

# UC Santa Cruz GEODES Computing ‘On Ramp’

Kelian Dascher-Cousineau and Valère Lambert

## Goals for today:

- Get a feeling for what a computer program is, how to start writing one and how to run it
- Practice putting some code together in Python and running it
- Feel comfortable knowing how to search for more information

# What does a computer do?

Two basic tasks:

- Performs **calculations** – potentially billions of calculations per second (compute power)
- **Remembers** results – potentially 100s of gigabytes of storage (memory)

What kinds of calculations?

- **Built-in** to a programming language
- Those that **you define** as a programmer

Computers do what you tell them to do, or more specifically what your instructions (code) tells them to do

## What is a program?

A detailed plan or procedure for solving a problem or performing a task with a computer

Specifically, it is an **ordered sequence of computational instructions** needed to achieve a solution/outcome

# What do computers actually understand?

Computers understand data in the form of 0s and 1s – very basic machine language (native language)

They do simple manipulations with those 0s and 1s:

- Move values to different positions in a chain
- Add, multiply, subtract, divide these values
- Compare these values, and if one is less than the other they can follow one step rather than the other

# What are programming languages?

A vocabulary and set of grammatical rules (like human language) that can be translated directly to computer language to communicate instructions for specific tasks

In other words, programming languages are a set of symbols and rules that form a bridge between human language and machine language

# Types of programming languages

Programming languages can be classified as lower-level versus higher-level

- Lower-level languages are closer to machine language vs. higher-level are closer to human language, like English

Examples of higher-level languages: Python, Matlab, R, HTML, Javascript, C, C++, Fortran

## Compiled language:

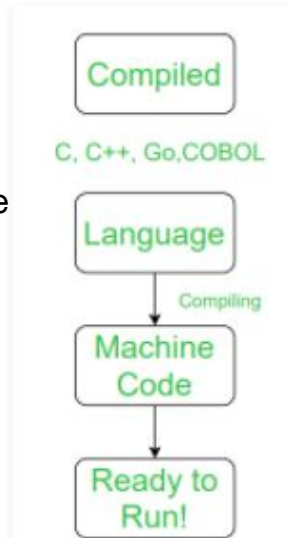
Code is compiled/translated into machine language and expressed as instructions in machine language

Compilation produces an executable file that is undecipherable by humans

e.g. C, C++, Fortran

Generally faster to run  
(already in machine language)

Errors may appear during compilation that prevent code from being translated



[geeksforgeeks.org](https://www.geeksforgeeks.org)

## Interpreted language:

Code is interpreted without compiling into machine language instructions.

Instructions are not directly executed by machine but instead read and executed by some other program

e.g. Python, Matlab, JavaScript, Perl

One step between code and execution

Interpreted programs can be modified and debugged while they are running



# What is a script? How does a script differ from a program?

A **script** is a series of commands within a file that is **interpreted and controls another application**

Example: code written in any interpreted language like Python

A **program** is a set of commands that **executes independent of any other application**.

Programs are compiled before use and then one runs the executable containing machine language instructions i.e. any code written in a compiled language

## 1. Example python **script**: hello\_world.py

```
# This Python script prints Hello World!  
print("Hello World! \n")
```

### **Running Python script from Terminal**

```
(onramp) valerelambert$ python hello_world.py  
Hello World!
```

**\*The Python script is read by the Python interpreter which calls compiled programs to execute instructions in the script**

## 2. Example C **program**: hello\_world.c

```
/* This C program prints Hello World! */  
#include<stdio.h>  
int main(void)  
{  
    printf("Hello World! \n");  
    return 0;  
}
```

### **Compiling program from Terminal**

```
(onramp) valerelambert$ gcc hello_world.c -o HelloWorld
```

### **Running C program from Terminal**

```
(onramp) valerelambert$ ./Helloworld  
Hello World!
```

# What is a computing environment?

Many problems are solved by computers making use of multiple computational devices, networks and software to perform different tasks, transfer and storage information.

This **collection of software and hardware used to solve a problem** is the computing environment

## What are package and environment managers?

Package managers keep track of what software is installed on your computer

- facilitate installing new software, upgrading software to newer versions and removing software

Environment managers keep track of collections of software packages used for specific projects

### But why?

Imagine you have two different Python projects that use a specific software package. One requires updating the package, however if the package is updated then the other project may not function properly.

An environment manager allows on to set up two environments with different version of the package

**Examples:** Anaconda, Homebrew, Pip

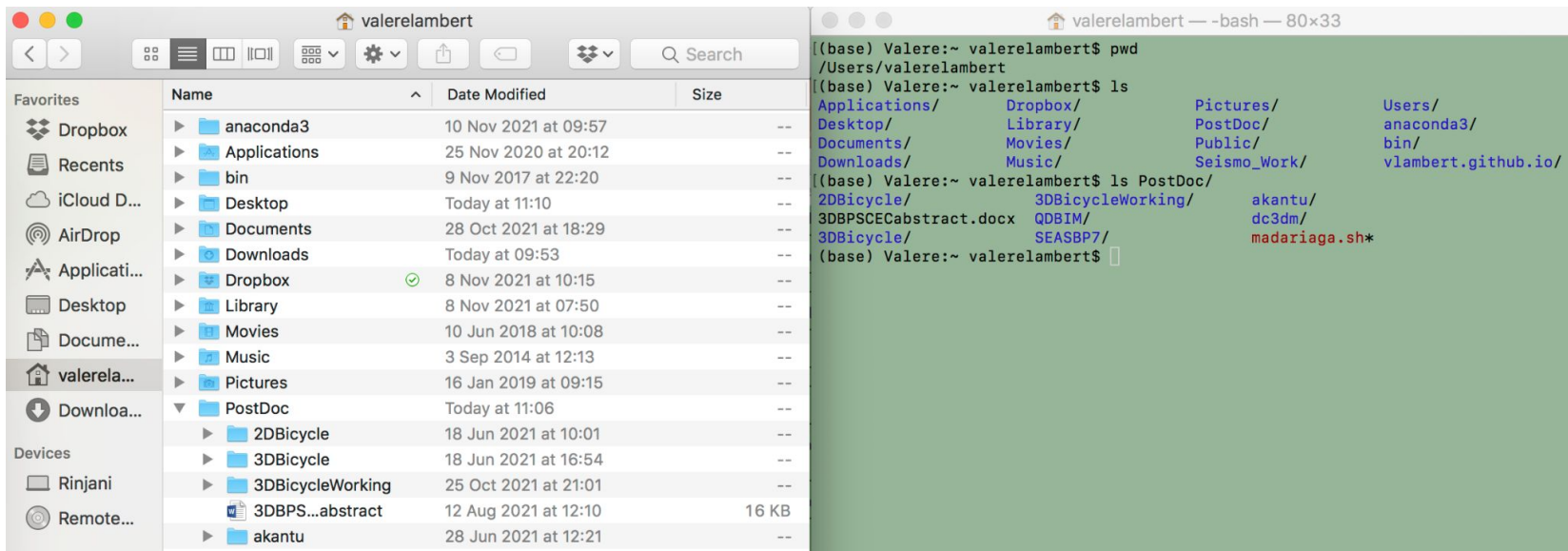
<https://docs.conda.io/projects/conda/en/4.6.0/downloads/52a95608c49671267e40c689e0bc00ca/conda-cheatsheet.pdf>

# Demystifying the terminal

The terminal is an application that provides text-based access to the operating system of your computer

- Provides a command-line interface compared to a graphical interface (like Finder on Mac)
- Generally faster than using graphical user interfaces (GUIs)

One can use different "shells" (e.g. bash, tcsh, zsh) which are interpreters that convert your commands in the terminal to machine instructions



# Some basic terminal commands

<https://www.techrepublic.com/article/16-terminal-commands-every-user-should-know/>

## Some basic commands:

`pwd` : "print working directory" or current directory  
`ls` : "List" or view the contents of a directory  
`cd` : "Change directory" that you are currently working in  
`touch` : make new empty file  
`mkdir` : "make a new directory"  
`open` : "open" file with default application  
`cp` : "copy" a file from one location to another  
`mv` : "move" a file from one place or format to another  
`rm` : "remove" or delete one or more files or directories

## Examples:

`pwd`  
`ls path/to/directory`  
`cd path/to/directory`  
`touch "newfilename"`  
`mkdir "NewDirectoryName"`  
`open "filename"`  
`cp "filename" "newfilename"`  
`mv "filename" "newdir/newfilename"`  
`rm "filename"`



# Integrated Development Environment (IDE)

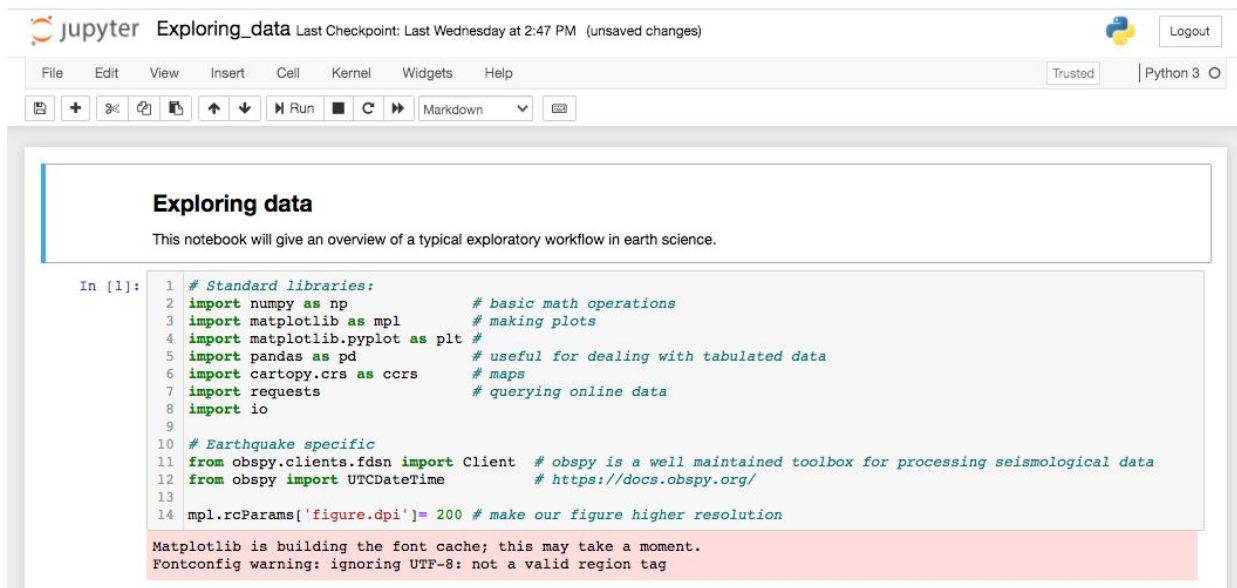
IDEs are software applications that facilitate writing a computer program

- **Editing source code** – create a blank file, write some lines of code
- **Building executables** - contain necessary compilers and interpreters
- **Debugging tools**

May also have special features like syntax highlighting or autocomplete if the IDE knows language syntax

## Common IDEs:

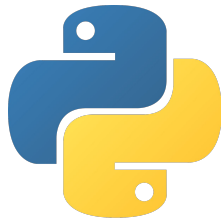
Spyder, PyCharm, Jupyterlab,  
Visual Studio Code, Eclipse,  
NetBeans, Komodo, XCode



The screenshot displays the JupyterLab web interface. At the top, the header shows the Jupyter logo, the notebook title 'Exploring\_data', and a timestamp 'Last Checkpoint: Last Wednesday at 2:47 PM (unsaved changes)'. A 'Logout' button is visible on the right. Below the header is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. To the right of the menu bar are 'Trusted' and 'Python 3' indicators. A toolbar with icons for file operations and execution is located below the menu bar. The main content area shows a notebook titled 'Exploring data' with a description: 'This notebook will give an overview of a typical exploratory workflow in earth science.' The notebook contains a code cell with the following Python code:

```
In [1]: 1 # Standard libraries:
2 import numpy as np           # basic math operations
3 import matplotlib as mpl     # making plots
4 import matplotlib.pyplot as plt
5 import pandas as pd          # useful for dealing with tabulated data
6 import cartopy.crs as ccrs   # maps
7 import requests              # querying online data
8 import io
9
10 # Earthquake specific
11 from obspy.clients.fdsn import Client # obspy is a well maintained toolbox for processing seismological data
12 from obspy import UTCDateTime        # https://docs.obspy.org/
13
14 mpl.rcParams['figure.dpi'] = 200 # make our figure higher resolution
```

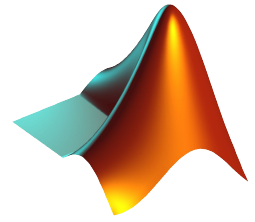
At the bottom of the code cell, a pink warning message is displayed: 'Matplotlib is building the font cache; this may take a moment. Fontconfig warning: ignoring UTF-8: not a valid region tag'.



# Python

## versus

# MATLAB



Both are high-level interpreted languages that are meant to be relatively easy to use and fast to write code in

**MATLAB** is a proprietary, closed-source software that is aimed at technical computing

- Full package including the language, IDE to write and run code, and many different functions
- Professionally maintained, easy to develop, run and debug code
- **High-performance language optimized for matrix manipulation** and program solving based on linear algebra
  - Many additional toolboxes such as image and digital signal processing
- **Expensive**, need to pay for additional toolboxes, not everyone can afford MATLAB and thus use your code

**Python** is a general purpose, **open-source** language meant to be **easy to read** and simple to implement

- Useful in science and engineering, but also used for general applications (e.g. web and game development)
- Can run on all major operating systems and CPU architectures
- Free and open-source, anyone can download, look at and modify source code – anyone can contribute!
- Generally need to add packages (e.g. Numpy, Scipy, Matplotlib) and want an IDE to write/modify code

Basic differences in syntax, significance of white space, calling functions and indexing sequences

e.g. comments in python start with “#” where as in MATLAB they use “%”

# Project Management Tips

<https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005510>

When starting a new project it is helpful to create a new directory to contain all relevant information

You might name the directory according to the research project or course assignment:

*mkdir "PathToNewDirectory/Project\_name"*

A common approach is then to create subfolders for different types of content:

- "Project\_name/doc" = text documents (manuscripts, code documentation, papers) relevant to project
- "Project\_name/src" = put code written for project
- "Project\_name/bin" = compiled programs
- "Project\_name/results" = generated files, intermediate data processing or simulated data,  
as well as final figures/tables

# Writing Readable Code

## 1. Give objects (variables, arrays, functions, etc) meaningful names:

Example:

Mass = 25.0; % kg

OR

M = 25.0; % kg

RATHER THAN

Var1 = 25.0;

Velocity = 1.5; % m/s

V = 1.5; % m/s

Var2 = 1.5;

## 2. Define functions for tasks that are regularly performed and call them throughout your program

- Make your code more compact rather than copy-pasting the same lines of code ( Python example )

## 3. Comment you code – it is challenging to comment code too much!

- Define objects (variables, functions, etc.),
- Define units for objects,
- Describe methods used to solve a problem,
- Motivation for parameter values,
- Create sections of your code with different headings,
- Draw diagrams, coordinate systems, etc.

**Comment code as your write it!**

Makes it easier for other people (and you!) to:

- Read your code,
- Understand what you intended your code to do,
- Debug your code

# Example of Commented Code in Matlab

<https://github.com/vlambert/QDBIM>

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %
3  % 2D Quasi-dynamic simulation of the evolution of
4  % slip and stress on a fault in antiplane strain
5  % governed by rate- and state-dependent friction
6  %
7  % Valere Lambert, 2018
8  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9  clear all;close all;
10
11  addpath('include/');
12  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
13  % Useful Functions
14  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
15
16  % boxcar function
17  BC=@(x) (x+0.5>=0)-(x-0.5>=0);
18
19  % Heaviside function
20  HS=@(x) 0+x>=0;
21
22  % ramp function
23  Ramp=@(x) x.*BC(x-1/2)+HS(x-1);
24

```

```

24
25  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
26  %
27  % Stress Interaction Functions
28  %
29  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
30  % Vertical fault in a half-space centered at x2 = 0
31  % 2D Antiplane problem (x2 x x3):
32  % Out-of-plane displacements u1 are non-zero, u2 = u3 = 0
33  % u1 are assumed uniform along strike, only vary in x2 x x3 plane
34  %
35  %
36  %
37  %
38  %
39  % Free surface (x3 = 0).
40  %
41  %
42  %
43  %
44  %
45  %
46  %
47  %
48  %
49  %
50  %
51  % density (kg/m^3)
52  rho = 2670;
53
54  % shear wave speed (m/s)
55  Vs = 3464;
56
57  % shear modulus (MPa)
58  G = rho*Vs^2/1e6;
59
60  % Elastostatic Green's functions for displacements and stress due to
61  % uniform slip on a rectangular patch (Okada 1985,1992)
62  % For 2D antiplane we only care about Sigma_{12}
63  % y-coordinates represent source, x-coordinates represent receiver
64
65  % Note that these solutions take into account a free-surface using
66  % the method of images
67
68  % Sigma_{12}
69  s12h=@(x2,x3,y2,y3,W) G*( ...
70  -(x3-y3)./((x2-y2).^2+(x3-y3).^2)+(x3+y3)./((x2-y2).^2+(x3+y3).^2) ...
71  +(x3-y3-W)./((x2-y2).^2+(x3-y3-W).^2)-(x3+y3+W)./((x2-y2).^2+(x3+y3+W).^2) ...
72  )/2/pi;
73

```

# Version Control and Code Repositories

Platforms for code hosting, version control and collaboration

Examples: Github, Bitbucket

<https://docs.github.com/en/get-started/quickstart/hello-world>

General idea:

1. Host copy of your code in the cloud (online) that may be shared with collaborators
2. “Pull” or download any updates to code that your collaborators have made to your computer
3. Make changes to “local” version of your code on your computer
4. Declare what changes you want to “push” to the cloud repository
5. Every time code is pushed to the cloud repository it creates a new “version” and the hosting service keeps a history of these versions, often showing the changes made between versions of code

Advantages:

- **Code backup and version control:** Copy of your code in cloud and history of “versions” of your code and track changes over time
- **Collaboration:** Share code with public repositories or private repositories with collaborators
- Useful for **Open data and software for publications** - AGU Publications data policy

## Example Git profile and list of repositories

Search or jump to... Pull requests Issues Marketplace Explore

Overview Repositories 11 Projects Packages

Find a repository... Type Language Sort New

**vlambert.github.io** Public  
CSS Updated 2 days ago

**3DBicycleWorking** Private  
Fortran 1 Updated 18 days ago

**QDBIM** Public  
MATLAB 1 Updated 19 days ago

**Valere Lambert**  
vlambert  
Edit profile  
10 followers · 6 following · 0 stars

Public and private repositories

Check different times code has been committed

vlambert / QDBIM

Public

Unwatch 1 Star 1 Fork 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 0 tags

Go to file Add file Code

vlambert Update QDBIM2D.m 3aae7dd 19 days ago 13 commits

include	update documentation	19 days ago
DieterichRuinaRegAging.m	update documentation	19 days ago
QDBIM2D.m	Update QDBIM2D.m	19 days ago
README.md	update documentation	19 days ago

README.md

This is a simple quasi-dynamic boundary element code for performing 2D antiplane simulations of sequences of earthquakes and aseismic slip (SEAS) utilizing Runge-Kutta 4/5 adaptive time-stepping to solve for the time history evolution of slip and stress.

The main file is QDBIM2D which calls the ode function DieterichRuinaRegAging that describes the state evolution of slip, shear stress, slip rate and the rate-and-state state variable following the regularized Dieterich-Ruina rate-and-state friction formulation using the aging law.

The physical problem setup in QDBIM2D is consistent with benchmark problem BP1-QD from the Southern California Earthquake Center Working Group for Advancing Simulations of Earthquakes and Aseismic Slip: [https://strike.scec.org/cvws/seas/benchmark\\_descriptions.html](https://strike.scec.org/cvws/seas/benchmark_descriptions.html)

The include directory contains additional functions related to modified versions of ode23 and ode45 to allow for the periodic output of the solution vector to disk, as otherwise the state vector history can easily exceed machine memory limits for large spatial domains and long time periods.

About

No description, website, or topics provided.

Readme

Releases

No releases published  
[Create a new release](#)

Packages

No packages published  
[Publish your first package](#)

Languages

MATLAB 100.0%



master

Commits on Oct 24, 2021

- Update QDBIM2D.m  
vlambert committed 19 days ago  
Verified [3aae7dd](#) <>
- update doc  
vlambert committed 19 days ago  
[df48d62](#) <>
- update documentation  
vlambert committed 19 days ago  
[c1e6f12](#) <>

Commits on Oct 23, 2021

- Update QDBIM2D.m  
vlambert committed 20 days ago  
Verified [cb6cc73](#) <>
- Update README.md  
vlambert committed 20 days ago  
Verified [9e62879](#) <>

Commits on Sep 13, 2018

- update readme  
vlambert committed on 13 Sep 2018  
[34915e0](#) <>

Different commits labelled by date and can also specify changes in "commit message"

\* good to write commit messages that are more descriptive of changes than shown here

Update QDBIM2D.m [Browse files](#)

master

vlambert committed 19 days ago Verified 1 parent df48d62 commit 3aae7ddcc0172df9b15ed3bc660b74a0239f9e8c

Showing 1 changed file with 2 additions and 2 deletions. Split Unified

QDBIM2D.m

```

@@ -148,10 +148,10 @@
148 148
149 149 % Estimates of some key parameters
150 150 Vwp = find(ss.a < ss.b); % VW region
151 - % Critical nucleation size ( h* = pi/2 GL / (b-a)^2 / sigma )
151 + % Critical nucleation size ( h* = pi/2 G b D_rs / (b-a)^2 / sigma )
152 152 hstar=min(pi/2*G*ss.Drs(Vwp),*ss.b(Vwp)./(ss.b(Vwp)-ss.a(Vwp)).^2./ss.sigma(Vwp));
153 153
154 - % Quasi-static cohesive zone ( coh0 = 9/32 GL/(b*sigma) )
154 + % Quasi-static cohesive zone ( coh0 = 9/32 G D_rs/(b*sigma) )
155 155 % Note that for this QD simulation the cohesive zone will not change,
156 156 % which would not be the case for a fully dynamic simulation
157 157 coh = min(9/32*pi*G*ss.Drs(Vwp)./(ss.b(Vwp)./(ss.b(Vwp)-ss.a(Vwp)).^2./ss.sigma(Vwp));

```

Can see what was changed between individual commits

# Resources

**Stack Exchange** – online forums where many questions have already been asked and answered

## Terminal commands

<https://www.techrepublic.com/article/16-terminal-commands-every-user-should-know/>

## Anaconda cheat sheet

<https://docs.conda.io/projects/conda/en/4.6.0/downloads/52a95608c49671267e40c689e0bc00ca/conda-cheatsheet.pdf>

**Common IDEs and text editors:** Spyder, PyCharm, Jupyterlab, atom, sublime, vim, emacs

## Useful python packages for science:

- NumPy (fundamental scientific computing)
- SciPy (data science)
- Matplotlib (2D plotting)
- Pandas (time series analysis),
- Scikit-learn (machine learning),
- Seaborn (pretty plots),
- Cartopy (maps)