University of Glasgow

EARTH4072 – Igneous Geology

Introduction to Computational Geosciences

WKSHP 1 | First Steps with Programming

Dr. Tobias Keller

Tobias.Keller@glasgow.ac.uk

@magmamatters

WORLD CHANGING GLASGOW

Intro Comp Geosci | Programme

Week	WKSHP I	WKSHP II	WKSHP III	WKSHP IV
19/10/2020	First Steps Coding	Comp Data Analysis	Comp Modelling I	Comp Modelling II

Comp Geosci | Intended Learning Outcomes

Introduction to Scientific Programming

- understand why scientific programming is useful
- take first steps with programming in Python
- become familiar with using Jupyter notebooks
- learn to use basic programming elements
  - variables, lists, arrays
  - indexing into lists and arrays
  - for- and while-loops
  - logic and if-conditions
  - functions
- know and apply best practices for programming

```
% update constitutive relations
txx = eta .* exx + chi .* txxo;       % x-normal stress
tzz = eta .* ezz + chi .* tzzo;       % z-normal stress
txz = etac.* exz + chic.* txzo;       % xz-shear stress

p   = - zeta .* Div_V + xi .* po;     % compaction pressure
p([1 end],:) = p([end-1 2],:);        % periodic boundaries
p(:,[1 end]) = p(:,[end-1 2]);

w   = - (K(1:end-1,:).*K(2:end,:)).^0.5 .* (diff(P,1,1)./h + 1);
w(:,[1 end]) = w(:,[end-1 2]);

u   = - (K(:,1:end-1).*K(:,2:end)).^0.5 .* (diff(P,1,2)./h);
u([1 end],:) = u([end-1 2],:);

% update z-reference velocity
Div_tz = diff(tzz(:,2:end-1),1,1)./h + diff(txz,1,2)./h;

res_W(:,2:end-1) = - Div_tz + diff(P(:,2:end-1),1,1)./h + diff(p(:,

res_W([1 end],:) = [sum(res_W([1 end],:),1)./2;sum(res_W([1 end],:)
res_W(:,[1 end]) = res_W(:,[end-1 2]);

W = Wi - alpha.*res_W.*dtW + beta.*(Wi-Wii);

% update x-reference velocity
Div_tx  = diff(txx(2:end-1,:),1,2)./h + diff(txz,1,1)./h;

res_U(2:end-1,:) = - Div_tx + diff(P(2:end-1,:),1,2)./h + diff(p(2:

res_U([1 end],:) = res_U([end-1 2],:);
res_U(:,[1 end]) = [sum(res_U(:,[1 end]),2)./2,sum(res_U(:,[1 end])

U = Ui - alpha.*res_U.*dtU + beta.*(Ui-Uii);

% update reference pressure
Div_V(2:end-1,2:end-1) = diff(U(2:end-1,:),1,2)./h + diff(W(:,2:end
Div_v(2:end-1,2:end-1) = diff(u(2:end-1,:),1,2)./h + diff(w(:,2:end

res_P = Div_V + Div_v;

res_P([1,end],:) = res_P([end-1,2],:);
res_P(:,[1,end]) = res_P(:,[end-1,2]);

P = Pi - alpha.*res_P.*dtP + beta.*(Pi-Pii);

% update liquid evolution equation (enforce min/max limits on f)

flxdiv_fromm;  % upwind-biased advection/compaction term for liquid

res_f = (f-fo)./dt - (theta.*Div_fV + (1-theta).*Div_fVo);

res_f([1,end],:) = res_f([end-1,2],:);
res_f(:,[1,end]) = res_f(:,[end-1,2]);

if ~mod(step,nop); res_f = res_f - mean(res_f(:)); end

f = fi - alpha.*res_f.*dt/50;
f = max(0.001/f0,min(0.999/f0, f ));

% check and report convergence every nup iterations
if ~mod(it,nup); report; end
```

Setting Expectations

How do you feel about programming

- Excited?
- Not bothered?
- Somewhat apprehensive?

What do you want to get out of this?

- excited to learn a new tool
- analysing and plotting data
- modelling natural processes

=> share your answers on Padlet

My story

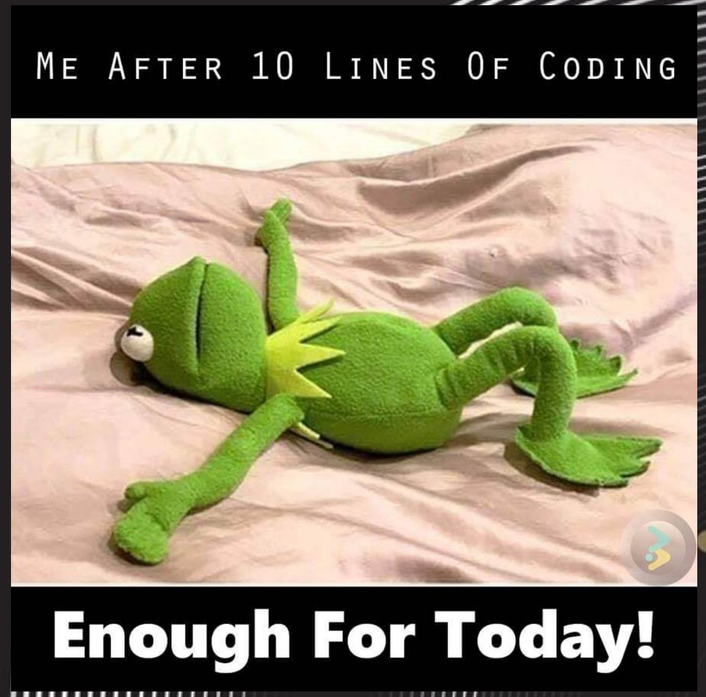
From least tech-savvy student to Lecturer in Comp Geosci in ten years.

–

If I could do it, so can you!

What's the trick?

- stay curious!
- Google it!
- try and try again!
- it's easier than you think!



University of Glasgow

Scientific Programming | Motivation

Why?

Flexibility

- custom-built tools for wide range of scientific tasks

Productivity

- automate workflows, work with big data, large calculations

Reproducibility

- others can repeat entire workflow at push of button

```
% update constitutive relations
txx = eta .* exx + chi .* txxo;       % x-normal stress
tzz = eta .* ezz + chi .* tzzo;       % z-normal stress
txz = etac.* exz + chic.* txzo;       % xz-shear stress

p   = - zeta .* Div_V + xi .* po;     % compaction pressure
p([1 end],:) = p([end-1 2],:);        % periodic boundaries
p(:,[1 end]) = p(:,[end-1 2]);

w   = - (K(1:end-1,:).*K(2:end,:)).^0.5 .* (diff(P,1,1)./h + 1);
w(:,[1 end]) = w(:,[end-1 2]);

u   = - (K(:,1:end-1).*K(:,2:end)).^0.5 .* (diff(P,1,2)./h);
u([1 end],:) = u([end-1 2],:);

% update z-reference velocity
Div_tz = diff(tzz(:,2:end-1),1,1)./h + diff(txz,1,2)./h;

res_W(:,2:end-1) = - Div_tz + diff(P(:,2:end-1),1,1)./h + diff(p(:,

res_W([1 end],:) = [sum(res_W([1 end],:),1)./2;sum(res_W([1 end],:)
res_W(:,[1 end]) = res_W(:,[end-1 2]);

W = Wi - alpha.*res_W.*dtW + beta.*(Wi-Wii);

% update x-reference velocity
Div_tx  = diff(txx(2:end-1,:),1,2)./h + diff(txz,1,1)./h;

res_U(2:end-1,:) = - Div_tx + diff(P(2:end-1,:),1,2)./h + diff(p(2:

res_U([1 end],:) = res_U([end-1 2],:);
res_U(:,[1 end]) = [sum(res_U(:,[1 end]),2)./2,sum(res_U(:,[1 end])

U = Ui - alpha.*res_U.*dtU + beta.*(Ui-Uii);

% update reference pressure
Div_V(2:end-1,2:end-1) = diff(U(2:end-1,:),1,2)./h + diff(W(:,2:end
Div_v(2:end-1,2:end-1) = diff(u(2:end-1,:),1,2)./h + diff(w(:,2:end

res_P = Div_V + Div_v;

res_P([1,end],:) = res_P([end-1,2],:);
res_P(:,[1,end]) = res_P(:,[end-1,2]);

P = Pi - alpha.*res_P.*dtP + beta.*(Pi-Pii);

% update liquid evolution equation (enforce min/max limits on f)

flxdiv_fromm;  % upwind-biased advection/compaction term for liquid

res_f = (f-fo)./dt - (theta.*Div_fV + (1-theta).*Div_fVo);

res_f([1,end],:) = res_f([end-1,2],:);
res_f(:,[1,end]) = res_f(:,[end-1,2]);

if ~mod(step,nop); res_f = res_f - mean(res_f(:)); end

f = fi - alpha.*res_f.*dt/50;
f = max(0.001/f0,min(0.999/f0, f ));

% check and report convergence every nup iterations
if ~mod(it,nup); report; end
```

Scientific Programming

Ideas and concepts

programming language

computer code

Programming Languages

purpose-driven vs. all-purpose

- some developed for specific use
- others used for many tasks

high- vs. low-level

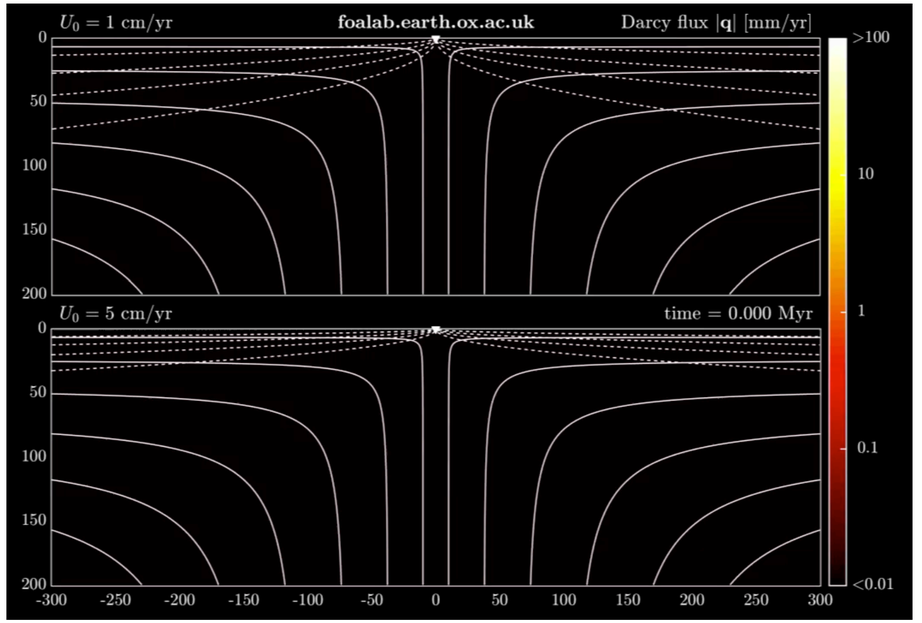
- low-level: close to computer language
- high-level: close to human language

interpreted vs. compiled

- interpreted: run code at push of button
- compiled: first compile code before run

open vs. licensed

- choose open where possible



Scientific Programming

ideas and concepts

programming language

computer code

Programming Languages

R

- high-level, open, good for statistics, data visualisation, data science

Matlab

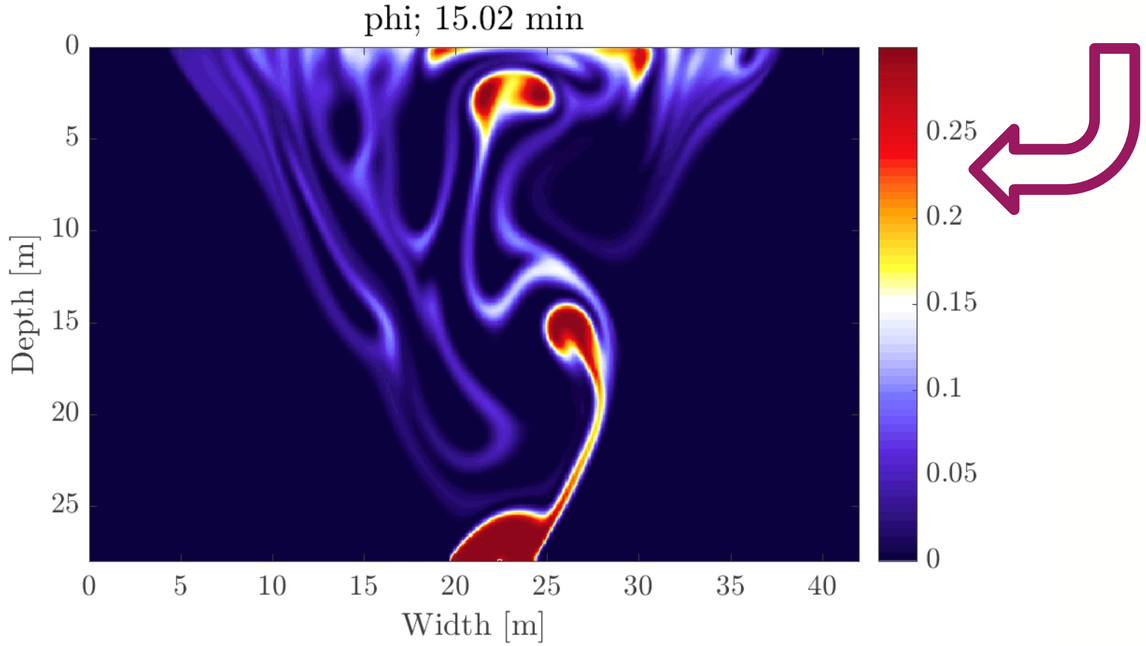
- high-level, licensed, good for numerical modelling, linear algebra

Python

- moderately high-level, open, widely used for almost anything

C / C++ / Fortran

- lower-level, open, widely used



Scientific Programming | Terminology

programme

script

loop

instruction

condition

code

command

algorithm

function

Scientific Programming | Terminology

instruction / ***command***	line of code instructing computer to perform one or a set of specific operations
condition	make following command conditional on whether a specified logic statement is true or false
loop	set of instructions executed several times in a row
function	set of instructions executed when function is called
algorithm	sequence of instructions, conditions, loops, functions designed to produce specific result
script	text file containing one or several instructions, functions, or algorithms
programme / ***code***	collection of scripts that form a computational tool

Scientific Programming | Best Practice

compare the codes!

- do you think they do the same thing?
- which one is easier to read, understand?
- what makes one better than the other?

```
 x0  = "./data/co2_mm_mlo.txt"

y, cc = np.loadtxt(x0, usecols=(2,4), unpack=True)
# print
print(y[:5])


 print(cc[:5])
# plot
  f = plt.figure(1,figsize=(10,6))


    ff  = f.add_subplot(111)
ff.plot(y,cc,'k-');
```

```
# load the Mauna Loa CO2 data, which is an array (table) with many columns of different data.
# We are only using two columns,
# the decimal date (column 2) and the monthly average CO2 concentrations (column 4),
# then unpack the results into two variables, Date and CO2

# load the dates and CO2 data into numpy arrays using the numpy function 'loadtxt'
Filename  = "./data/co2_mm_mlo.txt"
Date, CO2 = np.loadtxt(Filename, usecols=(2,4), unpack=True)

# let's print the first five values of each array to get an idea what the data looks like
print('Dates: ',Date[:5])
print('mean monthly CO2: ',CO2[:5])

# prepare figure
fig1 = plt.figure(1,figsize=(10,6))  # create figure to plot in
ax1  = fig1.add_subplot(111)  # create set of axes to plot in. (111) means "plot 1 in a 1-by-1 grid of subplots"

# plot CO2 time series
ax1.plot(Date,CO2,'k-');
```

Scientific Programming | Best Practice

documentation

- clearly document code
- add file headers
- comment blocks
- add readme file
- add how-to guide

```
 x0  = "./data/co2_mm_mlo.txt"

y, cc = np.loadtxt(x0, usecols=(2,4), unpack=True)
# print
print(y[:5])


 print(cc[:5])
# plot
  f = plt.figure(1,figsize=(10,6))


    ff  = f.add_subplot(111)
ff.plot(y,cc,'k-');
```

```
# load the Mauna Loa CO2 data, which is an array (table) with many columns of different data.
# We are only using two columns,
# the decimal date (column 2) and the monthly average CO2 concentrations (column 4),
# then unpack the results into two variables, Date and CO2

# load the dates and CO2 data into numpy arrays using the numpy function 'loadtxt'
Filename  = "./data/co2_mm_mlo.txt"
Date, CO2 = np.loadtxt(Filename, usecols=(2,4), unpack=True)

# let's print the first five values of each array to get an idea what the data looks like
print('Dates: ',Date[:5])
print('mean monthly CO2: ',CO2[:5])

# prepare figure
fig1 = plt.figure(1,figsize=(10,6))  # create figure to plot in
ax1  = fig1.add_subplot(111)  # create set of axes to plot in. (111) means "plot 1 in a 1-by-1 grid of subplots"

# plot CO2 time series
ax1.plot(Date,CO2,'k-');
```

Scientific Programming | Best Practice

commenting

- clearly comment code
- comment over blocks
- comment in-line
- describe & explain
- concise, clear language

```
 x0  = "./data/co2_mm_mlo.txt"

y, cc = np.loadtxt(x0, usecols=(2,4), unpack=True)
# print
print(y[:5])


 print(cc[:5])
# plot
  f = plt.figure(1,figsize=(10,6))


    ff  = f.add_subplot(111)
ff.plot(y,cc,'k-');
```

```
# load the Mauna Loa CO2 data, which is an array (table) with many columns of different data.
# We are only using two columns,
# the decimal date (column 2) and the monthly average CO2 concentrations (column 4),
# then unpack the results into two variables, Date and CO2

# load the dates and CO2 data into numpy arrays using the numpy function 'loadtxt'
Filename  =  ./data/co2_mm_mlo.txt
Date, CO2 = np.loadtxt(Filename, usecols=(2,4), unpack=True)

# let's print the first five values of each array to get an idea what the data looks like
print('Dates: ',Date[:5])
print('mean monthly CO2: ',CO2[:5])

# prepare figure
fig1 = plt.figure(1,figsize=(10,6))  # create figure to plot in
ax1  = fig1.add_subplot(111)  # create set of axes to plot in. (111) means "plot 1 in a 1-by-1 grid of subplots"

# plot CO2 time series
ax1.plot(Date,CO2,'k-');
```

Scientific Programming | Best Practice

structure

- clearly structure code
- divide into logical blocks
- consistent line spacing
- consistent indentation

```
x0  = "./data/co2_mm_mlo.txt"

y, cc = np.loadtxt(x0, usecols=(2,4), unpack=True)
# print
print(y[:5])


 print(cc[:5])
# plot
  f = plt.figure(1,figsize=(10,6))


    ff  = f.add_subplot(111)
ff.plot(y,cc,'k-');
```

```
# load the Mauna Loa CO2 data, which is an array (table) with many columns of different data.
# We are only using two columns,
# the decimal date (column 2) and the monthly average CO2 concentrations (column 4),
# then unpack the results into two variables, Date and CO2

# load the dates and CO2 data into numpy arrays using the numpy function 'loadtxt'
Filename  = "./data/co2_mm_mlo.txt"
Date, CO2 = np.loadtxt(Filename, usecols=(2,4), unpack=True)

# let's print the first five values of each array to get an idea what the data look like
print('Dates: ',Date[:5])
print('mean monthly CO2: ',CO2[:5])

# prepare figure
fig1 = plt.figure(1,figsize=(10,6))  # create figure to plot in
ax1  = fig1.add_subplot(111)  # create set of axes to plot in. (111) means "plot 1 in a 1-by-1 grid of subplots"

# plot CO2 time series
ax1.plot(Date,CO2,'k-');
```

Scientific Programming | Best Practice

naming

- use descriptive names
- avoid ambiguity
- avoid cryptic names
- consistent conventions

```
x0  = "./data/co2_mm_mlo.txt"

y, cc = np.loadtxt(x0, usecols=(2,4), unpack=True)
# print
print(y[:5])


 print(cc[:5])
# plot
 f = plt.figure(1,figsize=(10,6))


   ff = f.add_subplot(111)
ff.plot(y,cc,'k-');
```

```
# load the Mauna Loa CO2 data, which is an array (table) with many columns of different data.
# We are only using two columns,
# the decimal date (column 2) and the monthly average CO2 concentrations (column 4),
# then unpack the results into two variables, Date and CO2

# load the dates and CO2 data into numpy arrays using the numpy function 'loadtxt'
Filename    = "./data/co2_mm_mlo.txt"
Date, CO2 = np.loadtxt(Filename, usecols=(2,4), unpack=True)

# let's print the first five values of each array to get an idea what the data looks like
print('Dates: ',Date[:5])
print('mean monthly CO2: ',CO2[:5])

# prepare figure
fig1 = plt.figure(1,figsize=(10,6))  # create figure to plot in
ax1  = fig1.add_subplot(111)  # create set of axes to plot in. (111) means "plot 1 in a 1-by-1 grid of subplots"

# plot CO2 time series
ax1.plot(Date,CO2,'k-');
```

University of Glasgow

WKSHP I | First Steps with Sci Programming

Activity | Introduction to Programming with Python

Get started with *Jupyter notebooks*

- text cells and code cells
- use *Markdown* commands to format text cells
- use *Python* to program code cells

Get started with basic Python commands

- *calculator*: addition/subtraction, multiplication/division, power
- assign variables, use lists and indexing

Basic programming elements

- learn to use logic conditions, loops, and functions
- write your first simple algorithm!

```
% update constitutive relations
txx = eta .* exx + chi .* txxo;       % x-normal stress
tzz = eta .* ezz + chi .* tzzo;       % z-normal stress
txz = etac.* exz + chic.* txzo;       % xz-shear stress

p   = - zeta .* Div_V + xi .* po;     % compaction pressure
p([1 end],:) = p([end-1 2],:);        % periodic boundaries
p(:,[1 end]) = p(:,[end-1 2]);

w   = - (K(1:end-1,:).*K(2:end,:)).^0.5 .* (diff(P,1,1)./h + 1);
w(:,[1 end]) = w(:,[end-1 2]);

u   = - (K(:,1:end-1).*K(:,2:end)).^0.5 .* (diff(P,1,2)./h);
u([1 end],:) = u([end-1 2],:);

% update z-reference velocity
Div_tz = diff(tzz(:,2:end-1),1,1)./h + diff(txz,1,2)./h;

res_W(:,2:end-1) = - Div_tz + diff(P(:,2:end-1),1,1)./h + diff(p(:,

res_W([1 end],:) = [sum(res_W([1 end],:),1)./2;sum(res_W([1 end],:)
res_W(:,[1 end]) = res_W(:,[end-1 2]);

W = Wi - alpha.*res_W.*dtW + beta.*(Wi-Wii);

% update x-reference velocity
Div_tx  = diff(txx(2:end-1,:),1,2)./h + diff(txz,1,1)./h;

res_U(2:end-1,:) = - Div_tx + diff(P(2:end-1,:),1,2)./h + diff(p(2:

res_U([1 end],:) = res_U([end-1 2],:);
res_U(:,[1 end]) = [sum(res_U(:,[1 end]),2)./2,sum(res_U(:,[1 end])

U = Ui - alpha.*res_U.*dtU + beta.*(Ui-Uii);

% update reference pressure
Div_V(2:end-1,2:end-1) = diff(U(2:end-1,:),1,2)./h + diff(W(:,2:end
Div_v(2:end-1,2:end-1) = diff(u(2:end-1,:),1,2)./h + diff(w(:,2:end

res_P = Div_V + Div_v;

res_P([1,end],:) = res_P([end-1,2],:);
res_P(:,[1,end]) = res_P(:,[end-1,2]);

P = Pi - alpha.*res_P.*dtP + beta.*(Pi-Pii);

% update liquid evolution equation (enforce min/max limits on f)

flxdiv_fromm;  % upwind-biased advection/compaction term for liquid

res_f = (f-fo)./dt - (theta.*Div_fV + (1-theta).*Div_fVo);

res_f([1,end],:) = res_f([end-1,2],:);
res_f(:,[1,end]) = res_f(:,[end-1,2]);

if ~mod(step,nop); res_f = res_f - mean(res_f(:)); end

f = fi - alpha.*res_f.*dt/50;
f = max(0.001/f0,min(0.999/f0, f ));

% check and report convergence every nup iterations
if ~mod(it,nup); report; end
```