AYUDANTÍA
EXAMEN 1

Siéntense lo más adelante posible para leer código

TESTING

Florencia

Estructura Test

```
1   class EstructuraTest(unittest.TestCase):
2
3       def setUp(self):
4           # creamos variables necesarias para los test
5
6       def tearDown(self):
7           # cerramos lo creado en setUp
8
9       def test_function(self):
10          # test de una funcion
11
12  unittest.main()
```

DECORADORES

Andrés

P4d I1 2015-2

Implemente un decorador que haga que la función decorada sea re-ejecutada n veces con intervalos de espera de s segundos cada vez que la función decorada retorne False. Considere que n y s son parámetros del decorador.

GRAFOS

Andrés

P3 Examen 2015-1 (1/3)

```
4   class AN:
5
6       def __init__(self, sc=None, dd=None):
7           self.dd = dd
8           self.sc = sc
9           self.hc = {}
10
11      def tres(self, val, future_dd_val):
12          if self.sc == future_dd_val:
13              self.hc[val] = AN(sc=val, dd=self)
14              return True
15          for c in self.hc.values():
16              c.tres(val, future_dd_val)
17          return False
18
19      def nif(self, sc, cm_):
20          if self.sc == sc:
21              return cm_
22          for c in self.hc.values():
23              out = c.nif(sc, cm_ + [c.sc])
24              if out is not None:
25                  return out
26          return None
```

P3 Examen 2015-1 (2/3)

```
28      def deno(self, cm_):
29
30          if len(cm_) == 0:
31              aux = copy.copy(self.dd.hc[self.sc])
32              del self.dd.hc[self.sc]
33              return aux
34          next = cm_.pop(0)
35          return self.hc[next].deno(cm_)
36
37      def tres_ned_cm(self, ned, cm_):
38          if len(cm_) == 0:
39              self.hc[ned.sc] = ned
40              return
41
42          next = cm_.pop(0)
43          self.hc[next].tres_ned_cm(ned, cm_)
44
45      def inte(self, val_origen, val_destino):
46          cm_1 = self.nif(val_origen, [])
47          cm_2 = self.nif(val_destino, [])
48          aux_1 = self.deno(copy.copy(cm_1))
49          aux_2 = self.deno(copy.copy(cm_2))
50          self.tres_ned_cm(aux_2, cm_1[:-1])
51          self.tres_ned_cm(aux_1, cm_2[:-1])
```

P3 Examen 2015-1

1. ¿Qué hace cada método?
2. ¿A qué corresponde esta clase?
3. ¿Qué resulta de la ejecución lo siguiente?
    a. L55-64
    b. L65-66
    c. L67
    d. L68-70
    e. L71

P3 Examen 2015-1 (3/3)

```
54  if __name__ == "__main__":
55      an = AN(4)
56      an.tres(2, 4)
57      an.tres(3, 4)
58      an.tres(1, 2)
59      an.tres(5, 2)
60      an.tres(6, 3)
61      an.tres(8, 2)
62      an.tres(7, 3)
63      an.tres(7, 6)
64      an.tres(10, 11)
65      out = an.nif(7, [])
66      print(out)
67      an.deno(out)
68      an_n = AN(15)
69      cm_aux = [2, 5]
70      an.tres_ned_cm(an_n, cm_aux)
71      an.inte(2, 6)
```

a

b

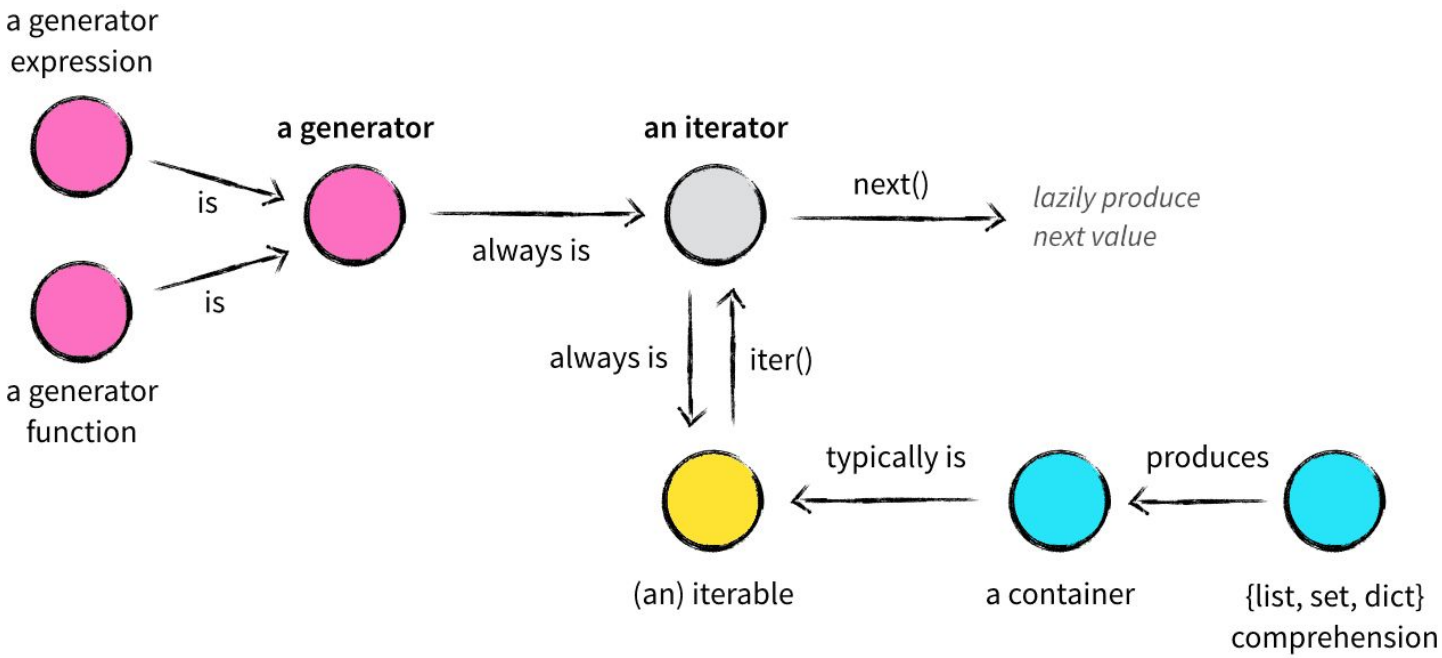
c

d

e

Nos vemos en el siguiente módulo.

En la CS203



Fuente: http://nvie.com/posts/iterators-vs-generators/

AYUDANTÍA
EXAMEN 2

NETWORKING

Florencia

Ejercicio Networking

Te piden crear un sistema de encuestas. Como usuario quieres ver las encuestas disponibles, ver sus resultados y poder contestarlas.

Asume que el servidor tiene las encuestas ya creadas y estas son solo de una pregunta de alternativa.

Explica cómo implementar en python una conexión TCP que pueda resolver el problema.

METACLASES

Benja

Metaclases

Métodos que se ejecutan al crear una clase:

- Construcción de la clase: __new__
- Inicialización de la clase: __init__

Método que se ejecuta con clase ya creada:

- Instanciación de la clase: __call__

¡Métodos __new__ e __init__ reciben los mismos argumentos! Sin embargo, en __new__ se trabaja con la metaclase.

P3 EXAMEN 2015-02

```
1   class B:
2       pass
3
4
5   class Meta(type):
6
7       def __new__(cls, name, bases, attr):
8           print('in 1 ... ')
9           print(cls)
10          print(name)
11          print(bases)
12          name = "cambio_" + name
13          bases = (B, )
14          attr['new'] = 1
15          return super().__new__(cls, name, bases, attr)
16
17      def __init__(cls, name, bases, attr):
18          print('in 2 ... ')
19          print(cls)
20          print('name: ', name)
21          print('bases: ', bases)
22          print('attr: ', attr)
23          print(cls.__name__)
24          return super().__init__(name, bases, attr)
```

P3 EXAMEN 2015-02

```
26  if __name__ == '__main__':
27
28      class A(metaclass=Meta):
29          pass
30
31      print(A.__mro__)
```

THREADING

Juan

Resumen rápido:

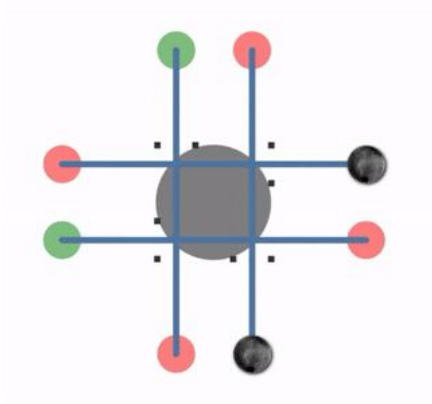
- Permite ejecutar varios procesos livianos en forma simultánea

- Pueden ser una función o una clase
  - Una clase debe implementar el método run

- Join: frena el programa principal

- Daemon: Si el programa principal termina, entonces el thread también

- Manejo de concurrencia con Locks
  - Cuidado con hacer un deadlock!



Chile vs Alemania

Explicar cómo haría una simulación de la final de la Copa Confederaciones. En específico, especificar qué objetos debiesen ser Threads y que consideraciones se debiesen tomar en cuenta.

Recomendación:

1. Modelar el problema
   a. ¿Qué cosas deberían ser threads?
2. ¿Algún thread depende de que otro finalice antes?
3. ¿Qué situaciones podrían generar concurrencia?