Generator expressions

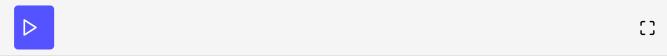
A generator expression is like a generator function without the function.

```
unique_characters = {'E', 'D', 'M', 'O', 'N', 'S', 'R', 'Y'}
gen = (ord(c) for c in unique_characters) #0
print (gen ) #0
#
#3

#69

print (next(gen))
#83

print (tuple(ord(c) for c in unique_characters)) #0
#(69, 83, 89, 77, 68, 79, 78, 82)
```



- ① A generator expression is like an anonymous function that yields values. The expression itself looks like a list comprehension, but it's wrapped in parentheses instead of square brackets.
- ② The generator expression returns... an iterator.
- ③ Calling next(gen) returns the next value from the iterator.
- ④ If you like, you can iterate through all the possible values and return a tuple, list, or set, by passing the generator expression to tuple(), list(), or set(). In these cases, you don't need an extra set of parentheses just pass the "bare" expression ord(c) for c in unique_characters to the tuple() function, and Python figures out that it's a generator expression.

Using a generator expression instead of a list comprehension can save both CPU and RAM. If you're building an list just to throw it away (e.g. passing it to tuple() or set()), use a generator expression instead!

Here's another way to accomplish the same thing, using a generator function:

```
unique_characters = {'E', 'D', 'M', 'O', 'N', 'S', 'R', 'Y'}

def ord_map(a_string):
    for c in a_string:
        yield ord(c)

gen = ord_map(unique_characters)
```

The generator expression is more compact but functionally equivalent.