



Sentinel Values in Python: Semantics, Double Dispatch, and the Limits of Typing

PyConDE / PyData 2026, April 14th

Florian Wilhelm



inovex



Dr. Florian Wilhelm

 **inovex** • HEAD OF DATA SCIENCE



FlorianWilhelm



FlorianWilhelm.info



florian.wilhelm@inovex.de



Mathematical Modelling



Modern Data Warehousing & Analytics



Personalisation & RecSys



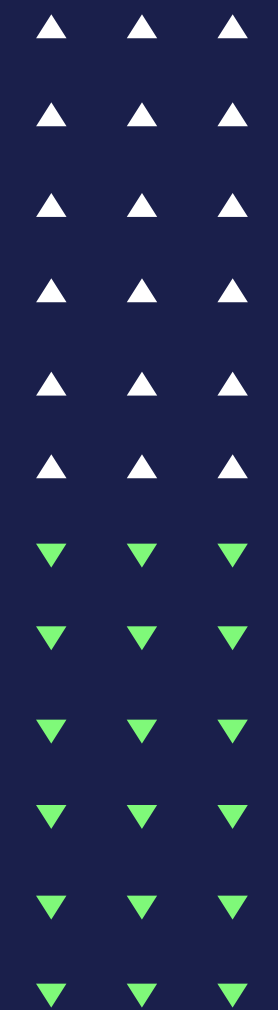
Uncertainty Quantification & Causality



Python Data Stack



OSS Contributor & Creator of PyScaffold



WE ARE INOVEX:

- › IT Project Center
- › Innovation & Excellence
- › Wide Range of Services



More Than
20 inovex Attendees on Site
6 Inspiring Talks
250 Python Users
500 Tech Heads Overall

AND: WE ARE **HIRING!**
Data & AI | Machine Learning



inovex

INNOVATE. INTEGRATE. EXCEED.

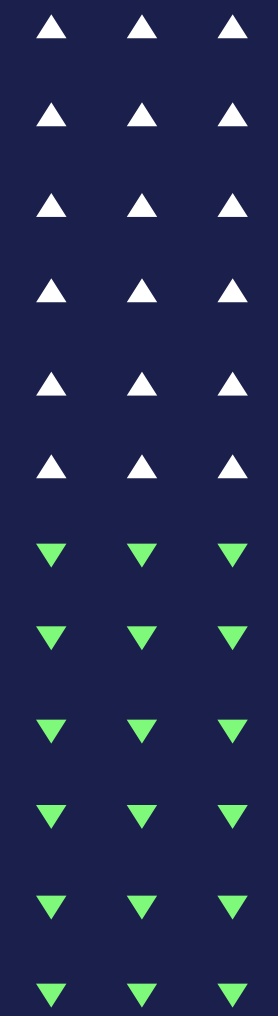


SOFTWARE • DATA & AI • INFRASTRUCTURE

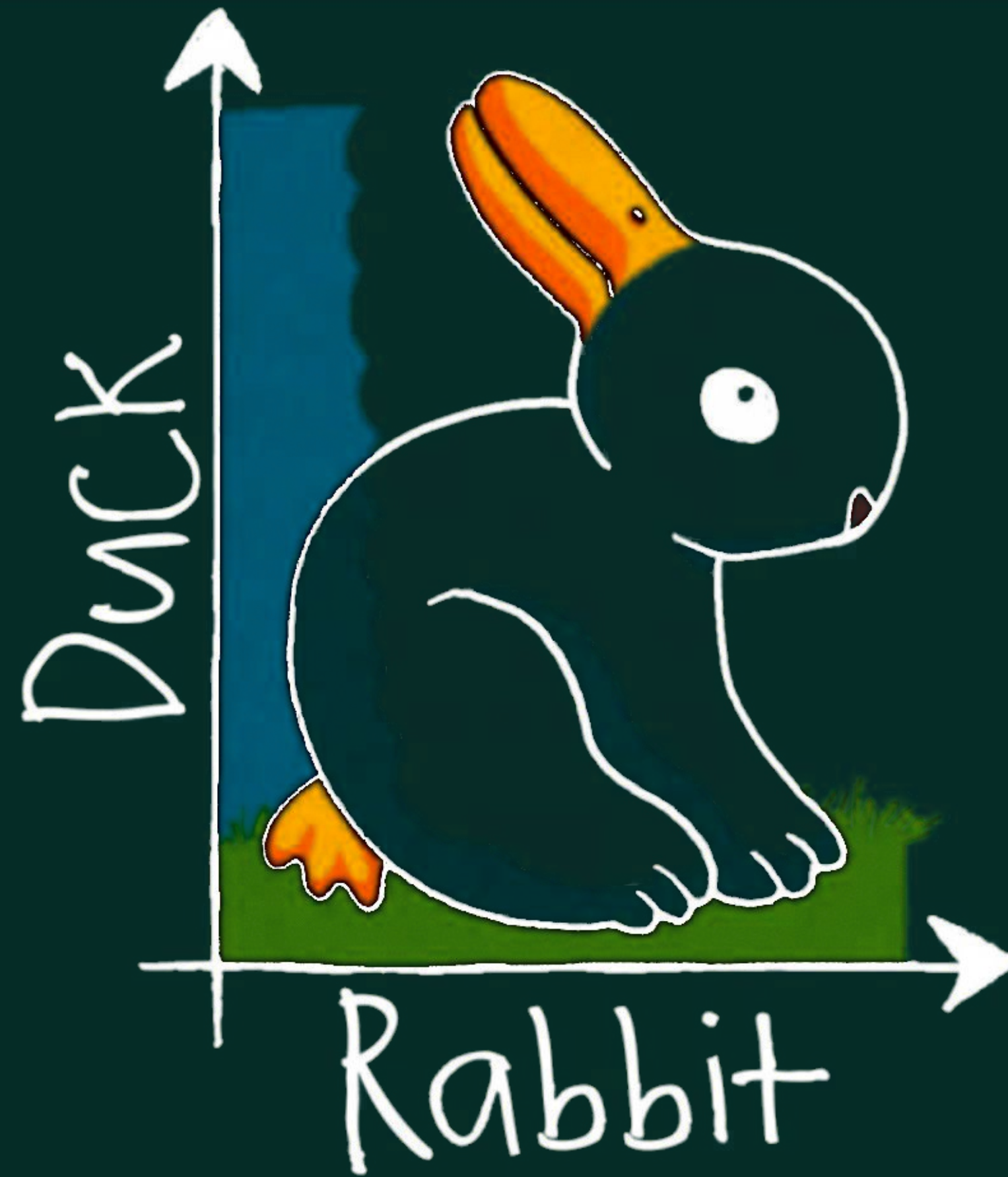
[inovex.de](https://www.inovex.de)

Agenda

1. Motivation: The Ambiguity of None
2. Hidden in Plain Sight: Built-in Sentinels
3. Creating Your Own Sentinels
4. Conflict: Sentinels vs. The Type System
5. Pickling, Serialization, and a Clean repr
6. Future: PEP 661 & Conclusion



inovex



Motivation: The Ambiguity of None

The Ambiguity of None



None is often used to express: **missing, unset, error, ...**



The Ambiguity of None

Usage as **missing** value, which collides with an actual value:



```
cache = {'key': 'value', 'some input': None}
cache.get('missing_key') # returns None
cache.get('some input')  # returns also None
# How to provide a 'default' that is never a cache value?
sentinel = "magic string that never appears elsewhere" # DON'T DO THIS!
cache.get('missing_key', sentinel)
```

Similar problem for *next(...)* in iterators or queues in worker pools



inovex

The Ambiguity of None

Usage as **unset** value, which collides with actual value:



```
def update_config(timeout=None):  
    if timeout is None:  
        # No timeout or some default timeout?
```



inovex

The Ambiguity of None

Usage as **error** value instead of an exception:



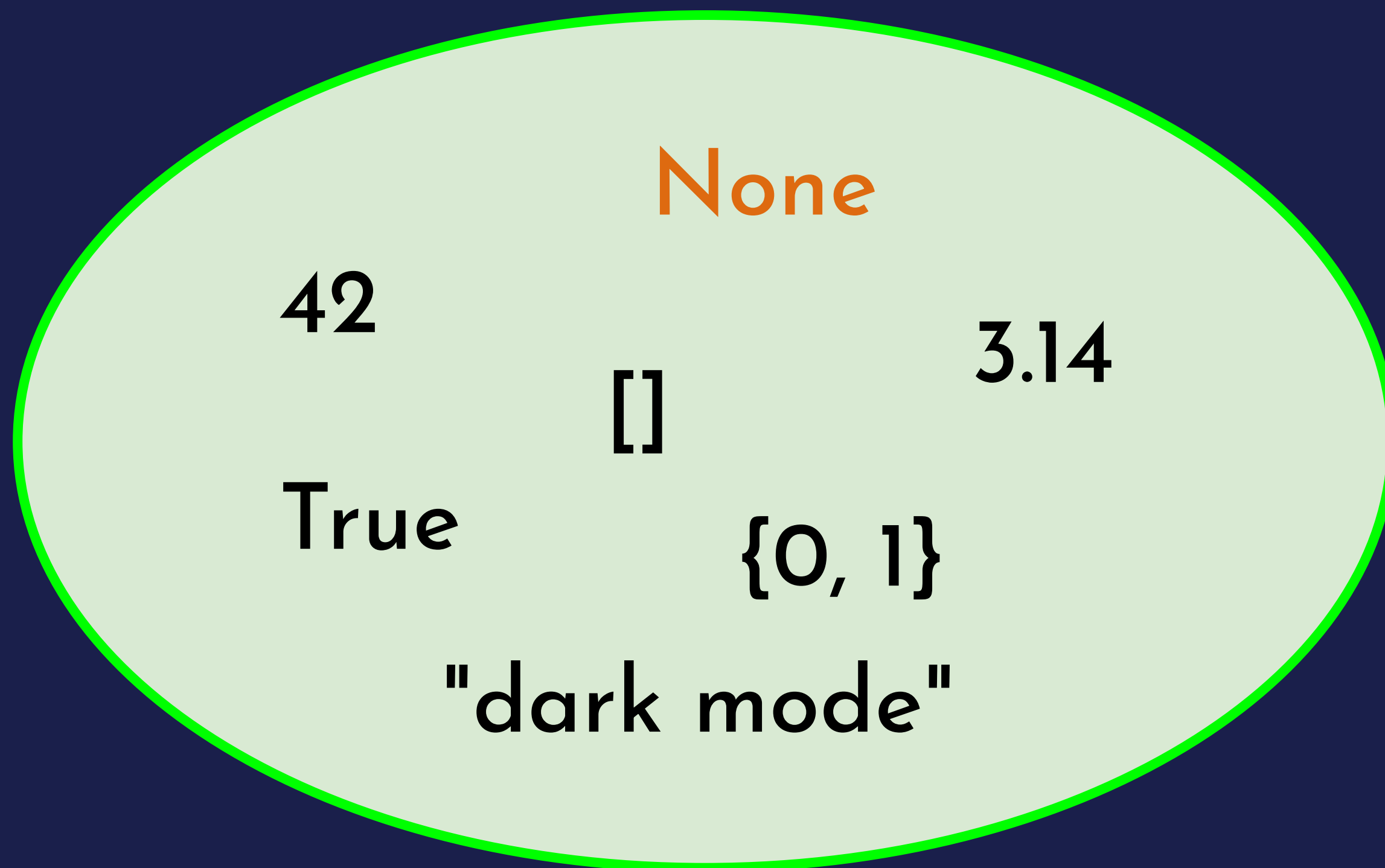
```
def fetch_user_profile(user_id):  
    try:  
        response = api.get(f"/users/{user_id}")  
        return response.json()  
    except TimeoutError:  
        # Endpoint down, no value could be retrieved  
        return None
```



inovex

Sentinels to resolve the Ambiguity

How to express these differences in semantics?

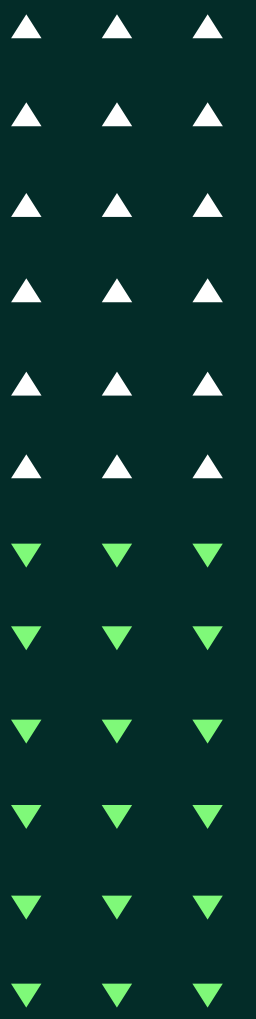
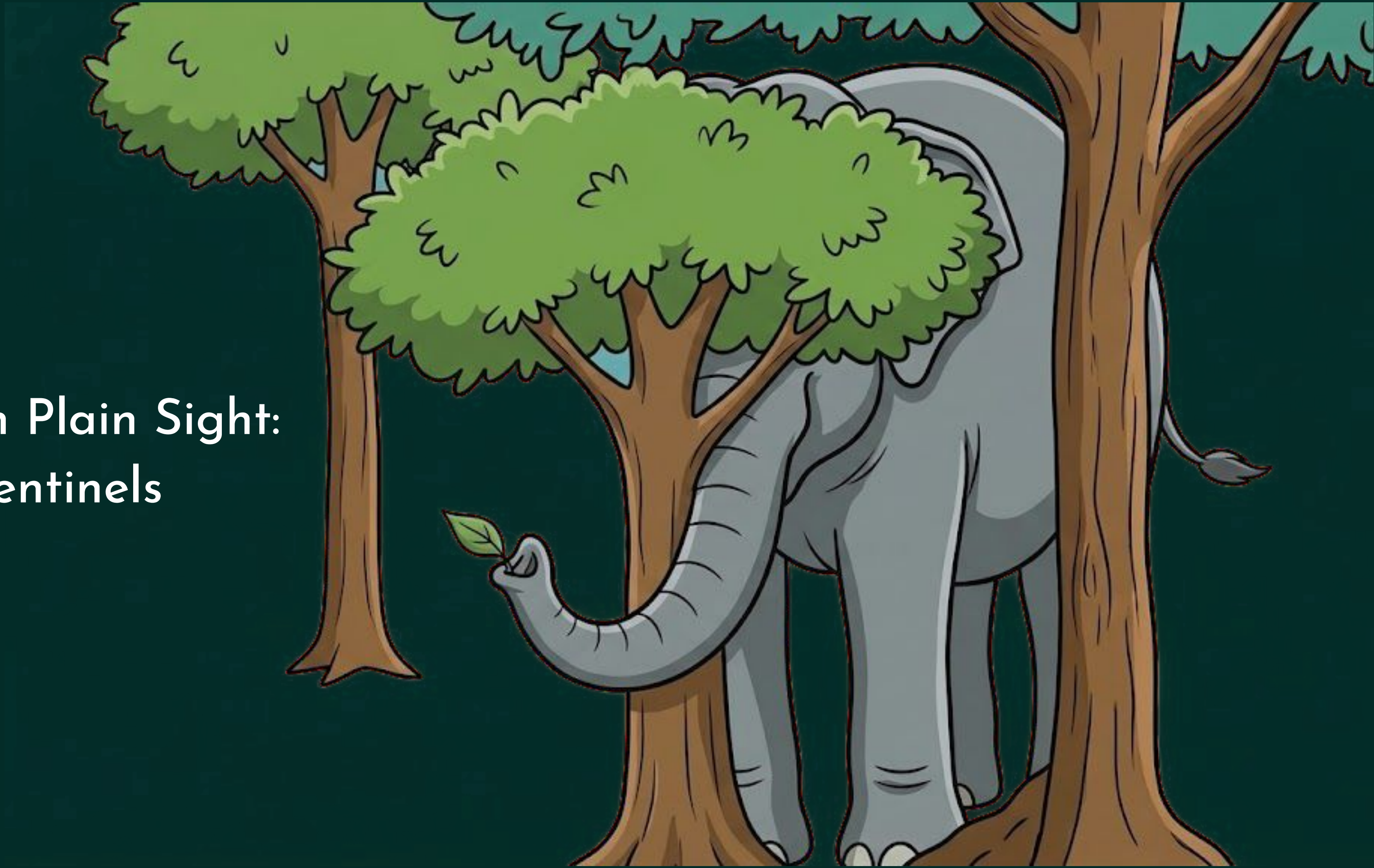


◆ Sentinel

Domain of valid values in the context



Hidden in Plain Sight: Built-in Sentinels



inovex

Ellipsis

Semantic meaning: Skip all other dimensions



```
import numpy as np
```

```
video_data = np.zeros((10, 480, 640, 3))
```

```
video_data[..., 0] = 255 # instead of writing video_data[:, :, :, 0]
```



inovex

Ellipsis

Semantic meaning: This is a stub for something else



```
# In a type hint stub file, math.pyi
```

```
def calculate_pi(precision: int) -> float: ...
```



inovex

Ellipsis

Can be also used for **Unset** instead of **None**



```
def smart_process(value=...):  
    if value is ...:  
        return "The user explicitly left this blank"  
    return f"Processing {value}"
```



inovex

NotImplemented in Double Dispatch

NotImplemented for "Find the implementation somewhere else"



```
class Price:
    def __init__(self, amount: float):
        self.amount = amount

    def __sub__(self, other):
        if isinstance(other, Price):
            return Price(self.amount - other.amount)
        return NotImplemented
```



inovex

NotImplemented in Double Dispatch



```
class PercentageCoupon:
    def __init__(self, percent_off: float):
        self.percent_off = percent_off

    def __rsub__(self, other):
        if isinstance(other, Price):
            reduction = other.amount * (self.percent_off / 100)
            return Price(other.amount - reduction)
        return NotImplemented
```



inovex

Experimental MISSING in PyDantic v2



```
from pydantic.experimental.missing_sentinel import MISSING

class UserUpdate(BaseModel):
    username: str
    bio: str | None | MISSING = MISSING

# Scenario A: User wants to CLEAR their bio (explicitly sending None)
user_a = UserUpdate(username="jdoe", bio=None)
print(f"Bio value: {user_a.bio}") # Output: None

# Scenario B: User didn't include 'bio' at all
user_b = UserUpdate(username="jdoe")
print(f"Is bio missing? {user_b.bio is MISSING}") # Output: True
```



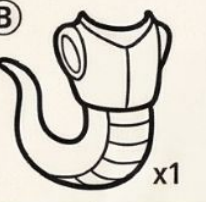
inovex

Creating Your Own Sentinels

SENTINEL



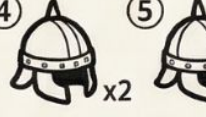
2



x1



3



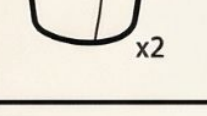
x2



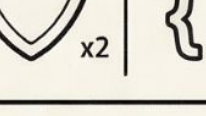
4



x2



5



x2



D



x2



E



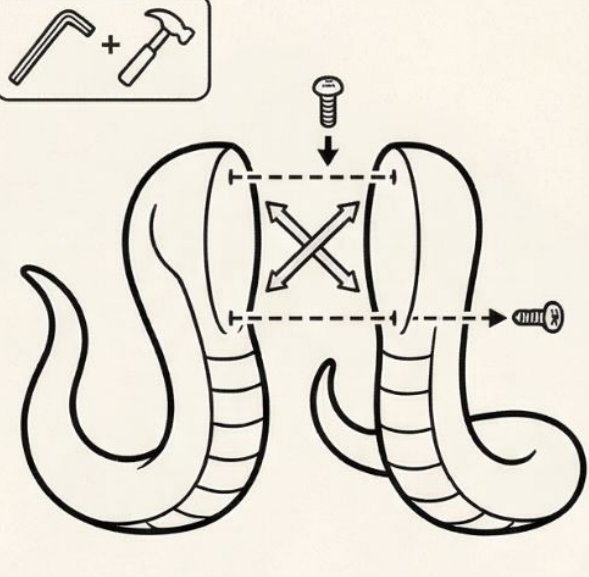
x2



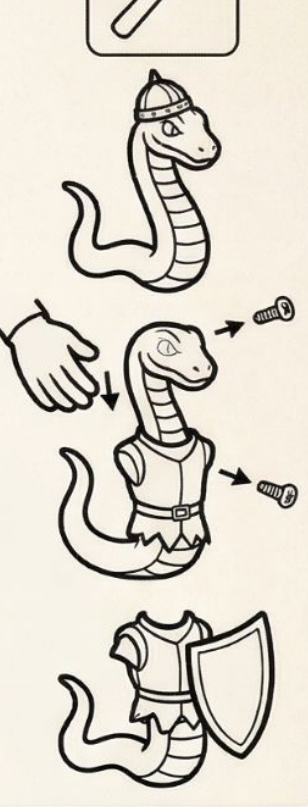
F



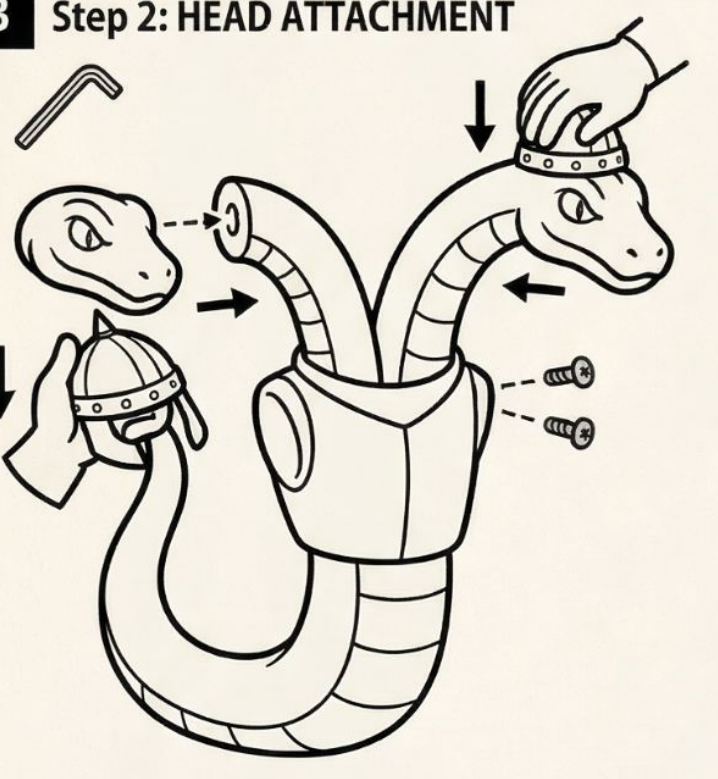
x2



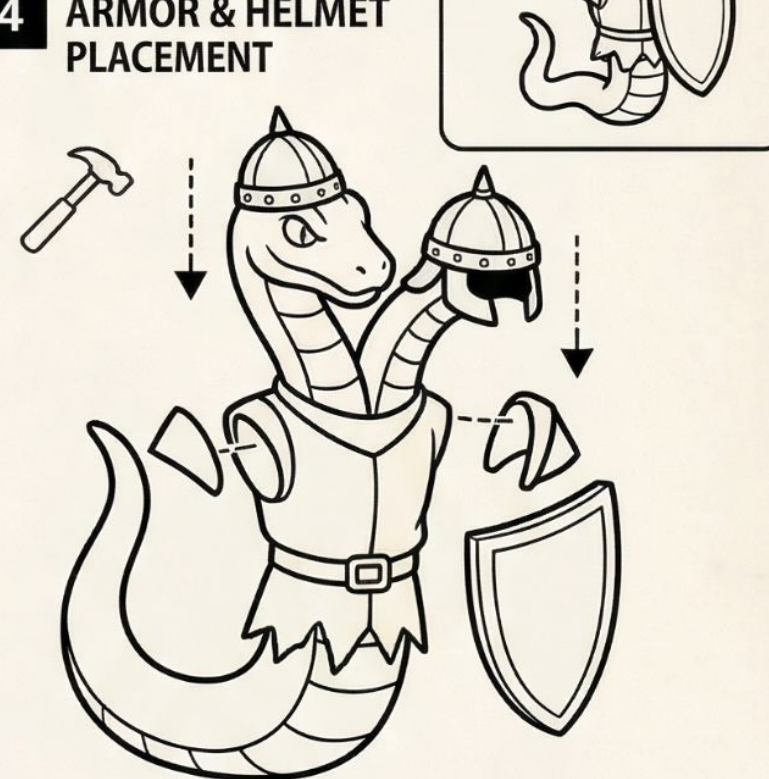
2



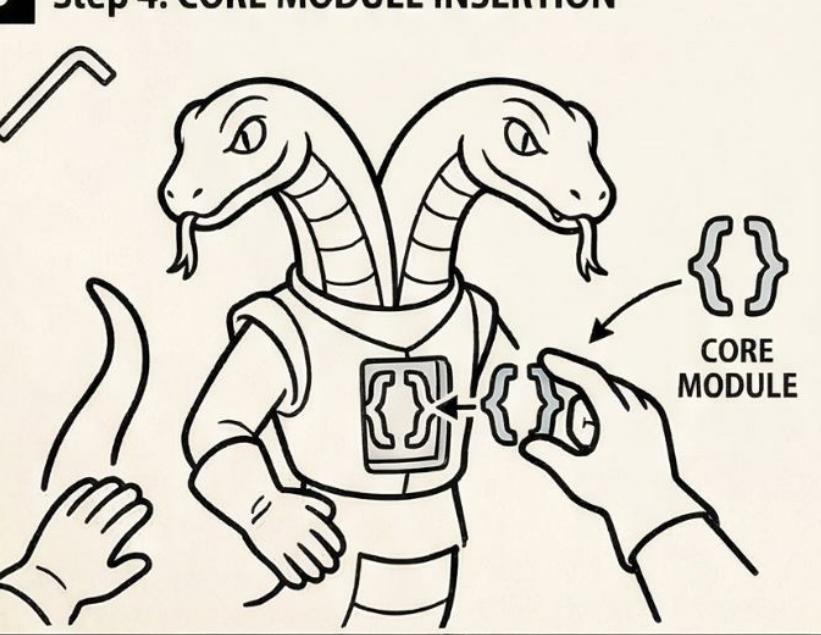
3



3 Step 2: HEAD ATTACHMENT



4 ARMOR & HELMET PLACEMENT



5 Step 4: CORE MODULE INSERTION



6 Step 5: COMPLETED SENTINEL



inovex

object() as Sentinel

object() serves as a unique, distinct value

```
MISSING = object()

def update_user(user_id, email=MISSING):
    """Updates a user. Passing email=None clears the email address."""
    if email is MISSING: # Do nothing
        return
    if email is None: # Remove current e-mail
        db_execute("UPDATE users SET email = NULL WHERE id = ?", user_id)
    else: # Normalize e-mail and update
        db_execute("UPDATE users SET email = ? WHERE id = ?", email.lower(), user_id)
```



inovex

PyConDE & PyData 2026

Thank you!



Dr. Florian Wilhelm

Head of Data Science & Mathematical Modelling

 florian.wilhelm@inovex.de

 inovex.de

 [@inovexlife](https://www.instagram.com/inovexlife)

 [@inovexgmbh](https://twitter.com/inovexgmbh)



inovex

© 2023



The Conflict: Sentinels vs. The Type System



Type Narrowing Does Not Work!

`object()` works really good at runtime but fails static type checking.



```
MISSING = object()
```

```
def update_user(user_id: int, email: str | object | None=MISSING) -> None:
    """Updates a user. Passing email=None clears the email address."""
    if email is MISSING: # Do nothing
        return
    if email is None: # Remove current e-mail
        db_execute("UPDATE users SET email = NULL WHERE id = ?", user_id)
    else: # Normalize e-mail and update
        db_execute("UPDATE users SET email = ? WHERE id = ?", email.lower(), user_id)
```

✗ Error: "object" has no attribute "lower"



inovex

Workaround with Type Guard and isinstance Check

```
from typing import TypeIs

class MissingType:
    __slots__ = ()
    def __repr__(self) -> str:
        return "MISSING"

MISSING = MissingType()

def is_missing[T](val: T | MissingType) -> TypeIs[MissingType]:
    return True if isinstance(val, MissingType) else False
```



TypeGuard

If you remember to use `is\_missing` instead of `is`...

```
def update_user(user_id: int, email: str | MissingType | None=MISSING) -> None:
    """Updates a user. Passing email=None clears the email address."""
    if is_missing(email): # Do nothing
        return
    if email is None: # Remove current e-mail
        db_execute("UPDATE users SET email = NULL WHERE id = ?", user_id)
    else: # Normalize e-mail and update
        db_execute("UPDATE users SET email = ? WHERE id = ?", email.lower(), user_id)
```

✓ ... everything is fine

How about a Magical Literal Strings?

```
MISSING = '__MISSING__'
```

```
def update_user(user_id: int, email: str | Literal[MISSING] | None=MISSING) -> None:
    """Updates a user. Passing email=None clears the email address."""
    if email is MISSING: # Do nothing
        return
    if email is None: # Remove current e-mail
        db_execute("UPDATE users SET email = NULL WHERE id = ?", user_id)
    else: # Normalize e-mail and update
        db_execute("UPDATE users SET email = ? WHERE id = ?", email.lower(), user_id)
```

✗ Error: Parameter 1 of Literal[...] is invalid

So we would need to keep repeating the magical string and what about collisions!

Literal Enums

```
from enum import Enum

class _MissingSentinel(Enum):
    VAL = 'MISSING'

MISSING = _MissingSentinel.VAL
type MissingType = Literal[_MissingSentinel.VAL]

def update_user(user_id: int, email: str | MissingType | None=MISSING) -> None:
    """Updates a user. Passing email=None clears the email address."""
    if email is MISSING: # Do nothing
        return
    if email is None: # Remove current e-mail
        db_execute("UPDATE users SET email = NULL WHERE id = ?", user_id)
    else: # Normalize e-mail and update
        db_execute("UPDATE users SET email = ? WHERE id = ?", email.lower(), user_id)
```

✓ ... everything is fine



inovex

So far...

Literal Enums serve as a type safe substitute for sentinels, but

- cumbersome to write and not easy to understand for everyone
- there is still a difference between the **type** and the **value**

MissingType in parameter type definition and **MISSING** as value

compared to

None in parameter type definition and **None** as value



inovex

Pickling, Serialization, and a Clean repr



The problem with Pickling, Copying and Serialisation

Identity check works not like intended for sentinels.

```
import pickle

MISSING = object()

data = pickle.dumps(MISSING)
loaded = pickle.loads(data)

print(loaded is MISSING) # False – new object created!
```



inovex

No problem with the Literal Enum Trick!

```
import pickle
from enum import Enum

class _MissingSentinel(Enum):
    VAL = 'MISSING'

MISSING = _MissingSentinel.VAL
type MissingType = Literal[_MissingSentinel.VAL]

data = pickle.dumps(MISSING)
loaded = pickle.loads(data)

print(loaded is MISSING)  # True!
```



Sentinels should be easy to understand in Debug Messages

```
from enum import Enum

class _MissingSentinel(Enum):
    VAL = 'MISSING'
    def __repr__(self) -> str:
        return self.value

MISSING = _MissingSentinel.VAL
type MissingType = Literal[_MissingSentinel.VAL]

repr(MISSING)  # 'MISSING'
```



The Future: PEP 661 & Conclusion



inovex

PEP 661 – Sentinel Values

Created June 6th 2021, this PEP proposes Sentinels as a core feature to address the drawbacks of using `object()`:

- uninformative and overly verbose error messages in stack trace
- typing as mentioned above
- is comparison fails in case of copying or unpickling.

Deferred on April 10th, 2025 by the steering council but an experimental implementation exists in `typing-extensions`

<https://github.com/python/steering-council/issues/258>

<https://peps.python.org/pep-0661/>

<https://typing-extensions.readthedocs.io/en/latest/#sentinel-objects>



Sentinel Values from Typing Extensions Library

```
from typing_extensions import Sentinel

Missing = Sentinel('Missing')

def update_user(user_id: int, email: str | Missing | None=Missing) -> None:
    """Updates a user. Passing email=None clears the email address."""
    if email is Missing: # Do nothing
        return
    if email is None: # Remove current e-mail
        db_execute("UPDATE users SET email = NULL WHERE id = ?", user_id)
    else: # Normalize e-mail and update
        db_execute("UPDATE users SET email = ? WHERE id = ?", email.lower(), user_id)
```

⚠ Only pyright with experimental settings supports this

Conclusion

- Be aware of the ambiguity when working with None
- Use sentinels to resolve ambiguity
- Always prefer built-in sentinels over custom ones or the ones provided by the library e.g. pydantic MISSING.
- Literal Enum are the only way practical, type-safe solution for custom sentinels currently.



inovex

PyConDE & PyData 2026

Thank you!



Dr. Florian Wilhelm

Head of Data Science & Mathematical Modelling

 florian.wilhelm@inovex.de

 inovex.de

 [@inovexlife](https://www.instagram.com/inovexlife)

 [@inovexgmbh](https://twitter.com/inovexgmbh)



inovex

© 2023