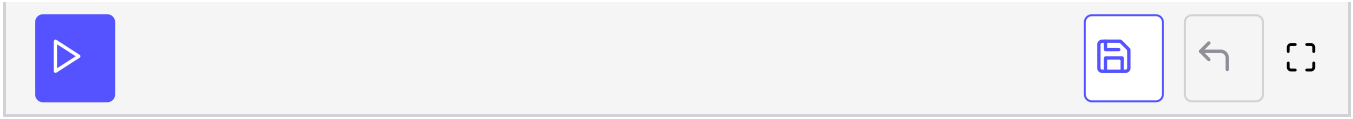
Process Communication

When it comes to communicating between processes, the multiprocessing modules has two primary methods: Queues and Pipes. The Queue implementation is actually both thread and process safe. Let's take a look at a fairly simple example that's based on the Queue code from the previous chapter:

```
from multiprocessing import Process, Queue


sentinel = -1

def creator(data, q):
    """
    Creates data to be consumed and waits for the consumer
    to finish processing
    """
    print('Creating data and putting it on the queue')
    for item in data:

        q.put(item)


def my_consumer(q):
    """
    Consumes some data and works on it

    In this case, all it does is double the input
    """
    while True:
        data = q.get()
        print('data found to be processed: {}'.format(data))
        processed = data * 2
        print(processed)

        if data is sentinel:
            break


if __name__ == '__main__':
    q = Queue()
    data = [5, 10, 13, -1]
    process_one = Process(target=creator, args=(data, q))
    process_two = Process(target=my_consumer, args=(q,))
    process_one.start()
    process_two.start()
```

```
    q.close()
    q.join_thread()

    process_one.join()
    process_two.join()
```



Here we just need to import Queue and Process. Then we two functions, one to create data and add it to the queue and the second to consume the data and process it. Adding data to the Queue is done by using the Queue's **put()** method whereas getting data from the Queue is done via the **get** method. The last chunk of code just creates the Queue object and a couple of Processes and then runs them. You will note that we call **join()** on our process objects rather than the Queue itself.

Wrapping Up

We have a lot of material here. You have learned how to use the multiprocessing module to target regular functions, communicate between processes using Queues, naming threads and much more. There is also a lot more in the Python documentation that isn't even touched in this chapter, so be sure to dive into that as well. In the meantime, you now know how to utilize all your computer's processing power with Python!