# Python Programming for Absolute Beginners

Python Programming for Absolute Beginners
Python Programming for Absolute Beginners

**Ingo Kleiber @ 39c3**

POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES

# Today‘s Goals

*You will be able to …*

- describe what programming essentially is about.
- name and describe some basic programming terminology.
- define requirements and design simple software.
- model simple problems in terms of data structures and basic algorithms.
- write basic scripts in Python in order to solve specific problems.
- describe the role of LLMs in modern software development.

# Programming

“It’s difficult not to have a love/hate relationship with computer programming if you have any relationship with it at all.”

(Rosenberg 2006)

# Code Along!

**You can code along!**

- Go to: https://github.com/IngoKl/39c3-python-programming-for-absolute-beginners
- Launch the "binder"

https://t1p.de/39c3python

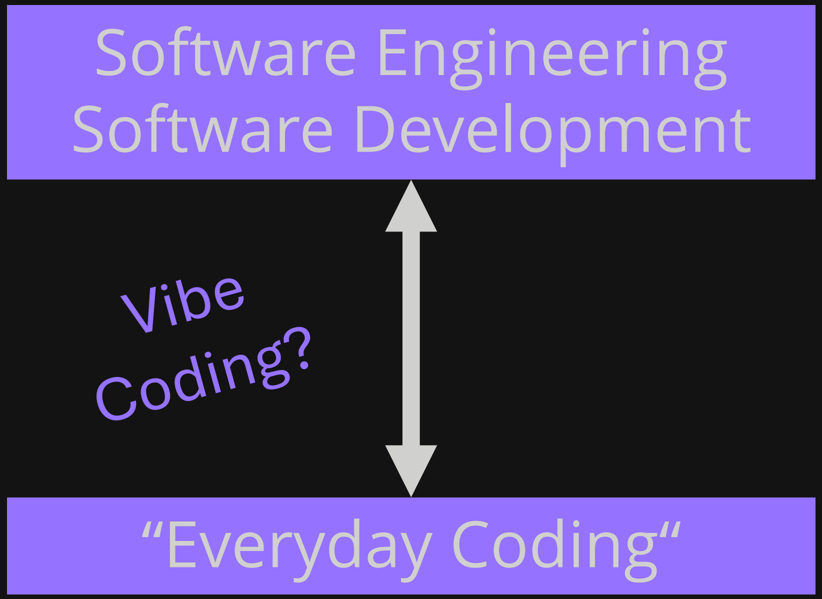
# Disclaimer

- This is an *introduction* to both programming and Python for those without any prior experience.
- Both the approaches as well as the code do not follow "best practices" but are optimized for understanding and learning.

THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111
THIS IS WRONG!!!!11111

# Programming is ...

- instructing machines and computers
- problem solving
- thinking differently (computationally)
- modeling problems and other things
- an art
- fun
- ...

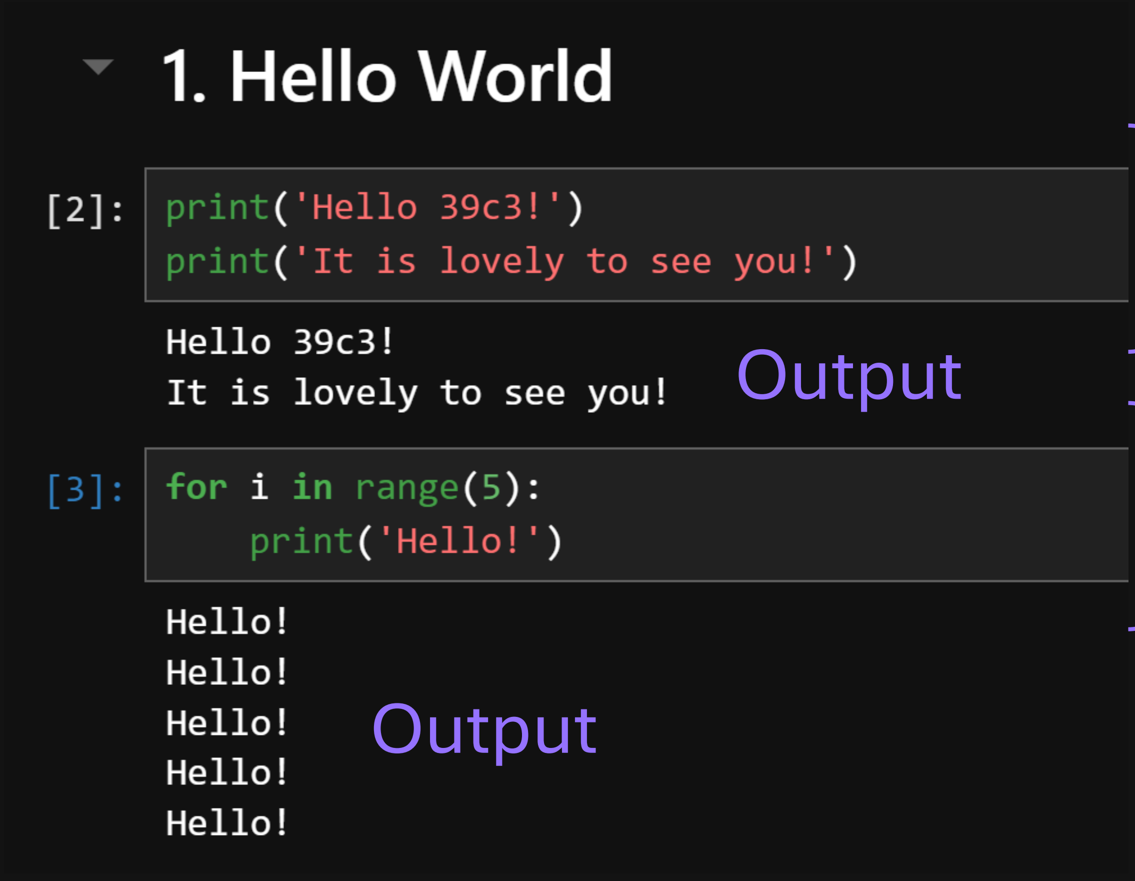


# The Python Programming Language

**Python**

- is one of hundreds of programming languages.
- is free, open, and available on almost any platform
- is modern and widely used
- has a great community
- is relatively easy to learn but hard to master

# What does Code Look Like?



## 1. Hello World

```
[2]: print('Hello 39c3!')
     print('It is lovely to see you!')
```

```
Hello 39c3!
It is lovely to see you!
```

Output

```
[3]: for i in range(5):
         print('Hello!')
```

```
Hello!
Hello!
Hello!
Hello!
Hello!
```

Output

Two **lines of code**, executed in order. Each line is one **command**.

A **block of code** with multiple indented lines. The block is a **unit of functionality**.

POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES

# A Real Problem
# The Pizza Problem!

At **Sue's Pizza**, you can order three types of pizza:

26cm

30cm

46x33cm

**Small** for 4.80

**Large** for 5.50

**Party** for 13.00

# A Real Problem
# The Pizza Problem!

At **Sue's Pizza**, you can order three types of pizza:

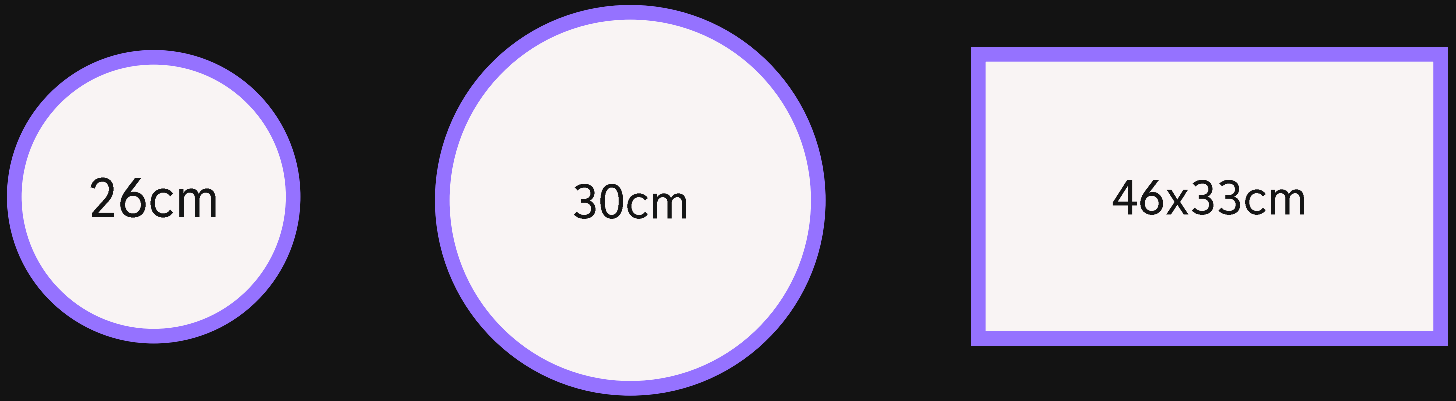
$A = \pi \times r^2$

$A = a \times b$

$A_S = 531 cm^2 \rightarrow 111/€$

$A_L = 707 cm^2 \rightarrow 128/€$ ✓

$A_P = 1518^2 \rightarrow 116/€$



**Small** for 4.80

**Large** for 5.50

**Party** for 13.00

# A Real Problem
# The Pizza Problem!

## User Story

As a pizza buyer, I want to be able to find the best pizza-to-price ration given a set of pizza sizes (e.g., 30cm,  46x33cm) and prices in order to maximize my pizza consumption given my measly budget.

Assumptions:

- More pizza for less money is always better
- Pizza shape is irrelevant

# A Real Problem
# The Pizza Problem!

For every (coding) **problem**, there are various solutions and approaches ...

In **programming**, some common measures for **good solutions** are:

*(1)* simplicity *(2)* reusability *(3)* testability *(4)* understandability

*(5)* compliance *(6)* maintainability *(7)* efficiency *(8)* robustness

→ We're aiming for a solution which is **just good enough**!

# A Real Problem
# The Pizza Problem!

- Determine sizes, prices, and shapes of n pizzas
- For each pizza, determine its area (A)
- For each pizza, calculate the pizza to Euro ration (PTER)
- Determine the best PTER

POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES

# Programming Basics

In order to do this, we are going to need **some basics** ...

- **Variables** = a container to put data in (r = 13)
- **Lists** = a list of data-things (e.g., variables) (l = [1,2,3])
- **Loops** = repeating something until some condition is met
- **If-Constructions** = do something if some condition is met
- **Functions** = a unit of code that completes a specific task
- **Dictionaries**

# Variables and Lists

```
a = 13
b = ‘Hello 39c3’
c = 39.3
```

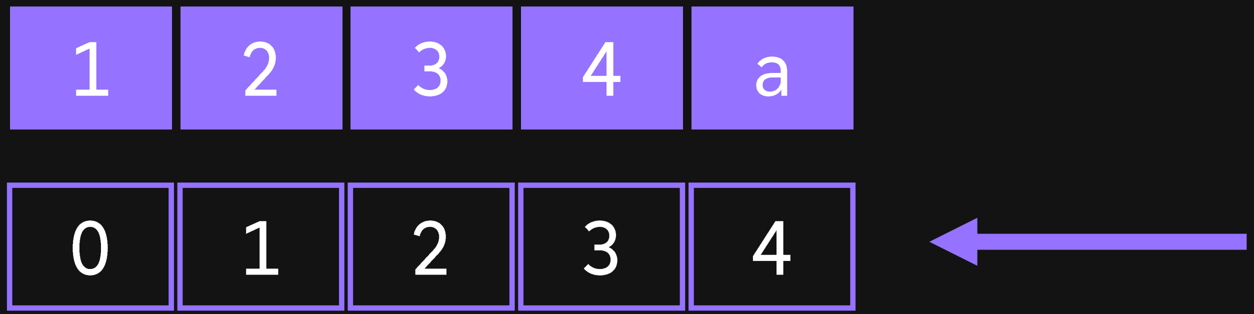
Three variables (containers) of three different types: *integer*, *string*, and *float*

```
l = [1, 2, 3, 4, a]
```

A list (named l) containing 4 integers and the variable a.

# Variables and Lists

l = [1, 2, 3, 4, a]



| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | a |
| 0 | 1 | 2 | 3 | 4 |

We always start counting at 0

l[0] → 1
l[3] → 4

# Variables and Lists

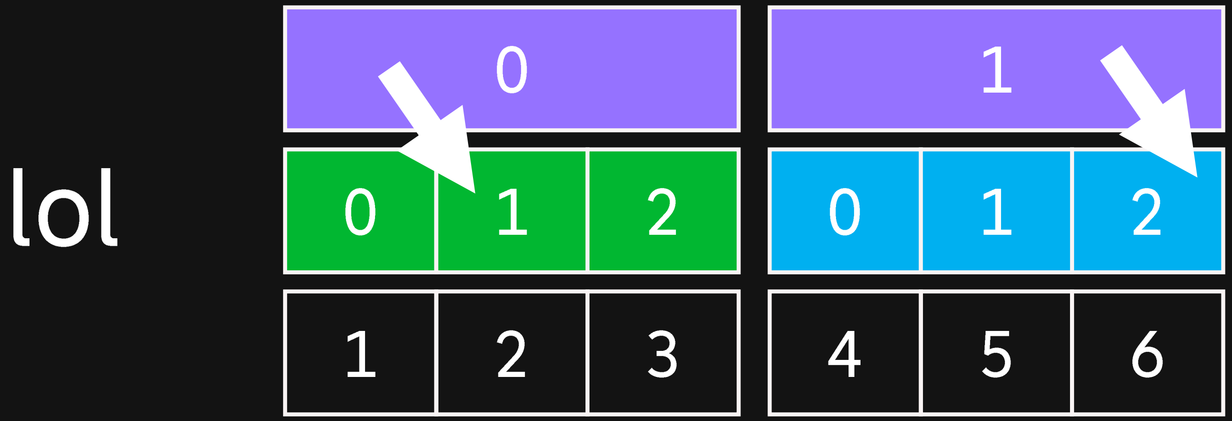
la = [1, 2, 3]
lb = [4, 5, 6]

lol = [la, lb] → [ [1,2,3], [4,5,6] ]

A list of lists

lol[0][1] → 2

# Variables and Lists



lol

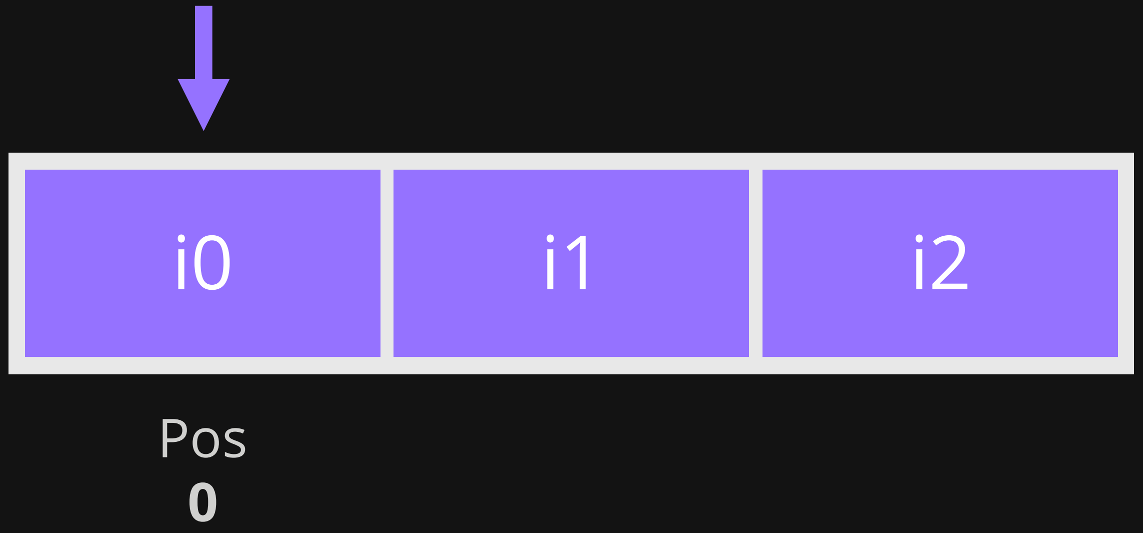
| 0 | | | 1 | | |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 0 | 1 | 2 |
| 1 | 2 | 3 | 4 | 5 | 6 |

lol[0][1] → 2

lol[1][2] → 6

# Loops

```
box = ['i0', 'i1', 'i2']
for item in box:
        print(item)
```



Step: **0**

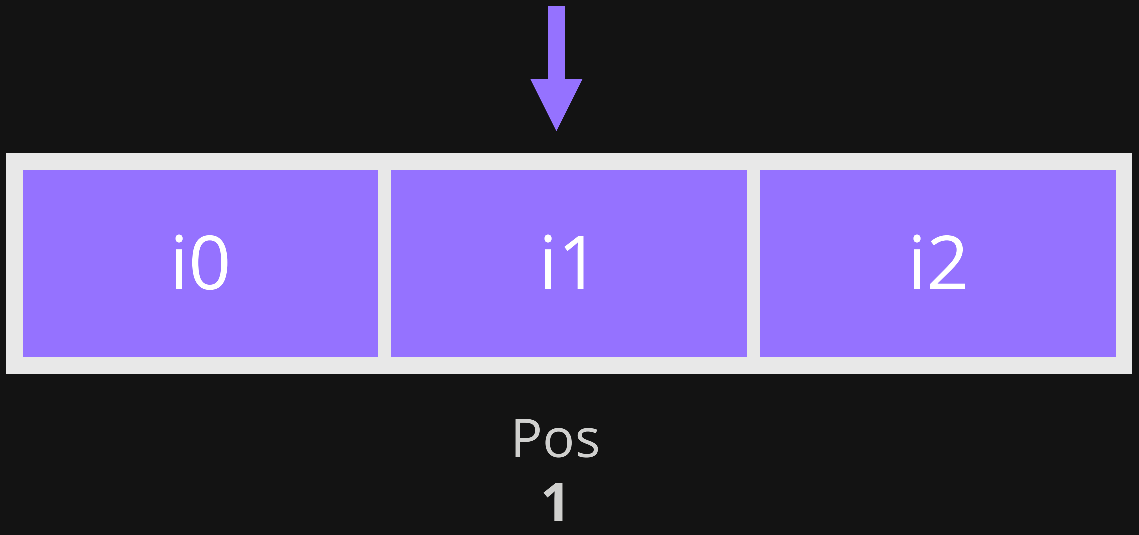
| i0 | i1 | i2 |
|---|---|---|

Pos
**0**

item: **i0**

# Loops

```
box = ['i0', 'i1', 'i2']
for item in box:
        print(item)
```



Step: **0**

i0 i1 i2

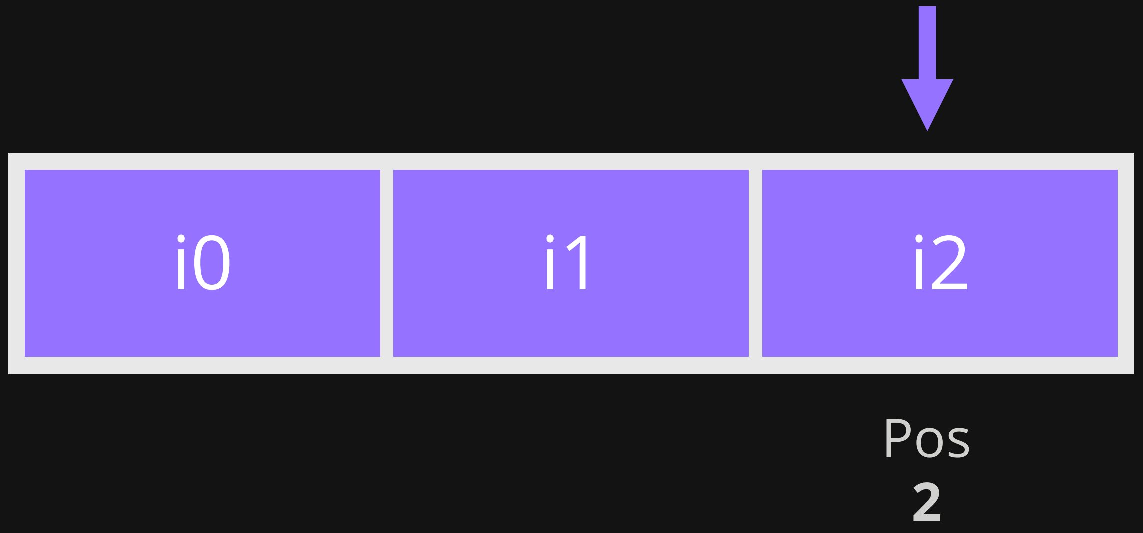
item: **i1**

Pos

**1**

# Loops

```
box = ['i0', 'i1', 'i2']
for item in box:
        print(item)
```



Step: **0**

item: **i2**

Pos
**2**

# If-Construction

```
a = 10

if a > 15:
    print (‘A is greater than 15’)
else:
    print (‘A is not greater than 15’)
```

# Functions

Two parameters which we pass to the function.

```
def add(a, b):
    result = a + b

    return result
```

What the function returns

```
add(5, 10) → 15
add(2, 2) → 4
```

POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES

# Modeling Pizza as a List

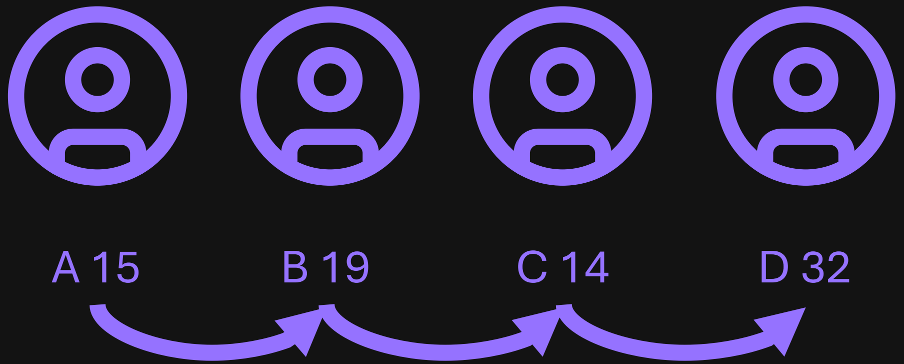
Type Size Price Shape

```
ps = [‘small’, [26, 0], 4.80, ‘circle’]
```

Shape is, implicitly, encoded here as well!

# Finding the Ideal Pizza

Imagine we wanted to **find the youngest and the oldest person** in the room ...

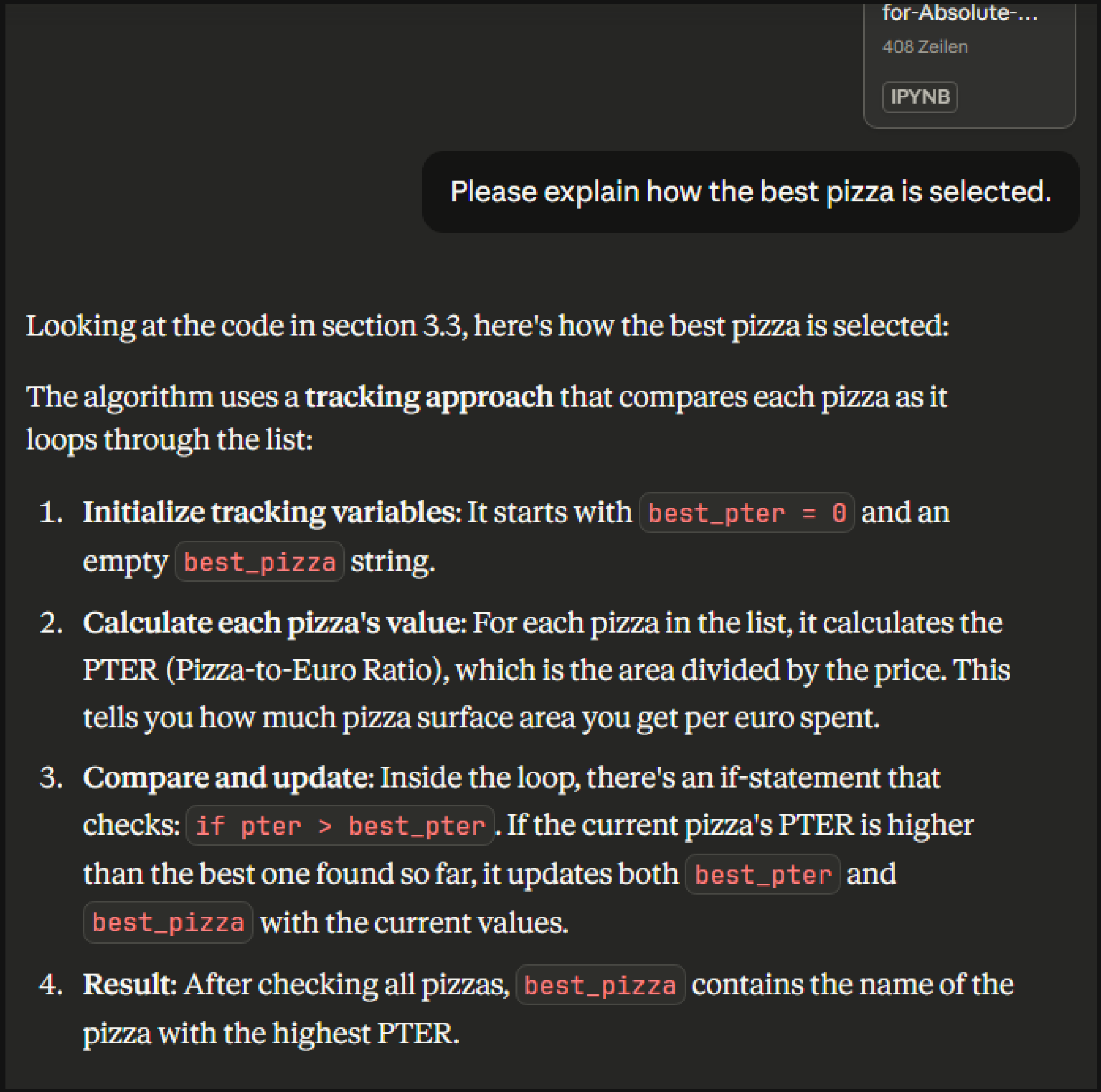


Youngest: A15, A15, C14, C14
Oldest: A15, B19, B19, D32

POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES

# AI and Software Development
## Explaining/Understanding Code

Here, Claude (Sonnet 4.5) has access to the whole codebase and explains the actual selection algorithm.

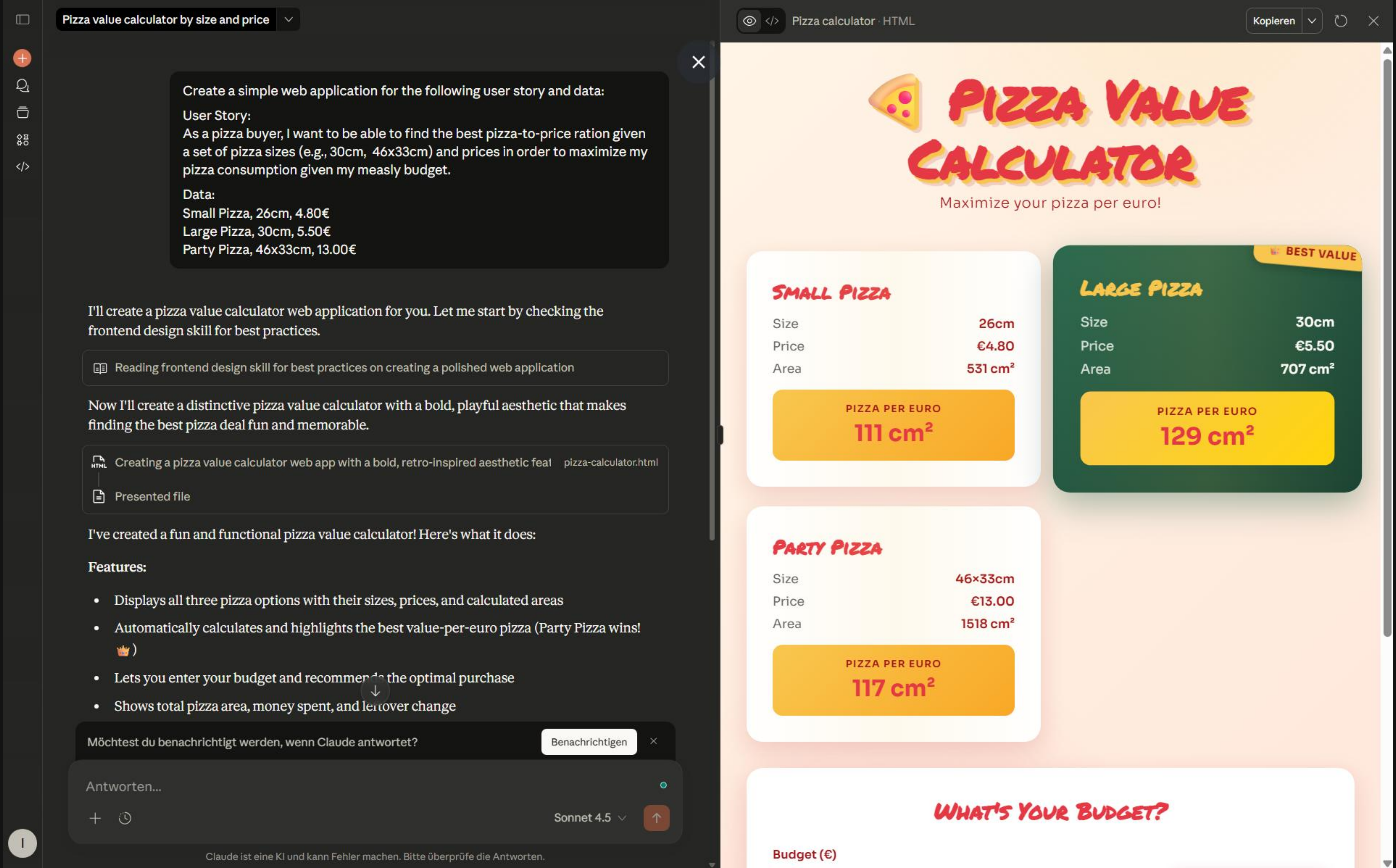


Please explain how the best pizza is selected.

Looking at the code in section 3.3, here's how the best pizza is selected:

The algorithm uses a **tracking approach** that compares each pizza as it loops through the list:

1. **Initialize tracking variables:** It starts with `best_pter = 0` and an empty `best_pizza` string.
2. **Calculate each pizza's value:** For each pizza in the list, it calculates the PTER (Pizza-to-Euro Ratio), which is the area divided by the price. This tells you how much pizza surface area you get per euro spent.
3. **Compare and update:** Inside the loop, there's an if-statement that checks: `if pter > best_pter`. If the current pizza's PTER is higher than the best one found so far, it updates both `best_pter` and `best_pizza` with the current values.
4. **Result:** After checking all pizzas, `best_pizza` contains the name of the pizza with the highest PTER.



Create a simple web application for the following user story and data:

User Story:
As a pizza buyer, I want to be able to find the best pizza-to-price ration given a set of pizza sizes (e.g., 30cm, 46x33cm) and prices in order to maximize my pizza consumption given my measly budget.

Data:
Small Pizza, 26cm, 4.80€
Large Pizza, 30cm, 5.50€
Party Pizza, 46x33cm, 13.00€

I'll create a pizza value calculator web application for you. Let me start by checking the frontend design skill for best practices.

Reading frontend design skill for best practices on creating a polished web application

Now I'll create a distinctive pizza value calculator with a bold, playful aesthetic that makes finding the best pizza deal fun and memorable.

Creating a pizza value calculator web app with a bold, retro-inspired aesthetic feat pizza-calculator.html

Presented file

I've created a fun and functional pizza value calculator! Here's what it does:

**Features:**

- Displays all three pizza options with their sizes, prices, and calculated areas
- Automatically calculates and highlights the best value-per-euro pizza (Party Pizza wins! 👑)
- Lets you enter your budget and recommen the optimal purchase
- Shows total pizza area, money spent, and le over change

Möchtest du benachrichtigt werden, wenn Claude antwortet?

Claude ist eine KI und kann Fehler machen. Bitte überprüfe die Antworten.

# Pizza Value Calculator

Maximize your pizza per euro!

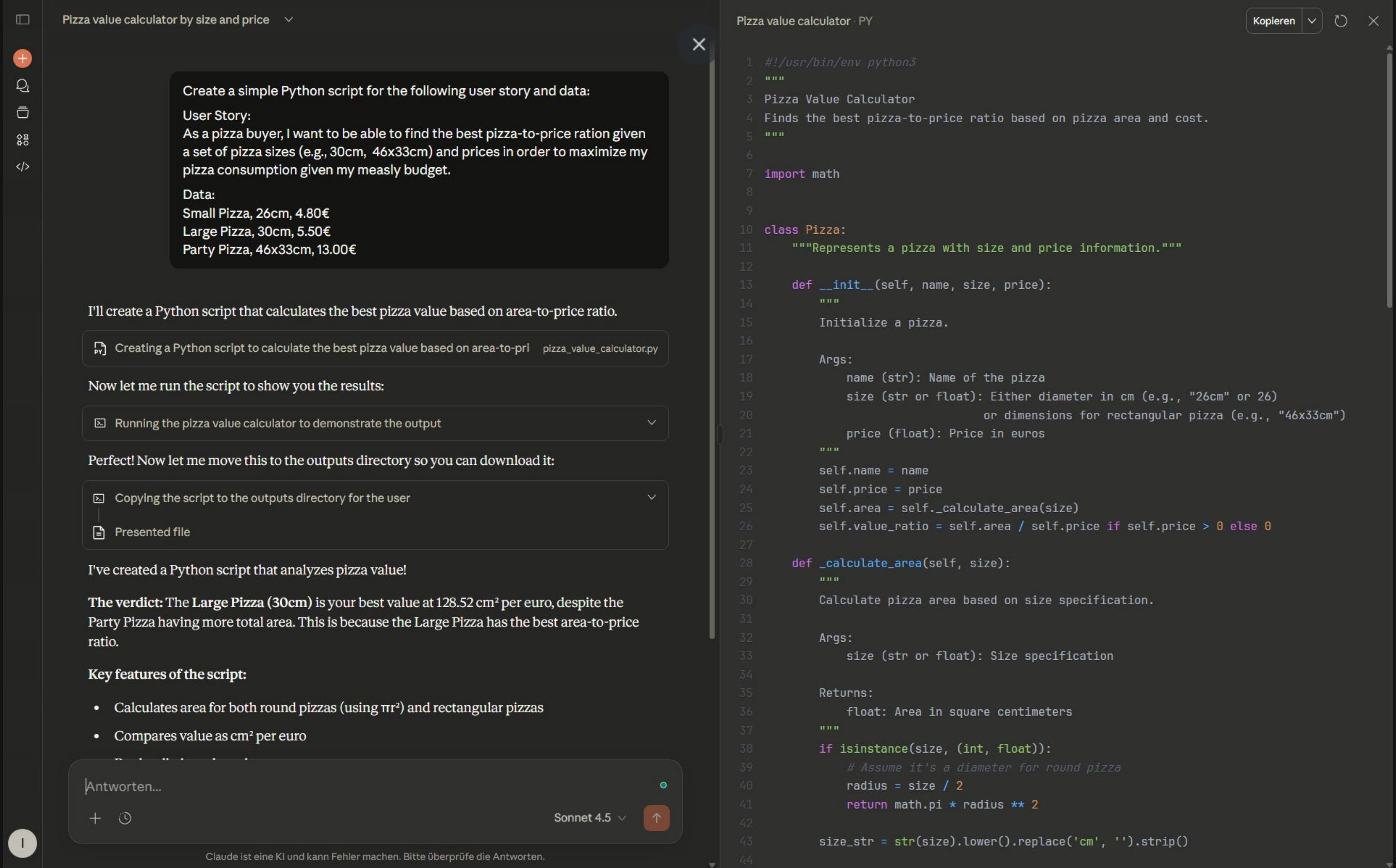
**Small Pizza**
Size 26cm
Price €4.80
Area 531 cm²
PIZZA PER EURO
111 cm²

BEST VALUE
**Large Pizza**
Size 30cm
Price €5.50
Area 707 cm²
PIZZA PER EURO
129 cm²

**Party Pizza**
Size 46×33cm
Price €13.00
Area 1518 cm²
PIZZA PER EURO
117 cm²

## What's Your Budget?

Budget (€)


Pizza value calculator by size and price
Pizza value calculator · PY
Kopieren

Create a simple Python script for the following user story and data:

User Story:
As a pizza buyer, I want to be able to find the best pizza-to-price ration given a set of pizza sizes (e.g., 30cm, 46x33cm) and prices in order to maximize my pizza consumption given my measly budget.

Data:
Small Pizza, 26cm, 4.80€
Large Pizza, 30cm, 5.50€
Party Pizza, 46x33cm, 13.00€

I'll create a Python script that calculates the best pizza value based on area-to-price ratio.

Creating a Python script to calculate the best pizza value based on area-to-pri pizza_value_calculator.py

Now let me run the script to show you the results:

Running the pizza value calculator to demonstrate the output

Perfect! Now let me move this to the outputs directory so you can download it:

Copying the script to the outputs directory for the user

Presented file

I've created a Python script that analyzes pizza value!

**The verdict:** The **Large Pizza (30cm)** is your best value at 128.52 cm² per euro, despite the Party Pizza having more total area. This is because the Large Pizza has the best area-to-price ratio.

**Key features of the script:**

- Calculates area for both round pizzas (using πr²) and rectangular pizzas
- Compares value as cm² per euro

Claude ist eine KI und kann Fehler machen. Bitte überprüfe die Antworten.

```python
#!/usr/bin/env python3
"""
Pizza Value Calculator
Finds the best pizza-to-price ratio based on pizza area and cost.
"""

import math


class Pizza:
    """Represents a pizza with size and price information."""

    def __init__(self, name, size, price):
        """
        Initialize a pizza.

        Args:
            name (str): Name of the pizza
            size (str or float): Either diameter in cm (e.g., "26cm" or 26)
                                 or dimensions for rectangular pizza (e.g., "46x33cm")
            price (float): Price in euros
        """
        self.name = name
        self.price = price
        self.area = self._calculate_area(size)
        self.value_ratio = self.area / self.price if self.price > 0 else 0

    def _calculate_area(self, size):
        """
        Calculate pizza area based on size specification.

        Args:
            size (str or float): Size specification

        Returns:
            float: Area in square centimeters
        """
        if isinstance(size, (int, float)):
            # Assume it's a diameter for round pizza
            radius = size / 2
            return math.pi * radius ** 2

        size_str = str(size).lower().replace('cm', '').strip()

```

POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES
POWER CYCLES

# Bonus Exercises

- How can we find the ideal (i.e., best priced) combination of pizzas for a given area that is being requested?
- What if we were looking to optimize for as much/little crust as possible?
- What about a second/third size dimension (i.e., height)?

# Contact

Ingo Kleiber

ikleiber@gmail.com

@ikleiber:uni.koeln

/in/@ingokleiber

https://kleiber.me

# Works Cited

Rosenberg, Scott. 2006. Dreaming in Code. New York: Three Rivers Press.