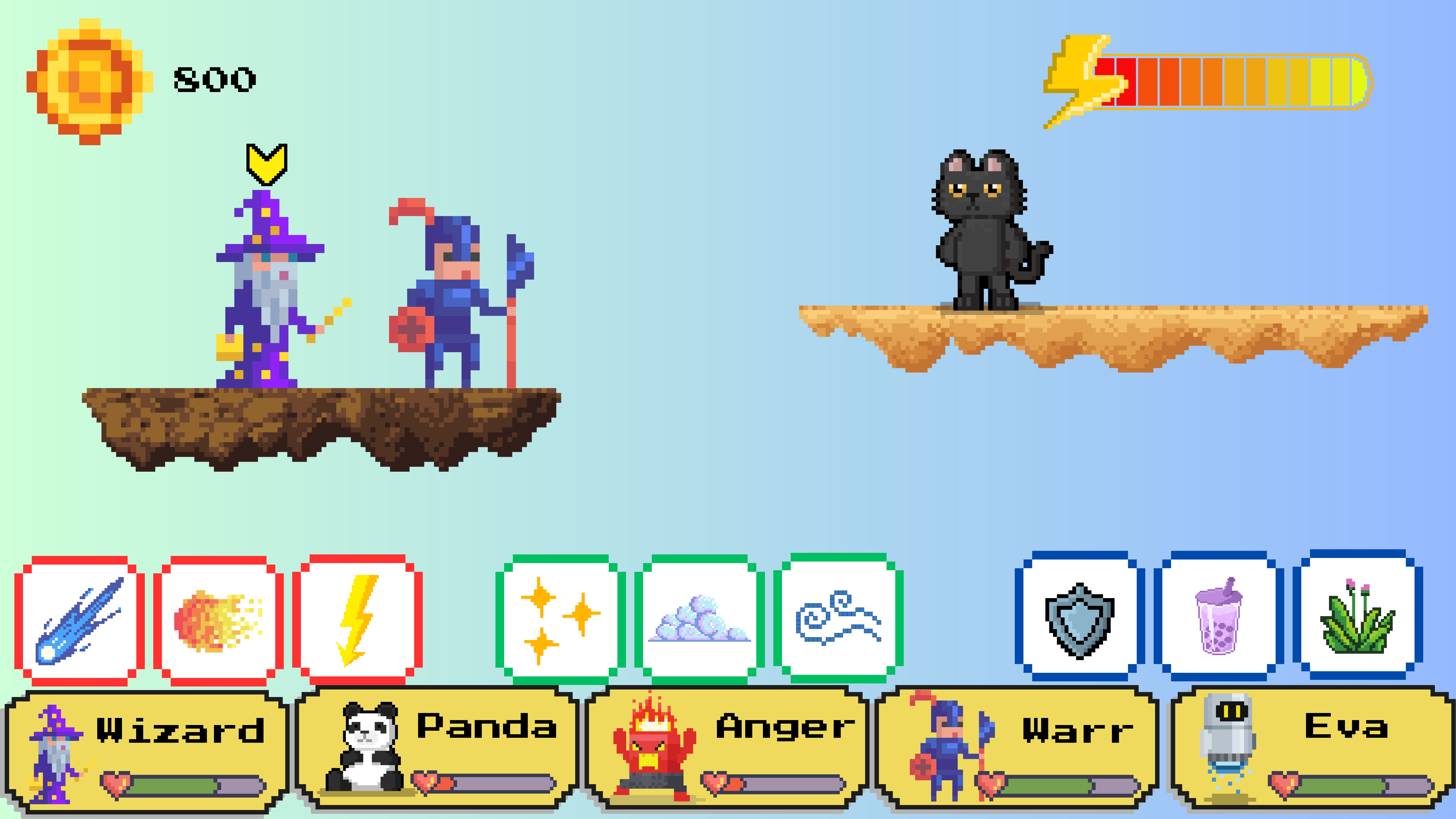
COMP10001 - Sem 2 2024 - Week 10

Foundations of Computing

Daksh Agrawal

PROJECT 2 OUT

START


800
Wizard
Panda
Anger
Warr
Eva

Level 1

History

Round 1:

1. - Haskell Heroine uses Recursion Rebuff to defend, protecting 2 health.
2. - Python Pal uses an item: Screen Repair Kit.

Round 2:

1. - Linux Legend uses Root Reckoning to attack, dealing 4 damage, attacking 2 players. Electricity: 5. Target: Haskell Heroine and Python Pal.
2. - Binary Bot uses an item: Debugging Tool.

Round 3:

1. Haskell Heroine uses Lambda Lunge to attack, dealing 3 damage, attacking 1 player. Electricity: 0. Target: Linux Legend.
2. Python Pal uses an item: RAM Boost.

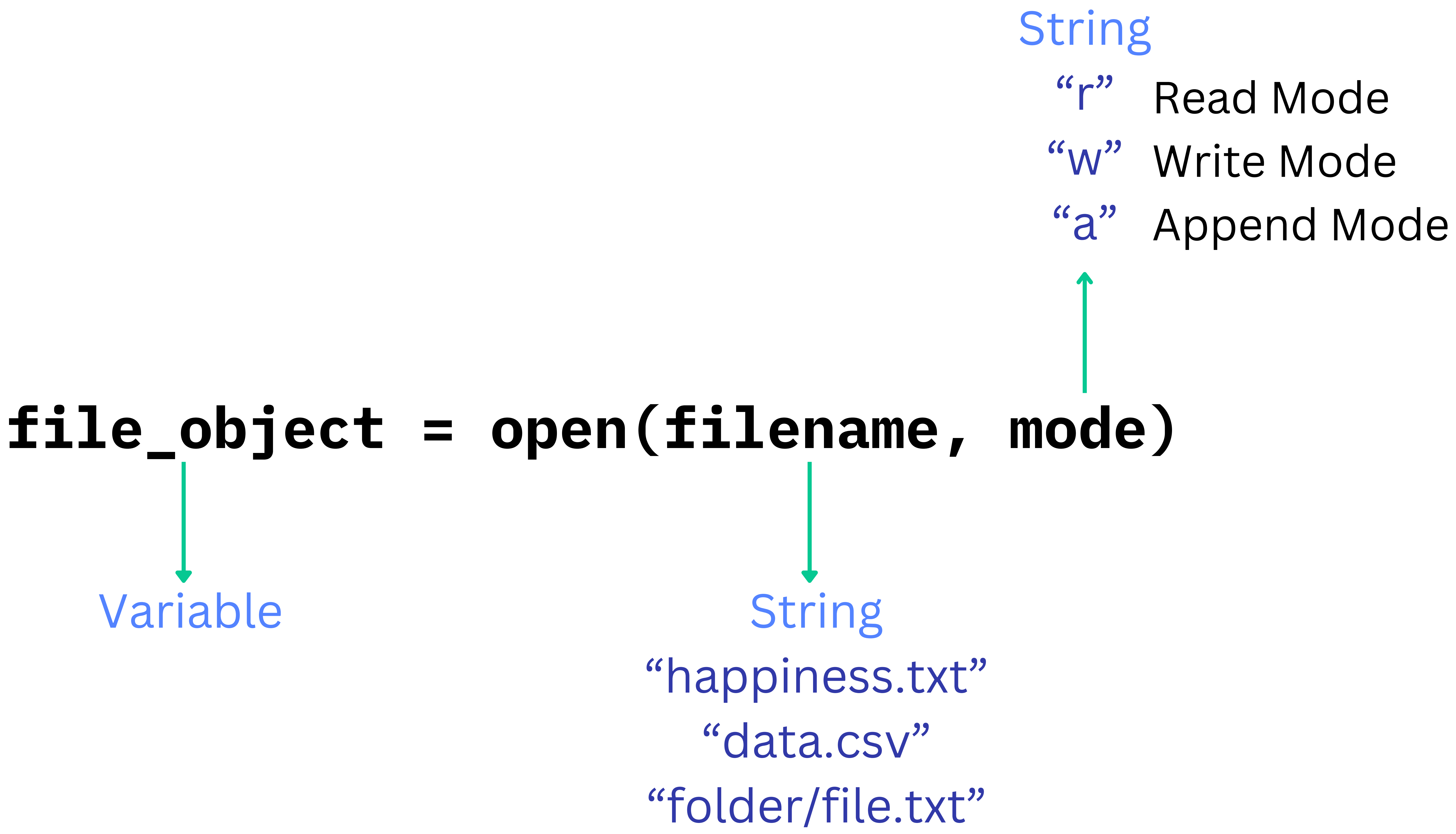
Round 4:

1. Linux Legend uses Kernel Kick to attack, dealing 4 damage, attacking 1 player. Electricity: 3. Target: Python Pal.
2. Binary Bot performs a swap: Network Ninja.

Level 1

player	attacks	defends	items	swaps	attempted damage	attempted protection	electricity used	turns taken
0	1	1	2	0	3	2	0	2
1	2	0	1	1	8	0	8	2

File Handling



```
file_object = open(filename, mode)
```

Read Mode “r”

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.

Write Mode “w”

~~Beautiful is better than ugly.~~
~~Explicit is better than implicit.~~
~~Simple is better than complex.~~
~~Complex is better than complicated.~~
~~Flat is better than nested.~~

Hello World

Append Mode “a”

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Hello World

```
zen = open(“zen.txt”, “r”)
zen.read()
“Beautiful is better than ugly.\nExplicit is better than
implicit.\nSimple is better than complex.\nComplex is better than
complicated.\nFlat is better than nested.”
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
```

```
zen = open(“zen.txt”, “r”)
zen.readline()
“Beautiful is better than ugly.”

zen.readline()
“Explicit is better than implicit.”
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
```

```
zen = open(“zen.txt”, “r”)
zen.readlines()
```

```
[“Beautiful is better than ugly.”,
“Explicit is better than implicit.”,
“Simple is better than complex.”,
“Complex is better than complicated.”,
“Flat is better than nested.”]
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
```

```
zen = open("zen.txt", "r")
zen.readlines()
zen.close()
```

Break the Connection, safely!

Now, fill in the blanks in the program below which reads from `in.txt` and writes to `out.txt`.

```
outfile = open("out.txt", "w")
with open("in.txt", "r") as infile:
    line_no = 1
    for line in infile.readlines():
        outfile.write(f"line: {line_no}, length: {len(line)}\n")
        line_no += 1
    outfile.write("The End")
outfile.close()
```

Level 2

```
comprpg save file

turn: 38

player: 0
money: 17
electricity: 17
n_active: 1
active_characters: ['Java Judger']
items: []
characters:
name: Java Judger
health: 14
defence: 0
effects: []
defeated: False
name: Python Pal
health: 0
defence: 0
effects: []
defeated: True
name: Haskell Heroine
health: 13
defence: 0
effects: []
defeated: False
name: Binary Bot
health: 20
defence: 0
effects: []
defeated: False
name: HTML Hero
health: 11
defence: 0
effects: []
defeated: False

player: 1
money: 11
electricity: 1
n_active: 1
active_characters: ['Python Pal']
items: []
characters:
name: Python Pal
health: 22
defence: 0
effects: []
defeated: False
name: C Charmer
health: 13
defence: 0
effects: []
defeated: False
name: Linux Legend
health: 4
defence: 0
effects: []
defeated: False
name: Binary Bot
health: 12
defence: 0
effects: []
defeated: False
name: Network Ninja
health: 0
defence: 0
effects: []
```

Level 2

Game

Turn

History[]

Player

player_id

characters[]

active_characters[]

items[]

money

electricity

n_active

Player

player_id

characters[]

active_characters[]

items[]

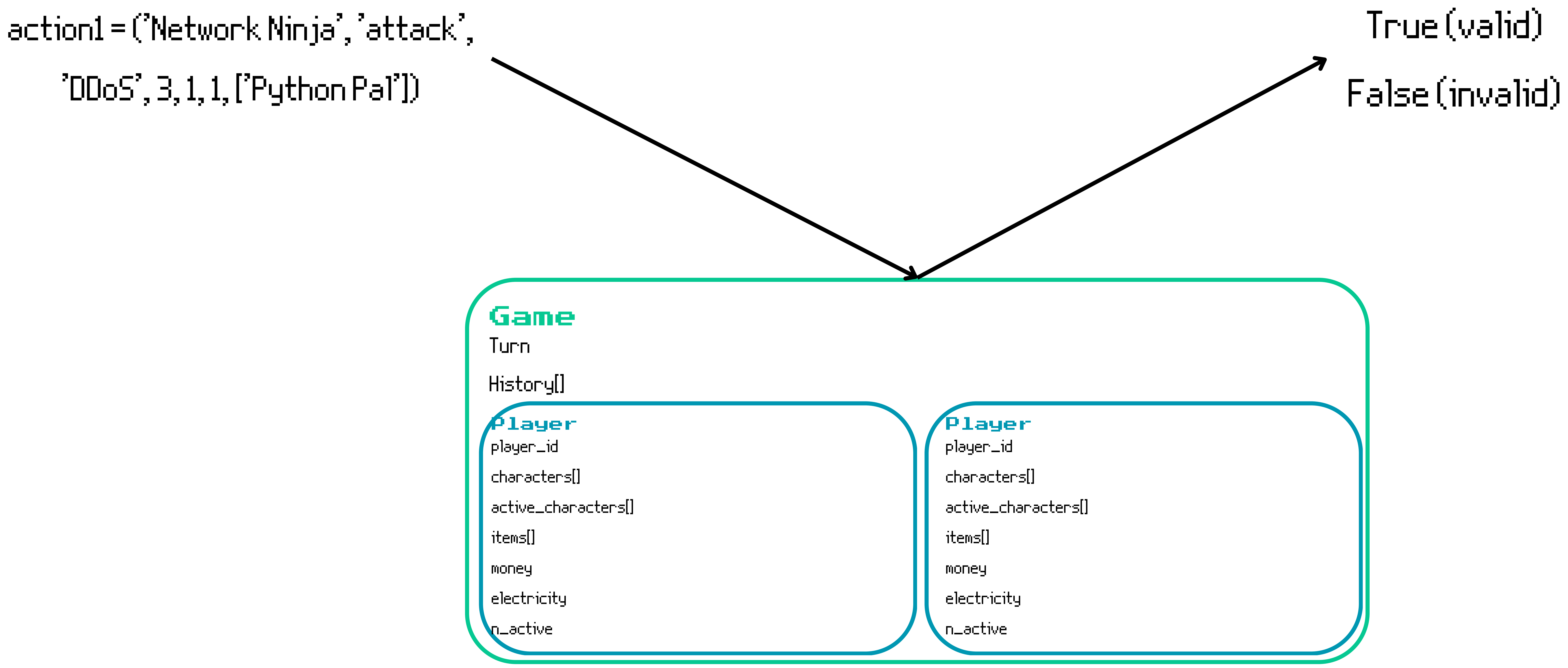
money

electricity

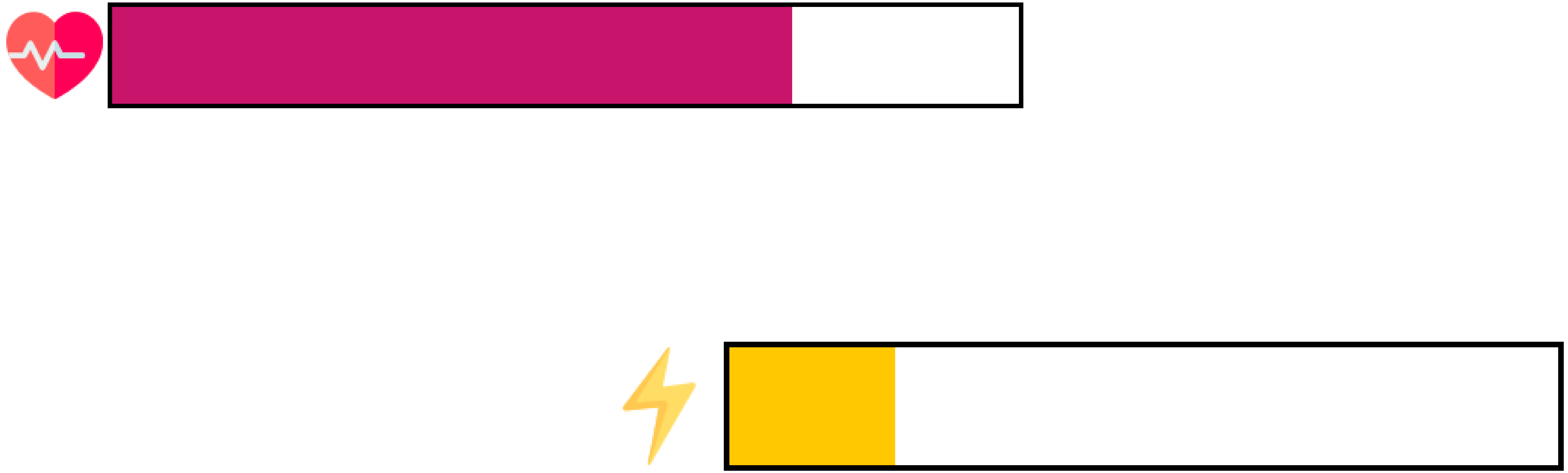
n_active

Classes

Level 3


action1 = ('Network Ninja', 'attack', 'DDoS', 3, 1, 1, ['Python Pal'])
True (valid)
False (invalid)
Game
Turn
History[]
Player
player_id
characters[]
active_characters[]
items[]
money
electricity
n_active
Player
player_id
characters[]
active_characters[]
items[]
money
electricity
n_active

Level 4



Images

Project 1

Review

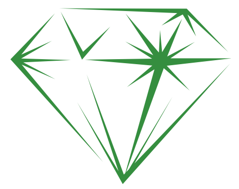
Project Submission Checklist

- No Magic Numbers, Strings, Lists, Dictionaries....
- EVERY Function should have a docstring
- Every Block of Code should have comment
- Question 2, 3 and 4 NEED helper functions!
- No Repeated Code

Stuck?

- Do it by hand, write down english instructions for your friend
- Code it up

Green Diamond Achieved?



- Task not complete yet.
- Think of corner cases, design few yourself.
- Fix Code Quality.

WORK ON YOUR PROJECT

And Ask Questions

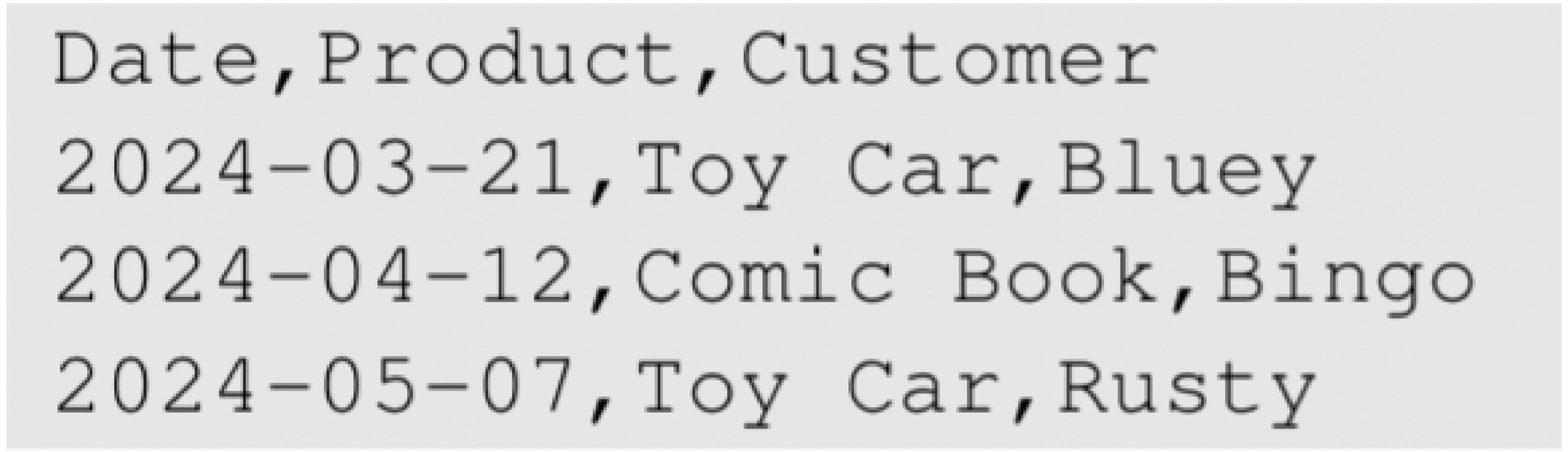
Write a function **sum_and_divide_x(seq, x)** that returns the sum of **seq** divided by **x**.

"Wrong type, can't sum" "Can't divide by 0" “Done”

```
>>> res1 = sum_and_divide_x([1,2,3], 2)
Done
>>> res1
3.0
>>> res2 = sum_and_divide_x([1,2,"hi"], 2)
Wrong type, can’t sum
Done
>>> type(res2)
<class ’NoneType’>
>>> res3 = sum_and_divide_x([1,2,3], 0)
Can’t divide by 0
Done
>>> type(res3)
<class ’NoneType’>
```

When handling csv files, there are a couple of ways we can get the data out of the csv file and into our program: **csv.reader** and **csv.DictReader**. Try to use **csv.DictReader** to solve this problem.

Write a function **count_sales(csv_filename)**, that takes a string csv filename, and returns a dictionary that counts the frequency of products sold. On the example file shown below, it should return **{'Toy Car': 2, 'Comic Book': 1}**.



```
Date,Product,Customer
2024-03-21,Toy Car,Bluey
2024-04-12,Comic Book,Bingo
2024-05-07,Toy Car,Rusty
```

You’ve found a secret message:

```
secret_message.txt
```

```
erkbvl ur kbvd tlmexr:
gxoxk zhggt zbox rhn ni
gxoxk zhggt exm rhn whpg
gxoxk zhggt kng tkhngw tgw wxlxkm rhn
gxoxk zhggt ftdx rhn vkr
gxoxk zhggt ltr zhhwurx
gxoxk zhggt mxee t ebx tgw ankm rhn
```

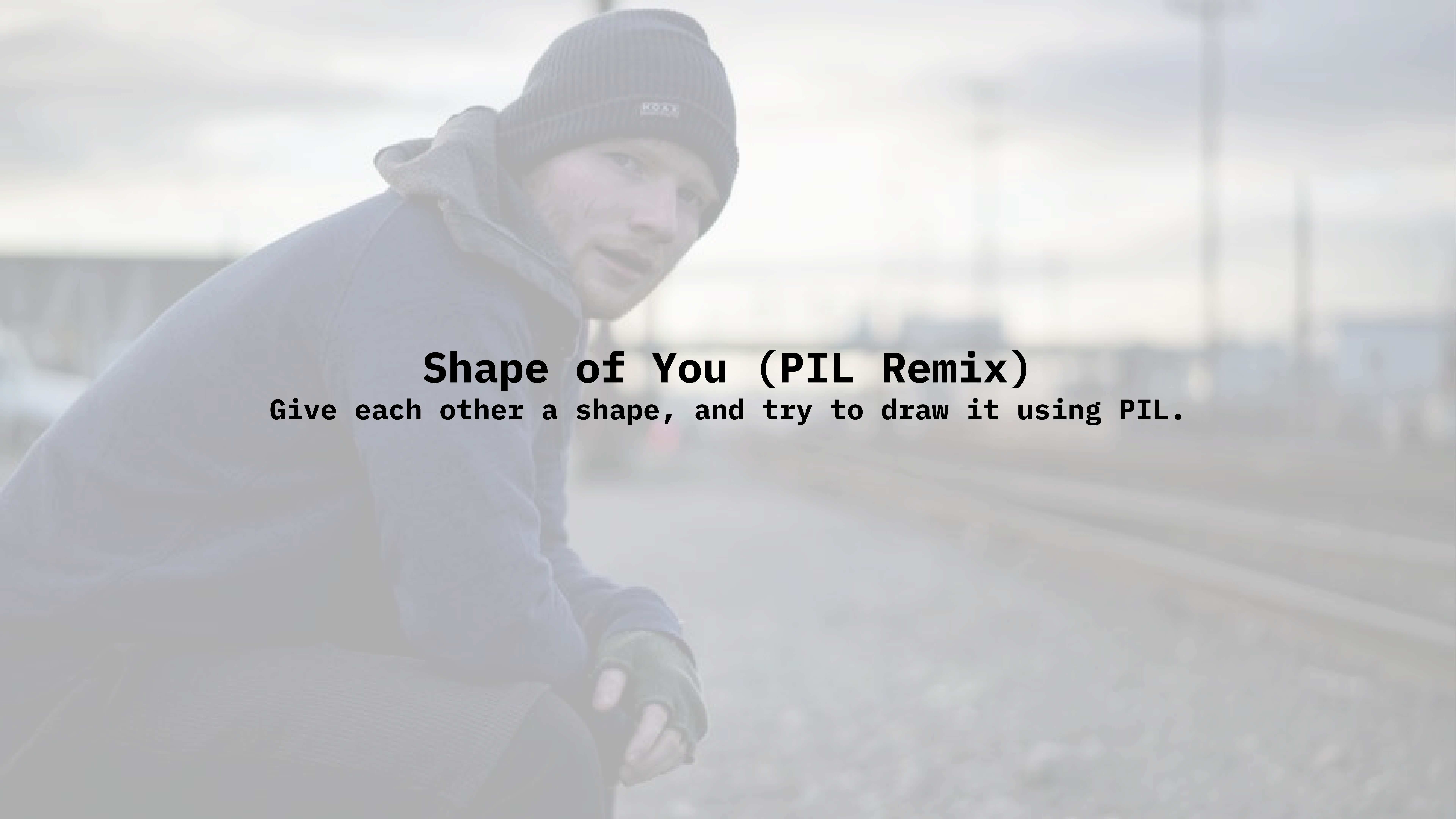
All that you know about the message is that it is encrypted by a basic shift cipher (also known as a Caesar cipher, where each letter is shifted by some constant number of places in the alphabet), any alphabetic character in the message is lowercase, and that it contains the string segment **'desert'**.
Write a function to decrypt the message that takes an **infilename**, **outfilename** and **segment** (all strings, and you can assume that all files exist). You can use a brute-force approach (try all possible values) to guess the number to shift by. You might find the functions **ord(character)** and **chr(number)** useful!

Revision Problems

1. Write a function that takes a lowercase string as input and prints the frequency of each vowel in the string. The printed results should be in alphabetical order. `vowel_counts('i love python')` should print:

   ```
   e 1
   i 1
   o 2
   ```

2. Write a function which takes two lists of integers and returns the average of the numbers which they both have in common. `in_common_average([1, 2, 3, 4, 5], [0, 2, 4, 6])` should return `3.0`



Shape of You (PIL Remix)

Give each other a shape, and try to draw it using PIL.