Functions

Idea: Helps us build bigger programs by collecting code into re-useable units

Real Life Examples:

Capsule Coffee Maker - place a capsule and water (input) and get coffee (output)

Vending machine - place money and set choice (input) and get a treat (output)

Functions should perform a clearly defined, specific task

“Define once and use forever!”
Functions

Good functions act like a **blackbox** - the user doesn’t need to know how the function works to use it.

Functions are **abstractions**: they abstract the details so we can focus on the big picture.

Functions allow us to write **modular** code. Modular code is organized in clearly defined sub-components. Each sub-component can be designed, implemented and tested independently.

Analogies: A car consists of independent modules such as lights, steering column, and brakes. A book consists of modules which build up such as sentences, paragraphs, sections, and chapters.
Function syntax

Syntax:

def <name>(<param1>,<param2>,...,<paramN>):

<body>

return <value>

parameters, or arguments, or inputs. You can have any number of these, including none!
Aside - Terminology

Programmers use the terms `void`, `None`, and `NULL` to indicate nothing.

Ex: a function with no return value is sometimes called a `void function`.

Ex. Python3 defines a special datatype called `NoneType` to represent variables that have nothing inside them.
Function examples

You’ve been using functions already: min(), Math.sqrt(), main(), len(), input()

But you can also define your own!
Advantages of functions

1. Re-useability - “define once and use forever”
2. Modularity - “top-down design”
   a. Split big problems into small, easy-to-solve problems
3. Easier to debug and maintain
   a. Cut & paste => bugs have to be fixed everywhere. Code in a function only has to be fixed once
4. Abstraction = “black box”
   a. Users do not need to know how it works in order to use it