Boosts From The Std Lib

A walk through itertools

by Abdur-Rahmaan Janhangeer

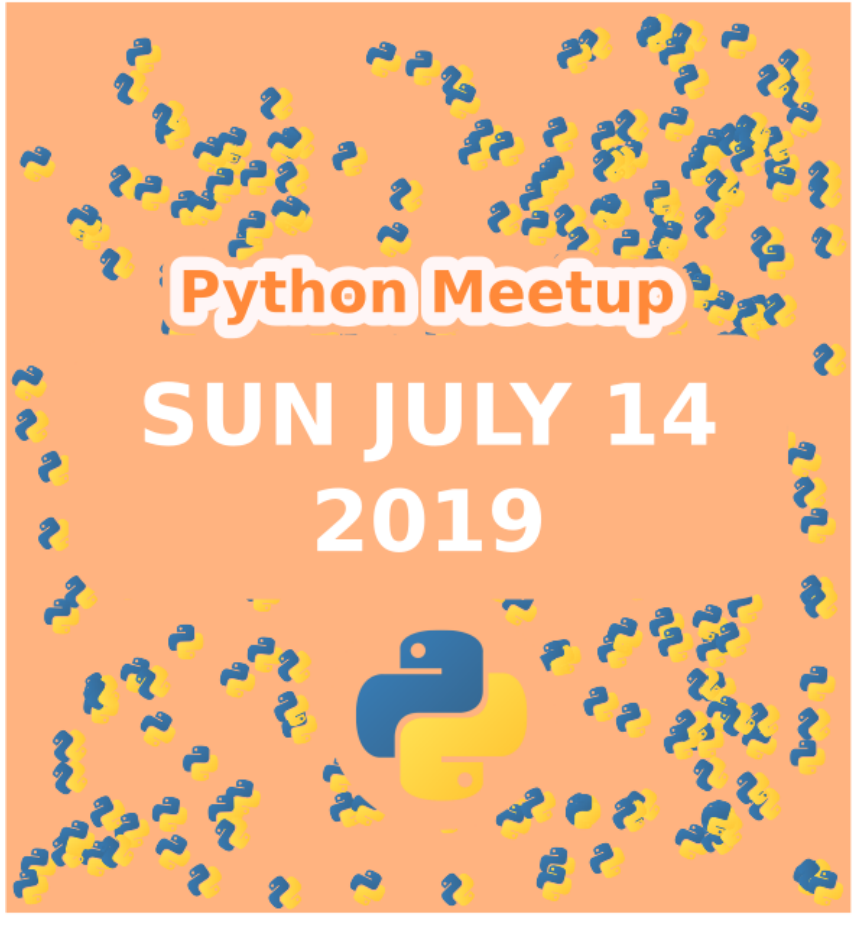
Python Mauritius User Group - PYMUG

github.com/pymug

twitter.com/pymugdotcom

linkedin.com/company/pymug/

pymug.com



A Delightful
Mix of Topics
&
A Distinguished
Personality

PYMUG

Python Mauritius UserGroup

pymug.com

itertools

Functions creating iterators for efficient looping

ITERTOOLS

INFINITE ITERATORS

ENDING ON SHORTEST INPUT

COMBINATRONICS - FOR COMBINATIONS

Infinite Interators

Infinite Iterators

- count( )
- cycle( )
- repeat( )

Infinite Iterators: count

not to be confused with `string.count()`

count

counts to infinity

```
from itertools import *


for i in count(10):
        print(i)
```

gives

```
10
11
12
13
...
```

count

can be stopped:

```
from itertools import *


for i in count(10, step=2):
        print(i)
        if i > 20:
                break
```

count: steps

what do step do? steps skip

```
for i in count(10, step=2):
        print(i)
        if i > 20:
                break
```

gives

```
10
12
14
16
18
20
22
```

count

try with

- negative numbers
- floats

Question

- What is the difference between count and range? Both have steps.
  A: Count counts to infinity

Infinite Iterators: cycle

as the name tells, cycles values

cycle

prints to infinity

```
for a in cycle('ABC'):
        print(a)
```

gives

```
A
B
C
A
B
C
...
```

cycle

we can also access one value at a time

```
alphs = cycle('ABC')

print(next(alphs))
print(next(alphs))
print(next(alphs))
print(next(alphs))
```

gives

```
A
B
C
A
```

Infinite Iterators: repeat

as the name tells, repeats for how much times you want

repeat

```
alphs = repeat('ABC', 4)

for _ in alphs:
        print(_)
```

gives out

```
ABC
ABC
ABC
ABC
```

similarly with next()

```
alphs = repeat('ABC', 4)

print(next(alphs))
print(next(alphs))
print(next(alphs))
print(next(alphs))
```

```
ABC
ABC
ABC
ABC
```

one more next produces an error since we specified 4

Iterators That Stop

Iterators That Stop

- accumulate()
- chain()
- chain.from_iterable()
- compress()
- dropwhile()
- takewhile()
- filterfalse()
- groupby()
- islice()
- starmap()
- tee()
- zip_longest()

Iterators That Stop: accumulate()

returns list of values

```
x = [1, 2, 3, 4, 5]
y = accumulate(x, operator.mul)

print(list(y))
```

gives

```
[1, 2, 6, 24, 120]
```

functools.reduce -> last value

default: operator.add (try removing operator.mul)

Iterators That Stop: chain()

'flatten' list -> turns iterators into single list

```
x = [1, 2, 3]
y = ['a', 'b', 'c']
z = [4, 5, 6]
a = chain(x, y, z)

print(list(a))
```

gives

```
[1, 2, 3, 'a', 'b', 'c', 4, 5, 6]
```

Iterators That Stop: chain.from_iterable()

converts nested list (one level) into list

```
x = [1, 2, 3]
y = ['a', 'b', 'c']
z = [x, y]
a = chain.from_iterable(z)

print(list(a))
```

gives

```
[1, 2, 3, 'a', 'b', 'c']
```

Iterators That Stop: compress()

filters acording to bool list

```
fruits = ['apple', 'banana', 'cauliflower', 'cherry']
checks = [True, True, False, True] # mask
a = compress(fruits, checks)

print(list(a))
```

```
['apple', 'banana', 'cherry']
```

Iterators That Stop: dropwhile()

starts taking values after first false

```
def iseven(x):
    return x % 2 == 0

x = [2, 4, 6, 8, 1, 10, 12, 14]
z = list(dropwhile(iseven, x))

print(z)
```

gives

```
[1, 10, 12, 14]
```

Iterators That Stop: takewhile()

starts taking values until first false

```
def iseven(x):
    return x % 2 == 0

x = [2, 4, 6, 8, 1, 10, 12, 14]
z = list(takewhile(iseven, x))

print(z)
```

gives

```
[2, 4, 6, 8]
```

Iterators That Stop: filterfalse()

returns all that evaluates to false

```
def iseven(x):
    return x % 2 == 0

x = [2, 4, 6, 8, 1, 10, 12, 14]
z = list(filterfalse(iseven, x))

print(z)
```

gives

```
[1]
```

Iterators That Stop: groupby()

groups by the key you tell

```
lst = [("lakesalt", 5, 6), ("thumbsup", 2, 4),
       ("origenes", 2, 5), ("lakesalt", 2, 6)]
groups = groupby(sorted(lst), key=lambda x: x[0])
for key, group in groups:
    print(key, list(group))
```

gives

```
lakesalt [('lakesalt', 2, 6), ('lakesalt', 5, 6)]
origenes [('origenes', 2, 5)]
thumbsup [('thumbsup', 2, 4)]
```

try list(group)[0][0]

Iterators That Stop: islice()

binds iteration by index

```
nums = ['a', 'b', 'c', 'd', 'e', 'f']
s = islice(nums, 3)

print(next(s))
print(next(s))
print(next(s))
print(next(s))
```

gives

```
a
b
c
Traceback (most recent call last):
  File "lab.py", line 10, in <module>
    print(next(s))
StopIteration
```

the fourth gives errors since we bounded it.

islice()

we can also add start index

```
nums = ['a', 'b', 'c', 'd', 'e', 'f']
s = islice(nums, 2, 4)

print(next(s))
print(next(s))
print(next(s))
```

gives

```
c
d
Traceback (most recent call last):
  File "lab.py", line 9, in <module>
    print(next(s))
StopIteration
```

Iterators That Stop: starmap()

Used instead of map() when argument parameters are already grouped in tuples from a single iterable (the data has been âœpre-zippedâ)

```
def mult(x, y, z):
    return x * y * z

nums = [(1, 2, 3), (4, 5, 6), (1, 2, 3)]

for element in starmap(mult, nums):
    print(element)
```

gives

```
6
120
6
```

Iterators That Stop: tee()

Returns independent iterators from a single iterable.

```
fruits = ['apple', 'banana', 'pear']
f1, f2, f3 = tee(fruits, 3)

print(list(f1), list(f2), list(f3))
```

gives

```
['apple', 'banana', 'pear']
['apple', 'banana', 'pear']
['apple', 'banana', 'pear']
```

Along the way: Zip

zips elements together

```
x = zip('abc', 'def')
print(list(x))
```

produces

```
[('a', 'd'), ('b', 'e'), ('c', 'f')]
```

```
for i in zip('abc', 'def'):
    print(i)
```

produces

```
('a', 'd')
('b', 'e')
('c', 'f')
```

Zip

```
for i in zip('abc', 'def', 'ghi'):
    print(i)
```

produces

```
('a', 'd', 'e')
('b', 'e', 'f')
('c', 'f', 'g')
```

Zip

but if not same:

```
x = zip('a', 'defgh')
print(list(x))
```

produces

```
[('a', 'd')]
```

Iterators That Stop: zip_longest()

same thing as previously in zip_longest

```
x = zip_longest('a', 'defgh')
print(list(x))
```

gives

```
[('a', 'd'), (None, 'e'), (None, 'f'), (None, 'g'),
(None, 'h')]
```

zip_longest()

we can also specify a fill value

```
x = zip_longest('a', 'defgh', fillvalue='#')
print(list(x))
```

gives

```
[('a', 'd'), ('#', 'e'), ('#', 'f'), ('#', 'g'), ('#', 'h')]
```

Combinatronics

That's for another time!