Video 25 : List Comprehensions / Generators

In this video I'll cover list comprehensions, generator functions, and generator expressions. A list comprehension executes an expression against an iterable, which I covered in the last video.

Note: While they are super powerful, try not to make list comprehensions that are hard to figure out for others. Here are some examples of what you can do with them.

CODE

```
# To multiply 2 times every value with a map we'd do
print(list(map((lambda x: x * 2), range(1, 11))))

# With a list comprehension we'd do
# Note that a list comprehension is surrounded by []
# because it returns a list
print([2 * x for x in range(1, 11)])

# To construct a list of odds using filter we'd
print(list(filter((lambda x: x % 2 != 0), range(1, 11))))

# To do the same with a list comprehension
print([x for x in range(1, 11) if x % 2 != 0])

# A list comprehension can act as map and filter
# on one line
# Generate a list of 50 values and take them to the power
# of 2 and return all that are multiples of 8

print([i ** 2 for i in range(50) if i % 8 == 0])

# You can have multiple for loops as well
# Multiply all values in one list times all values in
# another
print([x * y for x in range(1, 3) for y in range(11, 16)])

# You can put list comprehensions in list comprehensions
# Generate a list of 10 values, multiply them by 2 and
# return multiples of 8
print([x for x in [i * 2 for i in range(10)] if x % 8 == 0])
```

Python Problem for you to Solve

Generate a list of 50 random values between 1 and 1000 and return those that are multiples of 9. You'll have to use a list comprehension in a list comprehension. This is a hard one!

CODE

```
import random

print([x for x in [random.randint(1, 1001) for i in range(50)] if x % 9 == 0])
```

```
# List comprehensions also make it easy to work with
# multidimensional lists

multi_list = [[1, 2, 3],
          [4, 5, 6],
          [7, 8, 9]]

print([col[1] for col in multi_list])

# Get the diagonal by incrementing 0, 0 -> 1, 1 -> 2, 2
print([multi_list[i][i] for i in range(len(multi_list))])
```

Generator Functions

A generator function returns a result generator when called. They can be suspended and resumed during execution of your program to create results over time rather then all at once.

We use generators when we want to big result set, but we don't want to slow down the program by creating it all at one time.

Here I'll create a generator that calculates primes and returns the next prime on command.

CODE

```
def is_prime(num):
    # This for loop cycles through primes from 2 to
    # the value to check
    for i in range(2, num):

        # If any division has no remainder we know it
        # isn't a prime number
        if (num % i) == 0:
            return False
    return True

# This is the generator
def gen_primes(max_number):

    # This for loop cycles through primes from 2 to
    # the maximum value requested
    for num1 in range(2, max_number):

        if is_prime(num1):

            # yield is what makes this a generator
            # When called by next it will return the
            # next result
            yield num1

# Create a reference to the generator
prime = gen_primes(50)

# Call next for each result
```

```
print("Prime :", next(prime))
print("Prime :", next(prime))
print("Prime :", next(prime))
```

Generator Expressions

Generator expressions look just like list comprehensions but they return results one at a time. The are surrounded by parentheses instead of [ ]

CODE

```
double = (x * 2 for x in range(10))

print("Double :", next(double))
print("Double :", next(double))

# You can iterate through all results as well
for num in double:
    print(num)
```

That's it for now. The next video will be a big one and we will focus on threads. We'll learn about sleep(), strftime(), the Threading Module, Creating Threads, activeCount(), enumerate(), Subclassing Threads, run(), start(), is_alive(), getName(), setName(), join(), Synchronizing Threads, acquire(), release(), Lock() and more.