From the Bootcamp: Programming languages

- compiled languages: source code needs to be explicitly translated into machine code
  - generally requires two steps: compilation and linking
    - Lowest level: machine language
    - basic languages: C, Fortran
    - more advanced languages: C++, many others

- “interactive" languages
    - provide line-by-line or block-by-block execution as well as ability to run full existing programs
    - can be (but not always) less computationally efficient than compiled programs
    - generally provide more extensive libraries, e.g., for plotting
    - excellent for data exploration
    - Examples used in astronomy:
        - Python, with matplotlib for plotting
        - julia
        - IDL, with Astronomy Users Library
        - MATLAB
        - SuperMongo

- In astronomy, *Python is undoubtedly the most popular*; freely available widely used outside of astronomy/academia.
  - However, many (most?) computing-intensive applications are written in a fully compiled language, like C++.

- Languages evolve and change! Understand the principles and be open-minded!

From the Bootcamp: Programming principles

- Programming may be one of the more important skills you develop in the graduate program!
  - Think about coming out of NMSU with a programming portfolio!
- Assume that everything you write has an error in it (it usually does!): consider how you can test things!
- less is more (to a point)
  - don't repeat work, make the tool better and reusable
  - If your code has blocks that are very similar to each other, you can probably make it more efficient
  - Use functions/subroutines
- Think before you program: outline what needs to be included and consider how it can be made more modular
- Don't be too reluctant to rewrite/refactor code! Be self-critical.
- all software should be written as if someone else (e.g., future you) will use it: document
- With "interactive" languages, experiment "interactively", but produce all final output using saved programs, so it is repeatable
- Show your code to others! Make it available, e.g., through github
- Most astronomers think they are good programmers; most programmers would probably disagree, *realize what you don't know*, and *learn it as appropriate*
  - Read documentation
  - What astronomers *can* bring to the table: knowledge of analysis techniques, coming up with creative approaches/solutions to problems

From the Bootcamp: When to use what machines?

- Resource considerations
  - For computations lasting more than a few hours, consider using a computing server, and in parallel!
  - If your python code is taking long, convert to a compiled language.
    - Compiled languages (C, Fortran) are much faster than interpreted ones (Python)

Data types

What are the basic units/sizes of data?

How are different types of data (characters, integers, floating point, etc.) represented?

Data types

- bits, bytes, etc
  - bit (0/1)
  - byte: 8 bits
  - data types (bytes): character (ASCII) (1), short integer (2), integer (4/8), long integer (8), float (4), double (8)
  - kilobyte = 2^10 (=1024 bytes); megabyte = 2^20 bytes; gigabyte = 2^30 bytes; terabyte = 2^40 bytes; etc.
  - important to keep memory usage in mind when programming!
- How is hardware characterized in terms of data storage and latency?
  - hard disk space: plentiful (TB), but slow access. Analogous to a bookshelf. Non-volatile.
  - memory: less plentiful (MB-GB), faster access. Analogous to your long-term memory. Volatile
  - cache (L1, L2, L3): limited (KB-MB), ultra-fast access. Analogous to your short-term memory. Volatile

Practice

- An image cube is 2048x2048x47 reads: how much memory?

- A hydrodynamical simulation uses resolution 1024x1024x1024, with 5 field for each grid cell (density, 3-velocity, and energy).
  - How much memory is required if all numbers are
    - floats (4 bytes)?
    - doubles (8 bytes)?

Data types in Python

Warning: Python variables may not behave as you expect from other languages. In fact, they are perhaps better thought of as labels. Consider the following examples:

```
a=1

b=a

a=2

print(a,b)
```

```
a=[1,2,3]

b=a

a[0]=0

print(a,b)
```

- In python some objects are immutable, while others are mutable. Mutable objects can be changed, immutable ones can't. Bool, int, float, str, tuple are immutable, list, dict, set, numpy arrays are mutable.
- ``Variables'' are references to objects.
- be aware of copy module to make copies of objects

```
import copy

b=copy.copy(a)
```

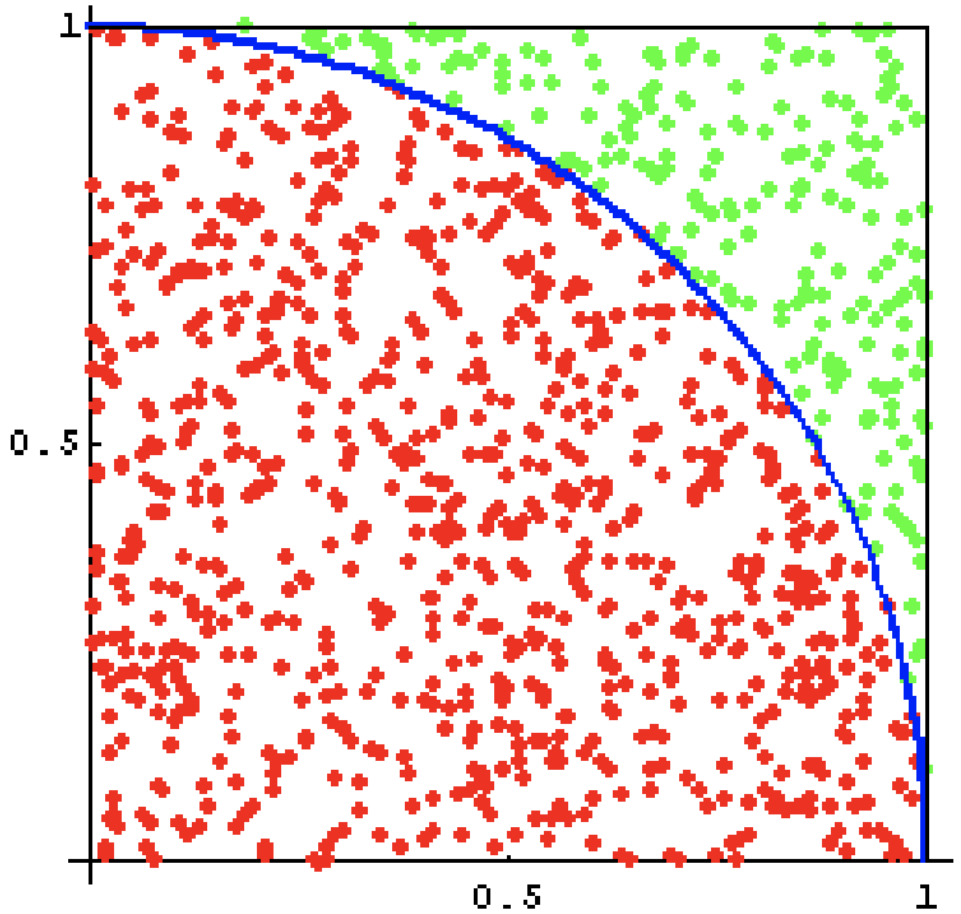
Code in this class will be graded according this rubric:

Computer Programming Grading Rubric

Back to Grading

Trait	Exceptional	Acceptable	Amateur	Unsatisfactory
Specifications	The program works and meets all of the specifications.	The program works and produces the correct results and displays them correctly. It also meets most of the other specifications.	The program produces correct results but does not display them correctly.	The program is producing incorrect results.
Readability	The code is exceptionally well organized and very easy to follow.	The code is fairly easy to read.	The code is readable only by someone who knows what it is supposed to be doing.	The code is poorly organized and very difficult to read.
Reusability	The code could be reused as a whole or each routine could be reused.	Most of the code could be reused in other programs.	Some parts of the code could be reused in other programs.	The code is not organized for reusability.
Documentation	The documentation is well written and clearly explains what the code is accomplishing and how.	The documentation consists of embedded comment and some simple header documentation that is somewhat useful in understanding the code.	The documentation is simply comments embedded in the code with some simple header comments separating routines.	The documentation is simply comments embedded in the code and does not help the reader understand the code.
Delivery	The program was delivered on time.	The program was delivered within a week of the due date.	The code was within 2 weeks of the due date.	The code was more than 2 weeks overdue.
Efficiency	The code is extremely efficient without sacrificing readability and understanding.	The code is fairly efficient without sacrificing readability and understanding.	The code is brute force and unnecessarily long.	The code is huge and appears to be patched together.

Calculating π via Statistical Sampling (aka Monte Carlo)



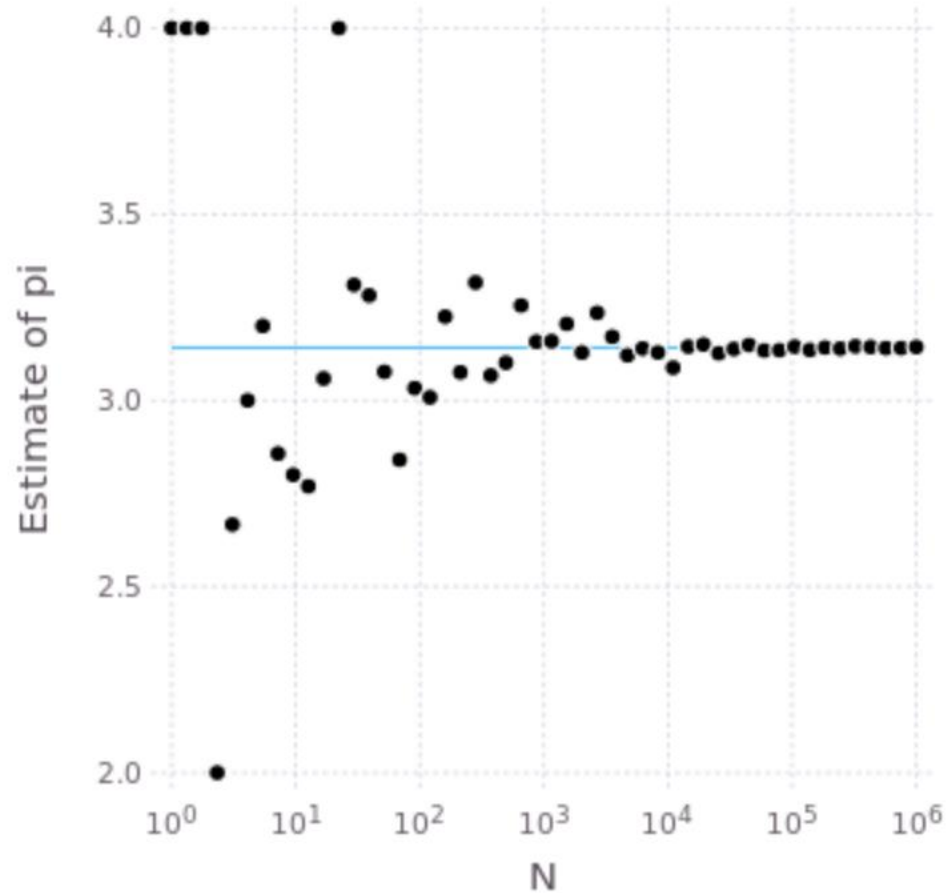
Area of circle section = π /4
Area of square = 1

Pseudo-code:

Define total number of tries
Loop through total number of tries
Define two random numbers
If number falls inside circle: hit

π = 4*hits/total

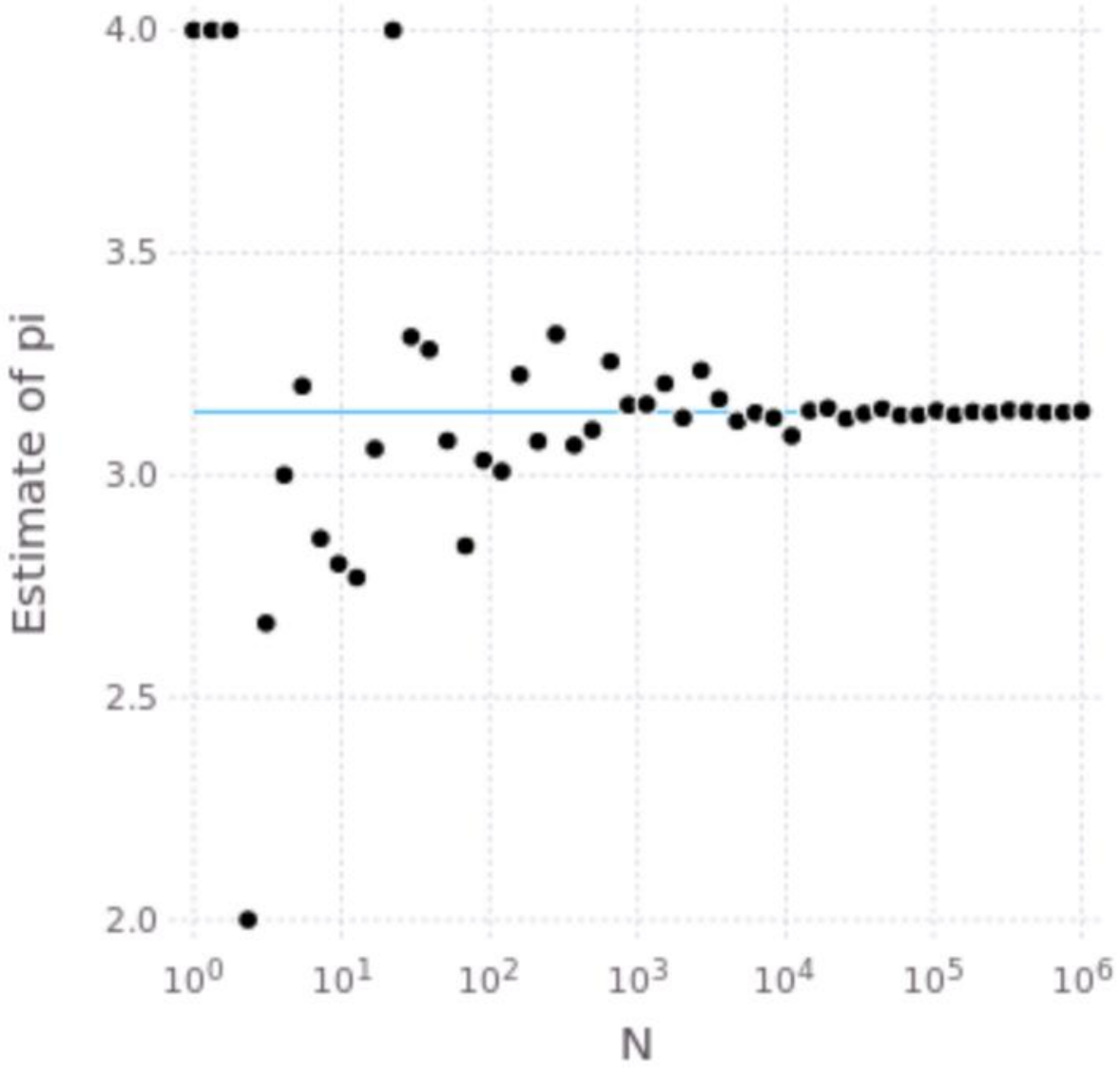
(No need to plot the graph, simply print the approximate value of π that you get.)


4.0
3.5
3.0
2.5
2.0
Estimate of pi
10^0
10^1
10^2
10^3
10^4
10^5
10^6
N

Slow convergence

Because the method is a shooting method, its noise is shot noise (Poisson noise), scaling with sqrt(N).

- $N=10^2$ tries gives 0.1 accuracy.
- For $O(10^{-2})$ accuracy, needs $N=10^4$
- Calculating π to 3 digits $O(10^{-3})$needs 10^6 tries.



Difference between Compiled and Interpreted Language

S.NO.	COMPILED LANGUAGE FAST	INTERPRETED LANGUAGE HIGH-LEVEL
1	A compiled language is a programming language whose implementations are typically compilers and not interpreters.	An interpreted language is a programming language whose implementations execute instructions directly and freely, without previously compiling a program into machine-language instructions.
2	In this language, once the program is compiled it is expressed in the instructions of the target machine.	While in this language, the instructions are not directly executed by the target machine.
3	There are at least two steps to get from source code to execution.	There is only one step to get from source code to execution.
4	In this language, compiled programs run faster than interpreted programs.	While in this language, interpreted programs can be modified while the program is running.
5	In this language, compilation errors prevent the code from compiling.	In this languages, all the debugging occurs at run-time.
6	The code of compiled language can be executed directly by the computer's CPU.	A program written in an interpreted language is not compiled, it is interpreted.
7	This language delivers better performance.	This language example delivers relatively slower performance.
8	Example of compiled language – C, C++, C#, CLEO, COBOL, etc.	Example of Interpreted language – JavaScript, Perl, Python, BASIC, etc.

Fortran

- **Fortran** (*Formula Translator*) is the **gold standard** for **high-performance computing**.
- The first (and some say the best) programming language for scientific and engineering applications.
- Gets a bad rep for being old, which is ***unwarranted***, and mostly due to continued use of older versions like Fortran 77
- Evolution of Fortran
  - Classic FORTRAN (uppercase spelling)
    - FORTRAN (1952-1957) – High level compared to assembly language. Control flow (if statements) and loops.
    - FORTRAN II (1958) – Subroutines and functions.
    - FORTRAN IV (also called Fortran 66) – logicals and booleans
    - FORTRAN 77 – else, improved do loops and I/O, includes, character data, save, intrinsic functions
  - Modern Fortran (lowercase spelling)
    - Fortran 90 – vectorization, inline comments, recursivity, modularity, dynamic memory allocation, pointers
    - Fortran 95 – improved high-performance (minor upgrade on Fortran 90)
    - Fortran 2003 – object-oriented programming
    - Fortran 2008 – concurrent programming (minor upgrade on Fortran 2003)
    - Fortran 2018 – native parallel capabilities, improved connectivity with C
- All versions are backward compatible:
  - Fortran I can run on a Fortran 2018 compiler.
    - Very important for projects that run for decades, like space missions.

“Hello World!” in Fortran 2018

```
program hello
  ! This is a comment line; it is ignored by the compiler
  print *, 'Hello, World!'
end program hello
```

```
$> gfortran hello.f90 -o hello
```

```
$> ./hello
Hello, World!
```

Unlike python, that relies on indentation for control flow, most languages use explicit begin and end statements. Here, **program** is finished with an **endprogram.**

Once the human-readable code is done, you use a compiler to convert it to machine code. Here, *gfortran* is used. The extension of the file is .f90, which stands for Fortran90, the first modern Fortran. Other versions of fortran (Fortran 95, 2003, 2008, and 2018 also use the .f90 extension).

The –o means output. The human-readable code hello.f90 will be compiled into a binary executable called “hello”

Once compiled, execute it like any binary.

“Hello World!” in C++

```cpp
// Your First C++ Program

#include <iostream>

int main() {
    std::cout << "Hello World!";
    return 0;
}
```

```
$> gfortran hello.f90 -o hello
```

```
gcc hello.cpp –o hello
```

```
$> ./hello
Hello, World!
```

Other languages vary mostly in syntax. In C, curly brackets give the beginning and end of a function.

The compilation is like in Fortran. The compiler for C++ is gcc.

“Hello World!” in Julia

Julia

```
print("Hello World")
```

```
julia helloworld.jl.
```

```
Hello World
```

High-performance and dynamically-typed, Julia combines the best of both worlds. The speed of a compiled language with the high-level approach of an interpreted language.

- *“But why isn’t it as popular as Python?”*

Beats me.