

SHUN NISHITSUJI, SHUYA SATO

オブジェクト指向言語と 関数型言語の対比

研究目的

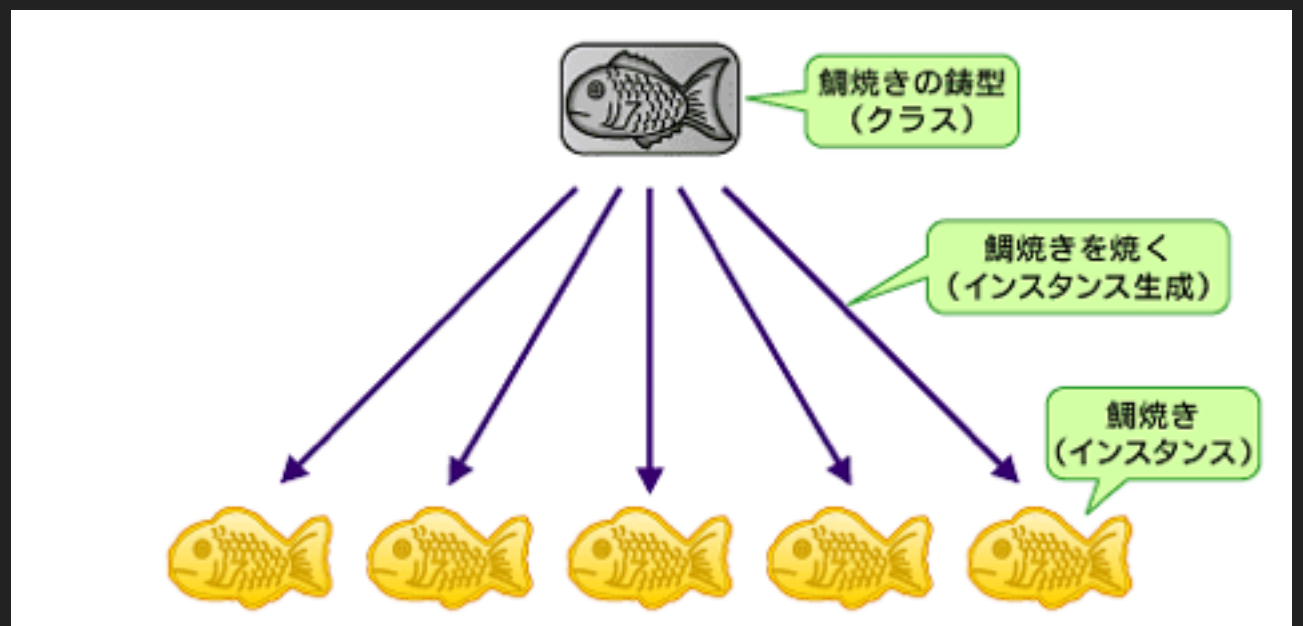
- ▶ オブジェクト指向言語、関数型言語を比較
- ▶ 仕組みの理解
- ▶ それぞれの利点、欠点
- ▶ 何が違うのか？

オブジェクト指向言語(OOPL)について

- ▶ 最初のOOPLは1960年代に登場したSimula
- ▶ 主な言語：C#, JAVA, Ruby, PHP, Python
- ▶ 重要な概念：クラス、継承、ポリモーフィズム
- ▶ 利点：大規模な開発に有利 欠点：可読性が下がる

クラス

- ▶ インスタンスを生成
- ▶ 設計図の役割
- ▶ 同じクラスから、一つ一つ別の性質を持たせることができる
- ▶ コードを整理する



継承

- ▶ 共通点があるものをまとめる
- ▶ 共通部分をまとめるクラス→スーパークラス
- ▶ 異なる部分をまとめて継承するクラス→サブクラス
- ▶ 継承はクラスの管理が目的

ポリモーフィズム

```
1 class Human
2   def initialize(item)
3     @name = item
4   end
5
6   def hello
7     print "Hello, my name is #{@name}.\n"
8   end
9 end
10
11 class Japanese < Human
12   def hello
13     print "よろしく、#{@name} です。 \n"
14   end
15 end
16
17 class Filipino < Human
```

```
18   def hello
19     print "Pagbati, ang pangalan ko ay #{@name}.\n"
20   end
21 end
22
23 def whoareyou(who)
24   puts ""
25   who.hello
26 end
27
28 shun = Filipino.new("Shun")
29 itiki = Japanese.new("イチキ")
30
31 whoareyou(shun)
32 # -> Hello, ito ay Shun.
33 whoareyou(itiki)
34 # -> よろしく、イチキ です。
```

関数型言語(FL)とは

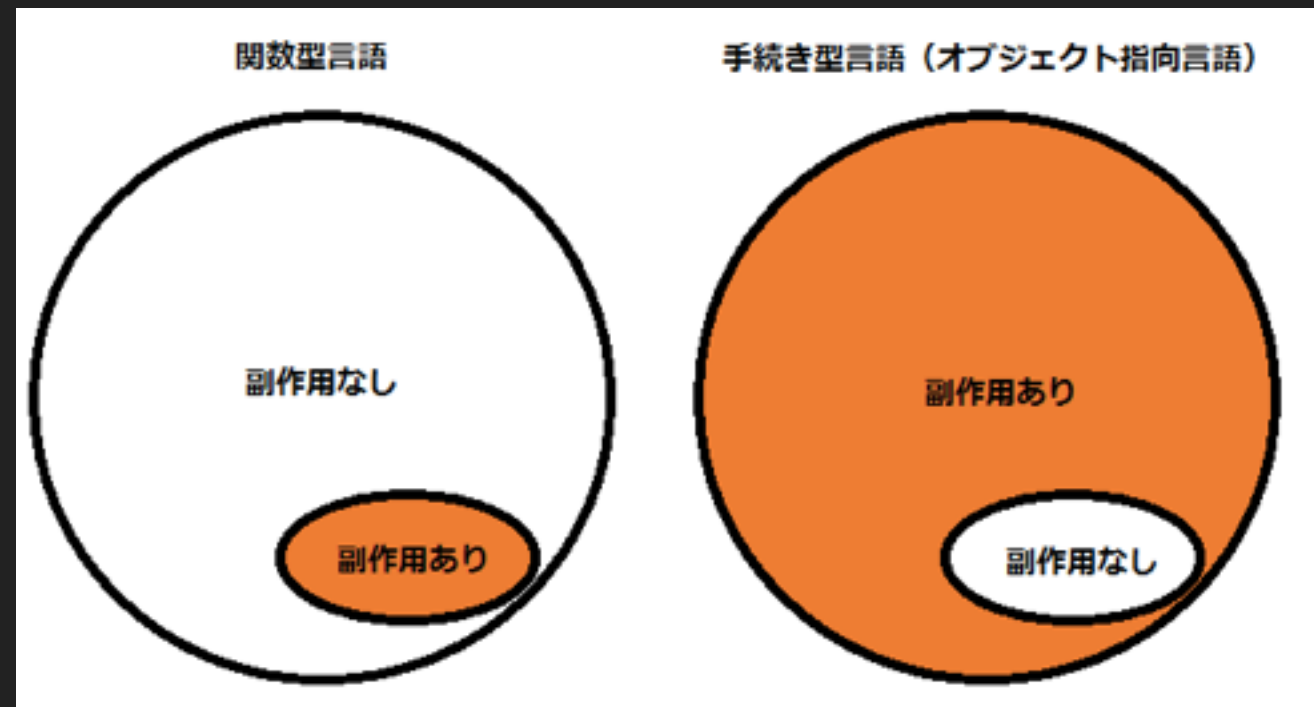
- ▶ 最初のFLは1970年代に登場したLisp
- ▶ 主な言語：Haskell, OCaml, Lisp
- ▶ 重要な特徴：イミュータブル、副作用が生じにくい
- ▶ 利点：バグの少ないコード 欠点：学習コストが高い
- ▶ $f(x) = x + 1 \leftarrow$ 数学
 $f\ x = x + 1 \leftarrow$ Haskell
function f(x) {
 return x + 1; \leftarrow PHP
}

イミュータブル（永続データ）

- ▶ データを作成後に変更できないこと
- ▶ 関数型の世界では状態が変わらない
- ▶ 変数を作る事を束縛と言う
- ▶ 関数も変数も常に同じ結果を返す

副作用とは

- ▶ やりたいこと以外の事が起こること
- ▶ 関数型では代入はしない
- ▶ 副作用が無いことで状態を考慮せずに済む
- ▶ 関数型ではモナドという概念で解決



そんなことない！

関数型のメリット

- ▶ 型がしっかりしてる→正確なプログラミング
- ▶ 関数に副作用がない→保守性が高い
- ▶ 関数を組み合わせやすい→汎用性が高い

迷路アルゴリズム

- ▶ 右図のようなテキストデータを入力
- ▶ SからGの道のりを出力
- ▶ 最短経路を出す

```
*****
*      *
* S *  * *****
*      * *****
*      *
*      *
*****
*
** *****
*      *          G *
* *      ***** *
*      * ***** *
*      *
*      *
```

オブジェクト指向型

- ▶ 言語：Ruby
- ▶ 純粋なオブジェクト指向
- ▶ 動的型付け言語

関数型

- ▶ 言語：Haskell
- ▶ 純粋な関数型
- ▶ 静的型付け言語

DEMO

まとめ

- ▶ それぞれの特徴を捉える
- ▶ どちらも使えるのが理想
- ▶ 考え方を合わせた言語



ご清聴ありがとうございました。