

# Python Fundamentals

Comprehensions, Iterables and Generators

Robert Smallshire  
🐦 @robsmallshire  
rob@sixty-north.com



Presenter

Austin Bingham  
🐦 @austin\_bingham  
austin@sixty-north.com



pluralsight  
hardcore developer training





# python Comprehensions

## Types of comprehensions

- list comprehensions
- set comprehensions
- dictionary comprehensions

## Style!

- declarative
- functional

- readable
- expressive
- effective





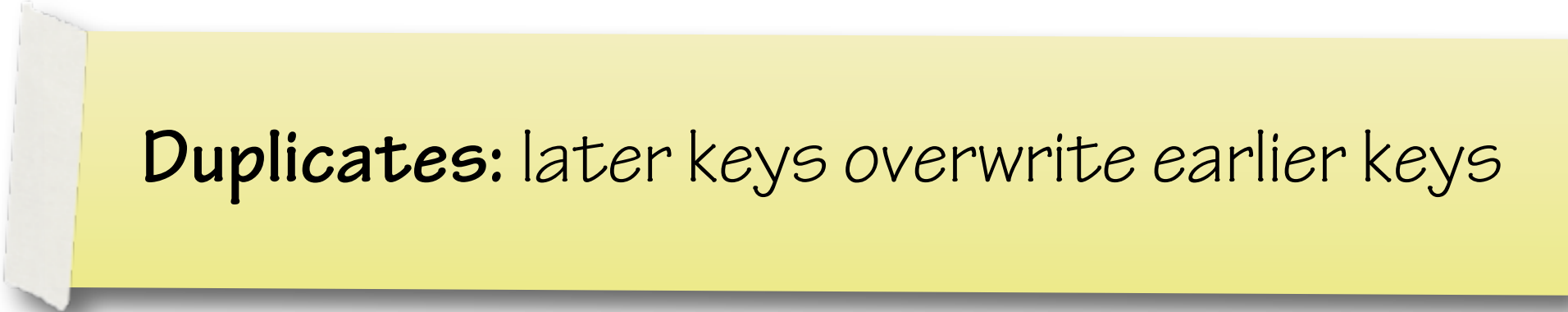
```
[ expr(item) for item in iterable ]
```



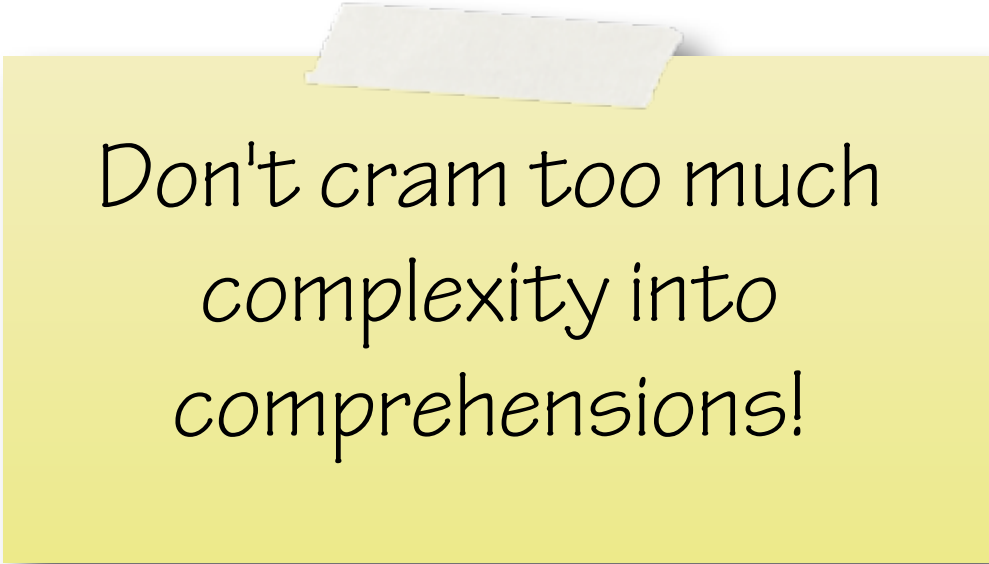
```
{ expr(item) for item in iterable }
```



```
{ key_expr:value_expr for item in iterable }
```



**Duplicates:** later keys overwrite earlier keys



Don't cram too much  
complexity into  
comprehensions!

```
>>> import os
>>> import glob
>>> file_sizes = {os.path.realpath(p): os.stat(p).st_size
...               for p in glob.glob('*.py')}
>>> pp(file_sizes)
{'/Users/pyfund/examples/exceptional.py': 400,
 '/Users/pyfund/examples/keypress.py': 778,
 '/Users/pyfund/examples/scopes.py': 133,
 '/Users/pyfund/examples/words.py': 1185}
```

Filtering works with:

- list comprehensions
- set comprehensions
- dictionary comprehensions



optional **filtering clause**

[ expr(item) for item in iterable if predicate(item) ]



Moment of Zen

Simple is better  
than complex

Code is written once  
But read over and over  
Fewer is clearer



# python **Iteration protocols**



# python Iteration protocols

## *Iterable protocol*

Iterable objects can be passed to the built-in `iter()` function to get an iterator.

```
iterator = iter(iterable)
```

## *Iterator protocol*

Iterator objects can be passed to the built-in `next()` function to fetch the next item.

```
item = next(iterator)
```

# Stateful generators

- Generators resume execution
- Can maintain state in local variables
- Complex control flow
- Lazy evaluation

```
def take(count, iterable):  
    "Take first count elements"  
    counter = 0  
    for item in iterable:  
        if counter == count:  
            return  
        counter += 1  
        yield item
```





# Laziness and the Infinite

- Just in Time Computation
- Infinite (or large) sequences
  - sensor readings
  - mathematical series
  - massive files

# Generator comprehensions

- Similar syntax to list comprehensions
- Create a generator object
- Concise
- Lazy evaluation



parentheses

(expr(item) for item in iterable)

# "Batteries Included" Iteration Tools



**Batteries Included**





# Comprehensions, Generators & Iterables Summary

## ■ Comprehensions

- Comprehensions are a concise syntax for describing lists, sets and dictionaries.
- Comprehensions operate on an iterable source object and apply an optional predicate filter and a mandatory expression, both of which are usually in terms of the current item.
- Iterables are objects over which we can iterate item by item.
- We retrieve an iterator from an iterable object using the built-in `iter()` function.
- Iterators produce items one-by-one-from the underlying iterable series each time they are passed to the built-in `next()` function



python

# Comprehensions, Generators & Iterables Summary

## ■ Generators

- Generator functions allow us to describe series using imperative code.
- Generator functions contain at least one use of the `yield` keyword.
- Generators are iterators. When advanced with `next()` the generator starts or resumes execution up to and including the next `yield`.
- Each call to a generator function creates a new generator object.
- Generators can maintain explicit state in local variables between iterations.
- Generators are lazy, and so can model infinite series of data.
- Generator expressions have a similar syntactic form to list comprehensions and allow for a more declarative and concise way of creating generator objects.



# python Getting Started – Summary

- **Iteration tools**

- Built-ins such as

- `sum()`
    - `any()`
    - `zip()`
    - `all()`
    - `min()`
    - `max()`
    - `enumerate()`

- Standard library `itertools` module

- `chain()`
    - `islice()`
    - `count()`
    - many more!