APS106

UNIVERSITY OF TORONTO

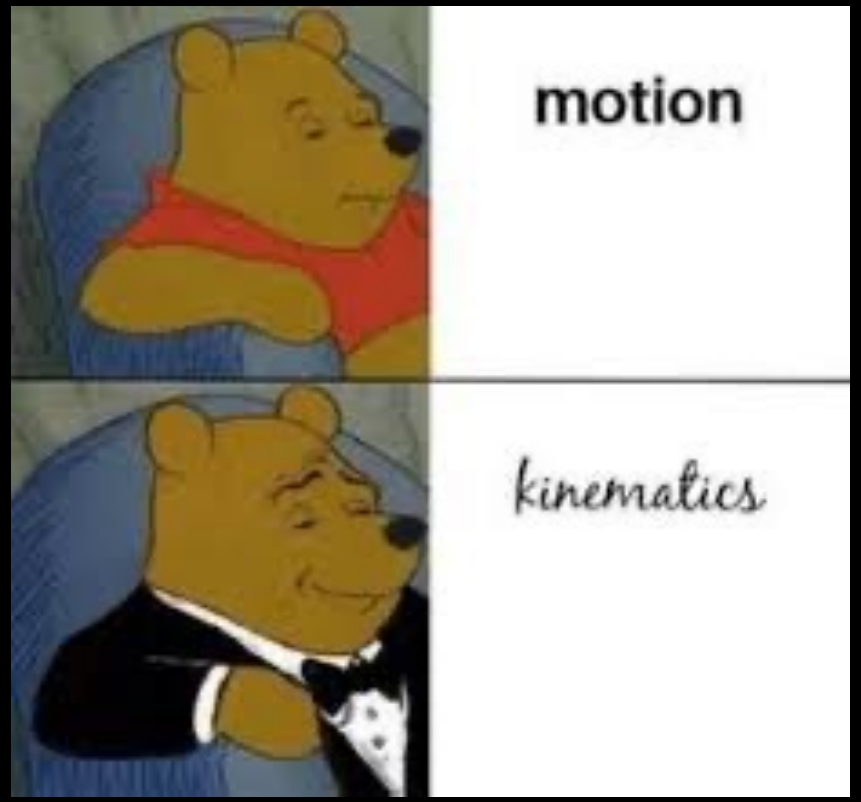
Design Problem 1: Forward Kinematics

Week 1 | Lecture 3 (1.3)

if nothing else, write #cleancode

Today's Content

- **Lecture 1.3**
  - **Engineering design**
  - **Design Problem: Forward Kinematics**



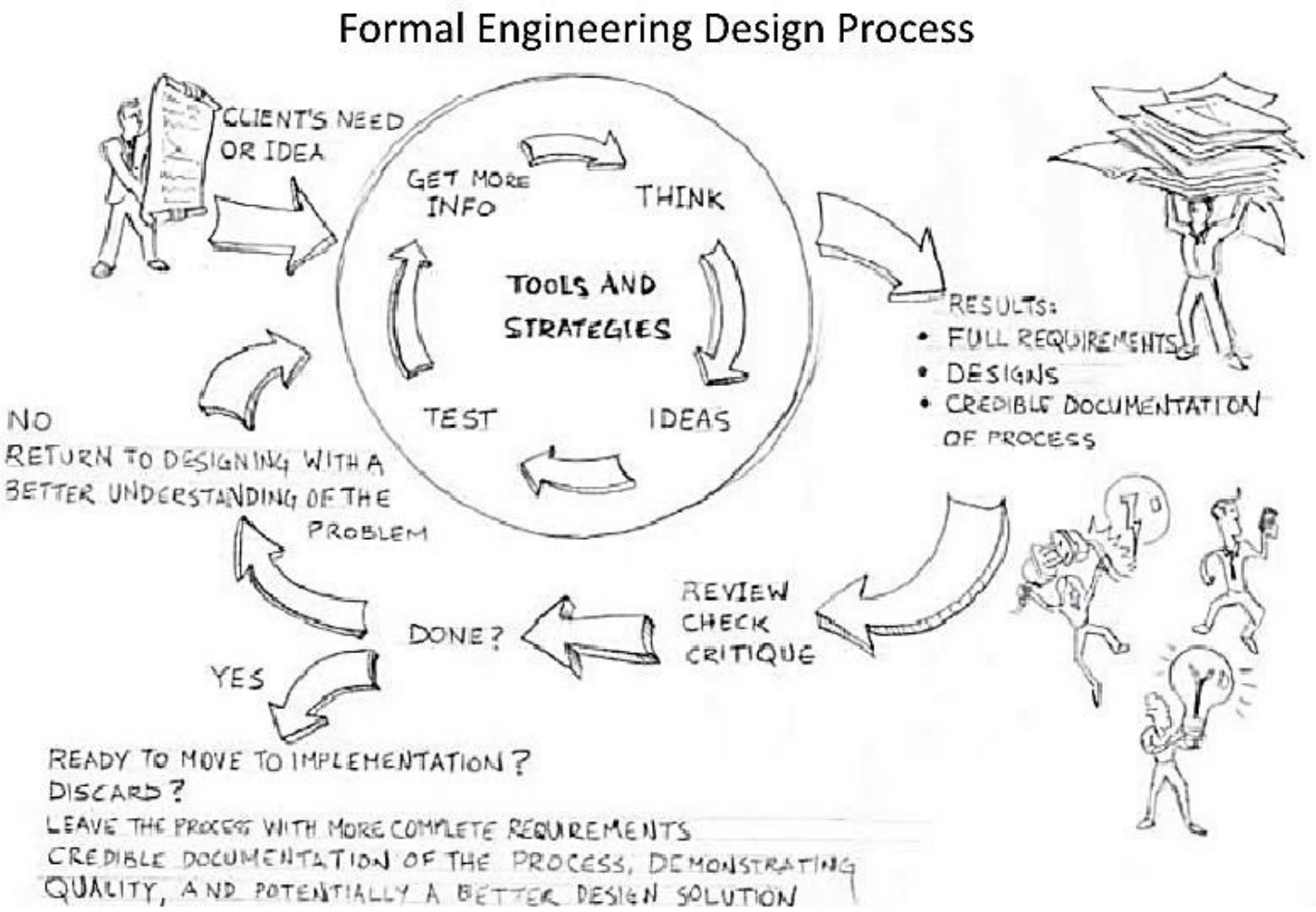
A Design Process for Programming

- APS111/112 - the design, implementation, testing, and documentation of technology to solve a particular problem.
- **Programming** is the design, implementation, testing, and documentation of a piece of software that solves a particular problem.
- The software might be part of a larger system (e.g., the avionics software of an aircraft, the accounting or human resources software of a business), but it represents the solution to a design problem (or part of a design problem).

A Design Process for Programming

- We will approach programing as an engineering design process and adapt the process you have already seen in APS111/112.

Taken from: Designing Engineers: An Introductory Text



A Design Process for Programming

- In the next lecture, we are going to talk about a detailed design process for programming, based on the engineering design processes that are key to any engineering.
- The steps are as follows:
- **Define the Problem.**
- **Define Test Cases.**
- **Generate Multiple Solutions.**
- **Select a Solution.**
- **Implement the Solution.**
- **Perform Final Testing.**

A Design Process for Programming

- **Define the Problem.**
- Develop a clear and detailed problem statement.
- Be clear on what needs to be done.
- Sometimes the problem will be easy enough (especially as you are learning programming) that the initial problem statement given by the client/prof is sufficient.
- More often, the problem is complex enough that forming a complete, explicit definition is a challenge itself and sometimes (even, often) the client doesn't really understand the problem him/herself.
- In such cases, research and iteration with the client is necessary.

A Design Process for Programming

- **Define Test Cases.**
- Work out specific test cases for which you know the answer.
- This will help in the solidifying the problem definition and provide you with tests once you have working code.
- Try to cover a reasonable span of possible cases that may come up.
- Think about strange cases that might break the code.
- Think about reasonable measures of efficiency, speed, and memory size.

A Design Process for Programming

- **Generate Many Creative Solutions.**
- Think about solutions and write them down. Try to be as creative as possible.
- A "solution" at this stage is two things:
  - An Algorithm Plan
  - A Programming Plan

A Design Process for Programming

- **Generate Many Creative Solutions.**
- Think about solutions and write them down. Try to be as creative as possible.
- A "solution" at this stage is two things:
  - **An Algorithm Plan**
  - A list of a few (from 4 or 5 to a dozen) steps that your algorithm will execute to solve the problem.
  - These are high-level steps that can correspond to many lines of code.
  - In real projects, these steps will themselves be subject to the design process (i.e. they will in turn be broken down into sub-steps perhaps may layers deep).

A Design Process for Programming

- **Generate Many Creative Solutions.**
- Think about solutions and write them down. Try to be as creative as possible.
- A "solution" at this stage is two things:
  - **A Programming Plan**
  - A list of steps you will take in programming the algorithm.
  - Sometimes this will be the form of programming, testing, and debugging each of the algorithm steps in order.
  - But it doesn't have to be that way.
  - Especially for larger systems, the algorithm steps may be designed and implemented by different people in parallel or you may choose to program, test, and debug the hardest step first to make sure you understand the problem enough.
  - Or you may decide to do the easiest steps first.

A Design Process for Programming

- **Select a Solution.**
- Evaluate the algorithm and programming plans you have generated.
- Does it appear that this solution will truly solve the problem?
- You may write some prototype code to understand if particular design ideas will work.
- Pick the best solution.
- If it is good enough, continue to Step 5, otherwise return to an earlier step (maybe even Step 1 as you have uncovered new parts of the problem definition).

A Design Process for Programming

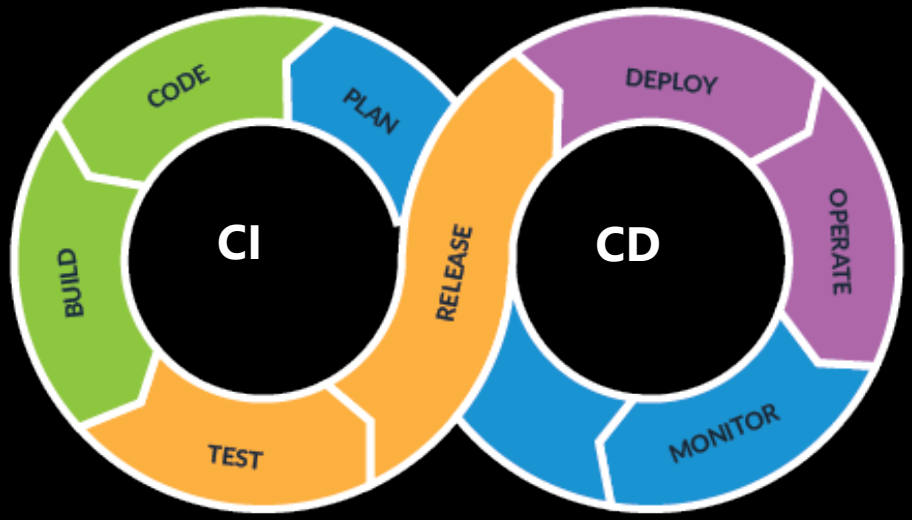
- **Implement the Solution.**
- Follow your chosen programming plan to implement the code.
- For each step in your programming plan, you should ensure that the code is working: it runs some "sub-tests" correctly.
- Even though it doesn't solve the whole problem, it should produce intermediate results that you can verify are correct.
- If it doesn't, you should debug it before moving onto the next step.
- Implementation includes the documentation in the code: functions should have well-written docstrings and comments should be used – it is better to over-comment than under-comment.

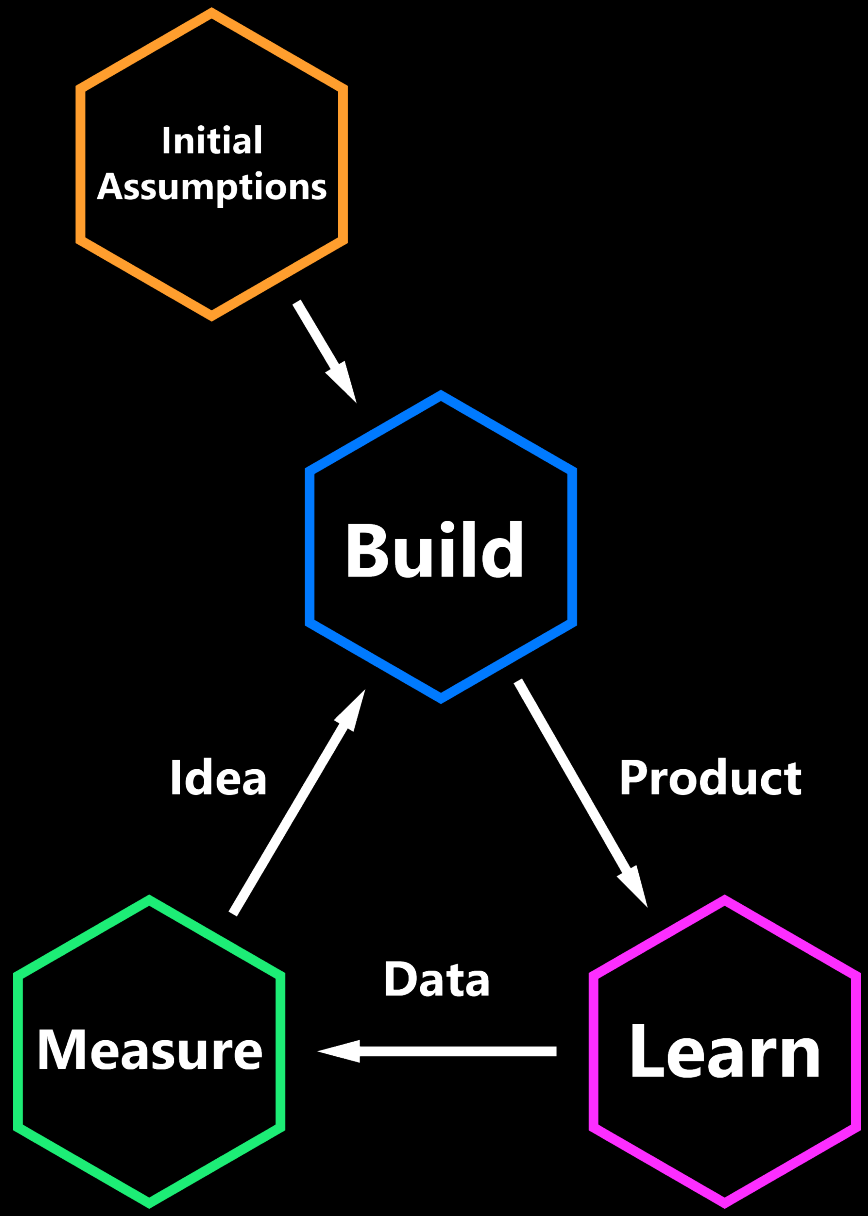
A Design Process for Programming

- **Perform Final Testing.**
- Evaluate the solution against the test metrics, ensuring everything is in order.
- If the solution is not satisfactory, you need to either return to Step 5 to debug the code or return to Step 1 to develop a better understanding of the problem.

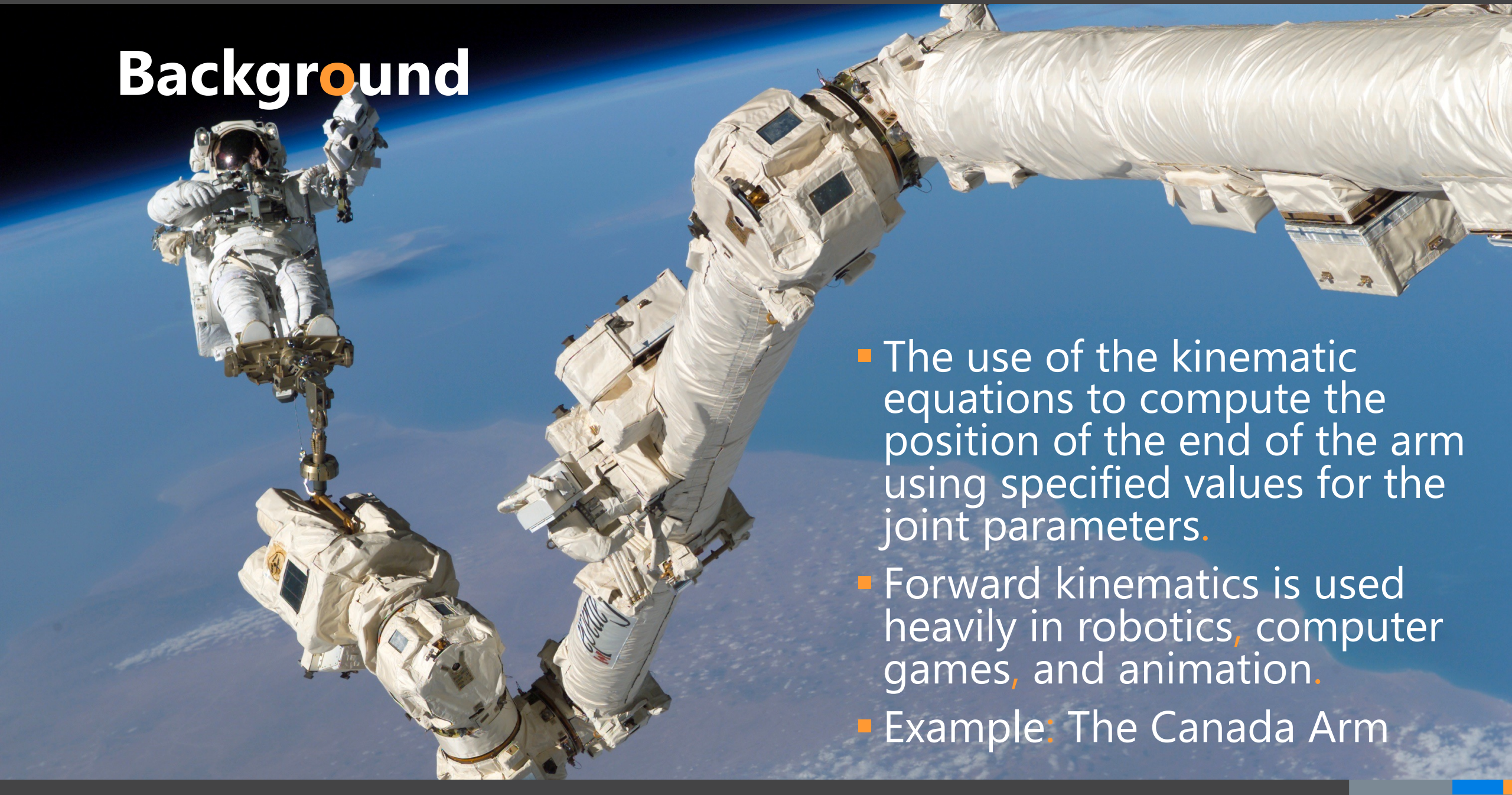
Design is ITERATIVE!

- Product Development **101**





Background



- The use of the kinematic equations to compute the position of the end of the arm using specified values for the joint parameters.
- Forward kinematics is used heavily in robotics, computer games, and animation.
- Example: The Canada Arm

The Canadarm

Design Problem 1: Forward Kinematics

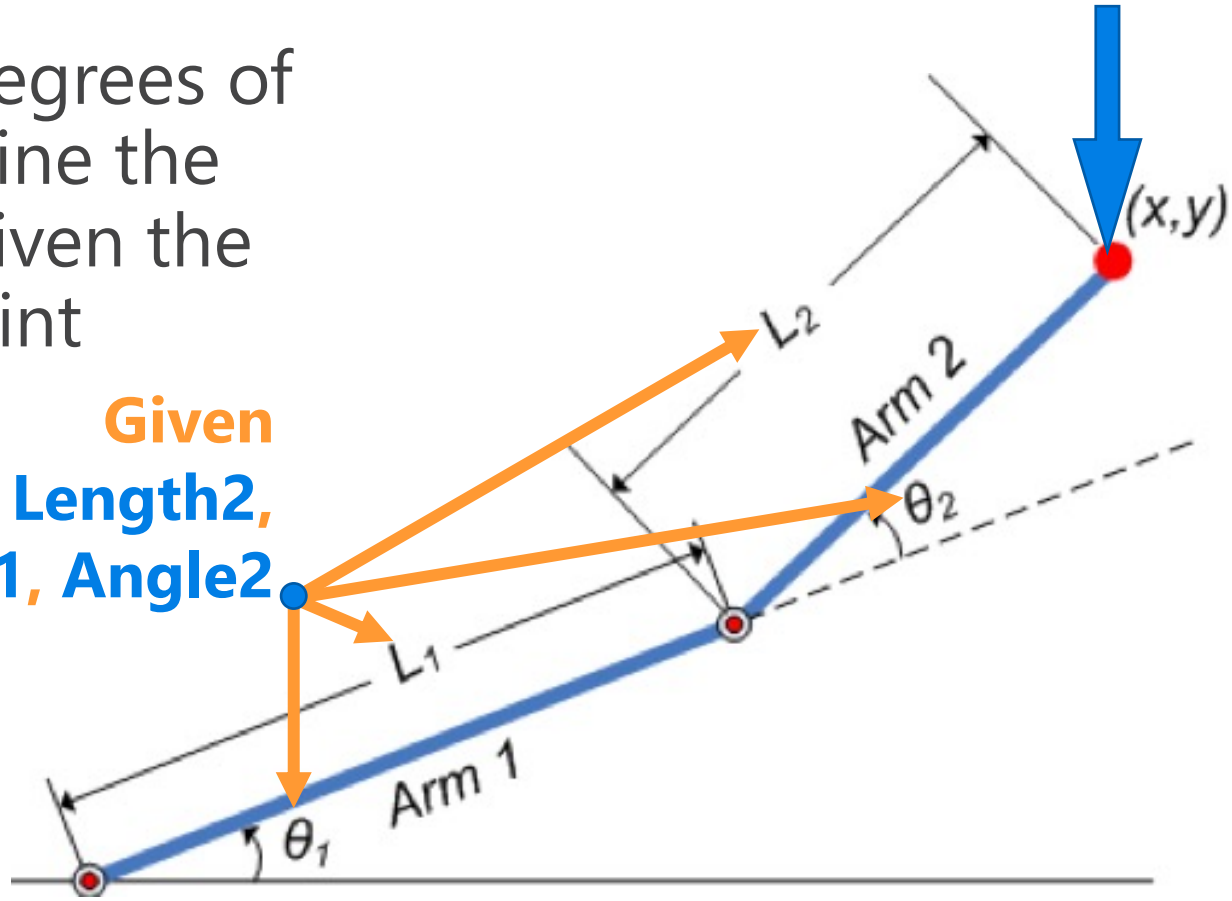
- **Problem Background**
- If you have a robotic arm (e.g., the Canadarm) with joints, it is important to be able to calculate where the end of the arm.
- Forward kinematics is the use of the kinematic equations of a robot to compute the position of the end of the arm from specified values for the joint parameters.

Design Problem 1: Forward Kinematics

- **Problem Background**
- Given a robotic arm with two degrees of freedom (see diagram), determine the position **(x, y)** of the effector given the component-arm lengths and joint angles.

Where is this point?

Given
Length1, Length2,
Angle1, Angle2



A Design Process for Programming

- Let's walk through the design process for programming in the context of the project.

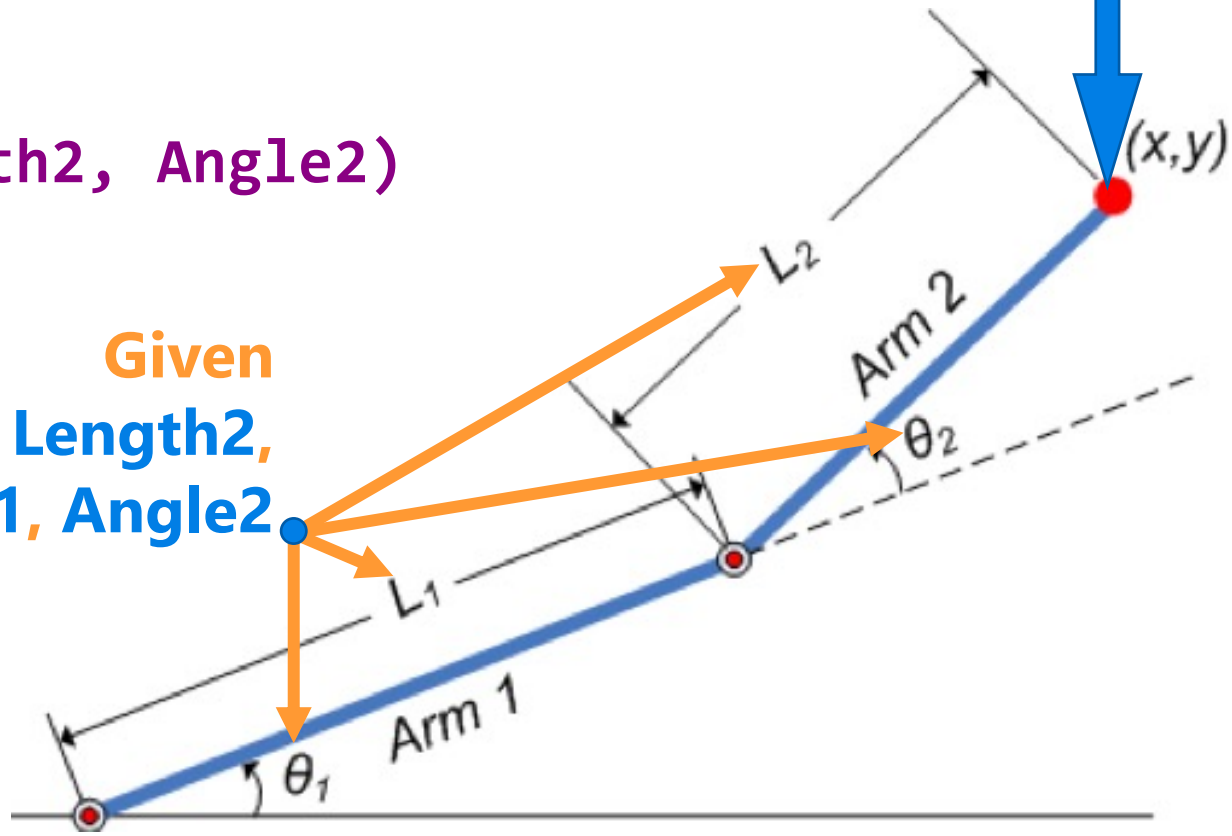
- Define the Problem.
- Define Test Cases.
- Generate Multiple Solutions.
- Select a Solution.
- Implement the Solution.
- Perform Final Testing.

Design Problem 1: Forward Kinematics

Where is this point?

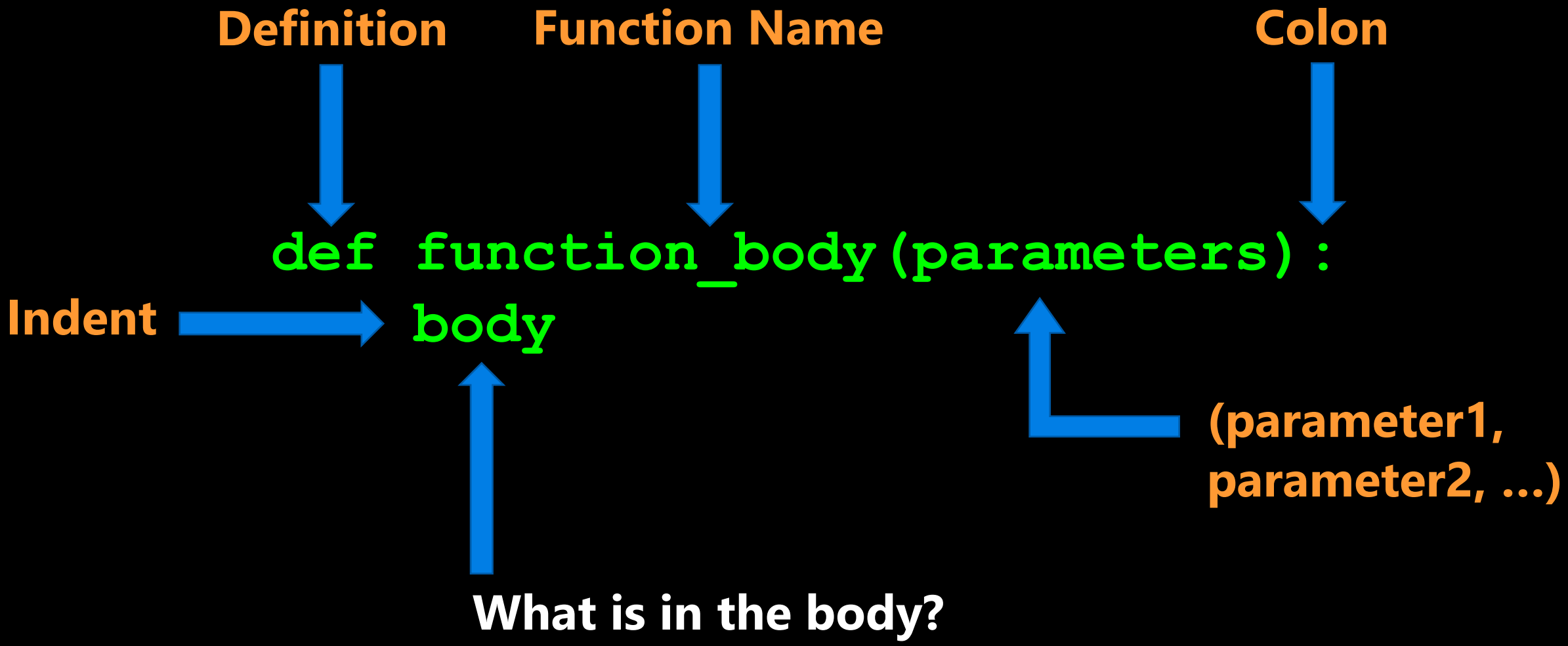
```
my_function(Length1, Angle1, Length2, Angle2)
```

Given
Length1, Length2,
Angle1, Angle2



Some reminders.

Function Definitions



Function Definitions

```
def function_body(parameters):
```

1. """DOCSTRING""" (optional)

2. Code that does the thing

3. return [expression]
   The return statement is optional and if it is not included, it's the same as writing return None

Calling Functions

- The general form of a function call:

```
function_name(arguments)
```

- Terminology
  - *argument*: a value given to a function.
  - *pass*: to provide an argument to a function.
  - *call*: ask Python to execute a function (by name).
  - *return*: give a value back to where the function was called from.

In **Python** names of variables and functions use low case and underscores.

```
function_name
Function_Name
FunctionName
```

Input

- Python has a built-in function named **input** for reading text from the user.
- The general form of a **input** function call:

```
input(argument)
```

- The argument is the text you want displayed to the user.
  - *"What is your name?"*
- The value returned by the **input**function is always a string.

Importing Functions and Modules

- The general for of an import statement is:
  - `import module_name`
- To access a function within a module:
  - `module_name.function_name`

APS106

UNIVERSITY OF TORONTO

Design Problem 1: Forward Kinematics

Week 1 | Lecture 3 (1.3)

if nothing else, write #cleancode