TUTORIAL 5: USING LLMS TO STRUCTURE DATA

Creating Business Value with Generative AI
Fall 2025

AARHUS BSS
DEPARTMENT OF MANAGEMENT
AARHUS UNIVERSITY

Magnus Bender

PLAN FOR TODAY

PLAN FOR TODAY

- Slides available online in Brightspace

1. Presentation

PLAN FOR TODAY

- Slides available online in Brightspace

1. Presentation
   - Annotating Python code with data types

PLAN FOR TODAY

- Slides available online in Brightspace

1. Presentation
   - Annotating Python code with data types
   - Custom data types in Python
   - Pydantic for data type enforcement

PLAN FOR TODAY

- Slides available online in Brightspace

1. Presentation
   - Annotating Python code with data types
   - Custom data types in Python
   - Pydantic for data type enforcement
   - Using OpenAI API together with pydantic

PLAN FOR TODAY

- Slides available online in Brightspace

1. Presentation
   - Annotating Python code with data types
   - Custom data types in Python
   - Pydantic for data type enforcement
   - Using OpenAI API together with pydantic
2. Use OpenAI API for analyzing and retrieving **structured** data on uCloud

PLAN FOR TODAY

- Slides available online in Brightspace

1. Presentation
   - Annotating Python code with data types
   - Custom data types in Python
   - Pydantic for data type enforcement
   - Using OpenAI API together with pydantic
2. Use OpenAI API for analyzing and retrieving **structured** data on uCloud
   i. Individually try out different approaches

PLAN FOR TODAY

- Slides available online in Brightspace

1. Presentation
   - Annotating Python code with data types
   - Custom data types in Python
   - Pydantic for data type enforcement
   - Using OpenAI API together with pydantic
2. Use OpenAI API for analyzing and retrieving **structured** data on uCloud
   i. Individually try out different approaches
   ii. Apply *your* small **use-case** example

PLAN FOR TODAY

- Slides available online in Brightspace

1. Presentation
   - Annotating Python code with data types
   - Custom data types in Python
   - Pydantic for data type enforcement
   - Using OpenAI API together with pydantic
2. Use OpenAI API for analyzing and retrieving **structured** data on uCloud
   i. Individually try out different approaches
   ii. Apply *your* small **use-case** example
      - Create pydantic data model

PLAN FOR TODAY

- Slides available online in Brightspace

1. Presentation
   - Annotating Python code with data types
   - Custom data types in Python
   - Pydantic for data type enforcement
   - Using OpenAI API together with pydantic
2. Use OpenAI API for analyzing and retrieving **structured** data on uCloud
   i. Individually try out different approaches
   ii. Apply *your* small **use-case** example
      - Create pydantic data model
      - Use OpenAI API for analyzing and retrieving

TYPE HINTS AND CUSTOM DATA TYPES

- Model structured data in Python
- Pydantic for enforcing data types

AARHUS BSS
DEPARTMENT OF MANAGEMENT
AARHUS UNIVERSITY

Magnus Bender

TYPE HINTS

```
text:str = "hello"
print(text)

number:int = 12
print(number)

decimal:int = 1.2
```

TYPE HINTS

```
text:str = "hello"
print(text)

number:int = 12
print(number)

decimal:int = 1.2
print(decimal)
```

TYPE HINTS

```
text:str = "hello"
print(text)

number:int = 12
print(number)

decimal:int = 1.2
print(decimal)
```

```
hello

12
```

TYPE HINTS

```
text:str = "hello"
print(text)
```

```
hello
```

```
number:int = 12
print(number)
```

```
12
```

```
decimal:int = 1.2
print(decimal)
```

```
1.2
```

TYPE HINTS

```
text:str = "hello"
print(text)                 hello

number:int = 12
print(number)               12

decimal:int = 1.2
print(decimal)              1.2

def str_repeat(s:str, n:int) -> str:
  return s * n

print(str_repeat("Hallo", 5))
print(str_repeat("Hallo", 2))
print(str_repeat(5, 5))
```

TYPE HINTS

```
text:str = "hello"
print(text)                 hello

number:int = 12
print(number)               12

decimal:int = 1.2
print(decimal)              1.2

def str_repeat(s:str, n:int) -> str:
  return s * n

print(str_repeat("Hallo", 5))       HalloHalloHalloHalloHallo
print(str_repeat("Hallo", 2))       HalloHallo
print(str_repeat(5, 5))
```

TYPE HINTS

```
text:str = "hello"
print(text)                 hello

number:int = 12
print(number)               12

decimal:int = 1.2
print(decimal)              1.2

def str_repeat(s:str, n:int) -> str:
  return s * n

print(str_repeat("Hallo", 5))       HalloHalloHalloHalloHallo
print(str_repeat("Hallo", 2))       HalloHallo
print(str_repeat(5, 5))             25
```

TYPE HINTS

```
text:str = "hello"
print(text)                 hello

number:int = 12
print(number)               12

decimal:int = 1.2
print(decimal)              1.2

def str_repeat(s:str, n:int) -> str:
  return s * n

print(str_repeat("Hallo", 5))     HalloHalloHalloHalloHallo
print(str_repeat("Hallo", 2))     HalloHallo
print(str_repeat(5, 5))           25
```

- Annotate a variable in Python with its data type
- Annotations can be added to variables, function parameters

! It is just a „Type Hint“, so the type is not enforced!

TYPE ENFORCEMENT

```python
from pydantic import validate_call

@validate_call
def str_repeat_safe(s:str, n:int) -> str:
  return s * n

print(str_repeat_safe("Hallo", 2))
print(str_repeat_safe(5, 5))
```

TYPE ENFORCEMENT

```python
from pydantic import validate_call

@validate_call
def str_repeat_safe(s:str, n:int) -> str:
  return s * n

print(str_repeat_safe("Hallo", 2))
print(str_repeat_safe(5, 5))
```

```
HalloHallo
```

TYPE ENFORCEMENT

```python
from pydantic import validate_call

@validate_call
def str_repeat_safe(s:str, n:int) -> str:
  return s * n

print(str_repeat_safe("Hallo", 2))
print(str_repeat_safe(5, 5))
```

```
HalloHallo
1 validation error for str_repeat_safe
  Input should be a valid string [type=
    string_type, input_value=5, input_type=int]
    For further information visit
      https://errors.pydantic.dev/2.11/v/string_t
```

TYPE ENFORCEMENT

```python
from pydantic import validate_call

@validate_call
def str_repeat_safe(s:str, n:int) -> str:
  return s * n

print(str_repeat_safe("Hallo", 2))
print(str_repeat_safe(5, 5))

@validate_call
def divide(x:int|float, y:int|float) -> float:
  return x / y

print(divide(1, 2))
print(divide(1, 0.5))
print(divide(1, "Hello"))
```

```
HalloHallo
1 validation error for str_repeat_safe
  Input should be a valid string [type=
    string_type, input_value=5, input_type=int]
    For further information visit
      https://errors.pydantic.dev/2.11/v/string_t
```

TYPE ENFORCEMENT

```python
from pydantic import validate_call

@validate_call
def str_repeat_safe(s:str, n:int) -> str:
  return s * n

print(str_repeat_safe("Hallo", 2))
print(str_repeat_safe(5, 5))

@validate_call
def divide(x:int|float, y:int|float) -> float:
  return x / y

print(divide(1, 2))
print(divide(1, 0.5))
print(divide(1, "Hello"))
```

```
HalloHallo
1 validation error for str_repeat_safe
  Input should be a valid string [type=
    string_type, input_value=5, input_type=int]
    For further information visit
      https://errors.pydantic.dev/2.11/v/string_t
0.5
2.0
2 validation errors for divide
1.int
  Input should be a valid integer, unable to parse string as an
    For further information visit https://errors.pydantic.dev/2
1.float
```

TYPE ENFORCEMENT

- Need to use the package `pydantic` and to add a `@validate_call` decorator to a function
- Now the type hints are enforced
- Union types using `|`, e.g., integer or floating point represented by `int|float`

```
from pydantic import validate_call

@validate_call
def str_repeat_safe(s:str, n:int) -> str:
  return s * n

print(str_repeat_safe("Hallo", 2))
print(str_repeat_safe(5, 5))

@validate_call
def divide(x:int|float, y:int|float) -> float:
  return x / y

print(divide(1, 2))
print(divide(1, 0.5))
print(divide(1, "Hello"))
```

```
HalloHallo
1 validation error for str_repeat_safe
  Input should be a valid string [type=
    string_type, input_value=5, input_type=int]
    For further information visit
      https://errors.pydantic.dev/2.11/v/string_t
0.5
2.0
2 validation errors for divide
1.int
  Input should be a valid integer, unable to parse string as an
    For further information visit https://errors.pydantic.dev/2
1.float
```

HINTS FOR ADVANCED TYPES

HINTS FOR ADVANCED TYPES

- Tuples
  - Tuple of two items with different types (in this order)
  - Tuple of multiple items of same type

```
tuple[int, str] → (1, "a")
tuple[int, ...] → (1, 1), (1, 2, 3)
```

HINTS FOR ADVANCED TYPES

- Tuples
  - Tuple of two items with different types (in this order)
  - Tuple of multiple items of same type
- Lists
  - List of multiple items of same type
  - List of multiple items of multiple types

```
tuple[int, str] → (1, "a")
tuple[int, ...] → (1, 1), (1, 2, 3)
```

```
list[str]       → ["a", "b"]
list[str|int]   → [1, "b"], ["a", 2]
```

HINTS FOR ADVANCED TYPES

- Tuples
  - Tuple of two items with different types (in this order)
  - Tuple of multiple items of same type
- Lists
  - List of multiple items of same type
  - List of multiple items of multiple types
- Dictionaries
  - ✓ Define type of keys and of values
  - Keys and values are strings
  - Keys and values are different

```
tuple[int, str] → (1, "a")
tuple[int, ...] → (1, 1), (1, 2, 3)
```

```
list[str]       → ["a", "b"]
list[str|int]   → [1, "b"], ["a", 2]
```

```
dict[str, str]  → {"a" : "A", "b" : "B"}
dict[str, str|bool]  → {"a" : "A", "b" : True}
```

STRUCTURE YOUR DATA: CUSTOM DATA TYPES

https://simple.wikipedia.org/wiki/List_of_European_countries

STRUCTURE YOUR DATA: CUSTOM DATA TYPES

Name	Area	Population	Capital
Czech Republic	78,866	10,190,213	Prague
Denmark	43,094	5,529,888	Copenhagen
Estonia	45,226	1,282,963	Tallinn

https://simple.wikipedia.org/wiki/List_of_European_countries

STRUCTURE YOUR DATA: CUSTOM DATA TYPES

Name	Area	Population	Capital
Czech Republic	78,866	10,190,213	Prague
Denmark	43,094	5,529,888	Copenhagen
Estonia	45,226	1,282,963	Tallinn

```
[{
        "name":"Czech Republic",
        "area": 78866,
        "population": 10190213,
        "capital" : "Prague"
},{
        "name":"Denmark",
        "area": 43094,
        "population": 5529888,
        "capital" : "Copenhagen"
},{
        "name":"Estonia",
        "area": 45226,
        "population": 1282963,
        "capital" : "Tallinn"
}]
```

https://simple.wikipedia.org/wiki/List_of_European_countries

STRUCTURE YOUR DATA: CUSTOM DATA TYPES

- Dictionaries are nice and helpful to structure data
- Excel sheet can be represented by a list of dictionaries
  - Each row is represented by a dictionary in the list, the keys are the column name
- Difficult to maintain, lots of redundancies

Name	Area	Population	Capital
Czech Republic	78,866	10,190,213	Prague
Denmark	43,094	5,529,888	Copenhagen
Estonia	45,226	1,282,963	Tallinn

```
[{
        "name":"Czech Republic",
        "area": 78866,
        "population": 10190213,
        "capital" : "Prague"
},{
        "name":"Denmark",
        "area": 43094,
        "population": 5529888,
        "capital" : "Copenhagen"
},{
        "name":"Estonia",
        "area": 45226,
        "population": 1282963,
        "capital" : "Tallinn"
}]
```

https://simple.wikipedia.org/wiki/List_of_European_countries

PYTHON CLASSES

```
class Train():
 def __init__(self, n:int, s:str, d:str):
    self.number = n
    self.start = s
    self.destination = d
    self.stations = []
```

PYTHON CLASSES

```
class Train():
 def __init__(self, n:int, s:str, d:str):
    self.number = n
    self.start = s
    self.destination = d
    self.stations = []

  def add_station(self, s:str):
    self.stations.append(s)
```

PYTHON CLASSES

```
class Train():
 def __init__(self, n:int, s:str, d:str):
    self.number = n
    self.start = s
    self.destination = d
    self.stations = []

  def add_station(self, s:str):
    self.stations.append(s)

my_train = Train(60440, "Aarhus H", "Fredericia St.")
```

PYTHON CLASSES

```
class Train():
 def __init__(self, n:int, s:str, d:str):
    self.number = n
    self.start = s
    self.destination = d
    self.stations = []

  def add_station(self, s:str):
    self.stations.append(s)

my_train = Train(60440, "Aarhus H", "Fredericia St.")
```

```
print(type(my_train))
```

```
<class '__main__.Train'>
```

PYTHON CLASSES

```
class Train():
 def __init__(self, n:int, s:str, d:str):
    self.number = n
    self.start = s
    self.destination = d
    self.stations = []

  def add_station(self, s:str):
    self.stations.append(s)

my_train = Train(60440, "Aarhus H", "Fredericia St.")

my_train.add_station("Aarhus H")
my_train.add_station("Viby Jylland St.")
my_train.add_station("Skanderborg St.")
…
my_train.add_station("Fredericia St.")
```

```
print(type(my_train))
  <class '__main__.Train'>
```

PYTHON CLASSES

```
class Train():
 def __init__(self, n:int, s:str, d:str):
    self.number = n
    self.start = s
    self.destination = d
    self.stations = []

  def add_station(self, s:str):
    self.stations.append(s)

my_train = Train(60440, "Aarhus H", "Fredericia St.")

my_train.add_station("Aarhus H")
my_train.add_station("Viby Jylland St.")
my_train.add_station("Skanderborg St.")
…
my_train.add_station("Fredericia St.")
```

```
print(type(my_train))
  <class '__main__.Train'>

print(my_train)
  <__main__.Train object at 0x10a11af90>
```

PYTHON CLASSES

```
class Train():
 def __init__(self, n:int, s:str, d:str):
    self.number = n
    self.start = s
    self.destination = d
    self.stations = []

  def add_station(self, s:str):
    self.stations.append(s)

my_train = Train(60440, "Aarhus H", "Fredericia St.")

my_train.add_station("Aarhus H")
my_train.add_station("Viby Jylland St.")
my_train.add_station("Skanderborg St.")
…
my_train.add_station("Fredericia St.")
```

```
print(type(my_train))
  <class '__main__.Train'>

print(my_train)
  <__main__.Train object at 0x10a11af90>

print(my_train.stations,
my_train.number)
  ['Aarhus H', 'Viby Jylland St.',
  'Skanderborg St.', 'Fredericia St.']
  60440
```

PYTHON CLASSES

```
class Train():
 def __init__(self, n:int, s:str, d:str):
    self.number = n
    self.start = s
    self.destination = d
    self.stations = []

  def add_station(self, s:str):
    self.stations.append(s)

my_train = Train(60440, "Aarhus H", "Fredericia St.")

my_train.add_station("Aarhus H")
my_train.add_station("Viby Jylland St.")
my_train.add_station("Skanderborg St.")
…
my_train.add_station("Fredericia St.")
```

- Create a structure containing attributes (variables) and methods (code)
- Values of variables are different for each instance
- The class provides a new *data type*

```
print(type(my_train))
  <class '__main__.Train'>

print(my_train)
  <__main__.Train object at 0x10a11af90>

print(my_train.stations,
my_train.number)
  ['Aarhus H', 'Viby Jylland St.',
  'Skanderborg St.', 'Fredericia St.']
  60440
```

PYTHON DATA CLASSES

- Similar to a *normal* class
- Focussed on representing data objects
- Define by giving
  - A name of the data class/ type, e.g., `MyStructuredData`
  - Add the attributes/ variables/ item the to contain, e.g., `name, unit_price, quantity_on_hand`

PYTHON DATA CLASSES

```python
from dataclasses import dataclass

@dataclass
class MyStructuredData:
  name: str
  unit_price: float
  quantity_on_hand: int = 0
```

- Similar to a *normal* class
- Focussed on representing data objects
- Define by giving
  - A name of the data class/ type, e.g., `MyStructuredData`
  - Add the attributes/ variables/ item the to contain, e.g., `name, unit_price, quantity_on_hand`

PYTHON DATA CLASSES

```python
from dataclasses import dataclass

@dataclass
class MyStructuredData:
  name: str
  unit_price: float
  quantity_on_hand: int = 0

  @property
  def total_cost(self) -> float:
    return self.unit_price * self.quantity_on_hand
```

- Similar to a *normal* class
- Focussed on representing data objects
- Define by giving
  - A name of the data class/ type, e.g., `MyStructuredData`
  - Add the attributes/ variables/ item the to contain, e.g., `name, unit_price, quantity_on_hand`

PYTHON DATA CLASSES

```python
from dataclasses import dataclass

@dataclass
class MyStructuredData:
  name: str
  unit_price: float
  quantity_on_hand: int = 0

  @property
  def total_cost(self) -> float:
    return self.unit_price * self.quantity_on_hand
```

- Similar to a *normal* class
- Focussed on representing data objects
- Define by giving
  - A name of the data class/ type, e.g., `MyStructuredData`
  - Add the attributes/ variables/ item the to contain, e.g., `name, unit_price, quantity_on_hand`

Optional:
Like a formula in Excel, calculate the value of a cell based on other values

PYTHON DATA CLASSES

- Similar to a *normal* class
- Focussed on representing data objects
- Define by giving
  - A name of the data class/ type, e.g., `MyStructuredData`
  - Add the attributes/ variables/ item the to contain, e.g., `name, unit_price, quantity_on_hand`

Optional:
Like a formula in Excel, calculate the value of a cell based on other values

```
from dataclasses import dataclass

@dataclass
class MyStructuredData:
  name: str
  unit_price: float
  quantity_on_hand: int = 0

  @property
  def total_cost(self) -> float:
    return self.unit_price * self.quantity_on_hand

my_data = MyStructuredData("Apple", 2.20)
print(my_data)

print(my_data.name, my_data.unit_price, my_data.quantity_on_hand)
```

PYTHON DATA CLASSES

- Similar to a *normal* class
- Focussed on representing data objects
- Define by giving
  - A name of the data class/ type, e.g., `MyStructuredData`
  - Add the attributes/ variables/ item the to contain, e.g., `name, unit_price, quantity_on_hand`

```python
from dataclasses import dataclass

@dataclass
class MyStructuredData:
  name: str
  unit_price: float
  quantity_on_hand: int = 0

  @property
  def total_cost(self) -> float:
    return self.unit_price * self.quantity_on_hand

my_data = MyStructuredData("Apple", 2.20)
print(my_data)

print(my_data.name, my_data.unit_price, my_data.quantity_on_hand)
```

Optional:
Like a formula in Excel, calculate the value of a cell based on other values

```
MyStructuredData(name='Apple', unit_price=2.2, quantity_on_hand=0)
```

```
Apple 2.2 0
```

PYTHON DATA CLASSES

- Similar to a *normal* class
- Focussed on representing data objects
- Define by giving
  - A name of the data class/ type, e.g., `MyStructuredData`
  - Add the attributes/ variables/ item the to contain, e.g., `name, unit_price, quantity_on_hand`

```python
from dataclasses import dataclass

@dataclass
class MyStructuredData:
  name: str
  unit_price: float
  quantity_on_hand: int = 0

  @property
  def total_cost(self) -> float:
    return self.unit_price * self.quantity_on_hand

my_data = MyStructuredData("Apple", 2.20)
print(my_data)

print(my_data.name, my_data.unit_price, my_data.quantity_on_hand)

my_data.quantity_on_hand = 5
print(my_data.total_cost)
```

Optional:
Like a formula in Excel, calculate the value of a cell based on other values

```
MyStructuredData(name='Apple', unit_price=2.2, quantity_on_hand=0)
```

```
Apple 2.2 0
```

PYTHON DATA CLASSES

- Similar to a *normal* class
- Focussed on representing data objects
- Define by giving
  - A name of the data class/ type, e.g., `MyStructuredData`
  - Add the attributes/ variables/ item the to contain, e.g., `name, unit_price, quantity_on_hand`

```python
from dataclasses import dataclass

@dataclass
class MyStructuredData:
  name: str
  unit_price: float
  quantity_on_hand: int = 0

  @property
  def total_cost(self) -> float:
    return self.unit_price * self.quantity_on_hand

my_data = MyStructuredData("Apple", 2.20)
print(my_data)

print(my_data.name, my_data.unit_price, my_data.quantity_on_hand)

my_data.quantity_on_hand = 5
print(my_data.total_cost)
```

```
MyStructuredData(name='Apple', unit_price=2.2, quantity_on_hand=0)
```

```
Apple 2.2 0
```

```
11.0
```

Optional:
Like a formula in Excel, calculate the value of a cell based on other values

OPENAI API & PYDANTIC

- Tell ChatGPT about your data model

AARHUS BSS
DEPARTMENT OF MANAGEMENT
AARHUS UNIVERSITY

Magnus Bender

PYDANTIC DATACLASSES

- Use pydantic and data classes together

➡ Data classes with type enforcement

PYDANTIC DATACLASSES

- Use pydantic and data classes together

➡ Data classes with type enforcement

```
from pydantic import BaseModel

class iPhoneReview(BaseModel):
  camera_quality: int
  battery_life: int
  price_worth: bool
  overall_stars: int
  review_title: str
  positives: list[str]
  negatives: list[str]
```

PYDANTIC DATACLASSES

- Use pydantic and data classes together

➡ Data classes with type enforcement

```python
from pydantic import BaseModel

class iPhoneReview(BaseModel):
  camera_quality: int
  battery_life: int
  price_worth: bool
  overall_stars: int
  review_title: str
  positives: list[str]
  negatives: list[str]
```

```python
review = iPhoneReview(
  camera_quality=5,
  battery_life=5,
  price_worth=True,
  overall_stars=5,
  review_title="",
  positives=["positive Test"],
  negatives=["negative Test"]
)

print(review)

print("Battery live rating:",
review.battery_life)
print("Is it worth the price?",
    "Yes" if review.price_worth else "No")
```

PYDANTIC DATACLASSES

- Use pydantic and data classes together

➡ Data classes with type enforcement

```python
from pydantic import BaseModel

class iPhoneReview(BaseModel):
  camera_quality: int
  battery_life: int
  price_worth: bool
  overall_stars: int
  review_title: str
  positives: list[str]
  negatives: list[str]
```

```python
review = iPhoneReview(
  camera_quality=5,
  battery_life=5,
  price_worth=True,
  overall_stars=5,
  review_title="",
  positives=["positive Test"],
  negatives=["negative Test"]
)

print(review)

print("Battery live rating:",
review.battery_life)
print("Is it worth the price?",
    "Yes" if review.price_worth else "No")
```

```
Battery live rating: 5
```

```
Is it worth the price? Yes
```

USE WITH CHATGPT

```
from openai import OpenAI
client = OpenAI(api_key="sk-proj-...")

response = client.responses.parse(
  model=model,
  input=[
    {"role":"user", "content": "Analyze the following review."},
    {"role":"user", "content": "“Works like magic - flawless!” (5 Stars); I’ve ..."}
  ],
  text_format=iPhoneReview,
)

print(type(response.output_parsed), response.output_parsed)
```

USE WITH CHATGPT

```
from openai import OpenAI
client = OpenAI(api_key="sk-proj-...")

response = client.responses.parse(
  model=model,
  input=[
    {"role":"user", "content": "Analyze the following review."},
    {"role":"user", "content": "“Works like magic - flawless!” (5 Stars); I’ve ..."}
  ],
  text_format=iPhoneReview,
)

print(type(response.output_parsed), response.output_parsed)
```

```
<class '__main__.iPhoneReview'>
camera_quality=5 battery_life=5 price_worth=True overall_stars=5 review_title='Works
like magic - flawless!' positives=[' Razor-sharp display', …] negatives=[]
```

IT'S YOUR TURN

IT'S YOUR TURN

1. Check out the notebook and run the cells of *Task 1*

IT'S YOUR TURN

1. Check out the notebook and run the cells of *Task 1*
2. Use your own use-case for *Task 2*
   i. Draft a fitting data model
   ii. Use the your data model with the OpenAI API

IT'S YOUR TURN

1. Check out the notebook and run the cells of *Task 1*
2. Use your own use-case for *Task 2*
   i. Draft a fitting data model
   ii. Use the your data model with the OpenAI API
3. I will present an exemplary solution for the time tracking via mails

IT'S YOUR TURN

1. Check out the notebook and run the cells of *Task 1*
2. Use your own use-case for *Task 2*
   i. Draft a fitting data model
   ii. Use the your data model with the OpenAI API
3. I will present an exemplary solution for the time tracking via mails

- There are more specific types defined by Pydantic https://docs.pydantic.dev/2.11/api/types/#pydantic.types.PositiveInt

IT'S YOUR TURN

1. Check out the notebook and run the cells of *Task 1*
2. Use your own use-case for *Task 2*
   i. Draft a fitting data model
   ii. Use the your data model with the OpenAI API
3. I will present an exemplary solution for the time tracking via mails

- There are more specific types defined by Pydantic https://docs.pydantic.dev/2.11/api/types/#pydantic.types.PositiveInt
- Is possible and helpful nest types →

```
class MyInnerType(BaseModel):
  value:str

class MyOuterType(BaseModel):
  elements:list[MyInnerType]
```

IT'S YOUR TURN

1. Check out the notebook and run the cells of *Task 1*
2. Use your own use-case for *Task 2*
   i. Draft a fitting data model
   ii. Use the your data model with the OpenAI API
3. I will present an exemplary solution for the time tracking via mails

Group: 10:00 - 12:00

- There are more specific types defined by Pydantic https://docs.pydantic.dev/2.11/api/types/#pydantic.types.PositiveInt
- Is possible and helpful nest types →

```
class MyInnerType(BaseModel):
  value:str

class MyOuterType(BaseModel):
  elements:list[MyInnerType]
```

IT'S YOUR TURN

1. Check out the notebook and run the cells of *Task 1*
2. Use your own use-case for *Task 2*
   i. Draft a fitting data model
   ii. Use the your data model with the OpenAI API
3. I will present an exemplary solution for the time tracking via mails

Group: 8:00 - 10:00

- There are more specific types defined by Pydantic https://docs.pydantic.dev/2.11/api/types/#pydantic.types.PositiveInt
- Is possible and helpful nest types →

```
class MyInnerType(BaseModel):
  value:str

class MyOuterType(BaseModel):
  elements:list[MyInnerType]
```

EXAMPLE

- Time Tracking via E-Mails

EXAMPLE:
TIME TRACKING VIA E-MAILS

- Each employee reports their daily time recordings and time spend per project via e-mail
- Transfer all the information from the e-mails to a structured Excel sheet

gpt-oss:120b

EXAMPLE: TIME TRACKING VIA E-MAILS

- Each employee reports their daily time recordings and time spend per project via e-mail
- Transfer all the information from the e-mails to a structured Excel sheet

Subject: Time Tracking Report – Monday, 2 September 2025

Dear Alex,

I began my workday at eight fifteen in the morning and wrapped up at five twenty-five in the afternoon. During that time I dedicated three hours and ten minutes to the Customer Portal redesign, starting at eight thirty and concluding at eleven forty. After a short break I shifted focus to the API integration for the new payment gateway, working from twelve fifteen until two twenty-five, which amounts to two hours and ten minutes. The remainder of the afternoon, from two thirty until five twenty-five, was spent on the quarterly performance analytics dashboard, where I logged exactly two hours and fifty minutes of development and testing.

Please let me know if you need any further details or clarification on any of the tasks.

Best regards,
Jordan

gpt-oss:120b

EXCEL: DATA MODEL

Name of Emplyee	Day of Work	Start of Workday	End of Workday	Hours Worked	Projects	Time per Project	Absence
Jordan	2. September 2025	8:15 am	5:25 pm	8h 10m	Customer Portal	3h 10m	no
					Payment Gateway API	2h 10m	
					Performance Analytics Dashboard	2h 50m	
					Breaks	total 40m	
...	...	...	...	...	...	...	...

PYDANTIC: DATA MODEL PART I

PYDANTIC: DATA MODEL PART I

```
from openai import OpenAI
from pydantic import BaseModel, computed_field

from datetime import date
from typing import Optional
```

PYDANTIC: DATA MODEL PART I

```python
from openai import OpenAI
from pydantic import BaseModel, computed_field

from datetime import date
from typing import Optional
```

```python
class StructuredTimeTrackingMail(BaseModel):
  employee_name:str
  day:date

  workday_start:HourMinute
  workday_end:HourMinute
  hours_worked:float

  projects:list[Project] = []
  breaks:list[Break] = []

  day_off:bool = False
  day_ill:bool = False
```

PYDANTIC: DATA MODEL PART I

```python
from openai import OpenAI
from pydantic import BaseModel, computed_field

from datetime import date
from typing import Optional

class HourMinute(BaseModel):
  hour:int
  minute:int
```

```python
class StructuredTimeTrackingMail(BaseModel):
  employee_name:str
  day:date

  workday_start:HourMinute
  workday_end:HourMinute
  hours_worked:float

  projects:list[Project] = []
  breaks:list[Break] = []

  day_off:bool = False
  day_ill:bool = False
```

PYDANTIC: DATA MODEL PART I

```python
from openai import OpenAI
from pydantic import BaseModel, computed_field

from datetime import date
from typing import Optional

class HourMinute(BaseModel):
  hour:int
  minute:int

class Project(BaseModel):
  name:str
  start:Optional[HourMinute] = None
  end:Optional[HourMinute] = None
  hours_spent:float
```

```python
class StructuredTimeTrackingMail(BaseModel):
  employee_name:str
  day:date

  workday_start:HourMinute
  workday_end:HourMinute
  hours_worked:float

  projects:list[Project] = []
  breaks:list[Break] = []

  day_off:bool = False
  day_ill:bool = False
```

PYDANTIC: DATA MODEL PART II

```
class Break(BaseModel):
      start:HourMinute
      end:HourMinute
```

PYDANTIC: DATA MODEL PART II

```python
class Break(BaseModel):
      start:HourMinute
      end:HourMinute

      @computed_field
      @property
      def duration(self) -> float:
            if self.start.hour == self.end.hour:
                  duration_hours = 0
                  duration_minutes = self.end.minute - self.start.minute
            else:
                  duration_hours = self.end.hour - self.start.hour - 1
                  duration_minutes = 60 - self.start.minute + self.end.minute

            return duration_hours + duration_minutes/60
```

PYDANTIC: DATA MODEL PART II

```python
class Break(BaseModel):
      start:HourMinute
      end:HourMinute

      @computed_field
      @property
      def duration(self) -> float:
            if self.start.hour == self.end.hour:
                  duration_hours = 0
                  duration_minutes = self.end.minute - self.start.minute
            else:
                  duration_hours = self.end.hour - self.start.hour - 1
                  duration_minutes = 60 - self.start.minute + self.end.minute

            return duration_hours + duration_minutes/60

      def __str__(self) -> str:
            return "Break({:02d}:{:02d} to {:02d}:{:02d} took {:.2f} hours)".format(
                  self.start.hour, self.start.minute,
                  self.end.hour, self.end.minute,
                  self.duration
            )
```

PYDANTIC: DATA MODEL PART II

```python
class Break(BaseModel):
      start:HourMinute
      end:HourMinute

      @computed_field
      @property
      def duration(self) -> float:
            if self.start.hour == self.end.hour:
                  duration_hours = 0
                  duration_minutes = self.end.minute - self.start.minute
            else:
                  duration_hours = self.end.hour - self.start.hour - 1
                  duration_minutes = 60 - self.start.minute + self.end.minute

            return duration_hours + duration_minutes/60

      def __str__(self) -> str:
            return "Break({:02d}:{:02d} to {:02d}:{:02d} took {:.2f} hours)".format(
                  self.start.hour, self.start.minute,
                  self.end.hour, self.end.minute,
                  self.duration
            )
```

```python
b1 = Break(
  start=HourMinute(hour=12, minute=00),
  end=HourMinute(hour=12, minute=15)
)
print(b1, b1.duration, b1.start.hour)
```

PYDANTIC: DATA MODEL PART II

```python
class Break(BaseModel):
      start:HourMinute
      end:HourMinute

      @computed_field
      @property
      def duration(self) -> float:
            if self.start.hour == self.end.hour:
                  duration_hours = 0
                  duration_minutes = self.end.minute - self.start.minute
            else:
                  duration_hours = self.end.hour - self.start.hour - 1
                  duration_minutes = 60 - self.start.minute + self.end.minute

            return duration_hours + duration_minutes/60

      def __str__(self) -> str:
            return "Break({:02d}:{:02d} to {:02d}:{:02d} took {:.2f} hours)".format(
                  self.start.hour, self.start.minute,
                  self.end.hour, self.end.minute,
                  self.duration
            )
```

```python
b1 = Break(
  start=HourMinute(hour=12, minute=00),
  end=HourMinute(hour=12, minute=15)
)
print(b1, b1.duration, b1.start.hour)
```

```
Break(12:00 to 12:15 took 0.25 hours) 0.25 12
```

DATA MODEL USAGE

```
s1 = StructuredTimeTrackingMail(
  employee_name="May",
  day=date(2025,10,3),
  workday_end=HourMinute(hour=8, minute=15),
  workday_start=HourMinute(hour=16, minute=45),
  hours_worked=8,
  # projects=[],
  # breaks=[],
  # day_off=False,
  # day_ill=False
)

print(s1.employee_name)
print(s1.day_ill)
print(s1.projects)
print(s1.workday_end)
```

DATA MODEL USAGE

```
s1 = StructuredTimeTrackingMail(
  employee_name="May",
  day=date(2025,10,3),
  workday_end=HourMinute(hour=8, minute=15),
  workday_start=HourMinute(hour=16, minute=45),
  hours_worked=8,
  # projects=[],
  # breaks=[],
  # day_off=False,
  # day_ill=False
)

print(s1.employee_name)
print(s1.day_ill)
print(s1.projects)
print(s1.workday_end)
```

```
May
False
[]
hour=8 minute=15
```

QUERY OPENAI API

Notebook with full implementation and working solution on uCloud!

```
client = OpenAI(api_key="sk-proj-...")

response = client.responses.parse(
  model="gpt-5-nano",
  input=[
    {"role":"user", "content": "Analyze the following e-mail an employee ..."},
    {"role":"user", "content": time_track_email}
  ],
  text_format=StructuredTimeTrackingMail,
)

print(type(response.output_parsed), response.output_parsed)
```

QUERY OPENAI API

Notebook with full implementation and working solution on uCloud!

```
client = OpenAI(api_key="sk-proj-...")

response = client.responses.parse(
  model="gpt-5-nano",
  input=[
    {"role":"user", "content": "Analyze the following e-mail an employee ..."},
    {"role":"user", "content": time_track_email}
  ],
  text_format=StructuredTimeTrackingMail,
)

print(type(response.output_parsed), response.output_parsed)
```

```
<class ,__main__.StructuredTimeTrackingMail'>

employee_name='Jordan'
day=datetime.date(2025, 9, 2)
…
```

DEPARTMENT OF MANAGEMENT
AARHUS UNIVERSITY