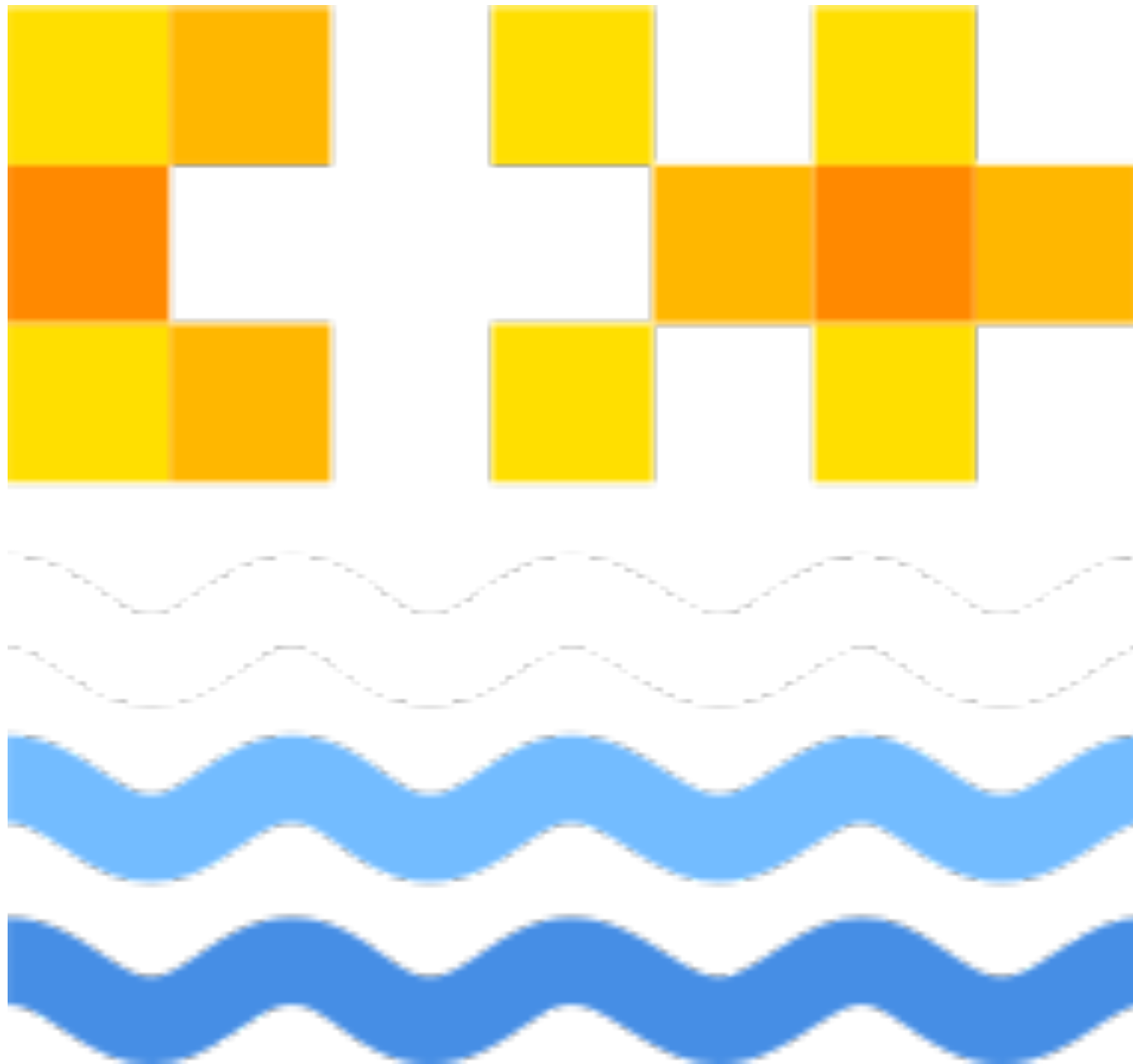




# Initial C++ — session 1

Tristan Brindle



# Register now for C++ on Sea!

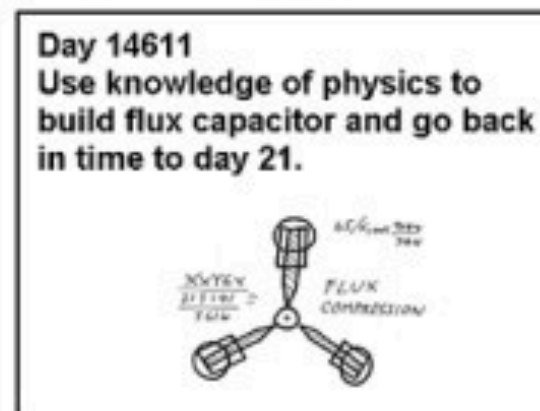
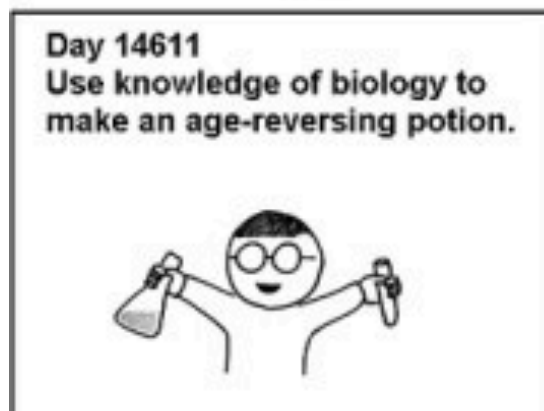
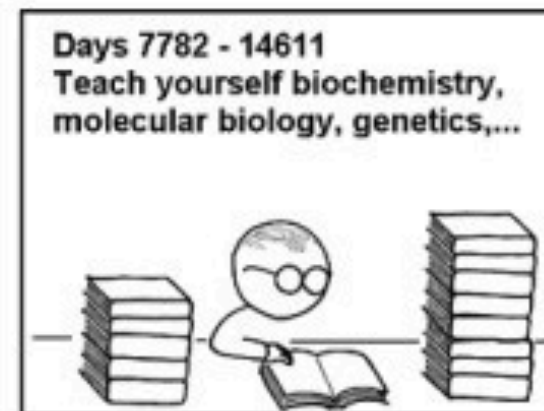
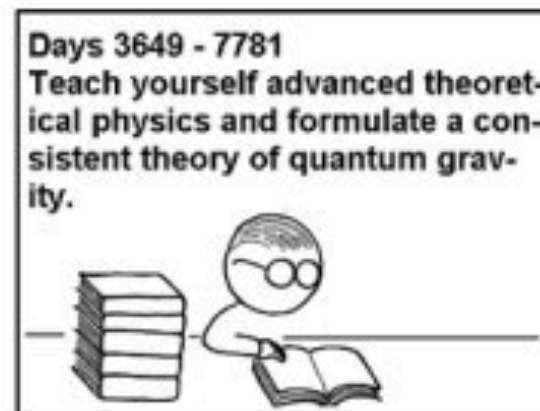
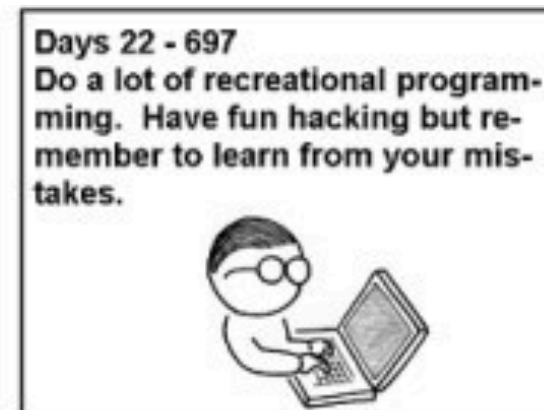
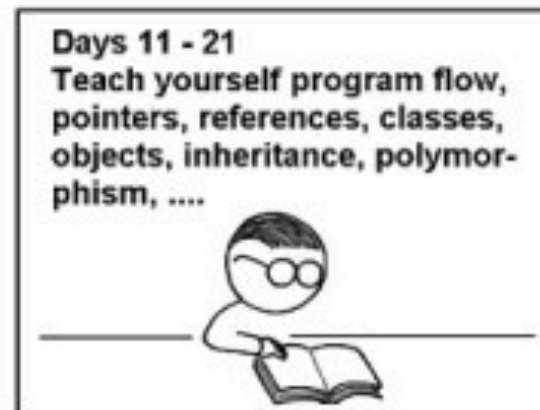
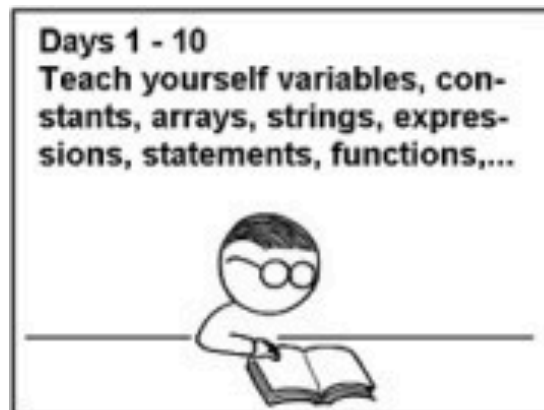
<https://cppponsea.uk/>

# About these sessions



- An introduction to C++
- A mixture of talks, class exercises and homework
- We can't turn you into an expert (sorry!)
- ...but we'll try to give you enough information to get started

# “Teach yourself C++ in 21 days”



As far as I know, this  
is the easiest way to  
“Teach Yourself C++ in 21 Days”.

# Modules

- We're dividing the course up into shorter modules
- Format: 4-8 weeks of tutorials and practise, followed by a short multiple-choice test to help us gauge everyone's progress
- This module is "Initial C++"
- No prior C++ knowledge assumed

# Feedback



- We'd love to hear from you!
- The easiest way is via the *cpplang* group on Slack — we have our own channel, *#cpplondonuni*
- Go to <https://cpplang.now.sh/> for an “invitation”

# Today's lesson plan



- Introduction to C++
- “Hello World”
- Types
- Functions in C++

# Why C++?



- Usually because it's *fast*
  - Direct access to hardware
  - Zero-overhead abstractions
  - Efficient resource usage
- Used everywhere
  - Everything from micro controllers to supercomputers
  - Games, financial trading, web browsers, etc etc etc



# Why *not* C++?



- Usually because it's *hard*
  - Partly true unfortunately
  - C++ allows access to low-level facilities
  - C++ has lots of features — use them wisely
  - Some warts and “gotchas” due to its age
- ...but it's not *that* hard! 😊

# A (very) brief history of C++



- 1979: Bjarne Stroustrup starts work on “C with Classes”
- 1983: C with Classes renamed C++
- 1990: ISO committee formed to standardise C++
- 1998: First standard version released (C++98)
- 2011: Major update to the standard (C++11)
- 2014, 2017: Further standard updates (C++14, C++17)
- 2020, 2023....?

# “Modern C++”



- C++11 changed the game
- Don't bother learning C++98
- Make sure any textbooks or online resources you use are teaching you *today's* C++.

**Any questions before  
we move on?**

# Exercise 1



1. Go to [wandbox.org](https://wandbox.org)
2. Enter this text
3. Click “Run”

```
// Our first C++ program!  
  
#include <iostream>  
  
int main()  
{  
    std::cout << "Hello world\n";  
    return 0;  
}
```

# Deconstructing “Hello World”



```
// Our first C++ program!
```

- This is a comment
- Inline comments start with two slashes (//) and continue to the end of the line
- Multiline comments start with /\* and end with \*/

# Deconstructing “Hello World”



```
#include <iostream>
```

- This line tells the compiler to include the contents of the `iostream` *header* in our program
- `iostream` is provided by the standard library and contains code to let us write to (and read from) the console
- `#include` is used to break large programs into smaller, manageable pieces, and to use code from other libraries (as we’ve done here)

# Deconstructing “Hello World”



```
int main()
```

- This line declares a *function* called “main” which returns an `int`(-eger) and takes no parameters
- Every C++ program contains a `main()` function, which is where execution starts.
- `main()` has some special rules



# Deconstructing “Hello World”



{

- A curly brace opens a *block*
- In this case, the block contains the *definition* of our `main()` function
- Blocks control object lifetimes in C++, as we'll see later

# Deconstructing “Hello World”



```
std::cout << "Hello world\n";
```

# Deconstructing “Hello World”



```
std::cout << "Hello world\n";
```

- `cout` (“console output”) is an object provided by the standard library for printing text
- As part of the standard library, it belongs to the `std` namespace, so we write `std::` to access it
- Later we’ll see a shortcut to avoid having to type `std::` everywhere, but use it with caution.

# Deconstructing “Hello World”



```
std::cout << "Hello world\n";
```

- The << symbol means (in this case) “pass the thing on the right to the *output stream* on the left”
- This is an example of *operator overloading* in C++
- Later, we’ll see other meanings of <<, and how to define the meaning of operators for our own types

# Deconstructing “Hello World”



```
std::cout << "Hello world\n";
```

- This is a *string literal*
- The `\n` at the end means “start a new line here”
- Sometimes you’ll see `(std::)endl` used as an alternative way to start a new line

# Deconstructing “Hello World”



```
std::cout << "Hello world\n";
```

- Most C++ statements end with a semicolon
- If you forget it, the compiler will usually tell you...
- ...but if you get strange errors, check that you’ve got your semicolons correct

# Deconstructing “Hello World”



```
return 0;
```

- The `return` keyword tells the program to leave the current function, returning the value (in this case 0) to the caller
- By convention, returning zero from `main()` tells the operating system that the program ran successfully, any other value indicates an error
- Remember how I said `main()` was special....?

# Deconstructing “Hello World”



}

- This closes the block we opened earlier
- When we leave a block, local variables defined in that block get destroyed
- This is the single best thing about C++ (really!)



**Any questions before  
we move on?**

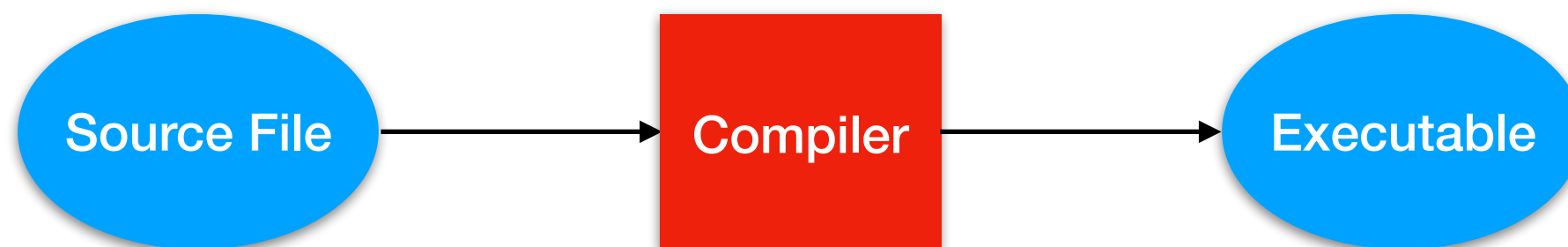
# The C++ compilation process



- C++ is a *compiled language*
- A program called a *compiler* takes our *source code* and turns it into an executable which we can run
- Many potential errors in our program can be caught by the compiler — we call these *compile-time errors*
- The compiler will also generate *warnings* for code which is legal but potentially buggy — don't ignore these!
- Wandbox is an example of an online compiler — it compiles and runs generated executable in a single step

# The C++ compilation process (simplified)

- Schematically, the process looks like this:



# Types

- In programming languages, a *type* is a way of giving meaning to some data
- The *type* of some data tells us what it represents and what we can do with it
- For example, we can multiply two numbers, but we cannot meaningfully multiply two strings

# Types

- C++ has many built-in (“fundamental”) types, such as `int`, `float`, `double`, `bool` etc
- The standard library has lots more commonly-used types such as `std::string` and `std::vector`
- The language provides us with many tools to define our own types, which we’ll learn about as the course progresses

# Types

- C++ is a *statically-typed* programming language
- This means that every variable has its type determined when the program is compiled
- C++'s *type safety* means that we can catch many potential errors before the program is run
- There are ways around the type safety rules, but avoid them if at all possible

**Any questions before  
we move on?**

# Functions



- C++ programs are composed of *functions*, small pieces of reusable code
- We've already seen the `main()` function
- Note: C++ has two kinds of functions, *member functions* and *non-member functions*. Today we're taking about non-member ("free") functions. We'll talk about member functions ("methods") later in the course.



# Functions



- The usual form of a function declaration is

```
return_type function_name(param_type param_name, ...)
```

- Every function in C++ returns zero or one value(s)
- If the function does not return a value, then the return type is `void`

# Functions

- For example, we can define a function which adds two ints like so:

```
int add(int a, int b)
{
    return a + b;
}
```

- This defines a function “add” which takes two parameters named a and b (both of type int) and returns a value of type int

# Functions

- To *call* (run) a function, we say `function_name(arguments)`, e.g

```
std::cout << add(3, 4); // prints 7
```

- In C++, a function must be *declared* before it can be called
- This means the function must appear in the source code *above* where it is used

# Exercise 2



- In your “hello world” program in Wandbox, write a function

```
void hello_cpp_london_uni()
```

which prints “Hello C++ London Uni” to the console

- Call this function from your `main()`

# Solution 2



```
void hello_cpp_london_uni()
{
    std::cout << "Hello C++ London Uni\n";
}

int main()
{
    hello_cpp_london_uni();
}
```

# Exercise 3



- In Wandbox, write a function `say_hello()` which takes a parameter of type `std::string` called `name`, and returns a string containing that name with “Hello ” in front
- Use this function to print “Hello <your name>” from `main()`, e.g. “Hello Tristan”
- You will need to add `#include <string>` near the top of your program to use `std::string`

# Solution 3

```
#include <iostream>
#include <string>

std::string say_hello(std::string name)
{
    return "Hello " + name;
}

int main()
{
    std::cout << say_hello("Tristan") << '\n';
}
```

# Summary



- This was only a very brief introduction to the wonderful world of C++
- We've learnt how "hello world" works
- We've learnt about types and type-safety
- We've learnt how to `#include` standard library headers
- We've learnt how to define functions
- We've been introduced to `std::string`



# Next time

- Intro to C++ part 2
- More on types, `const` and `auto`
- Variable declarations
- Value semantics and RAI
- Control flow

# Homework



- Read about `std::cin`, the input counterpart to `std::cout`
- Write a program which reads in a string using `std::cin`, and then prints “Hello, <name>”, where <name> is the string the user supplied
- Bonus: use `std::vector` to read in a list of strings. When an empty string is entered, print “Hello <name>” for each string that was entered so far, and then exit the program.

# Online resources



- <https://isocpp.org/get-started>
- [cppreference.com](http://cppreference.com) — The bible, but aimed at experts
- [cplusplus.com](http://cplusplus.com) — Another reference site, also has a tutorial section
- [learncpp.com](http://learncpp.com) — Free online tutorial, very up-to-date
- <https://www.pluralsight.com/authors/kate-gregory> - Comprehensive set of courses from an experienced C++ trainer (free trial)
- [reddit.com/r/cpp\\_questions](https://reddit.com/r/cpp_questions)
- Cpplang Slack channel — <https://cpplang.now.sh/> for an “invite”
- StackOverflow (but...)