Algorithms

COMP20290

Dr. Mark Matthews

Algorithms increasingly run the world

Plays a key role in modern technological innovation

Internet: Web search, packet routing, distributed file sharing.

Biology: Human genome project, protein folding.

Computers: Circuit layout, file system, compilers.

Computer graphics: Movies, video games, virtual reality.

Security: Cell phones, e-commerce, voting machines.

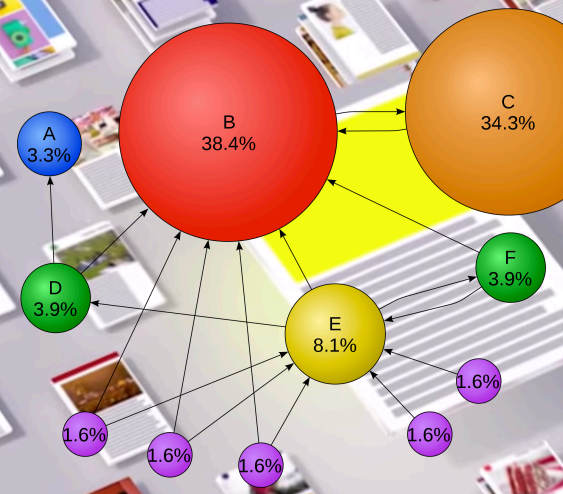
Multimedia: CD player, DVD, MP3, JPG, DivX, HDTV.

Transportation: Airline crew scheduling, map routing.

Physics: N-body simulation, particle collision simulation.

PageRank algorithm developed by Larry Page and Sergey Brin at Stanford in 1996

- Several iterative strategies are possible (linear algebra)
- Google Inc. is now a $131 bn company with 58,000 employees



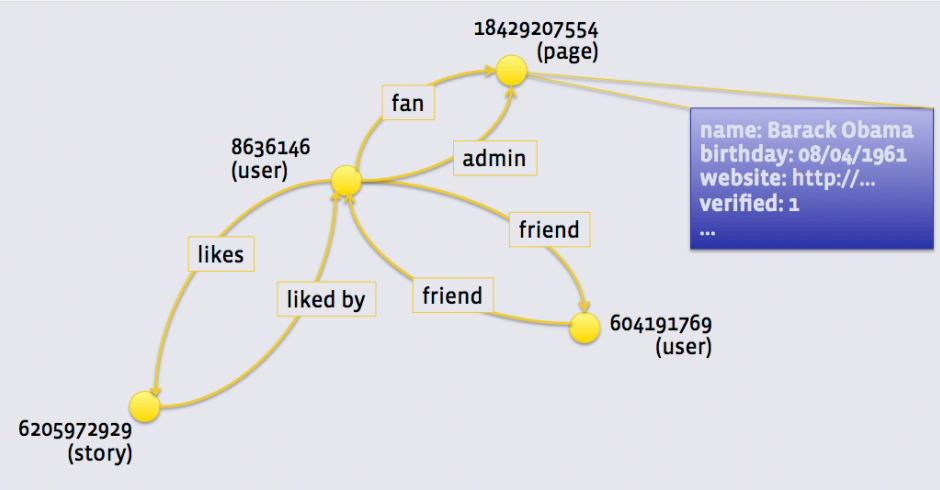
the PageRank value for a page u is dependent on the PageRank values for each page v contained in the set Bu (the set containing all pages linking to page u), divided by the number L(v) of links from page v.

So you have a toolshed of different techniques to consider when solving problems in your code.

Facebook does graph processing on a massive scale

2.7billion likes added every day (8Tb/day)

Unicorn is an online, in-memory social graph
2.5billion pieces of new content every day



The Facebook graph is the collection of entities and their relationships on Facebook. The entities are the nodes and the relationships are the edges. One way to think of this is if the graph were represented by language, the nodes would be the nouns and the edges would be the verbs. Every user, page, place, photo, post, etc. are nodes in this graph. Edges between nodes represent friendships, check-ins, tags, relationships, ownership, attributes, etc.

Algorithms are changing people's behaviour every day

My Video Went Viral. Here's Why

Veritasium 2.6M views • 8 months ago

My hypothesis is that the algorithm, rather than viewer preference, drives views on the site. As the algorithm shifts, various ...

Subtitles

YouTubers spend a lot of time testing and discussing what makes YouTube's content algorithm pick up and promote their video.

Algorithms are taking on a life of their own

Mysterious algorithm was 4% of trading activity last week

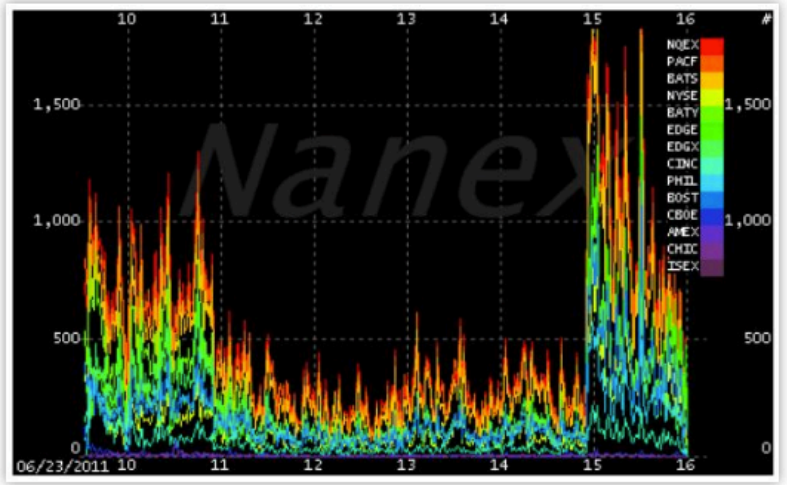
October 11, 2012

A single mysterious computer program that placed orders — and then subsequently canceled them — made up 4 percent of all quote traffic in the U.S. stock market last week, according to the top tracker of high-frequency trading activity.

The motive of the algorithm is still unclear, *CNBC* reports.

The program placed orders in 25-millisecond bursts involving about 500 stocks, according to Nanex, a market data firm. The algorithm never executed a single trade, and it abruptly ended at about 10:30 a.m. ET Friday.

[+]



Generic high frequency rrading chart (credit: Nanex)

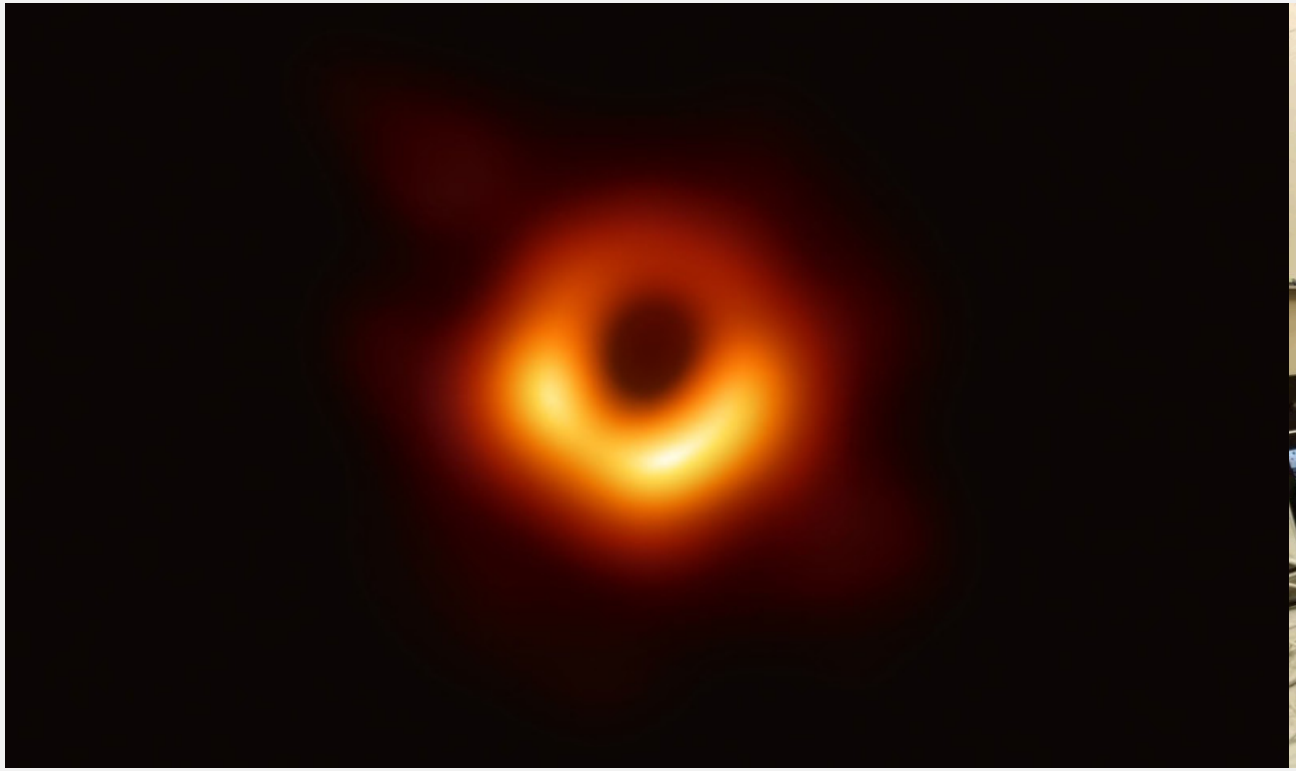
"My guess is that the algo was testing the market, as high-frequency frequently does," says Jon Najarian, co-founder of TradeMonster.com. "As soon as they add bandwidth, the HFT crowd sees how quickly they can top out to create latency." (*Read More*: Unclear What Caused Kraft Spike: Nanex Founder.)

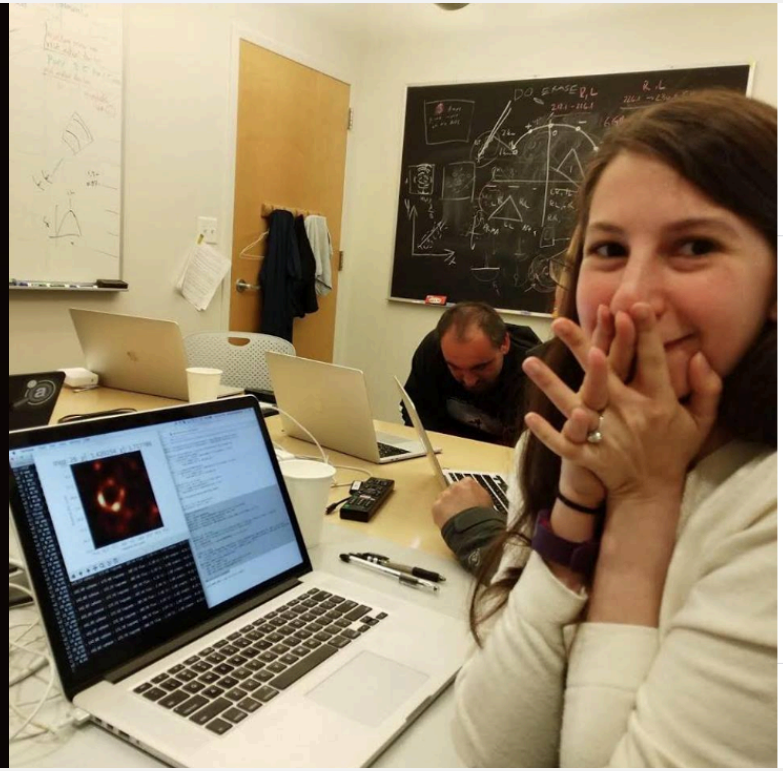
WATCH https://www.cnbc.com/video/3000116957

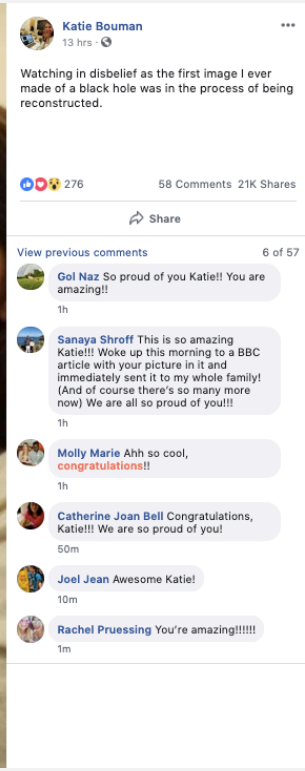
LISTEN to excellent BBC podcast on high frequency trading; https://www.bbc.co.uk/programmes/b0bjzntv

The motive of the algorithm is still unclear.

To solve problems that could not otherwise be solved



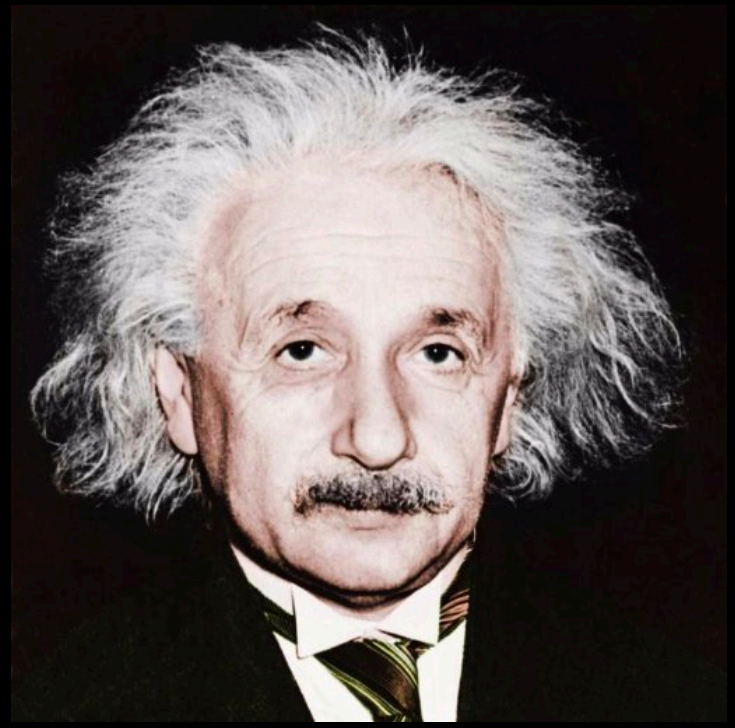




https://techcrunch.com/2019/04/10/the-creation-of-the-algorithm-that-made-the-first-black-hole-image-possible-was-led-by-mit-grad-student-katie-bouman/

To practice elegance

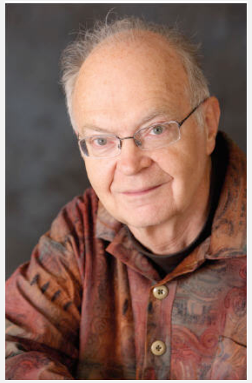
“If I had an hour to solve a problem I'd spend 55 minutes thinking about the problem and 5 minutes thinking about solutions.”



“A colleague recently claimed that he’d done only 15 minutes of productive work in his whole life. He wasn’t joking, because he was referring to the 15 minutes during which he’d sketched out a fundamental optimization algorithm.”

For intellectual stimulation

“ An algorithm must be seen to be believed. ” — Donald Knuth



“ For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing. ” — Francis Sullivan

FROM THE EDITORS

THE JOY OF ALGORITHMS

Francis Sullivan, Associate Editor-in-Chief

THE THEME OF THIS FIRST-OF-THE-CENTURY ISSUE OF *COMPUTING IN SCIENCE & ENGINEERING* IS ALGORITHMS. IN FACT, WE WERE BOLD ENOUGH—AND PERHAPS FOOLISH ENOUGH—TO CALL THE 10 EXAMPLES WE'VE SELECTED “THE TOP 10 ALGORITHMS OF THE CENTURY.”

Computational algorithms are probably as old as civilization. Sumerian cuneiform, one of the most ancient written records, consists partly of algorithm descriptions for reckoning in base 60. And I suppose we could claim that the Druid algorithm for estimating the start of summer is embodied in Stonehenge. (That's really hard hardware!)

Like so many other things that technology affects, algorithms have advanced in startling and unexpected ways in the 20th century—at least it looks that way to us now. The algorithms we chose for this issue have been essential for progress in communications, health care, manufacturing, economics, weather prediction, defense, and fundamental science. Conversely, progress in these areas has stimulated the search for ever-better algorithms. I recall one late-night bull session on the Maryland Shore when someone asked, “Who first ate a crab? After all, they don't look very appetizing.” After the usual speculations about the observed behavior of sea gulls, someone gave what must be the right answer—namely, “A very hungry person first ate a crab.”

The flip side to “necessity is the mother of invention” is “invention creates its own necessity.” Our need for powerful machines always exceeds their availability. Each significant computation brings insights that suggest the next, usually much larger, computation to be done. New algorithms are an attempt to bridge the gap between the demand for cycles and the available supply of them. We've become accustomed to gaining the Moore's Law factor of two every 18 months. In effect, Moore's Law changes the constant in front of the estimate of running time as a function of problem size. Important new algorithms do not come along every 1.5 years, but when they do, they can change the exponent of the complexity!

For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing. A colleague recently claimed that he'd done only 15 minutes of productive work in his whole life. He wasn't joking, because he was referring to the 15 minutes during which he'd sketched out a fundamental optimization algorithm. He regarded the previous years of thought and investigation as a sunk cost that might or might not have paid off.

Researchers have cracked many hard problems since 1 January 1900, but we are passing some even harder ones on to the next century. In spite of a lot of good work, the question of how to extract information from extremely large masses of data is still almost untouched. There are still very big challenges coming from more “traditional” tasks, too. For example, we need efficient methods to tell when the result of a large floating-point calculation is likely to be correct. Think of the way that check sums function. The added computational cost is very small, but the added confidence in the answer is large. Is there an analog for things such as huge, multidisciplinary optimizations? At an even deeper level is the issue of reasonable methods for solving specific cases of “impossible” problems. Instances of NP-complete problems crop up in attempting to answer many practical questions. Are there efficient ways to attack them?

I suspect that in the 21st century, things will be ripe for another revolution in our understanding of the foundations of computational theory. Questions already arising from quantum computing and problems associated with the generation of random numbers seem to require that we somehow tie together theories of computing, logic, and the nature of the physical world.

The new century is not going to be very restful for us, but it is not going to be dull either!

2 COMPUTING IN SCIENCE & ENGINEERING

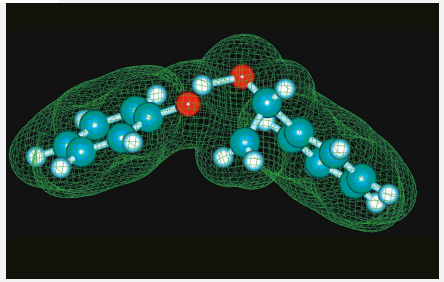
http://www.cs.fsu.edu/~lacher/courses/COT4401/notes/cise_v2_i1/editors.pdf

To solve problems that could not otherwise be solved

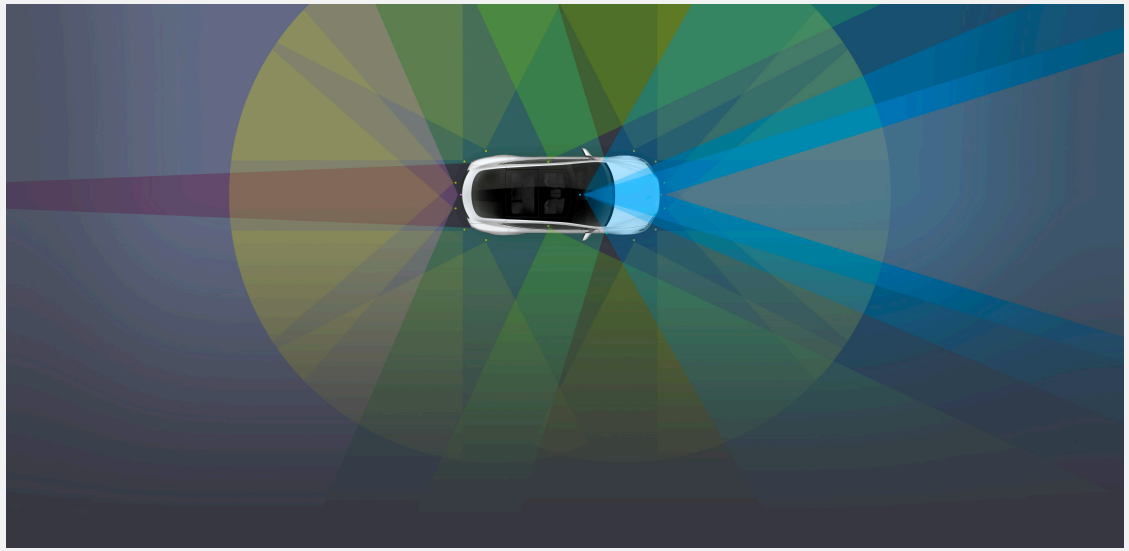
“ Computer models mirroring real life have become crucial for most advances made in chemistry today…. Today the computer is just as important a tool for chemists as the test tube. ”

— *Royal Swedish Academy of Sciences*
(Nobel Prize in Chemistry 2013)

Martin Karplus, Michael Levitt, and Arieh Warshel



Algorithms will be essential to solving future problems



Why Study Algorithms?

For profit!

Why Study Algorithms?

To get a job

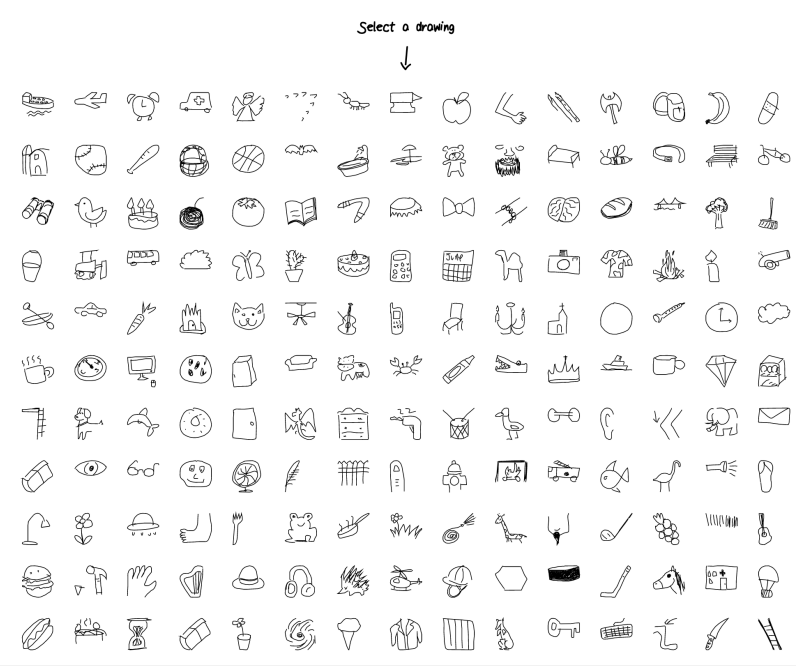
Why Study Algorithms?

For fun

https://quickdraw.withgoogle.com/

"Over 15 million players have contributed millions of drawings playing Quick, Draw! These doodles are a unique data set that can help developers train new neural networks, help researchers see patterns in how people around the world draw, and help artists create things we haven't begun to think of. That's why we're open-sourcing them, for anyone to play with."

https://quickdraw.withgoogle.com/data



Algorithms

COMP20290

Dr. Mark Matthews

What we will cover?

Algorithm Overview & History
Algorithm Analysis
Elementary Sort Algorithms
Advanced Sort Algorithms
Search Algorithms
Text Processing Algorithms
- Text Compression
- Tries (not an algorithm per se)
- String Searching

What you'll learn?

- Become a better programmer
- Understand how to evaluate algorithms
- How to develop algorithms
- Understand the key algorithmic strategies for solving problems
- Do well in technical interviews
- Understand the impact of computer science
- Getting more coding experience.

COMP20280 Data Structures

Dr. Aonghus Lawlor - COMP 20280
Runs alongside this course.

Our focus in this course is at the algorithmic level but data structures are very important to algorithms.

In our examples you may need to use some data structures but the focus is still on the algorithm

Lectures & Practicals

Traditional lectures. Introduce new material.

When	Where	Who
Tue 3pm	**B004**	**Mark**
Thurs 4pm	**B004**	**Mark**

Practicals. 11 practicals 1 hour long, split class into 2. No practicals this week.

When	Where	Who
????	**????**	**Luca**

Course Notes

Everything will be on moodle:

http://csmoodle.ucd.ie

Course Name:

COMP20029 Algorithms – 2019-2020

Key:

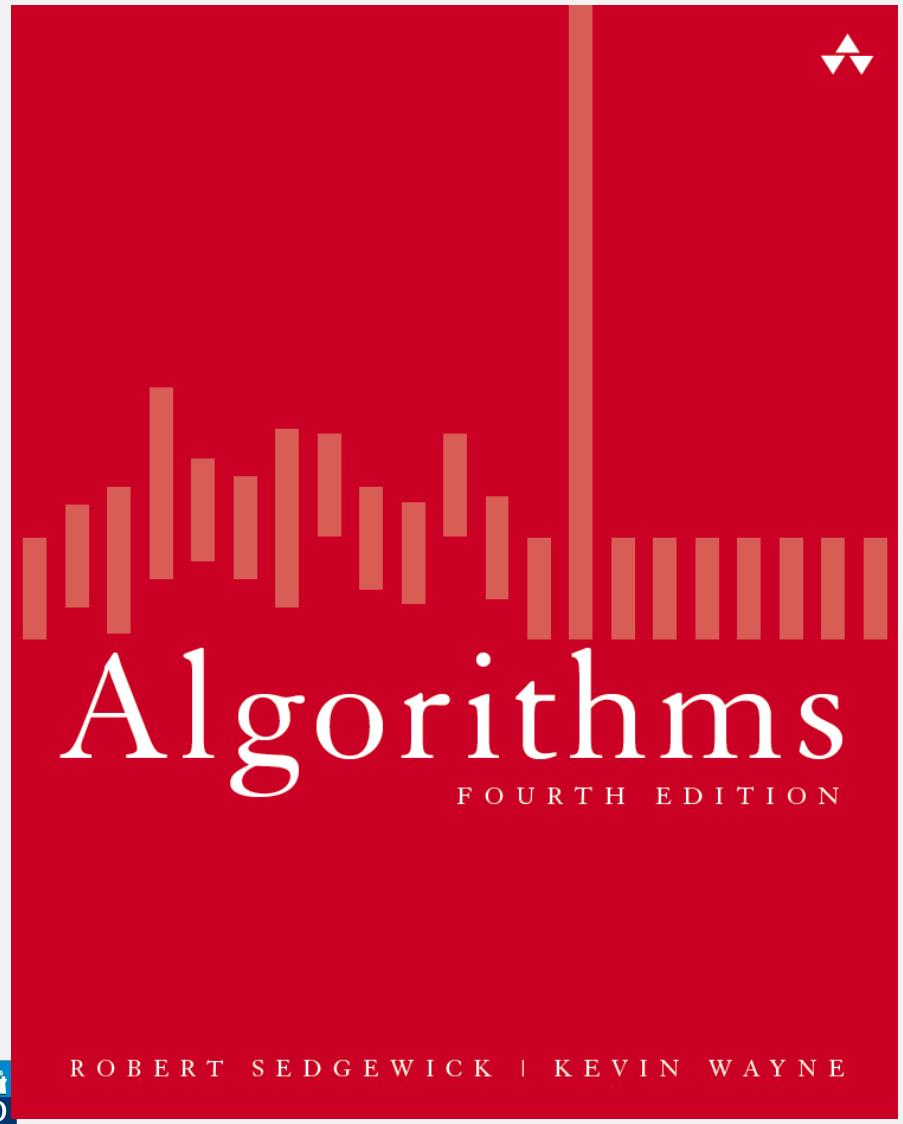
alg2020

Site is live now!

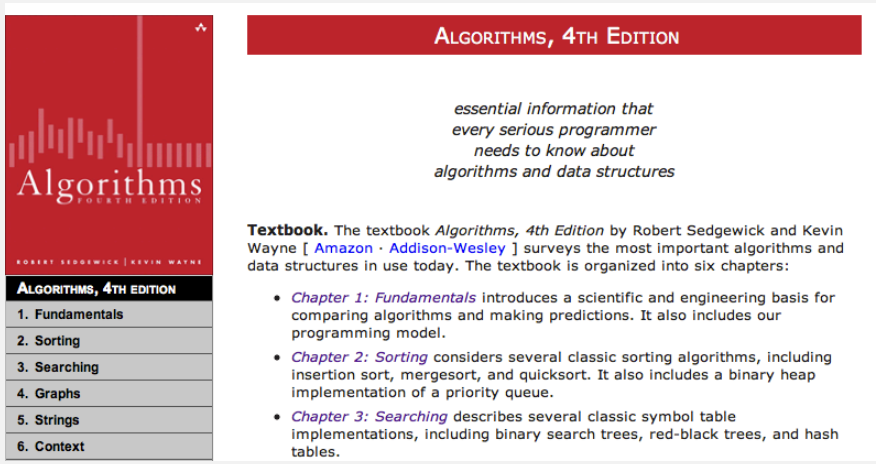
Forums:

Use the announcements forum to post questions or interesting resources you come across

Suggested Reading



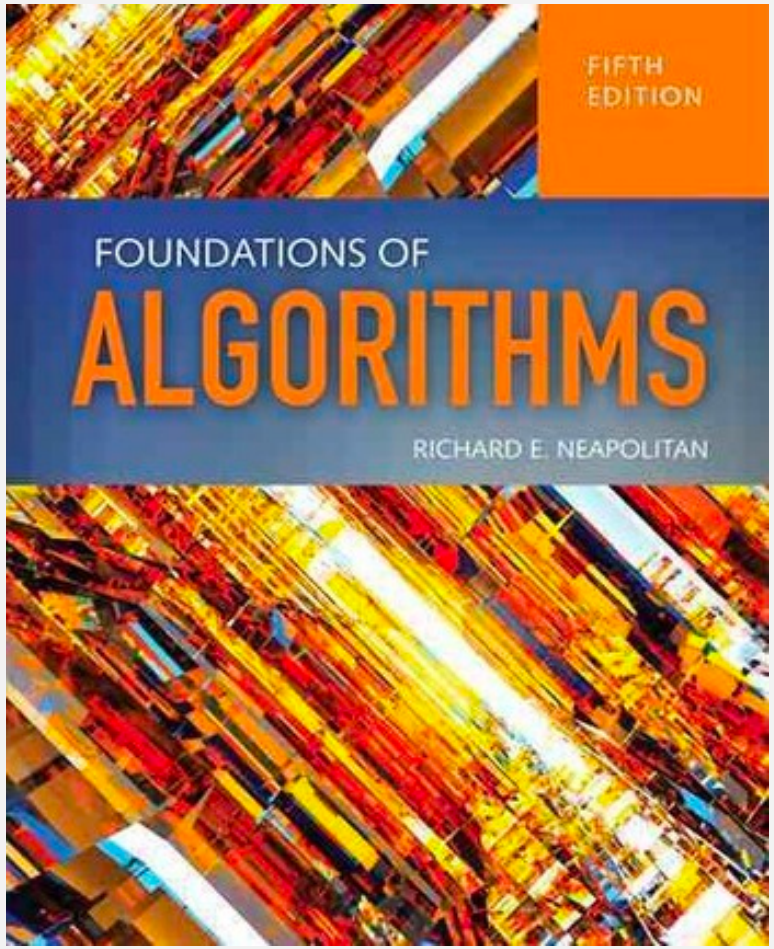
Algorithms
by
Robert Sedgewick (Author), Kevin Wayne (Author)



http://algs4.cs.princeton.edu

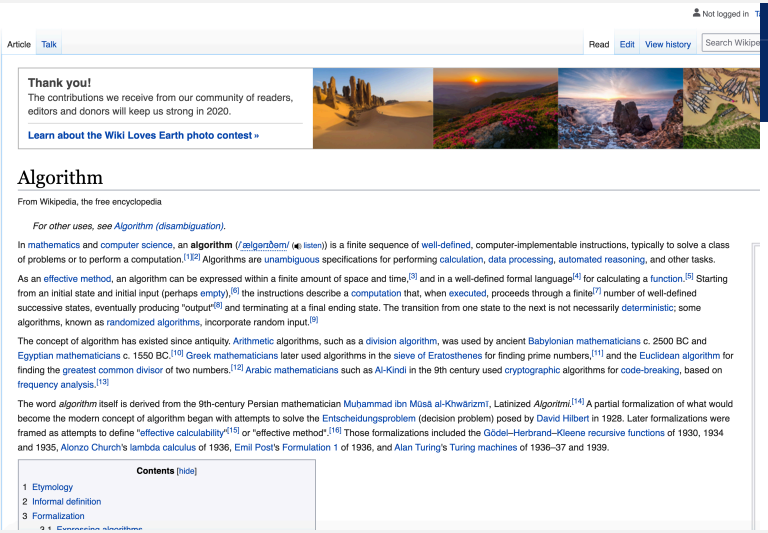
https://algs4.cs.princeton.edu/home/

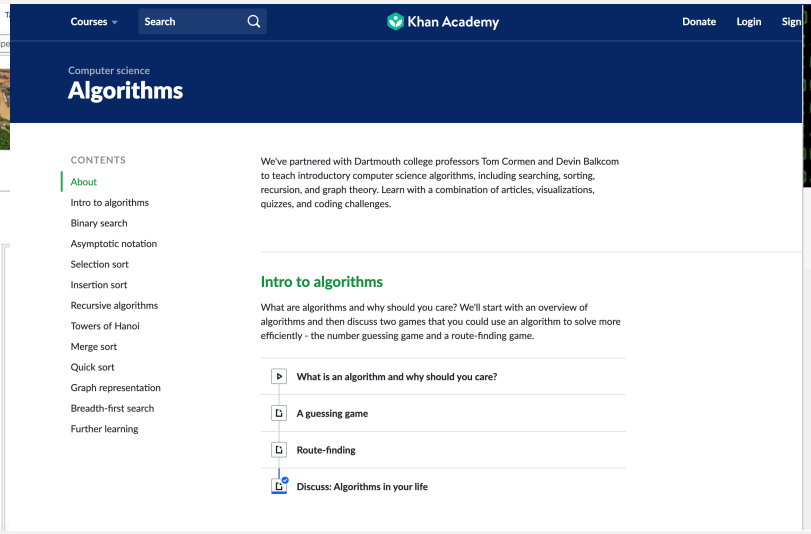
Suggested Resources

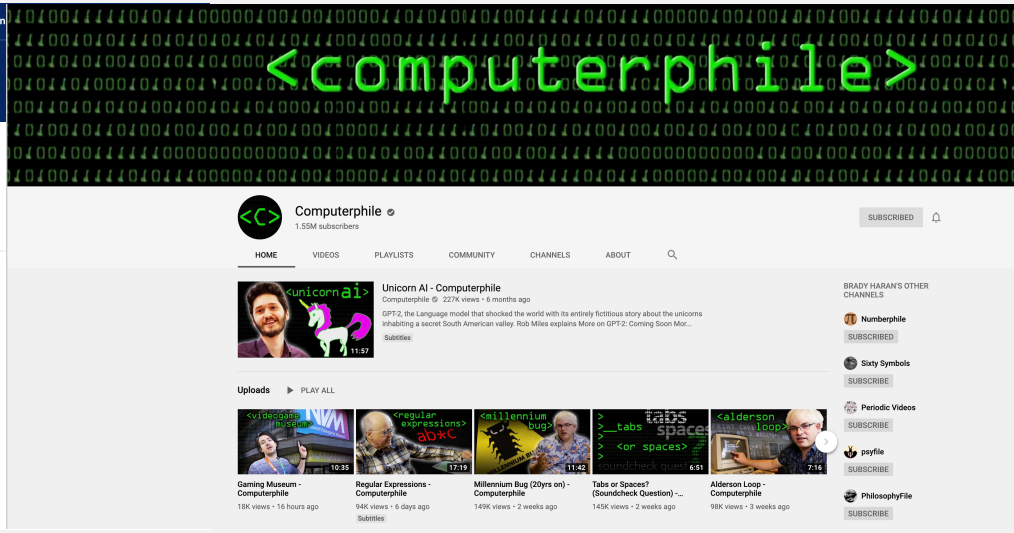


Foundations of Algorithms
by
Richard Neapolitan, Kumarss Naimipour

Resources


Not logged in
Article
Talk
Read
Edit
View history
Thank you!
The contributions we receive from our community of readers, editors and donors will keep us strong in 2020.
Learn about the Wiki Loves Earth photo contest »
Algorithm
From Wikipedia, the free encyclopedia
For other uses, see Algorithm (disambiguation).
In mathematics and computer science, an algorithm (/ˈælgərɪðəm/ (listen)) is a finite sequence of well-defined, computer-implementable instructions, typically to solve a class of problems or to perform a computation.[1][2] Algorithms are unambiguous specifications for performing calculation, data processing, automated reasoning, and other tasks.
As an effective method, an algorithm can be expressed within a finite amount of space and time,[3] and in a well-defined formal language[4] for calculating a function.[5] Starting from an initial state and initial input (perhaps empty),[6] the instructions describe a computation that, when executed, proceeds through a finite[7] number of well-defined successive states, eventually producing "output"[8] and terminating at a final ending state. The transition from one state to the next is not necessarily deterministic; some algorithms, known as randomized algorithms, incorporate random input.[9]
The concept of algorithm has existed since antiquity. Arithmetic algorithms, such as a division algorithm, was used by ancient Babylonian mathematicians c. 2500 BC and Egyptian mathematicians c. 1550 BC.[10] Greek mathematicians later used algorithms in the sieve of Eratosthenes for finding prime numbers,[11] and the Euclidean algorithm for finding the greatest common divisor of two numbers.[12] Arabic mathematicians such as Al-Kindi in the 9th century used cryptographic algorithms for code-breaking, based on frequency analysis.[13]
The word algorithm itself is derived from the 9th-century Persian mathematician Muḥammad ibn Mūsā al-Khwārizmī, Latinized Algoritmi.[14] A partial formalization of what would become the modern concept of algorithm began with attempts to solve the Entscheidungsproblem (decision problem) posed by David Hilbert in 1928. Later formalizations were framed as attempts to define "effective calculability"[15] or "effective method".[16] Those formalizations included the Gödel–Herbrand–Kleene recursive functions of 1930, 1934 and 1935, Alonzo Church's lambda calculus of 1936, Emil Post's Formulation 1 of 1936, and Alan Turing's Turing machines of 1936–37 and 1939.
Contents [hide]
1 Etymology
2 Informal definition
3 Formalization


Courses
Search
Khan Academy
Donate
Login
Sign
Computer science
Algorithms
CONTENTS
About
Intro to algorithms
Binary search
Asymptotic notation
Selection sort
Insertion sort
Recursive algorithms
Towers of Hanoi
Merge sort
Quick sort
Graph representation
Breadth-first search
Further learning
We've partnered with Dartmouth college professors Tom Cormen and Devin Balkcom to teach introductory computer science algorithms, including searching, sorting, recursion, and graph theory. Learn with a combination of articles, visualizations, quizzes, and coding challenges.
Intro to algorithms
What are algorithms and why should you care? We'll start with an overview of algorithms and then discuss two games that you could use an algorithm to solve more efficiently - the number guessing game and a route-finding game.
What is an algorithm and why should you care?
A guessing game
Route-finding
Discuss: Algorithms in your life


<computerphile>
Computerphile
1.55M subscribers
SUBSCRIBED
HOME
VIDEOS
PLAYLISTS
COMMUNITY
CHANNELS
ABOUT
Unicorn AI - Computerphile
Uploads
PLAY ALL
Gaming Museum - Computerphile
Regular Expressions - Computerphile
Millennium Bug (20yrs on) - Computerphile
Tabs or Spaces? (Soundcheck Question) -...
Alderson Loop - Computerphile
BRADY HARAN'S OTHER CHANNELS
Numberphile
Sixty Symbols
Periodic Videos
psyfile
PhilosophyFile


UCD
DUBLIN

Assessment

Assignment	30%
Final Examination	65%
Code repository submitted at end of semester.	5%

Practicals

Will be based on a recent topic from the course.
Will combine theory and/or implementation.

Don’t steal - Plagiarism

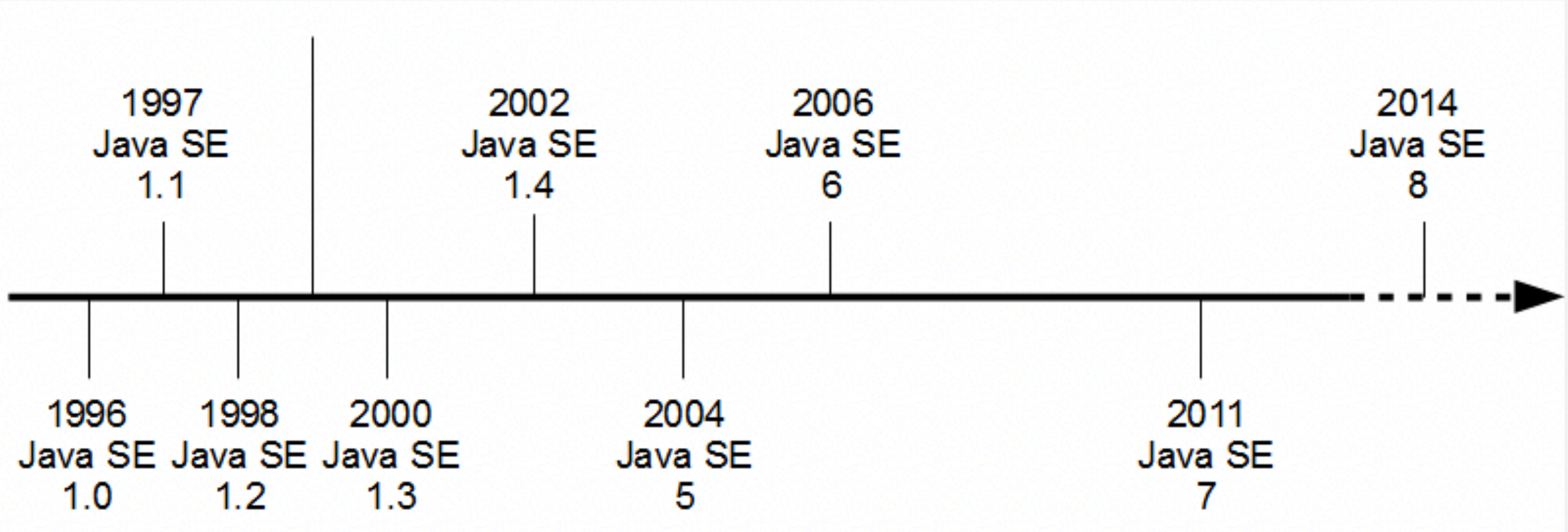
Plagiarism is a serious academic offence

- [Student Code, sections 6.2 & 6.3] or [UCD Registry Plagiarism Policy] or [CS Plagiarism policy and procedures]
- Our staff/demonstrators are proactive in looking for possible plagiarism
- Suspected plagiarism is investigated by the CS Plagiarism subcommittee
  - Usually includes an interview with student(s) involved
  - 1st offence: usually 0 or NG in the affected components
  - 2nd offence: more serious consequences e.g. UCD Disciplinary process
- Student who enables plagiarism is equally responsible for it

- Examples of plagiarism:
  - Copying the files of another student and submitting them as your own work
  - Copying some/all of an assignment from the Internet/book/etc without referencing it
  - Sharing images of your work with another student (by e-mail, FB messenger, WhatsApp, …)
  - A group of students working on a solution, then individually submitting the same work
  - Students collaborating at too detailed a level e.g. consulting each other after implementing a line/block/segment of code and sharing the results

Java Recap

Java timeline



Java recap

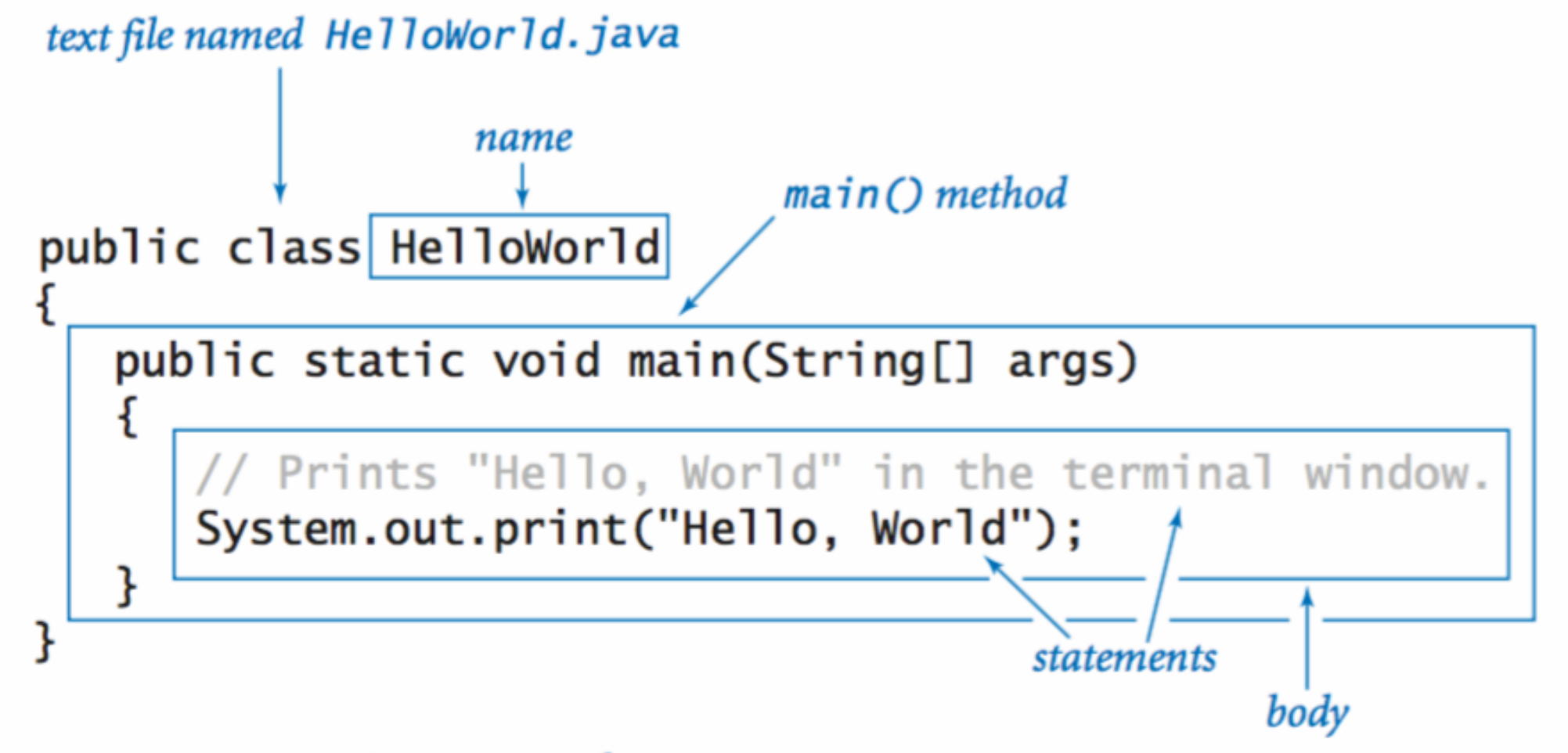
https://www.java.com/en/download/

https://www.eclipse.org/downloads/

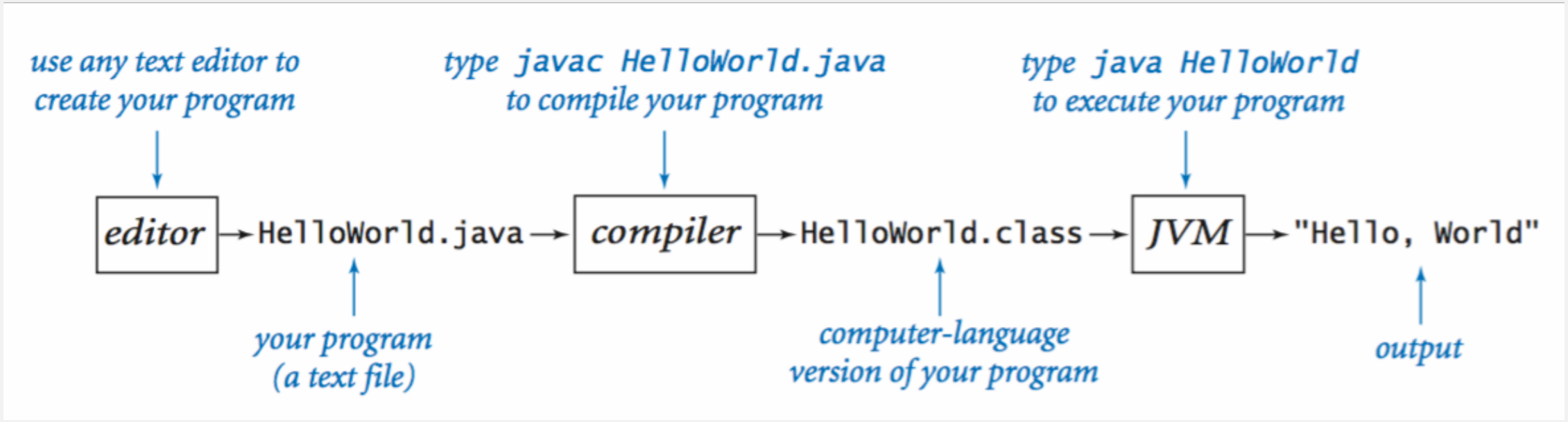
https://www.codecademy.com/learn/learn-java

Course content

HelloWorld


text file named HelloWorld.java
name
main() method
public class HelloWorld
{
public static void main(String[] args)
{
// Prints "Hello, World" in the terminal window.
System.out.print("Hello, World");
}
}
statements
body

Editing, compiling, and executing



Built in data types

type	*set of values*	*common operators*	*sample literal values*
`int`	integers	`+ - * / %`	`99 12 2147483647`
`double`	floating-point numbers	`+ - * /`	`3.14 2.5 6.022e23`
`boolean`	boolean values	`&& || !`	`true false`
`char`	characters		`'A' '1' '%' '\n'`
`String`	sequences of characters	`+`	`"AB" "Hello" "2.5"`

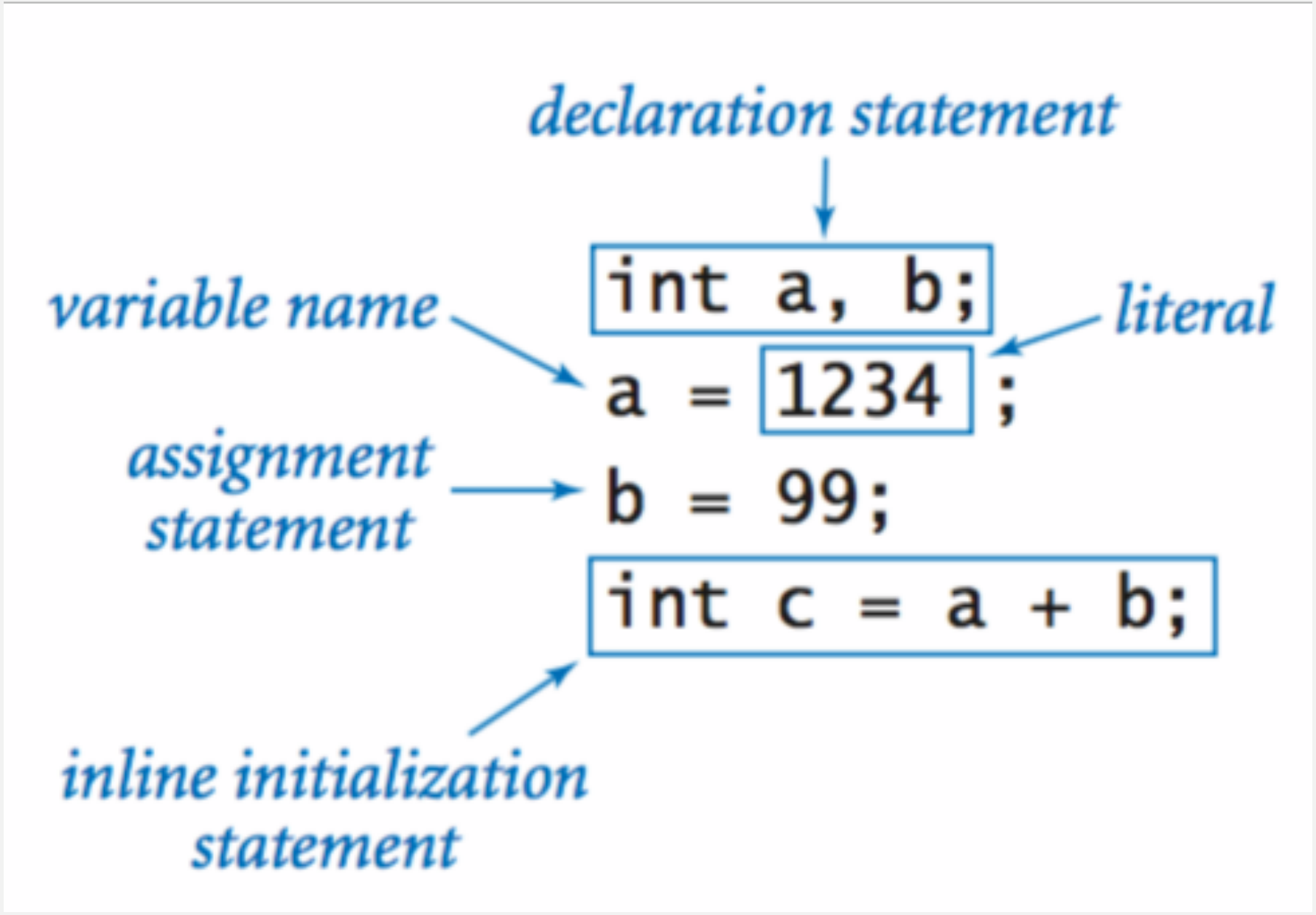
Fundamental Data Types

Primitive Type	What It Stores	Range
`byte`	8-bit integer	–128 to 127
`short`	16-bit integer	–32,768 to 32,767
`int`	32-bit integer	–2,147,483,648 to 2,147,483,647
`long`	64-bit integer	-2^{63} to $2^{63} - 1$
`float`	32-bit floating-point	6 significant digits (10^{-46}, 10^{38})
`double`	64-bit floating-point	15 significant digits (10^{-324}, 10^{308})
`char`	Unicode character	
`boolean`	Boolean variable	`false` and `true`

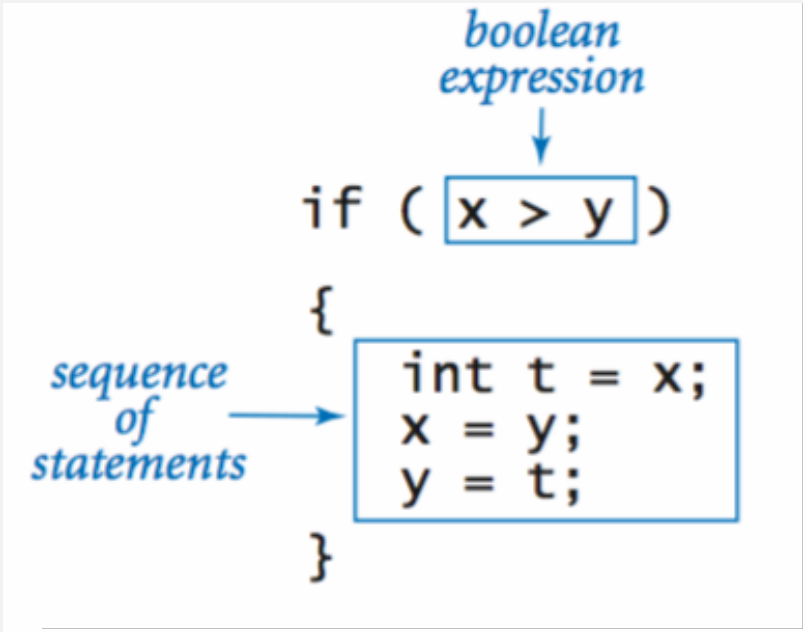
figure 1.2

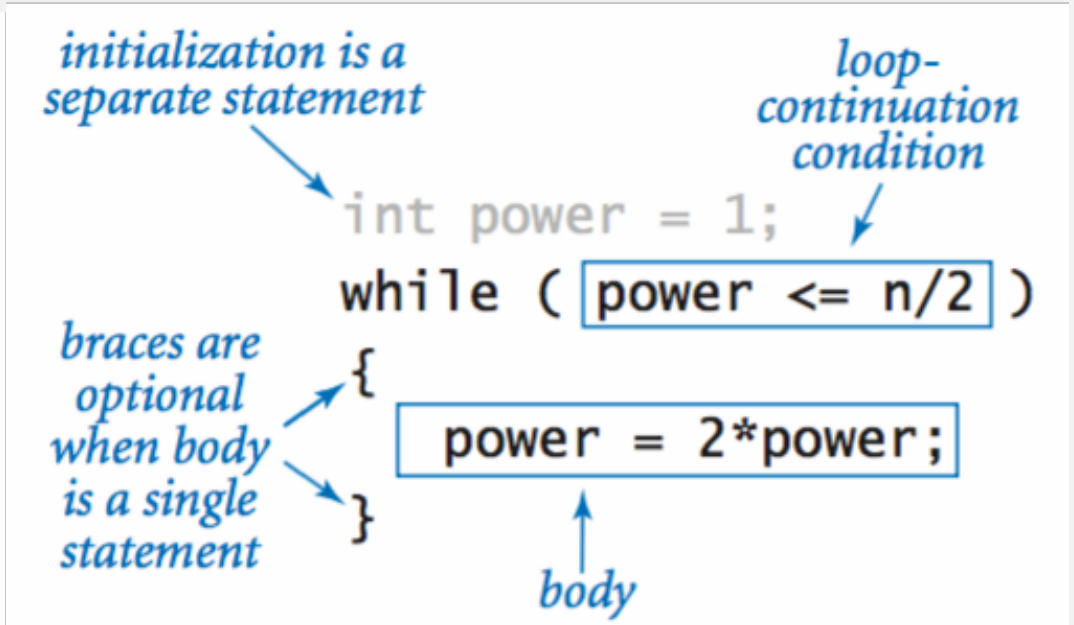
The eight primitive types in Java

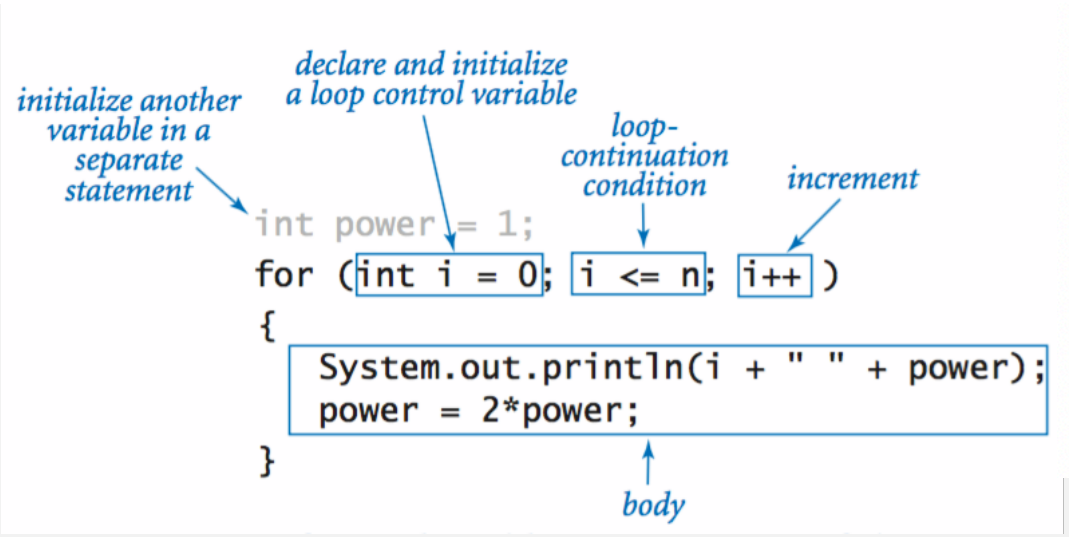
Declaration and assignment statements



loops

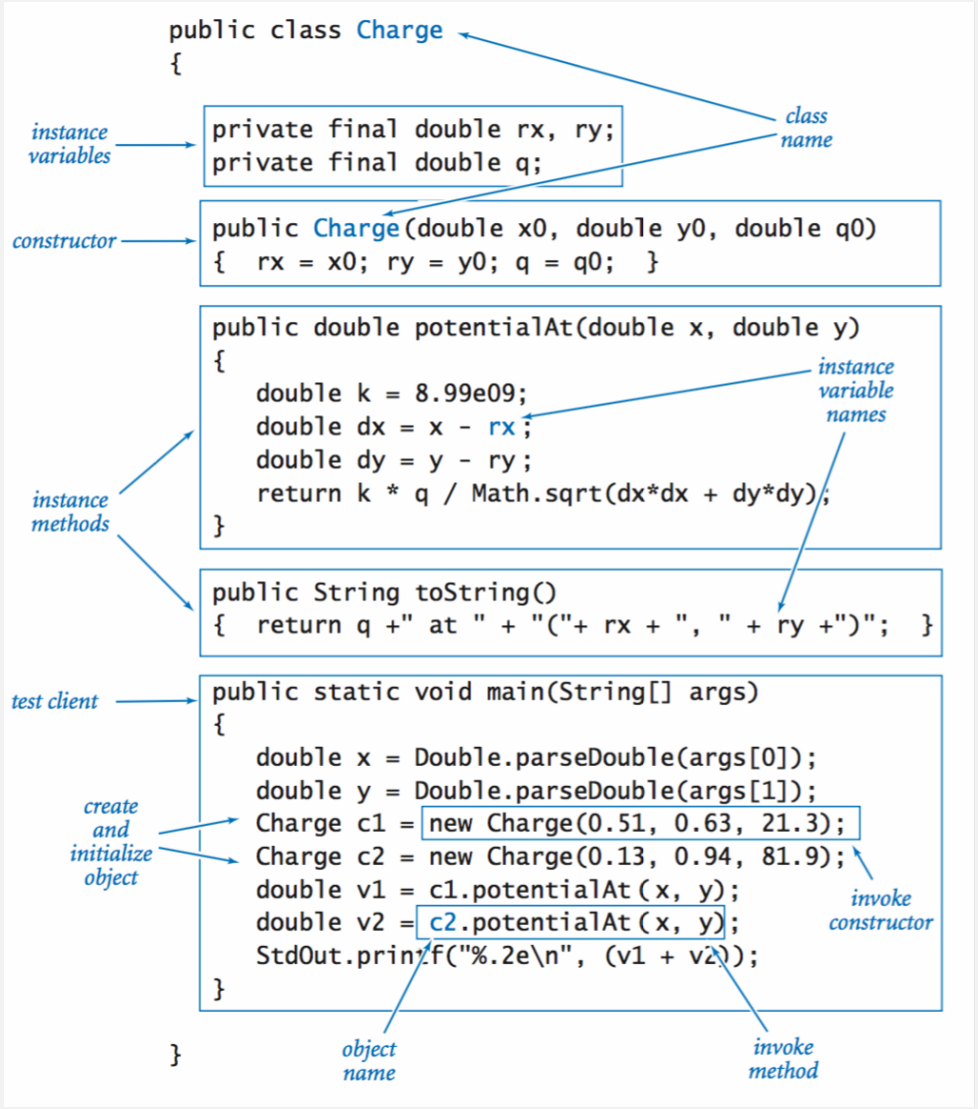




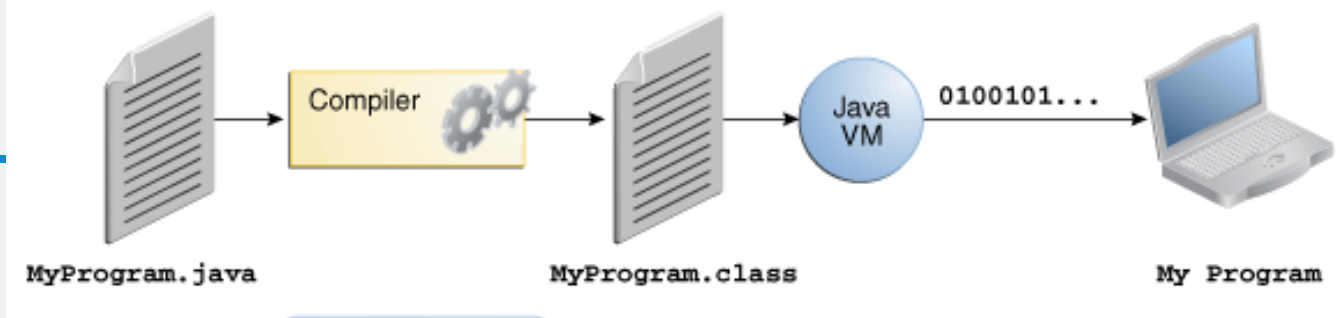



```
do
{  // Scale x and y to be random in (-1, 1).
   x = 2.0*Math.random() - 1.0;
   y = 2.0*Math.random() - 1.0;
} while (Math.sqrt(x*x + y*y) > 1.0);
```

Classes


public class Charge
{
private final double rx, ry;
private final double q;
public Charge(double x0, double y0, double q0)
{ rx = x0; ry = y0; q = q0; }
public double potentialAt(double x, double y)
{
double k = 8.99e09;
double dx = x - rx;
double dy = y - ry;
return k * q / Math.sqrt(dx*dx + dy*dy);
}
public String toString()
{ return q +" at " + "("+ rx + ", " + ry +")"; }
public static void main(String[] args)
{
double x = Double.parseDouble(args[0]);
double y = Double.parseDouble(args[1]);
Charge c1 = new Charge(0.51, 0.63, 21.3);
Charge c2 = new Charge(0.13, 0.94, 81.9);
double v1 = c1.potentialAt(x, y);
double v2 = c2.potentialAt(x, y);
StdOut.printf("%.2e\n", (v1 + v2));
}
}
instance variables
constructor
instance methods
test client
create and initialize object
class name
instance variable names
invoke constructor
object name
invoke method

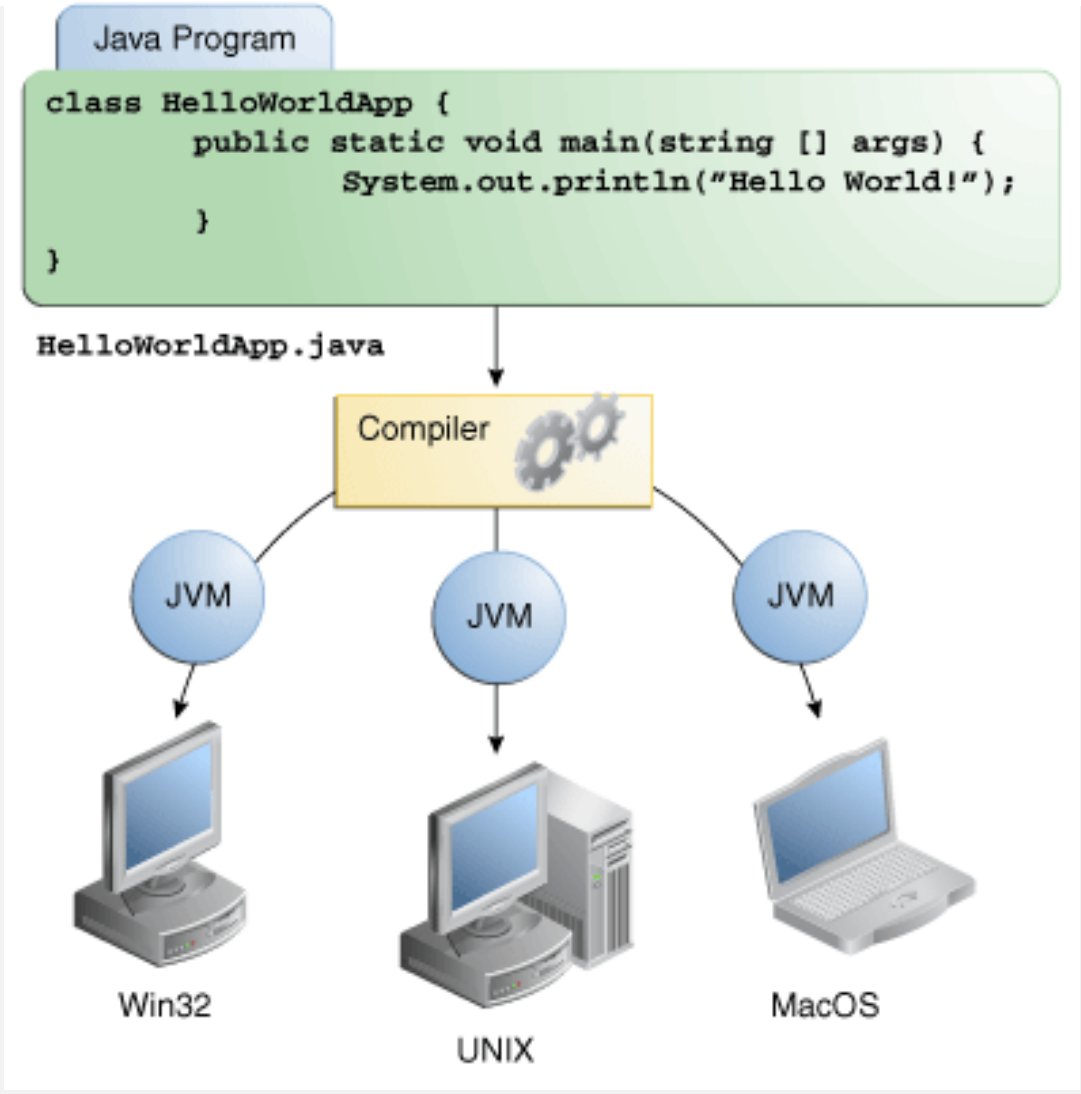
Java recap



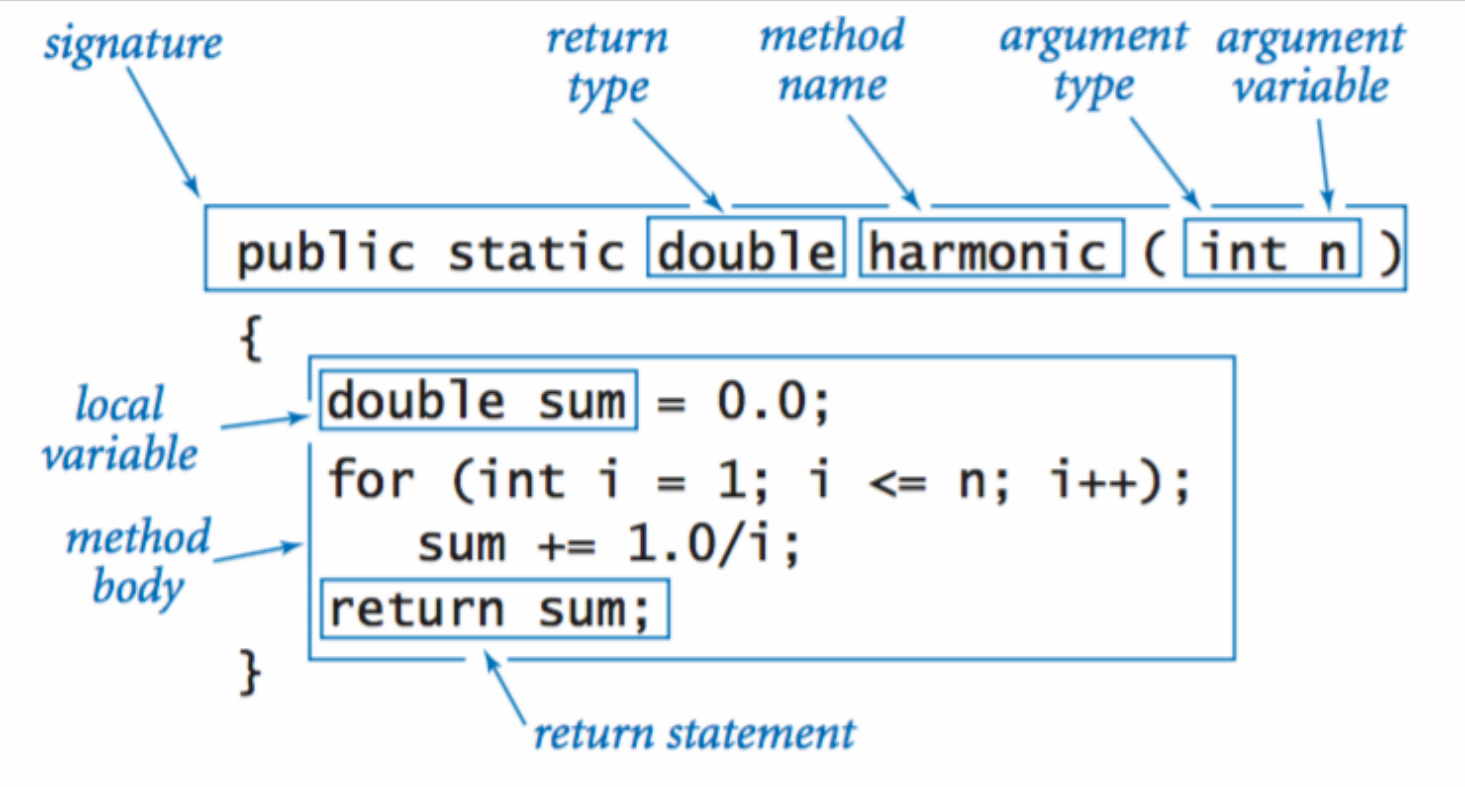
In Java, all source code is first written in plain Those source files are then compiled text files ending with the .java extension into .class files by the javac compiler.

A .class file does not contain code that is native to your processor; it instead contains *bytecodes* — the machine language of the Java Virtual Machine1 (Java VM).

The java launcher tool then runs your application with an instance of the Java Virtual Machine.



Functions


signature
return type
method name
argument type
argument variable
public static double harmonic (int n)
{
local variable
double sum = 0.0;
method body
for (int i = 1; i <= n; i++);
sum += 1.0/i;
return sum;
}
return statement