Introduction to Quantitative Geology
Lecture 2
Essentials of computing

Lecturer: David Whipp
david.whipp@helsinki.fi

11.3.2015
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 (I’m not either), and 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 is increasingly quantitative and basic programming skills are one of the fundamental quantitative skills that will help you be a better scientist (and set you apart).
Why learn to program?

• Rather than being restricted to using existing software, you will have the ability to develop 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.)
Python can be called directly from ArcGIS (!)
Why learn to program?

- Believe it or not, **programming is fun!** Breaking complex problems down into simpler pieces, developing a strategy for solving the problem and finally testing your solution is exciting and rewarding (when the code works…)

http://www.adafruit.com/products/1161
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?
What is a computer?

A computer is a machine that stores and manipulates information under the control of a changeable program.
What is a computer?

- 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?

- 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?

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

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

>>> print("2 + 2 =", 2 + 2)
2 + 2 = 4
```
What are computers good at?

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

Python prompt

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

Print function

```python
>>> print("2 + 2 =", 2 + 2)
2 + 2 = 4
```
What aren’t computers good at?

- Abstract or poorly defined tasks
- Calculate pi
What aren’t computers good at?

- Abstract or poorly defined tasks
- Calculate pi

The first 1000 digits of pi

3.1415926535 8979323846 2643383279 5028841971 6939937510 5820974944 5923078164 0628620899
8628034825 3421170679 8514808661 2643383279 5028841971 6939937510 5820974944 5923078164
0628620899 8628034825 3421170679 8514808661 2643383279 5028841971 6939937510 5820974944
5923078164 0628620899 8628034825 3421170679 8514808661 2643383279 5028841971 6939937510
5820974944 5923078164 0628620899 8628034825 3421170679 8514808661 2643383279 5028841971
6939937510 5820974944 5923078164 0628620899 8628034825 3421170679 8514808661 2643383279
5028841971 6939937510 5820974944 5923078164 0628620899 8628034825 3421170679 8514808661
2643383279 5028841971 6939937510 5820974944 5923078164 0628620899 8628034825 3421170679
8514808661 2643383279 5028841971 6939937510 5820974944 5923078164 0628620899 8628034825
3421170679 8514808661 2643383279 5028841971 6939937510 5820974944 5923078164 062862089
What aren’t computers good at?

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

Sisu supercomputer
~40,000 cores
~107,000 GB of RAM

www.csc.fi
What is a program?

# 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
What is a program?

- 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.

```python
# 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 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
Examples of different programming languages

Python

print("Hello, world!")

MATLAB

disp('Hello, world!')
Examples of different programming languages

Python

```python
print("Hello, world!")
```

Fortran 90

```fortran
program hello
    write(*,*) 'Hello, world!
end program hello
```

MATLAB

```matlab
disp('Hello, world!')
```

C

```c
#include <stdio.h>

int main(void)
{
    printf("Hello, world!\n");
    return 0;
}
```
Examples of different programming languages

Python
print("Hello, world!")

Fortran 90
program hello
  write(*,*) 'Hello, world!'
end program hello

MATLAB
disp('Hello, world!')

C
#include <stdio.h>
int main(void)
{
  printf("Hello, world!\n");
  return 0;
}

These are all examples of high-level programming languages, languages meant to be understood by humans. Computer hardware actually understands a very low-level language known as machine language.
Elements of a program

- The elements of a program are the different pieces of the program that are combined to produce the desired results when the code is executed.

- We will explore the components of a program in greater detail in the laboratory exercise this week, but it is helpful to consider a few components in advance.

- The goal is to introduce the concepts now, you’ll understand them better after the first laboratory exercise.
Names

- **Names** are given to many different elements in a program
- **Variables** are used to give names to values

```python
number
a
Daves_favorite_number
RidgeSpreadingRate
IceCreamFlavor3
```

- In Python, a name must begin with either a letter or the underscore “_” character, and be followed by any sequence of letters, digits or underscores
Names

<table>
<thead>
<tr>
<th>and</th>
<th>del</th>
<th>for</th>
<th>is</th>
<th>raise</th>
</tr>
</thead>
<tbody>
<tr>
<td>assert</td>
<td>elif</td>
<td>from</td>
<td>lambda</td>
<td>return</td>
</tr>
<tr>
<td>break</td>
<td>else</td>
<td>global</td>
<td>not</td>
<td>try</td>
</tr>
<tr>
<td>class</td>
<td>except</td>
<td>if</td>
<td>or</td>
<td>while</td>
</tr>
<tr>
<td>continue</td>
<td>exec</td>
<td>import</td>
<td>pass</td>
<td>yield</td>
</tr>
<tr>
<td>def</td>
<td>finally</td>
<td>in</td>
<td>print</td>
<td></td>
</tr>
</tbody>
</table>

Python keywords; Table 2.1, Zelle, 2010

- Some names (**keywords**) are “protected” and **cannot be used** as variables

```python
>>> for=2
File "<stdin>", line 1
  for=2
```

*SyntaxError: invalid syntax*

- Technically, all names in Python are known as **identifiers**
Expressions

• An **expression** is a fragment of program code that produces or calculates a new data value

```python
>>> 12345
12345
>>> "Hi there"
'Hi there'
```

• The expression is **evaluated (calculated)** by pressing Enter, for example
Expressions

• An expression is a fragment of program code that produces or calculates a new data value

```python
>>> 12345
12345
>>> "Hi there"
‘Hi there’
>>> x = 3
>>> x
```

• The expression is evaluated (calculated) by pressing Enter, for example
Expressions

- An expression is a fragment of program code that produces or calculates a new data value

```python
>>> 12345
12345
>>> "Hi there"
'Hi there'
>>> x = 3
>>> x
3
>>> print(x)
3
```

- The expression is evaluated (calculated) by pressing Enter, for example
Expressions

- Of course, expressions can be made more powerful (or complex) by using **operators**
- For math, you’re already familiar with many of these

<table>
<thead>
<tr>
<th>Operation</th>
<th>Symbol</th>
<th>Example syntax</th>
<th>Returned value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>2 + 2</td>
<td>4</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>4 - 2</td>
<td>2</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>2 * 3</td>
<td>6</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>4 / 2</td>
<td>2</td>
</tr>
<tr>
<td>Exponentiation</td>
<td>**</td>
<td>2 ** 3</td>
<td>8</td>
</tr>
</tbody>
</table>

- There are other **operators** in Python as well, but we’ll deal with them when we encounter them
Assignment statements

- **Assignment statements** have the general form
  
  `<variable> = <expression>`

- Thus, we can use **variables** to store the results of evaluated **expressions**

```
>>> DogCount = 47
>>> DogCount
47
>>> DogCount = DogCount - 5
>>> DogCount
42
```
Assignment statements

- **Assignment statements** have the general form
  `<variable> = <expression>

- Thus, we can use **variables** to store the results of evaluated **expressions**

  >>> DogCount = 47
  >>> DogCount
  47
  >>> DogCount = DogCount - 5
  >>> DogCount
  42
  >>> DogCount = 5 * DogCount - 43
  >>> DogCount
  167
Assignment statements

- An excerpt from our example on the previous slide illustrates a few important Python features

```
>>> DogCount = 47
>>> DogCount
47
>>> DogCount = DogCount - 5
>>> DogCount
42
```

- A variable can be assigned many times and will retain the value from its latest assignment

- A variable can be referenced in assigning itself a new value (!)
Comments

- As geologists, not computer scientists, comments are probably the most important element of the codes we’ll generate.

- Comments are text in the program that does not get executed.

- They’re ignored by Python, but important for human users.

```python
timenow = time * 31557600
```
Comments

- As geologists, not computer scientists, comments are probably the most important element of the codes we’ll generate.

- Comments are text in the program that does not get executed.

- They’re ignored by Python, but important for human users.

\[
timenow = \text{time} \times 31557600
\]

\[
\text{Time\_in\_seconds} = \text{time} \times 365.25 \times 24 \times 3600
\]
Comments

- As geologists, not computer scientists, comments are probably the most important element of the codes we’ll generate.
- Comments are text in the program that does not get executed.
- They’re ignored by Python, but important for human users.

```python
# Convert time in Ma to seconds
Time_in_seconds = time * 365.25 * 24 * 3600
```

```python
timenow = time * 31557600
Time_in_seconds = time * 365.25 * 24 * 3600
# Convert time to sec.
Time_in_seconds = time * 365.25 * 24 * 3600
```
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)
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!

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???

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

- **Computing** is an essential skill for the modern Earth scientist.

- We can tell **computers** what to calculate with precise instructions listed using **programming languages**.

- Writing code is not easy, but a simple strategy for developing **your code** will help make the process less painful and more efficient.
Lab preview

- Wednesday, we’ll start learning Python by fire!
- Exercise will introduce the basics of using Python