Python Training.....

-- Jeetendra Bhattad

Agenda

- Iterators
- Iterator User Defined Class / Objects
- Generators
- Assignments

Iterators

- Iterators are objects which support iteration.
- Iterator objects must have next & __iter__ methods.
- __iter__ should return the iterator itself.
- next should return the next element & raise StopIteration when finished.
- Any sequence can be turned into an iterator using the built-in function iter.

Simple Iterator Example

```
#!/usr/bin/python

def main(x):
    i = iter(range(x))
    print (next(i))
    print (next(i))
    print (next(i))
    print (next(i))

#boiler-plate
if __name__ == "__main__":
    main(5)
```

Making Class-Object's Iterable

```
#!/usr/bin/python

class AutoGenerate:
    def __init__(self, start, end, step=1):
        self.start = start
        self.end = end
        self.step = step
    def next(self):
        self.start += self.step
        if self.start >= self.end:
            raise StopIteration
        return self.start
    def __iter__(self):
        return self

def main():
    x = AutoGenerate(0, 100, 5)
    #for y in x:
    #    print y
    z = iter(x)
    print(next(z))
    print(next(z))
    print(next(z))

if __name__ == "__main__":
    main()
```

Iterators : yeild -> Generators

Yield : implies transfer of control is temporary & voluntary and function expects to get control again.

Generators are simple functions which returns an Object on which next() can be invoked, such that every call returns some value and should raise StopIteration exception when done.

```
#!/usr/bin/python

def SampleGenerator():

    yield 1

    yield 2

    yield 3

if __name__ == "__main__":

    for x in SampleGenerator():

        print (x)
```

Generators

```
#!/usr/bin/python
import re
def GeneratorGrep(pattern, file_name):
    fd = open(file_name)
    pat = re.compile(pattern)
    for line in fd:
        if pat.search(line):
            yield line
    fd.close()
def main():
    file_name = input("Enter File Name from which to extract single line comments:")
    comment_line_generator = GeneratorGrep("\A#", file_name)
    for line in comment_line_generator:
        print(line)
if __name__ == "__main__":
    main()
```

Decorators

It provides a very useful way to add additional functionality to existing functions and classes.

They are functions that wrap other functions or classes.

In simple terms it is a way to dynamically add new behavior to some objects.

Decorators

```
#!/usr/bin/python

def InitGenerator(func):
    def initialize(*args, **kwargs):
        gen = func(*args, **kwargs)
        next(gen)
        return gen
    return initialize

def ToUpper():
    while True:
        string = yield # coroutines
        print(string.upper())

@InitGenerator
def ToUpperDecorated():
    while True:
        string = yield
        print(string.upper())
```