1.1 Introducing the Python Programming Language

For the thousands of years of human history before the mid-twentieth century, humans collected, analysed, and created data by hand. Digital computers were a revolution not just in technology but in civilization because of their ability to store more data than could fit on all the sheets of paper in the world, and to perform computations on this data faster and more reliably than an army of humans. Today, we rely on complex computer programs to operate on data in a variety of ways, from sending messages back and forth with loved ones, organizing data in documents and media, to running simulations of physical, social, and biological systems.

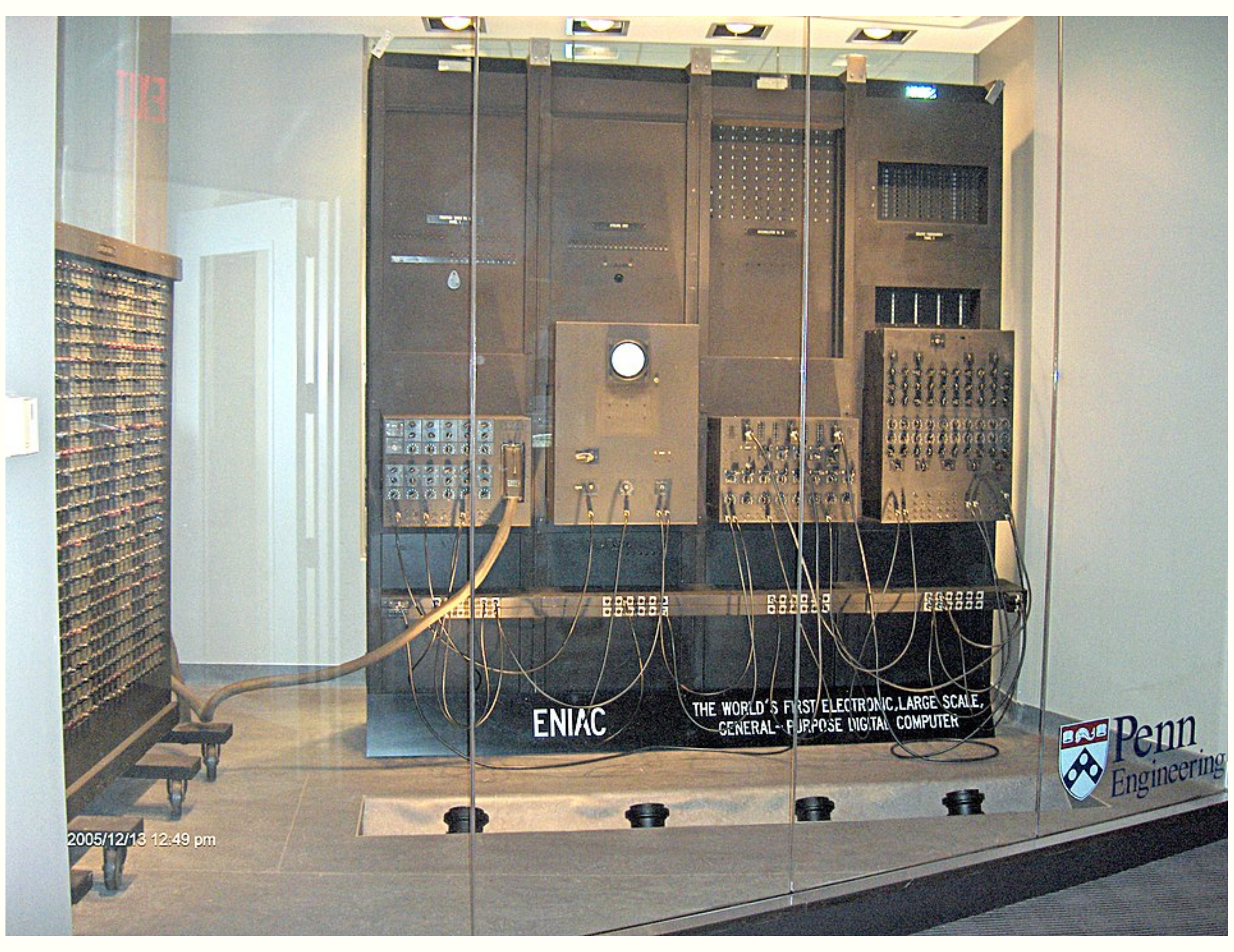
ENIAC
THE WORLD'S FIRST ELECTRONIC LARGE-SCALE, GENERAL-PURPOSE DIGITAL COMPUTER
Penn Engineering

Image of ENIAC, the first general purpose computer. The original uploader was TexasDex at English Wikipedia. - Transferred from en.wikipedia to Commons by Andrei Stroe using CommonsHelper, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=6557095.)

Yet for all their computational power, computers have one fundamental limitation: they have no agency, no inherent ability to make decisions about what to do. All they can do is take a set of (possibly very complex!) instructions, what we call a *computer program*, and execute those instructions—no more, and no less. And so if we, as computer scientists, want to harness the awesome power of computers, we need to learn how give these instructions in a way that a computer understands.

We need to learn how to speak to a computer.

What is a programming language?

Just as languages like English enable communication between humans, a **programming language** is a way of communicating a set of instructions to a computer. Like human languages, a programming language consists of a set of allowed words and the rules for putting those words together to form meaningful phrases. A programming language must be precise enough to be understood by a computer, and so uses a relatively small set of words and very structured rules for putting them together when compared with the rich complexities of any human language. Learning a programming language can be frustrating at first, because even a slight deviation from these rules results in the computer being unable to comprehend what we've written. But our time and efforts spent mastering the rules of a programming language yield a wonderful reward: the computer will not just understand what we're saying, but faithfully execute any instructions we write.

A **program** is simply the text of the instructions we wish to tell the computer to execute. We call this kind of text **program source code**, or **code** for short, to differentiate it from other forms of text. To write programs in a particular programming language, we need to understand two key properties of the language. The first is the **syntax** of the language, which is the name we give to the rules governing what constitutes a valid program in the language.[1] Before a computer can execute a program, it must read the instructions for the program; the syntax of the programming language specifies the expected format of these instructions.

[1] Programming language syntax is analogous to the rules of *grammar* governing human languages, which tell us how words are meant to be put together to form sentences.

The second concept is the **semantics** of the programming language, which refers to the rules governing the *meaning* of different instructions in the language.[2] Once the computer has read the instructions in a program, it begins executing them. The language semantics specifies what the computer should do for each instruction.

[2] This is analogous to the meanings of words in a human language.

The Python programming language

Just as there are thousands of human languages in the world today, each with their own vocabulary, grammar, and stylistic conventions, so too is there a plethora of programming languages that we can choose from. In this course, we'll use the Python programming language, which offers a simple, beginner-friendly syntax and a set of language instructions whose semantics are both very teachable and powerful enough to accomplish some truly amazing things, as we'll see throughout this course!

Now, neither our computer hardware nor operating system understand the Python programming language directly. Instead, the creators of Python wrote a program called the **Python interpreter**, whose job is to take programs written in the Python programming language and execute the instructions.[3] You can think of the Python interpreter as a mediator between you the programmer, writing instructions in Python code, and the computer hardware that actually executes instructions.

[3] So when you "download Python" onto your computer, what you're actually downloading and installing is this Python interpreter program.

There are two ways of writing code in the Python language and giving that code to the Python interpreter to execute it. The first way is to write full Python programs and save them as text files,[4] and then tell the Python interpreter to run that file. This is the standard way of creating Python programs: write the instructions, and then run them with the interpreter.

[4] Python programs use the `.py` file extension to distinguish them from other kinds of text files, like `.txt` files that you might have seen before.

The second way is to run the Python interpreter in an interactive mode, which we call the **Python console** or **Python shell**. In this mode, we can write small fragments of Python code and ask the Python interpreter to execute each fragment one at a time.[5] The Python console is useful for experimenting and exploring the language, as you get quick feedback after every single instruction you write. The drawback is that interactions with the interpreter in the Python console are lost every time you restart the console. So we'll use the following approach through the course: *use the Python console to learn about and experiment with the Python language, and write full programs in `.py` files so that you can save and edit them later.*

[5] We'll make the meaning of the word "fragment" more precise in the next section!

A demo of the Python console inside the PyCharm application. While there are many ways of using the Python interpreter, we'll be using PyCharm as the standard way of creating and running Python programs throughout this course.