The `inspect` Module

Austin Bingham
COFOUNDER - SIXTY NORTH

@austin_bingham

Robert Smallshire
COFOUNDER - SIXTY NORTH

@robsmallshire

```python
from itertools import chain


class Batch:
    def __init__(self, iterables=()):
        self._iterables = list(iterables)

    def append(self, iterable):
        self._iterables.append(iterable)

    def __iter__(self):
        return chain(*self._iterables)
```

```
    return Signature.from_callable(obj, follow_wrapped=follow_wrapped)
  File "/Users/sixty-north/.pyenv/versions/3.8.0/lib/python3.8/inspect.py", line
2842, in from_callable
    return _signature_from_callable(obj, sigcls=cls,
  File "/Users/sixty-north/.pyenv/versions/3.8.0/lib/python3.8/inspect.py", line
2296, in _signature_from_callable
    return _signature_from_builtin(sigcls, obj,
  File "/Users/sixty-north/.pyenv/versions/3.8.0/lib/python3.8/inspect.py", line
2107, in _signature_from_builtin
    raise ValueError("no signature found for builtin {!r}".format(func))
ValueError: no signature found for builtin <built-in function iter>
>>>
```

`chain()` is a class, not a function

It looks like a function because it violates the PEP8 naming conventions

Calling `chain()` is actually invoking a constructor

Any class defined in a module or imported into a module is in that module's namespace.

Functions implemented in C (or other languages) may be missing metadata and cause `signature()` to fail.

PEP 484 -- Type Hints

PEP:	484
Title:	Type Hints
Author:	Guido van Rossum <guido at python.org>, Jukka Lehtosalo <jukka.lehtosalo at iki.fi>, Łukasz Langa <lukasz at python.org>
BDFL-Delegate:	Mark Shannon
Discussions-To:	Python-Dev <python-dev at python.org>
Status:	Provisional
Type:	Standards Track
Created:	29-Sep-2014
Python-Version:	3.5
Post-History:	16-Jan-2015,20-Mar-2015,17-Apr-2015,20-May-2015,22-

`Signature` stores information on type annotations

Type annotations were introduced in PEP 484

They express the expected types for parameters and return values

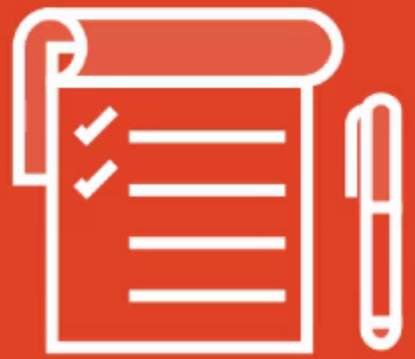
External tools use them to type-check Python code

Introspecting Type Annotations

```
>>> def num_vowels(text: str) -> int:
...     return sum(1 if c.lower() in 'aeiou' else 0
...                for c in text)
...
>>> import inspect
>>> sig = inspect.signature(num_vowels)
>>> sig.parameters['text']
<Parameter "text: str">
>>> sig.parameters['text'].annotation
<class 'str'>
>>> sig
<Signature (text: str) -> int>
>>> sig.return_annotation
<class 'int'>
>>> num_vowels.__annotations__
{'text': <class 'str'>, 'return': <class 'int'>}
>>>
```

`inspect` contains many more tools for introspection.

Summary



`inspect` provides support for advanced introspection

`inspect.ismodule()` determines if an object is a module

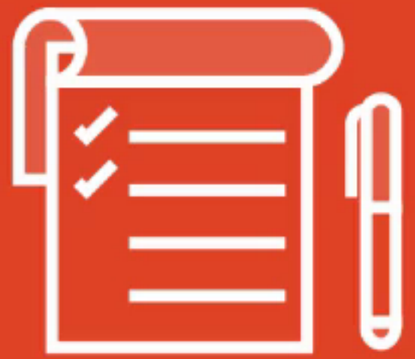
`inspect.getmembers()` returns the members of an object

It can accept a predicate for filtering its results

`inspect.signature()` provides information on a function's signature

It will throw a `ValueError` if the function is missing metadata

Summary



Names imported into a module become members of that module

`itertools.chain()` is a class, not a function