JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

Johns Hopkins Engineering

Module 9: Object Oriented Programming II

Python Pointer: Generators and Yield

EN605.206: Introduction to Programming in Python

Generator Functions (Generators)

A **generator** is a function that behaves like an iterator. Each element is "generated" by a **lazy iterator** and control is yielded back to the caller. Generators are of particular benefit over functions when you're dealing with a potentially large, or even infinite, amount of data. A generator is created using the `yield` keyword.

Do not attempt!

main.py

```python
def sequence(N):
    numbers = []

    for i in range(N):
        numbers.append(id)

    return numbers

N = 40000000

for i in sequence(N):
    print(i)
```

Function: not enough memory to store and return 40 million int's

main.py

```python
def sequence(N):
    for i in range(N):
        yield i

N = 40000000

for i in sequence(N):
    print(i)
```

```
3781
3782
3783
3784
3785
3786
3787
3788
3789
3790
3791
```

Generator: "generate" one value at-a-time, passing control back to the call each time

yield

To create a generator, we use the `yield` keyword instead of `return`. In a function we compute some result in its entirety then `return` that result, as well as execution control, back to the caller. In a generator, however, we "generate" (compute) the next value only and `yield` it to the caller. So we give control flow back, but only until the caller wants to retrive the next item in the generator.

main.py

```
def sequence():
    yield 1
    yield 2
    yield 3

gen = sequence()
print(gen)

for i in gen:
    print(i)
```

```
<generator object sequence at 0x7fec1583ecf0>
1
2
3
```

main.py

```
def sequence():
    yield 1
    print("Waiting...")
    yield 2

gen = sequence()
print(next(gen))
print(next(gen))
```

```
1
Waiting...
2
```

Generator Function Example

State of iterator is saved, function is paused

Generator object

Cannot iterate more than once

main.py

```python
def count_lines(file):
    f = open(file)
    lines = f.read().split("\n")
    return lines

def count_lines_gen(file):
    for line in open(file):
        yield line

i = 0
for line in count_lines("file.txt"):
    i += 1
print(i)

generator = count_lines_gen("file.txt")

i = 0
for line in generator:
    i += 1
print(i)

i = 0
for line in generator:
    i += 1
print(i)
```

Functions vs. Generators

Here are some of the key differences between a **generator** and a **function**.

Functions	Generators
Uses the `return` statement	Uses the `yield` statement
Begins executing when called	Returns a generator object
Control returned to caller when finished	Function pauses after `yield`, can be resumed
Stack space is deallocated after return	Stack space maintained after yield
Result is an iterable that can be iterated multiple times	Generator object can only be iterated over once