Digital World (2018)

Week 8, S2: Methods, Attributes, and Principles

Chris Poskitt

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Refresher: the __str__ method

b.socrative.com, POSKITT5665

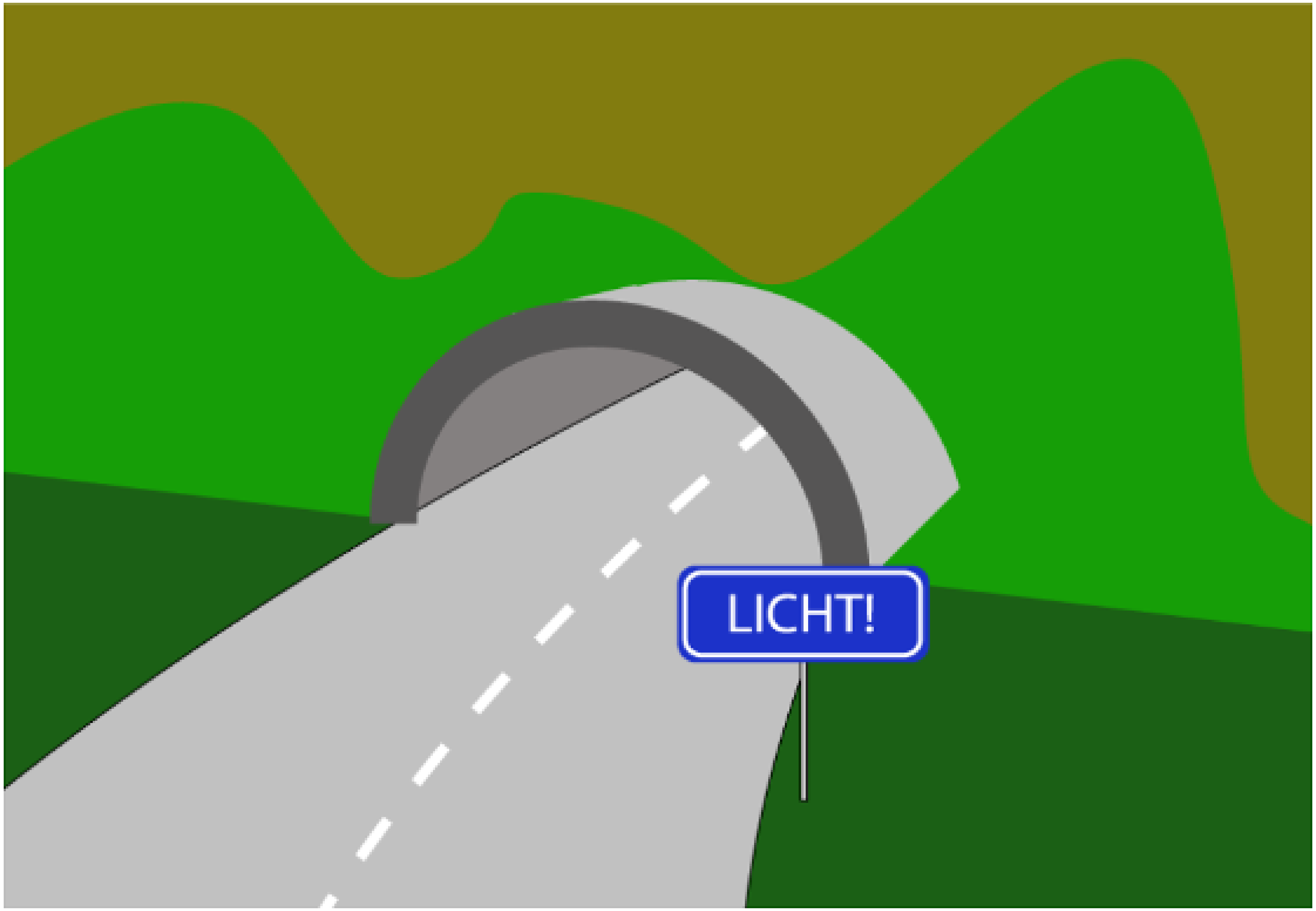
```
class Coordinate:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def __str__(self):
        return "(for 'x' you got {}, for 'y' you got {})".format(self.x, self.y)

p1 = Coordinate(5,6)
p2 = p1
p2.x = p1.y * 2

print(str(p1))
```

what's printed?

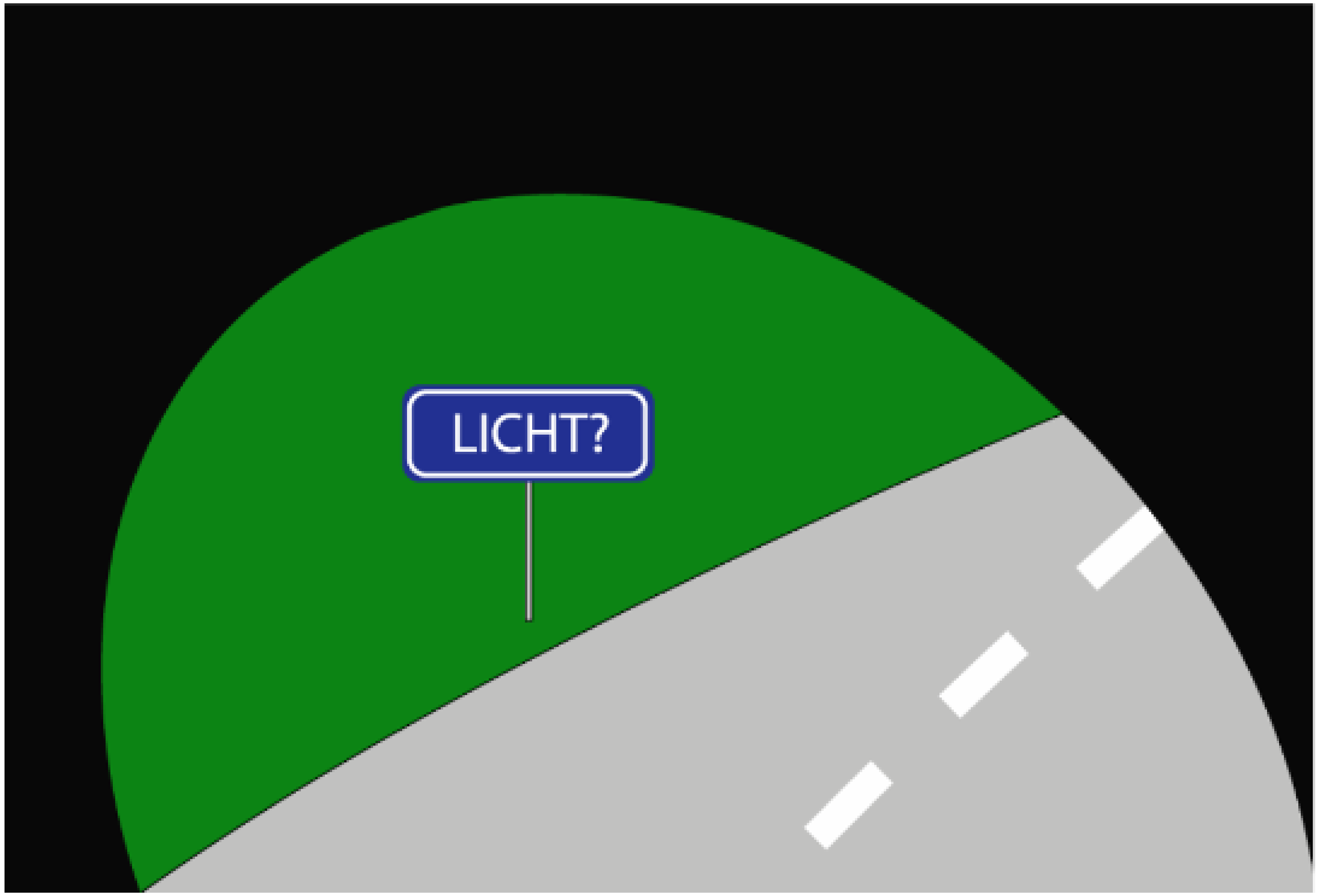
Today we will see:

- that methods can be commands or queries *(or both)*
- that data attributes can be public or private *(by convention)*
- some important object-oriented principles

Command



Query



Command or query?

b.socrative.com, POSKITT5665

```
fav_dishes_list.append(“laksam kelantan”)
```

```
robot.wheels(100, 100)
```

```
p1.distance_from_origin()
```

```
firebase.get(‘/movement_list’)
```

```
f.readline()
```

A principle: *command-query separation*

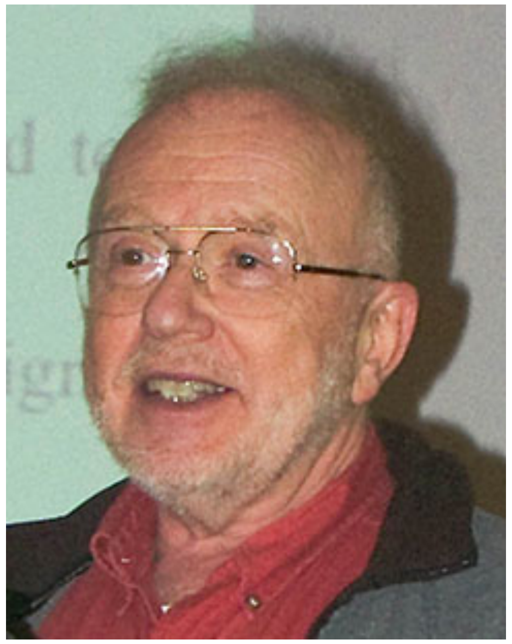
"asking a question shouldn't change the answer"

B. Meyer

? *how might we adapt f.readline() to this principle?*

Another principle: *information hiding*

if code chunk A doesn't need to know how B is implemented, don't make it know it; then when B changes, you needn't change A



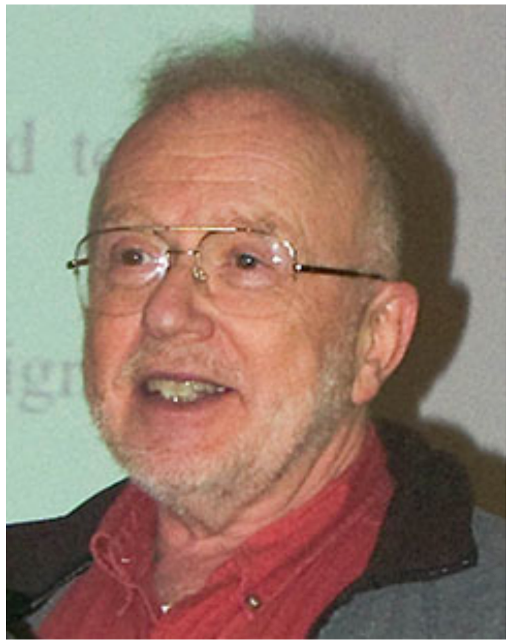
D. Parnas

Another principle: *information hiding*

if code chunk A doesn't need to know how B is implemented, don't make it know it; then when B changes, you needn't change A

$$\begin{tikzcd} \text{t1} \arrow[r] & \boxed{\text{temperature} \longrightarrow 24} \end{tikzcd}$$

t1.temperature = -300

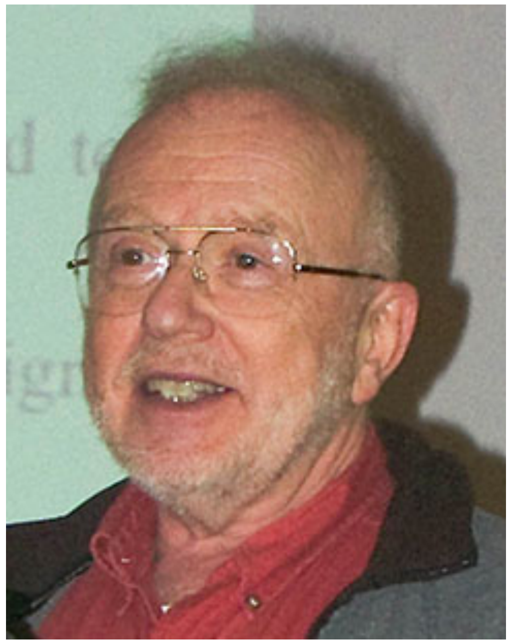

D. Parnas

Another principle: *information hiding*

if code chunk A doesn't need to know how B is implemented, don't make it know it; then when B changes, you needn't change A

$$\begin{tikzcd} \text{t1} \arrow[r] & \boxed{\text{temperature} \longrightarrow \text{-300}} \end{tikzcd}$$

t1.temperature = -300 X



D. Parnas

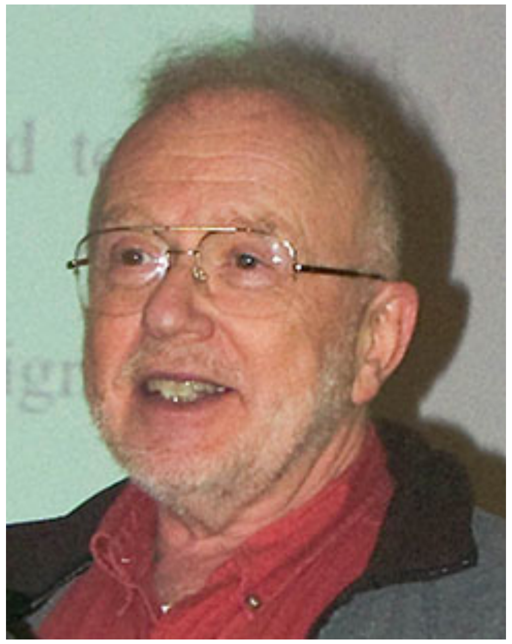
Another principle: *information hiding*

if code chunk A doesn't need to know how B is implemented, don't make it know it; then when B changes, you needn't change A

t1 ⟶ temperature —> -300

t1.temperature = -300 X

t1.set_temperature() = -300



D. Parnas

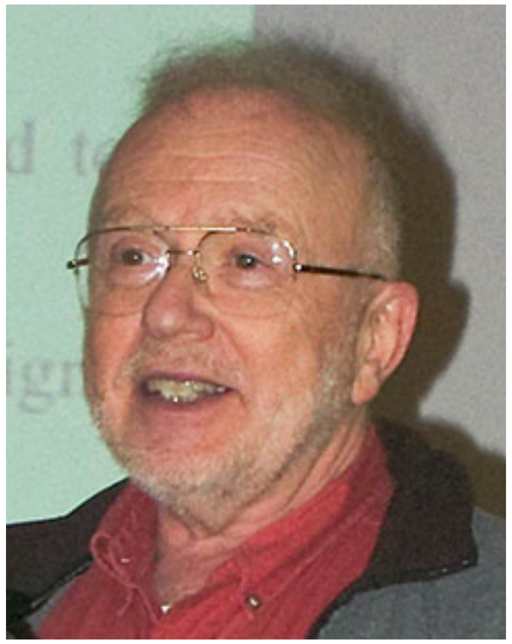
Another principle: *information hiding*

if code chunk A doesn't need to know how B is implemented, don't make it know it; then when B changes, you needn't change A

$$\begin{tikzcd} t1 \arrow[r] & \boxed{\text{temperature} \longrightarrow -273} \end{tikzcd}$$

t1.temperature = -300 X

t1.set_temperature() = -300 ✓


D. Parnas

Another principle: *information hiding*

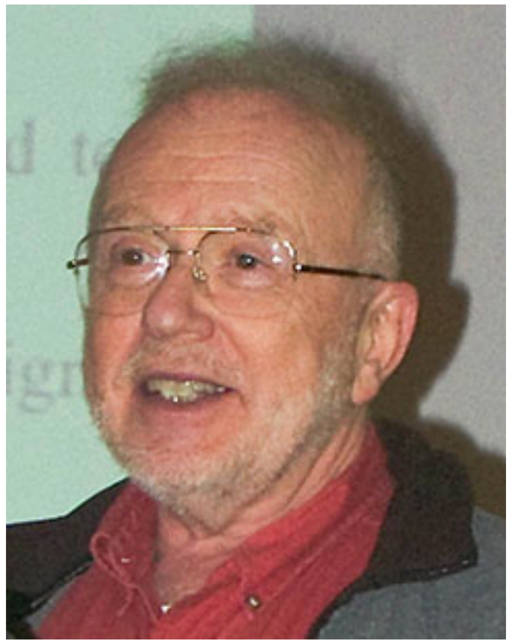
if code chunk A doesn't need to know how B is implemented, don't make it know it; then when B changes, you needn't change A

t1 ⟶ temperature —> -273

t1.temperature = -300 ✗

t1.set_temperature() = -300 ✓

t1.get_temperature() ✓


D. Parnas

“Private” attributes; get / set methods

- instead of modifying attributes directly, it’s better to provide stable interfaces to protect the program from change

- convention: use preceding underscores (“_attribute”) to indicate that _attribute is private

  => *i.e. not to be called from outside of the class*

- external “clients” instead call get or set methods to access or mutate the object state

BUT! *Uniform access principle*

"all services of an object should be available through a uniform notation, which does not betray whether they are implemented through storage or through computation"

we failed this test

our interface changed from
t1.temperature to
t1.get_temperature()

B. Meyer

Solution: the **property** function

- the built-in property function allows get / set methods to be accessed with uniform syntax *(as if it were an attribute)*

```
temperature = property(get_temperature, set_temperature)
```

- if t1.temperature is queried, t1.get_temperature() is called

- if there is an assignment t1.temperature = -300, then t1.set_temperature(-300) is called

Summary

- methods can be commands or queries *(or both — not advised)*
- data attributes can be public or private *(by convention)*
- “clients” of objects should interact with them via stable interfaces
- the property function allows clients to do so via a uniform interface