Descriptors — Programmable Attribute Access

Why this concept exists

Many attributes are not simple stored values. They require validation, computation, or controlled access while still behaving like normal attributes. Descriptors provide attribute-level behavior.

Target Usage

```
class Subscription:
    expiry_date = ExpiryDate()

sub = Subscription(start_date=10, duration=30)
sub.expiry_date
```

Coding Problem

Design an attribute that is computed dynamically, read-only, and always reflects current object state, without storing the value on the instance.

Baseline Solution

```
class ExpiryDate:
    def __get__(self, obj, objtype=None):
        return obj.start_date + obj.duration

class Subscription:
    expiry_date = ExpiryDate()

    def __init__(self, start_date, duration):
        self.start_date = start_date
        self.duration = duration
```

Extended Example: ORM-style Field

```
class IntegerField:
    def __set_name__(self, owner, name):
        self.name = name

    def __get__(self, obj, objtype=None):
        return obj.__dict__.get(self.name)

    def __set__(self, obj, value):
        if not isinstance(value, int):
            raise TypeError("Expected int")
        obj.__dict__[self.name] = value

class User:
    age = IntegerField()

    def __init__(self, age):
        self.age = age
```

Key Insight

Descriptors control behavior of attributes that exist at class creation time. They are invoked before __getattr__ and form the foundation of properties, ORM fields, and validation systems.