Attribute Access Control in Python

Why this concept exists

In real systems, not all attributes are known at class definition time. Configuration systems, feature flags, proxies, and adapters often need to respond to attribute access dynamically.

Target Usage

```
env = {
    "DB_HOST": "localhost",
    "DEBUG": True
}

config = Config(env)

config.DB_HOST
config.DEBUG
```

Coding Problem

Design an object that supports attribute-style access for keys that do not exist at class creation time, loads values lazily, and raises AttributeError for missing keys.

Baseline Solution

```
class Config:
    def __init__(self, source):
        self._source = source
        self._cache = {}

    def __getattr__(self, name):
        if name in self._source:
            value = self._source[name]
            self._cache[name] = value
            return value
        raise AttributeError(f"No such config key: {name}")
```

Key Insight

__getattr__ is invoked only after normal attribute lookup fails. Descriptors cannot solve this problem because the attribute object itself does not exist at class creation time.