PYTHON, TIPOS & PESSOAS

felipe@felipevr.com — github.com/fbidu

Speaker notes

- Bom dia a todos, bem vindos! É muito bom estar aqui mais uma vez junto da comunidade python. A última conferência que eu estive foi na Python Sudeste e eu não consegui ir ao Caipyra desse ano. Antes de começar eu queria saber quem aqui tá na primeira python brasil? Muito bom, sejam bem vindos! Uma das coisas mais legais e mais importantes de Python é isso, a nossa comunidade. Eu espero que vocês tenham uma ótima experiência na Pybr e voltem sempre! Agora, quem aqui já usou de Type hinting em Python?

- Hoje nós vamos discutir um pouco sobre tipos de dados, análise estática de código e as funcionalidades que temos para lidar com esses conceitos em Python

- O título original dessa palestra era Python, Tipos e Segurança. Eu decidi mudar esse título por razões que vão ficar mais claras ao final

Aperte a tecla "s" para ver as notas de apresentação!

OI, EU SOU O (FELIPE) BIDU!

- github.com/fbidu
- Instituto de Computação - UNICAMP
- Desenvolvedor Back End
- Arquiteto de Infra no Poppin

O QUE É UM TIPO?

Speaker notes

Para iniciar nossa discussão, vamos primeiro definir o que *é* um tipo

01100001

01100001

- Qual o significado dessa cadeia binária?

01100001

- Qual o significado dessa cadeia binária?
- Em decimal, é o número 97

01100001

- Qual o significado dessa cadeia binária?
- Em decimal, é o número 97
- Na tabela ASCII, é o caractere "a"

Speaker notes

- Por baixo dos panos, computadores processam bits, zeros e uns.
- Eventualmente, todo pedaço de informação é convertido para bits
- Imagine que seu computador encontre esses bits enquanto processa dados
- Como que essa sequência pode ser interpretada?
- *abrir resto do slide*
- O fato de que uma informação pode ser vista de duas formas diferentes gera uma possível ambiguidade
- Computadores não lidam muito bem com ambiguidades!

OPERAÇÕES

- De acordo com *como* entendemos uma informação, as operações possíveis mudam!

Speaker notes

- Se estamos lidando com strings ou inteiros, as coisas que nós podemos fazer com a informação é diferente

```
>>> 97 * 10
970

>>> 97 - 55
42
```

01100001 == 97?

Speaker notes

- Se esses bits representarem um inteiro, nós podemos fazer operações algébricas com ele
- Podemos usar essa informação como índice de uma lista, como elemento de comparação maior-menor com outros números, etc

```
>>> "a" * 10
'aaaaaaaaaa'

>>> "a".upper()
'A'

>>> "a".isalpha()
True

>>> "a".isdigit()
False
```

01100001 == "a"?

Speaker notes

- Por outro lado se essa cadeia de bits for um caractere, as operações possíveis são bem diferentes
- Nós agora podemos usar todas as operações de string, podemos realizar comparações com outras strings
- Mas nós não podemos mais usar essa informação como um índice de uma lista, por exemplo

Tipos permitem definir quais operações podemos ou não aplicar sobre um conjunto de informações!

```
>>> 80218 + "Python"

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Speaker notes

- Um Type Error ocorre quando violamos as operações que podem ser aplicadas sobre um tipo
- O que faz com que uma operação cause um type error depende da linguagem

Python:

```
>>> "Python Brasil " * 2
'Python Brasil Python Brasil '
```

Java:

```
String a = "Python Brasil ";

a * 2;

----------
|  Error:
|  bad operand types for binary operator '*'
|    first type:  java.lang.String
|    second type: int
|  a * 2
|  ^---^
```

Speaker notes

- Python, por exemplo, permite que a gente multiplique uma string por um número a operação é considerada válida e não causa problemas
- Em Java isso já não é possível. Aplicar o operador * entre strings e inteiros causam um type error

> *"[...] A type is a set of values and a set of functions that one can apply to these values."*

— Guido van Rossum, Ivan Levkivskyi - PEP 483 - The Theory of Type Hints

Speaker notes

- Existem alguns PEPs que são importantes para a discussão de hoje.
- O PEP 484 define os Type Hints que iremos tratar em breve e o PEP 483 lida com a teoria por trás disso tudo
- No PEP 483 os autores usam essa definição "um tipo é um conjunto de valores e um conjunto de funções que alguém pode aplicar nesses valores"
- Nesse mesmo PEP, os autores explicam que existem *diversas* definições de tipos na literatura de teoria da computação. De fato, é uma área grande e com bastante trabalho feito.
- No entanto, na palestra de hoje, eu vou lidar com a parte mais instrumental e mais prática de tipos e tipagem no Python

SISTEMAS DE TIPOS

- Diversos nomes diferentes

SISTEMAS DE TIPOS

- Diversos nomes diferentes
- Estático vs. Dinâmico

SISTEMAS DE TIPOS

- Diversos nomes diferentes
- Estático vs. Dinâmico
- Forte vs. Fraco

SISTEMAS DE TIPOS

- Diversos nomes diferentes
- Estático vs. Dinâmico
- Forte vs. Fraco
- Explícito vs. Implícito

SISTEMAS DE TIPOS

- Diversos nomes diferentes
- Estático vs. Dinâmico
- Forte vs. Fraco
- Explícito vs. Implícito
- (autor A) vs. (autor B)

felipe@felipevr.com — github.com/fbidu

Speaker notes

- Em termos de sistemas de tipos e como as linguagens de programação se comportam existe uma coleção considerável de nomenclaturas
- *exibir todo slide*

ESTÁTICO VS. DINÂMICO

O tipo de uma variável pode mudar?

felipe@felipevr.com — github.com/fbidu

Speaker notes

- Estático: Variáveis mantém o mesmo tipo durante toda a vida.
- Estático: A checagem de tipos é feita normalmente antes da execução
- Em linhas gerais, uma linguagem com tipagem estática não permite que o tipo de uma variável mude

FORTE VS. FRACO

Quão "chata" a linguagem é com tipos?

felipe@felipevr.com — github.com/fbidu

Speaker notes

- Forte: diversas definições! Muitas delas são informais
- Quanto mais uma linguagem garante que operações só possam ser aplicadas para certos tipos,mais "forte" ela é
- Ponteiros como em C, que apontam para regiões arbitrárias da memória "enfraquecem" o sistema de tipos

EXPLÍCITO VS. IMPLÍCITO

Você consegue saber o tipo de uma variável só lendo o código?

Speaker notes

- Numa linguagem com tipagem explícita, existem palavras reservadas obrigatórias que declaram o tipo. C, Java e Pascal são assim
- Já na tipagem implícita, o tipo de uma variável é inferido do contexto

Python tem um sistema de tipos **implícito**:

```
>>> ano = 2019
```

Python tem um sistema de tipos implícito, **dinâmico**:

```
>>> ano = 2019
>>> ano = str(ano)
```

Python tem um sistema de tipos implícito, dinâmico e **forte**:

```
>>> ano = 2019
>>> ano = str(ano)
>>> ano / 10

TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

ANÁLISE ESTÁTICA & TIPAGEM DINÂMICA

Juntando dois pedaços da conversa de hoje

felipe@felipevr.com — github.com/fbidu

Speaker notes

- Definido o sistema de tipos do Python em linhas gerais, podemos agora começar a considerar outra coisa importante na conversa de hoje, análise estática
- Olhando para os termos "estática" e "dinâmica" em cada lado desse título, dá pra sentir que teremos alguns problemas

PODEMOS CHECAR CÓDIGO DE DUAS FORMAS

PODEMOS CHECAR CÓDIGO DE DUAS FORMAS

1. Lendo o código

PODEMOS CHECAR CÓDIGO DE DUAS FORMAS

1. Lendo o código
2. Rodando o código

Speaker notes

- Em linhas gerais, nós podemos testar e validar coisas no nosso código de duas formas, lendo o código e acompanhando o que acontece e executando o código

Método de Análise	Tipo
Leitura	Estática
Execução	Dinâmica

Speaker notes

- Quando uma ferramenta - ou você! - tira conclusões do código sem executá-lo, temos uma análise estática
- Por outro lado, quando o código é de fato executado, a análise é dinâmica
- As duas formas de análise tem seus méritos e seus usos. Hoje vou focar nas vantagens da análise estática

ANÁLISE ESTÁTICA

ANÁLISE ESTÁTICA

1. Livre de efeitos do código

ANÁLISE ESTÁTICA

1. Livre de efeitos do código
2. Mais rápida de definir e executar

ANÁLISE ESTÁTICA

1. Livre de efeitos do código
2. Mais rápida de definir e executar
3. A cobertura é sempre 100%

Speaker notes

- A primeira vantagem da análise estática é que ela e livre de qualquer efeito que o código tem. Como o código não será executado, você não precisa se preocupar com o código apagar um arquivo, escrever num banco de dados, lançar um míssil, etc
- Uma análise estática costuma ser rápida para ser definida - basta você escolher sua ferramenta favorita de análise e configurar algumas coisinhas. Ela também é mais rápida de ser executada que a dinâmica. Uma análise dinâmica em geral depende de testes escritos por humanos que flexionam as partes do sistema para serem testadas
- No momento que um código existe, ele pode ser testado pelo seu analisador estático! Sem você ter que se preocupar com tocar no código através de testes ou coisa assim

⚠️⚠️ ALERTA ⚠️⚠️

Análise Dinâmica é MUITO importante, ela só não é o foco de hoje!

Speaker notes

- Só um breve comentário - eu estou focando em análise estática por uma escolha didática e por conta do tempo que a gente tem. Análise dinâmica é também muito importante!

Análise Estática é muito boa, muito legal, mas...

Speaker notes

- Voltando para análise estática, ela realmente é uma coisa muito boa, mas...

...ela depende do quanto a linguagem sozinha consegue contar pra ela

Speaker notes

- Ela é fundalmentamente limitada pelo que a linguagem consegue contar para o analisador sem de fato rodar o código

Tome esse pedaço de código em Java, por exemplo

```
int some_function(int a, int b) {
    return a.toUpperCase();
}
```

Speaker notes

- A gente nem precisa conhecer java pra estranhar o método "to upper case" sendo aplicado num inteiro. Esse é um nome de um método típico de strings e ali na declaração da função, eu to contando que "a" é um inteiro

Porém em Python...

```
def some_function(a, b):
    return a.upper()
```

Speaker notes

- Nesse caso, nem eu, nem você e nem ninguém tem a menor condição de saber se aquele pedaço de código faz sentido por conta da natureza dinâmica dos tipos em Python
- A ausência de documentação também não ajuda

ALERTA DE FLAME WAR

TIPAGEM DINÂMICA É UM PROBLEMA?

Questão de design intrínseco à linguagem!

Speaker notes

- Esse tipo de pergunta é uma das famosas flame wars - brigas que acontecem normalmente na internet entre pessoas que tem opiniões muito muito fortes sobre assuntos muito muito específicos e normalmente 100% subjetivos
- A tipagem dinâmica oferece uma série de benefícios, como deixar o código mais claro, mais legível e sem ficar parecendo um monte de instruções burocráticas
- A flexibilidade da tipagem dinâmica nos permite criar ferramentas complexas de metaprogramação, graças ao maior nível de introspecção que a linguagem ganha
- Honestamente, cada sistema de tipos tem suas vantagens assim como cada linguagem de programação tem seus casos de uso ótimos e péssimos. Não existe uma linguagem ótima. Perguntar qual linguagem é melhor é uma pergunta fundamentalmente incompleta. Nós precisamos considerar diversos aspectos do contexto antes de respondê-la. Se o seu prazo é curto e a urgência é grande, a melhor linguagem de programação é aquela que você sabe.

JUNTANDO OS DOIS MUNDOS

Como casar o dinamismo de um lado com a riqueza de informações do outro?

felipe@felipevr.com — github.com/fbidu

Speaker notes

- Apesar das vantagens da tipagem dinâmica, a tipagem estática também possui vantagens importantes.
- Num sistema de tipos estáticos, é possível detectar de forma mais rápida erros e bugs que podem ter impacto significativo na estabilidade e segurança do sistema
- Será que tem como juntar um pouco esses dois mundos?

GRADUAL TYPING!

GRADUAL TYPING!

Dinâmico quando desejado, estático quando necessário

GRADUAL TYPING!

Dinâmico quando desejado, estático quando necessário

Jeremy Siek, 2006

GRADUAL TYPING!

Dinâmico quando desejado, estático quando necessário

Jeremy Siek, 2006

Ótima referência para aprofundar na teoria de Gradual Typing

Speaker notes

- Mais ou menos em 2006, Jeremy Siek definiu o conceito de gradual typing, que permite combinar no código elementos dos dois mundos
- Apesar de ter sido definido formalmente em 2006, gradual typing é uma coisa que já acontecia de formas diferentes antes disso

NO MUNDO PYTHON...

NO MUNDO PYTHON...

- Debates sobre isso acontecem desde o ano 2000

NO MUNDO PYTHON...

- Debates sobre isso acontecem desde o ano 2000
- Referência legal: GvR - Type Hints - PyCon 2015

Speaker notes

- Dentro da comunidade Python, debates sobre essa questão remetem ao ano 2000 com uma primeira implementação de alguma coisa próxima de gradual typing vindo no PEP 3107, de 2006
- Eu não vou entrar em detalhes sobre o PEP 3107 porque ele não cumpre de fato com a noção atual de type hinting. Ele na verdade permite que a gente anote funções e seus argumentos com informações arbitrárias, que podem ser usadas como hints

TYPE HINTS

Dicas sobre o tipo

felipe@felipevr.com — github.com/fbidu

Speaker notes

- E vamos finalmente chegar no filé dessa minha palestra, a noção de Type Hint em si
- Type Hint é como implementamos a noção de gradual typing em Python
- Ou seja, é o que permite que a gente marque o tipo quando for conveniente

Qual assinatura de uma função `user_age` que recebe um ID do usuário e retorna sua idade?

Qual assinatura de uma função `user_age` que recebe um ID do usuário e retorna sua idade?

```
user_id = int, user_age = int
```

Qual assinatura de uma função `user_age` que recebe um ID do usuário e retorna sua idade?

```
user_id = int, user_age = int
```

```
user_id = string, user_age = int
```

Qual assinatura de uma função `user_age` que recebe um ID do usuário e retorna sua idade?

```
user_id = int, user_age = int
```

```
user_id = string, user_age = int
```

```
user_id = ObjectID, user_age = string
```

Speaker notes

- Considere o problema de montar uma função que recebe o ID de um usuário e retorna sua idade. Nós podemos pensar em diversas assinaturas para essa função
- ID sendo inteiro, retornando inteiro, ID sendo string retornando inteiro, se vc lida com bancos de dados Mongodb talvez seu ID seja um ObjectID e sei lá porque você faria isso mas vc pode resolver retornar a idade como uma string

Podemos resolver isso com docstrings, documentação, etc. Porém Type Hints oferecem uma forma estruturada e poderosa de lidar com isso!

```
def user_age(user_id: str) -> int: (...)

def some_function(a: int, b: int): -> str (...)
```

TÃO ACABANDO COM O MEU PYTHOOOON

Como assim tipo estático?? E os meus códigos???? Vou ter que mudar tudo de novo? A migração Python 2 - 3 nem acabou ainda!! aaaAAAaaaaa

CAAALMA!

CAAALMA!

- Anotação de tipos em Python é totalmente opcional!

CAAALMA!

- Anotação de tipos em Python é totalmente opcional!
- Código anotado e não anotado pode viver tranquilamente no mesmo projeto

CAAALMA!

- Anotação de tipos em Python é totalmente opcional!
- Código anotado e não anotado pode viver tranquilamente no mesmo projeto
- É possível colocar anotações num arquivo separado (spoiler)

CAAALMA!

- Anotação de tipos em Python é totalmente opcional!
- Código anotado e não anotado pode viver tranquilamente no mesmo projeto
- É possível colocar anotações num arquivo separado (spoiler)
- Não vai quebrar o código de ninguém

CAAALMA!

- Anotação de tipos em Python é totalmente opcional!
- Código anotado e não anotado pode viver tranquilamente no mesmo projeto
- É possível colocar anotações num arquivo separado (spoiler)
- Não vai quebrar o código de ninguém
- E Python não vai deixar de ter tipagem dinâmica!

It should also be emphasized that Python will remain a dynamically typed language, and the authors have no desire to ever make type hints mandatory, even by convention.

— PEP 484 - Type Hints

COMO ANOTAR TIPOS

felipe@felipevr.com — github.com/fbidu

Speaker notes

- Nos slides anteriors vimos um dos exemplos mais simples de anotação de tipos, após o argumento da função, colocamos dois pontos e o tipo
- Porém existem mais formas de fazer essa anotação de tipos

3 FORMAS DIFERENTES:

3 FORMAS DIFERENTES:

1. Function Annotations, como já vimos

3 FORMAS DIFERENTES:

1. Function Annotations, como já vimos
2. Stub Files. Anotações que ficam num arquivo separado

3 FORMAS DIFERENTES:

1. Function Annotations, como já vimos
2. Stub Files. Anotações que ficam num arquivo separado
3. Type Comments. Anotações que ficam em comentários com um formato específico

FUNCTION ANNOTATIONS

Quando a tipagem vive junto dos argumentos, dentro da declaração

felipe@felipevr.com — github.com/fbidu

- Forma mais simples e direta de anotar tipos
- Compatível com o PEP-3107
- Não é compatível com outras versões do Python antes da 3.5
- Se você não tiver necessidade de suportar versões antes da 3.5, comece por aqui!

Speaker notes

- Bem, function annotations são a estrutura que eu mostrei para vocês nos slides anteriores
- Ela se beneficia de uma estrutura montada anteriormente, no PEP 3107. Esse PEP permite que a gente defina anotações genéricas nos argumentos de uma função, mas não define nenhum significado pra essas anotações.
- Com o PEP 484 e os type hints, essas anotações ganham um significado
- Nossos colegas que lidam com teoria de linguagem de programação diriam que o PEP 3107 definiu uma sintaxe, uma estrutura e o PEP 484 definiu uma semântica ou seja, anexou um significado à estrutura

O MÓDULO `typing`

O MÓDULO `typing`

- Define diversas entidades que nos permitem fazer anotações ricas de tipos

O MÓDULO `typing`

- Define diversas entidades que nos permitem fazer anotações ricas de tipos
- Quer definir que uma função retorna uma lista contendo apenas números?

O MÓDULO `typing`

- Define diversas entidades que nos permitem fazer anotações ricas de tipos
- Quer definir que uma função retorna uma lista contendo apenas números?
- Quer especificar que o argumento de uma função deve ser uma outra função que retorna um tipo em especial?

O MÓDULO `typing`

- Define diversas entidades que nos permitem fazer anotações ricas de tipos
- Quer definir que uma função retorna uma lista contendo apenas números?
- Quer especificar que o argumento de uma função deve ser uma outra função que retorna um tipo em especial?
- Quer garantir que o tipo de retorno de uma função é o mesmo tipo de retorno da entrada, mesmo sem definir qual tipo é esse?

O MÓDULO `typing` É A SOLUÇÃO!

- A API do módulo `typing` é rica!
- Recomendo bastante dar uma lida no PEP-484, especialmente na seção sobre ele
- Nos próximos slides vou mostrar alguns exemplos de uso

```python
from typing import TypeVar, Iterable, Tuple

Number = TypeVar('Number', int, float)
Point = Tuple[Number, Number]
Space = Iterable[Point]

def center_of_mass(points: Space) -> Point:
    (...)

def distance(point_a: Point, point_b: Point) -> Number:
    (...)

def mean_distance(*points: Point) -> Number:
    (...)
```

Speaker notes

- Bom, vamos relembrar um pouco dos nossos momentos felizes nas aulas de Física. Um ponto no espaço é um conjunto ordenado de números, no plano de duas dimensões, são dois números. Um espaço no caso eu to definindo como sendo uma iterável qualquer, como uma lista, de pontos
- Na linha number = ... eu defino um tipo novo, o tipo Number e eu digo que um number pode ser um inteiro ou um float. Logo em seguida eu defino um Point como sendo uma tupla com dois Numbers. Finalmente eu defino um espaço como sendo um iterável contendo apenas pontos
- O centro de massa de um conjunto de pontos é mais ou menos o ponto onde o conjunto se equilibra. Em resumo se vc colocar seu dedo na direção do centro de massa do seu notebook, você consegue equilibrar ele na ponta do dedo. Eu não vou fazer isso ao vivo porque na chance de eu errar, o custo é meio alto.
- Mas enfim, a função centro de massa ela deve receber um espaço contendo vários pontos e então ela vai retornar um outro ponto, que é a localização desse centro. Vocês podem ver na definição da função center_of_mass como as anotações foram usadas para expressar esse significado
- A distância entre dois pontos é simplesmente um número. A função "distance" usa das anotações para mostrar que ela aceita dois argumentos, do tipo Point e retornará um número
- E finalmente o último exemplo desse slide mostra como podemos combinar anotações com a sintaxe de argumentos arbitrários. Essa última função nos indica que ela aceita uma quantidade arbitrária de pontos e retornará um número, que representa a distância média entre esses pontos

TAREFA PARA CASA

(Para os que tão achando tudo isso daqui legal)

Você poderia passar um argumento do tipo `Space` na última função?

Teste com as ferramentas que veremos em breve e veja a definição de "is consistent" no PEP 483 ou no artigo sobre Gradual Typing

```python
from typing import TypeVar
from bson import ObjectId

UserID = TypeVar("UserID", str, ObjectId)

def user_age(user_id: UserID) -> int:
    (...)
```

Speaker notes

- Aqui nós temos revisitamos o exemplo da idade do usuário. Considere que a sua função user_age pode aceitar o argumento user_id como sendo uma String ou um ObjectId conforme definido na biblioteca BSON.
- Você pode então definir um tipo arbitrário chamado UserID e definir sua função dessa forma

```python
from typing import Callable

BinaryOp = Callable[[int, int], int]
GenericOp = Callable[..., int]

def is_commutative(operation: BinaryOp) -> bool:
    (...)

def register_callback(callback: GenericOp):
    (...)
```

Speaker notes

- Nesse slide nós vemos como definir assinaturas de função que serão usadas como argumentos de outras funções. Ou seja, vamos passar uma função como parâmetro
- Pra isso nós usamos o tipo Callable. Na linha Binary.. eu defino uma operação binária. Uma operação binária é uma função que aceita dois inteiros e retorna outro inteiro. Operações como soma, subtração, multiplicação, são exemplos
- A sintaxe de um Callable é essa que está apresentada, o primeiro argumento é uma lista contendo o tipo de todos os argumentos usados, seguido de um último argumento que é o tipo de retorno.
- Na linha genericOp = ... eu defino uma operação genérica que aceita um número indefinido de argumentos de tipo também indefinido mas que retorna sempre um inteiro. Para definir esses argumentos genéricos nós usamos o operador "ellipsis" que são 3 pontos. Note que nesse caso os argumentos da função não são separados por colchetes.

STUB FILES

Para quando você não pode ou não quer adicionar anotações no seu código!

Speaker notes

- Tudo isso de type hint através de anotações é muito bom, muito legal mas em alguns casos a gente simplesmente não pode aceitar só versões mais novas de Python. As vezes precisamos manter compatibilidade com versões anteriores a

3.5 e etc. Como usar de tipos então?

Stub Files são anotações de tipo que ficam num arquivo `.pyi`

- Útil para quando o código em si é rodado também por interpretadores incompatíveis

- Útil para quando o código em si é rodado também por interpretadores incompatíveis
- Quando você quer dar anotações de tipos para algum código que não é seu

- Útil para quando o código em si é rodado também por interpretadores incompatíveis
- Quando você quer dar anotações de tipos para algum código que não é seu
- Quando a equipe não quer "sujar" o código com anotações

- Útil para quando o código em si é rodado também por interpretadores incompatíveis
- Quando você quer dar anotações de tipos para algum código que não é seu
- Quando a equipe não quer "sujar" o código com anotações
- Quando você quer liberar anotações de tipo gradativamente

Stub:

```
from typing import TypeVar
from bson import ObjectId

UserID = TypeVar("UserID", str, ObjectId)

def user_age(user_id: UserID) -> int: ...
```

Implementação:

```
def user_age(user_id):
    pass
```

Speaker notes

- Quem aqui já trabalhou com C/C++ talvez reconheça alguma semelhança entre essa estrutura e os headers dessas linguagens. Basicamente nós vamos criar um arquivo com extensão .pyi que irá conter todas as anotações de tipos
- Num arquivo paralelo manteremos a implementação em si, em Python puro

Desenvolvedores podem publicar anotações de tipos para outras libs

Veja o PEP 561 para mais informações sobre como distribuir esses arquivos

Speaker notes

- Stub files permitem que desenvolvedores distribuam anotações de tipos para bibliotecas escritas por terceiros, que podem ou não ainda ser mantidas.
- Esse é um padrão que permite ainda mais gradatividade na adoção de type hints, diminuindo a pressão para liberação e modificação de código

VARIABLE ANNOTATIONS

Type Hinting para variáveis também!

- O PEP 484 definia uma sintaxe baseada em comentários para anotar tipos de variáveis

```
euclidean = [] # type: List[Points]
```

- Essa sintaxe era dada como uma forma de facilitar o processamento do tipo de variáveis em casos complexos.
- No fim do PEP 484, os autores deixam o seguinte comentário

If type hinting proves useful in general, a syntax for typing variables may be provided in a future Python version.

- De fato, um novo PEP foi escrito para lidar com variáveis
- No Python 3.6 ele foi implementado – PEP 526

"Ah, que fantástico"

"Tudo o que eu queria, MAIS uma sintaxe nova"

- Felizmente, a sintaxe usada é herdada daquela que nós já vimos!

```
users: List[User]
age: int = 19
is_valid: bool
name: str = "Felipe"
```

Então Python agora me permite explicitar tipos?

SIM!

E agora o Python me retorna um erro assim que alguma invocação errada acontece?

NÃO

Speaker notes

- Mostrar o conteúdo e a execução de sample1_correct.py

...

então??

Python agora oferece uma forma centralizada de explicitar informações de tipos onde a desenvolvedora do código desejar.

PRA QUÊ SERVE ENTÃO?

PRA QUÊ SERVE ENTÃO?

- Essa informação pode ser capturada de forma automática

PRA QUÊ SERVE ENTÃO?

- Essa informação pode ser capturada de forma automática
- Podemos construir ferramentas de checagem mais complexas (e completas!)

PRA QUÊ SERVE ENTÃO?

- Essa informação pode ser capturada de forma automática
- Podemos construir ferramentas de checagem mais complexas (e completas!)
- Editores de texto e IDEs podem oferecer sugestões melhores

Speaker notes

- Nós podemos ficar na dúvida do que podemos fazer com isso então
- A grande questão é que com um padrão bem estruturado para oferecer essa informação, nós podemos escrever analisadores estáticos que são capazes de pegar erros que podem ter passado despercebidos pela equipe de dev.
- Isso tudo nos permite pegar toda uma família de bugs com mais agilidade e antes que esses bugs sejam colocados no ar

ANALISADORES

- Existem várias ferramentas que podem checar suas anotações de tipos!
- A mais famosa – e que deu origem a tudo isso, de certa forma – é o mypy
- Outra que tem ganho relevância especialmente em projetos grandes é o Pyre

- Pyre foi criado pelo Facebook para ser usado na base de código do Instagram, uma base Python com milhões de linhas de código feitas por diversos programadores
- Recomendo a talk Shannon Zhu - Leveraging the Type System to Write Secure Applications - PyCon 2019 sobre o assunto

FERRAMENTAS SOZINHAS NÃO FAZEM NADA!

Precisamos de **métodos**

Speaker notes

- E finalmente minha última dica é sobre esse monte de ferramentas que a gente tem

- Engenharia de Software é tão legal que ela consegue se introspectar

- Engenharia de Software é tão legal que ela consegue se introspectar
- Tem um problema com desenvolvimento de software?

- Engenharia de Software é tão legal que ela consegue se introspectar
- Tem um problema com desenvolvimento de software?
- Construa outro software pra resolver!

- Engenharia de Software é tão legal que ela consegue se introspectar
- Tem um problema com desenvolvimento de software?
- Construa outro software pra resolver!
- git, Jira, pylint, ...


LinusTechTips @LinusTechTip · Oct 17
too many tech tips..

Speaker notes

- É normal a gente sair de conferências se sentido sobrecarregados de dicas e ferramentas e técnicas novas

- Ter uma ampla carta de opções é fantástico

- Ter uma ampla carta de opções é fantástico
- Nos permite fazer muita coisa legal

- Ter uma ampla carta de opções é fantástico
- Nos permite fazer muita coisa legal
- Mas, especialmente numa equipe, consistência é fundamental!

- Liste as opções, discuta com a sua equipe e defina um padrão

- Liste as opções, discuta com a sua equipe e defina um padrão
- Não se preocupe em definir tudo, não se preocupe em acertar de primeira. Defina algumas linhas gerais

- Liste as opções, discuta com a sua equipe e defina um padrão
- Não se preocupe em definir tudo, não se preocupe em acertar de primeira. Defina algumas linhas gerais
- Qual analisador vocês vão usar? Com quais configurações?

- Liste as opções, discuta com a sua equipe e defina um padrão
- Não se preocupe em definir tudo, não se preocupe em acertar de primeira. Defina algumas linhas gerais
- Qual analisador vocês vão usar? Com quais configurações?
- E então recolha feedback da sua equipe quanto ao uso, modifique seus padrões se necessário

- Como de fato *usar* um analisador estático no desenvolvimento?
- Ferramentas instaladas que não são usadas não servem pra nada!
- Minha sugestão: *hooks pre-commit* e *CI*

Speaker notes

- Uma questão que fica é como usar de fato essas coisas
- Se você abrir o meu computador e o de muito dev por aí, você provavelmente vai encontrar um monte de ferramenta X que a gente instalou achando que ia ser a solução dos nossos problemas e que nunca usamos de fato
- No caso de analisadores estáticos em geral, eu gosto de uma abordagem em duas camadas, hooks pre-commit e scripts no servidor de CI

HOOKS PRE-COMMIT

- Códigos que o `git` pode invocar *antes* de arquivos sofrerem commit
- Recomendo o uso do pre-commit para gerenciar hooks
- Executam na máquina do desenvolvedor
- Evita que código não-conformante vá para o repositório

Speaker notes

- Eu gosto muito da ideia de hooks pre-commit porque eles são executados antes do código ser submetido e, caso eles falham, o código não passa
- Isso mantém o repositório limpo pois é preventivo. Você não precisa fazer um outro commit que vai corrigir alguma coisa que uma ferramenta pegou dps
- O problema de hooks pre-commit normais é que eles não são registrados como arquivos no git, então é difícil mantê-los de forma igual por toda uma equipe
- Eu gosto bastante do projeto pre-commit porque ele te apresenta uma forma estruturada de colocar seus hooks dentro do projeto
- Você pode colocar uma execução do seu analisado estático favorito aqui e todo mundo que mexer no projeto vai ter acesso as configurações

SERVIDOR DE CI

SERVIDOR DE CI

- CI é a prática de manter toda uma base de código íntegra

SERVIDOR DE CI

- CI é a prática de manter toda uma base de código íntegra
- Em geral existe um servidor em algum roda tarefas em cima do código novo como testes e verificações

SERVIDOR DE CI

- CI é a prática de manter toda uma base de código íntegra
- Em geral existe um servidor em algum roda tarefas em cima do código novo como testes e verificações
- Eu gosto de colocar todos os analisadores estáticos ali também

SERVIDOR DE CI

- CI é a prática de manter toda uma base de código íntegra
- Em geral existe um servidor em algum roda tarefas em cima do código novo como testes e verificações
- Eu gosto de colocar todos os analisadores estáticos ali também
- Evita que um dev que não configurou pre-commit ou não está acostumado com a equipe viole o estilo que foi combinado

- Erros no servidor de CI são uma ótima forma de disseminar boas práticas!

- Erros no servidor de CI são uma ótima forma de disseminar boas práticas!
- Ao invés de simplesmente colocar um erro cru, coloque alguma referência sobre a documentação de boas práticas ou até mesmo o contato de alguém mais sênior que pode ajudar com o problema

Speaker notes

- Isso é uma coisa que eu gosto bastante mas não vejo muita gente usando
- Erros no servidor de CI podem ser usados para disseminar as boas práticas da equipe
- Se as mesmas ferramentas são executadas no pre-commit e no CI e o código de alguém quebrou por conta de alguma coisa no pre-commit, isso indica que a pessoa não configurou direito o próprio ambiente
- O erro do CI é uma ótima oportunidade de colocar um texto claro sobre o que aconteceu, sobre as boas práticas e até mesmo colocar o contato de alguém mais sênior que se disponibilize a ajudar os colegas nessa situação

INFORMAÇÃO, SEGURANÇA & PRODUTIVIDADE

felipe@felipevr.com — github.com/fbidu

Speaker notes

- Nessa parte da minha palestra eu vou tocar por cima de alguns pontos que eu acho relevantes nessa discussão

- Desenvolver Software é uma tarefa humana
- Software é feito por humanos, muitas vezes para humanos
- Muitos problemas em Engenharia de Software são problemas de comunicação

Speaker notes

- Uma coisa que muitas vezes passa despercebida é como programação e produção de software é uma tarefa criativa, lotada de fatores humanos
- Por algum tempo - e algumas pessoas até hoje pensam assim - pensava-se que o software era uma coisa absolutamente objetiva, livre das "impurezas" dos humanos envolvidos no seu desenvolvimento
- Isso não poderia estar mais longe da verdade. O desenvolvimento de software é uma tarefa bastante criativa e que sofre bastante influência dos fatores humanos no processo
- Muitas vezes, equipes de software enroscam e não conseguem atingir sua eficiência máxima não por conta de razões técnicas, por que algum treinamento ou conhecimento está faltando ou coisa assim
- Muitas vezes as equipes acabam sofrendo muito com problemas que são relacionados com a interação entre as pessoas. Com atritos, com problemas de comunicação, com ego, com problemas em ouvir opiniões diferentes e etc

- Ferramentas estáveis, que tiram do programador pesos desnecessários são bem vindas

- Ferramentas estáveis, que tiram do programador pesos desnecessários são bem vindas
- Cada minuto economizado deixando de caçar por erros "bobos" pode ser usado para fazer outra coisa, trabalhar em outra tarefa, descansar, etc

- Nenhuma ferramenta é perfeita
- Mas quando adotamos práticas e técnicas que nos ajudam a lidar melhor com o próprio trabalho, todo mundo ganha

- Existe uma série de problemas de segurança que foram causados por erros tecnicamente pequenos
- Mas que vieram a existir por conta de programadores cansados, sobrecarregados, etc
- Heartbleed foi um ótimo exemplo

- Type Hinting pode nos ajudar com isso

- Type Hinting pode nos ajudar com isso
- Quanto maior a base de código, mais essa técnica pode se tornar útil

- Type Hinting pode nos ajudar com isso
- Quanto maior a base de código, mais essa técnica pode se tornar útil
- Nem ela – e nem nada – vai capturar e evitar todos os bugs

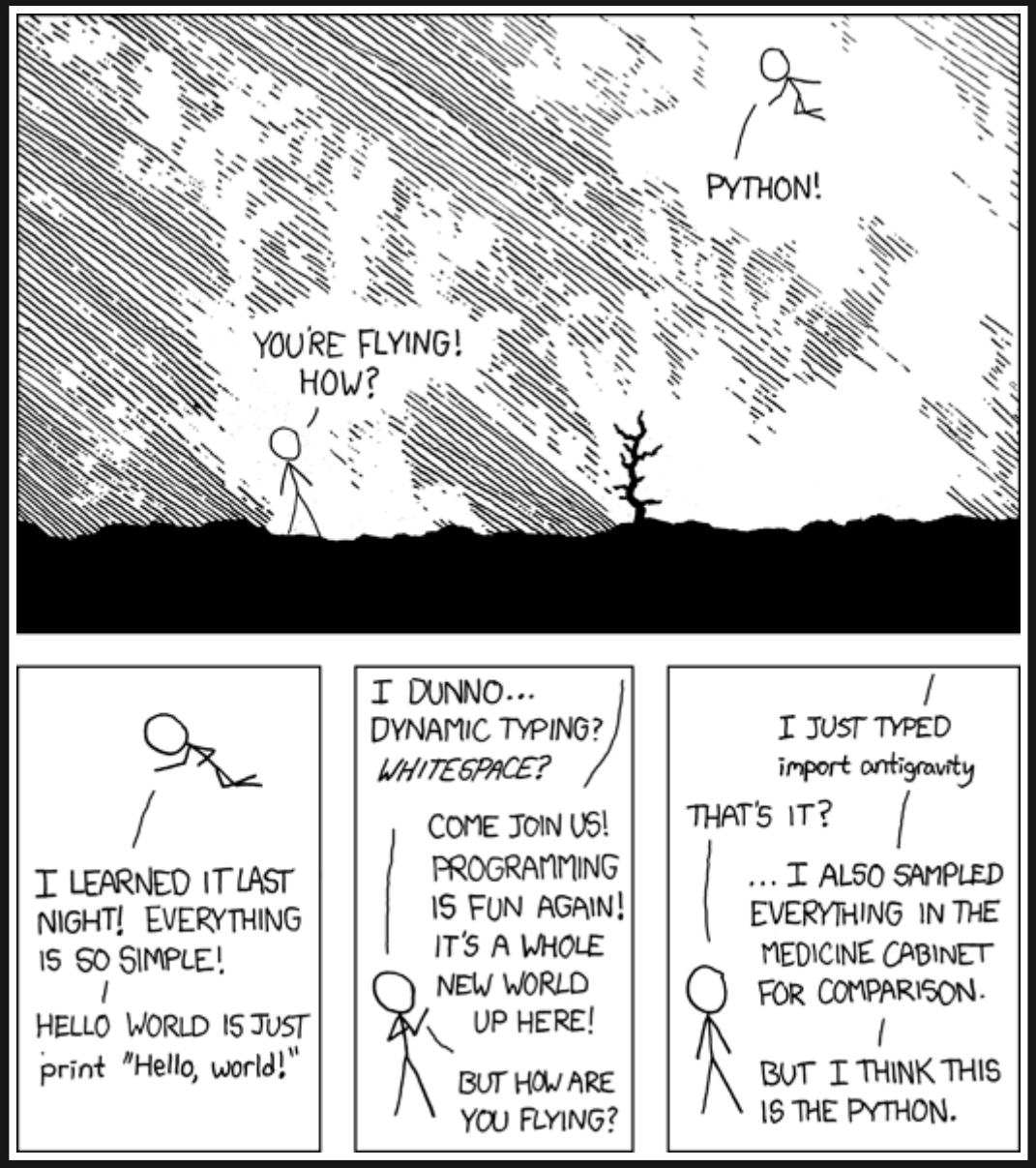
- Type Hinting pode nos ajudar com isso
- Quanto maior a base de código, mais essa técnica pode se tornar útil
- Nem ela – e nem nada – vai capturar e evitar todos os bugs
- Mas dando mais informações de uma forma elegante e organizada, podemos capturar mais bugs e nos ajudar a cometer menos erros

LEMBRE-SE DOS HUMANOS

"Mind the human"

felipe@felipevr.com — github.com/fbidu

- Programar em Python é divertido
- O que fez Python ser o que é somos nós, as pessoas


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.

Programming is fun again!

— xkcd 353

- Na hora de escolher entre técnicas e ferramentas

- Na hora de escolher entre técnicas e ferramentas
- Na hora de revisar o código de um colega

- Na hora de escolher entre técnicas e ferramentas
- Na hora de revisar o código de um colega
- Na hora de contratar alguém novo

LEMBRE-SE DOS HUMANOS!

- Type Hinting — ou qualquer outra técnica — ajudam quando elas ajudam os seres humanos por trás do código

felipe@felipevr.com — github.com/fbidu

Speaker notes

- O legal de uma python brasil é isso - a união da comunidade
- Ontem, por exemplo, eu tive o prazer de me juntar a essas pessoas fantásticas para jantar e conversar
- E entre tudo o que a gente falou, uma coisa todos nós concordamos que Python é o que é e chegou onde chegou por conta da preocupação com as pessoas
- Isso é uma coisa que todos nós devemos nos lembrar e praticar no nosso dia-a-dia, na nossa comunidade local, nas nossas empresas, etc
- E eu quero agradecer muito a Carol Willing, Melissa Weber, João Bueno, Lorena Mesa e Luciano Ramalho por terem também me lembrado disso
- E eu quero agradecer a todos vocês por estarem aqui e me ouvido nesse tempo. Somos nós quem formamos essa comunidade incrível e somos nós que tornamos tudo isso possível

MUITO OBRIGADO!

- felipe@felipevr.com
- github.com/fbidu
- Twitter @fevir0