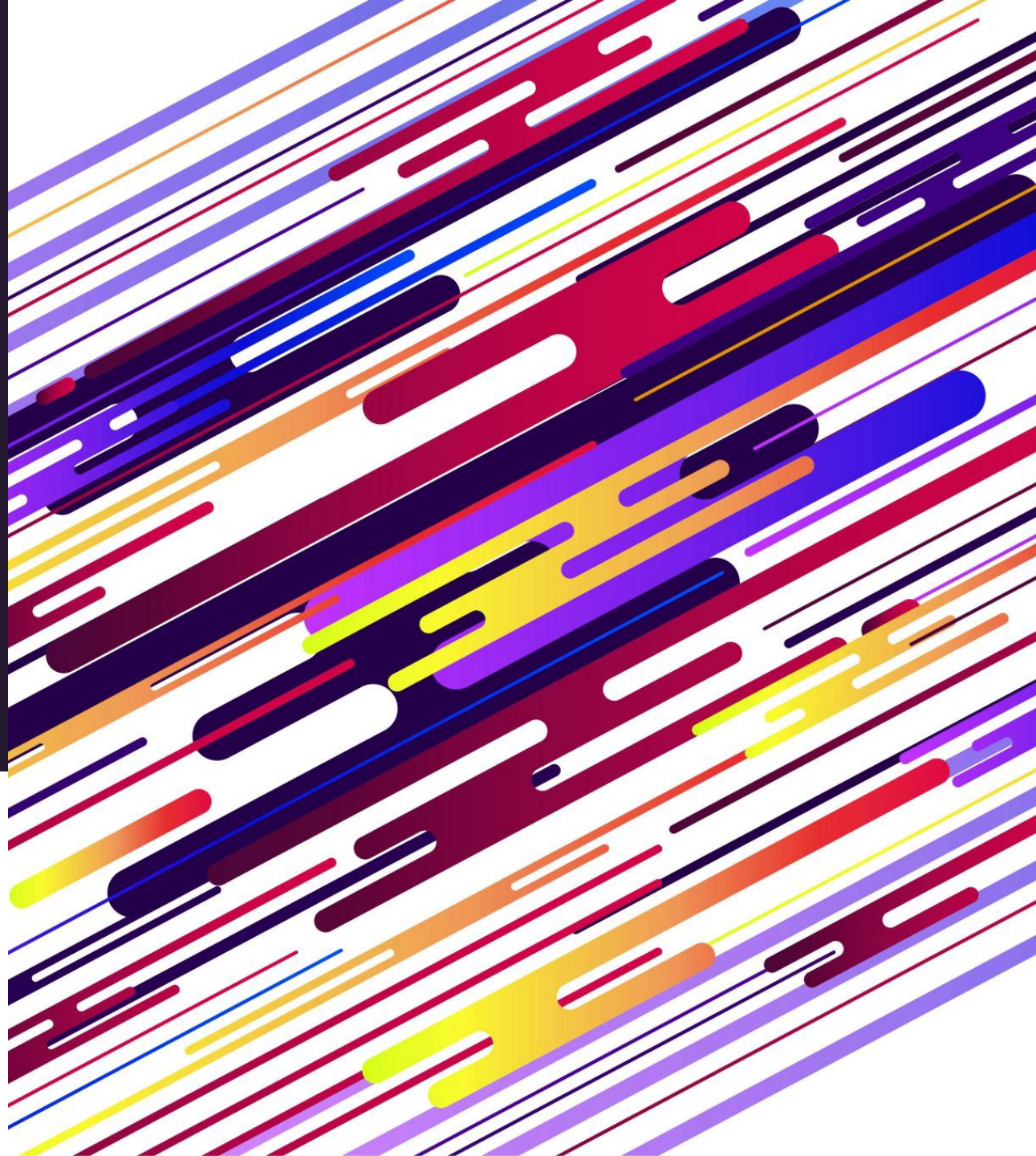
# Writing Better Python

Ynon Perek



# Course Agenda

Orientation Tips – Setup, Environment, Features

Functions Deep Dive

Text Processing

Object Oriented and Project Structure

Concurrency

# Recent Python Features

June 2018, Python 3.7:

- Async/await now reserved words
- Data Classes

# Recent Python Features

October 2019, Python 3.8:

1. Assignment expressions
2. Positional-only parameters
3. Equal in f-strings
4. Asyncio

# Recent Python Features

October 2020, Python 3.9:

- Union dicts
- string methods to remove prefixes and suffixes 😮

# Recent Python Features

October 2021, Python 3.10:

- Structural Pattern Matching
- Add Optional Length-Checking To zip (strict)
- Better Error Messages
- Many type-related improvements (we'll talk about them)

# Recent Python Features

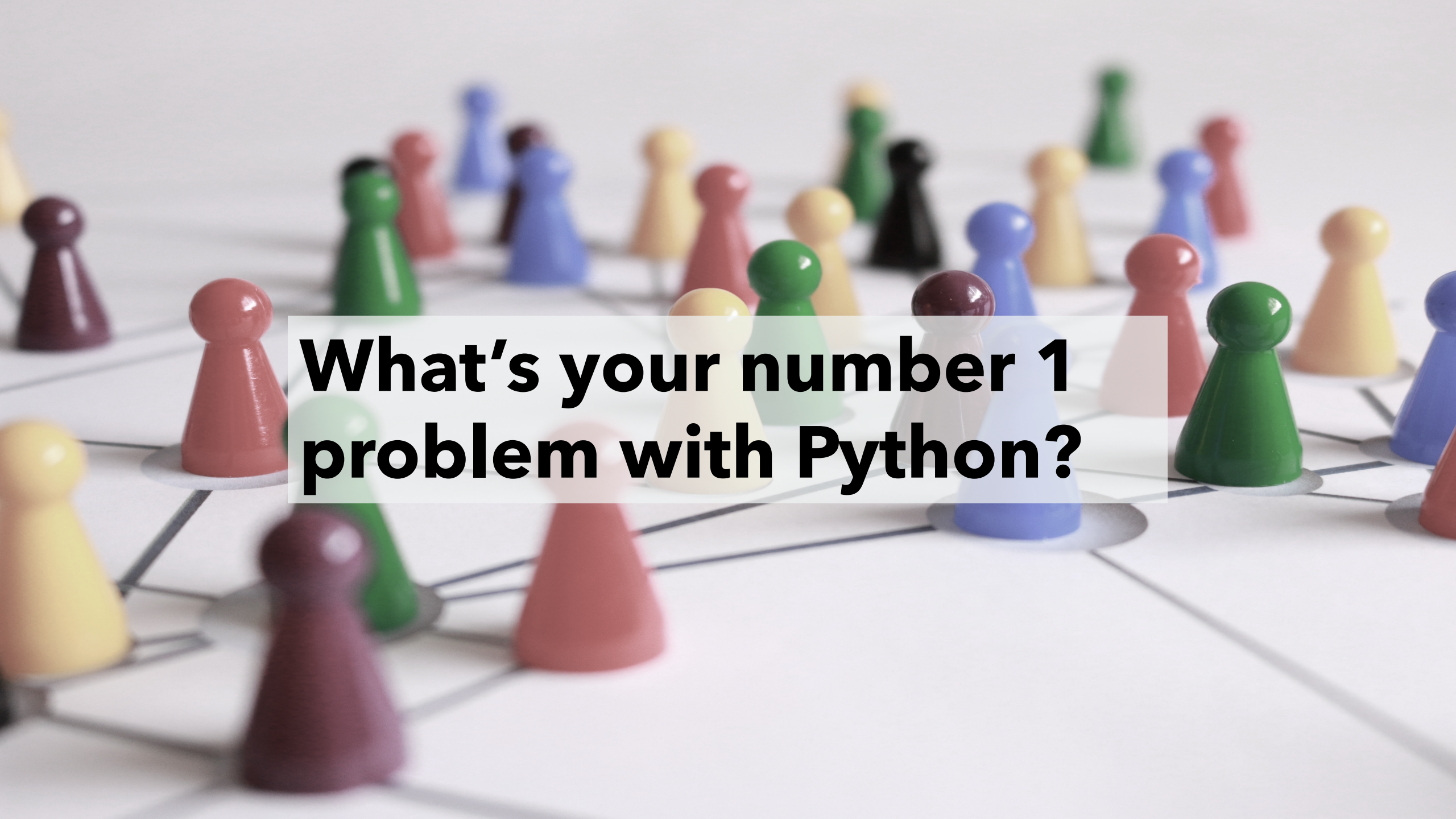
October 2022, Python 3.11:

- Python 3.11 is between 10-60% faster than Python 3.10
- Exception Groups
- Notes on exceptions

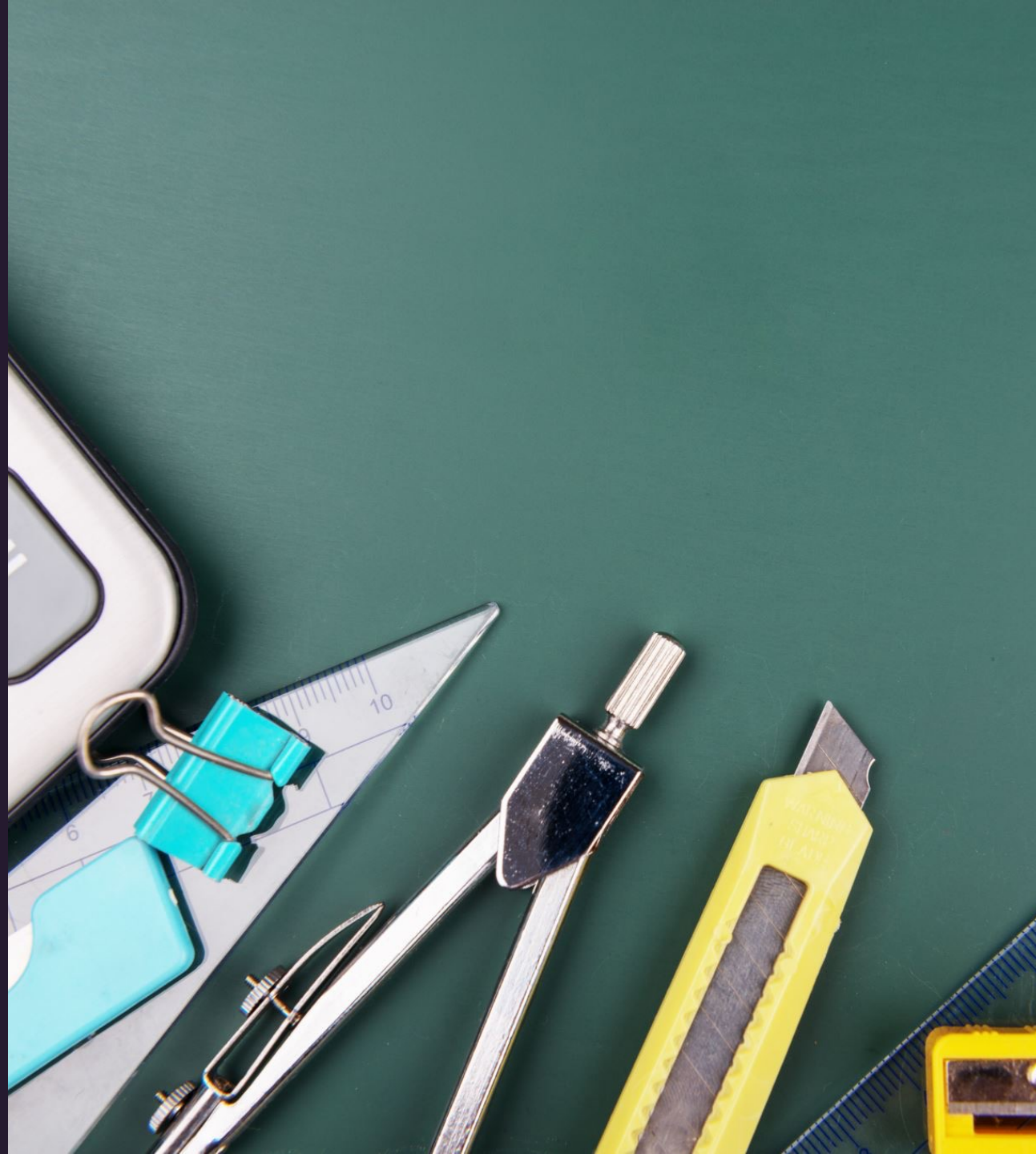
# Future Plans

Python 3.12:

- Better f-strings
- More improvements to the type system


What's your number 1
problem with Python?

# New Features - Examples



# Data Classes

```
from dataclasses import dataclass

@dataclass
class Point:
        x: int
        y: int

p = Point(10, 20)
```

# Assignment expressions

```
def my_rand1():
        x = random.random()
        if x > 0.5:
                return x / 2
        else:
                return x / 3
```

```
def my_rand2():
        if (x := random.random()) > 0.5:
                return x / 2
        else:
                return x / 3
```

# Assignment expressions

What happens here?

```
def my_rand2():
        if x := random.random() > 0.5:
                return x / 2
        else:
                return x / 3
```

# Prefix and Suffix removal

```
filename = '/home/ynon/python/myapp.py'

print(filename.removeprefix('/home/ynon'))
print(filename.removesuffix('.py'))
```

# Structural Pattern Matching

```
text = 'fw 10'

match text.split():
        case ['fw', steps]:
                print(f'Forward {int(steps)} steps’)
        case ['left', deg]:
                print(f'Left {int(deg)} degrees’)
        case ['right', deg]:
                print(f'Right {int(deg)} degrees’)
        case _:
                print(f"Unknown Command: {text}")
```

# Python IDE and Environments

Creating a virtual environment in pycharm

Activate this environment from command line

Freeze dependencies (requirements.txt)

# Q & A

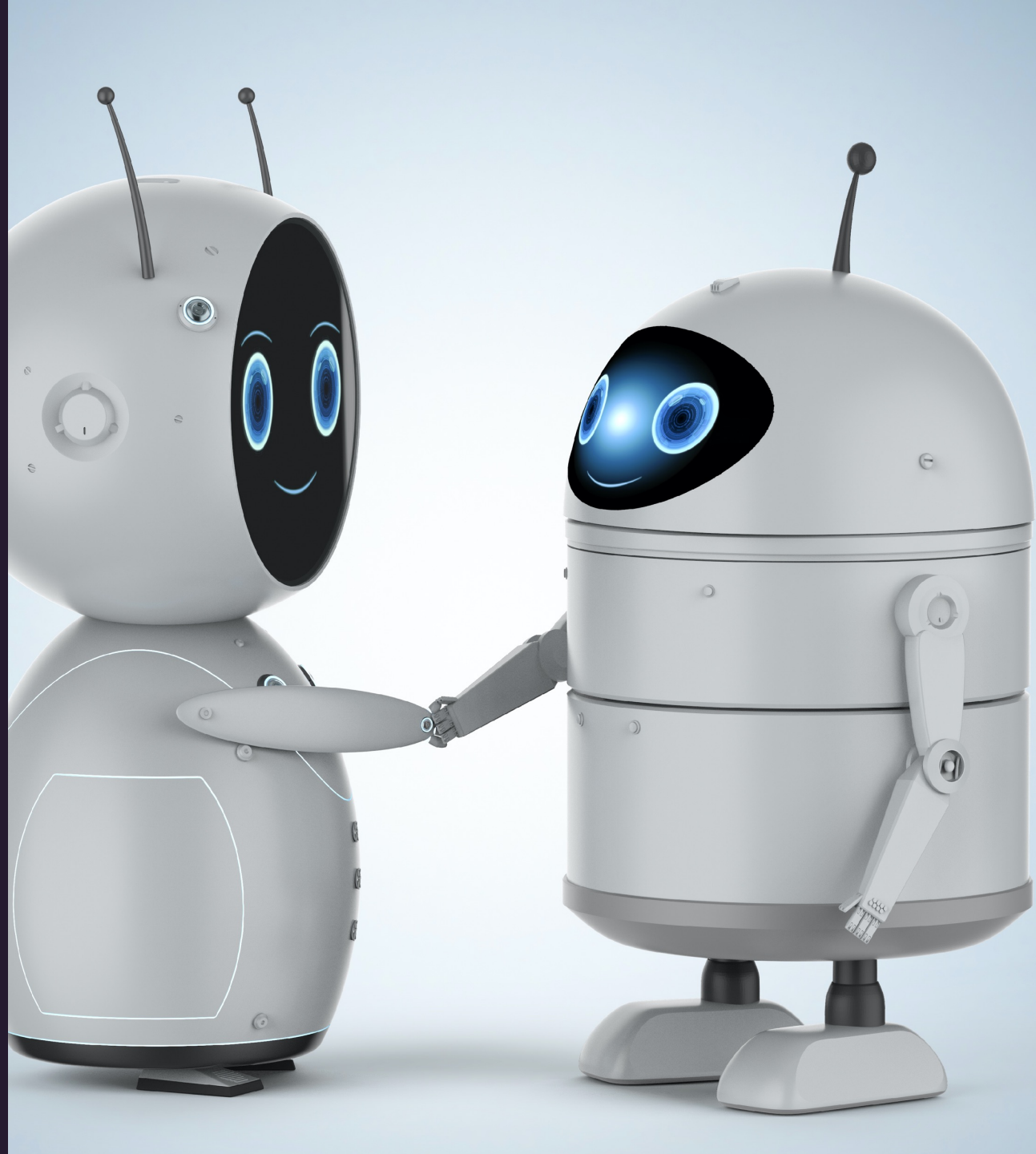
# Warm Up Exercises - Files

1. Write a python program to search for duplicate files in a directory (recursively):
    1. Use Path('…') to create a PosixPath ref to the input directory
    2. Use rglob to get all the files/directories recursively
    3. Use hashlib to calculate the sha256 hash for each file
    4. Group the files by their sha256 and show only groups that have more than one file with the same hash

# Warm Up Exercises - Files

1. Write a python program to count the number of files with each extension in a folder
    1. Use Path('…') to create a PosixPath ref to the input directory
    2. Use rglob to get all the files/directories recursively
    3. Group the files by their extensions and count how many files match each extension

# Python Type Hints



# How It Works

Use type hints to annotate the code

Use a tool (mypy / pycharm) to verify your code

**Python doesn’t care about type hints - they’re just for you**

# Living without Types

```
number = input("What is your favourite number?")
print("It is", number + 1)
```

# Living Without Types

```
items = [{'a': 10, 'b': 20, '_id': 1},
         {'a': 12, 'b': 31, '_id': 2}]

def print_item(items, index):
    i = items[index]
    del(i['_id'])
    print(i)

print_item(items, 0)
print_item(items, 1)
```

# Type Annotations Syntax

```
age: int = 1
child: bool

if age < 18:
        child = True
else:
        child = False
```

# Simple Types

```
x: int = 1

x: float = 1.0

x: bool = True

x: str = "test”

x: bytes = b"test"
```

# Collections

```
x: list[int] = [1, 2, 3]
x: set[int] = {1,2,3}
x: dict[str, float] = {"one": 1, "two": 2.0}
x: tuple[int, str, float] = (12, "name", 12.9)
```

# TypedDict

```
from typing import TypedDict

class Point2D(TypedDict):
    x: int
    y: int
    label: str

a: Point2D = {'x': 1, 'y': 2, 'label': 'good'}
```

# TypedDict

```
from typing import TypedDict, NotRequired, Literal

class Point2D(TypedDict):
    x: int
    y: int
    label: str
    color: NotRequired[Literal['blue', 'red', 'yellow', 'white', 'grey']]
```

# Functions

```
def stringify(num: int) -> str:
        return str(num)


def concat(numbers: list[float]) -> str:
        return ''.join(str(i) for i in numbers)
```

# Functions

```
def sum_even(*values: int):
        return sum(v for v in values if v % 2 == 0)

# prints 6
print(sum_even(2, 3, 4))
```

# Union

```
x: list[int|str] = [1, "wow", 2, "it's a union"]
```

# Automatic Type Inference

Doesn't always work

```
num = 920
sum_of_digits = 0

while num > 0:
sum_of_digits += (num % 10)
num = num / 10

print(sum_of_digits)
```

demo.py:6: error: Incompatible types in assignment (expression has type "float", variable has type "int")  [assignment]

# Q & A

# Exercise – Bulls & Cows

Write a Python program to play “Bulls and Cows” game:

1. The program randomizes a number of 5 different digits
2. User needs to guess the number
3. After each guess the program prints how many of the digits are correct but were not put in the correct position in the number, and how many of the digits are both correct and appear in the correct position in the number.

# Exercise - Dive

https://adventofcode.com/2021/day/2

# Exercise – Spaced Repetition (1/3)

1. In this exercise we'll write a spaced repetition practice app.
2. Implement a function `create_card` that takes two text strings for the front and back of a card. The function adds the strings to a card-deck saved in memory
3. Implement a function `play` that randomly selects a card from the deck, shows its front and the user has to type in the back.

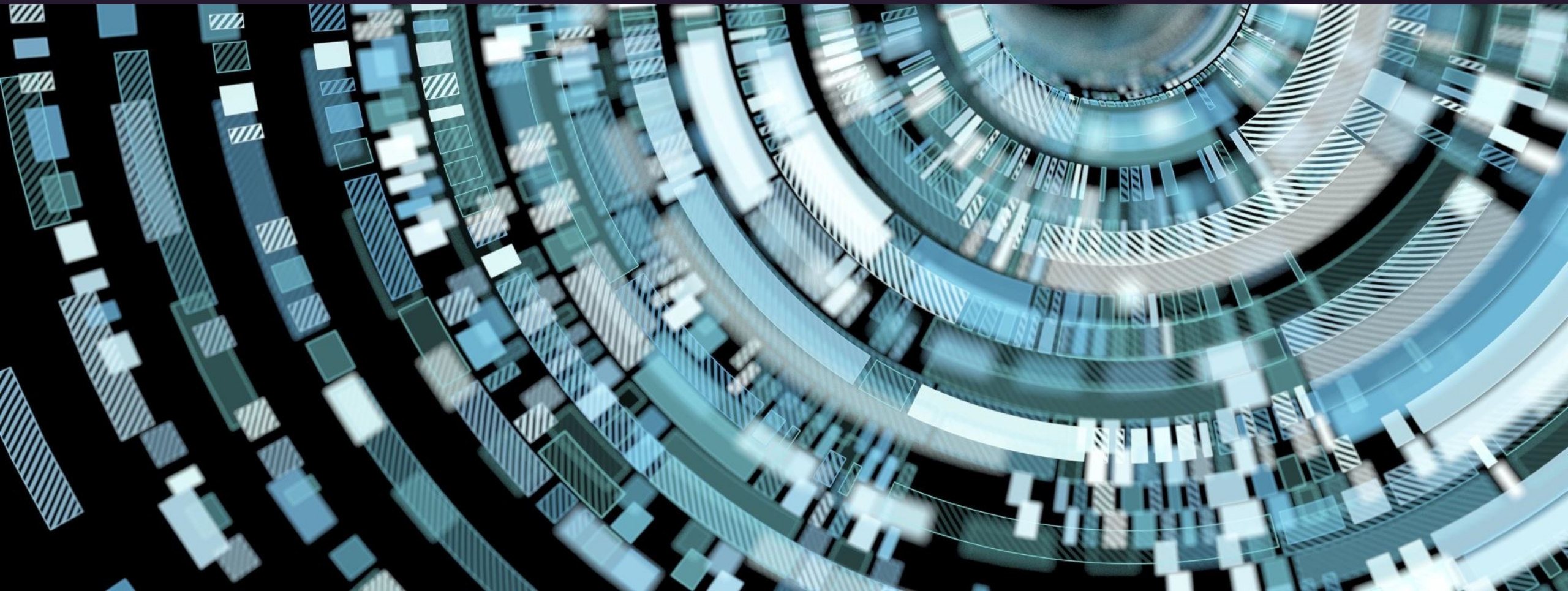
# Exercise – Spaced Repetition (2/3)

1. Save the card-deck to a file. After each item added re-save it.
2. Change `play` so it won't show the same cards again and again (think how you want to implement this).
3. Add an option to `play` to get a hint. If a user types `?` at the prompt, show them the first few characters of the `back` string.
4. Modify `play` so that it will sometimes show the back of the card, and the user will need to type the front part.

# Exercise – Spaced Repetition (3/3)

1. Allow a user to practice multiple decks
2. Each time they practice a deck they will get 4 questions from that deck
3. When the program starts it reads all the decks from a predefined folder
4. User is then presented with a list of all the decks and a prompt asking them to select:
    1. Add cards to deck
    2. Practice deck

# Python Virtual Environments



# Why Virtual Environment

1. Works on my machine
2. Version conflicts
3. Need to work on multiple projects

# Setup - Demo

1. Create a virtual environment
2. Install some packages into it
3. Freeze everything into requirements.txt file
4. Install the modules from requirements.txt to a new virtual environment

# Disadvantages of Virtual Env

1. Disk space
2. Can’t have dev only dependencies
3. Difficult to share env between projects
4. When adding dependencies need to check versions and modify requirements.txt

# Q & A

# Thanks For Listening