OIM3640 - Problem Solving and Software Design

Functions

How do we write code?

- What we have learned so far:
    - Basic understanding of programming language mechanisms
    - Ability to write separate files for different computations
    - Each file is some piece of code.
    - Code is composed of individual instructions.
- **Problems** with this approach
    - It may work for smaller problems but becomes messy for complex projects.
    - It is hard to manage details and keep track of information.
    - How do you know the right info is supplied to the right part of code?

Achieving Good Programming

- More code does not necessarily equate to good programming.
- A good programmer is evaluated by the **functionality** they deliver
- Introduce **functions**
  - An essential tool for **decomposition** and **abstraction**
- **Decomposition**: breaking down complex problems into smaller, manageable parts
- **Abstraction**: hiding complexity and presenting only essential information to the user.

Example - Projector

- A projector is a **black box**
  - You don’t know how it works
  - You only know the **interface**: input/output
  - You can connect any device that can communicate with the input
  - It somehow takes an image from the input source and projects it onto a wall, magnifying it
- **ABSTRACTION IDEA**: Understanding how the projector works is not necessary for its usage.

Example - Projector(s)

- Projecting (very) large image for Rio 2016 Opening Ceremony
  - Decomposition was achieved by dividing the task among multiple projectors
  - Each projector took input and produced a separate output
  - All projectors worked together to create the larger image
- **DECOMPOSITION IDEA**: By dividing the task among multiple devices, the end goal is achieved efficiently.

Apply these Ideas to Programming

- **DECOMPOSITION**
  - Break a complex problem into different self-contained pieces
  - A **self-contained** piece in programming refers to a component or module of code that can function independently.
- **ABSTRACTION**
  - Hides the details of a method's implementation, allowing the user to focus on the result of the computation.

Create Structure with DECOMPOSITION

- In previous example, we use separate devices.
- In **programming**, we divide code into modules.
  - They are **self-contained**
  - They help to break up the code into **smaller**, more manageable parts
  - They are designed to be **reusable**, making the code more efficient
  - They help keep the code **organized** and **coherent**
- In this lecture, we achieve decomposition through the use of **functions**.
- Later in the course, we will further explore decomposition using **classes** (in OOP).

Suppress Details with ABSTRACTION

- In previous example, there is no need to know how to build a projector.
- In **programming**, we view a piece of code as a **black box**
    - The internal details are **hidden** and not visible.
    - It is **not necessary** to know the internal details.
    - The focus is on the **input/output** and desired **result**.
    - Abstraction helps to hide tedious coding details
- We achieve abstraction with **function specifications** and/or **docstrings** which outline the purpose and usage of the code.

Functions

- **Functions** are reusable pieces or chunks of code in a program
- They are not executed until they are "**called**" or "**invoked**" in the program.
- Function characteristics:
    - A **name**, to identify and call the function
    - **Parameters** (0 or more), to provide input to the function
    - A **docstring** (optional but recommended), to describe the purpose and usage of the function
    - A **body**, containing the code to be executed when the function is called.

> Civilization advances by extending the number of operations we can perform without thinking about them.
>
> - Alfred North Whitehead (British mathematician and philosopher, 1861–1947)