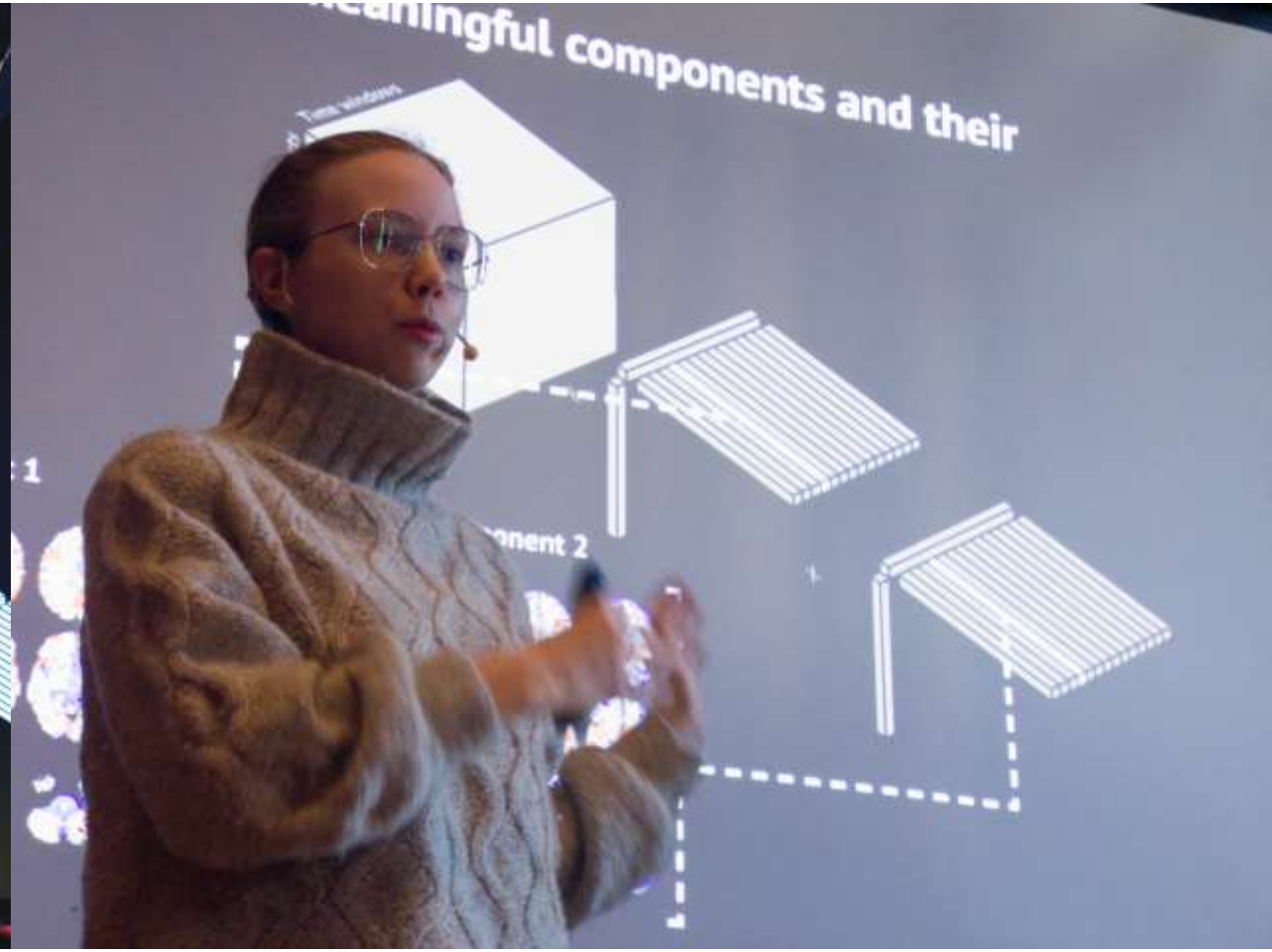
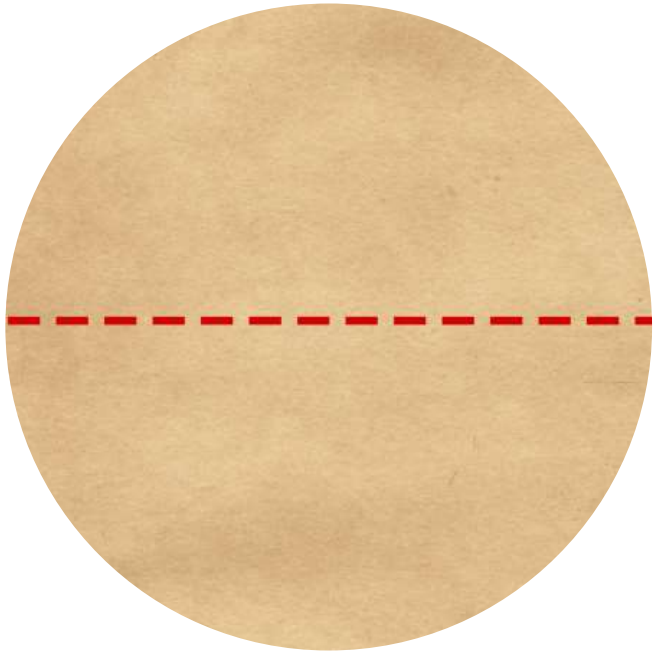

Teaching teachers
to teach Python:
what have
I learned?
Marie Roald
PyCon Italia
2023


An AO-ADMM approach to
constrained PARAFAC2
Marie Roald, Carla Schenker, Vince D. Calhoun, Tülay Adalı,


ningful components and their

In this talk I want to share some of the lessons I've learned from working on Python courses for teachers



Blank editor

Blank editor

Working code


Blank
editor
Working
code

Blank editor

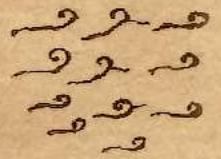
The "Complicated Problem Where Do We Even start??" Fog.

Working code

Enigmatic Error Enclave

Blank editor

Working code

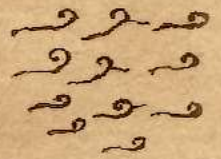


The "Complicated Problem Where Do We Even start??" Fog.

Enigmatic Error Enclave

Blank editor

Working code



The "Complicated Problem Where Do We Even start??" Fog.

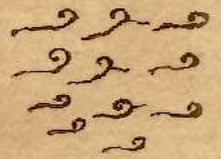
The "Wrong Result But No Error Message" Hollows

Enigmatic Error Enclave

Blank editor

Isles of Incomprehensible Identation

Working code



The "Complicated Problem Where Do We Even start??" Fog.

The "Wrong Result But No Error Message" Hollows

Enigmatic Error Enclave

Blank editor

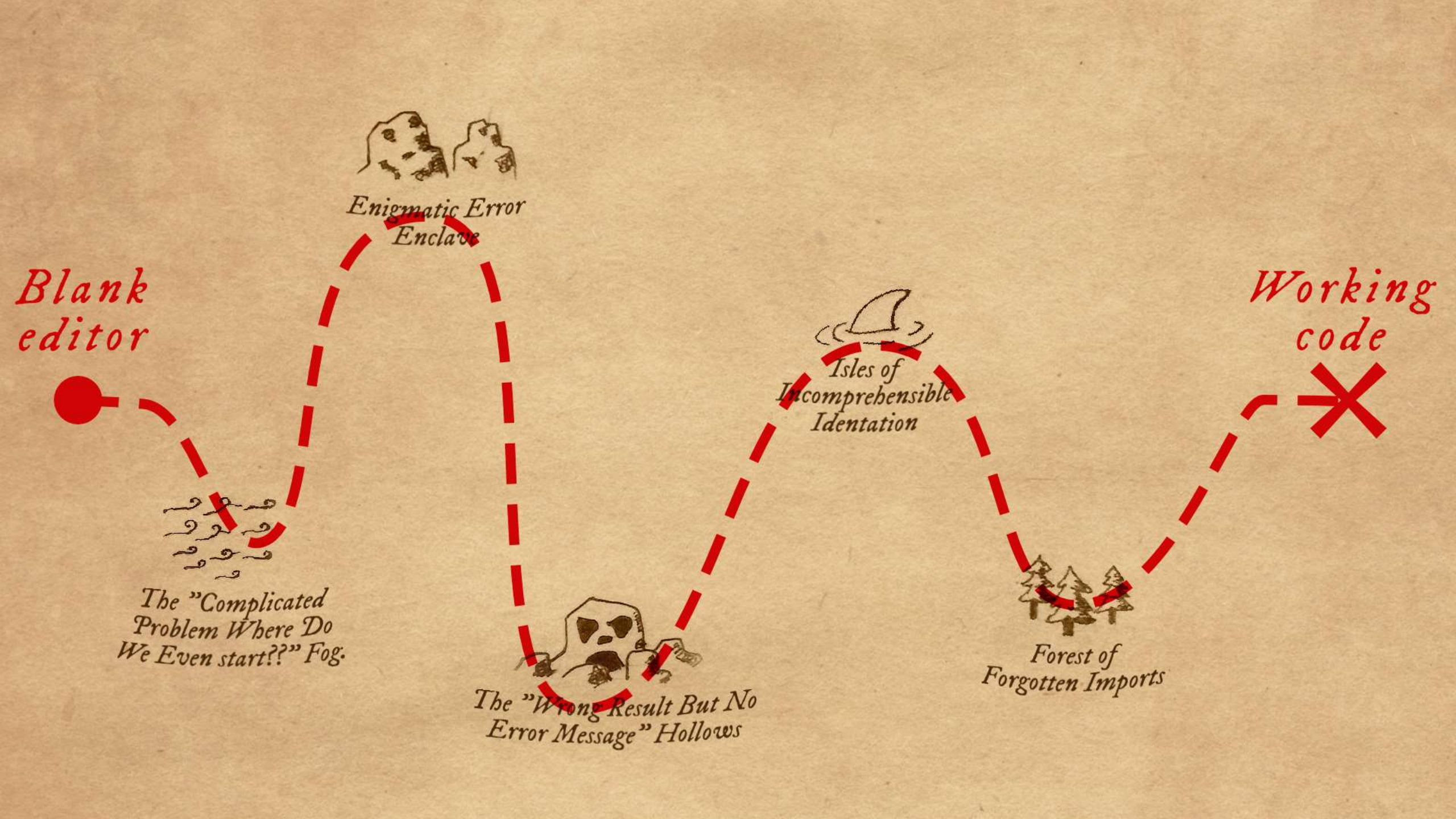
Isles of Incomprehensible Identation

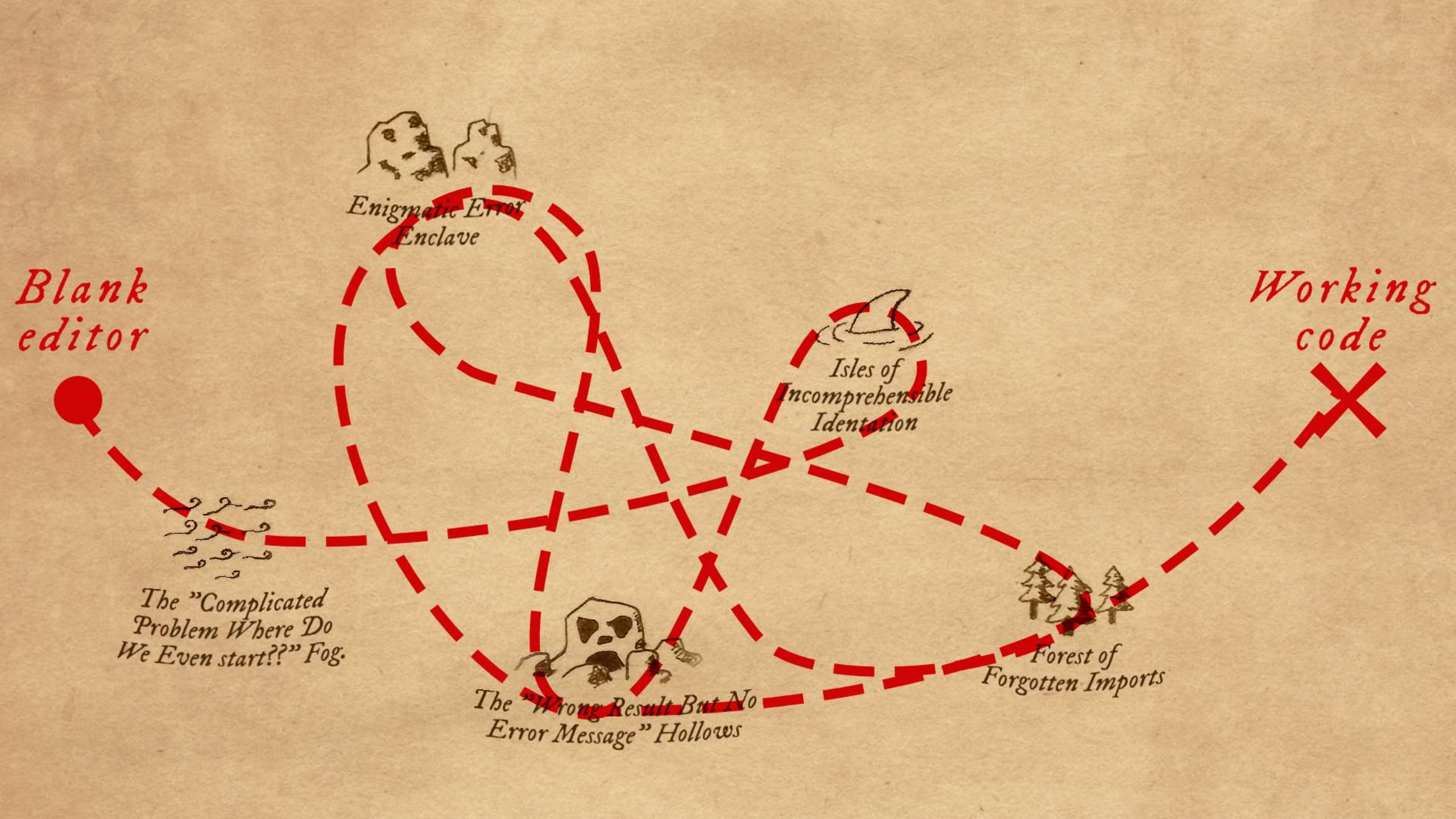
Working code

The "Complicated Problem Where Do We Even start??" Fog.

The "Wrong Result But No Error Message" Hollows

Forest of Forgotten Imports


Blank editor
Working code
Enigmatic Error Enclave
Isles of Incomprehensible Identation
The "Complicated Problem Where Do We Even start??" Fog.
The "Wrong Result But No Error Message" Hollows
Forest of Forgotten Imports


Blank editor
Working code
Enigmatic Error Enclave
Isles of Incomprehensible Identation
The "Complicated Problem Where Do We Even start??" Fog.
The "Wrong Result But No Error Message" Hollows
Forest of Forgotten Imports

Challenge: Coding is more than just the finished code

Challenge: Coding is more than just the finished code

Lesson learned: Use interactive live coding and not just static slides


for square in range(8):
for side in range(4)
pen.forward(100)
pen.right(90)
pen.right(45)
import turtle
pen = turtle.Turtle()


import turtle
pen = turtle.Turtle()
for square in range(3):
for side in range(4):
pen.forward(100)
pen.right(90)
pen.right(45)

Challenge: Talking about your code is important but, it does not come naturally for beginners

Challenge: Talking about your code is important but, it does not come naturally for beginners

Lesson learned: Treat «talking code» as a skill that needs practice just like writing code

A/B exercises help learners practice “talking code”

A/B exercises help learners practice “talking code”

Task A: Placing a planet in orbit

Part 1 - Making the program

As an evil scientist, you want to create a new artificial planet that orbits the sun. To ensure that your nefarious planet stays in orbit, you need to shoot it off with the correct speed, v. After doing some calculations, you find that the speed (measured in km/s) must be

$$v = \frac{365\,000}{\sqrt{d}}$$

where d is the distance between the sun and the artificial planet measured in kilometres.

Example runs:

```
How many km between the sun and your nefarious planet? 150000000
The velocity is: 29.802252083862 km/s

How many km between the sun and your nefarious planet? 220000000
The velocity is: 24.172315518437048 km/s

How many km between the sun and your nefarious planet? 180000000
The velocity is: 27.212413375822235 km/s
```

Task:

1. Create a program that asks the user for the distance between a planet and prints out the velocity of that planet.
2. Test your code by computing the velocity of the Earth ($d = 150\,000\,000$) and Venus ($d = 108\,000\,000$). Compare with the above example runs to see if your implementation is correct.
3. **Bonus exercise:** Create a utility that takes the velocity as input and prints the distance to the sun.

Task B: Escaping a new planet

Part 1 - Making the program

Your nemesis, an evil scientist, keeps placing you on foreign planets, and to get back to Earth, you first need to know how far the artificial planet you're stuck on is from the sun. Luckily, you got your watch, and from physics class, you remember that the distance, d, between the sun and a planet (measured in km) is given by

$$d = 2\,930\,000\,T^{\frac{2}{3}}$$

where T the duration of a year measured in earth days.

Example runs:

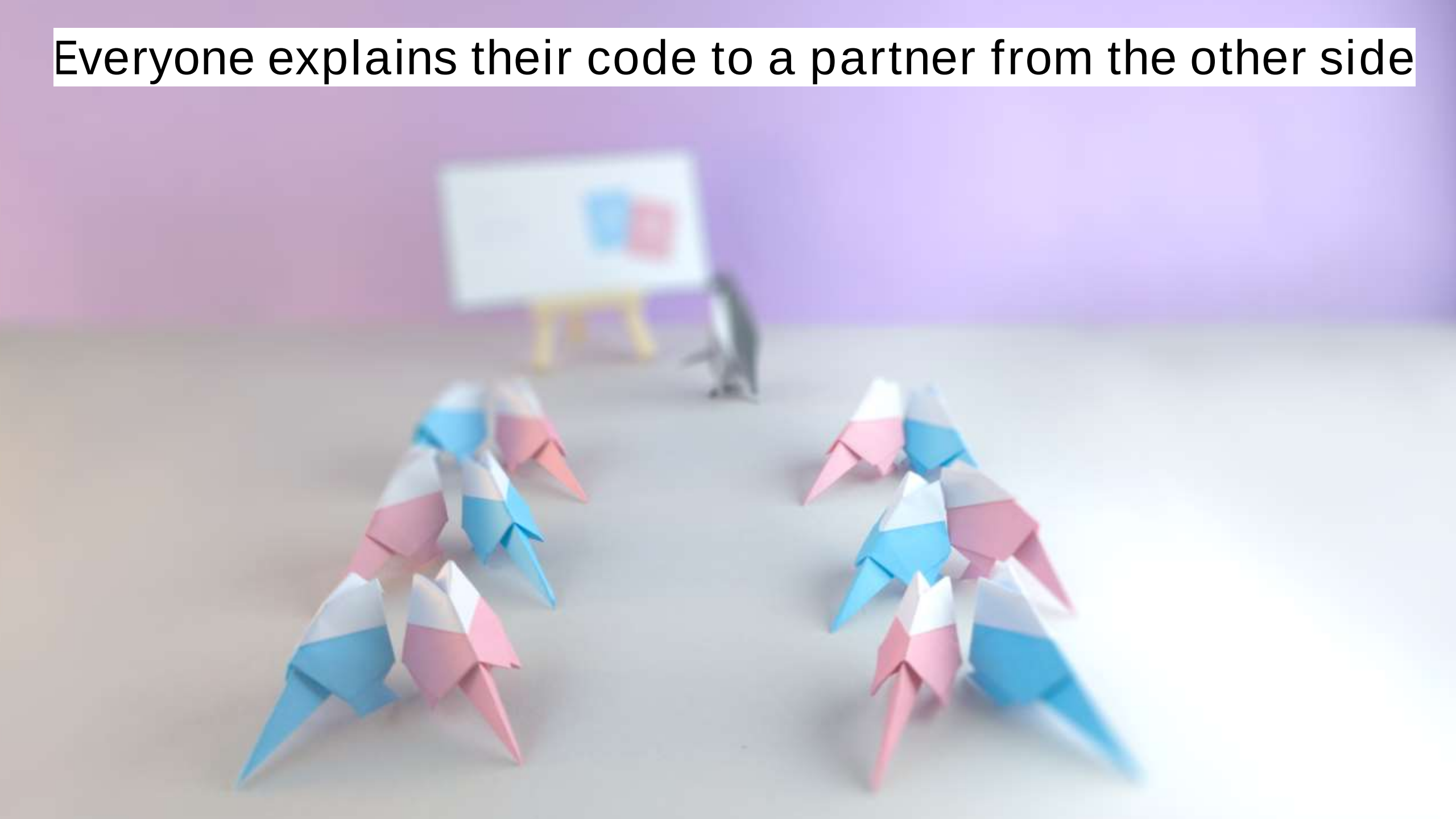
```
What is the duration of a year on the artificial planet (measured in earth days)? 365.25
The distance to the sun is: 149713548.26702202 km

What is the duration of a year on the artificial planet (measured in earth days)? 687
The distance to the sun is: 228124595.49833698 km

What is the duration of a year on the artificial planet (measured in earth days)? 225
The distance to the sun is: 188398028.80682413 km/s
```

Task:

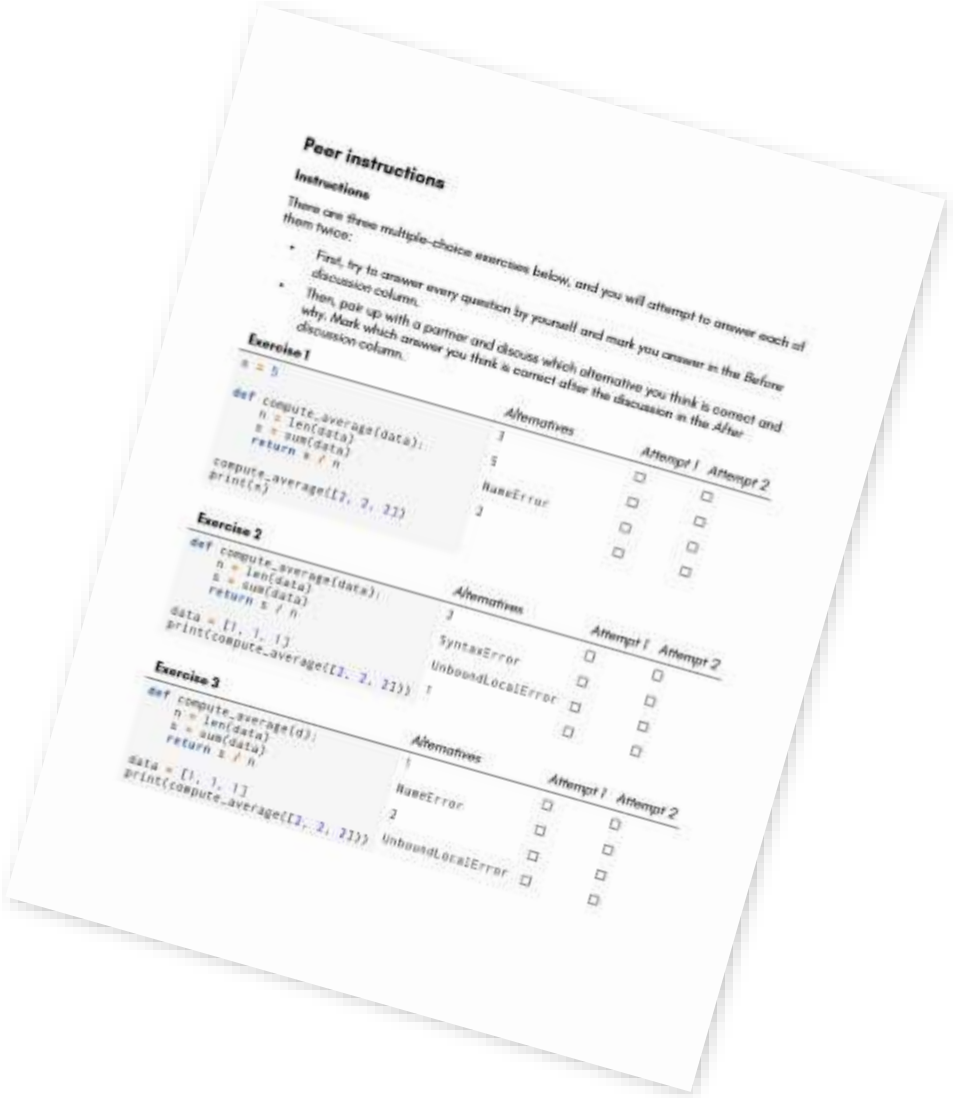
1. Create a program that asks the user for the duration of a year and prints out the distance between the sun and the planet.
2. Test your code by computing the distance between the sun and the Earth ($T = 365.25$). Mars ($T = 687$) and Venus ($T = 225$). Compare with the above example runs to see if your implementation is correct.
3. **Bonus exercise:** Create a utility that takes the distance to the sun as input and prints the duration of a year.

One side solves exercise A, and one side solves exercise B

Everyone explains their code to a partner from the other side



Peer instruction helps learners clarify their thinking



Peer instructions

Instructions

There are three multiple-choice exercises below, and you will attempt to answer each of them twice:

- First, try to answer every question by yourself and mark you answer in the *Before* discussion column.
- Then, pair up with a partner and discuss which alternative you think is correct and why. Mark which answer you think is correct after the discussion in the *After* discussion column.

Exercise 1

```
a = 5

def compute_average(data):
    n = len(data)
    s = sum(data)
    return s / n

compute_average([1, 2, 2])
print(a)
```

Alternatives	Attempt 1	Attempt 2
1	☐	☐
5	☐	☐
NameError	☐	☐
2	☐	☐

Exercise 2

```
def compute_average(data):
    n = len(data)
    s = sum(data)
    return s / n

data = [1, 1, 1]
print(compute_average([1, 2, 2]))
```

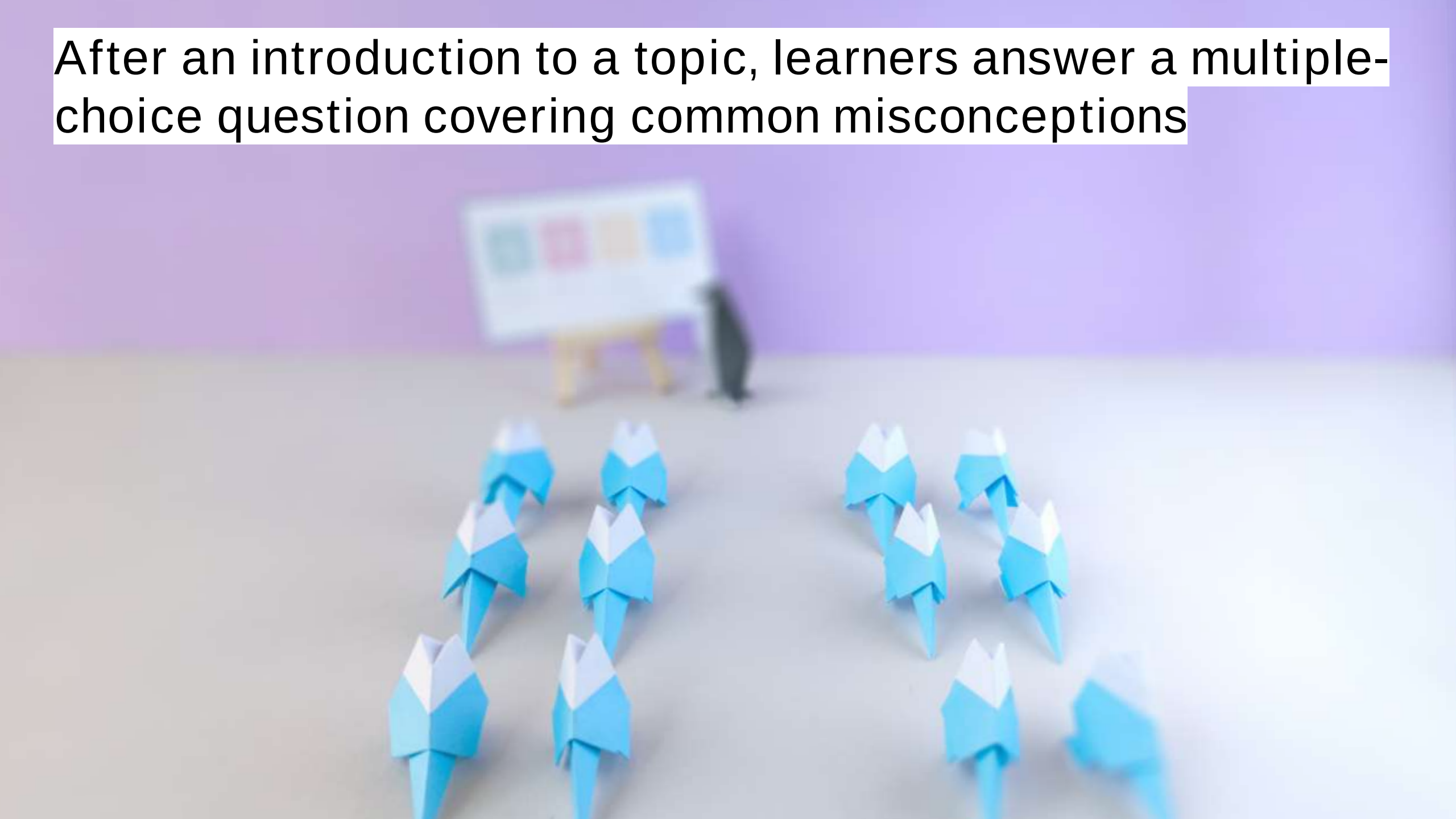
Alternatives	Attempt 1	Attempt 2
3	☐	☐
SyntaxError	☐	☐
UnboundLocalError	☐	☐
1	☐	☐

Exercise 3

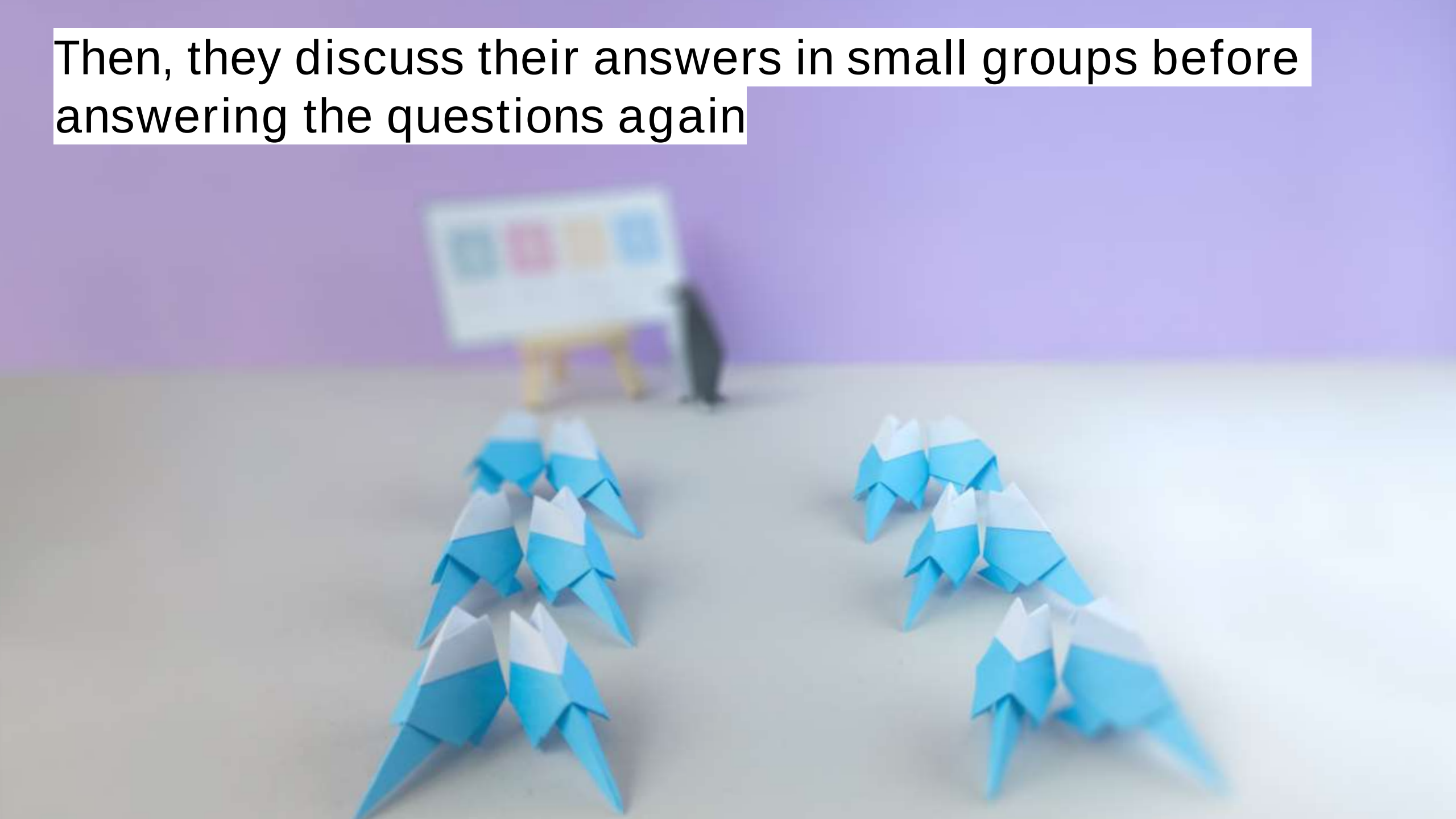
```
def compute_average(d):
    n = len(data)
    s = sum(data)
    return s / n

data = [1, 1, 1]
print(compute_average([1, 2, 2]))
```

Alternatives	Attempt 1	Attempt 2
1	☐	☐
NameError	☐	☐
2	☐	☐
UnboundLocalError	☐	☐

After an introduction to a topic, learners answer a multiple-choice question covering common misconceptions



Then, they discuss their answers in small groups before answering the questions again




SUCCESS!

Challenge: It's difficult to design exercises that are both beginner friendly and engaging

Boilerplate code slows things down and distract learners from the intended point of a lesson

```
import json
import matplotlib.pyplot as plt

with open("wolf_pair_count.json") as f:
    wolf_data = json.load(f)
wolf_count = wolf_data["count"]

max_population = 0
year_of_max_population = 0

for year, population in wolf_count.items():
    if population > max_population:
        max_population = population
        year_of_max_population = year

plt.plot(wolf_count.keys(), wolf_count.values())
plt.plot(year_of_max_population, max_population, 'o')
plt.axvline(year_of_max_population, linestyle='--')
plt.xlabel("Year")
plt.ylabel("Number of mated wolf pairs")
plt.title("Wolf population in Norway and Sweden")
plt.show()
```

Boilerplate code slows things down and distract learners from the intended point of a lesson

```
import json
import matplotlib.pyplot as plt

with open("wolf_pair_count.json") as f:
    wolf_data = json.load(f)
wolf_count = wolf_data["count"]
```

```
max_population = 0
year_of_max_population = 0

for year, population in wolf_count.items():
    if population > max_population:
        max_population = population
        year_of_max_population = year
```

```
plt.plot(wolf_count.keys(), wolf_count.values())
plt.plot(year_of_max_population, max_population, 'o')
plt.axvline(year_of_max_population, linestyle='--')
plt.xlabel("Year")
plt.ylabel("Number of mated wolf pairs")
plt.title("Wolf population in Norway and Sweden")
plt.show()
```

Boilerplate code slows things down and distract learners from the intended point of a lesson

```
import json
import matplotlib.pyplot as plt

with open("wolf_pair_count.json") as f:
    wolf_data = json.load(f)
wolf_count = wolf_data["count"]
```

```
max_population = 0
year_of_max_population = 0

for year, population in wolf_count.items():
    if population > max_population:
        max_population = population
        year_of_max_population = year
```

```
plt.plot(wolf_count.keys(), wolf_count.values())
plt.plot(year_of_max_population, max_population, 'o')
plt.axvline(year_of_max_population, linestyle='--')
plt.xlabel("Year")
plt.ylabel("Number of mated wolf pairs")
plt.title("Wolf population in Norway and Sweden")
plt.show()
```

a) Import the `json` library and use json.load to load the wolf data from the `"wolf_pair_count.json"` file and store it in `wolf_data`

b) Get the `"count"` index from `wolf_data` and store it in a variable named `wolf_count`

c) Create a for-loop that iterates over the year and population in `wolf_count.items()`

d) Create a variable max_population that starts at zero, and for each iteration, if the population is higher than `max_population`, then update `max_population` to be equal to the population that iteration.

e) Update the for-loop so you also store and update the year of the maximal population

f) Use `plot` from `matplotlib.pyplot` to plot the number of mated wolf pairs (`wolf_count.values()`) against year (`wolf_count.keys()`)

g) Update the plot so you highlight the year of the highest wolf population

h) Add an appropriate x-label, y-label and title to the plot

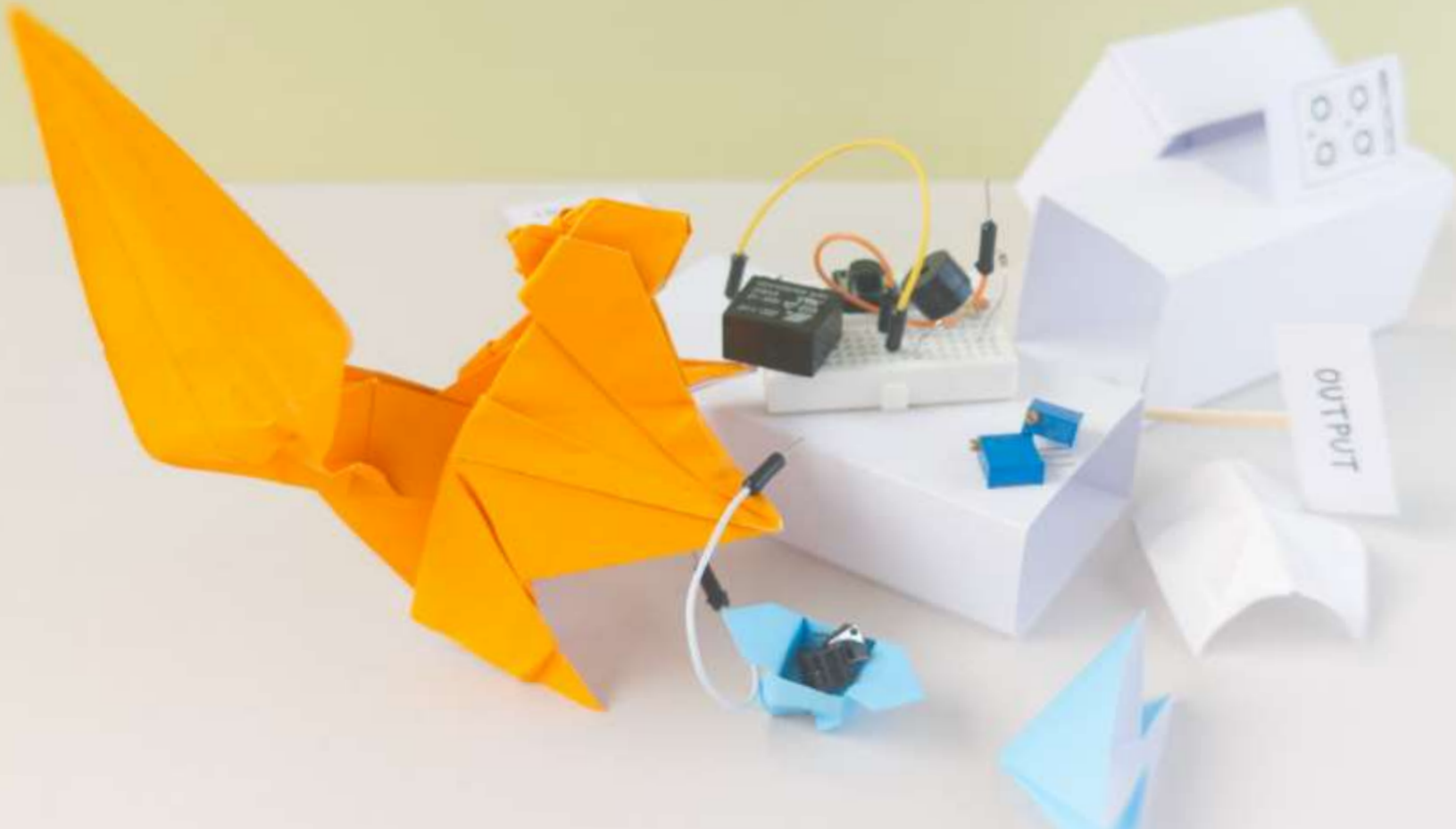
i) Run the code and look at the plot, which year has the highest wolf population?

Challenge: It's difficult to design exercises that are both beginner friendly and engaging

Lesson learned: Use the Use Modify Create framework to give beginners a more interesting and easier start


INPUT
OUTPUT


INP
OUTPUT


OUTPUT

Use

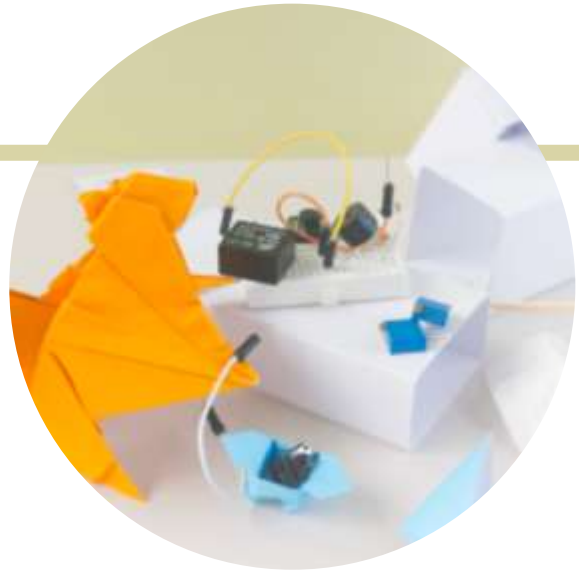
Run provided turtle code to draw a square

Modify

Modify the code to draw a square with different size and color

Create

Write code that draws a different shape



Use

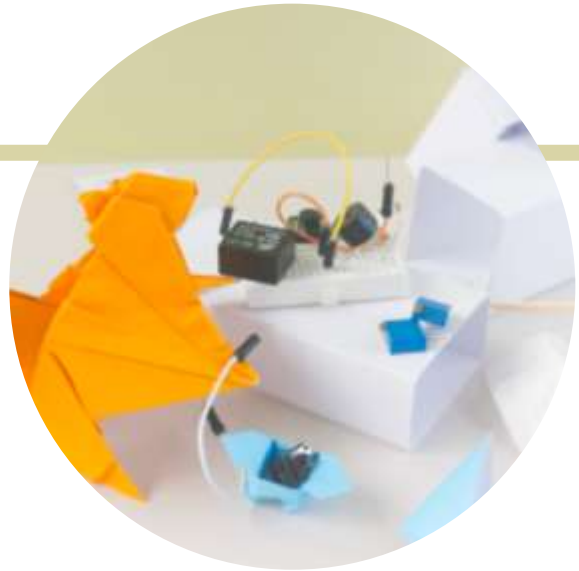
Use provided code to solve a problem

Modify

Modify the code to solve a modified problem

Create

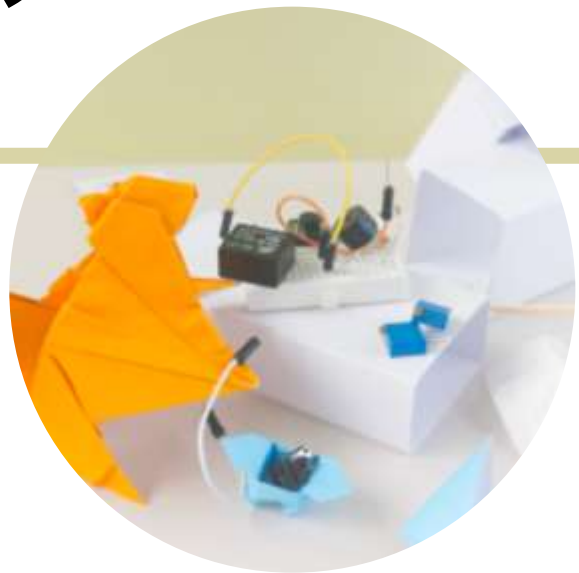
Write new code to solve a new problem



Use

Modify

Create



With UMC, the learner can get to the point earlier

```
import json
import matplotlib.pyplot as plt

with open("wolf_pair_count.json") as f:
    wolf_data = json.load(f)
wolf_count = wolf_data["count"]

max_population = 0
year_of_max_population = 0

for year, population in wolf_count.items():
    if population > max_population:
        max_population = population
        year_of_max_population = year

plt.plot(wolf_count.keys(), wolf_count.values())
plt.plot(year_of_max_population, max_population, 'o')
plt.axvline(year_of_max_population, linestyle='--')
plt.xlabel("Year")
plt.ylabel("Number of mated wolf pairs")
plt.title("Wolf population in Norway and Sweden")
plt.show()
```

With UMC, the learner can get to the point earlier

a) Download the start-code and data files and place them in a folder on your computer and run the code. What does the yellow dot and dashed line in the plot represent?

```
import json
import matplotlib.pyplot as plt

with open("wolf_pair_count.json") as f:
    wolf_data = json.load(f)
wolf_count = wolf_data["count"]

max_population = 0
year_of_max_population = 0

for year, population in wolf_count.items():
    if population > max_population:
        max_population = population
        year_of_max_population = year

plt.plot(wolf_count.keys(), wolf_count.values())
plt.plot(year_of_max_population, max_population, 'o')
plt.axvline(year_of_max_population, linestyle='--')
plt.xlabel("Year")
plt.ylabel("Number of mated wolf pairs")
plt.title("Wolf population in Norway and Sweden")
plt.show()
```

With UMC, the learner can get to the point earlier

```
import json
import matplotlib.pyplot as plt

with open("wolf_pair_count.json") as f:
    wolf_data = json.load(f)
wolf_count = wolf_data["count"]

max_population = 0
year_of_max_population = 0

for year, population in wolf_count.items():
    if population > max_population:
        max_population = population
        year_of_max_population = year

plt.plot(wolf_count.keys(), wolf_count.values())
plt.plot(year_of_max_population, max_population, 'o')
plt.axvline(year_of_max_population, linestyle='--')
plt.xlabel("Year")
plt.ylabel("Number of mated wolf pairs")
plt.title("Wolf population in Norway and Sweden")
plt.show()
```

a) Download the start-code and data files and place them in a folder on your computer and run the code. What does the yellow dot and dashed line in the plot represent?

b) Modify the code so the yellow dot shows the minimum population and the dashed line shows the year of the minimum population.

With UMC, the learner can get to the point earlier

```
import json
import matplotlib.pyplot as plt

with open("wolf_pair_count.json") as f:
    wolf_data = json.load(f)
wolf_count = wolf_data["count"]

max_population = 0
year_of_max_population = 0

for year, population in wolf_count.items():
    if population > max_population:
        max_population = population
        year_of_max_population = year

plt.plot(wolf_count.keys(), wolf_count.values())
plt.plot(year_of_max_population, max_population, 'o')
plt.axvline(year_of_max_population, linestyle='--')
plt.xlabel("Year")
plt.ylabel("Number of mated wolf pairs")
plt.title("Wolf population in Norway and Sweden")
plt.show()
```

a) Download the start-code and data files and place them in a folder on your computer and run the code. What does the yellow dot and dashed line in the plot represent?

b) Modify the code so the yellow dot shows the minimum population and the dashed line shows the year of the minimum population.

c) Modify the code so the yellow dot shows the year where the population decreased the most compared to the previous year. (Hint: Start by creating a variable `previous_population = 0` that you set equal to `population` at the end of each iteration

With UMC, the learner can get to the point earlier

```
import json
import matplotlib.pyplot as plt

with open("wolf_pair_count.json") as f:
    wolf_data = json.load(f)
wolf_count = wolf_data["count"]

max_population = 0
year_of_max_population = 0

for year, population in wolf_count.items():
    if population > max_population:
        max_population = population
        year_of_max_population = year

plt.plot(wolf_count.keys(), wolf_count.values())
plt.plot(year_of_max_population, max_population, 'o')
plt.axvline(year_of_max_population, linestyle='--')
plt.xlabel("Year")
plt.ylabel("Number of mated wolf pairs")
plt.title("Wolf population in Norway and Sweden")
plt.show()
```

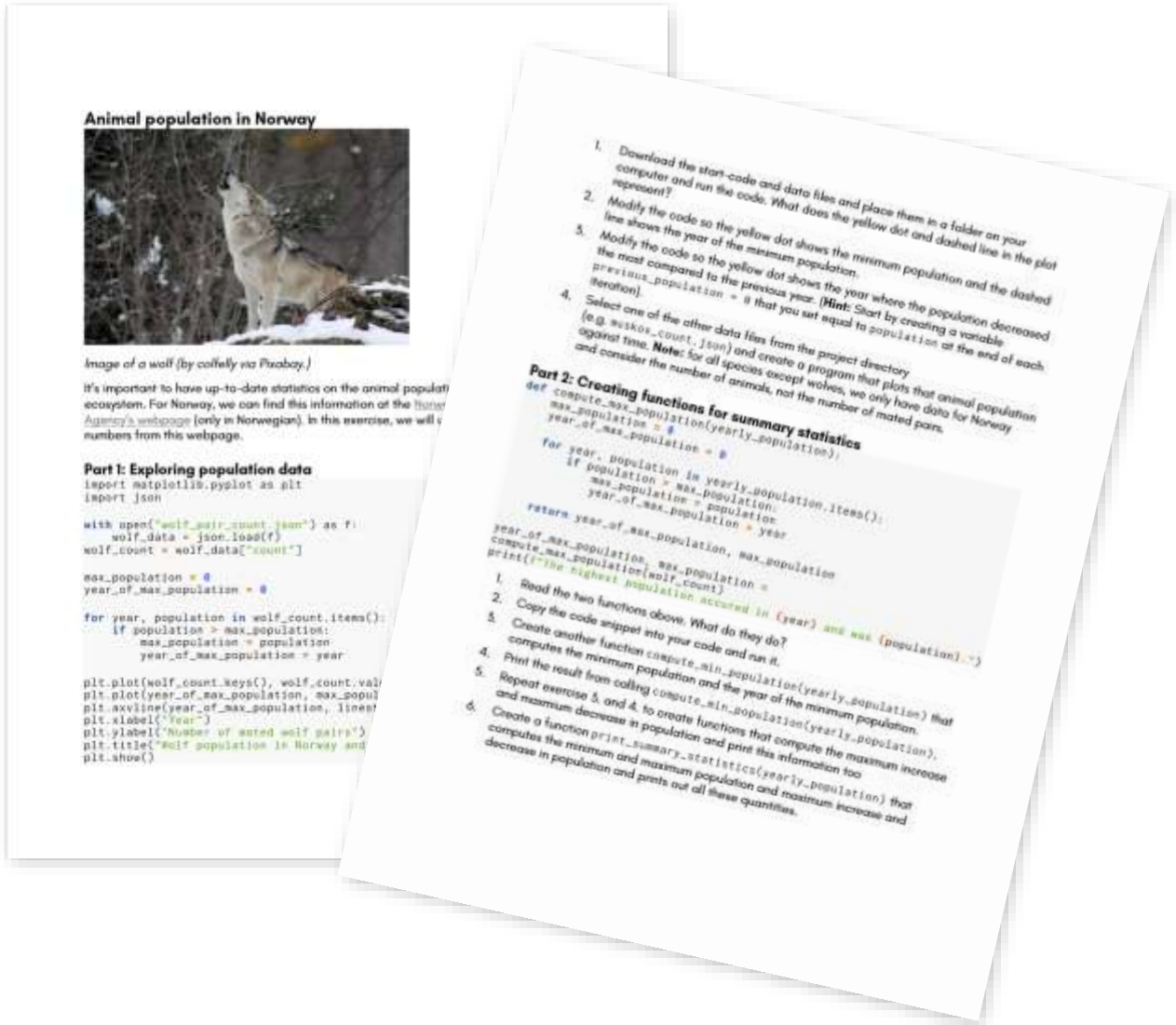
a) Download the start-code and data files and place them in a folder on your computer and run the code. What does the yellow dot and dashed line in the plot represent?

b) Modify the code so the yellow dot shows the minimum population and the dashed line shows the year of the minimum population.

c) Modify the code so the yellow dot shows the year where the population decreased the most compared to the previous year. (Hint: Start by creating a variable `previous_population = 0` that you set equal to `population` at the end of each iteration

d) Select one of the other data files from the project directory (e.g. `muskox_count.json`) and create a program that plots that animal population against time. Note: for all species except wolves, we only have data for Norway and consider the number of animals, not the number of mated pairs.

With UMC, the learner can engage in relevant exercises from the beginning



Animal population in Norway

Image of a wolf (by colfelly via Pixabay.)

It's important to have up-to-date statistics on the animal populati
ecosystem. For Norway, we can find this information at the Norw
Agency's webpage (only in Norwegian). In this exercise, we will
numbers from this webpage.

Part 1: Exploring population data

```
import matplotlib.pyplot as plt
import json

with open("wolf_pair_count.json") as f:
    wolf_data = json.load(f)
wolf_count = wolf_data["count"]

max_population = 0
year_of_max_population = 0

for year, population in wolf_count.items():
    if population > max_population:
        max_population = population
        year_of_max_population = year

plt.plot(wolf_count.keys(), wolf_count.val
plt.plot(year_of_max_population, max_popul
plt.axvline(year_of_max_population, lines
plt.xlabel("Year")
plt.ylabel('Number of mated wolf pairs')
plt.title("Wolf population in Norway and
plt.show()
```

1. Download the start-code and data files and place them in a folder on your computer and run the code. What does the yellow dot and dashed line in the plot represent?
2. Modify the code so the yellow dot shows the minimum population and the dashed line shows the year of the minimum population.
5. Modify the code so the yellow dot shows the year where the population decreased the most compared to the previous year. (**Hint:** Start by creating a variable `previous_population = 0` that you set equal to `population` at the end of each iteration).
4. Select one of the other data files from the project directory (e.g. `moskus_count.json`) and create a program that plots that animal population against time. **Note:** for all species except wolves, we only have data for Norway and consider the number of animals, not the number of mated pairs.

Part 2: Creating functions for summary statistics

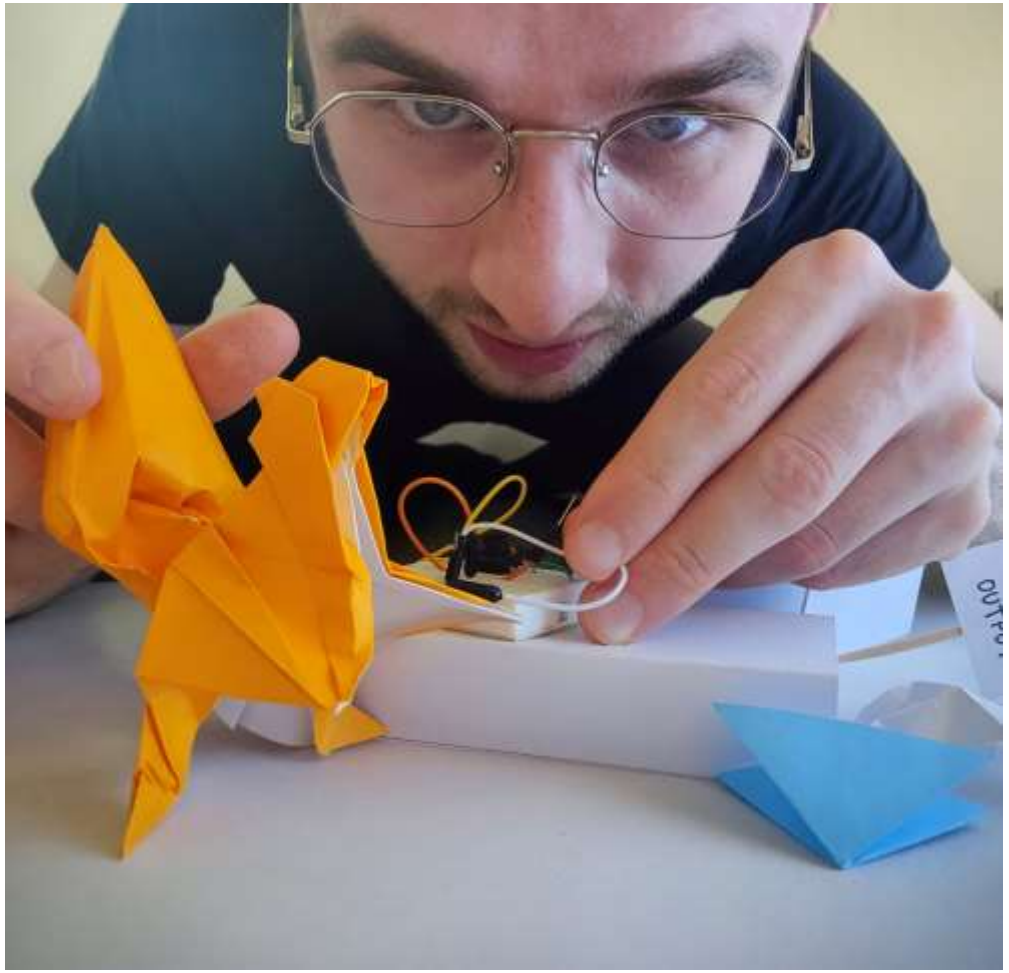
```
def compute_max_population(yearly_population):
    max_population = 0
    year_of_max_population = 0
    for year, population in yearly_population.items():
        if population > max_population:
            max_population = population
            year_of_max_population = year
    return year_of_max_population, max_population

year_of_max_population, max_population =
compute_max_population(wolf_count)
print(f"The highest population occured in {year} and was {population}.")
```

1. Read the two functions above. What do they do?
2. Copy the code snippet into your code and run it.
5. Create another function `compute_min_population(yearly_population)` that computes the minimum population and the year of the minimum population.
4. Print the result from calling `compute_min_population(yearly_population)`.
5. Repeat exercise 5, and 4, to create functions that compute the maximum increase and maximum decrease in population and print this information too
6. Create a function `print_summary_statistics(yearly_population)` that computes the minimum and maximum population and maximum increase and decrease in population and prints out all these quantities.

Here are some great resources if you want to dive deeper

- Miller, K, Lasry, N, Chu, K and Mazur, E. (2013). Role of physics lecture demonstrations in conceptual learning. Physical review special topics-physics education research, 9(2).
- Steele-Johnson, D, & Kalinoski, ZT. (2014). Error framing effects on performance: Cognitive, motivational, and affective pathways. The Journal of psychology, 148(1).
- Brown, NC and Wilson, G. (2018). Ten quick tips for teaching programming. PLoS Computational Biology, 14(4).
- Wilson, G. (2019). Ten quick tips for creating an effective lesson. PLoS Computational Biology, 15(4).
- Patitsas, E et al. (2019). Evidence that computer science grades are not bimodal. Communications of the ACM, 63(1).
- Nederbragt, A et al. (2020). Ten quick tips for teaching with participatory live coding. PLoS Computational Biology, 16(9).
- Lytle, N et al. (2019). Use, Modify, Create: Comparing Computational Thinking Lesson Progressions for STEM Classes. Proc 2019 ACM Conf Innov Technol Comput Sci Educ.

You can find all the examples together with the slides at GitHub.com/MarieRoald/PyConIt23



Special thanks to my origami artist Yngve Mardal Moe!

To summarise: coding is a journey, talking code is a skill that takes practise and UMC help beginners get to the fun faster

To summarise: coding is a journey, talking code is a skill that takes practise and UMC help beginners get to the fun faster

Questions?

GitHub.com/MarieRoald/PyConIt23

To summarise: coding is a journey, talking code is a skill that takes practise and UMC help beginners get to the fun faster

Questions?