SUSAN BAILEY CFG FULL-STACK HOMEWORK WEEK 3

1. WHAT IS OOP?

In Object Oriented Programming we combine a group of related variables and functions into a unit, or objects. We refer to these variables as properties, and the functions as methods.
An object can be a physical item, such as a house or car, but it can also be an appointment or booking, or a reservation at a restaurant.
An object is anything you want to store and process data with, and is also known as an entity.

Objects in OOP are data, and there are different types of object, and different ways to interact with them.

In Python, everything is an object, and we can create new objects, manipulate them, and then discard them.

Abstraction

Abstraction means to simplify reality.
To create objects, we need a class. It is a template for creating objects, it is code written to describe the object properties. It can be compared to a pastry cutter, because it can create many objects.
Operations are actions and behaviour that can be performed by the object, and are known as methods.
Each object is an instance of a class, creating an object from a class is called instantiation. Once the objects have been created their properties can be assigned values and used, making each object of the same type a unique entity. The property values assigned to an object are collectively known the state of the object. It's also possible to assign values to properties while an object is being instantiated, by means of a method known the constructor.

Polymorphism

'Poly' means many, and 'morphism' means form, meaning many forms. Polymorphism is a very powerful property.

For example, we can begin with a class of Animal.
If we also have classes of Cat, Dog, Rabbit, we can say these classes extend the Animal class.

We could create a new Animal array called Kingdom, and we could add cat, dog, rabbit to that array, because they are part of the same array of generic animal objects.

```
dogRover = newDog( );

animalSpot = newDog( );
```

Both the above statements are true because our Dog object can also be an Animal object, it is a child class of our Animal parent object.
We can create an array called kingdom that contains every single type of animal within it. If we decided to slice an index from the array, we would find many objects within it that are different, but still belong to the same class.

```
kingdom [ 0 ] = newDog( );
kingdom [ 1 ] = newCat( );
```

Again, both statements are still true because kingdom is our animal array, and we can include the newDog( ) and newCat( ) inside this array because they are a child class of the kingdom array.

```
for (animal a : kingdom ) {
        a.sleep ( );
        a.eat ( );
}
```

Each animal will still have it's on unique sleeping and eating patterns, however they can still be included within the kingdom array due to their class.
We can create this loop knowing that polymorphism means each object can be of the same parent class and still be different, all the time knowing our code will keep track of this for us, making polymorphism very powerful tool for us to use.

Encapsulation
Encapsulation is often referred to as 'information hiding' , because it hides the complexity of the inner detail of an object from the programmes that are making it. The data contained within an object and the functions that manipulate the data are bound together.
It describes the idea of bundling data and methods that work on data into one unit, like a class. This is often used to hide the internal information, or state of the object from the outside.

In large companies it's often the role of the senior programmer to write the classes that will be used by the junior programmers , it could be argued that the junior programmers only need to know the name of the class properties and methods, and what needs to be supplied when they are called. In simple terms, as long as they understand how to use the classes, they do not need to know the complexity inside them.

During our Python course we have learned to use methods such as push( ), len( ) and pop( ). These come built-in to the Python environment and we use them without questioning how they work, or where the logic for them is stored. We trust that they do work and they make many processes much faster.

There are three primary types of access modifiers across most programming languages -

Private
The most restrictive access modifier, 'private' allows access to the object only through the class itself.
Protected
This is one stage less secure than 'private', the protected modifier allows access to the state by other classes in the same package, and can be inherited by child classes.
Public
This imposes no restrictions and can be accessed and modified openly.

Access modifiers are specified through underscores and the beginning of a variable's or function's name. these can still be accessed so not entirely private, however access has been superficially restricted to enforce the semblance of restriction.

```
class Student:
   _name = "John" # protected data (single underscore before name)
   __roll = 10 # private data (double underscore before name)


def __init__(self, name, roll, school):
        self._name = name
        self.__roll = roll
        self.school = school # public data
```

Inheritance
The concept of inheritance was introduced with ES6, a Javascript update in 2015.

The idea of a class is a template for the making of an object. It can be compared to a cookie cutter.
Class = Cookie Cutter
Variable = Cookie

The parent / super class can pass down properties to a child class. The child class will inherit all the public and protected properties and methods from the parent class. The child class can also adapt the properties and add to them.

Inheritance is useful because it can save us repeating code, because class can be passed down from class to class.

I need to create a super class and inside it will be a constructor function.

```
class Particle {  this is the parent or super class
        constructor ( ) {
        }
        constructor1 ( ) { inheritance means these properties are copied into SquareP - these
                           functions become part of the squareP class with the use of 'extends'.
        }
        show ( ) {
    }
}

class SquareP extends Particle {  SquareP is the child class and it inherits from Particle
        constructor ( ) {
        super( ) ;     using the word 'super' means my variable will do exactly the same thing as in
                       the Particle class. Using the word 'super' here means I can run the 'parent
                       class'.
}       this.bright = random( 255 ); I can also change the variable properties to create something
                                     new.
        }
}
```

```
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  public function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

// Strawberry is inherited from Fruit
class Strawberry extends Fruit {
```

```
  public function message() {
    echo "Am I a fruit or a berry? ";
  }
}
$strawberry = new Strawberry("Strawberry", "red");
$strawberry->message();
$strawberry->intro();
?>
```

Encapsulation vs Abstraction

I'd like to expand on Encapsulation and Abstraction because these two aspects of OOP are complimentary.
As already discussed abstraction means to hide lower-level components within higher level entities to make them easier to program and manage. Abstraction helps developer in two ways -

1. Making it easier for end-users to interact with the logic and
2. Enabling developers to model objects and use them as complete units.

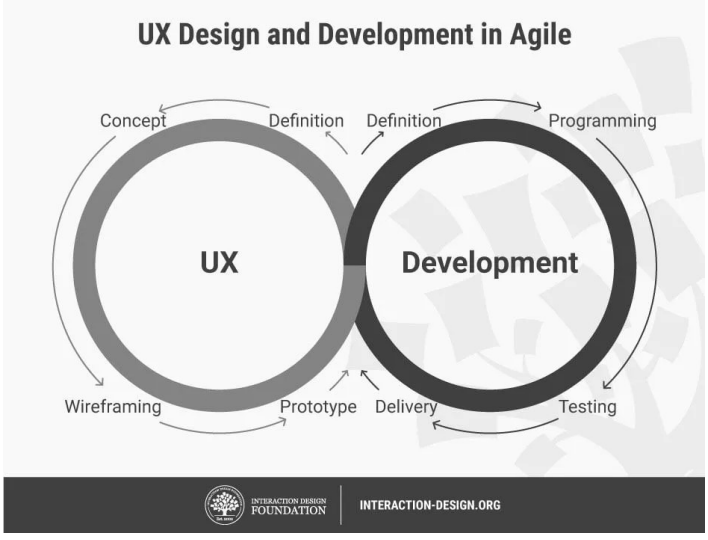
The bundling together of information and hiding of information - encapsulation - is similar to abstraction, and can be argued abstraction is a result of encapsulation.

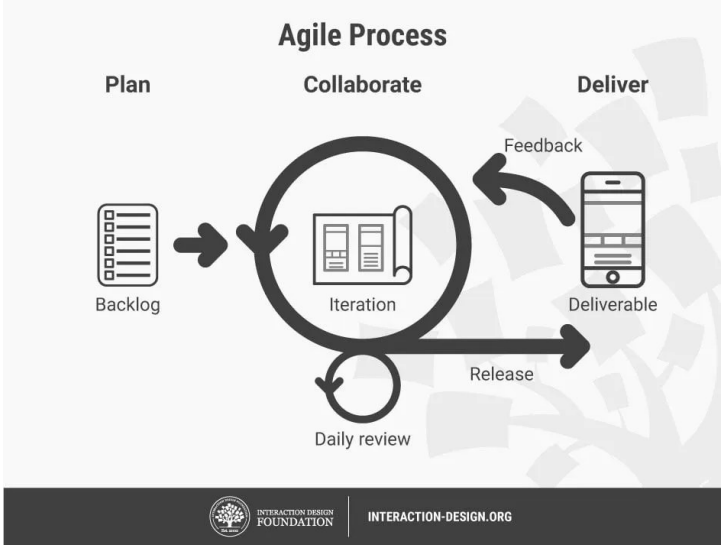
At a much higher level, packages such as operating systems, and website frameworks are also encapsulation, providing us with lightweight 'containers' of technology that follow the encapsulation model.

6. What is Agile Development?

Agile is a streamlined working model that is designed to be flexible and respond to design challenges as they emerge. It was introduced in the mid 90s, but wasn't rolled out through the industry until around 2001, when downloads were started to appear in the marketplace. That same year a group of tech professionals met in Utah to form 'the Agile Alliance' and wrote their manifesto.
Of it's twelve principles, four form the heart of Agile development -

Individuals and interactions over processes and tools.
Working software over comprehensive documentation.
Customer collaboration over contract negotiation.
Responding to change over following a plan.
Transparency and iteration are essential.





‘Sprints’ - the name given to the full life-cycle of each project - are typically around 1-4 weeks long, meaning a product can be designed, developed and shipped all within a short timeframe.

The first process is for the Product Manager to be given a brief by the client, after which the product manager will create a ‘backlog’ and work with the UX / UI designers to create a wireframe and graphic design for the client to approve.
The PM will then meet with the team to plan the project technical structure with the developers.
The entire development team of product manager, scrum master, designers and developers will meet in a daily ‘scrum’ to ensure the project stays on track and all team members are accountable for their work.
Once the product is finished the ensure team meets for a Sprint Review, which is an opportunity for each team member to discuss their contribution to the project.

At the end of each sprint, the team meets again to hold a retrospective, where they conduct a full review of the last print and discuss what went well, and implemented again, and what could be improved.

It is then expected that the product comes back again for development after gathering feedback and data from the customer, which gives the team another opportunity to refine and redevelop within another sprint.
This fast, iterative process is what makes Agile so relevant for the tech industry. The change in mindset from shipping completed, ‘boxed’ goods to delivering iterations of product is completely in step with the marketplace’s consumption of constantly changing sites and apps.

7. What is Waterfall Development?
Waterfall is a traditional, linear model of project management which follows a rigid model of development throughout the product lifestyle. The waterfall method inherits it’s model from traditional manufacturing, where planning and research are completed up front with the intention of building and shipping a ‘complete’ product. The product is handed from team to team in a sequential manner who complete their contribution to the product until it reaches the end of the production line and is ready for shipment.

The processes are as follows -
Requirement analysis
System design
Implementation
Testing
Deployment

Maintenance

The waterfall method is a simple model to understand and implement as it follows the lifecycle of the project. It is also a very disciplined method because so much time is given to planning, it is rare to miss a deadline because every stage has been carefully planned.

A waterfall development project requires careful planning up front before being designed and move onto production. After shipping this type of product would require only some periodic maintenance and upgrades,
The style of waterfall is suitable for large projects, such as large government projects, Windows or OSX platforms etc, which will be used by many people, because to adopt an iterative approach to this type of project could be very problematic for the users.

The Agile method is best suited to apps and websites where it is better to ship a ‘minimum viable product’ and then amend the product later after gathering feedback.
The waterfall method is best suited to large scale projects that require a lot of planning up front to throughly assess the project requirements, because there need to be very few upgrades or maintenance after shipping to avoid confusing or inconveniencing the customer.