C++

Basics

- `for(auto v: vArray)` - range over an iterable without explicit index
- `enum class` - Doesn't explicitly map to int, safer and recommended
- `func(&int)` - Reference to int, like pointer but dereferencing happens automatically, recommended
- Always pair unions with an enum to represent which type its supposed to take on
- Properties of an object accessed with `object.property`, of a pointer to an object using `pointer->property` (or simply `.` if it is a reference `&` not raw pointer), and members of a class are accessed using `std::cout`
- `lvalue` - an object that occupies some identifiable location in memory and can be assigned to
- `rvalue` - expressions that aren't `lvalues`, an object that isn't in memory, usually on the RHS in expression

Modularity

- Use the same header file for implementation and usage
- Errors are thrown to allow the user of a library to decide how to handle unexpected cases

OOP

- `complex operator+(complex a, complex b) { return a+=b; }` overloading default operators
- `new` - Assigns memory on the heap for the object and returns a pointer. Has to be explicitly deleted (even after it leaves scope). Useful to allow a variable to be accessed by its pointer from outside of the current scope (otherwise it'll be automatically deleted).
- Concrete classes - Same as built in types, constructor initializes any needed heap properties and `~Destructor()` is called if `delete` is called to deallocate (unreserve) it .
- representation - the properties / variables of a class, what stores memory
- abstract class, similar to an interface in Go, simply a collection of methods such a class must implement, can be used to specify what an argument is expected to have. Implemented as `class Implementor: public Abstract {}`, this is **inheritance**
- Polymorphism - one interface used to represent many other types which may satisfy it
- `virtual` - May be redefined later in a derived class, `virtual void x = 0` means it **must** be redefined otherwise the class cannot be instantiated, there is no default implementation.
- Base functions / properties can be accessed within subclass implementations
- Calling `delete` on an abstract object calls the destructor of the shallowest subclass (as it has access to the most "extra" properties)
- `dynamic_cast` can be used to check what derived class an abstract argument is
- Resource handle - A class that is responsible for managing underlying resources, these provide custom copy implementations to prevent violating validity, for example assigning a `vector` to another variable results in two vectors that refer to the **same** underlying elements. Such handles should implement a **copy constructor** and **copy assignment** operator `Vector& operator=(const Vector& a)` so underlying resources are correctly reallocated.
- Marking a constructor `explicit` prevents automatic type conversion
- Default copy / move operations in a parent class can be deleted using `Shape& operator=(const Shape&) =delete;`

Generics

- Prefixing a class / function with `template<typename T>` accepts a type as a generic argument, so that `T` can be used throughout implementations

- So-called function objects can be defined by implementing the () operator, for example `bool operator()(const T& x) const { return x<val; }`
- Type aliases can be defined: `using value_type = T` is a public property of all container classes in the standard library, accessed using `Class::value_type`