must inherit from a class in order to be compaticle, or at least be marked as implementing the protocol.

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @



Our typical stack

Annotations and ORN 000 Enforcing the contrac

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @

Summary 000

#### Duck vs platonic

Duck typing Interfaces Protocols are implemented implicitly. Object is compatible with a protocol if it implements required methods.

Platonic typing Classes must inherit from a class in order to be compaticle, or at lease be marked as implementing the protocol. Typing basics

Our typical stack

Annotations and ORM

Enforcing the contrac

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @

Summary 000

## Structural vs nominal

Structural typing Interfaces Protocols are implemented implicitly. Object is compatible with a protocol if it implements required methods.

Nominal typing Classes must inherit from a class in order to be compaticle, or at lease be marked as implementing the protocol.



They say	We say





▲□▶ ▲□▶ ▲ □▶ ▲ □▶ □ のへぐ

They say	We say
throw	raise



▲□▶ ▲□▶ ▲ □▶ ▲ □▶ □ のへぐ

They say	We say
throw	raise
array	list



They say	We say
throw	raise
array	list
list	deque





◆□ > ◆□ > ◆豆 > ◆豆 > ̄豆 = のへで

They say	We say
throw	raise
array	list
list	deque
blatant abuse of exceptions	StopIteration

Typing basics 0000000000000000

◆□ > ◆□ > ◆豆 > ◆豆 > ̄豆 = のへで

# The pythonish language

They say	We say
throw	raise
array	list
list	deque
blatant abuse of exceptions	StopIteration
interfaces	protocols

Typing basics 0000000000000000

◆□ > ◆□ > ◆豆 > ◆豆 > ̄豆 = のへで

# The pythonish language

They say	We say
throw	raise
array	list
list	deque
blatant abuse of exceptions	StopIteration
interfaces	protocols

Premise 00 Typing basics

Our typical stack

Annotations and ORM 200 Enforcing the contrac

イロト 不得 トイヨト イヨト

3

Summary 000

#### Static but duck-typed

```
package main
import "fmt"
type Duck interface {
        swim(x int, y int)
        quack() string
}
type Mallard struct {
        x, y int
}
func (m *Mallard) swim(x, y int) {
        m.x += x
        m.v += v
}
func (m Mallard) quack() string {
        return "Quack quaaaack"
}
func swimThenQuack(d Duck) {
        d.swim(1, 1)
        fmt.Println(d.quack())
3
func main() {
        donald := Mallard{x: 0, y: 0}
        swimThenQuack(&donald)
        fmt.Println(donald)
}
```



# Typing models

▲ロト ▲周ト ▲ヨト ▲ヨト ヨー のくで

- Strong vs weak typing
- Static vs dynamic typing
- Strict vs loose typing
- Structural vs nominal typing



# Typing models

▲ロト ▲周ト ▲ヨト ▲ヨト ヨー のくで

- Strong vs weak typing
- Static vs dynamic typing
- Strict vs loose typing
- Structural vs nominal typing
- Free vs fixed attributes

Premise 00 Typing basics 00000000000000 Our typical stack

Annotations and ORN 000 Enforcing the contrac

ヘロト 人間 とくほとくほとう

æ

Summary 000

# Our typical stack

Γ	Javascript
L	Strong
	Very loose
	Dynamic
	Structural
	Free attributes
Г	Python
	Strong
	Strict
	Dynamic
	Structural
	Free attributes
Γ	SQL
	Weak (foreign keys)
	Loose
	Static
	Nominal
	Fixed attributes

Premise 00 Typing basics

Our typical stack

Annotations and ORN 000 Enforcing the contrac

▲□▶ ▲□▶ ▲ □▶ ▲ □▶ □ のへぐ

Summary 000

# Our typical stack

Javascript		
Strong		
Very	loose	
Dynamic		
Structural		
Free attributes		
Python Models		
Strong	Strong	
Strict	Strict	
Dynamic	Static	
Structural	Nominal	
Free attributes	Fixed attributes	
SQL		
Weak (foreign keys)		
Loose		
Static		
Nominal		
Fixed attributes		

 Premise
 Typing basics
 Our typical stack
 Annotations and ORM
 Enforcin

 00
 000000000000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000
 000

Enforcing the contract 000

▲ロト ▲周ト ▲ヨト ▲ヨト ヨー のくで

Summary 000

## Weakness of SQL foreign keys

```
UPDATE books set author_id = (
    SELECT id FROM publishers
    WHERE name="Chilton Books"
);
```



# ORM improving type safety

- b = Book.objects.get(id=1)
- b.author = Publisher.objects.get(name='Chilton Books')

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @

Premise

g basics 0000000000 Our typical stack

Annotations and ORM • 00

Enforcing the contra

▲□▶ ▲□▶ ▲ □▶ ▲ □▶ □ のへぐ

Summary 000

### mypy enters the game

Javascript			
Strong			
Very loose			
Dynamic			
Structural			
F	ree attributes		
туру	Python	Models	
Strong	Strong	Strong	
String	Strict	Strict	
Static	Dynamic	Static	
Preference for nominal	Structural	Nominal	
Fixed attributes	Free attributes	Fixed attributes	
SQL			
Weak (foreign keys)			
Loose			
Static			
Nominal			
Fixed attributes			



#### Django and mypy working together



Premise 00 basics DOOOOOOOOO

Our typical stack

Annotations and ORM  $\circ \circ \bullet$ 

Enforcing the contra

Summary 000

# mypy and Django pros and cons

◆□ > ◆□ > ◆豆 > ◆豆 > ̄豆 = のへで



▲ロト ▲周ト ▲ヨト ▲ヨト ヨー のくで

# mypy and Django pros and cons

• Pro: Recognizes the relationship between column types and python types



# mypy and Django pros and cons

• Pro: Recognizes the relationship between column types and python types

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @

• Pro: Recognizes the idea of null



#### mypy and Django pros and cons

• Pro: Recognizes the relationship between column types and python types

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @

- Pro: Recognizes the idea of null
- Con: Can't handle problems with incomplete data



### mypy and Django pros and cons

• Pro: Recognizes the relationship between column types and python types

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @

- Pro: Recognizes the idea of null
- Con: Can't handle problems with incomplete data
- Con: Requires a mypy plugin

Premise 00 ing basics

Our typical stack

Annotations and ORN 000 Enforcing the contract  $\bullet \circ \circ$ 

◆□ > ◆□ > ◆豆 > ◆豆 > ̄豆 = のへで

Summary 000

# Considering the JSON

	Javascript				
	Very loose				
Dynamic					
Structural Free attributes			JSON		
				туру	Python
Strong	Strong	Strong			
String	Strict	Strict			
Static Dynamic		Static			
Preference for nominal	Structural	Nominal			
Fixed attributes	Free attributes	Fixed attributes			
	SQL				
	Weak (	foreign keys)			
Loose					
Static					
Nominal					
Fixed attributes					

Premis 00

ise Typing basics

asics 01

Our typical stack

Annotations and ORM 000 Enforcing the contract  $\circ \bullet \circ$ 

Summary 000

### One solution

	Typescript			
Strong				
Strict				
	Static			
Structural				
F	ixed attributes		JSON	OpenAPI3
туру	Python	Models	No typing above primitives	Schema
Strong	Strong	Strong		Tests
String	Strict	Strict		Code generation
Static	Dynamic	Static		
Preference for nominal Structural Nominal		Nominal		
Fixed attributes	Free attributes	Fixed attributes		
		SQL		
		Weak (foreign ke	ys)	
Loose				
Static				
Nominal				
Fixed attributes				

◆□ ▶ ◆□ ▶ ◆ 臣 ▶ ◆ 臣 ▶ ○ 臣 ○ のへで



Enforcing the contract



Premise 00	Typing basics	Our typical stack	Annotations and ORM	Enforcing the contract	Summary ●00	
Takeaways						

◆□▶ ◆□▶ ◆臣▶ ◆臣▶ 臣 のへぐ



• There are tools for code safety enforcement in a Python stack that are worth consideration

◆□ > ◆□ > ◆豆 > ◆豆 > ̄豆 = のへで



• There are tools for code safety enforcement in a Python stack that are worth consideration

▲ロト ▲周ト ▲ヨト ▲ヨト ヨー のくで

They are not yet perfect and we can't expect to catch all errors

Premise 00 Typing basics 00000000000000

Our typical stack

Annotations and ORN 000 Enforcing the contrac

Summary

#### Future can bring

▲□▶ ▲□▶ ▲ 三▶ ▲ 三▶ 三三 - のへぐ



#### • Support for more patterns in type annotations without plugins

◆□ > ◆□ > ◆豆 > ◆豆 > ̄豆 = のへで



#### Future can bring

000

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @

- Support for more patterns in type annotations without plugins
- Tools based on code annotations instead of descriptors ( strawberry-graphql, pydantic, )



#### Tools used

- django-stubs A distribution of code annotations for django complete with a mypy plugin
- spectacular A schema generator for django-rest-framework
- openapi-generator Code generator that can create boilerplate code for several languages/frameworks based on OpenAPI3.

▲□▶ ▲□▶ ▲□▶ ▲□▶ ▲□ ● ● ●