EPFL, CIVIL-127

Programming and software development for engineers

Teachers

- Prof. Alexandre Alahi
- Alok Menghrajani (unfortunately, absent today)

Questions, comments, feedback:

- Forum
- alok.menghrajani@epfl.ch or alexandre.alahi@epfl.ch
- Teaching Assistants

Purpose of this course

Course content

- Tools used by software engineers
- Python
- Software engineering techniques

Learning objectives

- Apply industry-standard best practices in Python programming.
- Develop clean, testable, and easily reviewable code.
- Test, debug, and fix bugs in a systematic and efficient manner.

Organization

- 45min weekly lectures
- 1h30 weekly labs
- ~3h weekly homework
- 12 sessions in a semester

Grading and due dates

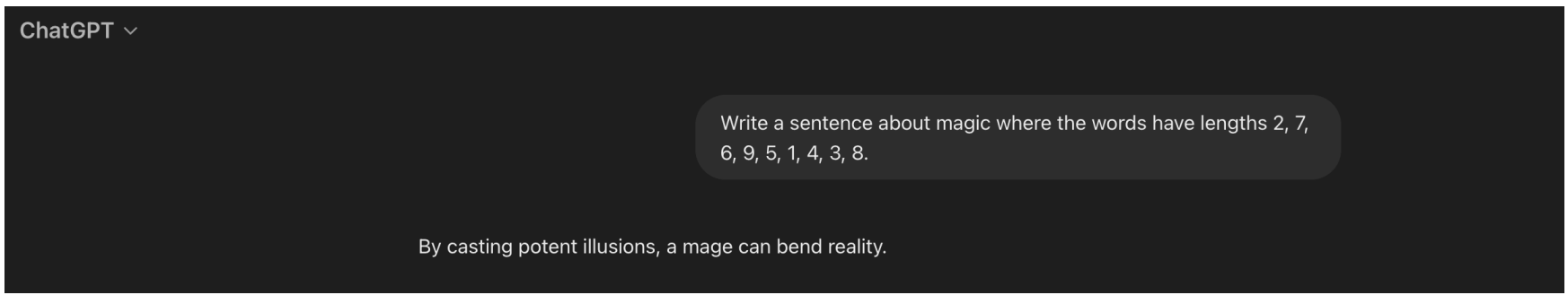
- 4th lab is graded (due on March 14th, 2025)
- 6th lab can earn you a bonus (due on March 28th, 2025)
- MCQ mid-term (on April 15th, 2025)
- Project (due on May 23rd, 2025)

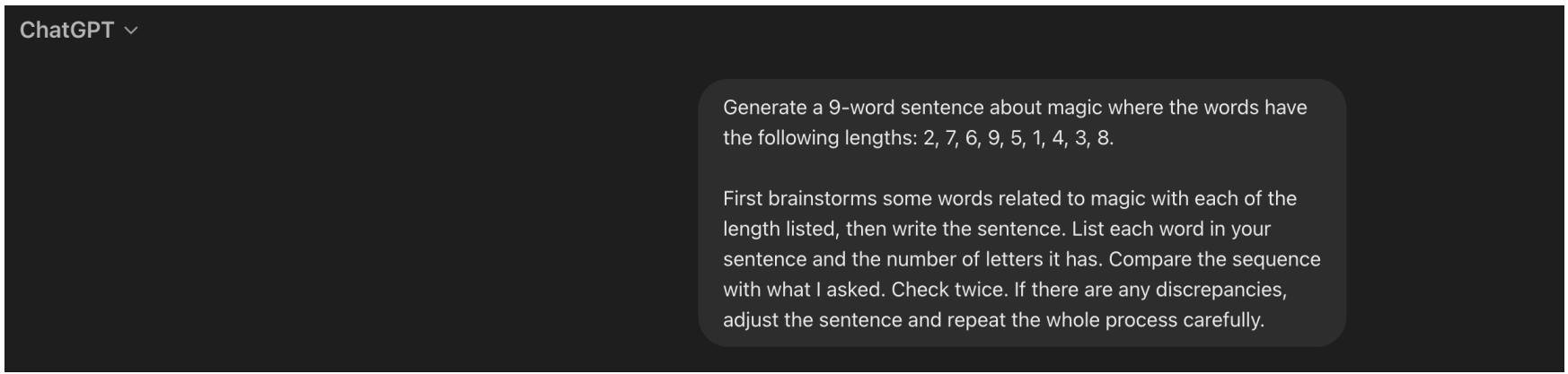
Today's AI tools

- You are allowed to use them…
- …our recommendation: first try to solve programming assignments without any AI tool and then see if or how an AI tool would have helped.

Today's AI tools

- Sometimes wrong
- Sometimes wrong in subtle ways
- Writing good prompts is a skill
- Can be a time sink


ChatGPT
Write a sentence about magic where the words have lengths 2, 7, 6, 9, 5, 1, 4, 3, 8.
By casting potent illusions, a mage can bend reality.


ChatGPT
Generate a 9-word sentence about magic where the words have the following lengths: 2, 7, 6, 9, 5, 1, 4, 3, 8.
First brainstorms some words related to magic with each of the length listed, then write the sentence. List each word in your sentence and the number of letters it has. Compare the sequence with what I asked. Check twice. If there are any discrepancies, adjust the sentence and repeat the whole process carefully.

> “I lost about an hour today because Cursor very confidently led me down a refactoring path based on a completely hallucinated (but reasonable-seeming!) understanding of json serialization in python.”

— Senior engineer on Jan 8th, 2025

Tomorrow’s AI tools

???

Learn by watching

- Read other people's code
- Watch other people write code
- Watch other people debug an issue

Learn by doing

- Labs and projects
- Optional additional programming assignments

Install Visual Studio Code and Python

- Make sure you have the correct Python version
- The first lab will help ensure your environment is setup correctly

Sokoban

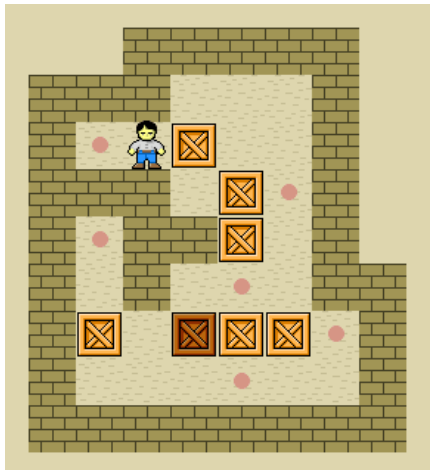


image by Carloseow

from https://en.wikipedia.org/wiki/Sokoban

- Puzzle game
- Implementation over first 6 labs
- We'll use pygame to build the graphical user interface

Python

Whitespace

Remember, in Python, whitespaces are important! They define each block's scope.

```
a = 1
if a == 2:
    print("foo")
print("bar")
```

Result

```
bar
```

Whitespace

Remember, in Python, whitespaces are important! They define each block's scope.

```
a = 1
if a == 2:
    print("foo")
    print("bar")
```

Result

For loops

```
for i in range(10):
    print(i)
```

Result:

```
0
1
2
3
4
5
6
7
8
9
```

For loops

```
for i in range(5, 10):
    print(i)
```

Result:

```
5
6
7
8
9
```

For loops

```
1    for i in range(5, 10, 3):
2        print(i)
```

Result:

```
5
8
```

For loops

```
for i in range(10, 5, -1):
    print(i)
```

Result:

```
10
9
8
7
6
```

For loops

```
for i in range(5, 10, -1):
    print(i)
```

Result:

For loops

```
for i in range(0, 10):
    if i % 3 == 0:
        continue
    if i % 7 == 0:
        break
    print(i)
```

Result:

```
1
2
4
5
```

Strings

- `"Hello Joe"` *double quotes*
- `'Hello Joe'` *single quote*
- `"Hello " + "Joe"` *concatenation*
- `"Hello {}".format("Joe")` *String's* `format()` *method*
- `"Hello {name}".format(name="Joe")` `format()` *method with names*
- `"Hello%s" % "Joe"` *printf-style*

User input

input() is a built-in function

```
1    name = input("What's your name? ")
2    print("Hello {}!".format(name))
```

Result:

```
What's your name? Joe
Hello Joe!
```

Reading data from a file

```
1    with open("poem.txt") as lines:
2        for line in lines:
3            print("> {}".format(line.rstrip("\r\n")))
```

Result:

```
> Blueprints of pure thought,
> crafting worlds from logic's thread,
> zeros bloom to life.
```

Lists

see Lists, Common Sequence Operations, and Mutable Sequence Types

```
a = [1, 2, 3]
print("1:", a)
a.append(4)
print("2:", len(a))
x = a[2]
print("3:", x)
b = [0] * 10
print("4:", b)
```

Result:

```
1: [1, 2, 3]
2: 4
3: 3
4: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Tuples

- Tuples are immutable lists
- Subset of lists API

```
x = (1, "foo")
print("1:", len(x))
print("2:", x[0])
print("3:", x[1])
(a, b) = x
print("4:", a)
print("5:", b)
y = 2, "bar"
print("6:", y)
c, d = x
print("7:", c, d)
```

Result:

```
1: 2
2: 1
3: foo
4: 1
5: foo
6: (2, 'bar')
7: 1 foo
```

Dictionnaries

- `dict()` or `{}`
- key-value data structure
- `d = {"foo": 123, "bar": 567}`
- iteration order is guaranteed to be insertion-order
- keys can be anything (int, tuple, strings, etc.)

Dictionnaries

```python
d = {"foo": 123, "bar": 567}

for k in d:
    print(k, d[k])
```

Result:

```
foo 123
bar 567
```

Dictionnaries

```python
d = {"foo": 123, "bar": 567}

for (k, v) in d.items():
    print(k, v)
```

Result:

```
foo 123
bar 567
```

Dictionnaries

```
d = {"foo": 123, "bar": 567}

for i, (k, v) in enumerate(d.items()):
    print(i, k, v)
```

Result:

```
0 foo 123
1 bar 567
```

List comprehension

```
y = [x * 2 for x in range(5)]
print(y)
```

Result:

```
[0, 2, 4, 6, 8]
```

List comprehension

Equivalent

```
y = []
for x in range(5):
    y.append(x * 2)
print(y)
```

Result:

```
[0, 2, 4, 6, 8]
```

2D arrays: plain list

```
WIDTH, HEIGHT = (5, 4)
grid = [0] * (WIDTH * HEIGHT)

def offset(x, y):
    if x < 0 or x >= WIDTH or y < 0 or y >= HEIGHT:
        raise Exception("invalid offset", x, y)
    return x * HEIGHT + y

def get(x, y):
    return grid[offset(x, y)]

def set(x, y, v):
    grid[offset(x, y)] = v

def printGrid():
    for y in range(HEIGHT):
        for x in range(WIDTH):
            print(get(x, y), end=" ")
        print()

set(4, 3, "A")
set(1, 2, "B")
printGrid()
```

2D arrays: plain list

Result:

```
0 0 0 0 0
0 0 0 0 0
0 B 0 0 0
0 0 0 0 A
```

2D arrays: list of lists

```
grid = [[0] * 4] * 5
print(grid)
```

Result:

```
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

2D arrays: list of lists

```
WIDTH, HEIGHT = (5, 4)
grid = [[0] * WIDTH] * HEIGHT

def get(x, y):
    return grid[y][x]

def set(x, y, v):
    grid[y][x] = v

def printGrid():
    for y in range(HEIGHT):
        for x in range(WIDTH):
            print(get(x, y), end=" ")
        print()

set(4, 3, "A")
set(1, 2, "B")
printGrid()
```

Don't do this! ⚠️

2D arrays: list of lists

Result:

```
0 B 0 0 A
0 B 0 0 A
0 B 0 0 A
0 B 0 0 A
```

We have an incorrect behavior.

2D arrays: list of lists

```
WIDTH, HEIGHT = (5, 4)
grid = [[0] * WIDTH for _ in range(HEIGHT)]

def get(x, y):
    return grid[y][x]

def set(x, y, v):
    grid[y][x] = v

def printGrid():
    for y in range(HEIGHT):
        for x in range(WIDTH):
            print(get(x, y), end=" ")
        print()

set(4, 3, "A")
set(1, 2, "B")
printGrid()
```

2D arrays: list of lists

Result:

```
0 0 0 0 0
0 0 0 0 0
0 B 0 0 0
0 0 0 0 A
```

2D arrays: dict

```
WIDTH, HEIGHT = (5, 4)
grid = {}

def get(x, y):
    if (x, y) not in grid:
      return 0
    return grid[(x, y)]

def set(x, y, v):
    grid[(x, y)] = v

def printGrid():
    for y in range(HEIGHT):
        for x in range(WIDTH):
            print(get(x, y), end=" ")
        print()

set(4, 3, "A")
set(1, 2, "B")
printGrid()
```

2D arrays: dict

Result:

```
0 0 0 0 0
0 0 0 0 0
0 B 0 0 0
0 0 0 0 A
```

pip install <package>

- Do leverage existing libraries

Popular packages

- https://docs.python.org/3.12/library/index.html
  - pprint
  - itertools
  - functools
  - datetime
  - colorful and colorama
  - …
- numpy
- scipy
- …