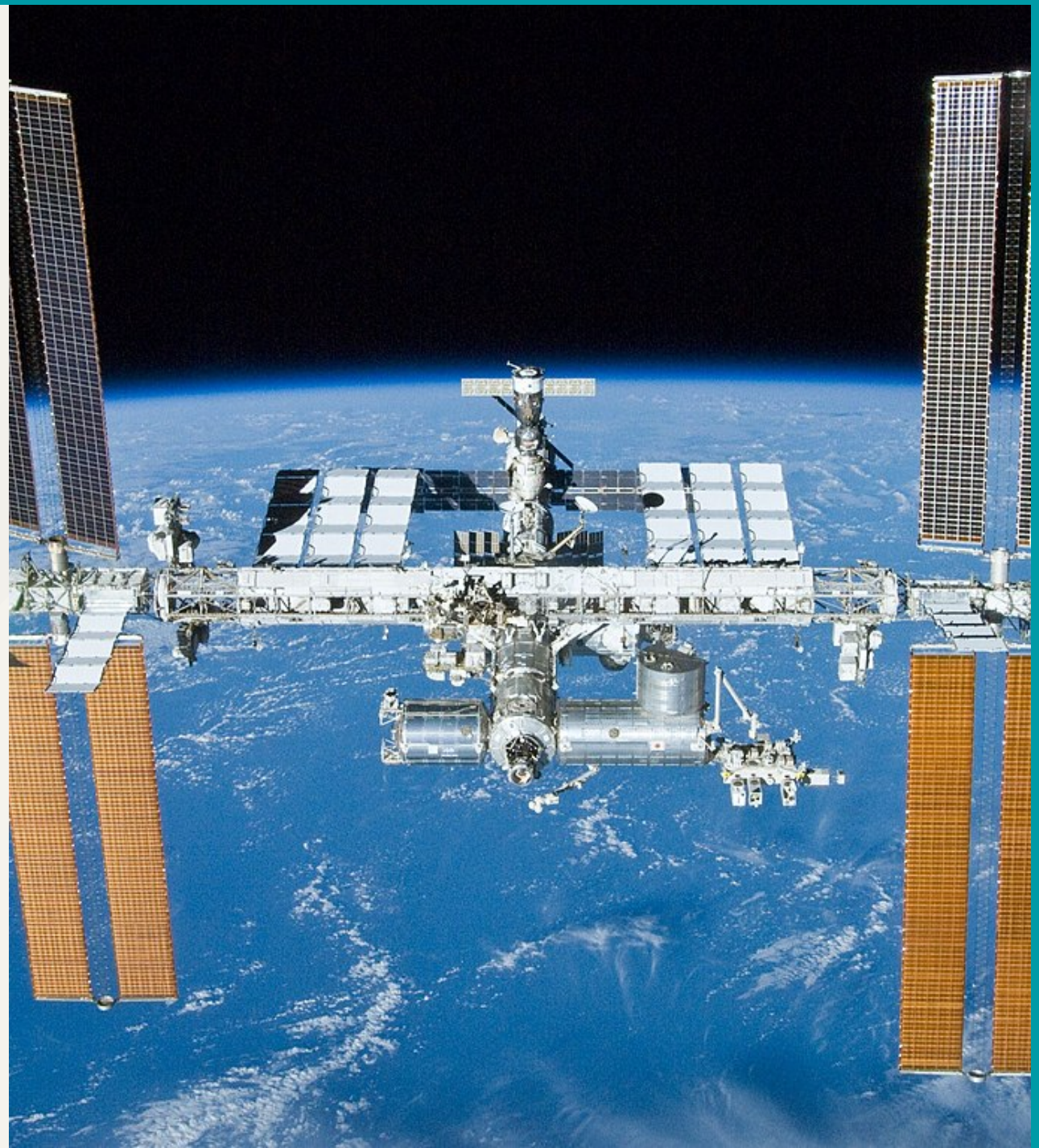
Functions

What we will cover...

1. Why do we want functions?
2. Anotomy of a function declaration.
3. Invoking a function.
4. Scope.

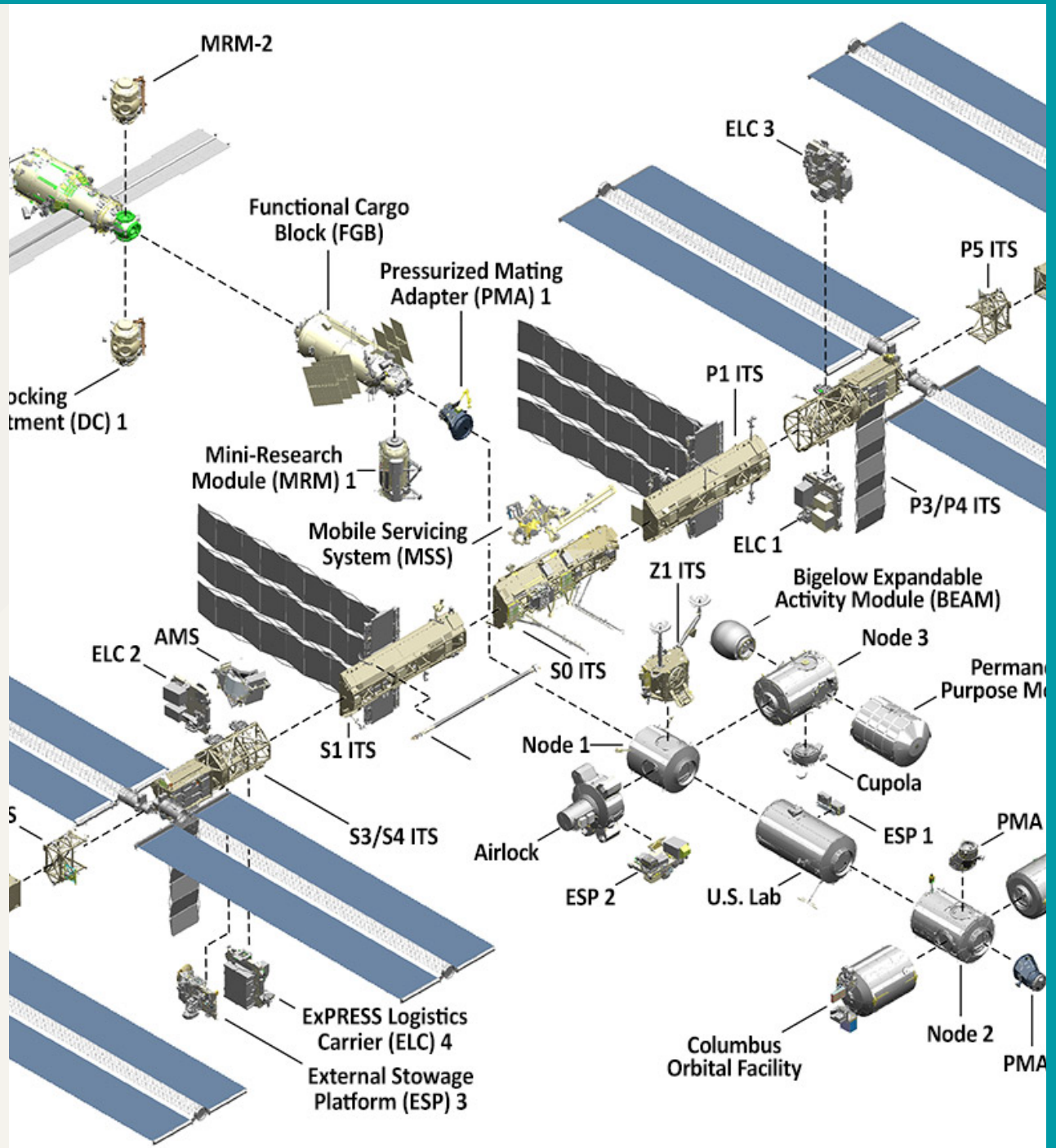
The International Space Station

The ISS was (is) a very complex project. It's a nice metaphor for any engineering project!



The International Space Station

The ISS consists of many small modules, many of which were built in different countries. They were first connected in outer space!!



Testable modules

The ISS worked because each module had a well defined **interface** through which it connected to other modules.

Programming can be thought of in the same way: we build a set of **units** that communicate with each other through well-defined interfaces.

In Python, we can use **functions** as those units.

Function definition

We create a new function in python with the `def` keyword

```
def
```

Function definition

We create a new function in python with the `def` keyword.

Next comes the **name** of the function (in this case, `add`).

```
def add
```

Function definition

We create a new function in python with the `def` keyword.

Next comes the **name** of the function (in this case, `add`).

The name must be followed by a set of parenthesis `()`

```
def add()
```

Function definition

We create a new function in python with the `def` keyword.

Next comes the **name** of the function (in this case, `add`).

The name must be followed by a set of parenthesis `()`, a colon `:`

```
def add():
```

Function definition

We create a new function in python with the `def` keyword.

Next comes the **name** of the function (in this case, `add`).

The name must be followed by a set of parenthesis `()`, a colon `:`, and then comes the function **body** on the next line.

```
def add():
    # body
```

Function body

The *body* can contain any valid Python code!

Note: the body *must* be indented exactly 4 spaces (your editor will place 4 spaces when you use the `tab` command).

```
def add():
    x = 5
```

Function body

Python uses *whitespace* to determine the function body. Here, the body only consists of `x = 5`. The line `y = 10` is not part of the function `add`!

```
def add():
    x = 5
y = 10
```

Function interface

This function does nothing!

```
def add():
    x = 5
```

In general, we want functions to *do some work*.

The **interface** of a function consists of:

1. Its "inputs"
2. Its "outputs"

Function interface

Function **parameters** (`a, b`) define the "inputs" of a function.

```
def add(a, b):
    x = 5
```

Function interface

Function **parameters** (`a, b`) define the "inputs" of a function.

The keyword `return` is used to return values from the function ("outputs").

Note: We are returning the **value** of `x`. The variable itself is not accessible outside the function body!

```
def add(a, b):
    x = a + b
    return x
```

Function invocation

Functions are tools. They are created once and, often, used many times!

The act of using a function is referred to as **calling** or **invoking**.

```
def add(a, b):
    x = a + b
    return x
```

Function invocation

In python, functions are called by writing the name of the function, followed by a set of parentheses `()`.

Function **arguments** go inside the parentheses, separated by `,`.

The function `add` is declared with two **parameters** (`a` and `b`), therefore, we call it with two **arguments** (`5` and `10`).

```
def add(a, b):
    x = a + b
    return x

add(5, 10)
```

Storing return values

If we call a function, and it returns something, we usually want that something!

We can store the return value in a variable, just the same as we declare, or overwrite, any variable.

```
def add(a, b):
    x = a + b
    return x

my_sum = add(5, 10)
```

Functions without parameters

Functions can be declared without parameters.

In that case they are called without arguments: `five()`.

```
def five():
    return 5

five() == 5
```

Functions without return values

Sometimes, we don't want functions to return anything.

Often this is the case when we want functions to perform **side effects**.

A side effect is something the function does above and beyond that which it returns. Printing to the terminal is an example of a side effect.

```
def print_double(num):
    x = num*2
    print(x)

print_double(10)
```

Scope

Functions can't change variables that are declared outside of the function.

This is called **scope**.

```
x = 5

def futile(num):
    x = num

futile(10)
print(x)
```

Scope

Scope - region of your program where your variable is defined.

Global variable - A variable defined in such a way that it can be accessed anywhere

Local variable - A variable only visible within the function where it is defined.

```
foo = 'bar'

def localer():
    foo = 'baz'
    return foo

def globaler():
    return foo

print(localer())
print(globaler())
```

Review

1. Why do we want functions?
2. Anotomy of a function declaration.
3. Invoking a function.
4. Scope.