C++

- `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
- 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
- `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 memory 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, 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 most derived 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.