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

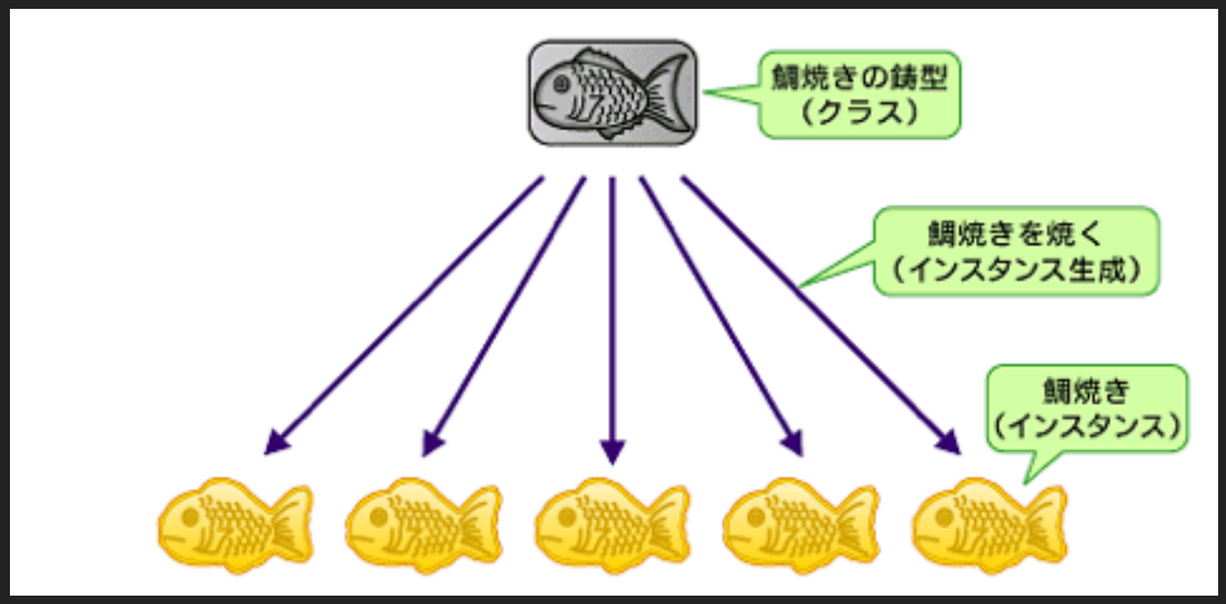
- オブジェクト指向言語、関数型言語を比較
- 仕組みの理解
- それぞれの利点、欠点
- 何が違うのか？

オブジェクト指向について

- 最初の言語は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

- ＝ 設計図
- インスタンスを生成
- 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
- 継承はクラスの管理が目的
- 共通部分をまとめたクラス→スーパークラス
- スーパークラスを継承したクラス→サブクラス

ポリモーフィズム

```ruby
class Human
  def initialize(item)
    @name = item
  end

  def hello
    print "Hello, my name is #{@name}.\n"
  end
end

class Japanese < Human
  def hello
    print "よろしく、#{@name} です。\n"
  end
end

class Filipino < Human
  def hello
    print "Pagbati, ang pangalan ko ay #{@name}.\n"
  end
end

def whoareyou(who)
  puts ""
  who.hello
end

shun = Filipino.new("Shun")
itiki = Japanese.new("イチキ")

whoareyou(shun)
# -> Hello, ito ay Shun.
whoareyou(itiki)
# -> よろしく、イチキ です。
```

関数型とは　- 1

- 最初の言語は1970年代に登場したLisp
- 　主な言語：　　Haskell, OCamel, Elixir
- 　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

関数型とは　-2

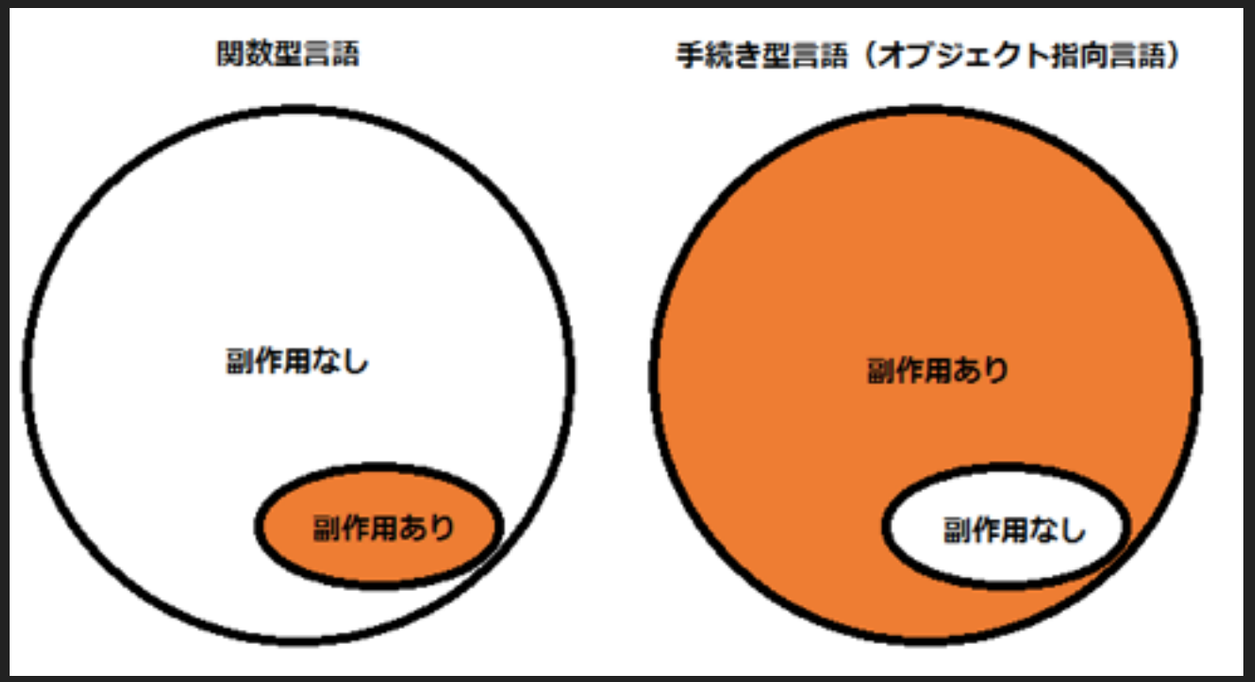
- コードは全て関数によって処理
- 関数型の関数は数学の関数とほぼ同義
- 数学：　　f(x) = x + 1
- Haskell：　f x = x + 1
- Ruby：　　def f(x)
  return x + 1;
  end

イミュータブル

- 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

- 期待している結果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



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

オブジェクト指向言語

- 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

- 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
- 相互の長所を活かす

参考文献

- 平澤 章　オブジェクト指向でなぜつくるのか
- Miran Lipovaca すごいHaskell たのしく学ぼう！
- 大川 徳之　関数プログラミング実践入門
- Bryan O'Sullivan, John Goerzen, Don Stewart　Real World Haskell

---

ご清聴ありがとうございました。