SuperFastPython.com Cheat Sheet for ProcessPoolExecutor

Why ProcessPoolExecutor?

Execute ad hoc functions that perform CPU-bound tasks asynchronously in new child processes, such as compute tasks or mathematical operations.

Create, Configure, Use

Imports
```
from concurrent.futures import *
```

Create, default config
```
pool = ProcessPoolExecutor()
```

Config number of workers
```
pool = ProcessPoolExecutor(max_workers=10)
```

Config multiprocessing context
```
ctx = multiprocessing.get_context('spawn')
pool = ProcessPoolExecutor(mp_context=ctx)
```

Config worker initializer function
```
pool = ProcessPoolExecutor(initializer=init, initargs=(a1, a2))
```

Shutdown and wait, not cancel tasks
```
pool.shutdown()
```

Shutdown no wait, not cancel tasks
```
pool.shutdown(wait=False)
```

Shutdown and wait, cancel tasks
```
pool.shutdown(cancel_futures=True)
```

Shutdown no wait, cancel tasks
```
pool.shutdown(wait=False, cancel_futures=True)
```

Context manager, shutdown automatically
```
with ProcessPoolExecutor() as e:
    # ...
```

Issue Async Tasks to the Pool

Issue one task asynchronously
```
future = pool.submit(task)
```

Issue one task with arguments
```
future = pool.submit(task, a1, a2)
```

Issue many tasks, collect Futures
```
futures = [pool.submit(task) for _ in range(5)]
```

Issue many tasks, iterate results in order
```
for res in pool.map(task, range(10)):
    # ...
```

Issue many tasks, iterate results with timeout
```
try:
    for r in pool.map(task, range(10), timeout=0.5):
    # ...
except TimeoutError as e:
    # ...
```

Issue many tasks in chunks, iterate results
```
for res in pool.map(task, range(10), chunksize=10):
    # ...
```

Wait for all tasks via futures
```
wait(futures)
```

Wait with a timeout in seconds
```
wait(futures, timeout=0.5)
```

Wait for first task
```
wait(futures, FIRST_COMPLETED)
```

Wait for for first task failure
```
wait(futures, FIRST_EXCEPTION)
```

Iterate futures in order completed
```
for fut in as_completed(futures):
    # ...
```

Use Futures (handles on async tasks)

Get result (blocking)
```
result = future.result()
```

Get result with exception
```
try:
    result = future.result()
except Exception as e:
    # ...
```

Get result with timeout in seconds
```
try:
    res = future.result(timeout=0.5)
except TimeoutError as e:
    # ...
```

Get an exception
```
exception = future.exception()
```

Get exception with timeout in seconds
```
try:
    e = future.exception(timeout=0.5)
except TimeoutError as toe:
    # ...
```

Cancel a running task
```
cancelled = future.cancel()
```

Check if task is running (not done)
```
if future.running():
    # ...
```

Check if task done (not running)
```
if future.done():
    # ...
```

Check if task cancelled
```
if future.cancelled():
    # ...
```

Add a task done callback
```
future.add_done_callback(myfunc)
```