Detectando excepciones en código Python

por david weil / dave / tenuki

Acerca de mí

•Me interesa la matemática, la fotografía, Japón, los lenguajes de programación y algunas cosas de seguridad informática..

•Programo en python desde el 2003 aprox., cuando trabajaba en Core Security..

•Conocí PyAr cuando eramos 5 en un bar, participé de la org. de la 1er. PyCon-Ar, etc..

Agenda

Introducción al problema
Otros lenguajes e intuición
Herramientas disponibles
Un modelo sencillo
Problemas
Ejemplos

Introducción al problema

From: Hernan ...
To: pyar@python.org.ar
Subject: Que exceptions tiene una función

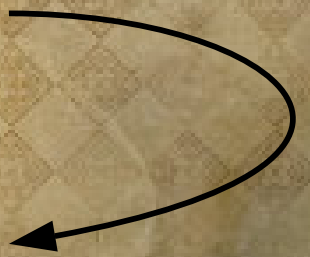
```
def f2(x):
    if not isinstance(x, str):
        raise TypeError
    return f1(x)

>>> inspect_exceptions(f2)
(KeyError, TypeError)
```

Intro: Otros lenguajes

- Smalltalk: no es posible especificarlas
- Java: es requerido especificarlas*
- Python…

Intro: Otros lenguajes

- Smalltalk: no es posible especificarlas
- Java: es requerido especificarlas*
- Python: **PEP0484** – Type Hints



"sentite libre de especificarlas..
..en un comentario" !?!

Intro: Intuición

¿qué tan difícil puede ser detectarlas?

- Una pavada, basta leer el código / hacer grep..
- No se puede!!
- Es **imposible**!!
- Algo podemos hacer..

Herramientas disponibles..

.. de análisis estático

•PyChecker: is a tool for finding bugs in python source code. (no estatico?)

•Pyflakes:A simple program which checks Python source files for errors.

•**PyLint:** en Python, el referente en análisis estático

•Pydev: Code analysis provides error finding in python programs. It finds common errors..

Herramientas disponibles: PyLint

- Modela python e incluye mas de 100 chequeos distintos! (y soporta **plugins**!)
- Nos permite trabajar con el source
- .. y también con el **Abstract Syntax Tree**
- Tiene un motor de **inferencia de tipos**!!!

.. manos a la obra!

Herramientas disponibles: PyLint

Implementar un plugin es extremadamente fácil:

1. Definir una subclase de **`BaseChecker`**.

2. Agregarle lo que vaya a implementar: **`IAstroidChecker`**, **`IRawChecker`** o **`ITokenChecker`**.

3. Definir los mensajes correspondientes a los chequeos que implemente en un diccionario en la clase, un nombre y una prioridad para definir el orden de ejecución.

Herramientas disponibles: PyLint

```
class MiPrimerChecker(BaseChecker):
  __implements__ = IAstroidChecker
  name = “Mi Primer chequeador”
  msgs = {
    'W0999':('Encontre raise en %s',
'try-found', 'se encontro un raise')}
  priority = -1

  def visit_raise(self, node):
    self.add_message('raise-found',
      line=node.line, node=node)
```

Un modelo sencillo

Funciones

- Llamadas entrantes
- Llamadas salientes
  - Filtrado / atrapado
- Excepciones lanzadas

Excepciones

- Jerarquía
- Filtrado / atrapado

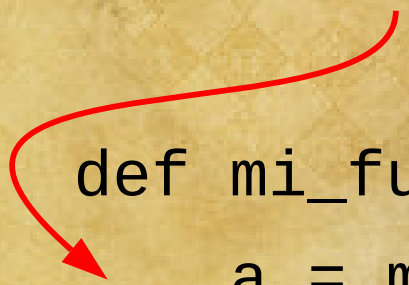
Un modelo sencillo: problema 1

Funciona bastante bien para casos sencillos..

```
class A:
  def f(self):
      raise Exception()

def mi_funcion():
  a = A()
  a.f()
```

Se pincha en casos como el siguiente:

```
def mi_otra_funcion():
    return A()

def mi_funcion():
    a = mi_otra_funcion()
    a.f()
```



Inferencia / extrema

Con PyLint tenemos una función, en cada nodo del AST que nos devuelve un iterador sobre los posibles tipos de una función: ***`node.infer`***

Tipo=**tipo**, **clase** ó **`_Yes`** (no sé/no determinado)

Inferencia "extrema": si no sabemos el tipo, pero sabemos el nombre del método, asumimos TODAS las clases que lo implementan!

Por ejemplo, el metodo: ***`__length__`***

Un modelo sencillo: problema 2

Los errores son excepciones

```
class A:
  pass


def mi_funcion():
  A().no_existente()


AttributeError: A
instance has no
attribute 'no_existente'
```

PyLint al rescate!

```
$ pylint not_found.py


E: 5, 0: Instance of 'A' has
no 'no_existente' member
(no-member)
```

Un modelo sencillo: problema 3

```
def caso1(x):
  return [x]+collatz(x/2)

def caso2(x):
  return [x]+collatz(3*k+1)

def collatz(x):
  if x<=1:
    return []
  if x%2==0:
    return caso1(x)
  return caso2(x)
```

No en realidad!

Los llamados recursivos no son un problema.

Al recorrer el grafo, si un método ya lo analizamos antes, listo!

Un modelo sencillo: problema 4

`import antigravity`

PyLint procesa únicamente los sources que uno especifica.. :-(

Resumen y detalles de implementación

- excepcion-directa / excepcion-encontrada
- Lo implementado hasta aca **no incluye** "conocimiento" de python, ejemplos:
  - `diccionario[item]`
  - `lista[slice]` o: `lista[item]`

  etc., es una prueba de concepto
- Falta definir "**entry-points**"
- No se chequea conformidad de operadores:
  - __getitem__, __len__, etc.. (seria un extra)

Conclusiones

- Caso general / respuesta exacta: **imposible**.
- **Casos particulares y soluciones aproximadas: hecho.**
- Hay muchas librerias en python para hacer CASI todo.. asi que para cualquier cosa que quieras hacer:
  - no es necesario re-inventar la rueda (aunque puede ser entretenido :-))
  - Ya debe existir algo nos simplifique un poco la tarea!

Ejemplos / ¿preguntas? / links

Python:

- https://bitbucket.org/tenuki/python-exception-inference
- https://www.python.org/dev/peps/pep-0484/
- http://www.pylint.org/
- http://www.pydev.org/manual_adv_code_analysis.html
- http://pychecker.sourceforge.net/
- https://github.com/pyflakes/pyflakes

Imágenes:

- Escher
- xkcd: Python

Licencia de la presentación: