

数据抽象



回顾

👁️ 函数的抽象

👁️ 将计算过程抽象出来

👁️ 一次定义，多次使用

👁️ 高阶函数

👁️ 递归

关于更好的编程

- ① 过程的抽象

- ② 数据的抽象

数据的抽象 (Data Abstraction)

图灵机 :

• 111111 0 1111 0 111111 0 11

• Lambda 演算 :

• $\text{Pair} \triangleq \lambda x. \lambda y. \lambda f. f\ x\ y$, $\pi_0 \triangleq \lambda p. p\ \text{True}$, $\pi_1 \triangleq \lambda p. p\ \text{False}$

• $\text{Tuple} \triangleq \lambda x_1. \lambda x_2. \dots \lambda x_n. \lambda f. f\ M_1\ M_2 \dots M_n$, $\pi_i \triangleq \lambda p. p\ (\lambda x_1. \lambda x_2. \dots \lambda. x_n. x_i)$

数据的抽象

- 现实中，我们要处理的数据往往要比1, 2, 3, 4, 5 这样的简单数字要复杂的多。
- 这些数据往往都具有复合结构
 - 如「日期」：“年”、“月”、“日”， 「位置」：“经度”、“纬度”
- 我们需要一种方式使得这些复合数据“粘合”起来，成为单个概念单元，然后我们处理这些复合数据就和普通类型的值一样。

类比函数抽象

- 函数抽象：我们使用抽象将函数的使用 and 实现分离，使得我们在使用函数时无需关心实现的细节。
- 数据抽象：将复合数据对象的表示和处理分离，使得我们在使用这些对象时，不必对数据的具体形式做过多的假设。

数据的抽象

◎ 关键思想：

- ◎ 1. “模块化” (`structure`) 程序使得其操作在“抽象”的数据上，而不是某种特定类型的数据。
- ◎ 2. 具体的数据表示应该作为程序的独立部分 (独立于对其使用的过程) 。

有理数 (Rational Numbers) 例子

$$\frac{\text{numerator}}{\text{denominator}}$$

- 有理数可以用分数精确的表达

 - 其可以用两个整数表达

- 如果我们应用除法的规则得到一个浮点数，我们可能会失去这种精确的表达

保留有理数精度

- 因此，如果我们想保留这种精确的表达，那么我们就需要某种能够结合 `numerator` 和 `denominator` 的表示方法。

抽象出表达

- 利用函数抽象的思想，我们可以先假定我们已经有了某种构造从 `numerator` 和 `denominator` 两个整数中构造有理数的方法（先不用管其实现）。
- 进一步假设，我们可以通过某种方法从一个给定的有理数得到其 `numerator` 和 `denominator`。

抽象出表达

构造子 (constructor) — `rational(n, d)` # returns a rational number

选择子 (Selectors) — `numer(x)` # returns the numerator of x

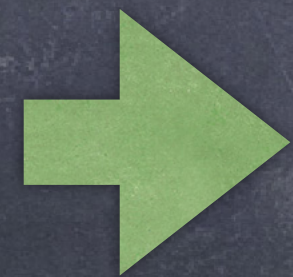
选择子 (Selectors) — `denom(x)` # returns the denominator of x

处理有理数

● 有了这些抽象的“有理数”表达，我们即可对其操作
(或者测试)

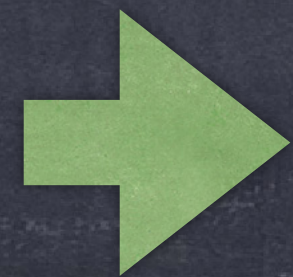
● 比如，可以对有理数进行和普通数据类型一样的计算：

$$\frac{3}{2} * \frac{3}{5} = \frac{9}{10}$$



$$\frac{nx}{dx} * \frac{ny}{dy} = \frac{nx * ny}{dx * dy}$$

$$\frac{3}{2} + \frac{3}{5} = \frac{21}{10}$$



$$\frac{nx}{dx} + \frac{ny}{dy} = \frac{nx * dy + ny * dx}{dx * dy}$$

有理数运算

```
def mul_rational(x, y):  
    return rational(numer(x) * numer(y),  
                    denom(x) * denom(y))
```

$$\frac{nx}{dx} * \frac{ny}{dy} = \frac{nx * ny}{dx * dy}$$

```
def add_rational(x, y):  
    nx, dx = numer(x), denom(x)  
    ny, dy = numer(y), denom(y)  
    return rational(nx * dy + ny * dx, dx * dy)
```

$$\frac{nx}{dx} + \frac{ny}{dy} = \frac{nx * dy + ny * dx}{dx * dy}$$

这些函数给了有
理数的抽象表示

```
rational(n, d) # returns a rational number  
numer(x) # returns the numerator of x  
denom(x) # returns the denominator of x
```


有理数运算

④ 对有理数的其他的一些操作：

```
def print_rational(x):  
    print(numer(x), '/', denom(x))
```

```
def rationals_are_equal(x, y):  
    return numer(x) * denom(y) == numer(y) * denom(x)
```


有理数的表达

- 现在我们来对之前抽象的有理数的表达进行具体的实现
- 我们可以利用python内部 (built-in) 提供的列表 (List)

有理数的表达

```
def rational(n, d):  
    """A representation of the rational number N/D."""  
    return [n, d]
```

Construct a List

```
def numer(x):  
    """Return the numerator of rational number X."""  
    return x[0]
```

Select item from a List

```
def denom(x):  
    """Return the denominator of rational number X."""  
    return x[1]
```

Select item from a List

不可再约分数 (Irreducible fraction) 形式

$$\frac{3}{2} * \frac{5}{3} = \frac{5}{2}$$

$$\frac{15}{6} * \frac{1/3}{1/3} = \frac{5}{2}$$

$$\frac{2}{5} + \frac{1}{10} = \frac{1}{2}$$

$$\frac{25}{50} * \frac{1/25}{1/25} = \frac{1}{2}$$

Greatest common divisor

```
from fractions import gcd
```

```
def rational(n, d):
```

```
    """A representation of the rational number N/D."""
```

```
    g = gcd(n, d) # Always has the sign of d
```

```
    return [n//g, d//g]
```


抽象界限 (Abstraction Barriers)

- 抽象有层次之分
- 每一层次的抽象，应该明晰具体能够做的操作
- 层次分离了使用数据抽象的函数和实现数据抽象的函数。

抽象界限

程序中需要进行有理数的操作

将有理数视作

只能使用

使用有理数执行计算

整个数据的值

```
add_rational,  
mul_rational,  
rationals_are_equal,  
print_rational
```

创造有理数，实现有理数的操作

分子和分母

```
rational, numer, denom
```

实现有理数的构造子和选择子

含有两个元素的列表

```
list, getitem
```

Python对列表的实现层

抽象界限

● 抽象界限的好处：它们使程序更易于维护和修改。

● 依赖于特定的数据表示的部分越少，那么将来修改数据表示时产生的变动也就越少。

抽象界限的违背

没有使用构造子

如果改变有理数的表示，就需要
做很多的修改

```
add_rational( [1, 2], [1, 4] )
```

```
def divide_rational(x, y):  
    return [ x[0] * y[1], x[1] * y[0] ]
```

没有使用选择子

没有使用构造子

数据的表示

● 什么是数据 (Data) ?

● 其实只需要构造子和选择子的行为是正确的即可

● 比如, 对于有理数而言, 如果我们从分子 n 和分母 d 构造出了有理数 x , 那么 $\text{numer}(x) / \text{denom}(x)$ 必须等于 n/d

数据的表示

- ② 数据抽象就是利用构造子和选择子来定义行为！
- ② 对于数据表示的实现，只要其行为满足定义，就是一个有效的表示
- ② 因此，“行为”确定了数据的表示！

有理数实现为函数

```
def rational(n, d):  
    def select(name):  
        if name == 'n':  
            return n  
        elif name == 'd':  
            return d  
    return select
```

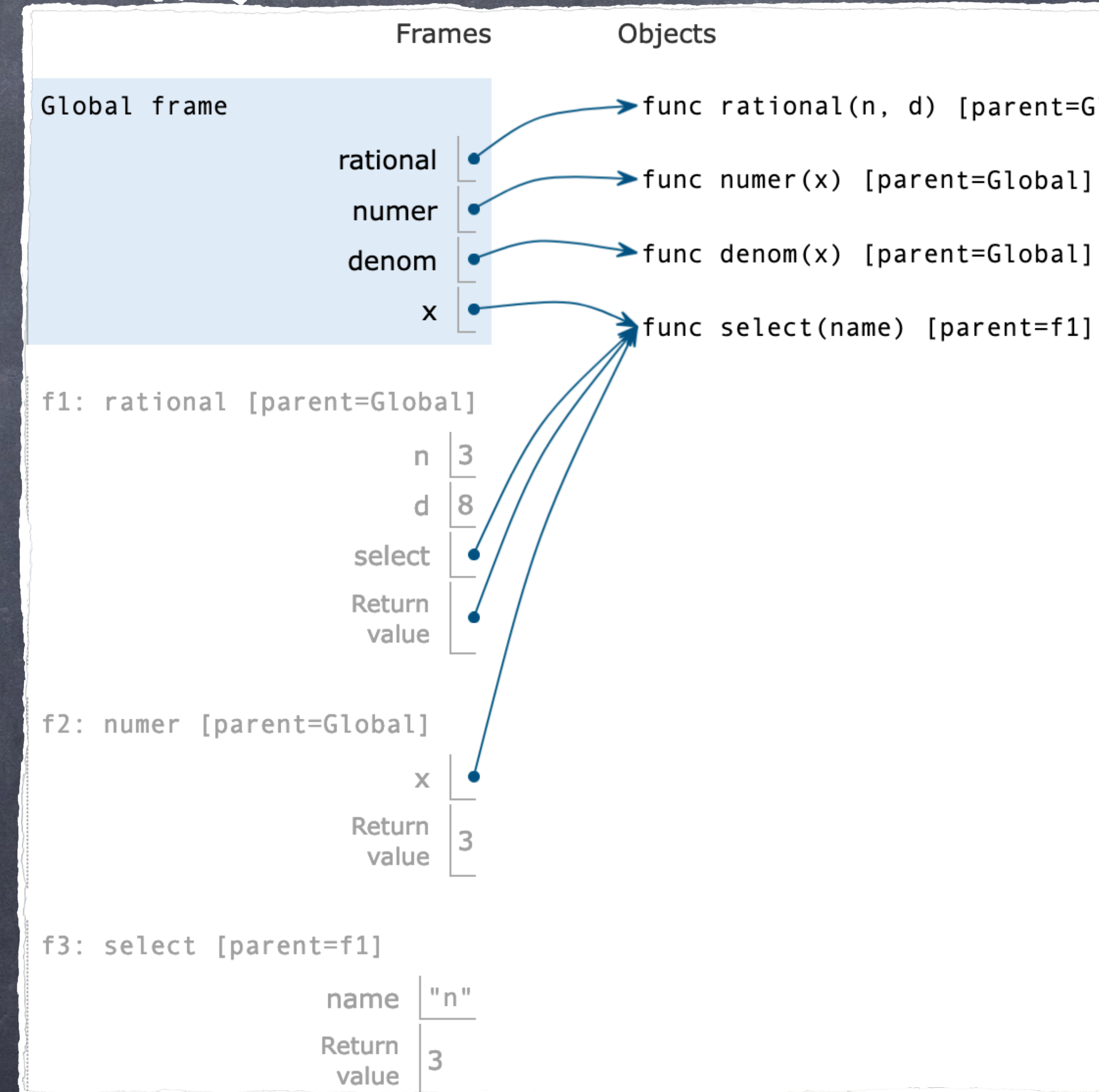
这个函数代表了一个有理数

构造子是一个高阶函数

```
def numer(x):  
    return x('n')
```

选择子调用了x

```
def denom(x):  
    return x('d')
```



Any questions ?