Setting up your computing environment

Installation

Operating system and environment:

MAC or Linux are the preferred operating system in this course. Windows can be used, but here we do not provide instructions of how to install the required software.

This year we provide *virtual machine* (light linux ubuntu distribution), which can be run on any operating system. This is useful for those who do not want (or do not succeed) to install the necessary software on their own computer. Or you might want to try the virtual machine just for fun!

Essential Software:

- Python, and its packages in particular numpy, scipy, matplotlib, weave (There is a nice package ”Enthought Python Distribution” http://www.enthought.com/products/edudownload.php which allows easy installation on every operating system.)
- C++ compiler, such as gcc, but any other C++ compiler should be fine.
- Text editor for coding (for example Emacs, Aquamacs, Enthought's IDLE)
- make to execute makefiles (not necessary on windows)

Highly Recommended Software:

- Fortran compiler, such as gfortran or intel fortran
- BLAS& LAPACK library for linear algebra (most likely provided by vendor)
- open_mp enabled fortran and C++ compiler

Very Useful Software:

- gnuplot for fast plotting.
- gsl (Gnu scientific library) for implementation of various scientific algorithms.

If you want to use your own Windows machine, you should find a way to install these packages in a functional way.

Installing virtual box

If you do not want/succeed to install the necessary software, you should download the file : `http://hauleweb.rutgers.edu/downloads/509/509.ova` (warning: 4.8GB file, it might take a while)

Then you should install **VirtualBox** to run the provided virtual machine: `https://www.virtualbox.org`

Finally, start the **VirtualBox** and navigate to *File/Import Appliance*, and choose the downloaded *509.ova* file.

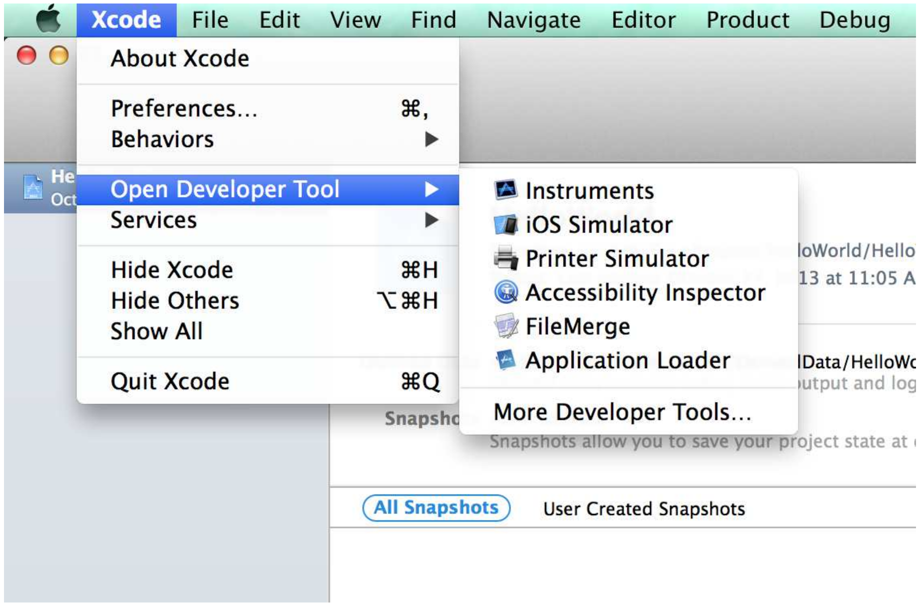
Then click *Start* and wait for the linux to start. Once linux is running, you can start a terminal *Konsole* and start *emacs* in the terminal. You can navigate to

```
~/ComputationalPhysics/mandelbroat
```

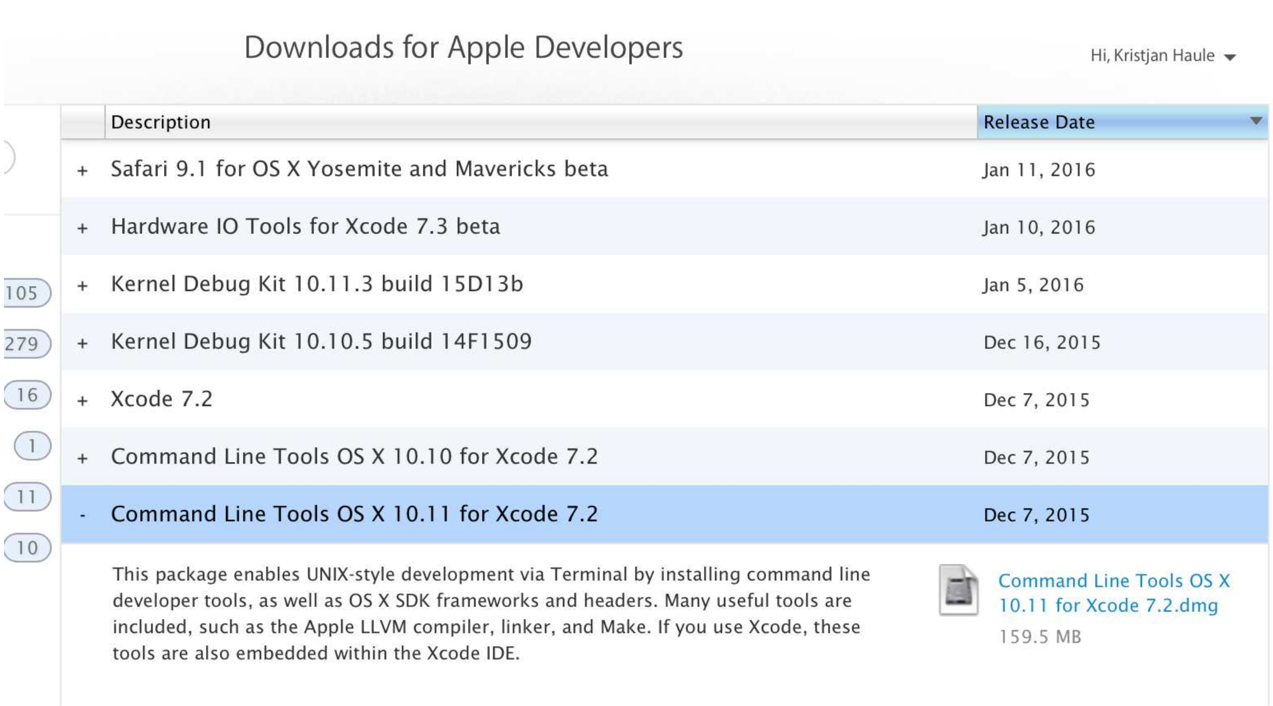
and examine the files we will discuss in the first lecture. If you need username, use *student*, and *student123*.

Installation on MAC

- Install **Xcode** package from App Store.
- Install ``Command Line Tools'' from Apple’s software site. For Mavericks and lafter, open Xcode program, and choose from the menu Xcode -> Open Developer Tool -> More Developer Tools...



You will be linked to the Apple page that allows you to access downloads for Xcode. You wil have to register as a developer (free). Search for the Xcode Command Line Tools in the search box in the upper left. Download and install the correct version of the Command Line Tools, for example for OS ”El Capitan” and Xcode 7.2, you need `Command Line Tools OS X 10.11 for Xcode 7.2`



Downloads for Apple Developers

Hi, Kristjan Haule

	Description	Release Date
+	Safari 9.1 for OS X Yosemite and Mavericks beta	Jan 11, 2016
+	Hardware IO Tools for Xcode 7.3 beta	Jan 10, 2016
+	Kernel Debug Kit 10.11.3 build 15D13b	Jan 5, 2016
+	Kernel Debug Kit 10.10.5 build 14F1509	Dec 16, 2015
+	Xcode 7.2	Dec 7, 2015
+	Command Line Tools OS X 10.10 for Xcode 7.2	Dec 7, 2015
-	Command Line Tools OS X 10.11 for Xcode 7.2	Dec 7, 2015

This package enables UNIX-style development via Terminal by installing command line developer tools, as well as OS X SDK frameworks and headers. Many useful tools are included, such as the Apple LLVM compiler, linker, and Make. If you use Xcode, these tools are also embedded within the Xcode IDE.

Command Line Tools OS X 10.11 for Xcode 7.2.dmg
159.5 MB

Apple's Xcode contains many libraries and compilers for Mac systems. For example, it should include `BLAS` and `LAPACK` libraries. To use these libraries, we will use a linker option: `-framework Accelerate`.

However, Xcode does not come anymore with `gnu` compilers (such as `gnu-c==gcc` and `gnu-c++==g++`). Instead `gcc` and `g++` point to apple's own `Clang` compiler. Unfortunately `Clang` does not yet support `open_mp` (multicore) instructions. Moreover, `Clang` does not include `fortran` compiler, while gnu project does. In the newest versions of `gcc`, `gfortran` is already included as part of the project.

- Next we will install `homebrew` for easier installation of `gcc, fortran, gsl & gnuplot`.
  The long instructions of how to install the `homebrew` are available at `http://brew.sh`. In summary, you need to paste the following into the Terminal prompt:

  ```
  /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
  ```

  and follow the instructions.

- After installing homebrew, you can check for any issues with the install by typing

  ```
  brew doctor
  ```

  On some OSX versions you may run into file permission issues with OSX's new SIP process. To get around this you can change ownership on the `homebrew` directory by typing:

  ```
  sudo chown -R $(whoami):admin /usr/local
  ```

  Now that homebrew is installed, we can install `gcc` and `gsl`.

- Install `gcc` with included `gfortran`:
  - To check the gcc package, you can type in the Terminal prompt

```
brew search gcc
```

- To install `gcc` and `gfortran` type:

  ```
  brew install gcc --without-multilib
  ```

  the extra option adds `open_mp` support.

Warning: this will take a lot of time

- `Gnu scientific library`:
  - To check if available type

    ```
    brew search gsl
    ```

  - To insall gsl type

    ```
    brew install gsl
    ```

- To install `gnuplot` type:

  ```
  brew install gnuplot --with-x11
  ```

  If you previously unsuccessfully installed gnuplot, then first type:

  ```
  brew uninstall gnuplot
  ```

  followed by

  ```
  brew install gnuplot --with-x11
  ```

- Finally, we will install Python with its packages. It should always be installed after Xcode was installed. Xcode comes with a stripped-down version of Python, however, native python usually does not contain scipy support. To check the current python installation, you can type

  ```
  python
  import scipy
  scipy.test()
  ```

  If it survives for 10 seconds or more, the installation is OK. If not, follow the instructions below.

  Note: In this course, we will use **Python 2.7** (the alternative is 3.6).

  (1) The easiest installation of Python and its packages (numpy, scipy, matplotlib) is provided by ”Enthought Python Distribution” (`http://www.enthought.com/products/edudownload.php`). (The drawback: it is a commercial package which you get for free for a limited time of one year, provided you give your rutgers email.)

  ∗ Start at

`http://www.enthought.com/products/edudownload.php` and register using rutgers email. You will get email to activate you account and download Enthought for one year. Follow the instructions in the email. Steps are summarized below (might be outdated).

∗ Download using link from the email.

∗ navigate to ”installers” and click on the version for your OS.

∗ Now you can start “Canopy” and check in the “package manager” which packages are installed. You should see `scipy`, `numpy`, `matplotlib`. You should install also `weave` package. To see that scipy works, you will also need `pytest`.

∗ you can also type `python` in the command line to check it out. You can now repeat the above check for `scipy` installation.

(2) An alternative (to ”Enthought”) is ”Anacoda” distribution: `https://www.anaconda.com` You should install this only if you do not want to use ”Enthought”. Instructions to install can be found here: `https://www.anaconda.com/download/`
”Anaconda” distribution is available also for linux and windows.

(3) Yet another alternative of installing Python with scipy/numpy/matplotlib on MAC: In

version 10.7x on MAC, one can use the following steps:

Install a second package manager named **pip** . Do the following

∗ *sudo easy_install pip*

∗ *sudo pip install numpy*

∗ *sudo pip install scipy*

∗ *sudo pip install ipython*

∗ *sudo pip install matplotlib*

∗ You can type ”python” in the terminal and start using it :-)

(4) Similar to the above installation is :

`http://docs.python-guide.org/en/latest/starting/installation/`

Note that this page contains installation instructions for other operating systems, and you might want to try it when other (more straighforward) methods fail.

(5) Even more ways of installing python with scipy/numpy/matplotlib on MAC:

∗ compile/install scipy and numpy from source yourself (this will always work, but you need some experience). For more info see `https://www.python.org/downloads/` and `http://www.scipy.org/Installing_SciPy/Mac_OS_X`.

∗ Distribution Scipy superpack, a bunch of precompiled binaries `http://fonnesbeck.github.com/ScipySuperpack/`

∗ Distribution MacPorts, probably the most flexible option that allows you to install and maintain a python distribution `http://www.macports.org`

Installation on Linux/Ubuntu

0.0.1 Operating System

If you are a linux fan (as I was for a long time), you will need to use steps which are sketched below. Note that these instructions are quite outdated as they are many years old, but I hope still useful.

Currently one of the most popular brands of linux is Ubuntu (Apart from Android, but we can not do computation on the smart phone yet) .
Very brief instructions for installing Ubuntu:

- Goto `http://www.ubuntu.com/getubuntu/download`
- Download the current version with long support (18.04.1 LTS LTS). Save the ISO image.
- Familiarize yourself with installation instructions: `https://tutorials.ubuntu.com/tutorial/tutorial-install-ubu`
- **Be very carefull:** If you want to keep your windows, you need to make a backup of your windows system first. You have to choose to ”shrink” the existing partition and use free space on your hard drive. Do not ”erase entire disc”.

If you have experience, you can manually edit partition table: resize the windows partition to smaller size, add linux partition (one big ext3 partition and one small swap partition).

0.0.2 Linux/Ubuntu software

Install the following software:

- Install most common development tools: gcc, gdb,...
  In Ubuntu, you can do the following:
  - goto: System/Administration/Synaptic Package Manager
  - Search for ”build-essential”, and ”Mark for installation”
  - ”Apply”
- Install ”gnuplot” (using Synaptic Package Manager)
- Install ”emacs” (using Synaptic Package Manager)
- Install ”Intel math kernel library”, which includes blas and lapack (see instructions below)

- Install ”Fortran compiler” (for Intel compiler see instructions below). Far easier is installation of gfortran, which can be achieved through Synamptic Manager, and should be enough for us.
- Check if Python is installed (name of the package ”python” and ”python-dev” in Synaptic Package Manager)
- Install ”python-numpy” using Synaptic Package Manager (numeric python)
- Install ”python-scipy” using Synaptic Package Manager (scientific python)
- Install ”python-matplotlib” using Synaptic Package Manager (for plotting).
- Install ”ipython” using Synaptic Package Manager (nice frontend).

0.0.3 Intel Fortran Compiler on linux (optional):

Intel fortran compile is probably the best fortran compiler on the market. If you want to develop computationally intensive code in fortran, you should install it. For c/c++ gnu compile is comparable to Intel, hence no need to install intel c++, although intel c++ might be slightly faster.

Unfortunately, Intel is making
it very hard to install its compilers. You can google *Free Tools for Students, Intel* or start here `https://software.intel.com/en-us/qualify-for-free-software/s`

This is not necessary anymore for this class, as we can do evrything needed with gfortran compiler (GNU version).

Why SciPy?

Because it comes with great packages (very well tested). Most of them written in Fortran and wrapped into Python. Available subpackages:

- fft, fft2, fftn – discrete Fourier transform
- fftpack
- integrate - Numeric Integration
- interpolate - Interpolation, including multidimensional splines!
- linalg - All routines of linear algebra from LAPACK and BLAS
- optimize - Great minimization package
- sparse - suport for sparse matrices
- special - special functions like Bessel, Error and Air functions,...
- stats - statistics
- fast input, output (loadtxt, savetxt)

Introduction to computing in high level languages

We will use the following programming languages:

- Fotran

  We will learn how to use existing fortran subroutine and packages in modern languages.

  ↑ A lot of scientific code written in fortran → we need to use it.

  ↓ For todays standards, it is obsolete.

  – Developed by IBM in the 1950s (John W. Backus 1953).

  ↓ Many releases (Fortran, Fortran II, Fortran III, Fortran 66, Fortran 77, Fortran 90, Fortran 95, Fortran 2003, Fortran 2008).

  ↓ The language keeps changing substantially (Problems with maintainability).

  ↓ Many implementations, but no standard compiler: Intel fortran, gnu, PGI fortran,...

- C++

  Nowadays, most common language for numerical algorithms (when speed is crucial).

  ↑ Execution nowadays as fast as fortran (or C). It was developed for system programming (unix kernel): very fast, general purpose, great for very large scale problems.

  ↑ Very powerful: object oriented, supports templates, exceptions (C with classes).

  – Middle-level language: Not for writing front-end or web-page.

  – Developed by Bjarne Stroustrup in 1979 at Bell Labs.

  ↑ Mature and solid language: ANSI-ISO standard implemented some years ago. First C++ standard ratified in 1998, few corrections were made in 2003 due to ”defect reports”.

  ↑ Very popular implementation of ANSI standard under gnu licence: **gcc**.

  ↓ More complex than most of other modern languages. Very hard to fully master it.

- Perl

  ↑ Very popular scripting language (mainly due to web scripting ”cgi-scripts”).

  ↑ Open source, developed by Larry Wall from NASA, in 1987.

  ↑ Comprehensive Perl Archive Network (CPAN) repository contains more than 13,500 modules developed by more than 6,500 authors.

  ↓ An expert converted from Perl to Python: ”... He replied that Perl was fine up to a hundred lines or so, but beyond that he sort of hit a wall, and it got hard to manage the complexity. He now thought Python was a much better language for readability, maintainability, and modifiability.”

  ↓ Loosing agains python lately.

- Python

  Recently it is gaining popularity in the world of scripting.

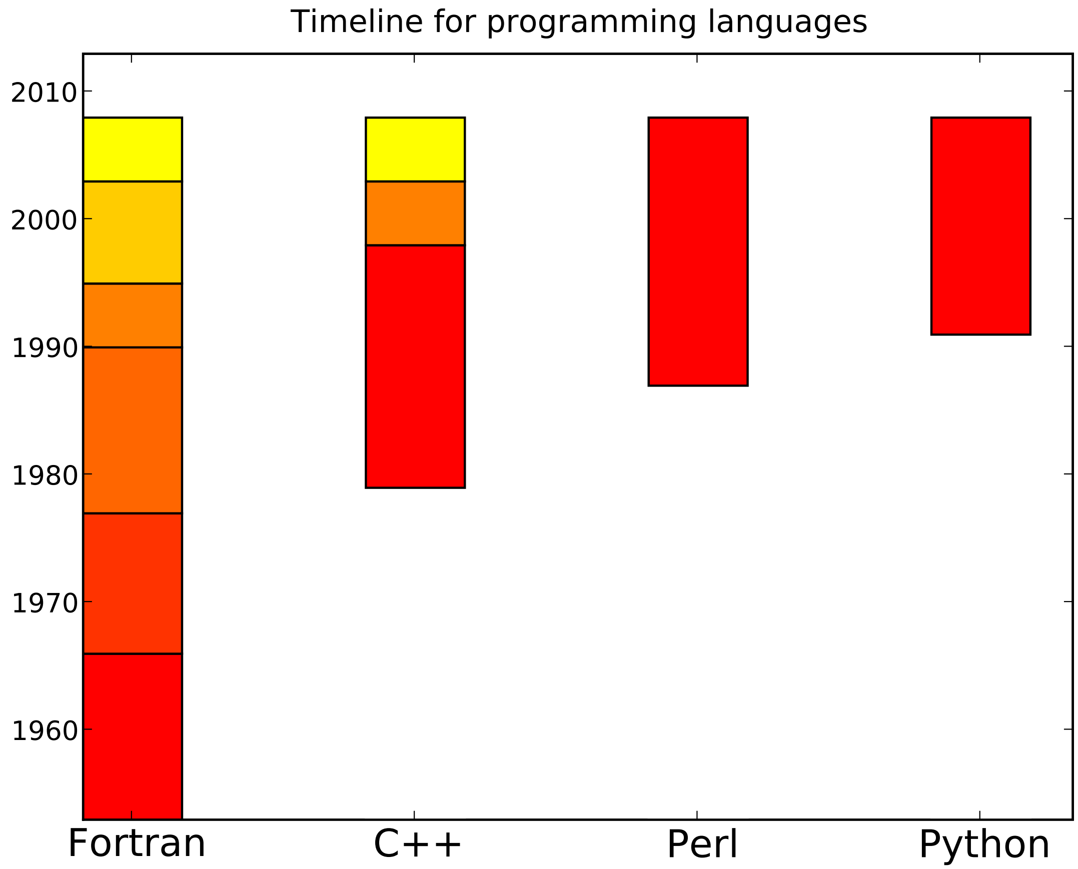
  ↑ Easier to maintain and read than perl code.

  ↑ Open source.

  ↑ Excellent tools for web development (cherryPy, Cheetah).

  ↑ Relatively young: first released by Guido van Rossum in 1991.

  ↓ Very alive: version 3.2 released January 16, 2011. Unfortunately many modifications in 3.x!

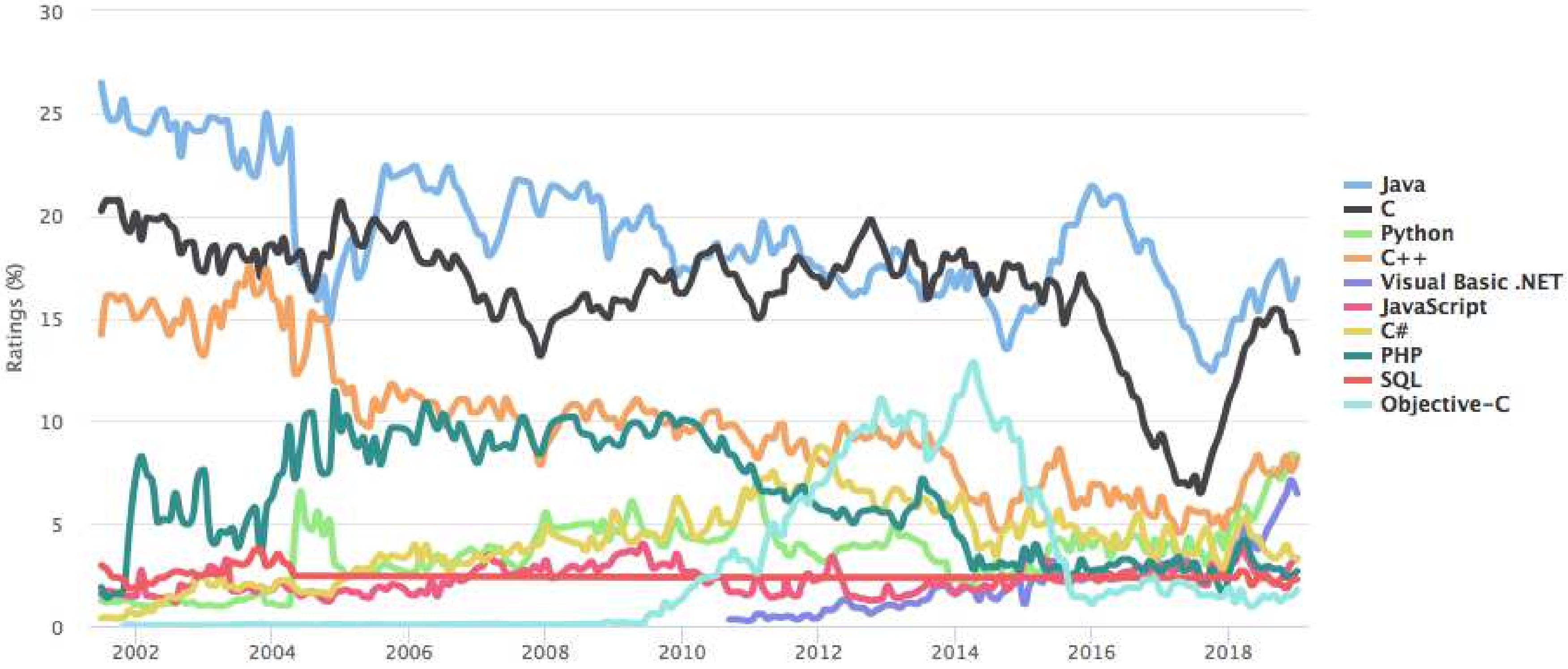


Note that `Swift` was announced in 2014 (very new language), developed by Apple.

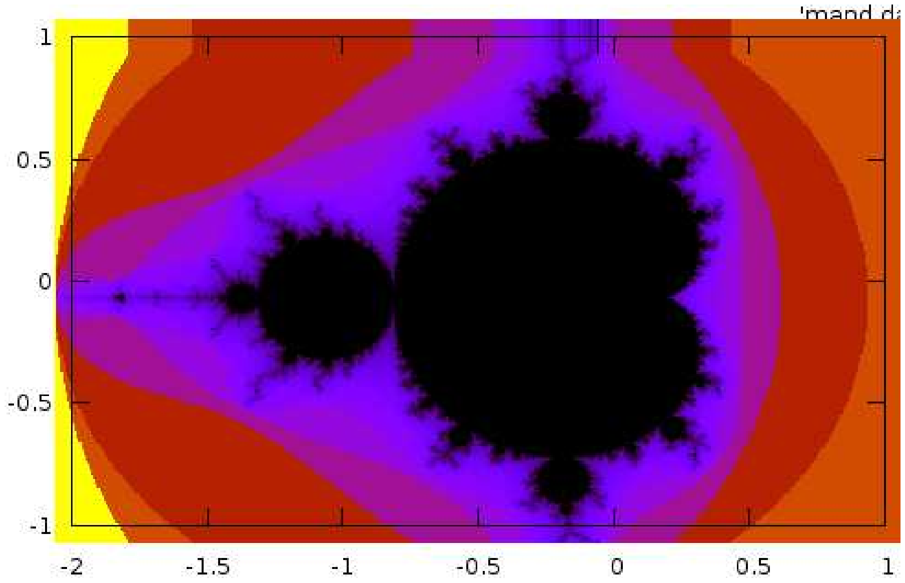
Which language is most popular

0.1 Timeline 2019

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

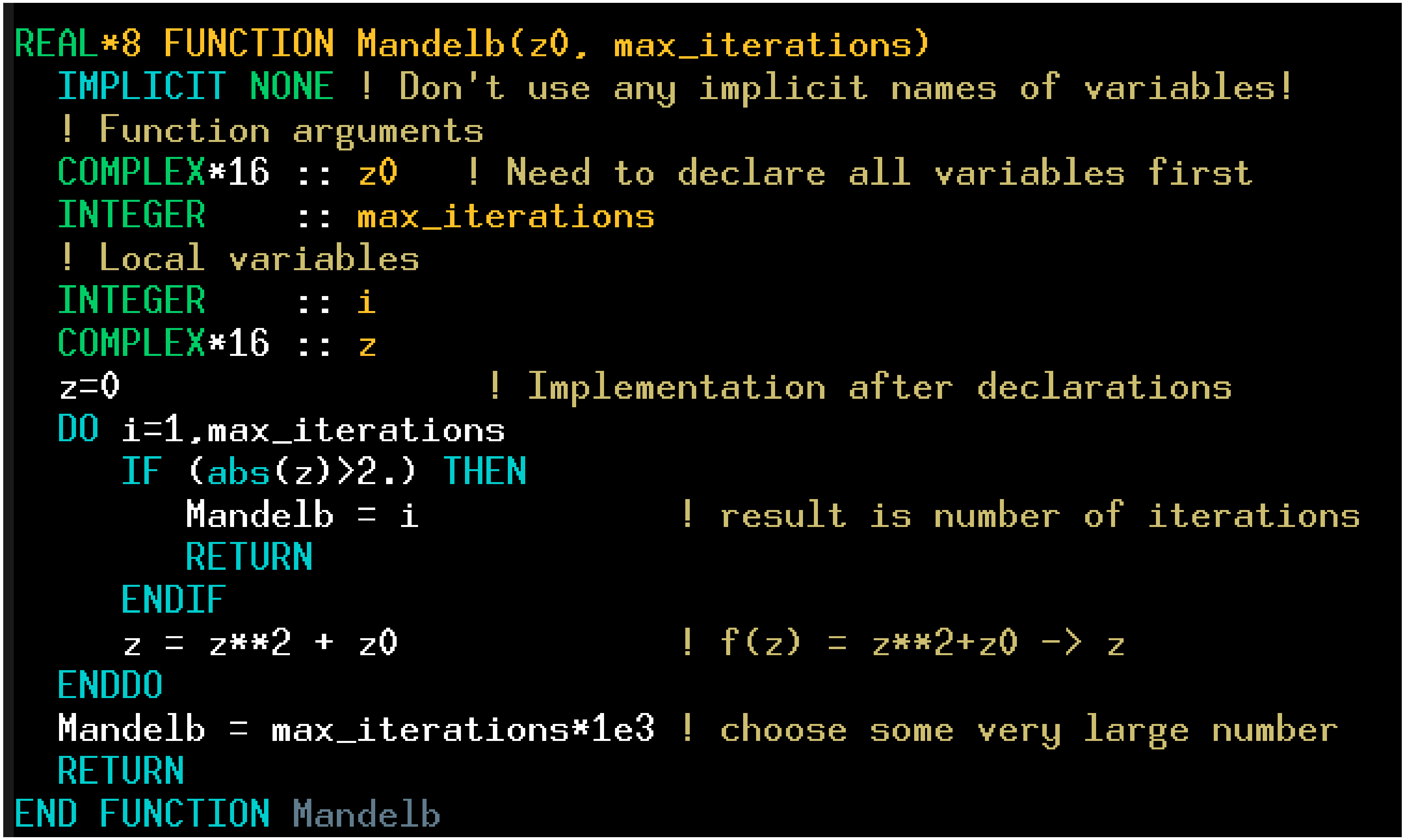


Comparison of languages by generating Mandelbrot set:

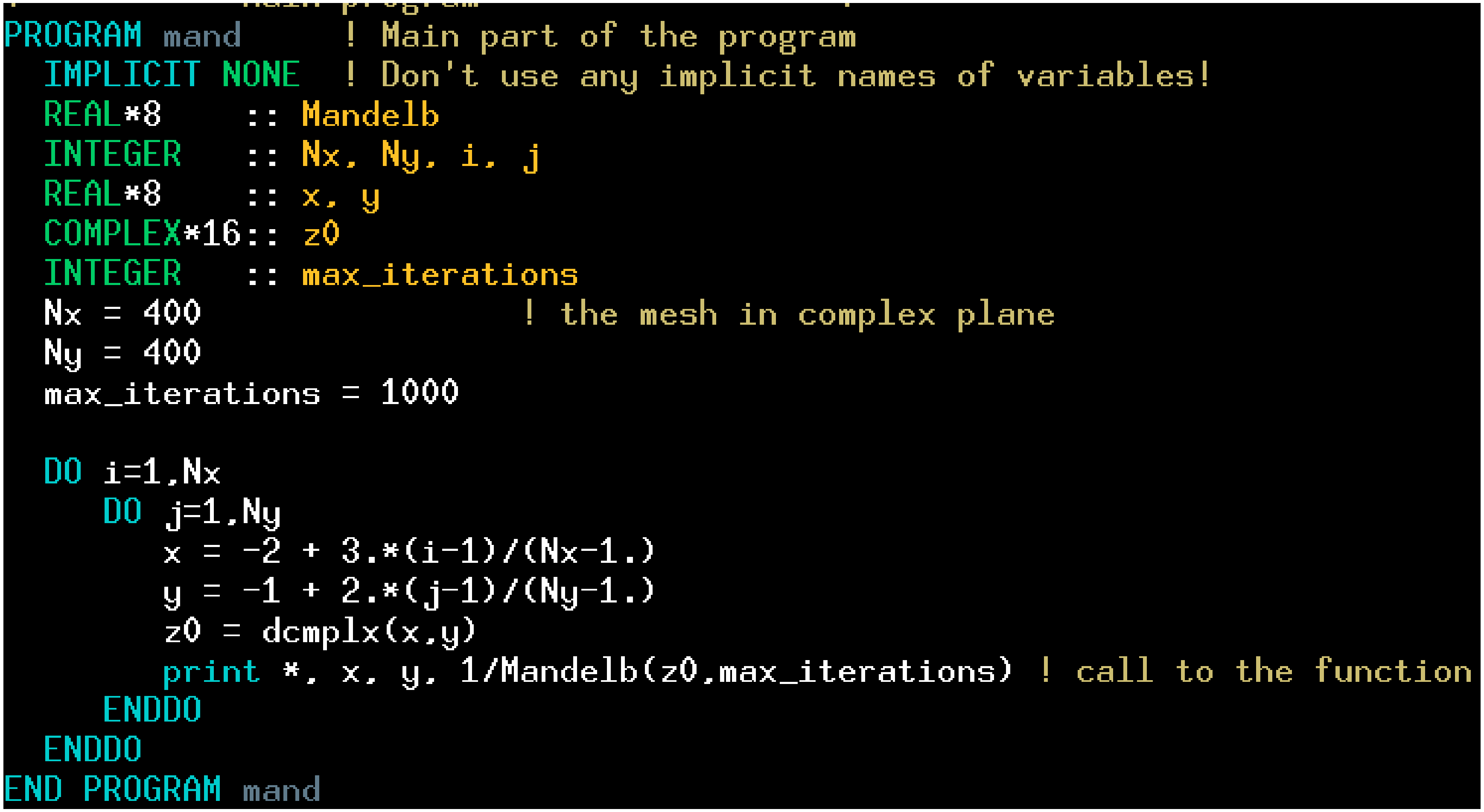


Wikipedia: The Mandelbrot set M is defined by a family of complex quadratic polynomials $f(z) = z^2 + z_0$ where z_0 is a complex parameter. For each z_0, one considers the behavior of the sequence $(0, f(0), f(f(0)), f(f(f(0))), \cdots)$ obtained by iterating $f(z)$ starting at $z = 0$, which either escapes to infinity or stays within a disk of some finite radius. The Mandelbrot set is defined as the set of all points z_0 such that the above sequence does not escape to infinity.

Implementation in Fortran:

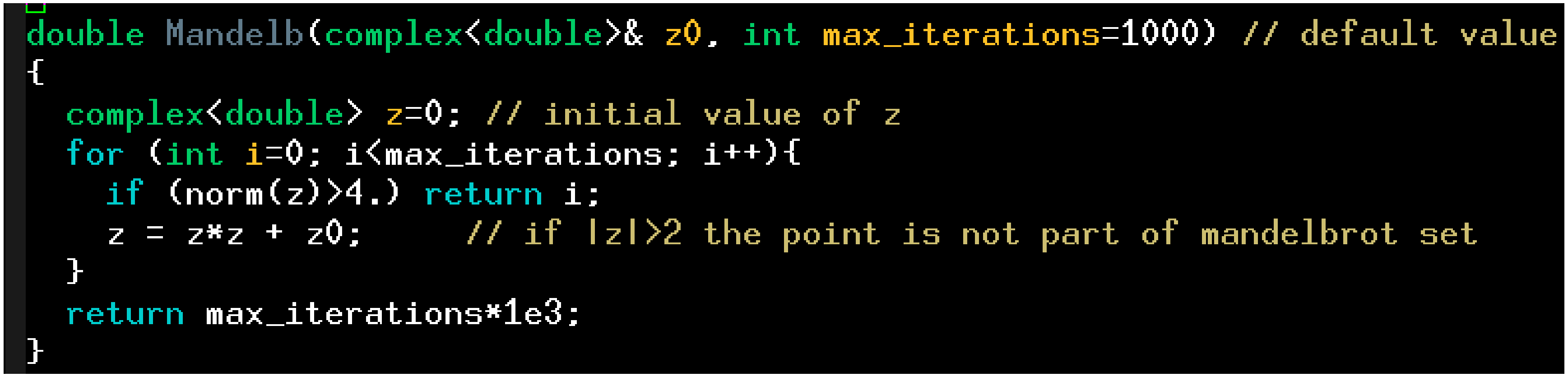


```fortran
REAL*8 FUNCTION Mandelb(z0, max_iterations)
  IMPLICIT NONE ! Don't use any implicit names of variables!
  ! Function arguments
  COMPLEX*16 :: z0   ! Need to declare all variables first
  INTEGER    :: max_iterations
  ! Local variables
  INTEGER    :: i
  COMPLEX*16 :: z
  z=0                 ! Implementation after declarations
  DO i=1,max_iterations
     IF (abs(z)>2.) THEN
        Mandelb = i            ! result is number of iterations
        RETURN
     ENDIF
     z = z**2 + z0             ! f(z) = z**2+z0 -> z
  ENDDO
  Mandelb = max_iterations*1e3 ! choose some very large number
  RETURN
END FUNCTION Mandelb
```

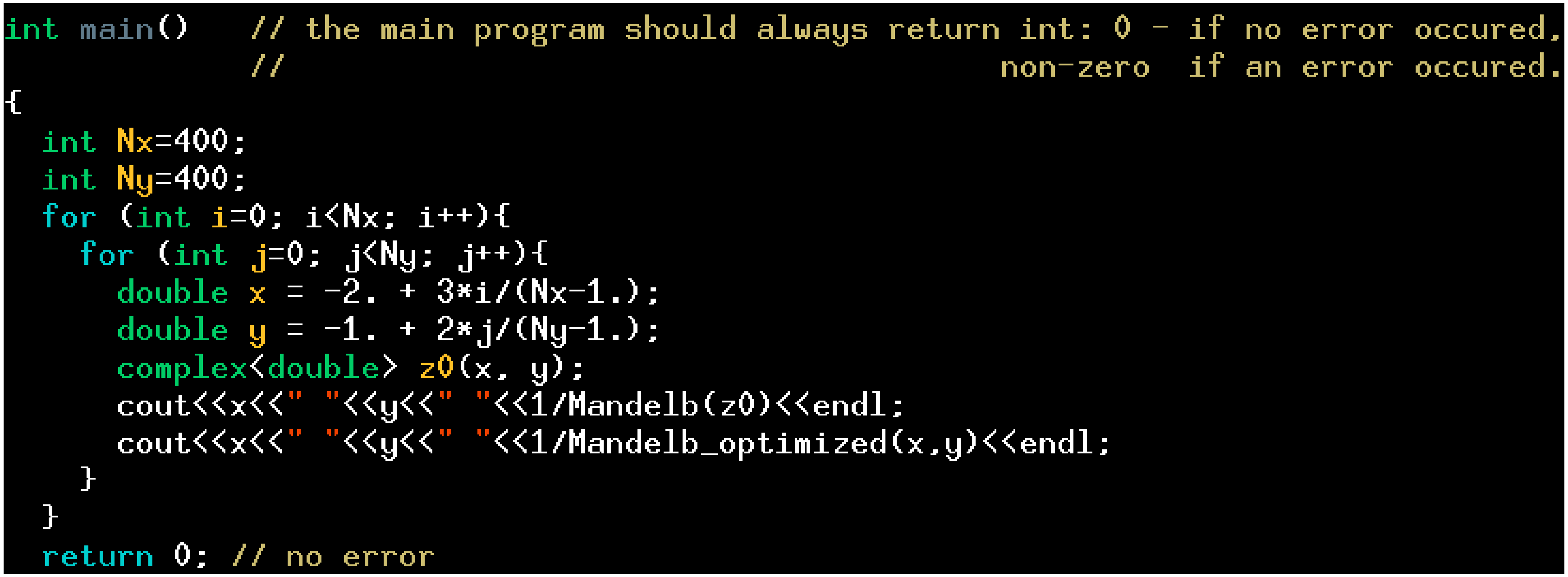


```
!           Main program                  !
PROGRAM mand     ! Main part of the program
  IMPLICIT NONE  ! Don't use any implicit names of variables!
  REAL*8    :: Mandelb
  INTEGER   :: Nx, Ny, i, j
  REAL*8    :: x, y
  COMPLEX*16:: z0
  INTEGER   :: max_iterations
  Nx = 400                ! the mesh in complex plane
  Ny = 400
  max_iterations = 1000

  DO i=1,Nx
     DO j=1,Ny
        x = -2 + 3.*(i-1)/(Nx-1.)
        y = -1 + 2.*(j-1)/(Ny-1.)
        z0 = dcmplx(x,y)
        print *, x, y, 1/Mandelb(z0,max_iterations) ! call to the function
     ENDDO
  ENDDO
END PROGRAM mand
```

Implementation in C++:

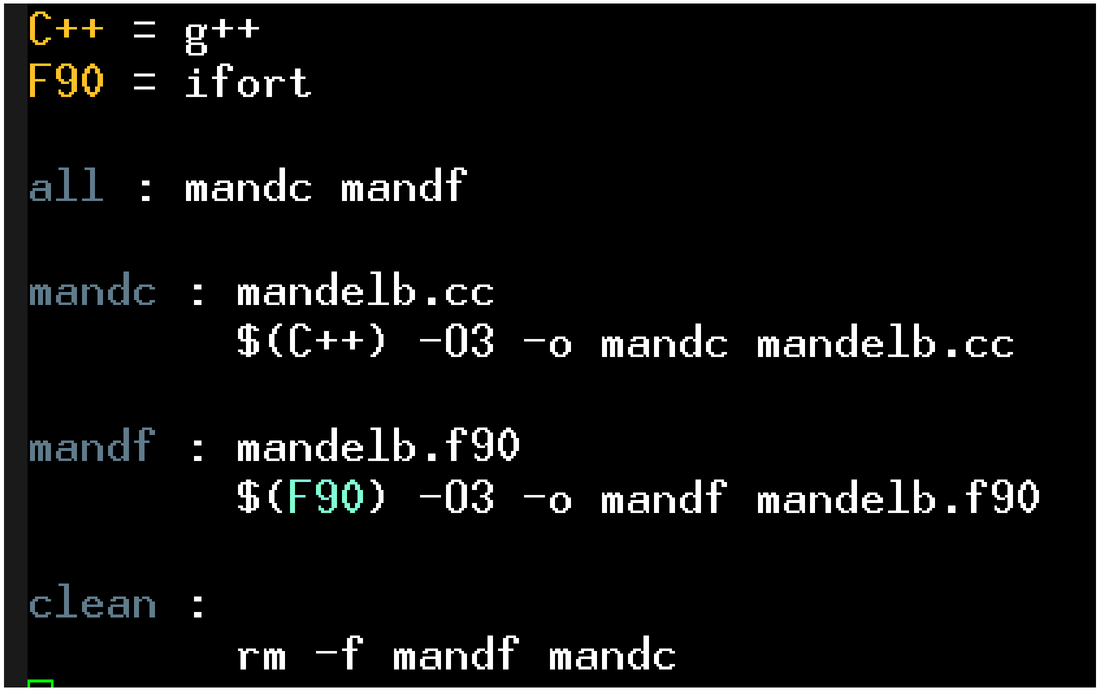


```
double Mandelb(complex<double>& z0, int max_iterations=1000) // default value
{
  complex<double> z=0; // initial value of z
  for (int i=0; i<max_iterations; i++){
    if (norm(z)>4.) return i;
    z = z*z + z0;      // if |z|>2 the point is not part of mandelbrot set
  }
  return max_iterations*1e3;
}
```



```
int main()    // the main program should always return int: 0 - if no error occured,
              //                                   non-zero  if an error occured.
{
  int Nx=400;
  int Ny=400;
  for (int i=0; i<Nx; i++){
    for (int j=0; j<Ny; j++){
      double x = -2. + 3*i/(Nx-1.);
      double y = -1. + 2*j/(Ny-1.);
      complex<double> z0(x, y);
      cout<<x<<" "<<y<<" "<<1/Mandelb(z0)<<endl;
      cout<<x<<" "<<y<<" "<<1/Mandelb_optimized(x,y)<<endl;
    }
  }
  return 0; // no error
```

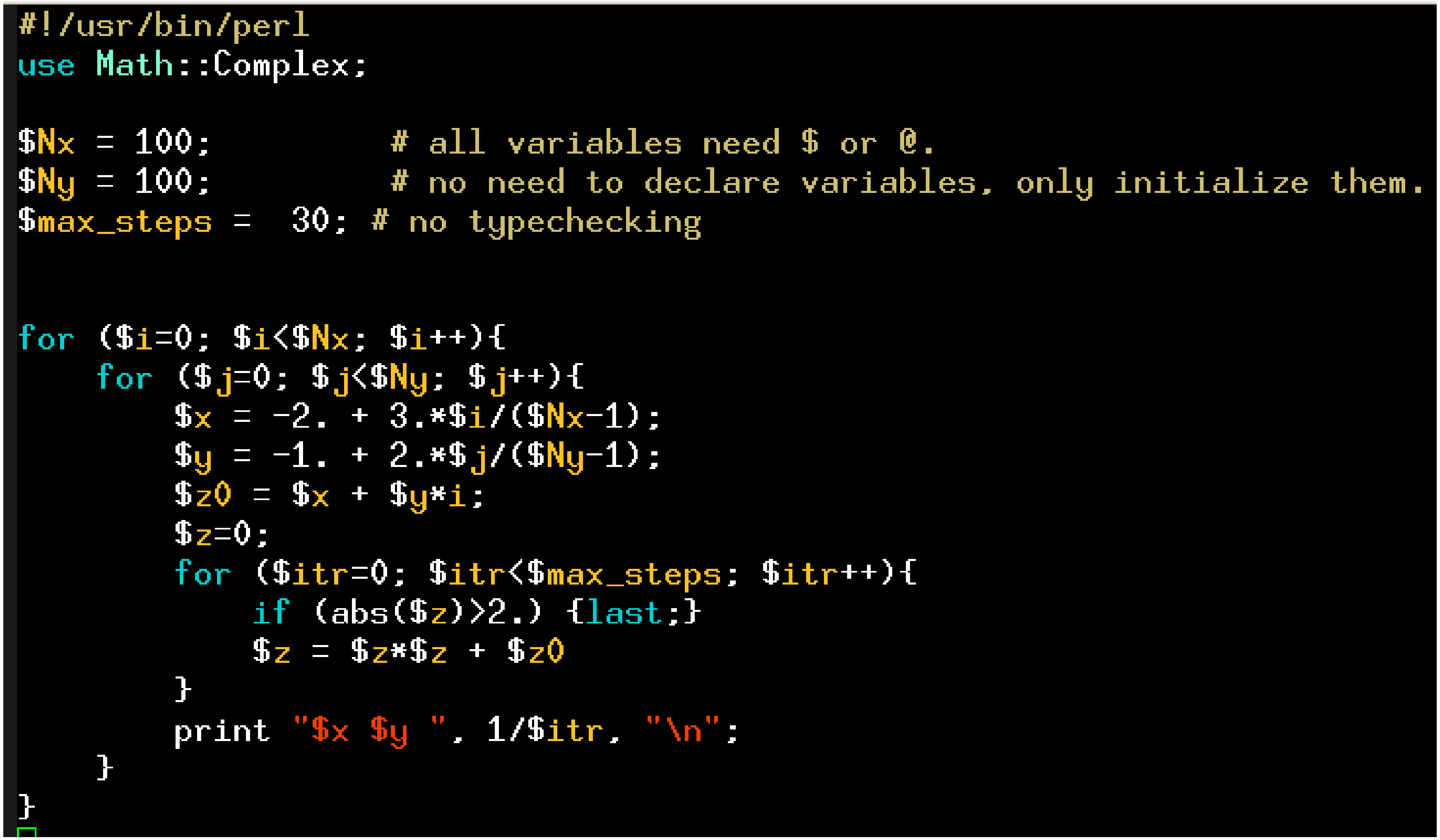
Both C++ and fortran code needs to be compiled (compilation allows optimization):



```
C++ = g++
F90 = ifort

all : mandc mandf

mandc : mandelb.cc
        $(C++) -O3 -o mandc mandelb.cc

mandf : mandelb.f90
        $(F90) -O3 -o mandf mandelb.f90

clean :
        rm -f mandf mandc
```

Perl code does not need to be compiled. It is interpreter.

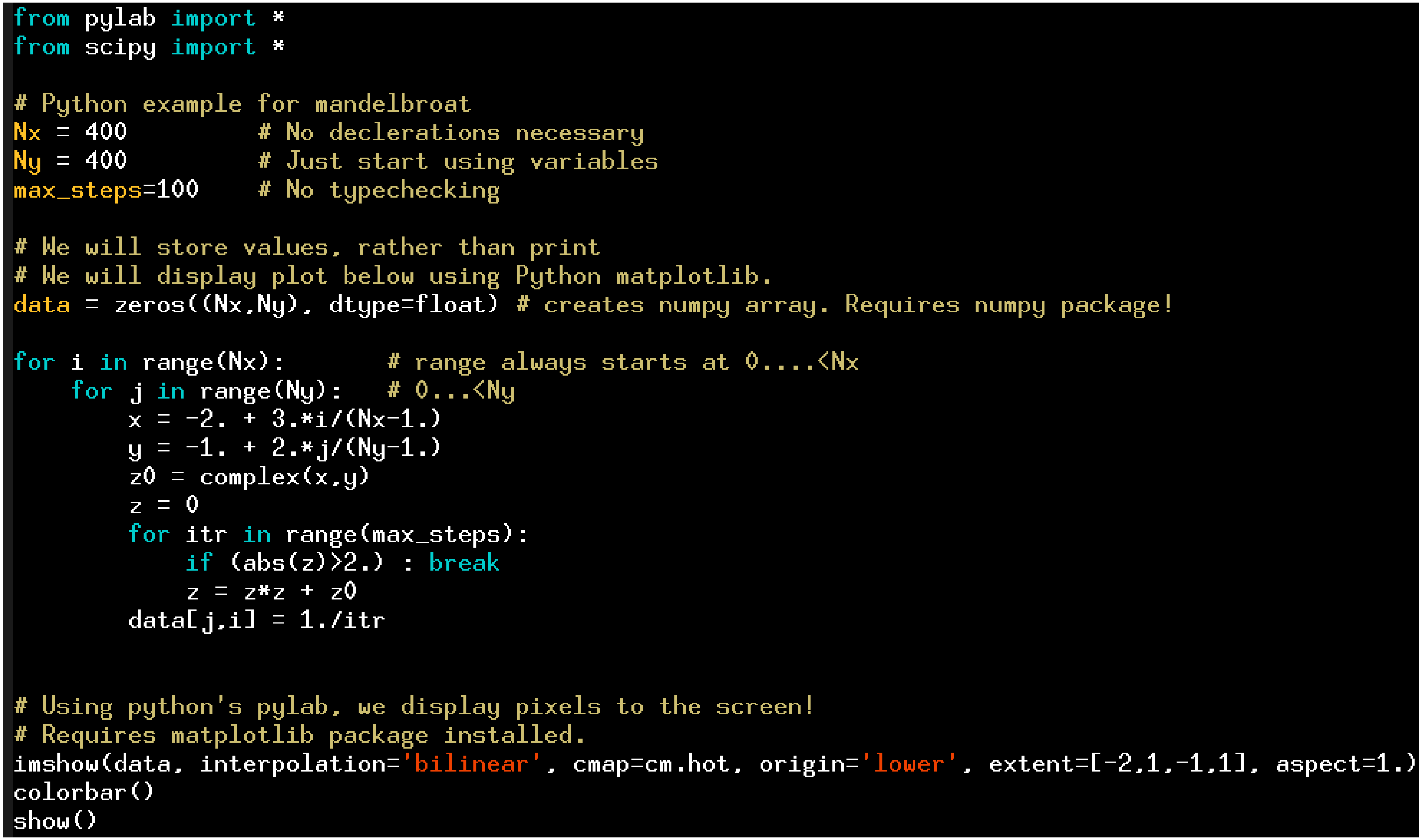
The call to subroutine is skipped due to optimization:



```
#!/usr/bin/perl
use Math::Complex;

$Nx = 100;         # all variables need $ or @.
$Ny = 100;         # no need to declare variables, only initialize them.
$max_steps =  30; # no typechecking


for ($i=0; $i<$Nx; $i++){
    for ($j=0; $j<$Ny; $j++){
        $x = -2. + 3.*$i/($Nx-1);
        $y = -1. + 2.*$j/($Ny-1);
        $z0 = $x + $y*i;
        $z=0;
        for ($itr=0; $itr<$max_steps; $itr++){
            if (abs($z)>2.) {last;}
            $z = $z*$z + $z0
        }
        print "$x $y ", 1/$itr, "\n";
    }
}
```

Python is interpreter as well.

The call to subroutine is again skipped for optimization:



```
from pylab import *
from scipy import *

# Python example for mandelbroat
Nx = 400         # No declerations necessary
Ny = 400         # Just start using variables
max_steps=100    # No typechecking

# We will store values, rather than print
# We will display plot below using Python matplotlib.
data = zeros((Nx,Ny), dtype=float) # creates numpy array. Requires numpy package!

for i in range(Nx):       # range always starts at 0....<Nx
    for j in range(Ny):   # 0...<Ny
        x = -2. + 3.*i/(Nx-1.)
        y = -1. + 2.*j/(Ny-1.)
        z0 = complex(x,y)
        z = 0
        for itr in range(max_steps):
            if (abs(z)>2.) : break
            z = z*z + z0
        data[j,i] = 1./itr


# Using python's pylab, we display pixels to the screen!
# Requires matplotlib package installed.
imshow(data, interpolation='bilinear', cmap=cm.hot, origin='lower', extent=[-2,1,-1,1], aspect=1.)
colorbar()
show()
```

Testing examples

Type: `make`

the following compilation is executed:

```
g++ -O3 -o mandc mandc.cc
ifort -O3 -o mandf mandf.f90
```

Execute and check the time:

```
time mandf > mand.dat                  | user    0m2.652s
time mandc > mand.dat                  | user    0m0.816s
time perl mandp.pl > mand.dat          | user    69m14.017s
time python mandp.py > mand.dat        | user    1m02.124s
```

- Both interpreters are substantially slower than comilers.
- Python is substantially faster than perl (surprise).
- C++ code was slightly optimized for performance and bits fortran90. In case of good optimization in both languages, they should be comparable.

The codes produce three column output $x, y, color$ and need a plotting program to display results. In `gnuplot` the following command plots the output:

```
set view map
splot 'mand.dat' with p ps 3 pt 5 palette
```

or call the script by

```
gnuplot gnu.sh
```

Plotting

Many plotting programs are available and everybody is free to chose his favorite. Some of most common programs and packages

- **gnuplot** `http://www.gnuplot.info/`
  - Freely available (part of gnu)
  - Works under mac, linux and cigwin
  - Plots 2D (and some support for 3D)
  - Quite powerful but somewhat clumsy to make publication quality figures
  - It is very fast and probably the best for ”preview”.
- **python-matplotlib**
  - Free, part of Python library
  - requires some coding but can produce publication quality figures
  - many examples available at `http://matplotlib.org/gallery.html`

  (Author John Hunter recently died `http://numfocus.org/johnhunter/`)

Homework:

- Set up your environment! C++, Python, fortran, BLAS&LAPACK (or vecLib).
- If you are familiar with coding, write your own mandelbrot version of the code.
  If not, download Mandelbrot code written in fortran, C++, perl and python. Execute them and check that they work properly.
- Test your gnuplot by plotting mandelbrot set from generated file `mand.dat`.