Discover asyncio event loop
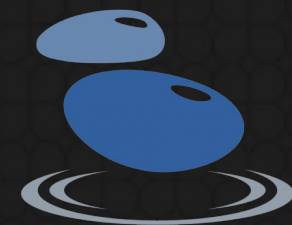
Pycon 2014, Lyon

eNovance
Multi-Cloud Experts

Victor Stinner
victor.stinner@gmail.com

Victor Stinner

- Python core developer since 2010
- github.com/haypo/
- bitbucket.org/haypo/
- Working for **eNovance** on OpenStack

eNovance
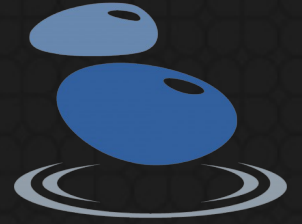Multi-Cloud Experts

python™

Disclamer

- Simplified code snippets close to asyncio, but different
- No error handling nor optimization
- (asyncio handles errors and is optimized)
- Code written for Python 3

python™

Agenda

1. Callbacks
2. Timers
3. Selectors
4. Generator
5. Coroutine and BaseTask
6. Future and Task

Callbacks

```python
class CallbackEventLoop:
    def __init__(self):
        self.callbacks = []

    def call_soon(self, func):
        self.callbacks.append(func)

    def execute_callbacks(self):
        callbacks = self.callbacks
        self.callbacks = []
        for cb in callbacks:
            cb()
```

Callbacks

Code

loop.call_soon(hello_world)

Callbacks

hello_world()

Output

python

Callbacks

Code

```
loop.call_soon(hello_world)
loop.execute_callbacks()
```

Callbacks

hello_world()

Output

Hello World!

Callbacks

Code
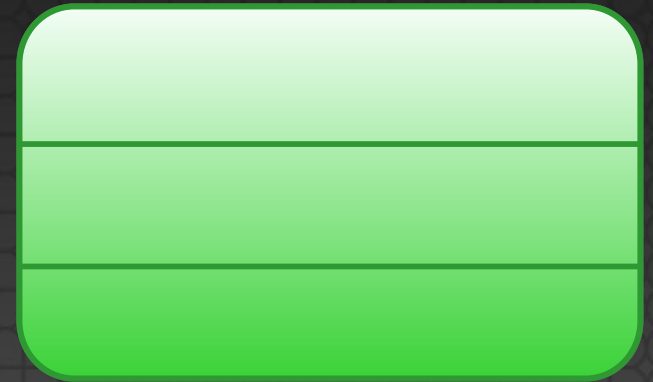
```
loop.call_soon(hello_world)
loop.execute_callbacks()
```

Callbacks

Output

Hello World!

python

Timers

```python
class TimerEventLoop(CallbackEventLoop):

    def __init__(self):
        super().__init__()
        self.timers = []

    def call_at(self, when, func):
        timer = (when, func)
        self.timers.append(timer)

    ...
```
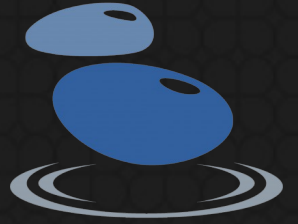
Timers

```python
class TimerEventLoop(CallbackEventLoop):
    ...
    def execute_timers(self):
        now = time.time()
        new_timers = []
        for when, func in self.timers:
            if when <= now:
                self.call_soon(func)
            else:
                new_timers.append((when, func))
        self.timers = new_timers

        self.execute_callbacks()
```

Timers

Code

loop.call_at(1, hello_world)

Output

Timers

(1, hello_world)

Callbacks

Timers

Code

```
loop.call_at(1, hello_world)
loop.call_at(5, exit)
```

Timers

(1, hello_world)

(5, exit)

Output

Callbacks

Timers

Code

```
loop.call_at(1, hello_world)
loop.call_at(5, exit)
loop.call_at(2, good_bye)
```
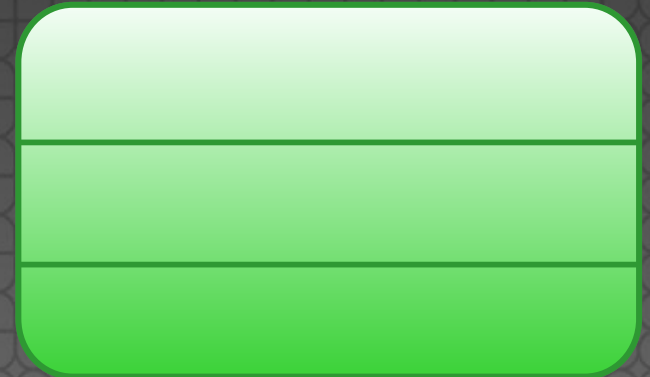
Output

Timers

```
(1, hello_world)
(5, exit)
(2, good_bye)
```
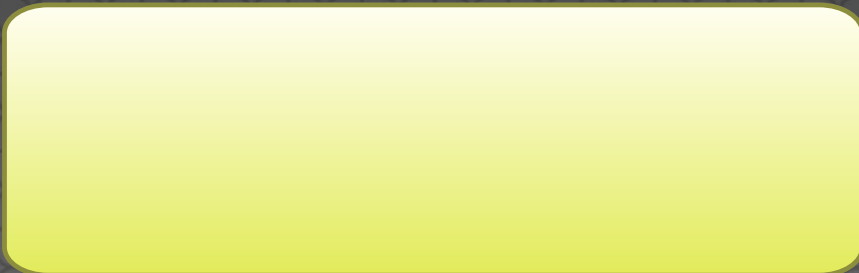
Callbacks

Timers

Code

```
loop.call_at(1, hello_world)
loop.call_at(5, exit)
loop.call_at(2, good_bye)
loop.execute_timers()
```

Output

Timers

```
(1, hello_world)
(5, exit)
(2, good_bye)
```

Callbacks

```
hello_world()
good_bye()
```

python

Timers

Code

```
loop.call_at(1, hello_world)
loop.call_at(5, exit)
loop.call_at(2, good_bye)
loop.execute_timers()
```

Output

Hello World!
Good bye.

Timers

(5, exit)
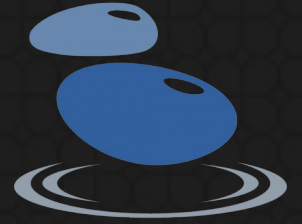
Callbacks

hello_world()

good_bye()

Timers

Code

```
loop.call_at(1, hello_world)
loop.call_at(5, exit)
loop.call_at(2, good_bye)
loop.execute_timers()
```

Timers

(5, exit)

Callbacks

Output

Hello World!
Good bye.

python™

Selector

```python
from selectors import DefaultSelector

class SelectorEventLoop(TimerEventLoop):

    def __init__(self):
        super().__init__()
        self.selector = DefaultSelector()

    def add_reader(self, sock, func):
        self.selector.register(sock,
                        selectors.EVENT_READ,
                        data=func)
    ...
```

python

Selector

```python
class SelectorEventLoop(TimerEventLoop):
    ...
    def select(self):
        timeout = self.compute_timeout()

        events = self.selector.select(timeout)
        for key, mask in events:
            func = key.data
            self.call_soon(func)

        self.execute_timers()
```

Selector

```python
class SelectorEventLoop(TimerEventLoop):
    ...
    def compute_timeout(self):
        if self.callbacks:
            # already something to do
            return 0
        elif self.timers:
            next_timer = min(self.timers)[0]
            timeout = next_timer - time.time()
            return max(timeout, 0.0)
        else:
            # blocking call
            return None
```

Selector

Code

```
s, c = socket.socketpair()
loop.add_reader(s, reader)
```

Selector

s: idle

Output

Callbacks

Selector

Code

```
s, c = socket.socketpair()
loop.add_reader(s, reader)
c.send(b'abc')
```

Selector

s: read event

Callbacks

Output

Selector

Code

```
s, c = socket.socketpair()
loop.add_reader(s, reader)
c.send(b'abc')
loop.select()
```
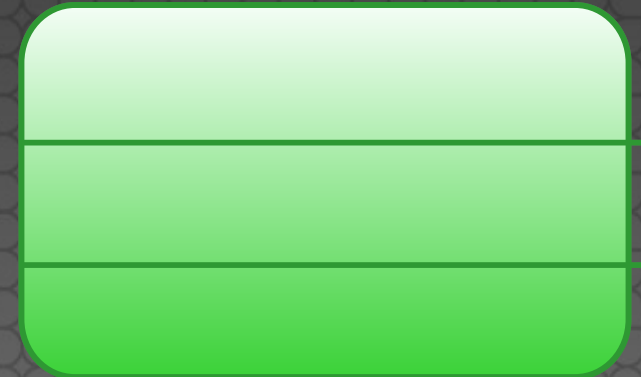
Output

Selector

s: read event

Callbacks

reader()

Selector

Code

```
s, c = socket.socketpair()
loop.add_reader(s, reader)
c.send(b'abc')
loop.select()
```

Output

Received: b'abc'

Selector

s: idle

Callbacks

reader()

Selector

Code

```
s, c = socket.socketpair()
loop.add_reader(s, reader)
c.send(b'abc')
loop.select()
```

Output

Received: b'abc'
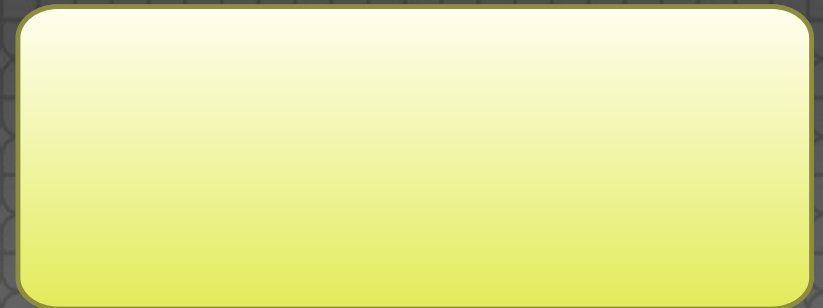
Selector

s: idle

Callbacks

Generator

Code

```
gen = producer()
```

producer() generator

→ yield "start"
return "stop"

Output

Generator

Code

```
gen = producer()
print(next(gen))
```
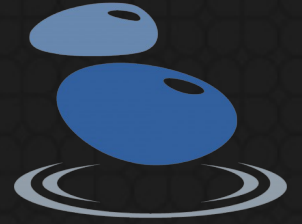
producer() generator

```
yield "start"
return "stop"
```

Output

```
start
```

Generator

Code

```
gen = producer()
print(next(gen))
try:
    next(gen)
except StopIteration as e:
    print(e.value)
```

producer() generator

```
yield "start"
return "stop"
```

Output

```
start
stop
```

eNovance
Multi-Cloud Experts

python™

Coroutine

```
def my_coroutine(future):
    res = yield from future
    return res
```

yield from: read the PEP 380

BaseTask

```python
class BaseTask:
    def __init__(self, coro):
        self.coro = coro

    def step(self):
        try:
            next(self.coro)
        except StopIteration:
            pass
```

BaseTask

Code

```
coro = test_coro()
task = BaseTask(coro)
```

test_coro() coroutine

```
print("begin")
yield from ["hack"]
print("end")
```

Output

BaseTask

Code

```
coro = test_coro()
task = BaseTask(coro)
task.step()
```

test_coro() coroutine

```
print("begin")
yield from ["hack"]
print("end")
```

Output

```
begin
```

BaseTask

Code

```
coro = test_coro()
task = BaseTask(coro)
task.step()
task.step()
```

test_coro() coroutine

```
print("begin")
yield from ["hack"]
print("end")
```

Output

```
begin
end
```

Future

```python
class Future:

    def __init__(self, loop):
        self.loop = loop
        self.callbacks = []
        self._result = None

    def add_done_callback(self, func):
        self.callbacks.append(func)

    def result(self):
        return self._result
...
```

Future

```python
class Future:
    ...
    def set_result(self, result):
        self._result = result

        for func in self.callbacks:
            self.loop.call_soon(func)

    def __iter__(self):
        # used by "yield from future"
        yield self
```
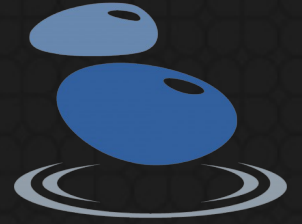
Task

```python
class Task:

    def __init__(self, coro, loop):
        self.coro = coro
        loop.call_soon(self.step)
    ...
```

Task

```python
class Task:
    ...
    def step(self):
        try:
            result = next(self.coro)
        except StopIteration:
            return

        if isinstance(result, Future):
            result.add_done_callback(self.step)
```
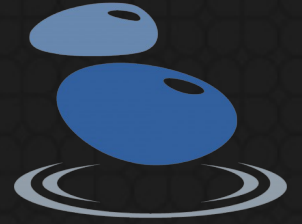
Sleep

```python
def sleep(delay, loop):
    fut = Future(loop)
    cb = functools.partial(fut.set_result,
                           None)
    loop.call_later(delay, cb)
    yield from fut
```

Questions?

http://docs.python.org/dev/library/asyncio.html

http://github.com/haypo/
http://bitbucket.org/haypo/

Victor Stinner
victor.stinner@gmail.com

Thanks David Malcom
for the LibreOffice model

http://dmalcolm.livejournal.com/