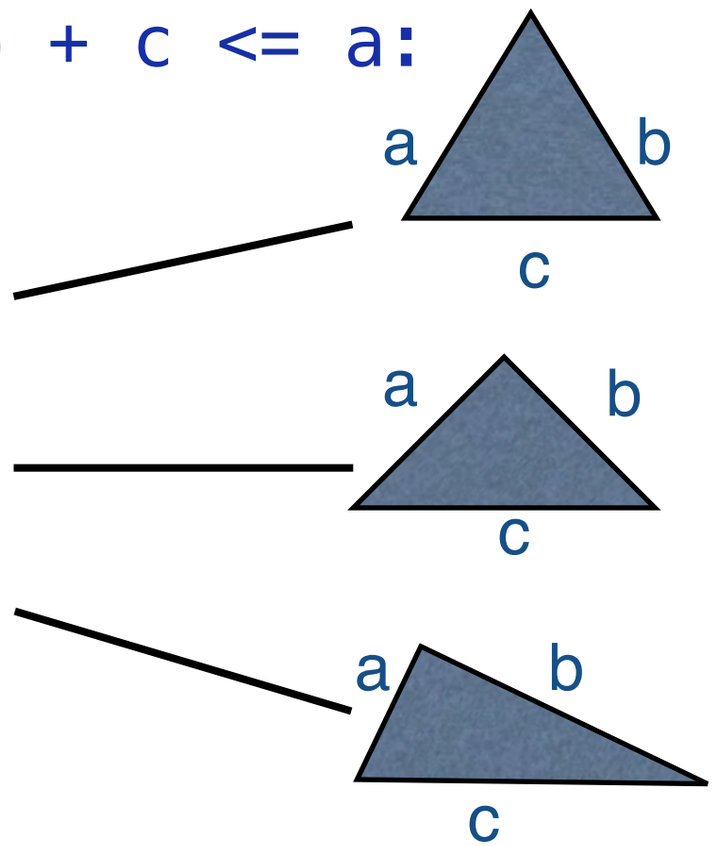
TP1: RandPy

Generación Automática de Casos de Test - 2016

(C) Zeller,Kampmann,Hoesller

Un Ejemplo

```
def triang(a, b, c):
    if a + b <= c or a + c <= b or b + c <= a:
        return NOT_A_TRIANGLE
    elif a == b == c:
        return EQUILATERAL_TRIANGLE
    elif a == b or b == c or a == c:
        return ISOSCELES_TRIANGLE
    else:
        return SCALENE_TRIANGLE
```



Random Driver

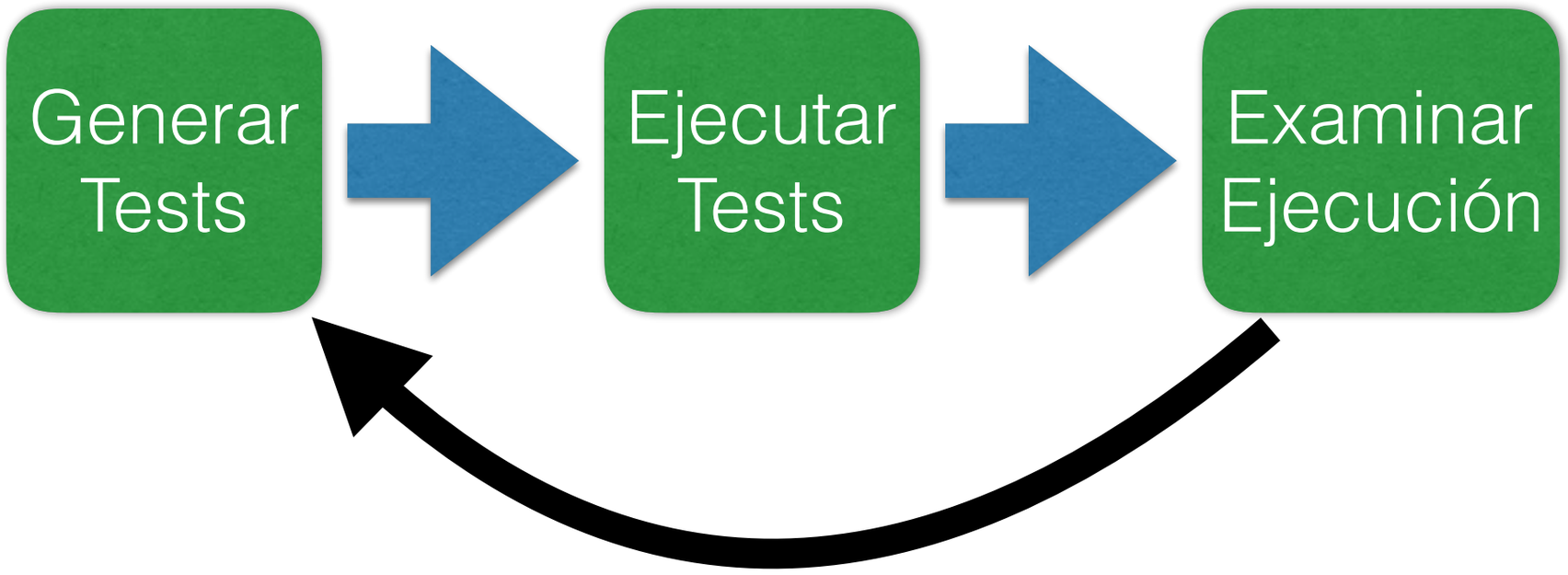
```
from Triangle import triangle
import random

def random_tester():
    i = 0
    while i < 1000:
        a = random.randint(0, 100)
        b = random.randint(0, 100)
        c = random.randint(0, 100)
        result = triangle(a, b, c)
        print a, b, c, result
        i = i + 1
```

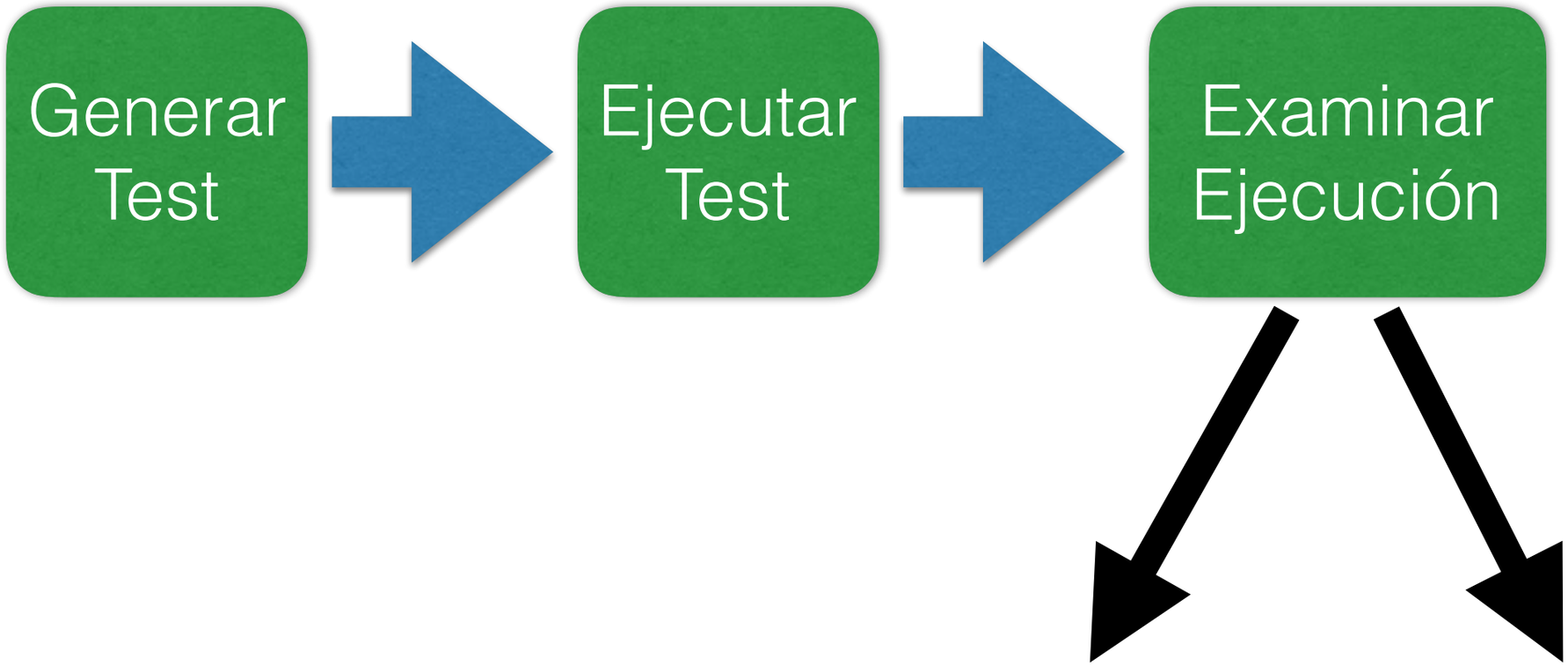
Un Driver Genérico

- Queremos un *generador de tests* tal que:
  - toma una *función arbitraria*
  - provee input apropiado para cada tipo

Random Testing guiado por Feedback

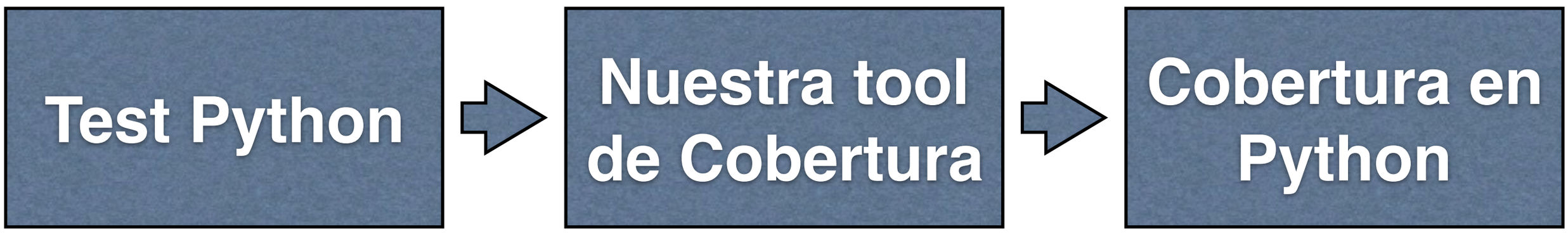


Random Testing



Infraestructura: Taller #1

- Tenemos la posibilidad de Medir la Cobertura de un Test Python



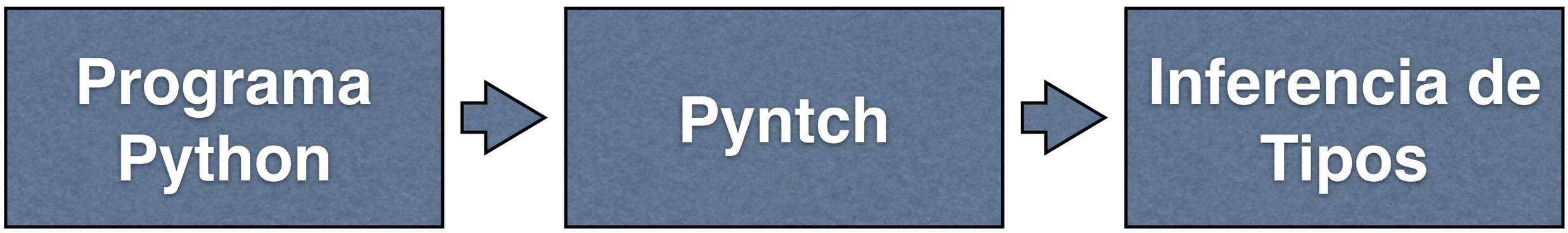
Un Driver Genérico

- Queremos un *generador de tests* tal que:
  - toma una *función arbitraria*
  - provee input apropiado para cada tipo

Pero Python no tiene tipado estático!

Pyntch

- Pyntch es un analizador de código estático para Python
- Analiza las invocaciones existentes



Pyntch – Static code analyzer for Python

www.unixuser.org/~euske/python/pyntch/

Reader

News More Work Papers Apple Travel Shop Money Search Comics Mail Scripting ICST2014 Proceedings

Pyntch

Static code analyzer for Python

Homepage Recent Changes

Last Modified: Thu Oct 29 00:29:04 JST 2009

Table of Contents:

www.unixuser.org/~euske/python/pyntch/

Download:
http://www.unixuser.org/~euske/python/pyntch/pyntch-20091028.tar.gz (50KBytes)

Discussion: (for questions and comments, post here)
http://groups.google.com/group/pyntch-users/

View the source:
http://code.google.com/p/pyntch/source/browse/trunk/pyntch

What's It?

Pyntch (pron. "*pinch*", originally means PYthoN Type CHecker) is a static code analyzer for Python programming language. It detects possible runtime errors before actually running a code. If you have been constantly bothered by a TypeError or AttributeError caused by giving a wrong type of objects, Pyntch is a tool for you. Pyntch examines a source code and infers all possible types of variables, attributes, function arguments, and return values of each function or method (take a look at a sample output below). Then it detects possible exceptions caused by type mismatch, attribute

Ejemplo Pyntch

```
if __name__ == '__main__':
    # use pyntch to get type information
    pyntch_wrapper.setup()
    pyFilename = os.getcwd() + "../examples/pyntch_example.py"
    module = pyntch_wrapper.addFile(pyFilename)
    pyntch_wrapper.runAnalysis()

    # display all functions of pyntch_example
    for f in module.children:
        if isinstance(f, pyntch.function.FuncType):
            print("Function " + f.name)
            for argname in f.argnames:
                print("  argName:" + argname)
                argTypes = f.space[argname].types
                types_str = print_pyntch_types(argTypes)
                print("  argTypes: " + types_str)
```

Ejemplo Pyntch

```
""" Returns a string with a sequence of type declarations """
def print_pyntch_types(ts):
    type_str = "["
    for t in ts:
        if not type_str == "[":
            type_str += ","
        type_str += print_pyntch_type(t)
    type_str += "]"
    return type_str
```

```
""" Returns a string for a single type declaration """
def print_pyntch_type(t):
    if isinstance(t, pyntch.aggregate_types.ListObject):
        elem_types = t.elemall.types
        return "list" + print_pyntch_types(elem_types)
    elif isinstance(t, pyntch.aggregate_types.DictObject):
        key_types = t.key.types
        value_types = t.value.types
        key_types_str = print_pyntch_types(key_types)
        value_types_str = print_pyntch_types(value_types)
        return "dict(" + key_types_str + "->" + value_types_str + ")"
    elif isinstance(t, pyntch.aggregate_types.TupleObject):
        elem_types = t.elemall.types
        return "tuple" + print_pyntch_types(elem_types)
    elif isinstance(t, pyntch.aggregate_types.SetObject):
        elem_types = t.elemall.types
        return "set" + print_pyntch_types(elem_types)
    else:
```

```
elif isinstance(t, pyntch.aggregate_types.DictObject):
    key_types = t.key.types
    value_types = t.value.types
    key_types_str = print_pyntch_types(key_types)
    value_types_str = print_pyntch_types(value_types)
    return "dict(" + key_types_str + "->" + value_types_str + ")"
elif isinstance(t, pyntch.aggregate_types.TupleObject):
    elem_types = t.elemall.types
    return "tuple" + print_pyntch_types(elem_types)
elif isinstance(t, pyntch.aggregate_types.SetObject):
    elem_types = t.elemall.types
    return "set" + print_pyntch_types(elem_types)
else:
    type_str = str(t)
    if type_str == "<int>":
        return "<int>"
    elif type_str == "<bool>":
        return "<bool>"
    elif type_str == "<float>":
        return "<float>"
    elif type_str == "<long>":
        return "<long>"
    elif type_str == "<basestring>" or type_str=="<unicode>" or type_str=="<str>":
        return type_str
    else:
        raise Exception("Unknown Pyntch primitive type" + str(t))
```

Enunciado

- Desarrollar un random test generator para funciones Python que reciben sólo tipos primitivos:
    - Ejemplo: cgi_decode(“Hello World”)
    - Utilizar pyntch para inferir automáticamente el tipo de cada argumento de las funciones
        - Ignorar funciones “privadas” (empiezan con “_”)
    - Completar las clases RandPy y TestGenerator

RandPy - Argumentos

- **targetPyFile**: el archivo .py con el módulo target (str)
- **maxIterations**: la cantidad máxima de iteraciones del algoritmo (int). El valor -1 significa que se ignora.
- **maxTime**: la cantidad de segundos máximo a utilizar en la generación (int)
- **output_dir**: la carpeta donde se almacenarán los tests generados (str)
- **random_seed**: la semilla de aleatoriedad para tener reproducibilidad (int)

RandPy.generate_tests()

```
tests = list()
while quedaTiempo() and (maxIterations==-1 or faltanIteraciones()):
    t = generamos un nuevo test
    r = ejecutamos el nuevo test
    if hubo TypeError:
      descartamos t
    if cobertura de branches y/o líneas aumentó:
      tests.append(t)
      escribimos t en la carpeta outputDir
    else:
      descartamos t
```

RandPy

TestGenerator Genera un nuevo Test

TestExecutor Ejecuta un Test

TestWriter Escribe un Test

TestCall

- Es una invocación a una función bajo test usando una lista de argumentos
- Por ejemplo:
  - `cgi_decode(“Hello World”)`

Nombre de la función

Lista de Argumentos

TestArgument

- Es un valor para ser usado como argumento en un **TestCall**
- Se compone de:
  - Un tipo: ‘constant’, ‘list’, ‘tuple’, ‘dict’, ‘set’
  - El valor: 0, 1, [], [0], {“Juan”:0} que coincide con el valor

Ejemplo

cgi_decode(“Hello World”)

```
TestCall(“cgi_decode”,
[TestArgument(“constant”,”Hello World”)])
```

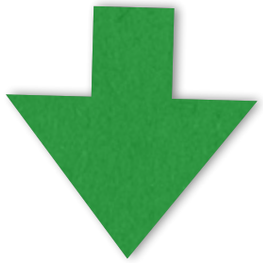
Ejemplo

max(-1,100)

```
TestCall(“max”,
[TestArgument(“constant”,-1),
TestArgument(“constant”,100)])
```

Ejemplo

average([0])



```
TestCall(“max”,
[TestArgument(“list”,
[TestArgument(“constant”,0)])]
```

TestExecutor

- Esta clase se encarga de ejecutar un TestCall
- Crea un TestExecutor para cgi_decode.py:

```
exec = TestExecutor(“cgi_decode.py”)
```

- Permite ejecutar un TestCall usando funciones de cgi_decode.py:

```
err_code exec.execute(test_call)
```

- Los valores posibles de err_code son “OK”, “Exception” o “TypeError”

TestWriter

- Esta clase permite escribir un TestCall como un archivo .py
- Crea un nuevo TestWriter que escribirá el archivo como un test para el módulo cgi_decode en el directorio “output”

```
w = TestWriter(“cgi_decode”,”output”)
```

- Escribe un nuevo archivo que contendrá el test_call con el sufijo 10

```
w.writeToFile(test_call,10)
```

TestGenerator

- Es la clase encargada de generar aleatoriamente un TestCall
- Crear un nuevo TestGenerator para el módulo “cgi_decode” usando la semilla 0

```
g = TestGenerator(“cgi_decode”,0)
```

- Genera aleatoriamente un nuevo test call para el módulo indicado en el constructor

```
t = g.generate_new_test_call()
```

TestGenerator

```
class TestGenerator:

    def __init__(self,module_name,random_seed):

        # COMPLETAR

        pass


def generate_new_test_call(self):

    # COMPLETAR

    return None
```

RandPy

```
class RandPy:

    def generate_tests(self, targetPyFile,
    maxTests, maxTime, output_dir, random_seed):

        # COMPLETAR

        return None
```

Enunciado

- Completar las clases RandPy y TestGenerator para tener un generador aleatorio de tests
- El generador debe construir valores aleatorios de:
  - Tipos básicos: <float>, <int>, <long>, <bool>, <long>, <str>
  - Tipos compuestos: listas, tuplas, diccionarios, conjuntos

Enunciado

- En la carpeta “examples” está el benchmark a utilizar
- RandPy debe funcionar correctamente (ie no crashear) con todos los examples y usando distintas semillas de aleatoriedad
- El test suite debe ser minimal (cada test contribuye a mejorar el coverage de líneas y branches)

Evaluación

- Ejecutar 10 veces RandPy sobre cada uno de los módulos de la carpeta "examples"
- Reportar el promedio de cobertura de líneas y de branches

Example	Avg. Line Cov.	Avg. Branch Cov.
arrays.py		
cgi_decode.py		
convexhull.py		
coord.py		
encryption.py		
levenshtein.py		
persons.py		
sets.py		
sorting.py		
triangle.py		
tuples.py		
years.py		

Requisitos

- Python >=3.4
- Pyntch
- Eclipse (>= 4.5.0) + PyDev

Entrega: Miércoles 28 de Septiembre