Functional Programming

Математика программирования

Что делает FP … FP ?

- Иммютабельность
- Декларативный стиль
- Чистые функции
- Анонимные функции
- Функции вышего порядка

Что такое иммютабельность? И зачем?

```
x = something # immutable type
print x
func(x)
print x              # prints the same thing

x = something # mutable type
print x
func(x)
print x             # might print something different
```

Декларативный стиль

```
def fun(value):
  return value + 1

data = [1, 2, 3]

map(fun, data) # [1, 2, 3] -> [2, 3, 4]
```

“Грязные функции” или сайд-эффекты

```
some_global_list = []

def some_operation(value):
    some_global_list.append(value)

some_operation('data')
```

Чистые функции

```
def some_operation(value, data):
    return data + [value]

some_local_list = []
new_list = some_operation('data', some_local_list)
```

И что же такое Referential Transparency?

```
def fun(value)
       return value + 1

fun(something)    ⇔    something + 1      # как в математике ^_^
```

Анонимные функции

```
fun = lambda x: x + 1

data = [1, 2, 3]

map(fun, data) # [1, 2, 3] -> [2, 3, 4]
```

Функции высшего порядрка

- Функции, что оперируют функциями
- Могут принимать функции
- Могут возвращать функции
- map, filter, reduce из модуля functools

Теория Категорий, не пугайтесь

- Категории, Объекты и Морфизмы
- Функторы
- Монады

Работа со значениями в контексте

```
def divM(value, divider):
        ...

div100by = lambda x: divM(100, x)

def sqrtM(value):
    ...


application = returnM(4) flatMap div100by flatMap sqrtM

error, result = application.result
```

Что можно представить контекстом?

- Опциональность (А или пусто)
- Вариативность (или А или Б)
- Асинхронность (в итоге А)
- Комбинирование (А или список Б)
- Успех выполнения (А или ошибка Б)
- Все что угодно

Монады

- Return : A => M[A]
- Bind : (M[A], A => M[B]) => M[B], высший порядок

Простая монада Maybe из Haskell

- Опциональность
- Return: A => Maybe(A), но по сути Maybe
- Bind: A => Maybe(B) => Maybe(B)
- Just
- Nothing

Пример в Python

```
class _MaybeMonad(object):

    def __init__(self, just=None, nothing=False):
        self._just = just
        self._nothing = nothing

    def flatMap(self, fun):
        if not self._nothing:
            res = fun(self._just)
            assert isinstance(res, _MaybeMonad), ("In Maybe context function result must be Maybe")
            self._just = res._just
            self._nothing = res._nothing
        return self

    def result(self):
        return (self._nothing, self._just)
```

Вспомогательные операции

```
nothing = lambda: _MaybeMonad(nothing=True)

just = lambda val: _MaybeMonad(just=val)

returnM = just
```

А теперь как использовать

```
def divM(value, divider):
    if divider:
        return just(value // divider)
    return nothing()


def sqrtM(value):
    if value >= 0:
        return just(math.sqrt(value))
    return nothing()
```

application = returnM(4) flatMap div100by flatMap sqrtM

Но к сожелнию Python не для этого

- Императивный стиль
- Итераторы
- Генераторы
- Немного лямбд
- Явность