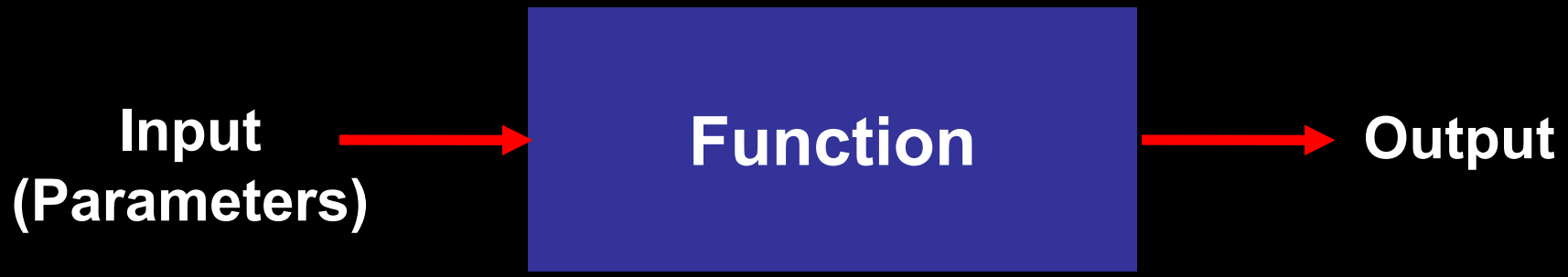
LT 10a - Data Abstraction

Data Abstraction

- Abstraction means hiding details. To look at some object from a higher-perspective.
- Data Abstraction is hiding private data. The user should only need to know how to access the data with the given constructors or accessors.

Recall : We use Functional Abstraction to manage complex program.

Black Box



No need to know HOW it does the job!

Just need to know WHAT it does. ☺

Abstract Data Type (ADT)

Traditionally, data abstraction and functional abstraction combine into the concept of abstract data types (ADT).

Abstract Data Type (ADT)

- The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented.
- It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.

Guidelines for creating an Abstract Data Type (ADT)

Constructors

- Means to create compound data from primitive data.

Getters (Accessors)

- Means to access individual components of compound data.

Setters (Modifiers)

- Means to modify the individual components of compound data.

Guidelines for creating an Abstract Data Type (ADT)

Accessories :

Predicates

- Means to ask (True/False) questions about compound data.

Printers

- Means to display compound data in human-readable form.

Why use Setters and Getters?

- We want to hide the data of the object as much as possible.
- It hides how the data is being handled behind the scenes.
- It allows us to impose validation on the values that the fields are being set to.

Data Abstraction

Demo 1
Demo 2

Example : Coordinate ADT

```
def make_pt(x, y):
   return (x, y)
def get_x(pt):
    return pt[0]
def get_y(pt):
    return pt[1]
def print_pt(pt):
    s = '('+str(pt[0])+','+ str(pt[1])+')'
    return s
```

Example : Coordinate ADT

```
A = make_pt(1,2)
B = make_pt(5,6)

>>> get_x(A)          1
>>> get_y(A)          2
>>> get_x(B)          5
>>> get_y(B)          6

>>> print_pt(A)       '(1,2)'
>>> print_pt(B)       '(5,6)'
```

Example : Coordinate ADT

```
def midpt(pt1, pt2):    # mid point
    pass

def grad(pt1, pt2):     # gradient
    pass

def dist(pt1, pt2):     # distance
    pass

def eqn(pt1, pt2):      # equation y=mx+c
    pass
```

Example : Rational Number ADT

Rational number: n/d

- e.g. 3/5, -1/2
- n: numerator, integer
- d: denominator, non-zero integer

Wishful Thinking

Assume we already have the following:

- **`def make_rat(n, d)`**
  - Returns a rational number with numerator *n*, denominator *d*
  - Constructor
- **`def get_numer(rat)`**
  - Returns the numerator of rational number
  - Getter (Assessor)
- **`def get_denom(rat)`**
  - Returns the denominator of rational number
  - Getter (Assessor)

Rational Number ADT Package

If we want to write the following functions:

- Addition
- Subtraction
- Multiplication,
- Division, … etc.

we will need to understand the arithmetic operations.

Arithmetic Operations

Addition:

$$\frac{n_1}{d_1} + \frac{n_2}{d_2} = \frac{n_1 d_2 + n_2 d_1}{d_1 d_2}$$

Arithmetic Operations

Subtraction:

$$\frac{n_1}{d_1} - \frac{n_2}{d_2} = \frac{n_1 d_2 - n_2 d_1}{d_1 d_2}$$

Arithmetic Operations

Multiplication:

$$\frac{n_1}{d_1} \times \frac{n_2}{d_2} = \frac{n_1 n_2}{d_1 d_2}$$

Arithmetic Operations

Division:

$$\frac{n_1}{d_1} \div \frac{n_2}{d_2} = \frac{n_1 d_2}{d_1 n_2}$$

Predicates

Equality:

$$\frac{n_1}{d_1} = \frac{n_2}{d_2} \quad \text{if and only if} \quad n_1 d_2 = n_2 d_1$$

Predicates

Whole Number:

$$\frac{n_1}{d_1} = k \quad \text{where } k \text{ is an integer, ie } \; n_1 \,\%\, d_1 = 0$$

Read a CSV File – individual fields

```
import csv
with open("csvfile.csv") as f:
    content = csv.reader(f)
    next(content)
    for name, gender, ht, wt in content:
        print(name, gender, ht, wt)
```