Python is not so fast

Python up to 20x slower than C++
Python is not so fast
Multi-threading or processing

Process

Thread #1

Thread #2

Time

Process

Thread #1

Thread #2

Time
Parallel Processing
Event Loop

1. Open window
2. Define draw callback
3. Define other callbacks
4. Main loop:
   - Check for event
   - Call callback

      Draw func
      Keyboard func
      Mouse func

...
Event Loop

- Pause Code
- Schedule Code For Later
- Cancel Code
- Asynchronous Code
- Switches on Waiting Code
- Shares like a CPU
import asyncio
import string

async def useless_demo_function(action):
    print("{0} is starting".format(action))
    await asyncio.sleep(1)
    print("{0} is done".format(action))

loop = asyncio.get_event_loop()
actions = [loop.create_task(useless_demo_function(x))
          for x in string.ascii_lowercase[:5]]
wait_actions = asyncio.wait(actions)
res = loop.run_until_complete(wait_actions)
print("All of them are complete. We can move on")
Coroutines

a is starting
b is starting
c is starting
d is starting
e is starting
a is done
b is done
c is done
d is done
e is done
All of them are complete. We can move on
Coroutines

```python
actions = [loop.create_task(useless_demo_function('a', 1)),
           loop.create_task(useless_demo_function('b', 0.2)),
           loop.create_task(useless_demo_function('c', 1)),
           loop.create_task(useless_demo_function('d', 0.1)),
           loop.create_task(useless_demo_function('e', 1))]
```
async def useless_demo_function(action, sleep_time):
    print("{0} is starting".format(action))
    await asyncio.sleep(sleep_time)
    print("{0} is done".format(action))

loop = asyncio.get_event_loop()
wait_actions = asyncio.wait(actions)
res = loop.run_until_complete(wait_actions)
print("All of them are complete. We can move on")
Coroutines

a is starting
b is starting
c is starting
d is starting
e is starting
d is done
b is done
a is done
c is done
e is done
Coroutines

```python
import asyncio
import time

async def useless_demo_function(action, sleep_time):
    print("{0} is starting".format(action))
    await time.sleep(sleep_time)
    print("{0} is done".format(action))

loop = asyncio.get_event_loop()
actions = [loop.create_task(useless_demo_function('a', 1))]
wait_actions = asyncio.wait(actions)
loop.run_until_complete(wait_actions)
```
Coroutines

```python
a is starting
({"<Task finished coro=<useless_demo_function() done, defined at <stdin>:1> exception=TypeError("object NoneType can't be used in 'await' expression",>, set()```
Coroutines

```python
>>> exit()
Task exception was never retrieved
future: <Task finished coro=<useless_demo_function() done, defined at <stdin>:1> exception=TypeError("object NoneType can't be used in 'await' expression"),>
Traceback (most recent call last):
  File "<stdin>", line 3, in useless_demo_function
TypeError: object NoneType can't be used in 'await' expression
```
Coroutines

@coroutine
def sleep(delay, result=None, *, loop=None):
    """Coroutine that completes after a given time (in seconds)."""
    if delay == 0:
        yield
        return result
    if loop is None:
        loop = events.get_event_loop()
    future = loop.create_future()
    h = future._loop.call_later(delay,
                                  futures._set_result_unless_cancelled,
                                  future, result)
    try:
        return (yield from future)
    finally:
        h.cancel()
Non Blocking Code

def asyncio create_connection(...):
    ...
    sock = socket.socket(family=family, type=type, proto=proto)
    sock.setblocking(False)
    ...
    yield from self.sock_connect(sock, address)
Web Scraper

```python
import asyncio
import aiohttp

FIND_WORD = "the"
pages = [
    "http://lucumr.pocoo.org/",
    "http://planetpython.org/",
    "https://doughellmann.com/blog/",
    "http://sayspy.blogspot.co.il/",
    "https://www.blog.pythonlibrary.org/"
]
async def get_data(session, URL):
    with aiohttp.Timeout(10):
        print("{} starting".format(URL))
        async with session.get(URL) as response:
            print("{} got response".format(URL))
            content = await response.text()
            print("{} got content".format(URL))
            return [URL, content.count(FIND_WORD)]
async def get_urls(session, sites, loop):
    results = await \n    asyncio.wait([loop.create_task(get_data(session,url))
        for url in sites])
    return results
Web Scraper

```python
loop = asyncio.get_event_loop()
with aiohttp.ClientSession(loop=loop) as session:
    res = loop.run_until_complete(get_urls(session, pages, loop))
print()
print("Results:")
for data in res[0]:
    print(data.result())
```
Web Scraper

http://lucumr.pocoo.org/ starting
http://planetpython.org/ starting
https://doughellmann.com/blog/ starting
http://sayspy.blogspot.co.il/ starting
https://www.blog.pythonlibrary.org/ starting
http://planetpython.org/ got response
http://planetpython.org/ got content
https://doughellmann.com/blog/ got response
https://doughellmann.com/blog/ got content
http://lucumr.pocoo.org/ got response
http://lucumr.pocoo.org/ got content
http://sayspy.blogspot.co.il/ got response
http://sayspy.blogspot.co.il/ got content
https://www.blog.pythonlibrary.org/ got response
https://www.blog.pythonlibrary.org/ got content
Web Scraper

Results:
['http://sayspy.blogspot.co.il/’, 399]
['http://planetpython.org/’, 625]
['https://www.blog.pythonlibrary.org/’, 214]
['http://lucumr.pocoo.org/’, 12]
['https://doughellmann.com/blog/’, 75]
Exception handling

```python
pages = [
    "http://lucumr.pocoo.org/",
    "http://planetpython.org/",
    "https://doughellmann.com123/blog/",
    "http://sayspy.blogspot.co.il/",
    "https://www.blog.pythonlibrary.org/"]

async def get_data(session, URL):
    try:
        with aiohttp.Timeout(10):
            async with session.get(URL) as response:
                content = await response.text()
                return [URL, content.count(FIND_WORD)]
    except BaseException as e:
        return [URL, "Error: {0}".format(e)]
```
Exception handling

Results:
['http://sayspy.blogspot.co.il/', 399]
['http://lucumr.pocoo.org/', 12]
['https://www.blog.pythonlibrary.org/', 214]
['http://planetpython.org/', 634]
['https://doughellmann.com123/blog/', 'Error: [Errno -2] Cannot connect to host doughellmann.com123:443 ssl:True [Name or service not known]']
Exception handling

```python
loop = asyncio.get_event_loop()
with aiohttp.ClientSession(loop=loop) as session:
    res = loop.run_until_complete(get_urls(session, pages, loop))
    print()
    print ("Results:"),
    for data in res[0]:
        try:
            print (data.result())
        except BaseException as e:
            print(e)
```
Exception handling

Results:
['http://sayspy.blogspot.co.il/', 399]
['http://lucumr.pocoo.org/', 12]
['https://www.blog.pythonlibrary.org/', 214]
['http://planetpython.org/', 634]
[Errno -2] Cannot connect to host doughellmann.com123:443
ssl:True [Name or service not known]
Using Blocking Functions

class LoggingMeta(type):
    def __new__(cls, name, bases, attrs):
        global _class_elements
        global SYSTEM_NAME
        logstash_parms = {}
        for item, attr_val in attrs.items():
            if isinstance(attr_val, types.FunctionType):
                attrs[item] = log_wrapper(attr_val)
        ...
        #if logstash_parms:
        init_logger()
        return super(LoggingMeta, cls).__new__(cls, name, bases, attrs)
Using Blocking Functions

# decorator function to log function usage.
def wrapper(func, *args, **kwargs):
    if _logging:
        return func(*args, **kwargs)
    else:
        loop = asyncio.get_event_loop()
        tasks = [
            loop.run_in_executor(
                None, generate_log, func, *args, **kwargs)
        ]
        loop.call_soon(asyncio.wait(tasks))
        return do_func(func, *args, **kwargs)
THANK YOU

plus.google.com/+RedHat
linkedin.com/company/red-hat
youtube.com/user/RedHatVideos
facebook.com/redhatinc
twitter.com/RedHatNews