PEP 492

Coroutines with async and await syntax

async/await

```
async def read_data(db):
    data = await db.fetch('SELECT ...')

@asyncio.coroutine
def read_data_old(db):
    data = yield from db.fetch('SELECT ...')
```

async def aka async function

- await
- async with
- async for
- ~~yield~~
- ~~yield from~~

async for

```
async for i in conn.execute(tbl.select()):
    process(i)
```

Asynchronous Iterator

```
async def __aiter__(self):
    return self

async def __anext__(self):
    ret = await self.fetchone()
    if ret is not None:
        return ret
    else:
        raise StopAsyncIteration
```

Async generator with yields

async yield is out of PEP 492 scope

async with

```
async with conn.begin() as tr:
    await conn.execute(tbl.delete()
                               .where(tbl.c.id=234))
```

Context manager

```
async def __aenter__(self):
    return self

async def __aexit__(self,
                    exc_type,
                    exc_val,
                    exc_tb):
    await self.close()
```

100% backward compatibility

```
async def f():
    await asyncio.sleep(1.0)

loop = asyncio.get_event_loop()
loop.run_until_complete(f())
```