Visualising OOP

Quick intro…

- Web developer turned Software Developer
- Teach Python, JavaScript, PHP, Data Analysis & Software Application Development
- Cert III to Diploma level (Joondalup, East Perth and Northbridge)

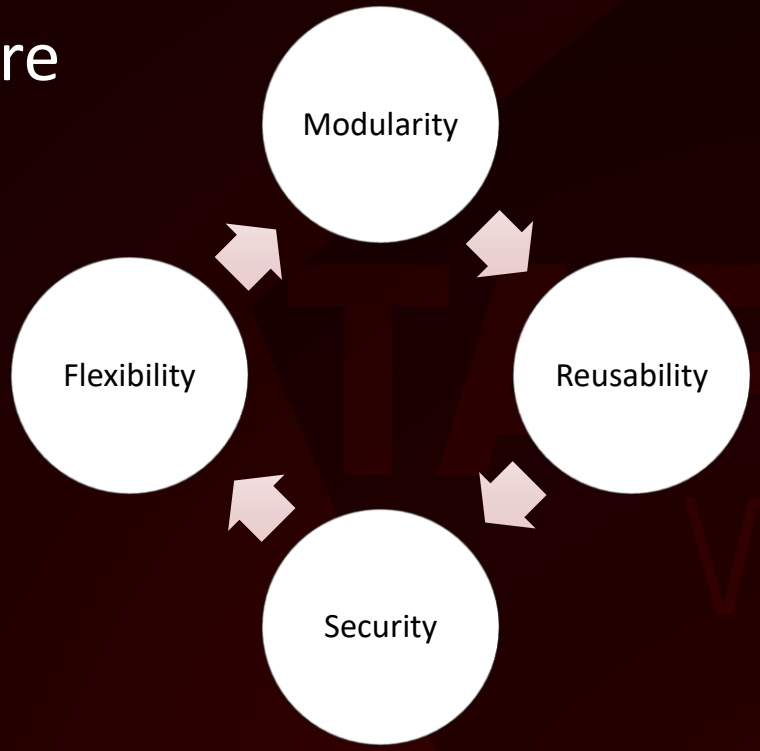
Staring early with OOP

Are we there yet?

- start with understanding the basics
- grappling with how to express logic in a new language
- you might start with simple programs
- gradually moving towards more complex code structure.
- as you progress, you'll start to see common patterns of code.

Benefits of OOP

- Code can be broken down into pieces that are easier to manage & debug.
- Code created for one program can be used in other programs
- Hiding code helps keep data safe from outside interference and misuse.
- Allows a single access point route to code making it more flexible and usable on a larger scale.



Consider your use of API's…

Variables *aka* Properties

- **Declaration & Assignment -** assigning a value to a property
- **Types –** properties can have several
- **Properties** - can be used in expressions

```
x = 5
y = "Hello"
z = x * 10
print(z)
```

Dealing with 2D – Data Structures

```
const contact = {
        firstName: "John",
        lastName: "Doe",
        age: 30,
        address: {
                street: "123 Main St",
                City: "Anytown",
                zipCode: "12345"
        },
        skills: ["JavaScript", "HTML",          "CSS"],
        socialMedia: {
                twitter: "@johndoe",
                linkedIn: "linkedin.com/in/johndoe"
        },
        isActive: true
};
```

Classes – Our blueprint

- blueprint for creating objects
- defines a set of properties & methods
- creates our objects when we ask them

```
// Declaration
class Organisation {

      constructor(name, location) {
            // Properties
            this.name = name;
            this.location = location;
      }

      // Methods
      displayInfo() {
            console.log(`Organisation:
            ${this.name}, Location:
            ${this.location}`);
      }
```

Creating our Object

- an instance of a class has its own unique data
- instance is created by calling the class as if it were a function.
- properties of the instance can be accessed by using dot notation

```
// Example usage
const org = new Organisation(”NM Tafe", ”Perth");
org.displayInfo();
```

Handling the Object

- contains properties (properties) and behaviours (methods).
- **properties** - data stored in the object e.g. `org.name`
- **behaviours** - the actions that the object can perform, defined by its methods e.g. `org.display_info()`

```
org = Organisation("XYZ Corp", "Perth")
org.display_info()
```

4 Pillars of OOP

- **Encapsulation** - wrapping up methods & properties in a class
- **Abstraction** – only showing the essential properties & methods
- **Inheritance** – creating new classes from existing classes
- **Polymorphism** – a class to handle multiple different properties & behaviours

Encapsulation

- bundling of data (properties) & methods that act on that data into a single place (a class).
- data is secure as it can only be accessed through the properties & methods of the class
- gives us a reusable template to use throughout our code

Encapsulation in code

```
class Organisation {
        constructor(name, location) {
                this.name = name; // This is a property (a coloured ball in the jar)
                this.location = location; // This is another property
        }

        displayInfo() { // This is a method (another coloured ball)
                console.log(`Organisation: ${this.name}, Location: ${this.location}`);
        }
}

// Creating an instance of Organisation (the jar)
const org = new Organisation("XYZ Corp", "Perth");

// Accessing properties and methods (looking at the coloured balls in the jar)
console.log(org.name);
console.log(org.location);
org.displayInfo();
```

Inheritance

- one class can use the properties & behaviours of another (parent/child)
- code can simply be reused & not re-written
- the new class(child) is free to have its own properties & behaviours

Inheritance in Code

```
class Contact extends Organisation { // Contact is a subclass of Organisation
        constructor(name, location, contactName, isClient) {
                super(name, location); // Inherits properties from Organisation
                this.contactName = contactName; // This is a new property
                this.isClient = isClient; // This is another new property
        }
        displayContactInfo() { // This is a new method
                const clientStatus = this.isClient ? "a client" : "not a client";
                console.log(`Contact: ${this.contactName}, ${clientStatus} of ${this.name}`);
        }
        displayContactType() {
                return this.isClient ? "Client" : "Non-client";
        }
}

// Creating an instance of Contact (the smaller jar)
const contact = new Contact("XYZ Corp", "Perth", "John Doe", true);

// Accessing properties and methods (looking at the colored balls in the jar)
console.log(contact.isClient); // Outputs: true
contact.displayContactInfo(); // Outputs: Contact: John Doe, a client of XYZ Corp
```

Abstraction

- hide certain important behaviours & properties
- child class only see what it requires (inheritance)
- child class can override (re-write what in inherits)
- hides complexity but allows for reuse

Abstraction in Code

```
class Project extends Organisation {
        constructor(name, location, projectName) {
                super(name, location);
                this.projectName = projectName;
        }

        // Overriding the displayInfo method
        displayInfo() {
                console.log(`Project: ${this.projectName}, Organisation:
                ${this.name}, Location: ${this.location}`);
        }
}

// Creating an instance of Project
const project = new Project("XYZ Corp", "Perth", "Project Alpha");

// Displaying project info
project.displayInfo();
```

Polymorphism

- one class can use the properties & behaviours of another (parent/child)
- code can simply be reused & not re-written
- the new class(child) is free to have its own properties & behaviours

Polymorphism in Code

```
// Project class
class Project {
        constructor(project_name, contact) {
                this.project_name = project_name;
                this.contact = contact;
        }

        display_project_info() {
                console.log(`Project: ${this.project_name}, Contact:
                ${this.contact.contact_name}, Type: ${this.contact.display_contact_type()}`);
        }
}

// Creating instances of Contact and Project
let client_contact = new Contact("XYZ Corp", "Perth", "John Doe", true);
let non_client_contact = new Contact("XYZ Corp", "Perth", "Jane Doe", false);

let project1 = new Project("Project Alpha", client_contact);
let project2 = new Project("Project Beta", non_client_contact);

// Displaying project info
project1.display_project_info();
project2.display_project_info();
```

Disclaimer

I won't be taking any hard questions on this …. sorry