Today we are going to introduce custom types, what they are and why we use them. Benjamin Lee Whorf was a famous linguist, he said "Language shapes the
way we think, and determines what we can think about.". We humans can think in concepts and we can refine it in real-time, computers cannot do that, if
you want to compare natural languages and programming languages, we think and we speak using objects, verbs and a grammar, that is just the syntax a
programming language should obey, the names we use are the names of the variables, while the concepts combined are the types, a type is a specific
interpretation, implementation of a concept, the verbs can be seen as the functions, I can combine them to make sentences, which can be combined to make
a story that is our code. What if I do a complicated stuff and I need a complex concept or to split one into simpler ones, that is what custom types
are made for, the relation among objects is what is called OOP, today we are going to see custom types, I decide their name and should coincide with
a particular implementation of a concept. The first custom type is enum. Imagine you are going to code the driver of a traffic light, you need to perform
some actions or you are working on self-driving cars and take different decisions according to the light, at some point you have to understand the color
and according to the color you take an actions, you have three options, you can have three ifs, this is not readable, it would be much nicer and readable
if I can use green, red and yellow instead of numbers, it makes the code easier to maintain and to debug, enum introduces a type, named color here. It
appears also in C, the name I want to choose for this new type, I want a concept of color and I just want to dinstiguish among the three, I say that,
they are integer variables by default, enum is a way to generate constants with different numbers, suppose you have one million colors and you are sure
that each of them will have a different number, once I have introduced that type I can use it as an int, char, bool, I can write the function dwim and use
a color as an input. If you have a lot of options you can use if, else if, else, something that is more readable is a switch case statement, between
parentheses the variable, you can use as an argument just int, char, and enum. It is a sort of archaic. You cannot use strings, vectors, and so on. in {}
we have all the cases that have then the value of that variable, remember to put break which means exit from the switch case, otherwise it starts executing
the actions of the next case, it is a good idea to put a default action that is executed when none is matched, you use the default keyword at the end.

Even though they are integers, you cannot assign an integer to enum, the other way around, enum playing enum can be converted implicitly to integers. I can
construct a color from an integer, if you create an enum with something outside the range, it will end up in the default section of the switch case. Since
C++11 we have what is called scoped enum, the syntax is similar, do not switch class and enum, what is the difference, now our colors have a longer
name, their name is prepended with what he wrote there, introduced a scope named color and those variables are inside the scope, :: is the operator of
scope resolution, so include them in fully qualified name. The scope may help you to collect functions, data, options with the same names that belong
to different concepts, what if you have something different, like other colors, or the same color can have different meanings, like green for traffic
light or signs, the same name with different qualifier names and I am sure I do not have clashes. Apart from that there is another feature, they do not
implicitly convert to integers, so that is no more valid, you can force to have an integer from a scoped enum calling the constructor of integer. Why
scoped? I can define a custom beginning and the others will adapt. using namespace std allows me to avoid to write std::, it is like from package import *
in Python, you are populating the environment with those functions and I do not need the fully qualified name. But we can have name clashing, the solution
is to use a scope enum (or change name). Let's try to put some light on namespaces with the next file. A namespace is a way to combine classes inside
a scope with a given name, I can define a namespace, outside the namespace the fully qualified name would be choose_name::variable, we can have
nested namespaces, here we also have one variable and three functions, if I want to change that variable that would be a global varibale, whose fully
qualified name is choose_a_name::variable. The full name is that one. What you usually do when you write a library you put all the material in one
namespace that will be named like the library. That is the idea, I combine everything into one namespace so that I will not have a conflit with other
objects. You can use an unnamed namespace, namespace {} and the content will be visible only within the compile unit, if that is written in a .cc file that
will be where it is visible, for them you do not need the name, or only ::var, this is to force the compiler to look in the global namespace, somebody
likes that. Same story for the functions, in addition you have to follow or the nested namespaces, if I want to refer to those symbols, if I want

to redefine those functions, I have to specify its fully qualified name and go on. Or I can reopen the namespace and within it I can reach it without using the fully
qualified name. namespace is not done in the main, you can use namespace, you can also do using namespace choose_a_name::nested. If you want to use
just one symbol or two symbols without the fully qualified name, I can say using choose_a_name::nested::function, for example using std::endl and using std::cout, I have
populated the global namespace with them. That's enough for types devoted to an option. We played with different containers, didn't you ask yourself
how is it possible? How did they implement them without memory leak, one works in the heap and the other in the stack, how they introduced the concept
of vector, of array, how defined the custom type vector? They did through classes, the first name of C++ was indeed C-with-classes, they were the first
feature added on top of C, in C there are struct they can contain just data no functions, in C++ struct and class are the same thing, the only difference
is that by default members of a struct are public, members of a class are private (by default), both of them can contain data and functions, from now
on he will say classes, from the concept they are the same. Suppose I want to introduce the concept of a Point, a particular implementation if I lie in
two dimensions, three and so on, another design choice is having two variables separated, or having an array of doubles of different dimensions, how can
I define a class or a struct, note the ; at the end of a struct, usually you put ; at the end of a declaration, this is the first time we introduce that
concept so we use ;, if we want to implement that function outside the class the fully qualified name is the name of the structure::point, data
are accessed in a different way, for the class pointc, I define the function print when I declare, there is no ; if you do it might raise a warning. Data
are not accessible from outside, how can I declare a class, class and object have a difference, object with respect to memory, class is the type, I introduced
an object of type ps, objects are instantiations of a class, each object has nothing to do with the other. When you define a function outside a class
or inside there is a difference, functions that are defined inside a class are replaced throughout your source code as you use it, functions defined outside
are usually not inlined, functions that are very short are defined inside, functions that are longer than 1-2 lines of code you do not get any advantage
so you define outside otherwise your binary blows up (what happened in early versions of Windows), by default you are never sure what the compiler will inline,

for sure within class, maybe outside. I can access data using the . operator, I call the function of a class using the . notation as well, the same as in
C or the equivalent of % in Fortran. I can use the public keyword to declare as such what follows. struct == class public. Defining private variables
in a struct is never a good idea. Public functions can access public members, why may I want to have private/public? You want your object to always be
in a meaningful state. Suppose we want the implementation of the concept of a date, which level of detail? It depends. Suppose we just have up to day, if those
three variables are public then the user can change their values without any problems, what if by mistake the user says that the day is different, nobody
can prevent him but if I use them as private and I force the user to change them through a public interface that I doublechecked to produce correct
results, it guarantees the user the object is in a meaningful state. While Fortran you access through %, the syntax when you have pointer of a class is
different, the definition is the same, can be initialized the same way, I access the data with ->, the arrow, if use . it is a compiler error. If you
have a reference you can use ., a reference is just an alias, another way to call a guy, you can have built-in arrays of your new type. I can use
std::arrays, same story. I can use std::vector and push_back, notation. I can say (*p).x, the same as p->x. Let's have a short break. These variables
inside a class are not global variables, whereas those with a namespace are global. Data are put in memory one after the other, usually the size of a class
is >= than the sum of the sizeof the members, because of alignment, memory is accessed by word, a compiler will not start reading from 0, the memory
consumption depends also on the order, double int double is not like double double int, this is a detail. There are constructors and destructors, you
can have as many constructors as you want, but only one destructor, when you instantiate an object a constructor is invoked, which one depends on you, they
are defined as functions but do not have return types and have the same name as the class, as functions they can accept as many arguments as they want
or nothing, no arguments, a constructor that can be called without any argument is called default constructor, a destructor uses the ~ syntax. {} also calls
the default constructor, () is a compile error because the compiler thinks you are defining a function in that way (which is wrong), remember function
overloading, constructors are particular types of functions, the destructor is automatically called once the function goes outside of scope, there are

particular constructors that are defined by default by the compiler, since C++11 you have to define also a destructor, we will see later. The compiler
can define three default constructors + two other functions and other structures. The compiler can generate default behavior for you. I can call the
constructor with () and the arguments, it works but I recommend to stick with {} because it works with the default one. Let's see the syntax of the constructor,
small function should be defined inside a class, the fully qualified name is the same as for the function, there are two places where I can assign
values to variables, I have foo::foo(list of arguments), then I have :some_stuff and {}, so there are two places where I can assign values to variables, before {} where you define
the function, here you are calling the constructors of your variables, of your members, this class has those parameters and we have called their constructors, we can
use {} everywhere, this is why it is called universal and unique. So after : is for constructors, then you have to follow the same order of when you have declared the
variable if swap you get warnings. Once this called a segmentation fault for a master thesis, she had an undefined value, a random value, be consistent
for the order you initialize the variables. You cannot have variables and functions with the same name, it can be natural to define both function and
variable "size". The use of const is not mandatory and he will show us where they are, just conceptual. The destructor is defined with the ~. When I need
a constructor and what should I pass as an argument, if a class has something invariant then it is a good idea to pass to a constructor. With std::vector
there is a default constructor that builds a vector of size 0, I can invoke with {3, 4}, one of the few cases where different results, in this case we have
a vector of two elements, if (3, 4) we obtain a vector of length 3 with all elements of size 4. You cannot call functions after ::. Initialize the variables,
what is the difference. In this region we cannot have _s = s; we can have () or {}, he prefers {}, in this situation the variable is constructed with
the right value, with the = is first constructed with the right value and then changed, you cannot use {} in the {} because the variable has already been
constructed, so cannot use it within {}, inside the constructor I simply call the default one, I cannot do anything else. It would be nice to print those
objects, how can I do that? Operator overloading, what are they? Functions with particular names, called at particular times, to implement the << I have
to write operator<< (the name of a function), it takes as argument a reference and the name of a variable. The function can return the same guy. When we

write std::cout<<a; what happened the compiler did
std::cout.operator<<(std::cout, a), if we write std::cout << a <<
"hello"; we have the first
argument is what is returned by operator<< and the second is that
one, it is operator<<(operator<<(std::cout, a), "hello"), operators
are functions that
can be called in a more friendly way. Suppose I want to define the
difference of two Point objects, I can do with operator-(const Point
&p1, p2) and can
return a Point, this is what is called syntactic sugar. He is using
the & in operator<< because he needs to modify os such that it
contains what I
want to print on screen, so we should use a reference to a pointer,
in return because we want to return the same guy. If I do not put it
I get the same
output but I am less effective because I copy the same thing many
times, the last & because we do not know how big is my class, it can
be huge, if I
pass a reference I will be only 64 bits so I can be faster. Things
are constructed top-down and are destroyed bottom-up. Correction, if
you call with ()
you have undefined behavior. We are going to implement kind of the
std::array. a pointer that points to the actual array in the heap,
one public
constructor, that takes the size of the vector, constructs elements
that are pointers and so on, in the call to new we are acquiring
memory, resources,
to not have memory leak just call delete[] in the destructor, this
approach is called RAII, Resource Acquisition Is Initialization, the
const for size()
means that function does not modify my arguments, try to remove and
recompile, it is no more optional it is mandatory otherwise will not
compile. I can
define operator subscripting, two versions, the const is different
and is allowed by function overloading, I can define the operator <<
(put_to_), bad
news about pointers, operator overloading does not apply to
pointers, either I first dereference otherwise I need the full name
of the function. Of course
this does not apply to the references, study the file, try to remove
the const see why it does not compile and we will discuss on
Thursday. Default
constructors of built-in types and pointers do not do anything.