# Topic Exploration Pack

# Dining philosophers problem worksheet

The dining philosophers problem is a classic way of representing how computers can handle deadlock situations in concurrent processing. <http://en.wikipedia.org/wiki/Dining_philosophers_problem>

There are 5 philosophers sitting around a table eating spaghetti and discussing philosophy. **A philosopher needs 2 chopsticks to eat**

There are 4 things that can happen:

* Think for a while
* Pick up chopticks
* Eat for a while
* Put down chopsticks

The idea is to design an algorithm that can ensure that all of the philosophers get a chance to eat

**Activity** – Sit in a circle of 5 people. You can use pens to represent chopsticks if you want. Act out the scenario. Use the below box to record your observations about what is happening

**Act 1** - You are not allowed to speak or communicate in any way

**Act 2** - You are allowed to communicate with the other philosophers

Which was easier and why?

Observations

As you will have found, the part where you have to pick up the chopsticks is the hard part. That is because it is a shared resource. Therefore being able to pick up both chopsticks is called a race condition

We also don’t want a philosopher to pick up a chopstick that is already being used by another person. That is why we should implement a ‘lock’ system where we put a lock on each chopstick once it has been picked up so that it cannot be accessed by another philosopher. This is called a mutex or ‘Mutual exclusion’ lock. Mutual exclusion is a process whereby we make sure that two concurrent processes are in their critical section at the same time.

**However**, if everyone picked up their left chopstick first then this would create a real problem!

So, how can we prevent this?

There are different ways that we can react to deadlock:

* Ignore
* Detect and recover
* Avoid it
* Prevent it

In the example above, if the deadlock was ignored a philosopher might starve

If we somehow want to detect that there is a deadlock we might look into average times that philosophers think and eat for. If something seems abnormal we might choose to terminate the process.

Avoiding/preventing deadlock is the best thing you can do in this case. In the example code([*http://rosettacode.org/wiki/Dining\_philosophers#Python*](http://rosettacode.org/wiki/Dining_philosophers#Python) ) In this instance deadlock is avoided by never waiting while holding a fork. The fork is released and you could even try going to a different place at the table to check if any other forks are free. Of course, if you put a maximum amount of time that a philosopher can eat for, that would also help matters.