SOLID principles - cheatsheet with examples

S -Single Responsibility

A hammer should not be used as a screwdriver

Bad:

```
class Hammer
{
   function screw()
   {
   }

   function nail()
   {
   }
}
```

Better:

```
class Hammer
{
   function nail()
   {
   }
}

class ScrewDriver
{
   function screw()
   {
   }
}
```

O - Open/Close Principle

Don't modify your dick to fit the condom

Bad:

```
class Dick
{
   function __construct($size)
   {
    ...
   }

   function offerDick()
   {
    if ($this->isLarge()) {
        return $this->compressed();
    }
    return $this->uncompressed();
   }
}

class Condom
{
    function wearCondom()
    {
        Dick D = new Dick('mega');
        $this->insert(D->offerDick());
    }
}
```

Better:

```
interface IOffer
{
    function offerDick();
}

class TinyDick implements IOffer
{
    function offerDick()
    {
        ...
    }
}

class MonsterDick implements IOffer
{
    function offerDick()
```

```
    {
        ...
    }
}

class Condom
{
    function wearCondom()
    {
        IOffer D = new MonsterDick();
        // bonus points if you get D ready before wearCondom()
        $this->insert(D->offerDick());
    }
}
```

L - Liskov Substitution

Don't put things in the pussy that don't resemble a dick

Bad:

```
class Dick
{
    function stop()
    {
    }

    function move()
    {
    }
}

class Dildo extends Dick
{
    function stop()
    {
        return $this->powerOff();
    }

    function move()
    {
        return $this->vibrate();
    }
}

class TrafficCone extends Dick
{
    function stop()
    {
        return $this->stopTraffic();
    }

    function move()
    {
        throw \BadMethodCallException();
    }
}

class Pussy
{
    function playTime()
    {
        Dick D = DickFactory->getRandomDick();
        try {
            $this->insert(D);
        } catch (\Exception $e) {
            $this->goToHospital();
```

```
        }
    }
}
```

Better:

```
class Dick
{
    function stop() ...
    function move()...
}

class Dildo extends Dick
{
    function stop()...
    function move()...
}

class TrafficCone extends Stopper
{
    function stop()
    {
        return $this->stopTraffic();
    }
}

class Pussy
{
    function playTime()
    {
        Dick D = DickFactory->getRandomDick();
        // bonus points if you get dick before playTime()
        try {
            $this->insert(D);
        } catch (\Exception $e) {
            $this->goToHospital();
        }
    }
}
```

I - Interface Segregation

If it(interface) is too big, it wont fit

Bad:

```
interface ILegacyCode
{
    function fondle();
    function kiss();
    function anal();
    function blow();
}

class NewBoyFriend implements ILegacyCode
{
    ...

    function blow()
    {
        return null;
    }
}
```

Better:

```
interface ILover
{
    function fondle();
    function kiss();
}

interface IThot
{
    function anal();
    function blow();
}

interface ILegacyCode implements ILover, IThot
{
}

class NewBoyFriend implements ILover
{
    ...
}
```

D - Dependency Inversion

Don't cook your meal in the restaurant. Instead, place an order with the waiter

Bad:

```
class Hooker
{
    function doThis()
    {
    }

    function doThat()
    {
    }

    function doThisAfterThat()
    {
    }
}

class Customer
{
    function do()
    {
        Hooker H = new Hooker();
        H->doThis();
        H->doThat();
        H->doThisAfterThat();
    }
}
```

Better:

```
interface ISatisfier
{
    function satisfy();
}
class Hooker implements ISatisfier
{
    function satisfy()
    {
        $this->doThis();
        $this->doThat();
        $this->doThisAfterThat();
    }
}

class Customer
```

```
{
    function do()
    {
        ISatisfier hooker = new Hooker();
        // bonus points if you pass hooker to do()
        hooker->satisfy();
    }
}
```