Class overview today - September 7, 2022

- **A taste of Python**
  - Introductions and practical course information
  - Elements of a computer and computer programs
  - An introduction to our course computing environment
  - A taste of Python

Geo-Python

A taste of Python

Lecturer: David Whipp
david.whipp@helsinki.fi

7.9.2022

Who are we?

- **Lecturers**
  - Dave Whipp - Geo-Python
  - Christoph Fink - AutoGIS
- **Assistants**
  - Mikko Kangasmaa
  - Justus Poutanen

Course websites

- Geo-Python/AutoGIS 1 (Period I)
  https://geo-python.github.io

- AutoGIS 2 (Period II)
  https://autogis.github.io

Class meetings in Period I

- On-site lessons
  - Wednesdays 9:15-12:00
- Optional work sessions
  - Thursdays 12:15-16:00
  - Fridays 10:15-14:00
- You can feel free to attend either work session (or both)

“AUTOGIS”

PERIOD 1:

- **GEOG-329-1, Automating GIS-processes 1, Geo-Python**
- Introduction to programming, data analysis and visualization

PERIOD 2:

- **GEOG-329-2, Automating GIS-processes 2, Geography**
- Spatial data management, analysis and visualization

5 + 5 ECTS

Who are you?

- We'd like to know a bit about who you are, and ask that you direct your web browser or phone to a real-time poll at https://geo-python.github.io/poll

Goals of this part of the course

There are basically three goals in this part of the course

1. Introduce the **Python programming language**
2. Develop **basic programming skills**
3. Discuss **essential (good) programming practices** needed by young scientists

Some motivation

Goals of this lecture

- Provide an overview of **basic computing practices**, and why you should learn them

- Define **computers** and **programming languages**, and how they operate

- Look at the components of a **computer program** and a strategy for writing your own code

Learning to program

- A significant part of this course will be development of basic **programming skills** that will help you write and use simple numerical models
  - I know you're not computer scientists - we aren't either
  - Our goal is take small steps to learn together
- Do you really need to know how to program? **Yes.**
  - You might not be a superstar, but learning to write simple codes can be very useful

Why learn to program?

- Geology and geography are becoming increasingly quantitative and basic programming skills are one of the fundamental skills that will help you be a better scientist

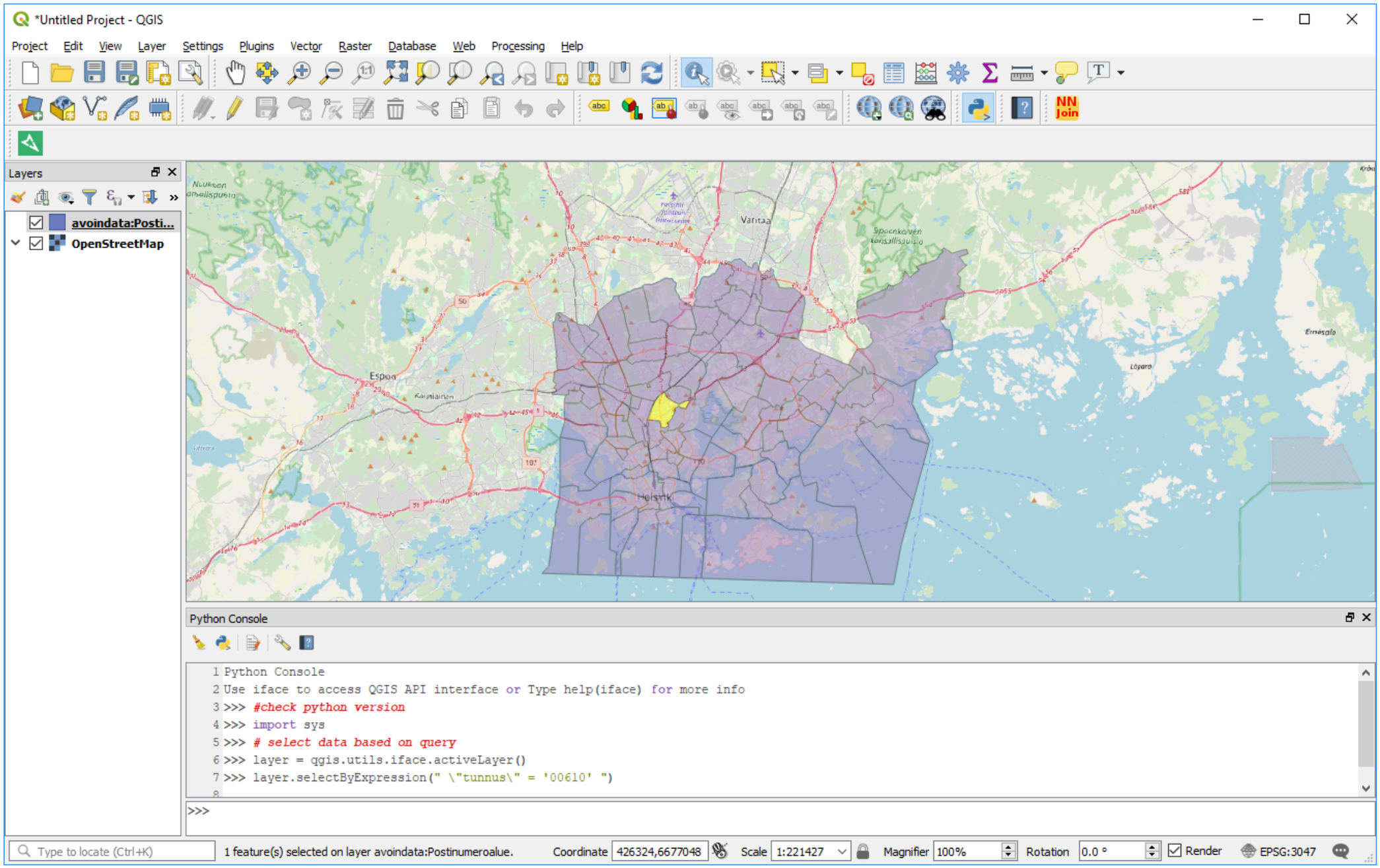
Why learn to program?

IPython: Users/whipp

```
In [7]: average_geoscientist = 100

In [8]: programming_factor = 1000

In [9]: quantitative_geoscientist = average_geoscientist * programming_factor

In [10]: quantitative_geoscientist > average_geoscientist
Out[10]: True

In [11]:
```

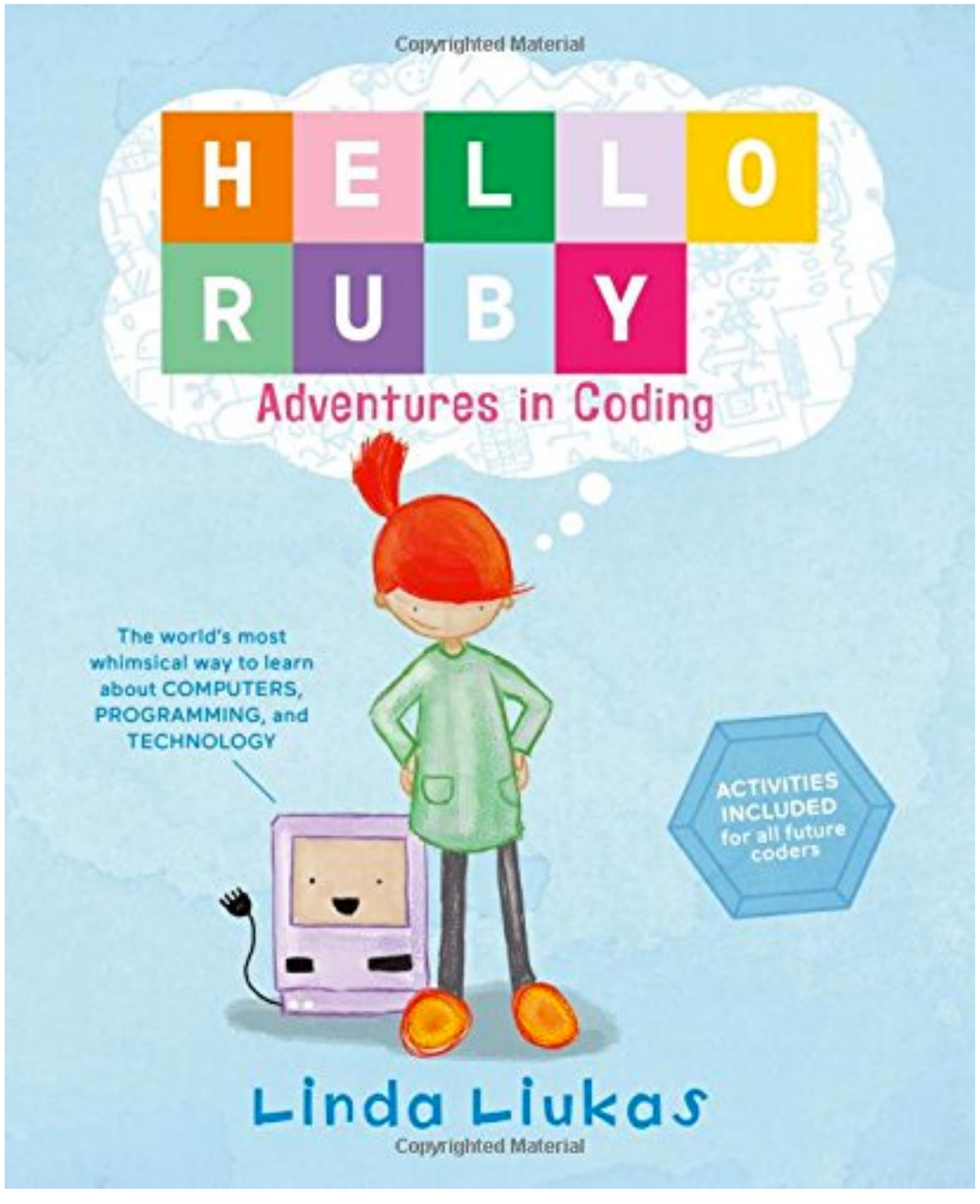
- You can extend existing software by developing your own solutions when solutions do not exist or are inefficient
  - Many software packages offer the ability to extend their capabilities by adding your own short programs (e.g., ArcGIS, ParaView, Google Earth, etc.)

You can interact with GIS software using Python



Python console in QGIS

Why learn to program?



- Believe it or not, **programming is fun!** It involves
  - Breaking complex problems down into simpler pieces
  - Developing a strategy for solving the problem
  - Testing your solution
- All of this can be exciting and rewarding (when the code works…)

The scientific method…

…and how programming can make you a better scientist

1. Define a question
2. Gather information and resources (observe)
3. Form an explanatory hypothesis
4. Test the hypothesis by performing an experiment and collecting data in a reproducible manner
5. Analyze the data
6. Interpret the data and draw conclusions that serve as a starting point for new hypothesis
7. Publish results
8. Retest (frequently done by other scientists)

Learning to program can help us…

1. Define a question
2. **Gather information and resources** (observe)
3. Form an explanatory hypothesis
4. Test the hypothesis by **performing an experiment and collecting data** in a reproducible manner
5. **Analyze the data**
6. **Interpret the data** and draw conclusions that serve as a starting point for new hypothesis
7. Publish results
8. Retest (frequently done by other scientists)

Good programming practices can help us…

1. Define a question
2. Gather information and resources (observe)
3. Form an explanatory hypothesis
4. Test the hypothesis by performing an experiment and collecting data **in a reproducible manner**
5. Analyze the data
6. Interpret the data and draw conclusions that serve as a starting point for new hypothesis
7. Publish results
8. **Retest** (frequently done by other scientists)

What is a computer?

What is a computer?

- Let's crowdsource: https://geo-python.github.io/poll
- Add your thoughts on what comprises a computer
- Vote for options you support

What is a computer?

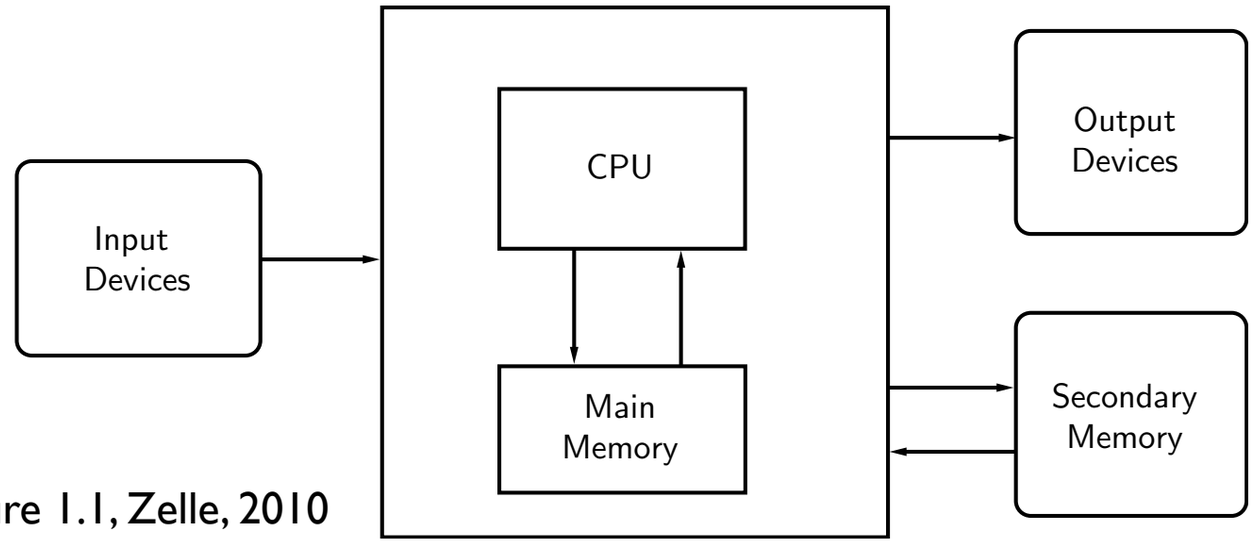


Figure 1.1, Zelle, 2010

- A **computer** is a machine that stores and manipulates information under the control of a changeable program

What is a computer?

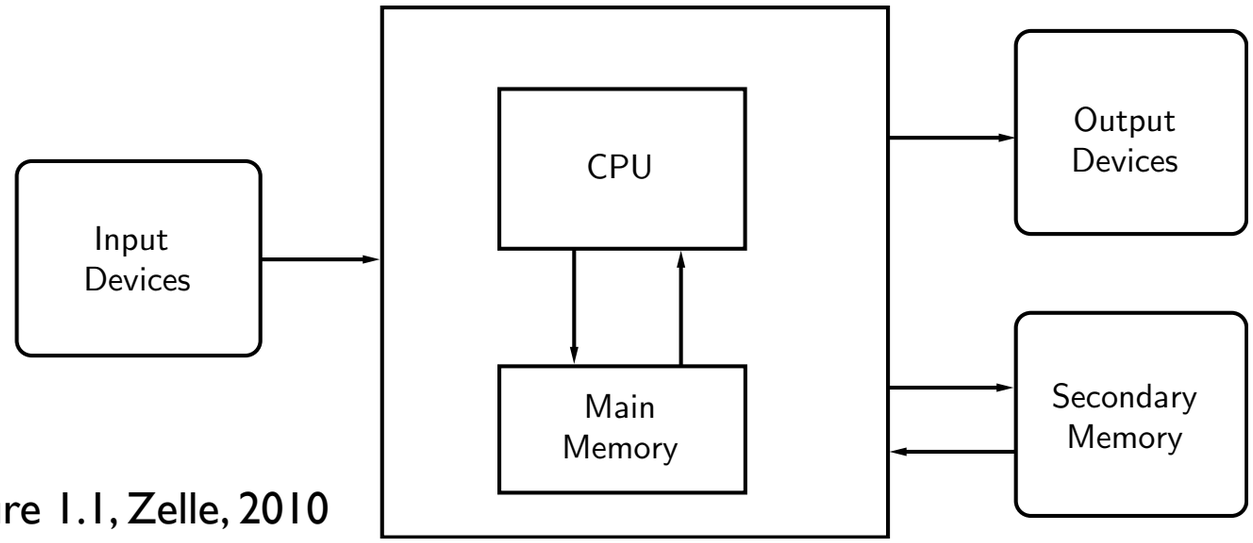


Figure 1.1, Zelle, 2010

- A **computer** is a machine that stores and **manipulates information** under the control of a changeable program
  - Information can be input, modified into a new/useful form and output for our interpretation

What is a computer?

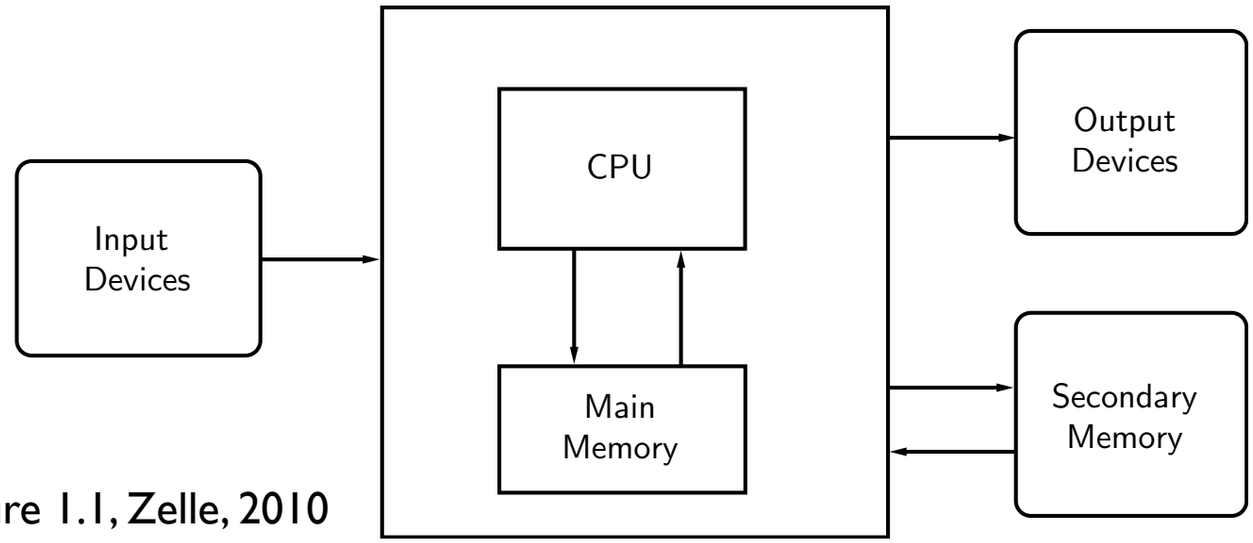


Figure 1.1, Zelle, 2010

- A **computer** is a machine that stores and manipulates information under the control of a **changeable program**
  - Controlled by a computer program that can be modified

What are computers good at?

```
>>> print(2 + 2)
4
```

```
>>> print("2 + 2 =",2 + 2)
2 + 2 = 4
```

- Well-defined, clear tasks
  - Add 2 + 2 and return the answer
- Data storage/manipulation
- Repetitive calculations
- Processing data or instructions

What are computers good at?

Python prompt

Print function

```
>>> print(2 + 2)
4
```

Returned value

```
>>> print("2 + 2 =",2 + 2)
2 + 2 = 4
```

- Well-defined, clear tasks
  - Add 2 + 2 and return the answer
- Data storage/manipulation
- Repetitive calculations
- Processing data or instructions

What aren't computers good at?

- Abstract or poorly defined tasks
  - Calculate pi

What aren't computers good at?

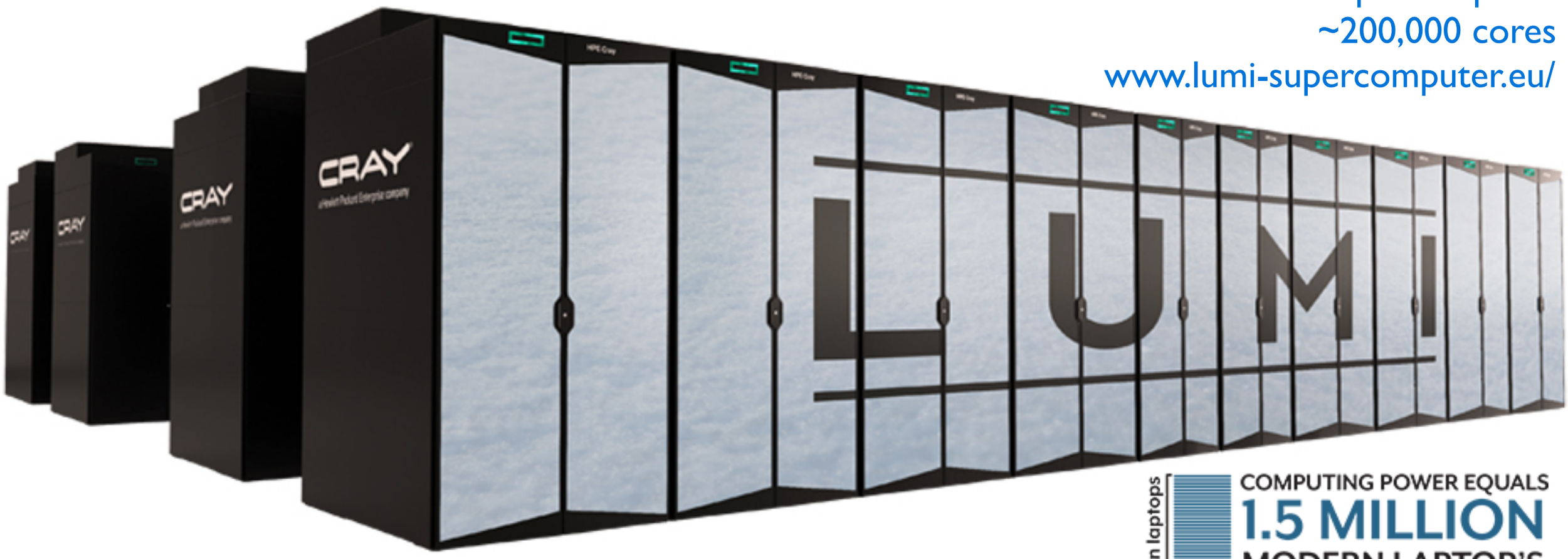
```
3.1415926535 8979323846 2643383279 5028841971 6939937510 5820974944 5923078164 0628620899
  8628034825 3421170679 8214808651 3282306647 0938446095 5058223172 5359408128 4811174502
  8410270193 8521105559 6446229489 5493038196 4428810975 6659334461 2847564823 3786783165
  2712019091 4564856692 3460348610 4543266482 1339360726 0249141273 7245870066 0631558817
  4881520920 9628292540 9171536436 7892590360 0113305305 4882046652 1384146951 9415116094
  3305727036 5759591953 0921861173 8193261179 3105118548 0744623799 6274956735 1885752724
8912279381 8301194912 9833673362 4406566430 8602139494 6395224737 1907021798 6094370277
0539217176 2931767523 8467481846 7669405132 0005681271 4526356082 7785771342 7577896091
7363717872 1468440901 2249534301 4654958537 1050792279 6892589235 4201995611 2129021960
8640344181 5981362977 4771309960 5187072113 4999999837 2978049951 0597317328 1609631859
5024459455 3469083026 4252230825 3344685035 2619311881 7101000313 7838752886 5875332083
8142061717 7669147303 5982534904 2875546873 1159562863 8823537875 9375195778 1857780532
1712268066 1300192787 6611195909 2164201989
```

The first 1000 digits of pi

- Abstract or poorly defined tasks
- Calculate pi

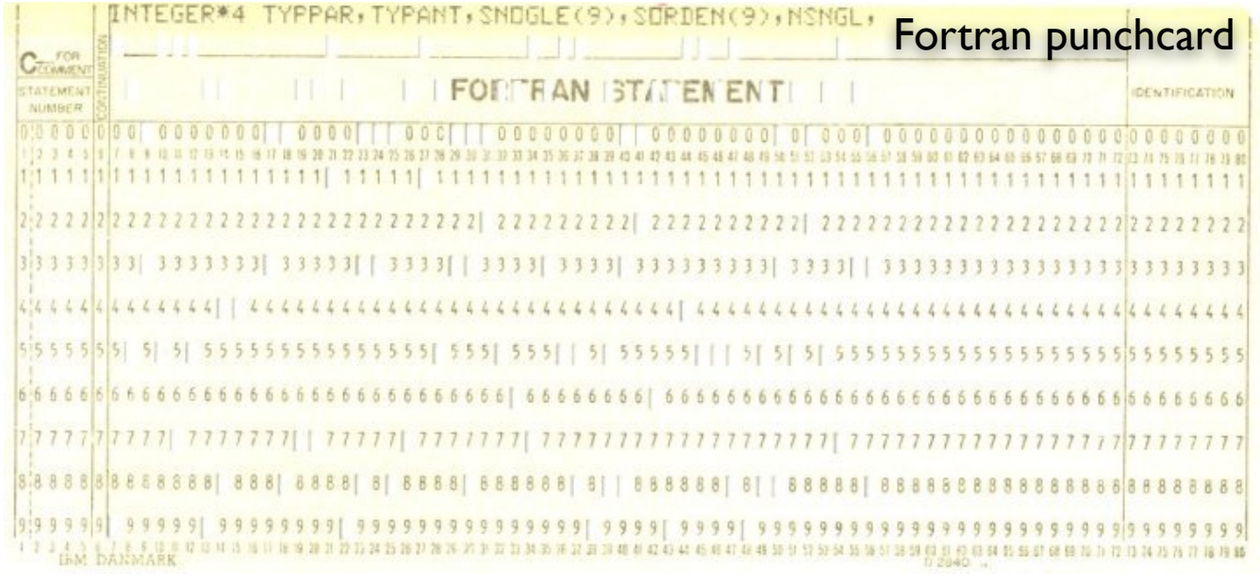
What aren't computers good at?

Lumi supercomputer
~200,000 cores
www.lumi-supercomputer.eu/



- Tasks that are not computable
  - Computer, where are my car keys?
  - Some problems simply cannot be solved, or require too much computing power

What is a program?

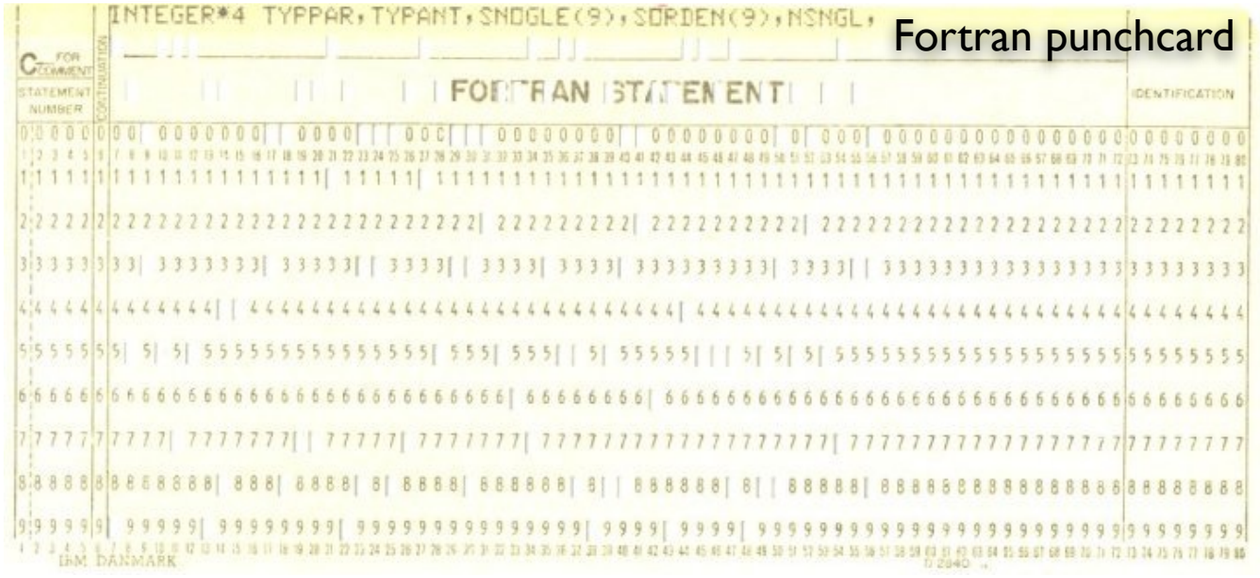


Fortran punchcard

```
# Define plot variables
misfit = NA_data[:,0]
var1 = NA_data[:,1]
var2 = NA_data[:,2]
var3 = NA_data[:,3]
clrmin = round(min(misfit),3)
clrmax = round(min(misfit),2)
trans = 0.75
ptsize = 40
```

Python source code

What is a program?



Fortran punchcard

```
# Define plot variables
misfit = NA_data[:,0]
var1 = NA_data[:,1]
var2 = NA_data[:,2]
var3 = NA_data[:,3]
clrmin = round(min(misfit),3)
clrmax = round(min(misfit),2)
trans = 0.75
ptsize = 40
```

Python source code

- A **program** is a detailed list of step-by-step instructions telling the computer exactly what to do
- The program can be changed to alter what the computer will do when the code is executed
- **Software** is another name for a program

What is a programming language?

- A **computer language** is what we use to 'talk' to a computer
  - Unfortunately, computers don't *yet* understand our native languages
- A **programming language** is like a code of instructions for the computer to follow
  - It is exact and unambiguous
  - Every structure has a precise form (**syntax**) and a precise meaning (**semantics**)
- Python is just one of many programming languages

Developing a program

- Coming up with a specific list of instructions for the computer to follow in order to accomplish a desired task is not easy
- The following list will serve us as a **general software development strategy**

1. Analyze the problem
2. Determine specifications
3. Create a design
4. Implement the design
5. Test/debug the program
6. Maintain the program (if necessary)

Let's consider an example

- As an American, I was raised in a country that uses Fahrenheit for temperatures
  - 70°F is lovely
  - 90°F is hot
  - Water freezes at 32°F

- The problem here in Finland is that I don't always know what I should wear to work when I find weather reports with temperatures in degrees Celsius
  - **I think a simple program could help**

Developing a program

1. Analyze the problem

    - Before you can solve a problem, you must figure out exactly what should be solved

Developing a program

1. Analyze the problem

   - Before you can solve a problem, you must figure out exactly what should be solved

2. Determine specifications

   - Describe exactly what the program will do
   - Don’t worry about how it will work. Determine the input and output values and how they should interact in the program

Developing a program

3. Create a design

- What is the overall structure of the program? How will it work?
- It is often helpful to write out the code operation in **pseudocode**, precise English (or Finnish) describing the program. Be specific!

Developing a program

3. Create a design

   - What is the overall structure of the program? How will it work?
   - It is often helpful to write out the code operation in **pseudocode**, precise English (or Finnish) describing the program. Be specific!

4. Implement the design

   - If you've done a good job with the previous steps, this should be fairly straightforward. Take your pseudocode and 'translate' it into Python

Developing a program

5. Test/debug the program

- Now you can put your new Python code to the test (literally) by running it to see whether it reproduces the expected values
- For any test, you should know the correct values in advance of running your code. How else can you confirm it works???

Developing a program

5. Test/debug the program

   - Now you can put your new Python code to the test (literally) by running it to see whether it reproduces the expected values
   - For any test, you should know the correct values in advance of running your code. How else can you confirm it works???

6. Maintain the program

   - If you've written something that will be shared by other users, a helpful programmer will continue to add features that are requested by the users

Recap

- **What is a computer?**

- What is a program?

- What are some of the steps in developing a program?

Recap

- What is a computer?
- **What is a program?**
- What are some of the steps in developing a program?

Recap

- What is a computer?
- What is a program?
- **What are some of the steps in developing a program?**

References

Zelle, J. M. (2010). *Python programming: an introduction to computer science* (2nd ed.). Franklin, Beedle & Associates, Inc.

Our first taste of Python

- Open a web browser and navigate to https://geo-python.github.io/

Puupyton / Green tree python