

A language that doesn't affect the way you think about programming, is not worth knowing.

- Alan Perlis

CSE341

Programming Languages

Lecture 1.1 – September 2019

Introduction

© 2012-2019 Yakup Genç

Largely adapted from V. Shmatikov, J. Mitchell and R.W. Sebesta

Today

- Introductions
- Administrative Details
- Introduction to Programming Languages

September 2019

CSE341 Lecture 1.1

2

Introductions

- Instructor: Dr. Yakup Genc
 - yakup.genc@gtu.edu.tr
 - Room BilMuh 251
- TA:
 - TBA
- Office hours for Dr. Genc
 - Wednesdays 8:30-9:30am

September 2019

CSE341 Lecture 1.1

3

Admin: Communication

- The course communication will be done via Moodle
 - <https://bilmuh.gtu.edu.tr/moodle/course/view.php?id=323>
- Make sure you register as soon as possible

September 2019

CSE341 Lecture 1.1

4

Admin: Prerequisites

- You should know/have
 - Data structures
 - Good working knowledge of C/C++ & Java

September 2019

CSE341 Lecture 1.1

5

Admin: Attendance etc.

- Policy:
 - Attendance is mandatory (%70)
 - Failure to meet requirement will result in a fail
 - No late admittance

September 2019

CSE341 Lecture 1.1

6

Admin: Grading

- Grading (tentative)
 - %40 – Homework and projects
 - Average less than “40 out of 100” will directly result in “FF”
 - %25 – Midterm
 - Less than “30 out of 100” will directly result in “FF”
 - %35 – Final
 - Less than “35 out of 100” will directly result in “FF”
- Rules
 - No attendance → no pass
 - Zero tolerance on cheating

September 2019

CSE341 Lecture 1.1

7

Homework and Projects

- Homework
 - Details to come soon
- You will learn “Common Lisp”
- Rules
 - Hand in on the due date
 - No late submission will be accepted
 - Discuss among yourselves but do your own work, no cheating allowed

September 2019

CSE341 Lecture 1.1

8

Reading Material

- Book:
 - Concepts of Programming Languages by R. W. Sebesta, Eleventh Edition
 - Chapters will be assigned for reading
- Other reading materials will be provided as needed

PROGRAMMING LANGUAGES

Programming Languages

What's a programming language?

A programming language is an artificial language designed to communicate instructions to a machine, particularly a computer [Wikipedia].

A language is a "conceptual universe" providing a framework for problem-solving and useful concepts and programming methods [Perlis].

How many programming languages are there?

Thousands!

Which one to use?

We'll hopefully have a good idea after completing this course.

Top Programming Languages – IEEE Spectrum

IEEE Top Programming Languages: Design, Methods, and Data Sources

The *IEEE Spectrum* Top Programming Languages app synthesizes 12 metrics from 10 sources to arrive at an overall ranking of language popularity. The sources cover contexts that include social chatter, open-source code production, and job postings.

<https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2019>

Top Programming Languages – IEEE Spectrum

What is Tracked

Starting from a list of over 300 programming languages gathered from GitHub, we looked at the volume of results found on Google when we searched for each one using the template “X programming” where “X” is the name of the language. We filtered out languages that had a very low number of search results and then went through the remaining entries by hand to narrow them down to the most interesting. We labeled each language according to whether or not it finds significant use in one or more of the following categories: Web, mobile, enterprise/desktop, or embedded environments.

Our final set of 52 languages includes names familiar to most computer users, such as Java, stalwarts like Cobol and Fortran, and languages that thrive in niches, like Haskell. We gauged the popularity of each using 11 metrics across 8 sources in the following ways:

September 2019

CSE341 Lecture 1.1

13

Top Programming Languages – IEEE Spectrum

Google Search

We measured the number of hits for each language by using Google’s API to search for the template “X programming.” This number indicates the volume of online information resources about each programming language. We took the measurement in June 2019, so it represents a snapshot of the Web at that particular moment in time. This measurement technique is also used by the oft-cited [TIOBE rankings](#).

Google Trends

We measured the index of each language as reported by [Google Trends](#) using the template “X programming” in June 2019. This number indicates the demand for information about the particular language, because Google Trends measures how often people search for the given term. As it measures searching activity rather than information availability, Google Trends can be an early cue to up-and-coming languages. Our methodology here is similar to that of the Popularity of Programming Language [\(PYPL\) ranking](#).

September 2019

CSE341 Lecture 1.1

14

Top Programming Languages – IEEE Spectrum

Twitter

We measured the number of hits on Twitter for the template “X programming” for the 12 months ending June 2019 using the [Twitter Search](#) API. This number indicates the amount of chatter on social media for the language and reflects the sharing of online resources like news articles or books, as well as physical social activities such as hackathons.

GitHub

[GitHub](#) is a site where programmers can collaboratively store repositories of code. Using the [GitHub API](#) and GitHub tags, we measured two things for the 12 months ending June 2019: (1) the number of new repositories created for each language, and (2) the number of active repositories for each language, where “active” means that someone has edited the code in a particular repository. The number of new repositories measures fresh activity around the language, whereas the number of active repositories measures the ongoing interest in developing each language.

Stack Overflow

[Stack Overflow](#) is a popular site where programmers can ask questions about coding. We measured the number of questions posted that mention each language for the 12 months ending June 2019. Each question is tagged with the languages under discussion, and these tags are used to tabulate our measurements using the [Stack Exchange API](#).

Reddit

[Reddit](#) is a news and information site where users post links and comments. On Reddit we measured the number of posts mentioning each of the languages, using the template “X programming” from June 2018 to June 2019 across any subreddit on the site. We collected data using the [Reddit API](#).

September 2019

CSE341 Lecture 1.1

15

Top Programming Languages – IEEE Spectrum

Hacker News

[Hacker News](#) is a news and information site where users post comments and links to news about technology. We measured the number of posts that mentioned each of the languages using the template “X programming” for the 12 months ending June 2019. Just like those used by the websites Topsy, Stack Overflow, and Reddit, this metric also captures social activity and information sharing around the various languages. We used the [Algolia Search API](#).

CareerBuilder

We measured the demand for different programming languages on the [CareerBuilder](#) job site. We measure the number of fresh job openings (those that are less than 30 days old) on the U.S. site that mention the language. Because some of the languages we track could be ambiguous in plain text—such as D, Go, J, Processing, and R—we use strict matching of the form “X programming” for these languages. For other languages we use a search string composed of “X AND programming,” which allows us to capture a broader range of relevant postings. We collected data in June 2019 using the [CareerBuilder API](#).

IEEE Job Site

We measured the demand for different programming languages in job postings on the [IEEE Job Site](#). Because some of the languages we track could be ambiguous in plain text—such as D, Go, J, Processing, and R—we use strict matching of the form “X programming” for these languages. For other languages we use a search string composed of “X AND programming,” which allows us to capture a broader range of relevant postings. Because no externally exposed API exists for the IEEE Job Site, data was extracted using an internal custom-query tool in June 2019.

IEEE Xplore Digital Library

IEEE maintains a digital library with over 3.6 million conference and journal articles covering a range of scientific and engineering disciplines. We measured the number of articles that mention each of the languages in the template “X programming” for the years 2018 and 2019. This metric captures the prevalence of the different programming languages as used and referenced in scholarship. We collected data using the [IEEE Xplore API](#).

September 2019

CSE341 Lecture 1.1

16

Top 10

September 2019

CSE341 Lecture 1.1

17

Top Programming Languages – IEEE Spectrum

Rank	Language	Type	Score
1	Python	☰ ☒ ☑	100.0
2	Java	☰ ☒ ☑	96.3
3	C	☒ ☑ ☑	94.4
4	C++	☒ ☑ ☑	87.5
5	R	☑	81.5
6	JavaScript	☰	79.4
7	C#	☰ ☒ ☑ ☑	74.5
8	Matlab	☑	70.6
9	Swift	☒ ☑	69.1
10	Go	☰ ☑	68.0

September 2019

September 2019

CSE341 Lecture 1.1

18

Top Programming Languages – IEEE Spectrum

Language Rank	Types	Spectrum Ranking
1. Python	☰ ☒ ☑	100.0
2. C++	☒ ☑ ☑	99.7
3. Java	☰ ☒ ☑	97.5
4. C	☒ ☑ ☑	96.7
5. C#	☰ ☒ ☑	89.4
6. PHP	☰	84.9
7. R	☑	82.9
8. JavaScript	☰ ☒	82.6
9. Go	☰ ☑	76.4
10. Assembly	☑	74.1

September 2018

September 2019

CSE341 Lecture 1.1

19

Top Programming Languages – IEEE Spectrum

Language Rank	Types	Spectrum Ranking
1. Python	☰ ☒ ☑	100.0
2. C	☒ ☑ ☑	99.7
3. Java	☰ ☒ ☑	99.4
4. C++	☒ ☑ ☑	97.2
5. C#	☰ ☒ ☑	88.6
6. R	☑	88.1
7. JavaScript	☰ ☒	85.5
8. PHP	☰	81.4
9. Go	☰ ☑	76.1
10. Swift	☒ ☑	75.3

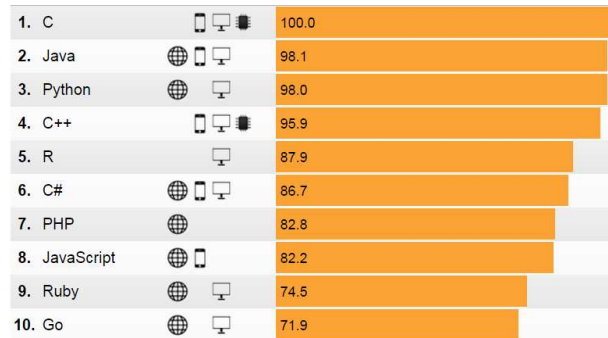
September 2017

September 2019

CSE341 Lecture 1.1

20

Top Programming Languages – IEEE Spectrum



September 2016

September 2019

CSE341 Lecture 1.1

21

Top 20

September 2019

CSE341 Lecture 1.1

22

Top Programming Languages – IEEE Spectrum



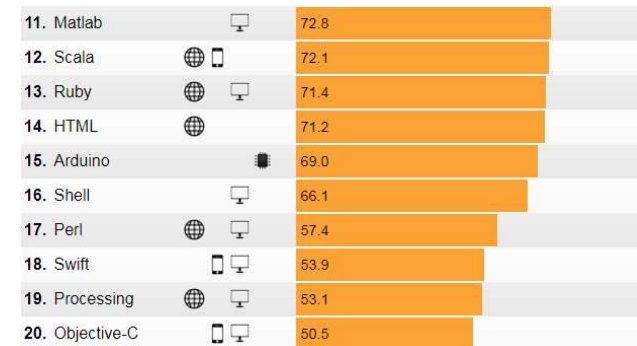
September 2019

September 2019

CSE341 Lecture 1.1

23

Top Programming Languages – IEEE Spectrum



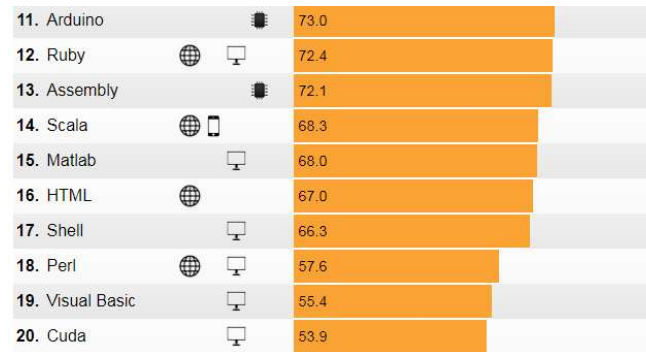
September 2018

September 2019

CSE341 Lecture 1.1

24

Top Programming Languages – IEEE Spectrum



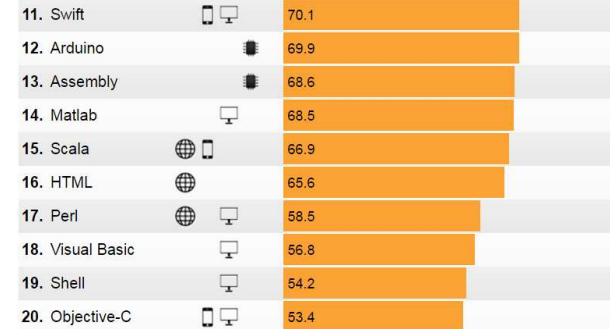
September 2017

September 2019

CSE341 Lecture 1.1

25

Top Programming Languages – IEEE Spectrum



September 2016

September 2019

CSE341 Lecture 1.1

26

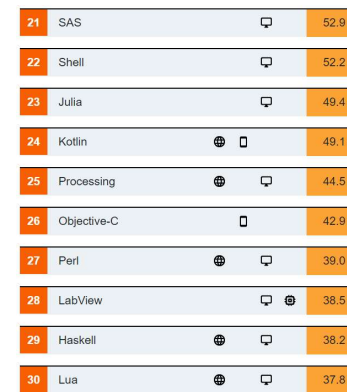
Top 30

September 2019

CSE341 Lecture 1.1

27

Top Programming Languages – IEEE Spectrum



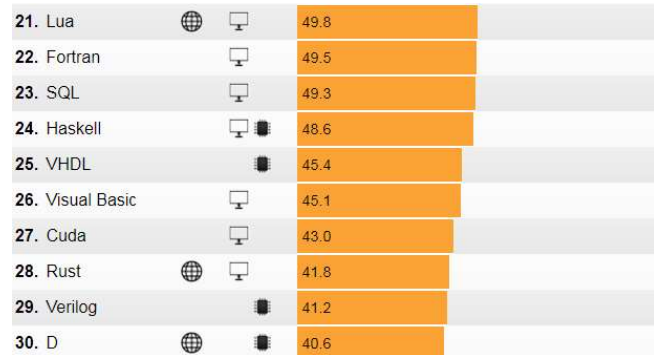
September 2019

September 2019

CSE341 Lecture 1.1

28

Top Programming Languages – IEEE Spectrum



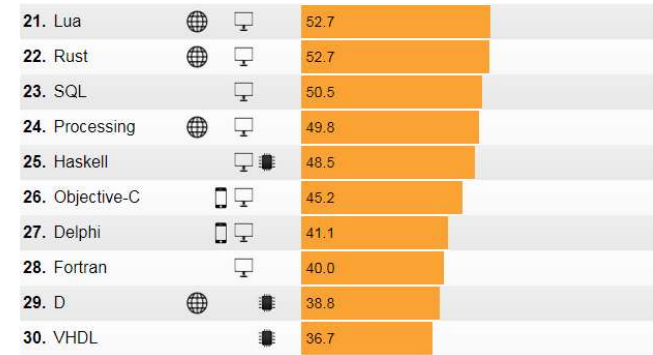
September 2018

September 2019

CSE341 Lecture 1.1

29

Top Programming Languages – IEEE Spectrum



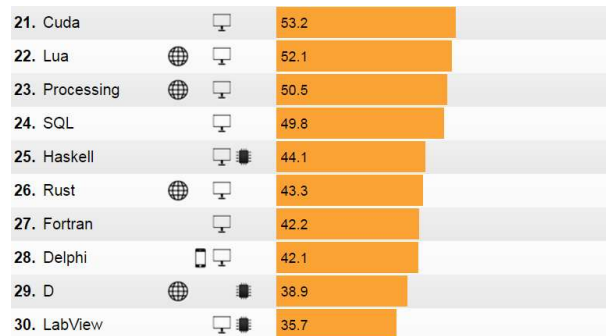
September 2017

September 2019

CSE341 Lecture 1.1

30

Top Programming Languages – IEEE Spectrum



September 2016

September 2019

CSE341 Lecture 1.1

31

Top 50

September 2019

CSE341 Lecture 1.1

32

Top Programming Languages – IEEE Spectrum

Rank	Language	Type	Score	Rank	Language	Type	Score
1	Python	⊕ ☐ ☐	100.0	11	Arduino	⊕	67.2
2	Java	⊕ ☐ ☐	96.3	12	HTML/CSS	⊕	66.8
3	C	☐ ☐ ☐	94.4	13	PHP	⊕	65.1
4	C++	☐ ☐ ☐	87.5	14	Assembly	⊕	63.7
5	R	☐	81.5	15	SQL	☐	63.4
6	JavaScript	⊕	79.4	16	Dart	⊕ ☐	57.4
7	C#	⊕ ☐ ☐	74.5	17	Rust	⊕ ☐ ☐	55.5
8	Matlab	☐	70.6	18	Scala	⊕ ☐ ☐	55.3
9	Swift	☐ ☐	69.1	19	Ruby	⊕ ☐	55.1
10	Go	⊕ ☐	68.0	20	Visual Basic	☐	55.1

September 2019

September 2019

CSE341 Lecture 1.1

33

Top Programming Languages – IEEE Spectrum

21	SAS	☐	52.9	31	Cuda	☐	37.3
22	Shell	☐	52.2	32	VHDL	⊕	36.0
23	Julia	☐	49.4	33	Verilog	⊕	33.4
24	Kotlin	⊕ ☐	49.1	34	ABAP	☐	33.0
25	Processing	⊕ ☐	44.5	35	Delphi	⊕ ☐ ☐	30.1
26	Objective-C	☐	42.9	36	Fortran	☐	29.4
27	Perl	⊕ ☐	39.0	37	Clojure	⊕ ☐	28.1
28	LabView	☐ ☐	38.5	38	Apache Groovy	⊕ ☐	27.3
29	Haskell	⊕ ☐	38.2	39	Scheme	☐ ☐	27.0
30	Lua	⊕ ☐	37.8	40	TCL	☐ ☐	26.9

September 2019

September 2019

CSE341 Lecture 1.1

34

Top Programming Languages – IEEE Spectrum

41	Lisp	☐	26.6
42	D	☐ ☐ ☐	25.6
43	Ada	☐ ☐	24.8
44	Cobol	☐	24.1
45	Erlang	☐ ☐	23.9
46	Prolog	☐	23.5
47	Forth	⊕	18.3
48	OCaml	⊕ ☐	16.0
49	J	☐	12.7
50	Ladder Logic	⊕	12.2

September 2019

September 2019

CSE341 Lecture 1.1

35

Top Programming Languages – TIOBE

TIOBE Index for September 2019

September Headline: PHP is struggling to keep its top 10 position

PHP has been in the TIOBE index top 10 since the start of the index in 2001. It was even TIOBE's programming of the year in 2004. Till the end of 2009 everything went fine, but soon after that PHP was going downhill from 10% to 5% market share in 2 years' time. In 2014 it halved again to 2.5%. So what happened to PHP? From its start PHP was the Visual Basic for web design: easy to learn, easy to deploy, but mainly used by web designers with a limited software engineering background. The downside of PHP's simplicity was that it was relatively easy to shoot security holes in it. PHP has been struggling with this for a long time. In 2014 PHP's biggest supporter Facebook launched Hack as an alternative for PHP because it was not scalable. And after that, JavaScript, TypeScript and Python became the lingua franca for web development. So the question is how PHP is going to survive in this jungle? Let's see happens.

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. It is important to note that the TIOBE index is not about the best programming language or the language in which most lines of code have been written.

The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found [here](http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html).

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

September 2019

CSE341 Lecture 1.1

36

Top Programming Languages – TIOBE

Sep 2019	Sep 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.661%	-0.78%
2	2		C	15.205%	-0.24%
3	3		Python	9.874%	+2.22%
4	4		C++	5.635%	-1.76%
5	6	▲	C#	3.399%	+0.10%
6	5	▼	Visual Basic .NET	3.291%	-2.02%
7	8	▲	JavaScript	2.128%	-0.00%
8	9	▲	SQL	1.944%	-0.12%
9	7	▼	PHP	1.863%	-0.91%
10	10		Objective-C	1.840%	+0.33%
11	34	▲	Groovy	1.502%	+1.20%
12	14	▲	Assembly language	1.378%	+0.15%
13	11	▼	Delphi/Object Pascal	1.335%	+0.04%
14	16	▲	Go	1.220%	+0.14%
15	12	▼	Ruby	1.211%	-0.08%

September 2019

TIOBE Programming Community Index for September 2019
CSE341 Lecture 1.1 37

Top Programming Languages – TIOBE

Sep 2018	Sep 2017	Change	Programming Language	Ratings	Change
1	1		Java	17.436%	+4.75%
2	2		C	15.447%	+8.06%
3	5	▲	Python	7.653%	+4.67%
4	3	▼	C++	7.394%	+1.83%
5	8	▲	Visual Basic .NET	5.308%	+3.33%
6	4	▼	C#	3.295%	-1.48%
7	6	▼	PHP	2.775%	+0.57%
8	7	▼	JavaScript	2.131%	+0.11%
9	-	▲	SQL	2.062%	+2.06%
10	18	▲	Objective-C	1.509%	+0.00%
11	12	▲	Delphi/Object Pascal	1.292%	-0.49%
12	10	▼	Ruby	1.291%	-0.64%
13	16	▲	MATLAB	1.276%	-0.35%
14	15	▲	Assembly language	1.232%	-0.41%
15	13	▼	Swift	1.223%	-0.54%

September 2019

TIOBE Programming Community Index for September 2018
CSE341 Lecture 1.1 38

Top Programming Languages – TIOBE

Sep 2017	Sep 2016	Change	Programming Language	Ratings	Change
1	1		Java	12.667%	-5.55%
2	2		C	7.382%	-3.57%
3	3		C++	5.565%	-1.09%
4	4		C#	4.779%	-0.71%
5	5		Python	2.963%	-1.32%
6	7	▲	PHP	2.210%	-0.64%
7	6	▼	JavaScript	2.017%	-0.91%
8	9	▲	Visual Basic .NET	1.982%	-0.36%
9	10	▲	Perl	1.952%	-0.38%
10	12	▲	Ruby	1.933%	-0.03%
11	18	▲	R	1.816%	+0.13%
12	11	▼	Delphi/Object Pascal	1.782%	-0.39%
13	13		Swift	1.765%	-0.17%
14	17	▲	Visual Basic	1.751%	-0.01%
15	8	▼	Assembly language	1.639%	-0.78%

September 2019

TIOBE Programming Community Index for September 2017
CSE341 Lecture 1.1 39

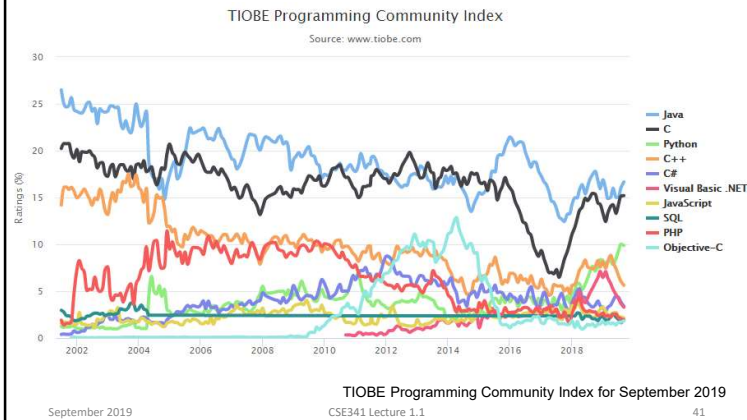
Top Programming Languages – TIOBE

Sep 2016	Sep 2015	Change	Programming Language	Ratings	Change
1	1		Java	18.236%	-1.33%
2	2		C	10.955%	-4.67%
3	3		C++	6.657%	-0.13%
4	4		C#	5.493%	+0.58%
5	5		Python	4.302%	+0.64%
6	7	▲	JavaScript	2.929%	+0.59%
7	6	▼	PHP	2.847%	+0.32%
8	11	▲	Assembly language	2.417%	+0.61%
9	8	▼	Visual Basic .NET	2.343%	+0.28%
10	9	▼	Perl	2.333%	+0.43%
11	13	▲	Delphi/Object Pascal	2.169%	+0.42%
12	12		Ruby	1.965%	+0.18%
13	16	▲	Swift	1.930%	+0.74%
14	10	▼	Objective-C	1.849%	+0.03%
15	17	▲	MATLAB	1.826%	+0.65%

September 2019

TIOBE Programming Community Index for September 2016
CSE341 Lecture 1.1 40

Top Programming Languages – TIOBE



Top Programming Languages – TIOBE

Programming Language	2019	2014	2009	2004	1999	1994	1989
Java	1	2	1	1	9	-	-
C	2	1	2	2	1	1	1
Python	3	7	5	6	23	21	-
C++	4	4	3	3	2	2	2
Visual Basic .NET	5	9	-	-	-	-	-
C#	6	5	6	7	18	-	-
JavaScript	7	8	8	8	16	-	-
PHP	8	6	4	5	-	-	-
SQL	9	-	-	89	-	-	-
Objective-C	10	3	29	37	-	-	-
Perl	17	11	7	4	3	10	19
Lisp	32	15	17	13	13	5	3
Pascal	222	16	14	88	6	3	21

TIOBE Programming Community Index for September 2019

September 2019 CSE341 Lecture 1.1 42

Top Programming Languages – TIOBE

Year	Winner
2018	Python
2017	C
2016	Go
2015	Java
2014	JavaScript
2013	Transact-SQL
2012	Objective-C
2011	Objective-C
2010	Python
2009	Go
2008	C
2007	Python
2006	Ruby
2005	Java
2004	PHP
2003	C++

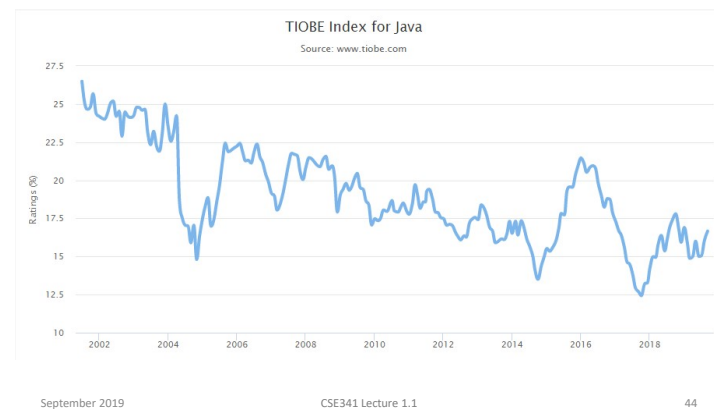
TIOBE Programming Community Index for September 2019

September 2019

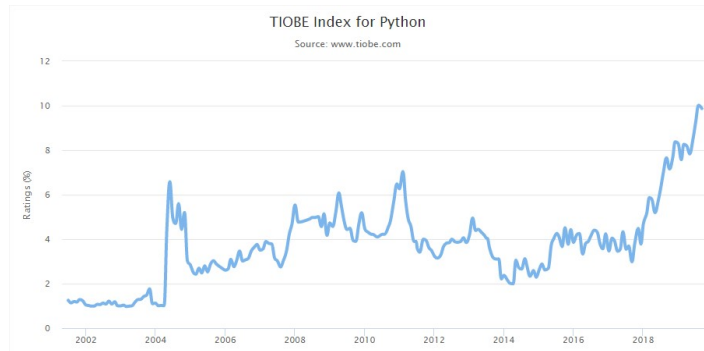
CSE341 Lecture 1.1

43

Java or Python



Java or Python

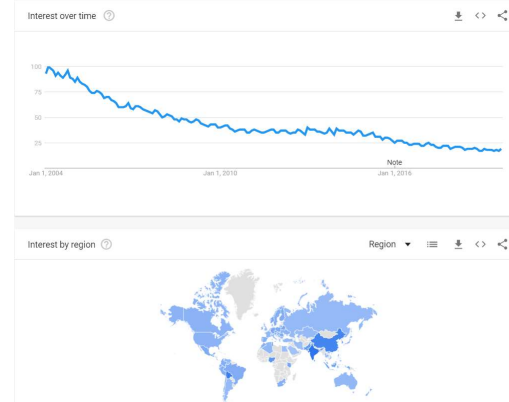


September 2019

CSE341 Lecture 1.1

45

Java or Python

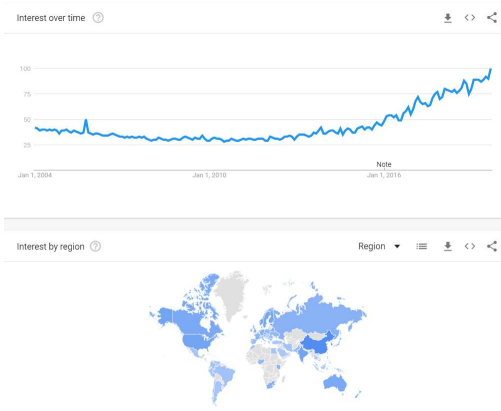


September 2019

CSE341 Lecture 1.1

46

Java or Python

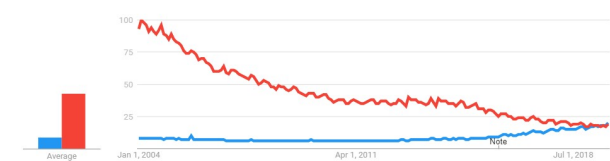


September 2019

CSE341 Lecture 1.1

47

Java or Python



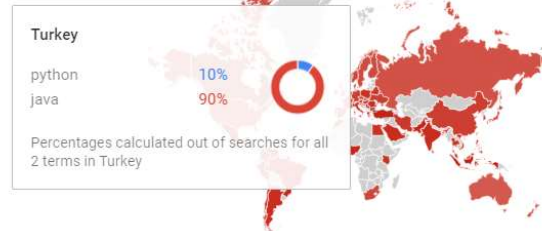
September 2019

CSE341 Lecture 1.1

48

Java or Python

python java



September 2019

CSE341 Lecture 1.1

49

Java or Python

python java



September 2019

CSE341 Lecture 1.1

50

Programming Languages – Popularity

<https://redmonk.com/sograpy/2019/03/20/language-rankings-1-19/>

September 2019

CSE341 Lecture 1.1

51

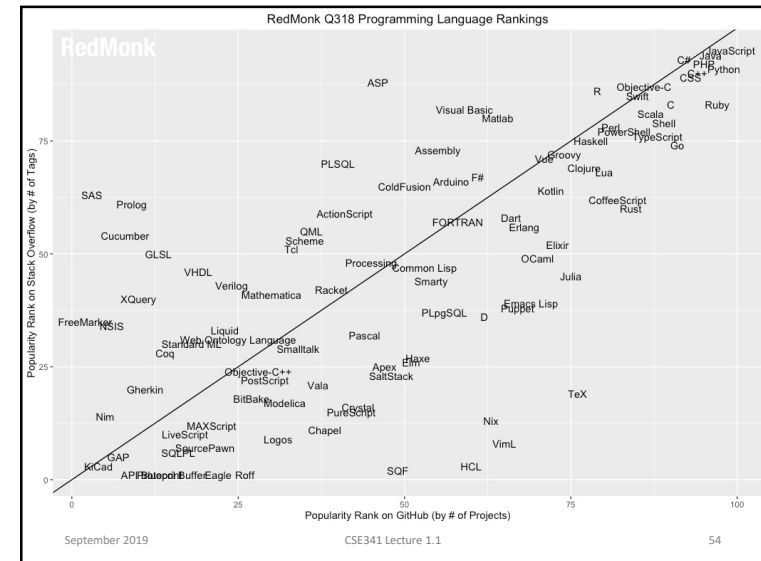
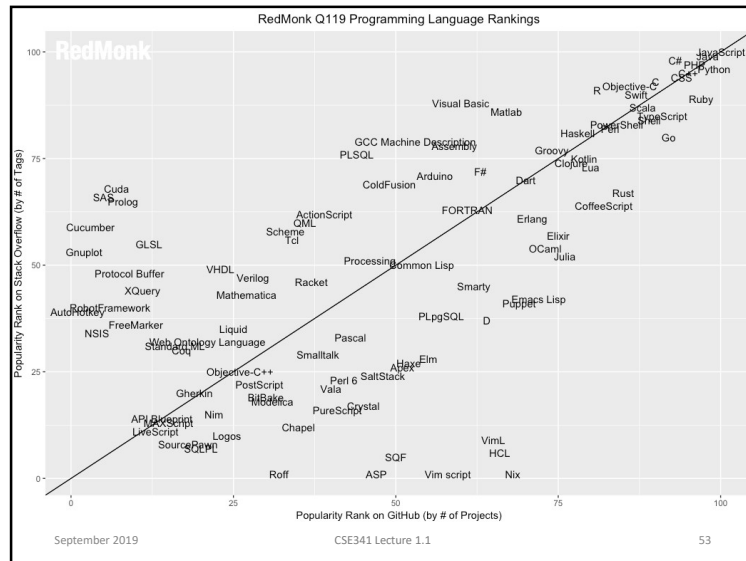
Programming Languages – Popularity

- The data source used for the GitHub portion of the analysis is the GitHub Archive. We query languages by pull request in a manner similar to the one GitHub used to assemble the 2016 State of the Octoverse. Our query is designed to be as comparable as possible to the previous process.
- Language is based on the base repository language. While this continues to have the caveats outlined below, it does have the benefit of cohesion with our previous methodology.
- We exclude forked repos.
- We use the aggregated history to determine ranking (though based on the table structure changes this can no longer be accomplished via a single query.)
- For Stack Overflow, we simply collect the required metrics using their useful [data explorer](#) tool.

September 2019

CSE341 Lecture 1.1

52



Programming Languages – Popularity

- 1 JavaScript
- 2 Java
- 3 Python
- 4 PHP
- 5 C#
- 6 C++
- 7 CSS
- 8 Ruby
- 9 C
- 10 Objective-C
- 11 Swift
- 12 TypeScript
- 13 Scala
- 14 Shell
- 15 Go
- 16 R
- 17 PowerShell
- 18 Perl
- 19 Haskell
- 20 Kotlin

... there was little movement within our Tier 1 languages. Generally speaking, the top ten to twelve languages in these rankings tend to be relatively static, with changes both rare and minor in nature. While the landscape remains fantastically diverse in terms of technologies and approaches employed, including the variety of programming languages in common circulation, code written and discussion are counting metrics, and thus accretive. This makes growth for new languages tougher to come by the higher they ascend the rankings – which makes any rapid growth that much more noticeable.

September 2019

CSE341 Lecture 1.1

55

Programming Languages – Popularity

TypeScript (+4): When we ran these rankings a year ago at this time, TypeScript had surged into the Top 20 landing at #17. It didn't quite match that jump in this run, but movement within the top 20 is much more difficult to accomplish so its four spot bump is notable for that reason alone. It is also notable because by moving up four spots, it finds itself in 12th place, just outside the Top 10 and right behind Swift – the fastest growing language in the history of these rankings. The language certainly benefits from its JavaScript proximity, as well as safety features such as the optional static type-checking. But features alone are never enough by themselves to propel a language this far this quickly – it must be leveraged by a wide base of growing projects – all of which explains why TypeScript's trajectory is significant and sustainable.

September 2019

CSE341 Lecture 1.1

56

Programming Languages – Popularity

Go (-1), R (-1): With TypeScript jumping into the twelfth spot on the rankings, something had to give and in part it was Go and R which dropped one spot respectively into a tie for #15. In the grand scheme of things, this is relatively meaningless as the difference between one spot and another is often superficial, particularly the further down the list a language is found. This is particularly true for the R language, which continues to demonstrate robust, near Tier 1 usage thanks to a vibrant base of analytical and data science use cases. Given the domain specific nature and comparatively narrow focus of R, its prospects probably do not include a Top 10 placement; the mid second tier is likely its ceiling. For Go, on the other hand, it is reasonable to question what its stagnation in this second tier means for the language's future. It is highly regarded technically, and enjoys popularity across a wide variety of infrastructure projects. To date, however, it has not demonstrated an ability or inclination to follow in the footsteps of languages such as Java and expand its core use cases.

September 2019

CSE341 Lecture 1.1

57

Programming Languages – Popularity

Kotlin (+8), Scala (-1), Clojure (-3), Groovy (-3): One of the primary questions we had going into this quarter's rankings was whether or not JVM-based languages such as Clojure, Groovy and Scala could repeat the last rankings' performance in which all three grew while newcomer Kotlin declined. We now have a clear answer to that question, and it's no. For this quarter, at least, Kotlin grew substantially while all three of its fellow JVM-based counterparts declined. Kotlin jumped so far, in fact, that it finally broke into the Top 20 at #20 and leapfrogged Clojure (#24) and Groovy (#24) while doing so. It's still well behind Scala (#13), but Kotlin's growth has been second only to Swift in this history of these rankings so it will be interesting to see what lies ahead in the next run or two.

September 2019

CSE341 Lecture 1.1

58

Programming Languages – Popularity

Julia: For a language that isn't even in the Top 30, Julia continues to attract questions about its performance and future. Its growth has been more tortoise than hare, but it's up another two spots to place #34. While there is no technical basis for comparison, it is worth noting that three years ago in our Q1 rankings TypeScript made a similar modest jump from #33 to #31. That is not to say that Julia is destined to follow in TypeScript's footsteps, of course, but rather to serve as a reminder that while it's uncommon languages can transition quickly from periods of slow, barely measurable growth to high, sustained growth quarter after quarter.

September 2019

CSE341 Lecture 1.1

59

Programming Languages – Popularity

Rust: Last on our list is Rust, which neither grew nor declined but instead held steady at #23. This may be disappointing for its more ardent fans, which include some high profile and highly accomplished technologists, but Rust's glacial ascent is relatively unsurprising. Targeting similar if lower level workloads than Go, a language itself that has plateaued in terms of its placement amongst these rankings, Rust suffers from the limits of a lower popularity ceiling while not receiving quite the same attention that Go did as a product of Google generally and people like Rob Pike specifically. By comparison, Rust's ascent has been much more workmanlike, winning its serious fans over one at a time. It's also worth noting that even if Rust never gets much beyond where it is today, it's still ranking higher than well known languages such as the aforementioned Clojure and Groovy, as well as CoffeeScript, Dart or Visual Basic. Not bad for a systems language.

September 2019

CSE341 Lecture 1.1

60

WHY THIS COURSE

September 2019

CSE341 Lecture 1.1

61

Why this Course?

Programming Language Concepts

- A language is a “conceptual universe” (Perlis)
 - Framework for problem-solving
 - Useful concepts and programming methods
- Understand the languages you use, by comparison
- Appreciate history, diversity of ideas in programming
- Be prepared for new programming methods, paradigms, tools

J. Mitchell

September 2019

CSE341 Lecture 1.1

62

Why this Course?

Critical thought

- Identify properties of language, not syntax or sales pitch
- Language and implementation

Every convenience has its cost

- Recognize the cost of presenting an abstract view of machine
- Understand trade-offs in programming language design

J. Mitchell

September 2019

CSE341 Lecture 1.1

63

Trends

Commercial trend over past 5+ years

- Increasing use of type-safe languages: Java, C#, ...
- Scripting languages, other languages for web applications

Teaching trends

- Java replaced C as most common intro language
 - Less emphasis on how data, control represented in machine
- Objective C

J. Mitchell

September 2019

CSE341 Lecture 1.1

64

Trends

Research and development trends

- **Modularity**
 - Java, C++: standardization of new module features
- **Program analysis**
 - Automated error detection, programming env, compilation
- **Isolation and security**
 - Sandboxing, language-based security, ...
- **Web 2.0**
 - Increasing client-side functionality, mashup isolation problems

J. Mitchell

September 2019

CSE341 Lecture 1.1

65

Example



What is D?

D is the culmination of decades of experience implementing compilers for many diverse languages and has a unique set of features:

- *high level* constructs for great modeling power
- *high performance*, compiled language
- static typing
- direct interface to the operating system API's and hardware
- blazingly fast compile-times
- memory-safe subset (SafeD)
- *maintainable, easy to understand* code
- gradual learning curve (C-like syntax, similar to Java and others)
- compatible with C application binary interface
- limited compatibility with C++ application binary interface
- multi-paradigm (imperative, structured, object oriented, generic, functional programming purity, and even assembly)
- built-in error detection (contracts, unittests)
- ... and many more [features](#).

<https://tour.dlang.org/>

September 2019

CSE341 Lecture 1.1

66

```
import std.stdio;
import std.algorithm;
import std.range;

void main()
{
    // Let's get going!
    writeln("Hello World!");

    // Take three arrays, and without allocating any new
    // memory, sort across all the arrays inplace
    int[] arr1 = [4, 9, 7];
    int[] arr2 = [5, 2, 1, 10];
    int[] arr3 = [6, 8, 3];
    sort(chain(arr1, arr2, arr3));
    writeln("%s\n%s\n%s\n", arr1, arr2, arr3);
}
```

Example



Major Design Goals of D

- Enable writing fast, effective code in a straightforward manner.
- Make it easier to write code that is portable from compiler to compiler, machine to machine, and operating system to operating system. Eliminate undefined and implementation defined behaviors as much as practical.
- Provide syntactic and semantic constructs that eliminate or at least reduce common mistakes. Reduce or even eliminate the need for third party static code checkers.

<https://dlang.org/overview.html#goals>

September 2019

CSE341 Lecture 1.1

67

Example



Major Design Goals of D

- Support memory safe programming.
- Support multi-paradigm programming, i.e. at a minimum support imperative, structured, object oriented, generic and even functional programming paradigms.
- Make doing things the right way easier than the wrong way.
- Have a short learning curve for programmers comfortable with programming in C, C++ or Java.

<https://dlang.org/overview.html#goals>

September 2019

CSE341 Lecture 1.1

68

Example



Major Design Goals of D

- Provide low level bare metal access as required. Provide a means for the advanced programmer to escape checking as necessary.
- Be compatible with the local C application binary interface.
- Where D code looks the same as C code, have it either behave the same or issue an error.
- Have a context-free grammar. Successful parsing must not require semantic analysis.
- Easily support writing internationalized applications - Unicode everywhere.

<https://dlang.org/overview.html#goals>

September 2019

CSE341 Lecture 1.1

69

Example



Major Design Goals of D

- Incorporate Contract Programming and unit testing methodology.
- Be able to build lightweight, standalone programs.
- Reduce the costs of creating documentation.
- Provide sufficient semantics to enable advances in compiler optimization technology.
- Cater to the needs of numerical analysis programmers.
- Obviously, sometimes these goals will conflict. Resolution will be in favor of usability.

<https://dlang.org/overview.html#goals>

September 2019

CSE341 Lecture 1.1

70

Example



Object Oriented Programming

- **Classes**
D's object oriented nature comes from classes. The inheritance model is single inheritance enhanced with interfaces. The class Object sits at the root of the inheritance hierarchy, so all classes implement a common set of functionality. Classes are instantiated by reference, and so complex code to clean up after exceptions is not required.
- **Operator Overloading**
Classes can be crafted that work with existing operators to extend the type system to support new types. An example would be creating a bignumber class and then overloading the +, -, * and / operators to enable using ordinary algebraic syntax with them.

<https://dlang.org/overview.html#goals>

September 2019

CSE341 Lecture 1.1

71

Example



Functional Programming

Functional programming has a lot to offer in terms of encapsulation, concurrent programming, memory safety, and composition. D's support for functional style programming include:

- Pure functions
- Immutable types and data structures
- Lambda functions and closures


<https://dlang.org/overview.html#goals>

September 2019

CSE341 Lecture 1.1

72

Example



Productivity

- **Modules**
 - Source files have a one-to-one correspondence with modules.
- **Declaration vs Definition**
 - Functions and classes are defined once. There is no need for declarations when they are forward referenced. A module can be imported, and all its public declarations become available to the importer.

<https://dlang.org/overview.html#goals>

September 2019

CSE341 Lecture 1.1

73

Example



Resource Management

- **Automatic Memory Management**

D memory allocation is fully garbage collected. With garbage collection, programming gets much simpler. Garbage collection eliminates the need for tedious, error prone memory allocation tracking code. This not only means much faster development time and lower maintenance costs, but the resulting program frequently runs faster.
- **Explicit Memory Management**

Despite D being a garbage collected language, the new and delete operations can be overridden for particular classes so that a custom allocator can be used.
- **RAII**

RAII is a modern software development technique to manage resource allocation and deallocation. D supports RAII in a controlled, predictable manner that is independent of the garbage collection cycle.

<https://dlang.org/overview.html#goals>

September 2019

CSE341 Lecture 1.1

74

Example



Performance

- **Lightweight Aggregates**

D supports simple C style structs, both for compatibility with C data structures and because they're useful when the full power of classes is overkill.
- **Inline Assembler**

Device drivers, high performance system applications, embedded systems, and specialized code sometimes need to dip into assembly language to get the job done. While D implementations are not required to implement the inline assembler, it is defined and part of the language. Most assembly code needs can be handled with it, obviating the need for separate assemblers or DLLs.
- **Many D implementations will also support intrinsic functions**

analogously to C's support of intrinsics for I/O port manipulation, direct access to special floating point operations, etc.

<https://dlang.org/overview.html#goals>

September 2019

CSE341 Lecture 1.1

75

Example



- **Reliability**

A modern language should do all it can to help the programmer flush out bugs in the code. Help can come in many forms; from making it easy to use more robust techniques, to compiler flagging of obviously incorrect code, to runtime checking.
- **Contracts**

Contract Programming (invented by Dr. Bertrand Meyer) is a technique to aid in ensuring the correctness of programs. D's version of Contracts includes function preconditions, function postconditions, class invariants, and assert contracts. See Contracts for D's implementation.
- **Unit Tests**

Unit tests can be added to a class, such that they are automatically run upon program startup. This aids in verifying, in every build, that class implementations weren't inadvertently broken. The unit tests form part of the source code for a class. Creating them becomes a natural part of the class development process, as opposed to throwing the finished code over the wall to the testing group. Unit tests can be done in other languages, but the result is kludgy and the languages just aren't accommodating of the concept. Unit testing is a main feature of D. For library functions it works out great, serving both to guarantee that the functions actually work and to illustrate how to use the functions. Consider the many library and application code bases out there for download on the web. How much of it comes with any verification tests at all, let alone unit testing? The usual practice is if it compiles, we assume it works. And we wonder if the warnings the compiler spits out in the process are real bugs or just nattering about nits. Along with Contract Programming, unit testing makes D far and away the best language for writing reliable, robust systems applications. Unit testing also gives us a quick-and-dirty estimate of the quality of some unknown piece of D code dropped in our laps - if it has no unit tests and no contracts, it's unacceptable.
- **Debug Attributes and Statements**

Now debug is part of the syntax of the language. The code can be enabled or disabled at compile time, without the use of macros or preprocessing commands. The debug syntax enables a consistent, portable, and understandable recognition that real source code needs to be able to generate both debug compilations and release compilations.
- **Exception Handling**

The superior try-catch-finally model is used rather than just try-catch. There's no need to create dummy objects just to have the destructor implement the finally semantics.
- **Synchronization**

Multithreaded programming is becoming more and more mainstream, and D provides primitives to build multithreaded programs with. Synchronization can be done at either the method or the object level.

<https://dlang.org/overview.html#goals>

September 2019

CSE341 Lecture 1.1

76

Example



Project Management

Versioning

D provides built-in support for generation of multiple versions of a program from the same text. It replaces the C preprocessor `#if/#endif` technique.

Deprecation

As code evolves over time, some old library code gets replaced with newer, better versions. The old versions must be available to support legacy code, but they can be marked as deprecated. Code that uses deprecated versions will be normally flagged as illegal, but would be allowed by a compiler switch. This will make it easy for maintenance programmers to identify any dependence on deprecated features.

<https://dlang.org/overview.html#goals>

September 2019

CSE341 Lecture 1.1

77

Example



Support for Robust Techniques
 Dynamic arrays instead of pointers
 Reference variables instead of pointers
 Reference objects instead of pointers
 Garbage collection instead of explicit memory management
 Built-in primitives for thread synchronization
 No macros to inadvertently slam code
 Inline functions instead of macros
 Vastly reduced need for pointers
 Integral type sizes are explicit
 No more uncertainty about the signed-ness of chars
 No need to duplicate declarations in source and header files.
 Explicit parsing support for adding in debug code.
 Compile Time Checks
 Stronger type checking
 No empty ; for loop bodies
 Assignments do not yield boolean results
 Deprecating of obsolete APIs
 Runtime Checking
 assert() expressions
 array bounds checking
 undefined case in switch exception
 out of memory exception
 In, out, and class invariant Contract Programming support

<https://dlang.org/overview.html#goals>

September 2019

CSE341 Lecture 1.1

78

Example



Resource Management

- Automatic Memory Management
- D memory allocation is fully garbage collected. With garbage collection, programming gets much simpler. Garbage collection eliminates the need for tedious, error prone memory allocation tracking code. This not only means much faster development time and lower maintenance costs, but the resulting program frequently runs faster.
- For a fuller discussion of this, see garbage collection.
- Explicit Memory Management
- Despite D being a garbage collected language, the new and delete operations can be overridden for particular classes so that a custom allocator can be used.
- RAII
- RAII is a modern software development technique to manage resource allocation and deallocation. D supports RAII in a controlled, predictable manner that is independent of the garbage collection cycle.

<https://dlang.org/overview.html#goals>

September 2019

CSE341 Lecture 1.1

79

Tentative Schedule

Week 1: Introduction

Weeks 2-4: Lexical and Syntax Analysis

Week 5: Names, Bindings and Scope

Week 6: Data Types

Week 7: Expressions and Assignments

Week 8: Control and Subprograms

Week 9: Functional Programming

Week 10: Encapsulation

Week 11&12: Object Oriented

Week 13: Exception Handling & Concurrency

Week 14: Logic Programming



September 2019

CSE341 Lecture 1.1

80

Any Questions or Comments?