



PYTHON DATA SCIENCE TOOLBOX II

List comprehensions



Populate a list with a for loop

```
In [1]: nums = [12, 8, 21, 3, 16]

In [2]: new_nums = []

In [3]: for num in nums:
...:     new_nums.append(num + 1)

In [4]: print(new_nums)
[13, 9, 22, 4, 17]
```



A list comprehension

```
In [1]: nums = [12, 8, 21, 3, 16]
```

```
In [2]: new_nums = [num + 1 for num in nums]
```

```
In [3]: print(new_nums)  
[13, 9, 22, 4, 17]
```



For loop and list comprehension syntax

```
In [1]: new_nums = [num + 1 for num in nums]
```

```
In [2]: for num in nums:  
...:     new_nums.append(num + 1)
```

```
In [3]: print(new_nums)  
[13, 9, 22, 4, 17]
```



List comprehension with range()

```
In [1]: result = [num for num in range(11)]
```

```
In [2]: print(result)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```



List comprehensions

- Collapse for loops for building lists into a single line
- Components
 - Iterable
 - Iterator variable (represent members of iterable)
 - Output expression



Nested loops (1)

```
In [1]: pairs_1 = []

In [2]: for num1 in range(0, 2):
...:     for num2 in range(6, 8):
...:         pairs_1.append(num1, num2)

In [3]: print(pairs_1)
[(0, 6), (0, 7), (1, 6), (1, 7)]
```

- How to do this with a list comprehension?



Nested loops (2)

```
In [1]: pairs_2 = [(num1, num2) for num1 in range(0, 2) for num2  
in range(6, 8)]
```

```
In [2]: print(pairs_2)  
[(0, 6), (0, 7), (1, 6), (1, 7)]
```

- Tradeoff: readability



PYTHON DATA SCIENCE TOOLBOX II

Let's practice!



PYTHON DATA SCIENCE TOOLBOX II

Advanced comprehensions



Conditionals in comprehensions

- Conditionals on the iterable

```
In [1]: [num ** 2 for num in range(10) if num % 2 == 0]
Out[1]: [0, 4, 16, 36, 64]
```

- Python documentation on the % operator:

The `%` (modulo) operator yields the remainder from the division of the first argument by the second.

```
In [1]: 5 % 2
Out[1]: 1
```

```
In [2]: 6 % 2
Out[2]: 0
```



Conditionals in comprehensions

- Conditionals on the output expression

```
In [2]: [num ** 2 if num % 2 == 0 else 0 for num in range(10)]  
Out[2]: [0, 0, 4, 0, 16, 0, 36, 0, 64, 0]
```



Dict comprehensions

- Create dictionaries
- Use curly braces `{}` instead of brackets `[]`

```
In [1]: pos_neg = {num: -num for num in range(9)}
```

```
In [2]: print(pos_neg)
```

```
{0: 0, 1: -1, 2: -2, 3: -3, 4: -4, 5: -5, 6: -6, 7: -7, 8: -8}
```

```
In [3]: print(type(pos_neg))
```

```
<class 'dict'>
```



PYTHON DATA SCIENCE TOOLBOX II

Let's practice!



PYTHON DATA SCIENCE TOOLBOX II

Introduction to generators



Generator expressions

- Recall list comprehension

```
In [1]: [2 * num for num in range(10)]  
Out[1]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

- Use `()` instead of `[]`

```
In [2]: (2 * num for num in range(10))  
Out[2]: <generator object <genexpr> at 0x1046bf888>
```




List comprehensions vs. generators

- List comprehension - returns a list
- Generators - returns a generator object
- Both can be iterated over



Printing values from generators (1)

```
In [1]: result = (num for num in range(6))
```

```
In [2]: for num in result:  
.....:     print(num)
```

```
0  
1  
2  
3  
4  
5
```

```
In [1]: result = (num for num in range(6))
```

```
In [2]: print(list(result))
```

```
[0, 1, 2, 3, 4, 5]
```



Printing values from generators (2)

```
In [1]: result = (num for num in range(6))
```

```
In [2]: print(next(result))
```

```
0
```

Lazy evaluation

```
In [3]: print(next(result))
```

```
1
```

```
In [4]: print(next(result))
```

```
2
```

```
In [5]: print(next(result))
```

```
3
```

```
In [6]: print(next(result))
```

```
4
```



Generators vs list comprehensions

IPython Shell

```
In [1]: [num for num in range(10**1000000)]
```

```
In [2]: |
```

IPython Shell

```
In [1]: [num for num in range(10**1000000)]
```

```
In [2]: |
```

Your session has been disconnected.
The performed operation was too
resource-intensive.

Restart Session

Generators vs list comprehensions

IPython Shell

```
In [1]: (num for num in range(10**1000000))
```

```
Out[1]: <generator object <genexpr> at 0x7f8aca2601f8>
```

```
In [2]:
```



Conditionals in generator expressions

```
In [1]: even_nums = (num for num in range(10) if num % 2 == 0)
```

```
In [2]: print(list(even_nums))  
[0, 2, 4, 6, 8]
```



Generator functions

- Produces generator objects when called
- Defined like a regular function - `def`
- Yields a sequence of values instead of returning a single value
- Generates a value with `yield` keyword



Build a generator function

sequence.py

```
def num_sequence(n):  
    """Generate values from 0 to n."""  
    i = 0  
    while i < n:  
        yield i  
        i += 1
```




Use a generator function

```
In [1]: result = num_sequence(5)
```

```
In [2]: print(type(result))  
<class 'generator'>
```

```
In [3]: for item in result:  
.....:     print(item)
```

```
0
```

```
1
```

```
2
```

```
3
```

```
4
```



PYTHON DATA SCIENCE TOOLBOX II

Let's practice!



PYTHON DATA SCIENCE TOOLBOX II

Wrap-up: comprehensions



Re-cap: list comprehensions

- Basic

```
[output expression for iterator variable in iterable]
```

- Advanced

```
[output expression + conditional on output for iterator variable  
in iterable + conditional on iterable]
```



PYTHON DATA SCIENCE TOOLBOX II

Let's practice!