Comprehension

Sitian Chen

SP19 Python2

What is Comprehension?

As list comprehension returns list, they consists of brackets containing the expression which needs to be executed for each element along with the for loop to iterate over each element.

Writing shorter and effective codes and execute codes faster

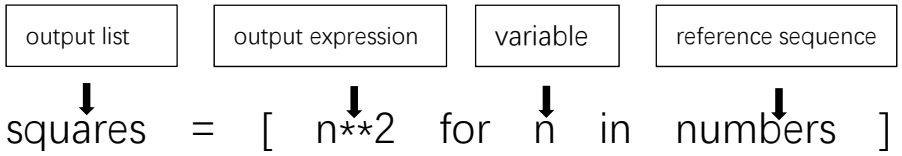
- List comprehensions are more concise to write
- list comprehensions might run much faster than manual for loop statements (often roughly twice as fast)

List + For Loop

```
numbers = [1, 2, 3, 4]
squares = []

for n in numbers:
  squares.append(n**2)

print(squares)  # Output: [1, 4, 9, 16]
```

Comprehension List

```
numbers = [1, 2, 3, 4]
squares = [n**2 for n in numbers]

print(squares)  # Output: [1, 4, 9, 16]
```



Example

E1 **Conditionals in Comprehension**

```
List =[i for i in range(20) if i%2==0]
```

E2 **Nested Conditionals in Comprehension**

```
List =[i for i in range(8) if i%2==0 if i%3==0]
```

E3 **if/else in Comprehension**

```
List= ["Even" if i%2==0 else "Odd" for i in range(10)]
```

E4 **Nested loop in Comprehension**

```
List= [[i*j for j in range(1,11)] for i in range(7,9)]
```

E5 **Dictionary in Comprehension**

List = ['Hello', 'World', 'IBM', 'Apple']

Newlist= [s.lower() for s in List]

Generator

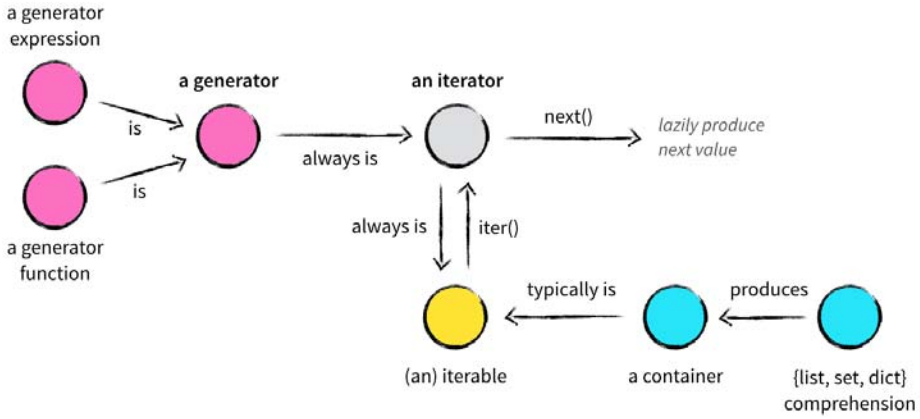
Sitian Chen

SP19 Python2

What is Generator?

Generator functions allow you to declare a function that behaves like an iterator. Unlike normal functions that return a value and exit, generator functions automatically suspend and resume their execution and state around the point of value generation.

- **Generator functions** are coded as normal 'def' statements, but use 'yield' statements to return results one at a time, suspending and resuming their state between each.
- **Generator expressions** are similar to the list comprehensions of the prior section, but they return an object that produces results on demand instead of building a result list.



Function

```
def square_numbers(nums):
    result=[]
    for i in nums:
        result.append(i*i)
    return result

my_nums=square_numbers([1,2,3,4,5])

print (my_nums)
#[1, 4, 9, 16, 25]
```

Generator function

```
def square_numbers(nums):

    for i in nums:
        yield (i*i)


my_nums=square_numbers([1,2,3,4,5])

print (my_nums)
#<generator object square_numbers at 0x0000000002B2B9A8>
```

Generator function - next()

```
def square_numbers(nums):
    for i in nums:
        yield (i*i)

my_nums=square_numbers([1,2,3,4,5])

print (next(my_nums))
#1
print(next(my_nums))
#4
```

```
for num in my_nums:
    print (num)
#1
#4
#9
#16
#25
```

```
my_nums =(i*i for i in [1,2,3,4])
print(next(my_nums))
#1
print(list(my_nums))
#[4, 9, 16]
```

```
import random

def lottery():
    # returns 6 numbers between 1 and 40
    for i in range(6):
        yield random.randint(1, 40)

    # returns a 7th number between 1 and 15
    yield random.randint(1,15)

for random_number in lottery():
    print("And the next number is...", random_number)
```