Pre-URP 0804

Function 재구현

1. Set 인터페이스 및 관련 구조체 수정

Differential 함수를 구현할 때 미분연산자 $\frac{d}{dx}$ 의 domain과 codomain 등을 설정하기 어렵다고 판단
Map 인터페이스와 Function 구조체를 이용해 역할을 분담하여 미분은 Map으로서 보도록 했었음.

구현하던 중 Map 인터페이스의 필요성이 없다고 판단
Differential 함수를 Function으로 볼 방법을 모색함.

```
type Set interface {
        Contains(any) bool
}

type GeneralSet struct {
        ElementType reflect.Type
}
```

기존 Number 구조체가 가지는 Contains 메소드에 주목

ElementType을 통해 포함관계를 체크하는 방식

이는 정수, 실수 등의 집합에만 적용할 수 있는 성질이 아님

```
var Real = GeneralSet{reflect.TypeOf(1.00)}

var Integer = GeneralSet{reflect.TypeOf(1)}

func (N GeneralSet) Contains(x any) bool {

        if reflect.TypeOf(x) == N.ElementType {
                return true
        }
        return false
} // GeneralSet 구조체는 전부 Set 인터페이스의 조건을 만족한다.
```

함수 집합을 대변하는 FunctionSpace 변수를 GeneralSet 구조체로 선언

```
type Function struct {
        domain      GeneralSet
        codomain    GeneralSet
        maprelation func(x any) any
}

var FunctionSpace = GeneralSet{reflect.TypeOf(exfunc1)}
```

$\frac{d}{dx} : \mathcal{F}(x) \to \mathcal{F}(x)$ 의 함수로 생각할 수 있음

```
func Differential(F Function) Function {
        var Derivative Function
        Derivative.domain = F.domain
        Derivative.codomain = F.codomain
        Derivative.maprelation = Diffrel(F)
        return Derivative
}
```

```
 func Diffrel(F Function) func(x any) any {
        return func(x any) any {
                if IsSafe(x, Real) {
                        point := x.(float64)
                        slope := ((F.Computation(point + epsilon)).(float64)
                                - (F.Computation(point)).(float64))
                                / epsilon
                        return slope
                }
                return nil
        }
}
```

```
func DefiniteIntegral(F Function, a, b float64) float64 {
        if IsSafe(a, Real) && IsSafe(b, Real) {
                n := bigNum
                h := (b - a) / n
                result := 0.0
                for i := 0; i <= int(n); i++ {
                        x := a + float64(i)*h
                        fx := F.Computation(x).(float64)
                        if i == 0 || i == int(n) {
                                result += fx / 2
                        } else {
                                result += fx
                        }
                }
                return result * h
        }
        return math.NaN()
}
```