EPFL, CIVIL-127

Programming and software development for engineers

in-depth evaluation

- Log onto moodle and stay on the moodle home page (dashboard, not the course page).
- Click on the arrow to the top right of the screen which will reveal a block that contains the entitled “In-depth evaluation” tile (please note: all evaluations will be together in the evaluation tile on the moodle home page, and not separate in each course moodle page).
- Select your course and complete the feedback.

Let's build GPT: from scratch, in code, spelled out.

- The best introduction you could ask for, by Andrej Karpathy
- https://www.youtube.com/watch?v=kCc8FmEb1nY
  - 2h video
  - Python code

FizzBuzz

- Write a program which counts from 1 to 100 and prints
  - Fizz Buzz when i is a multiple of both, 3 and 5
  - Fizz if i is a multiple of 3 but not of 5
  - Buzz if i is a multiple of 5 but not of 3
  - i otherwise
- Use timeit to see how fast your implementation runs

v1

```
import timeit
import sys

def fizzbuzz(n):
   for i in range(1, n+1):
       if i % 3 == 0 and i % 5 == 0:
           print("Fizz Buzz")
       elif i % 3 == 0:
           print("Fizz")
       elif i % 5 == 0:
           print("Buzz")
       else:
           print(i)

result = timeit.timeit(lambda: fizzbuzz(100),
number=100000)
print(result, file=sys.stderr)
```

Time the first implementation:

```
$ python3 v1.py > /dev/null
2.581426207907498
```

v1

```
import timeit
import sys

def fizzbuzz(n):
   for i in range(1, n+1):
       if i % 3 == 0 and i % 5 == 0:
           print("Fizz Buzz")
       elif i % 3 == 0:
           print("Fizz")
       elif i % 5 == 0:
           print("Buzz")
       else:
           print(i)

result = timeit.timeit(lambda: fizzbuzz(100),
number=100000)
print(result, file=sys.stderr)
```

v2

```
import timeit
import sys

def fizzbuzz(n):
   j = 1
   for i in range(1, n+1):
       if j == 3 or j == 6 or j == 9 or j == 12:
           print("Fizz")
       elif j == 5 or j == 10:
           print("Buzz")
       elif j == 15:
           print("Fizz Buzz")
           j = 0
       else:
           print(i)
       j = j + 1

result = timeit.timeit(lambda: fizzbuzz(100), number=100000)
print(result, file=sys.stderr)
```

Division is expensive, right?

```
import timeit
import sys

def fizzbuzz(n):
   j = 1
   for i in range(1, n+1):
       if j == 3 or j == 6 or j == 9 or j == 12:
           print("Fizz")
       elif j == 5 or j == 10:
           print("Buzz")
       elif j == 15:
           print("Fizz Buzz")
           j = 0
       else:
           print(i)
       j = j + 1

result = timeit.timeit(lambda: fizzbuzz(100), number=100000)
print(result, file=sys.stderr)
```

Let's make sure both version give the same result:

```
$ python3 v1.py > v1.out
$ python3 v2.py > v2.out
$ diff v1.out v2.out
```

Time the second implementation:

```
$ python3 v2.py > /dev/null
2.7923777499236166
```

It's slower!

Why is v2.py slower than v1.py?

v1

```
import timeit
import sys

def fizzbuzz(n):
   for i in range(1, n+1):
       if i % 3 == 0 and i % 5 == 0:
           print("Fizz Buzz")
       elif i % 3 == 0:
           print("Fizz")
       elif i % 5 == 0:
           print("Buzz")
       else:
           print(i)

result = timeit.timeit(lambda: fizzbuzz(100),
number=100000)
print(result, file=sys.stderr)
```

v3

```
import timeit
import sys

def fizzbuzz(n):
   for i in range(1, n+1):
       if i % 3 == 0:
           if i % 5 == 0:
               print("Fizz Buzz")
           else:
               print("Fizz")
       elif i % 5 == 0:
           print("Buzz")
       else:
           print(i)

result = timeit.timeit(lambda: fizzbuzz(100),
number=100000)
print(result, file=sys.stderr)
```

Third implementation

```
import timeit
import sys

def fizzbuzz(n):
   for i in range(1, n+1):
       if i % 3 == 0:
           if i % 5 == 0:
               print("Fizz Buzz")
           else:
               print("Fizz")
       elif i % 5 == 0:
           print("Buzz")
       else:
           print(i)

result = timeit.timeit(lambda: fizzbuzz(100),
number=100000)
print(result, file=sys.stderr)
```

Again, first check that the result is the same:

```
$ python3 v3.py > v3.out
$ diff v1.out v3.out
```

Time the third implementation:

```
$ python3 v3.py > /dev/null
2.3346250830218196
```

It's faster...

Making programs run faster is hard

- Need to fully understand what's going on
  - How the processors works (hint: there a many processors, each with many cores)
  - How the memory and caches work (hint: there are layers of caches)
  - How the operating system works
  - How the Python compiler/runtime works (hint: there's a GIL)
  - How statistics work (hint: our differences might have been within the noise margin)
- Usually, optimized code is harder to write, read, and maintain
  - Write the simplest and correct implementation first
  - Only optimize once you know your bottleneck or if you don't have other options
  - 99% of the time, you don't need to over-optimize – computers are fast and typical applications don't process huge amounts of data

Really fast FizzBuzz

- https://codegolf.stackexchange.com/questions/215216/high-throughput-fizz-buzz/269772#269772

American Standard Code for Information Interchange (ASCII)

The **hexadecimal** set:

00	nul	01	soh	02	stx	03	etx	04	eot	05	enq	06	ack	07	bel
08	bs	09	ht	0a	nl	0b	vt	0c	np	0d	cr	0e	so	0f	si
10	dle	11	dc1	12	dc2	13	dc3	14	dc4	15	nak	16	syn	17	etb
18	can	19	em	1a	sub	1b	esc	1c	fs	1d	gs	1e	rs	1f	us
20	sp	21	!	22	"	23	#	24	$	25	%	26	&	27	'
28	(	29	)	2a	*	2b	+	2c	,	2d	-	2e	.	2f	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3a	:	3b	;	3c	<	3d	=	3e	>	3f	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4a	J	4b	K	4c	L	4d	M	4e	N	4f	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5a	Z	5b	[	5c	\	5d	]	5e	^	5f	_
60	`	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6a	j	6b	k	6c	l	6d	m	6e	n	6f	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7a	z	7b	{	7c	|	7d	}	7e	~	7f	del

- Standardized in 1960s
- 128 symbols (0x00 to 0x7f)
- 1 byte per symbols
- Several special characters
- A-Z
- a-z
- 0-9
- a few common symbols

Unicode

- Process began in early 1990s
- Today: 292531 codepoints (with room to expand ~4x)
- Support for almost all languages
  - Bonjour
  - 你好
  - مرحبًا
  - سلام
  - नमस्ते
  - こんにちは
- Emojis
  - 😂❤️👍👻

Ligatures and combining characters

- Font rendering engines can combine multiple characters
- Aesthetics
  - f + i => fi
- Skin-tones
  - 👍, 👍🏻, 👍🏼, 👍🏽, 👍🏾, 👍🏿
- Flags
  - 🇨 + 🇭 => 🇨🇭

Unicode doesn’t define the graphic

- The font defines the exact look for each character
- As a result, conversation across different devices can become confusing:

  “Come and bring your 🫗” (newer Samsung phones)
  “Come and bring your 🔫“ (older Samsung phones)

  “😬“ (older iPhone)
  “😁“ (every other device)

Issues

- Supporting so many characters introduces hard problems
- Equivalent characters
  - e + ◌́ (é) is equivalent to é
- Identical looking characters
  - e (latin e) looks similar/same as е (cyrillic e)
- Sorting
  - é does not come after z (hopefully)
- Searching
- Password handling
- …

Wikipedia

wikipedia.org

Finish update

WIKIPEDIA

The Free Encyclopedia

English
6,974,000+ articles

日本語
1,457,000+ 記事

Deutsch
3.001.000+ Artikel

中文
1,470,000+ 条目 / 條目

Português
1.146.000+ artigos

Read Wikipedia in your langua

Lietuvių
Magyar
Македонски
مصرى
Bahasa Melayu
Bahaso Minangkabau
မြန်မာဘာသာ
Nederlands
日本語
Norsk (bokmål)
Norsk (nynorsk)
Нохчийн
Oʻzbekcha / Ўзбекча
Polski
Português
Қазақша / Qazaqşa / قازاقشا
Română
Shqip
Simple English
Sinugboanong Binisaya
Slovenčina
Slovenščina
Српски / Srpski
Srpskohrvatski / Српскохрватски
Suomi
Svenska
தமிழ்
Татарча / Tatarça
తెలుగు
ภาษาไทย
Тоҷикӣ
تۆرکجه
Türkçe
Українська
اردو
Tiếng Việt
Winaray
中文
Русский
粵語

Wikipedia is hosted by the Wikimedia Foundation, a non-profit organization that also hosts a range of other projects.
You can support our work with a donation.

Download Wikipedia for Android or iOS

Save your favorite articles to read offline,

Commons
Free media collection

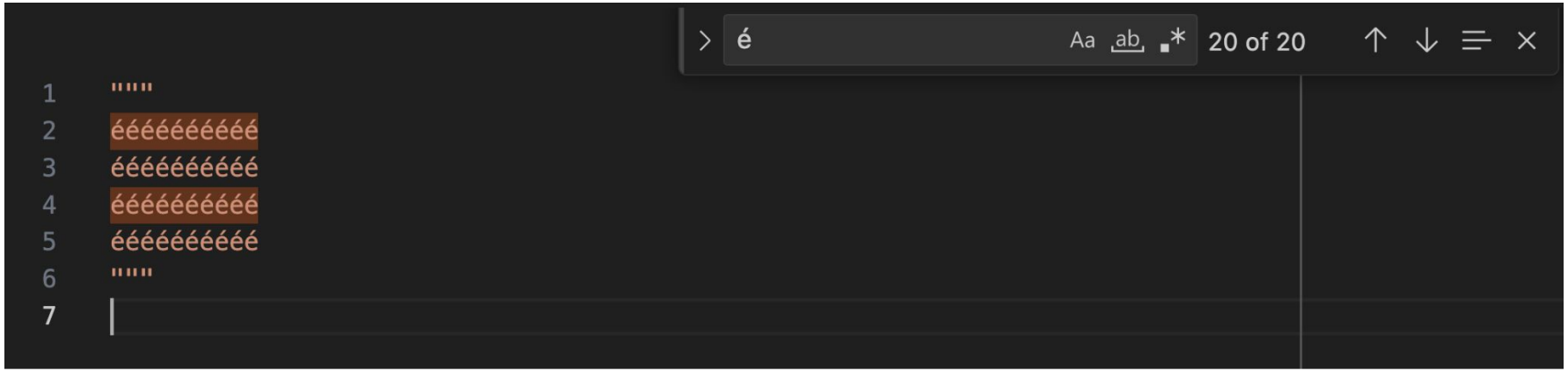
Wikibooks
Free textbooks

Wiktionary
Free dictionary

Wikidata
Free knowledge base

Search

- VSCode does not canonicalize Unicode characters
- In this example, we only find 20 out of the 40 occurrences of é



Abusing Unicode!

```
e = 10
if True:
   e = 123
print(e) # prints 10
```

- This code does not do what you think it does!
- Should programming languages forbid using Unicode characters in source code? In variable names? Only allow a subset of Unicode?
- You can inspect what's actually going on using `ord()`, a hex-editor, or enabling features in VSCode to draw boxes around non-ascii characters.

UTF-8

- Encodes Unicode using a variable number of bytes (1 to 4 bytes)
- 1-byte UTF-8 coincides with ASCII (what were the odds!)

Code point ↔ UTF-8 conversion

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0yyyzzzz			
U+0080	U+07FF	110xxxyy	10yyzzzz		
U+0800	U+FFFF	1110wwww	10xxxxyy	10yyzzzz	
U+010000	U+10FFFF	11110uvv	10vvwwww	10xxxxyy	10yyzzzz

Marble solitaire puzzle

Image source: Amazon product listing, maybe a copyright violation ¯_(ツ)_/¯

Marble solitaire puzzle

- Rules
  - Goal is to be left with one marble
  - A marble can capture up, down, left, right by hopping over one other marble
  - O O . => . . O
- Try it online
- Can you write a computer program to find a solution?

Marble solitaire puzzle: implementation sketch

```
class Board:
   ...

   def solve(self, depth) -> bool:
       if self.filled == 81:
           # We have one marble left, we are done
           return []
       moves = self.get_valid_moves()
       for move in moves:
           self.apply_move(move)
           t = solve(self, depth+1)
           if t is not None:
               # We have found a solution
               t.append(move)
               return t
           self.undo_move(move)
       # We don't have a solution
       return None
```

- Recursive solution
- We apply a move and then undo it if it doesn’t yield a solution

Pentominoes & co.

- Once you can write a marble solver, you can write a pentominoes solver using the exact same strategy!

Given a list of numbers reach a sum

- E.g. [4, 5, 17, 9] + 144
  - (4 + 5) * 17 - 9
- You can solve this problem in the exact same way
- You recursively try to build different trees

Sudoku

9	8	5	4		1			
				3				
1		6						
			5					
4		2			9			3
	9			6	3	4		
	6			1				
			3		6			5
2				8				1

- Fill numbers from 1-9 so that there are no duplicates among each row, column, and smaller 3x3 squares
- Can you write a computer program to find a solution?

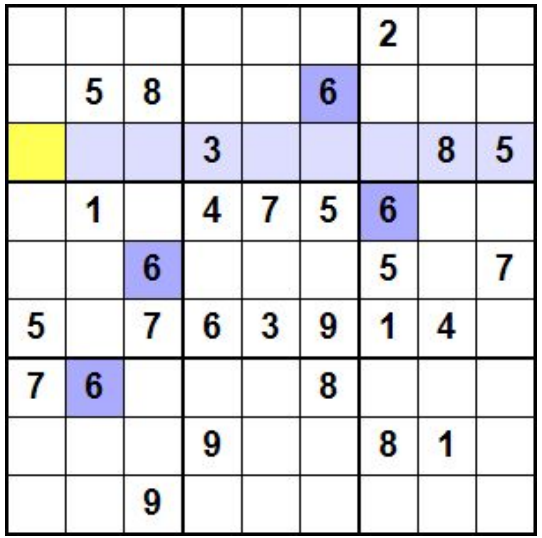
Sudoku

- “Easy” puzzles can be solved by only solving for “naked singles”
- Naked single == cell which can only take one value

		3				8		5
	1		6	4				9
9	7		2	5				
						9		
1			4	7	2			6
		8						
				2	9		5	4
2				1	4		3	
4		1				7		

Sudoku

- “Hard” puzzles can be solved by solving for “naked singles” + “hidden singles”
- Hidden single == value which can only be assigned to a single cell



						2		
	5	8			6			
			3				8	5
	1		4	7	5	6		
		6				5		7
5		7	6	3	9	1	4	
7	6				8			
			9			8	1	
		9						

Sudoku

- “Very hard” puzzles require making a guess and then backtracking if the guess is incorrect

Sudoku: data structures

- We need a board, which holds 9x9 cells
- To solve for naked singles:
  - Each cell starts with a set of possible numbers
  - When a cell gets a value *v*, all the other cells on the same row, column, and smaller square must remove *v* from their set
  - If a cell only has one possible number left, it takes that value
- To solve for hidden singles:
  - We can try to create a data structure to keep track of hidden singles, but keeping the data structure up-to-date can be complicated
  - It's probably easiest to scan the rows, columns and inner squares for numbers where only one possibility appears (243 scans, but there's opportunities to early exit)

Sudoku: implementation sketch

```
class Board:
   ...

   def solve(self) -> bool:
       if self.filled == 81:
           # We are done
           return self
       for c in self.cells():
           if len(c.choices) > 1:
               for choice in list(c.choices):
                   self.pick(c, choice)
                   t = board2.solve()
                   if t is not None:
                       return t

                   # WE CAN'T UNDO!
       return None
```

- Recursive solution
- But we can’t undo!

Sudoku: implementation sketch

```
class Board:
   ...

   def solve(self) -> bool:
       if self.filled == 81:
           # We are done
           return self
       for c in self.cells():
           if len(c.choices) > 1:
               for choice in list(c.choices):
                   board2 = self.copy()
                   board2.pick(c, choice)
                   t = board2.solve()
                   if t is not None:
                       return t
       return None
```

- We can copy our entire state prior to recursing

Sudoku: alternative implementation

- Z3 is a powerful SMT solver (a SMT solver is a superset of a SAT solver)
- See 4th solution to exercise 2.3

Regular Expressions (regexp)

- Patterns used to find/replace text
- Regular expressions work at the character level

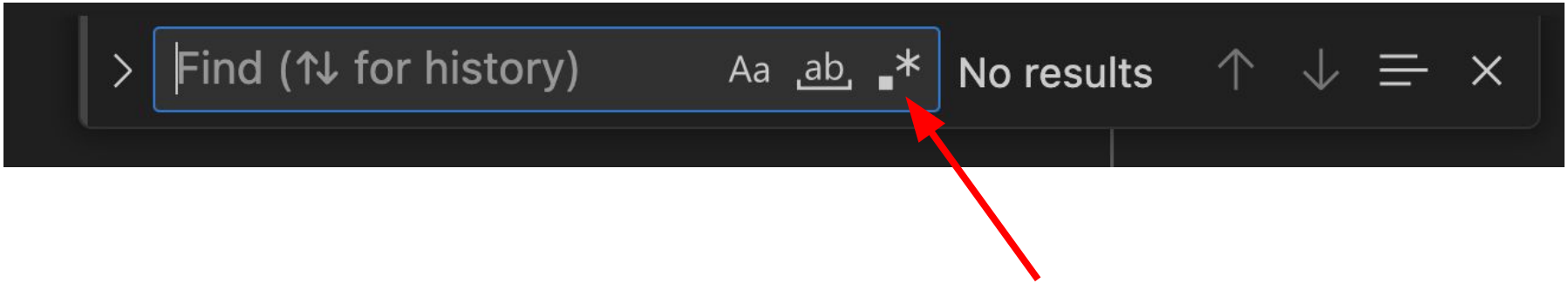
Regular Expressions (regexp)

- `.` wildcard, match any character
- Repetition (greedy by default)
  - `+` match previous character one or more times
  - `*` match previous character zero or more times
  - `?` match previous character zero or one time
  - `{min, max}`where *min* and *min* are numbers
- `[...]` set of characters or ranges to match
  - `[ax2]` will match a, x, or 2
  - `[a-m0-9]` will match characters in the range a to m or 0 to 9
- `[^...]` set of characters or ranges to not match
  - `[^ax2]` will match everything except a, x, or 2
  - `[^A-Z]` will match everything except the range A to Z
- `(...)` grouping
  - `\n` enables matching previously matched groups, where *n* is a number
- `|` or operator
- `^` and `$` anchor to start and end of line (or text)
- `\.` to match an actual dot (`[.]` also works)
- And more (read the docs)

Regular Expressions

- Examples
  - `[hc]?at` will match "at", "cat", "hat". It will also match "sat"
  - `f.+bar` will match "xyzfobar", "foobar", "fooobar" but not "fbar"
  - `[A-Z][a-z]*` will match "Hi", "Hello" but not "hello"
  - `(.)(.)\2\1` will match "abba" and "aaaa" but not "xyxy"
  - `(foo)|(bar)` will match "foo" and "bar"
  - `(foo)|bar` will match "fooar" and "bar" but not "foo"
- Regular expressions can be hard to decipher, make sure you comment them

Regular Expressions: in VSCode



Regular Expressions: command line

- grep, egrep
- sed

Software-engineering(*) related methodologies

- At the individual level
  - Pomodoro
  - Getting Things Done (GTD)
  - How to email
- At the team level
  - Agile / Scrum
  - Waterfall
  - Lean / Kanban
  - ...

* also used in other engineering fields

Technical interviews

- Coding or problem solving puzzles
- Multiple phone screens
  - Coding interviews using coderpad or similar tool
- Multiple on-site interviews
  - White board coding
  - Design questions
  - Q&A sessions
- The technical interview is often similar across software engineering, data analysts, ML, and product/project managers
- Candidates are usually allowed to pick their preferred programming language, the interviewer has to adapt
- Use books, online resources, and mock interviews to practice

Coding questions types

- Simple
  - Find first duplicate character in a file
  - Find or build palindromes
  - Combinatorics (e.g. all triplets which sum to N) or largest subsets
- Medium
  - Merge two sorted lists
  - Do something with trees or graphs in general
  - Min edit distance of two strings
  - Football scores and other DP problems
- Hard
  - Marble solver
  - Bignum library
  - Mini regular expression matcher
  - m-th smallest value in k sorted arrays

Interview Tips

- Write clean, readable code. Even if it's just an interview, pretend it's code that will need to be maintained
- Communicate. Explain your assumptions, your thought process, …
- Practice with books, online resources, friends, mock interviews
- Be comfortable with the data structures and algorithms which come up in your field
- Get to a (partial or complete) solution first, then try to improve it
- Be honest, ask questions. Don't try to outsmart your interviewers
- When in doubt, companies lean towards no-hire, so apply to lots of companies

Gain coding experience

- Contribute to your favorite open source projects
- Advent of code: speed coding, easy to medium difficulty, with an emphasis on algorithms – lots of discussions and solutions available on reddit and github
- TopCoder and CodeJam: hard problems, emphasis on algorithms – some of the solutions might not be easy to find
- Project Euler: usually, difficult math problems
- Leet Code: organized by topics
- Code Golf: size optimization
- r/dailyprogrammer
- Cryptopals: hard, cryptography related puzzles
- Olympiads: hard. Google a country's local chapter for sample questions
- ICFP Programming Contest: very hard (borderline research topics)
- And lots of other sites...

These puzzles often get used as coding interview questions