SuperFastPython.com Cheat Sheet for Python multiprocessing

Why multiprocessing?

Use processes for tasks that are CPU-bound, such as compute tasks and math operations.

Create, Config, Use Process Objects

Import
```
from multiprocessing import *
```

Create, run target function
```
process = Process(target=task)
```

Config process name
```
process = Process(name='MyProcess')
```

Config daemon process (background process)
```
process = Process(daemon=True)
```

Extend process
```
class CustomProcess(Process):
    def run():
        # ...
```

Start process (non-blocking)
```
process.start()
```

Join process, wait to finish (blocking)
```
process.join()
```

Join process with timeout
```
process.join(timeout=5)
```

Check if process is running (not finished)
```
if process.is_alive():
    # ...
```

Terminate process (SIGTERM)
```
process.terminate()
```

Kill process (SIGKILL)
```
process.kill()
```

Access process native PID
```
process.pid
```

Locks and Events

Locks protect critical section, events are safe flags.

Mutex lock
```
lock = Lock()
lock.acquire()
# ...
lock.release()
```

Mutex lock, context manager
```
lock = Lock()
with lock:
    # ...
```

Reentrant mutex lock, protect critical section
```
lock = RLock()
with lock:
    with lock:
       # ...
```

Semaphore, set num positions
```
semaphore = Semaphore(10)
semaphore.acquire()
# ...
semaphore.release()
```

Semaphore, context manager
```
semaphore = Semaphore(10)
with semaphore:
    # ...
```

Create event, then set event
```
event = Event()
event.set()
```

Check if event is set
```
if event.is_set():
    # ...
```

Wait for event to be set (blocking)
```
event.wait()
```

Wait for event with timeout
```
if event.wait(timeout=0.5):
    # ...
```

Condition Variables and Barriers

Conditions for wait/notify, barriers for syncing.

Condition variable
```
condition = Condition()
condition.acquire()
# ...
condition.release()
```

Wait on condition to be notified (blocking)
```
with condition:
    condition.wait()
```

Wait on condition for expression (blocking)
```
with condition:
    condition.wait_for(check)
```

Notify any single thread waiting on condition
```
with condition:
    condition.notify(n=1)
```

Notify all threads waiting on condition
```
with condition:
    condition.notify_all()
```

Barrier, set number of parties
```
barrier = Barrier(5)
```

Arrive and wait at barrier (blocking)
```
barrier.wait()
```

Arrive and wait at barrier with timeout
```
barrier.wait(timeout=0.5)
```

Module Functions

Utilities for working with processes.

List of all active child processes
```
children = active_children()
```

Number of logical CPU cores in system
```
num = cpu_count()
```

Process instance for current process
```
process = current_process()
```

Process instance for parent process
```
process = parent_process()
```

Add multiprocessing support, if frozen (win32)
```
freeze_support()
```

Create context with start method
```
ctx = get_context('spawn')
```

Set the default method for child processes
```
set_start_method('spawn')
```

Get a list of all supported start methods
```
methods = get_all_start_methods()
```

Get the current default start method
```
method = get_start_method()
```

Protect the entry point
```
if __name__ == '__main__':
    # ...
```

Managers

Create and use manager
```
manager = Manager()
manager.start()
# ...
manager.shutdown()
```

Manager, via context manager, shared lock
```
with Manager() as manager:
    lock = manager.Lock()
```

Shared ctypes

Share data between processes safely.

Create a shared integer value
```
var = Value('i', 100)
```

Create a shared floating point value
```
var = Value('f', 2.2)
```

Access a shared value
```
data = var.value
```

Change a shared value
```
var.value = 200
```

Create a shared integer array
```
array = Array('i', (1, 2, 3, 4, 5))
```

Access all values in an array
```
for item in array:
    # ...
```

Change one value in an array
```
array[0] = 22
```

Pipes

Create unidirectional pipe
```
receiver, sender = Pipe()
```

Create bidirectional pipe (duplex)
```
conn1, conn2 = Pipe(duplex=True)
```

Send object via pipe
```
sender.send('Hello there')
```

Receive object via pipe (blocking)
```
data = receiver.recv()
```

Block until there is data to receive from pipe
```
receiver.poll()
```

Block until data to receive, with timeout
```
if receiver.poll(timeout=0.5):
    # ...
```

Queues

Via Queue, SimpleQueue, JoinableQueue

Create queue
```
queue = Queue()
```

Create queue with limited capacity
```
queue = Queue(100)
```

Add item to queue (blocking, if limited)
```
queue.put(item)
```

Add item, with timeout (blocking if limited)
```
try:
    queue.put(timeout=0.5)
except queue.Full as e:
    # ...
```

Retrieve item from queue (blocking)
```
item = queue.get()
```

Retrieve item from queue, with timeout
```
try:
    item = queue.get(timeout=0.5)
except queue.Empty as e:
    # ...
```

Check if queue is empty
```
if queue.empty():
    # ...
```

Check if queue is full
```
if queue.full():
    # ...
```

Get current capacity of queue
```
capacity = queue.qsize()
```