W01-Course Introduction & OOP Concepts

Intermediate Programming

Dr. Laura Climent

Personal Background

- Present: Lecturer in UCC
- 2019: Lecturer in CIT
- 2016: Postdoc in Insight, UCC
- 2013: Ph.D Computer Science in Universitat Politècnica de València
- 2010: M.Sc Artificial Intelligence and B.Sc in Computer Science, Universitat Politècnica de València

Dr. Laura Climent | Lecturer University College Cork (UCC) | Computer Science & Information Technology School
l.climent@cs.ucc.ie | https://sites.google.com/site/lauraclimentaunes/| 5924 | G.67

Teaching

Previous Teaching

1. **Linear Data Structures and Algorithms**
2. **Non-Linear Data Structures and Algorithms**

BSc and BSc Honors

3. **Decision Analytics**
4. **Research Methods**
5. **Information Analytics**

MSc

Teaching

1. **Data Analytics for Digital Humanities I**
2. **Intermediate Programming (OOP)**

} 1st semester

3. **Data Analytics for Digital Humanities II**
4. **To be defined**

} 2nd semester

Research

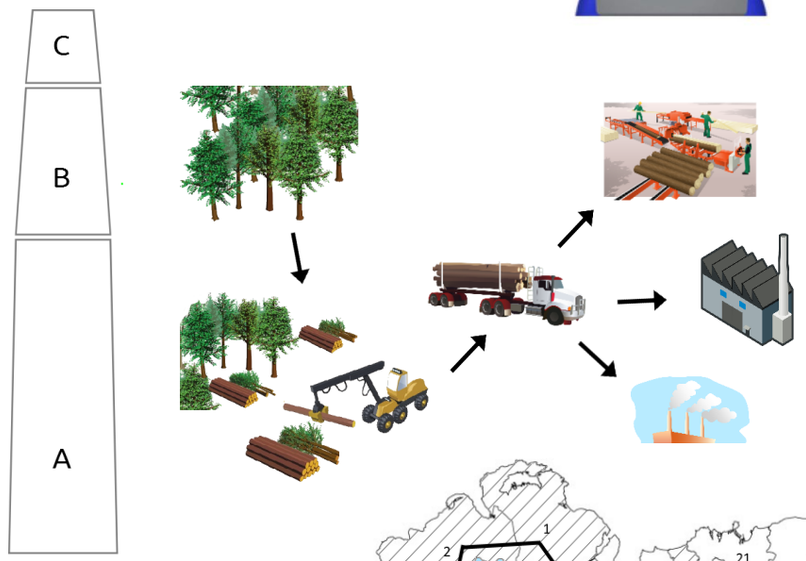
Research Experience

- Optimization & Decision Analytics:
  - Constraint Programming (CP).
  - Mixed Integer Linear Programming (MILP).
  - Metaheuristics.
  - Boolean Satisfiability (SAT).

- Data Science & Analytics:
  - Combine Data Analytics & Data Mining (DM) with decision analytics techniques.

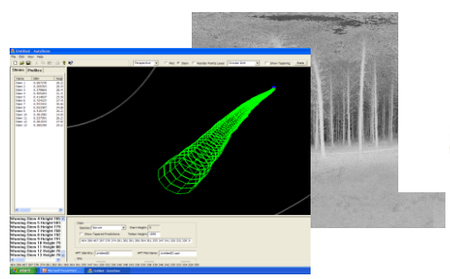
Sustainability Uncertainty

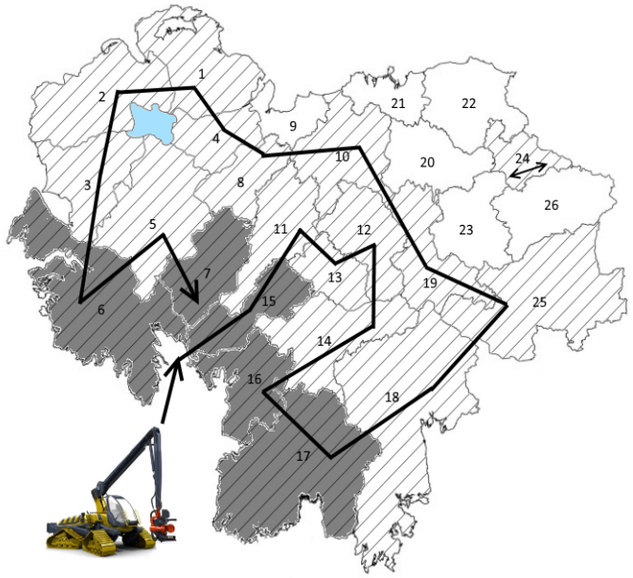
Research Applications



SMARTeBuses

TREEMETRICS





My website: https://sites.google.com/site/lauraclimentaunes/

Co-authors & PhD Supervision

Research Group IA-GPS

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

- Prof. Federico Barber
- Dr. Miguel A. Salido

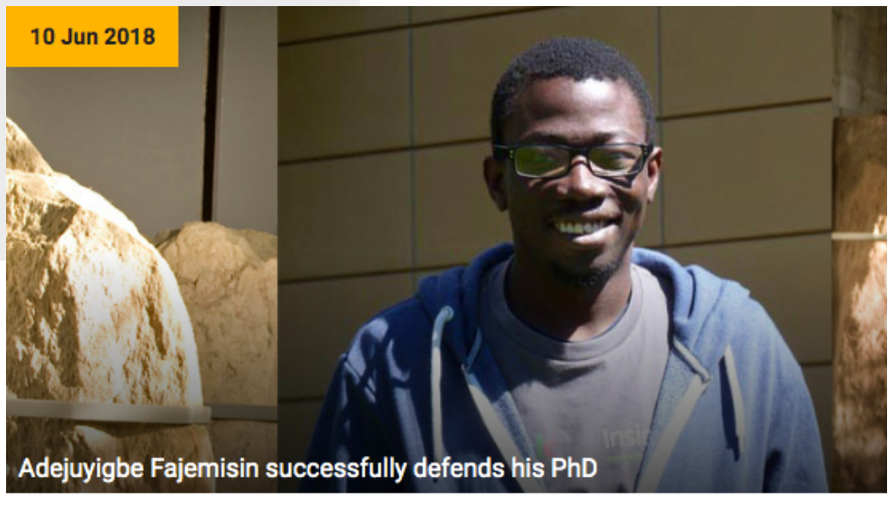
Insight Centre for Data Analytics

Insight

ucc
Coláiste na hOllscoile Corcaigh, Éire
University College Cork, Ireland

- Prof. Barry O'Sullivan
- Prof. Eugene Freuder
- Dr. Richard J. Wallace
- Dr. Steven D. Prestwich
- Adejuyigbe O. Fajemisin

Thesis title: *"An analytics-based decomposition approach to large-scale bilevel optimisation"*



- Insight

Group:

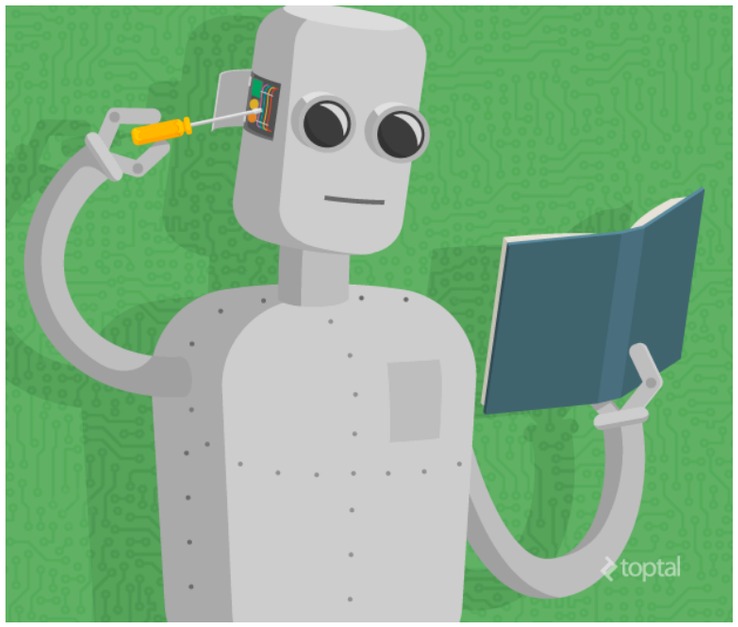
Decision Analytics & Optimization.

- crt-ai.ie

SFI Centre for research training in artificial intelligence.

Learning Outcomes

- Have a look to the module descriptor



What We Will Cover

- What is Object Oriented Programming (OOP)
- Advantages and Disadvantages of OOP
- Introducing Key Concepts of OOP

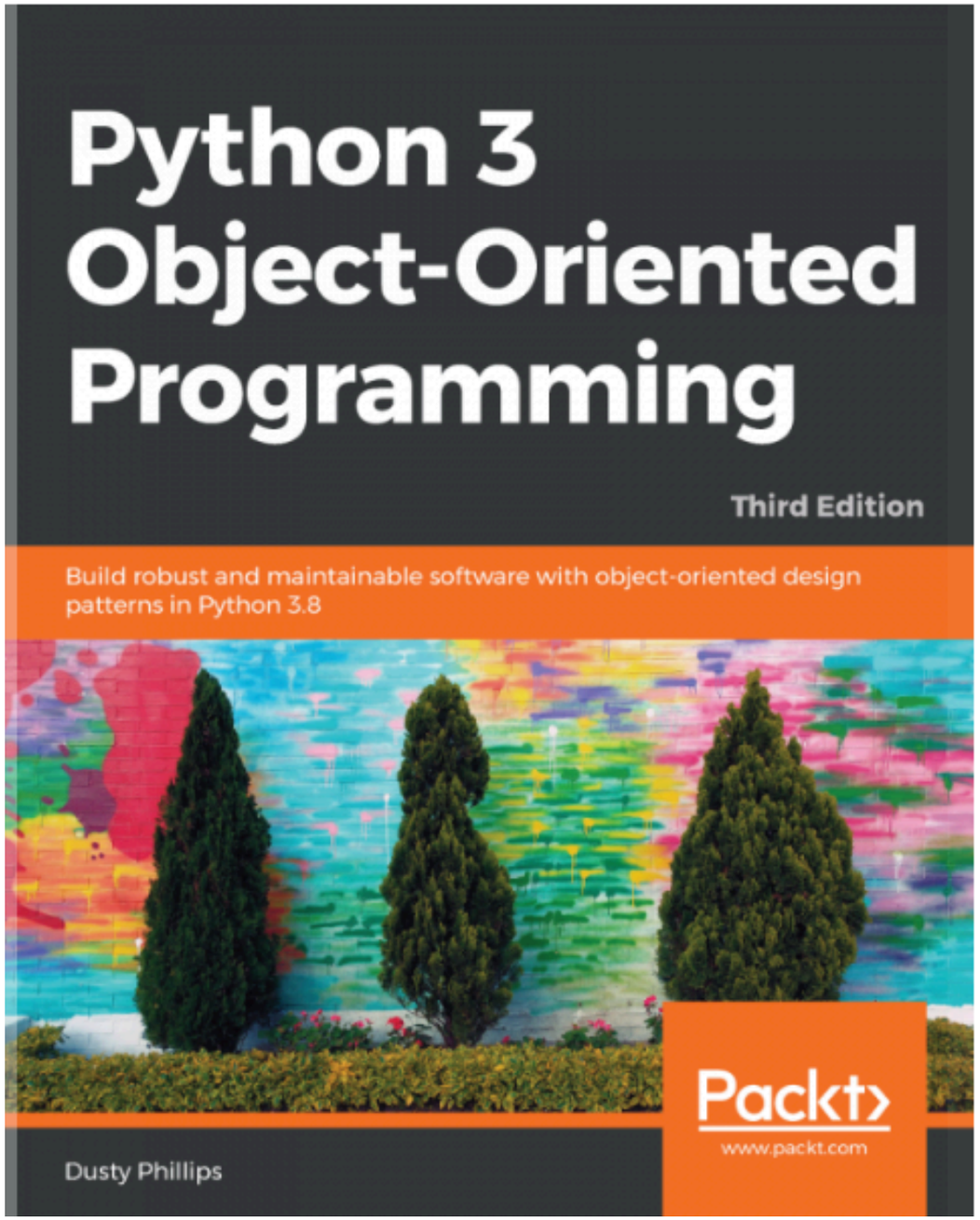
What We Will Cover

- Further features of the Python Programming Language
  - Classes and Object Oriented Programming with Python, including:
    - OOP Design
    - Encapsulation
    - Inheritance
    - Polymorphism
- Develop using a series of Python Libraries to develop Graphical User Interfaces (tkinter), System libraries and others
- Generators and Special Methods

Material

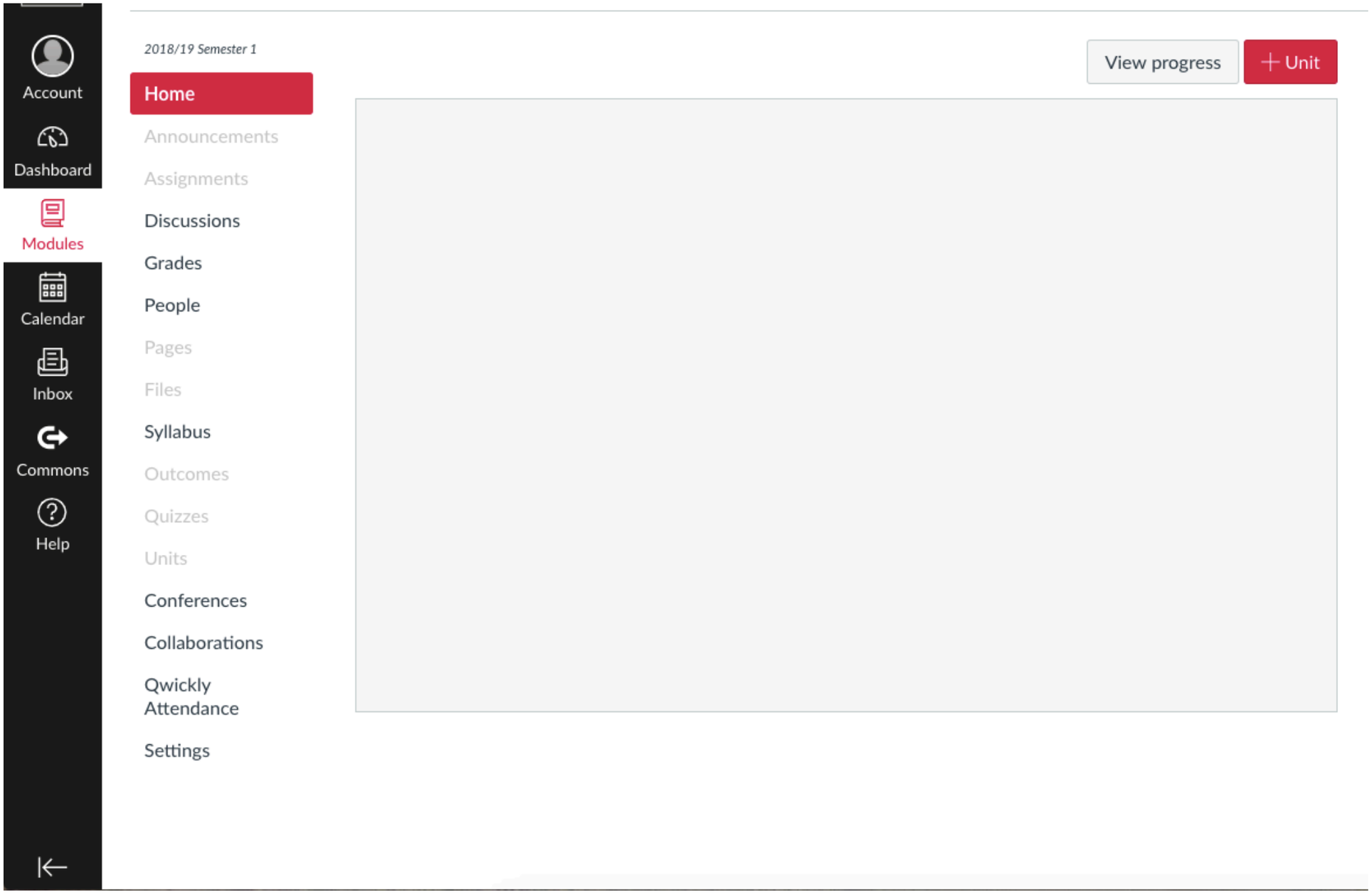
- Lecture notes (slides)
  - available via Canvas

- Additional material
  - book
  - useful links
  - code examples
  - notes written by me



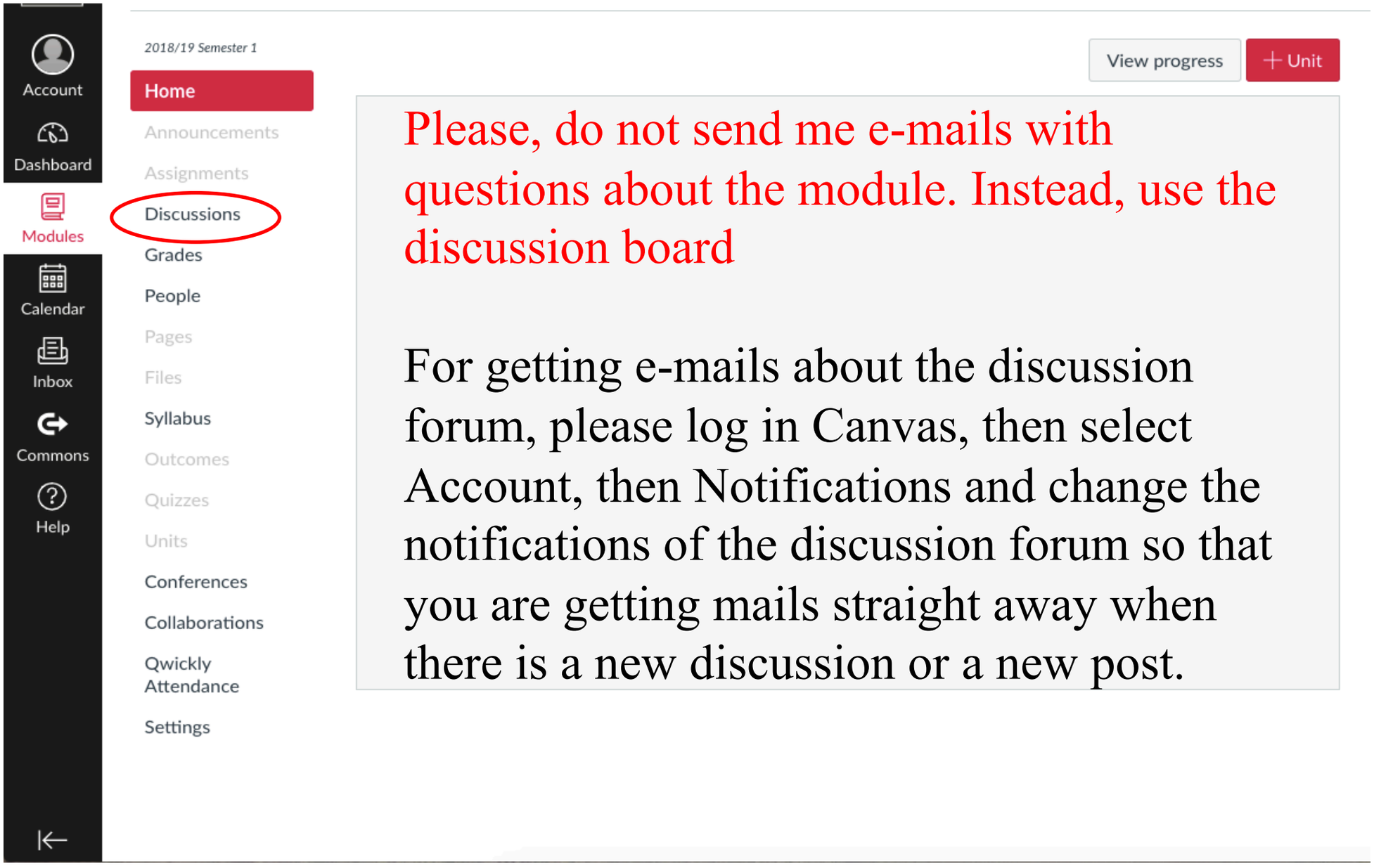
Methodology

- Have a look at least weekly to the CIT canvas



Methodology

- Use the discussion board to ask questions



Methodology

- 1h classroom session (twice per week):
  - Explanation of the concepts and application of them via code examples.
  - Dynamic interaction teacher/students regarding the code and concepts. Solving exercises during the class (5/10 min. for completing the exercise).

- 1h lab session (every week):
  - Exercises to be completed by the students. Solve individual doubts in time allocated for this purpose
  - 2 weeks prior to the deadline of each assignment the students could work on it.

Methodology

- Technical problems with software installation, connection, etc. please:

1. Check: http://www.cs.ucc.ie/help/

2. If you still could not solve your technical problem, contact: http://www.cs.ucc.ie/help/?page_id=183

Evaluation

Continuous assessment
100%

-1 x 35 marks assignment
-1 x 40 in-class test
-1 x 25 marks assignment

Evaluation

- Every student registered for a degree is expected to attend all lectures, tutorials, laboratory classes etc. A student will not be permitted to enter for an examination at the conclusion of a module, if attendance at the module is not considered satisfactory by the Registrar and Senior Vice President for Academic Affairs following a report by the lecturer concerned and/or Head of Department/Discipline/School responsible for that module. The decision of the Registrar and Senior Vice President is subject to the appeal of the Academic Council of the University.
  https://www.ucc.ie/en/cacsss/ug/currentstudents/problem_arise/

Evaluation

- If a lecturer suspects, you of plagiarism they will bring this to the attention of the Module Examination Board (MEB).

- A student gets **0 marks** for the **whole** assignment, if the student plagiarises/lets that somebody plagiaries from the student. This penalty applies as well if the plagiarism involves only some parts of the assignment rather than the whole assignment.

Evaluation

- Plagiarism is presenting someone else's work as your own. It is a violation of UCC Policy and there are strict and severe penalties.
- You must read and comply with the UCC Policy on Plagiarism www.ucc.ie/en/exams/procedures-regulations/
- The Policy applies to *all* work submitted, including software.
- You must expect that your work will be checked for evidence of plagiarism or collusion.
- If in doubt ask your module lecturer *prior* to submission. Better safe than sorry!

Motivation

The content of the module starts from here!

Motivation

- Why do we need more then?
- Can you think in a case/scenario ?

Once code gets complicated…

How do we leverage others' (reuse, extend and otherwise interact) code?

How do we package code so that others can use it?

How do we produce code so that it is easily maintained?

How to create libraries of code for yourself and others and ensure it is working correctly and robustly?

How do we organise large projects and allow several teammates to work together?

Advantages of OOP

- Admits parallel code development for groups of developers
- Helps us write cleaner code that has fewer errors
  - And when errors occur, our code is easier to fix
- Helps us to reuse code
- Allows us to use related code organisation and design techniques

Advantages from Industry Perspective

- Consider the costs involved in developing a couple software project:
  - Building and Fittings
  - Equipment
  - Utilities
  - Developers

Advantages from Industry Perspective

- Consider the costs involved in developing a couple software project:
  - Building and Fittings
  - Equipment
  - Utilities
  - Developers

100 Developers at 30k to 70k per annum is an expensive resource

Advantages from Industry Perspective

- Consider the costs involved in developing a couple software project:
  - Building and Fittings
  - Equipment
  - Utilities
  - Developers

Want to:

- maximise the time spent developing new code in parallel
- Reduce time spent fixing bugs (and the number of bugs!)

Disadvantages of OOP

- As an individual, it can be difficult to see the advantages of OOP

- There is more code with OOP, and OOP Design takes more effort

    - Not ideal for prototyping or developing a one off project

- But no one is saying every piece of code that we write in Python has to be object oriented

Object Oriented Programming (OOP)

- What are your first thoughts about what is OOP?
- Can you think in a case/scenario ?

OOP Fundamentals

- **Object Oriented Programming** is a programming paradigm in which you express to the computer the problem you want to solve by:

1. Modelling every concept of your problem via an object…

a car

a coffee

a ball

OOP Fundamentals

- Object Oriented Programming is a programming paradigm in which you <u>express</u> to the computer the problem you want to solve by:

2. Modelling the interactions of your problem via <u>object actions</u>…

- The ball bounced

- The coffee was poured

- The car got dirty.

Difference between Procedure Oriented and Object Oriented Programming

- Procedural programming creates a step by step program that guides the application through a sequence of instructions. Each instruction is executed in order.
- Procedural programming also focuses on the idea that all algorithms are executed with functions and data that the programmer has access to and is able to change.
- Object-Oriented programming is much more similar to the way the real world works; it is analogous to the human brain. Each program is made up of many entities called objects.
- Instead, a message must be sent requesting the data, just like people must ask one another for information; we cannot see inside each other's heads.

Featuers of OOP

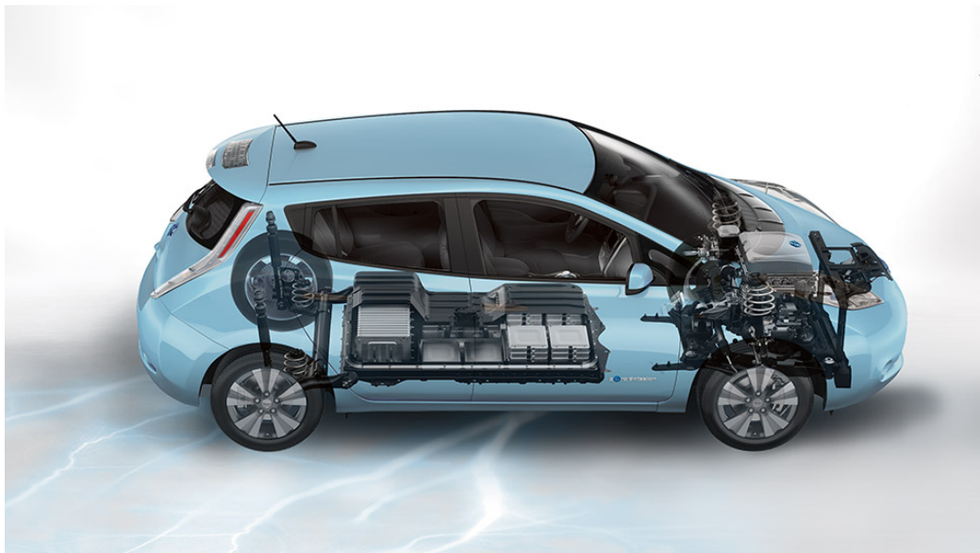
- Ability to simulate real-world event much more effectively
- Code is reusable thus less code may have to be written
- Data becomes active
- Better able to create GUI (graphical user interface) applications
- Programmers are able to produce faster, more accurate and better-written applications

Object-oriented programming (OOP)

- Object-oriented programming (OOP) seeks to define a program in terms of the *things* in the problem (files, molecules, buildings, cars, people, etc.), what they need, and what they can do.

OOP is a programming paradigm

"Class Car"

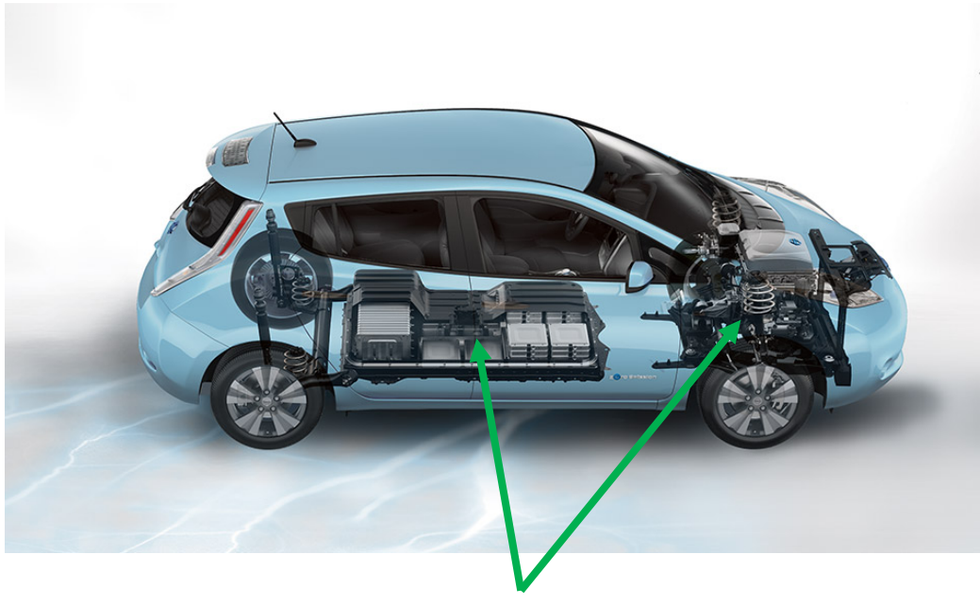


Object-oriented programming (OOP)

- OOP defines *classes* to represent these things.
- Classes can contain data and methods (internal functions).
- Classes control access to internal data and methods. A public interface is used by external code when using the class.
- This is a highly effective way of modeling real world problems inside of a computer program.

"Class Car"

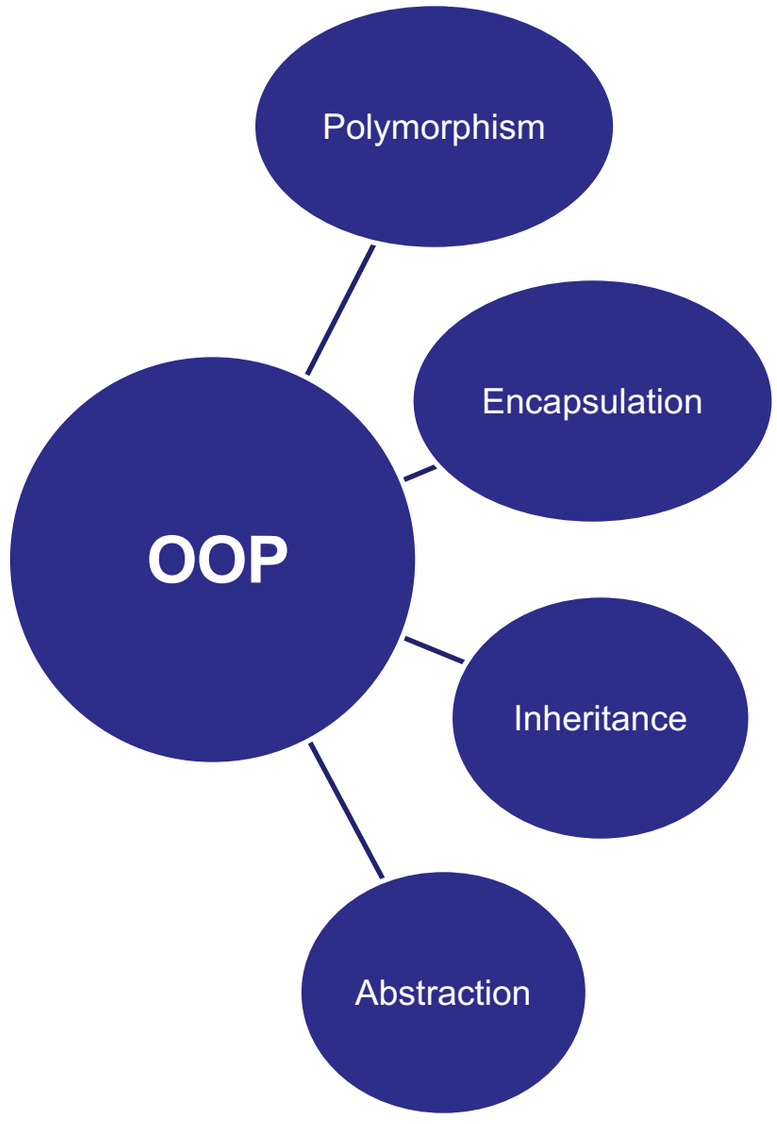
public interface



private data and methods

The formal concepts in OOP

- Object-oriented programming (OOP):
  - Defines *classes* to represent data and logic in a program. Classes can contain members (data) and methods (internal functions).
  - Creates *instances* of classes, aka *objects*, and builds the programs out of their interactions.
- The core concepts in addition to classes and objects are:
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Abstraction



Core Concepts

- Encapsulation
  - As mentioned while building the C++ class in the last session.
  - Bundles related data and functions into a class (wrap together)
- Inheritance
  - Builds a relationship between classes to share class members and methods
- Abstraction
  - The hiding of members, methods, and implementation details inside of a class.
- Polymorphism
  - The application of the same code to multiple data types

Abstract Data Type (ADT)

1. Constant & Variables: Storage location paired with an associated symbolic name, and containing some quantity of information referred to as a value.
   - Constant: Information is known and fixed
   - Variable: Information known or unknown and dynamic

Abstract Data Type (ADT)

2. Datatype: Classification identifying one of various types of data. It determines:
   i. The possible values this type can take.
   ii. The operations that can be done with values of this type.
   iii. The meaning of the data and the way values of that type can be stored.

- Example:
  - The set of values of the Boolean datatype are true and false.
  - The type supports most logical operations: &&, ||, !, etc.
  - The data represents a logical statement and it can be stored in a computer memory with the values 0 and 1 (using a single byte).

Abstract Data Type (ADT)

3. Data Structure: A container with a particular way of organising data (either contiguous or node-based structures).

- ❑ It allows to store a collection of variables. It is associated with functions, which allow the manipulation of the stored variables.
  - Static typing programming languages (e.g., C or Java):
    - All the elements of the collection must have the same datatype
  - Dynamic typing programming languages (e.g., Python):
    - Elements of the collection can have different datatypes

Abstract Data Type (ADT)

What is the key point of a Data Structure?

i. It provides a concrete representation for the data and its organisation.

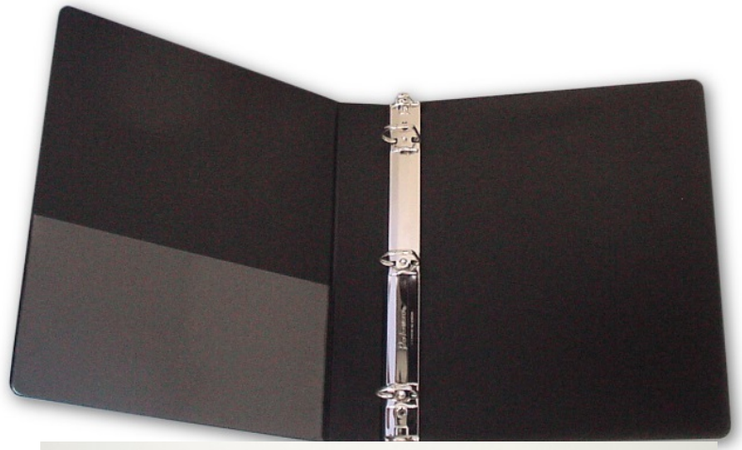
ii. Associated to the point of view of a software developer.

Associated with the *how*

Abstract Data Type (ADT)

- In our introductory example, both the notebook and the ring binder are data structures.
  - Two different kind of containers that we can use to store notes.
  - They are significantly different (example):

- The notebook does not require new pages until it is full.
- The ring binder requires inserting a new page every time you want to write something new.



Abstract Data Type (ADT)

4. Abstract Data Type: High-level definition of a data type.
   It defines the data type *declarative semantics*:
   i.e., imagine, we are using certain type of data:
   a) What does this data type represents for us?
   b) What are the possible values our datatype can take?
   c) What are the possible operations we can perform in a data of this type? What does each of these operations do?

Abstract Data Type (ADT)

What is the key point of an Abstract Data Type?

i. They are mathematical models of data types.

ii. Associated to the point of view of a software user.

Associated with the *what*

Abstract Data Type (ADT)

- In our introductory example, the dustjacket is an abstract data type.
  a) It represents our notes container for the module Linear Data Structures and Algorithms.
  b) It can contain n >= 0 pages of notes, one after another.
  c) We can add (1) buy a new dustjacket, (2) get how many pages of notes are there, (3) add a new page of notes, (4) remove a page of notes and (5) get a page of notes.

Abstract Data Type (ADT)

- In our introductory example, the dustjacket is an abstract data type.
  - Whether the dustjacket contains internally a notebook, a ring binder or other: We don't know and we do not care!
  - We know we can add new notes, remove some notes, see how many pages of notes are there, get a concrete page of the notes, etc. That's all we want!

Abstract Data Type (ADT)

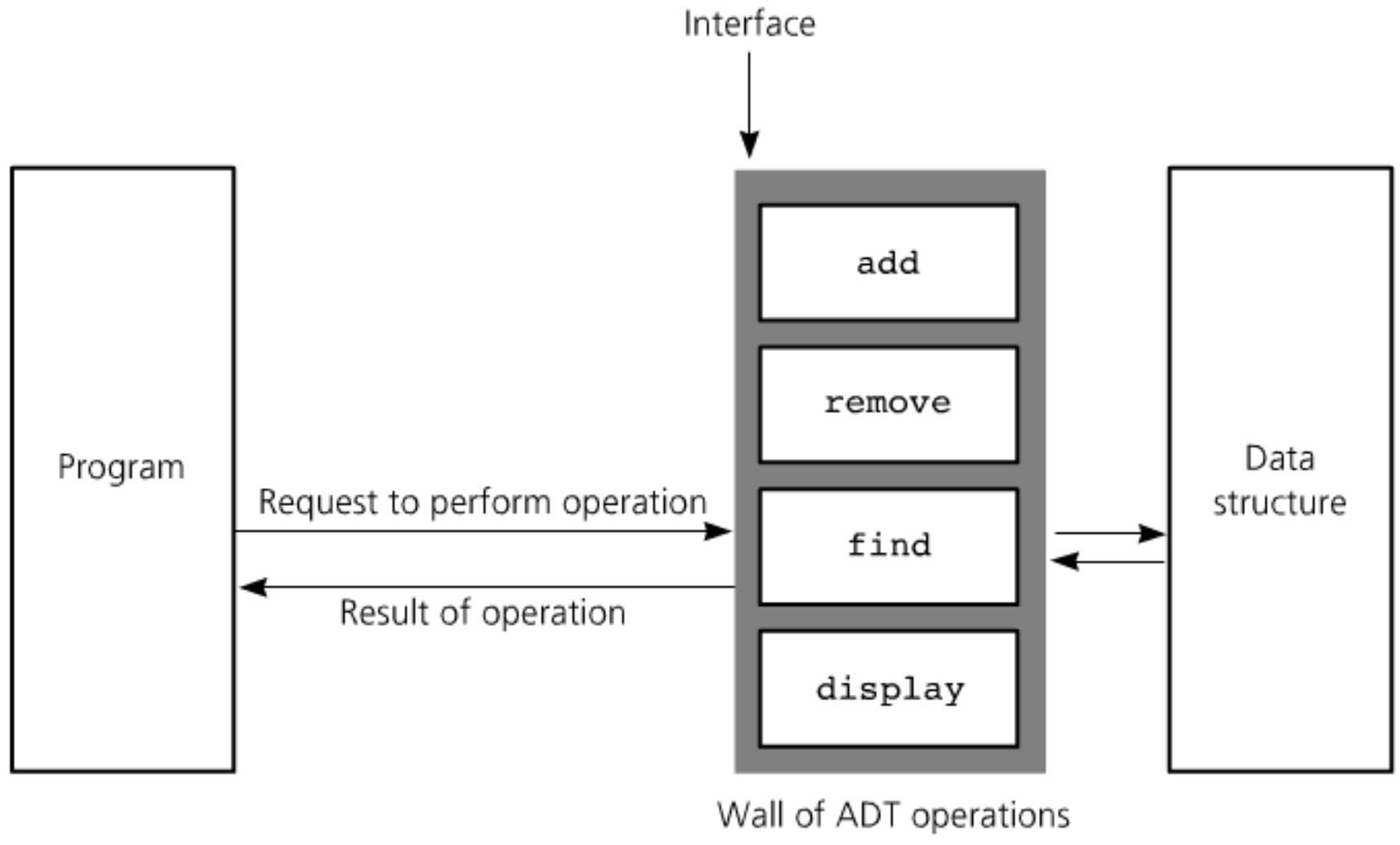
- An Abstract Data Type (from now on ADT) separates the:
  - Specification of the data (***what*** *operations can be performed*).
  - Implementation of the data (***how*** *are these operations actually implemented*).

Abstract Data Type (ADT)

- To do this separation, and ADT has two parts:
  1. Public / external part: User's view of the ADT or the description of the ADT:
     - How does the object looks like, how is it structured.
     - What operations can we do with it. This is the description of the ADT.
  2. Private / internal part: Developer's view of the ADT or the implementation of the ADT:
     - How is the data actually stored (data structure).
     - How are the operations actually implemented.

Abstract Data Type (ADT)

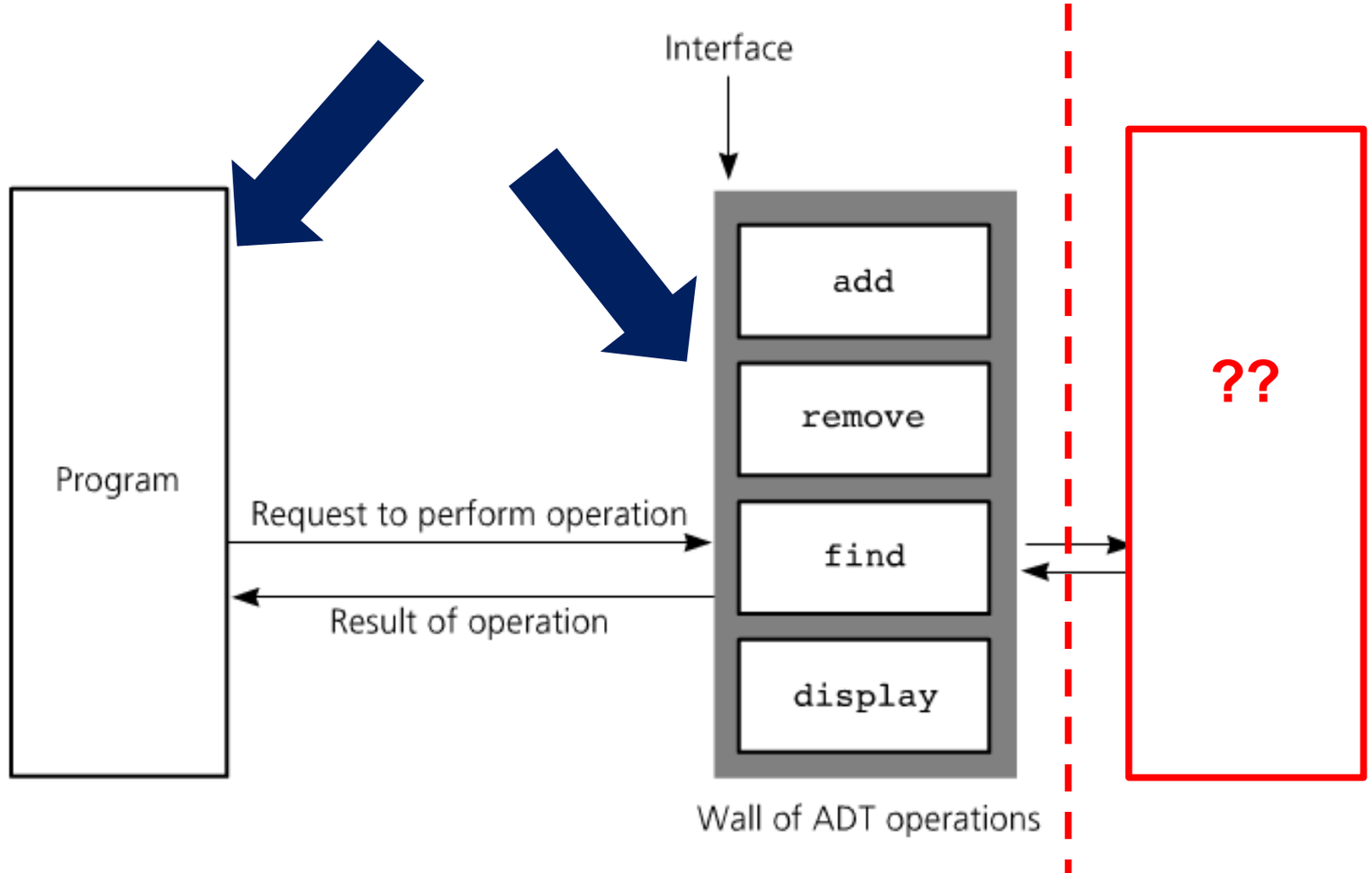
- An ADT provides an interface (a protocol of communication) between:
  - The data structure: Providing a concrete representation of the data.
  - The program user: Needs to make use of the data and its operations.



Abstract Data Type (ADT)

When you are defining the ADT public part (side of the interface)…
…you have to think of your future Software user!

- You describe the data type you are going to offer she/he.
- You describe the operations she/he can do with this data type.



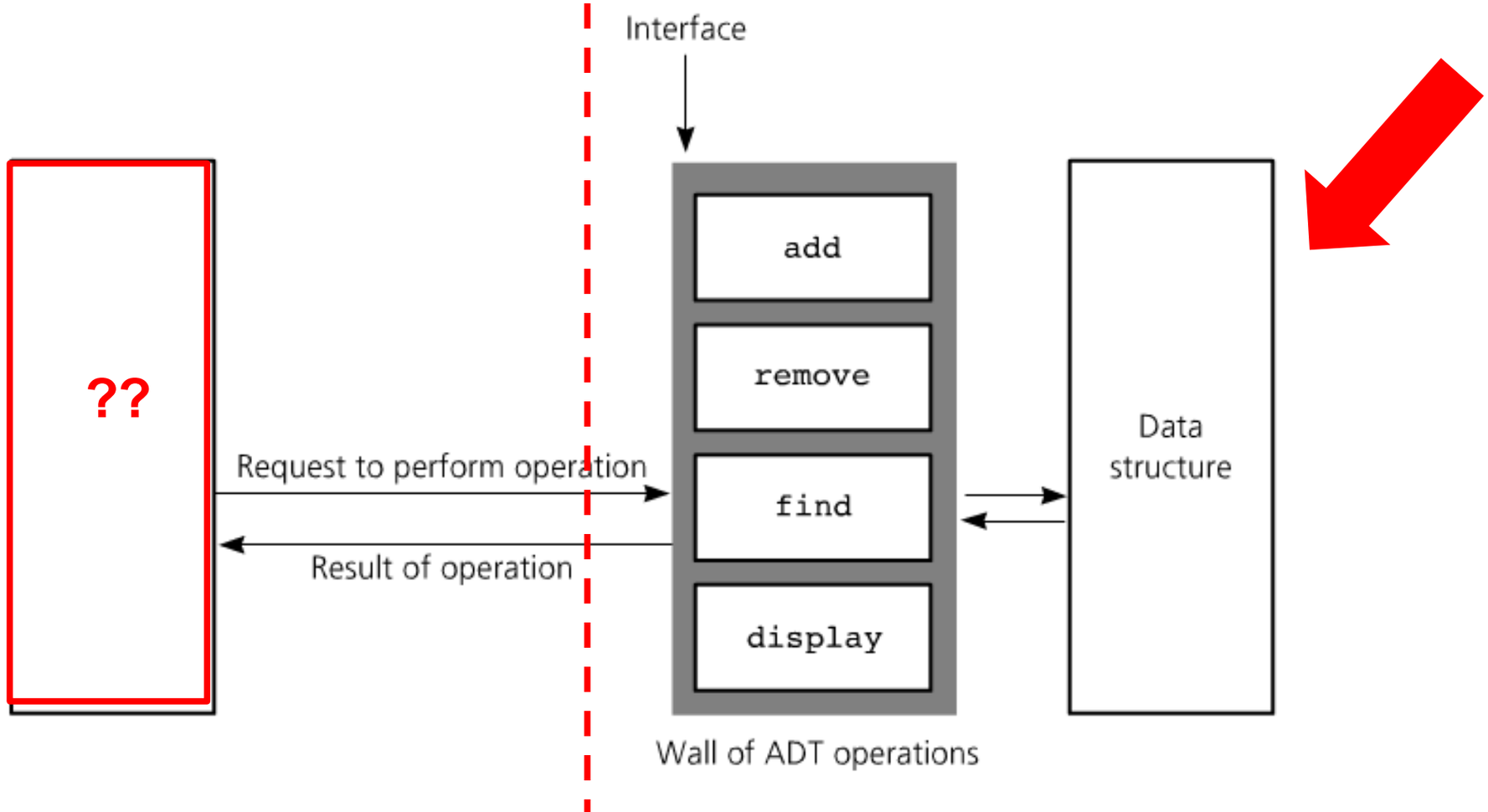
Abstract Data Type (ADT)

- The description of the operations must be:
  - Concrete enough, so as to:
    - Specify clearly how to call the operation (inputs/outputs).
    - Specify completely the operation's effect on the data type.
  - Generic enough, so as to:
    - Don't not specify how the data is stored (data structure).
    - Don't specify the algorithms that are performing the operation.

Abstract Data Type (ADT)

When you are defining the ADT private part (side of the interface)…
…you have to think as a software developer!

- You use / define concrete data structure(s) to represent the data.
- You use / define concrete algorithms to implement all operations.



Advantages of Using ADTs

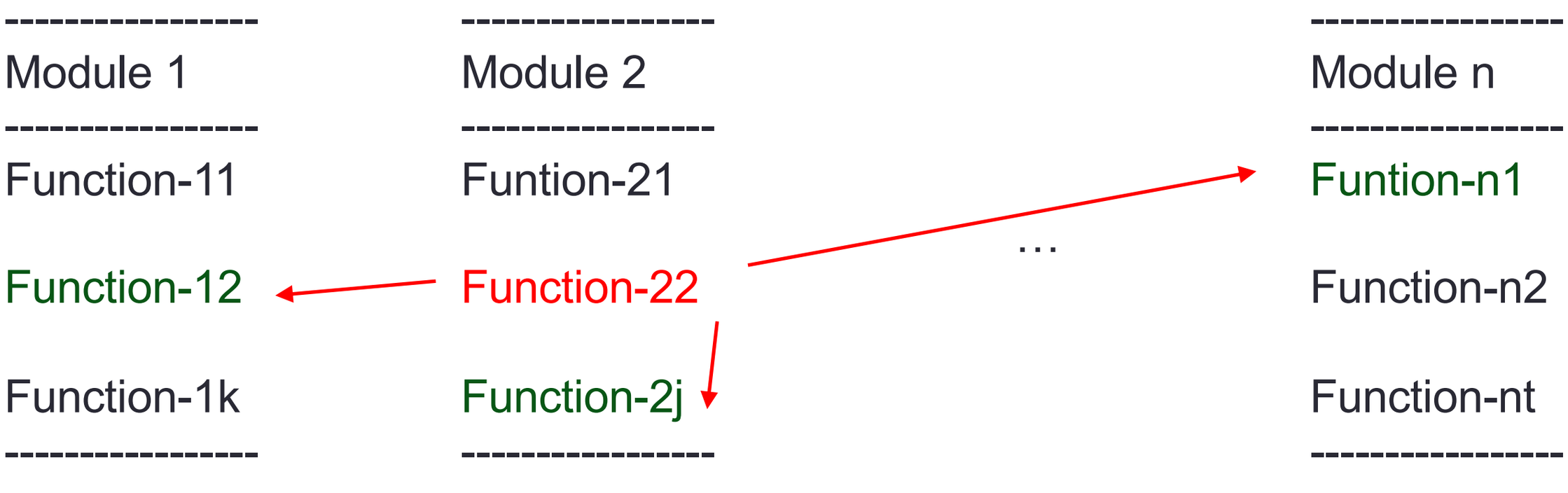
1. Helps modularity in programming:

- ❑ As the size and complexity of a program grows programming tasks become more difficult.
- ❑ In this context, the decomposition of the problem into multiple small and independent modules (modularity) allows to better model the problem domain.
- ❑ It also improves readability and maintenance of programs (code is easier to understand).

Advantages of Using ADTs

1. Helps <u>modularity</u> in programming:

- Big application, with multiple functionality.
  - If you have to change a functionality requirement and this functionality is isolated in a piece of code, then it is easy to locate it and modify it. Otherwise…nightmare!



Advantages of Using ADTs

2. Helps team programming:

- Several programmers working independently on their own modules before combining them into one big program or application.
- Programming team ‘A’ create the ADT MyData:
  - They specify the ADT public part: An abstraction of some data type and its operations. They release a manual describing the ADT.
  - They also implement the ADT private part: They use concrete data structures to represent the data and concrete algorithms to perform the operations.

Advantages of Using ADTs

2. Helps team programming:

- From that moment on, programming teams ‘B’, ‘C’, ‘D’, etc. can use MyData and its operations.
  - They don’t know how MyData is represented nor how the operations are done, but they don’t need to!

Advantages of Using ADTs

2. Helps <u>team</u> programming. Case study:

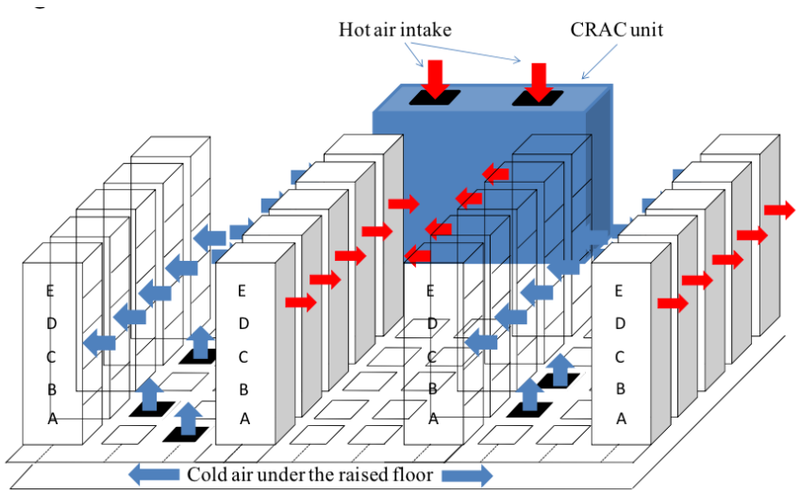
- ❑ European Union Project in Smart Cities GENiC:
  Globally Optimised Energy Efficient Data Centres
  - Green Computing
  - Sustainability.
  - Renewable energy sources
- ❑ 7 partners:
  - Architecture with more than 20 software components (modules).

Advantages of Using ADTs

2. Helps team programming. Case study:

❑ Our team:

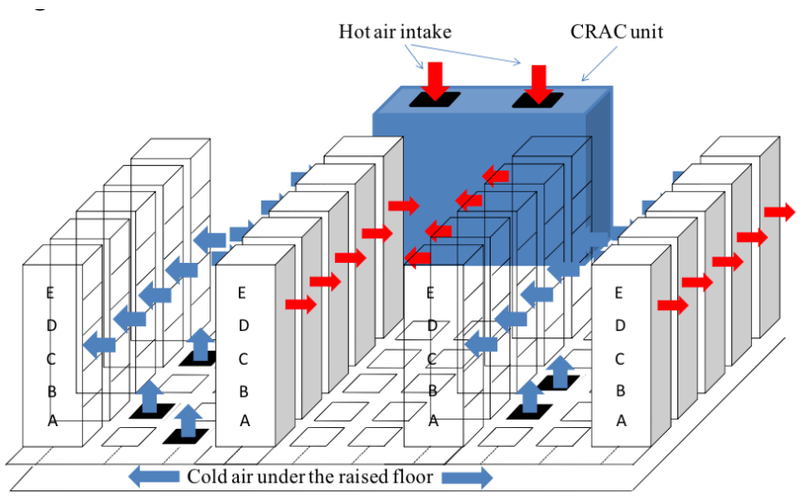
- Created a ‘Energy Optimisation Constraint Solver’.
- Basic operation: You pass it a DC configuration and it optimises the distribution of the Virtual Machines among the servers.
- Other operations: Temperature / Performance configurations, etc.



Advantages of Using ADTs

2. Helps team programming. Case study:

- ❑ Our team → We have no idea of DC configurations.
- ❑ Teams B → Know about DC settings (how many VMs predicted, etc), but have no idea of how to optimise the DC.
- ❑ No problem, we define the ADT **'Constraint Solver'** and its operation **optimise** and team B can use it with their DC settings.



Advantages of Using ADTs

2. Helps team programming. Case study:

- Finally, implementations of ADTs can be changed (e.g., for efficiency) without requiring changes to the program that uses the ADTs.
  - Imagine we make a new version the implementation of our ADT 'Constraint Solver' (because we have discovered a more efficient algorithm to optimise the energy of a DC).
  - If we re-implement our operation **optimise** but we maintain the same operation specification (in terms of inputs / outputs)…
    …Team 'B' can use our new version of the Constraint Solver without changing a single line of their code!!!

Python vs. Pure OO Languages

- Languages such as Java are pure OO languages. They enforce principles of OO strictly.

- Python doesn't require OO to be used.

- Python also emphasises freedom to use the language in any way. Developers implement encapsulation based on trust.

- Python is more OO in some ways than Java, allowing the user of Multiple Inheritance and other features.

Next Time:

- Classes