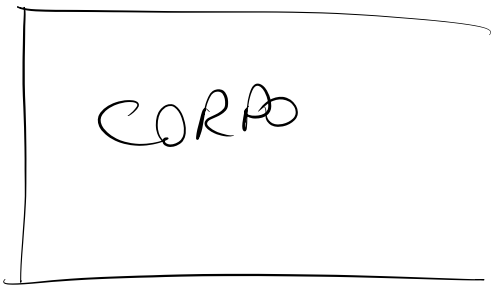
TIPI FUNZIONALI

FUNZIONI hanno un tipo

- `func Pippo(x int, y float64) bool {`



→ SEGNATURA (o tipo) DELLA FUNZIONE PIPPO

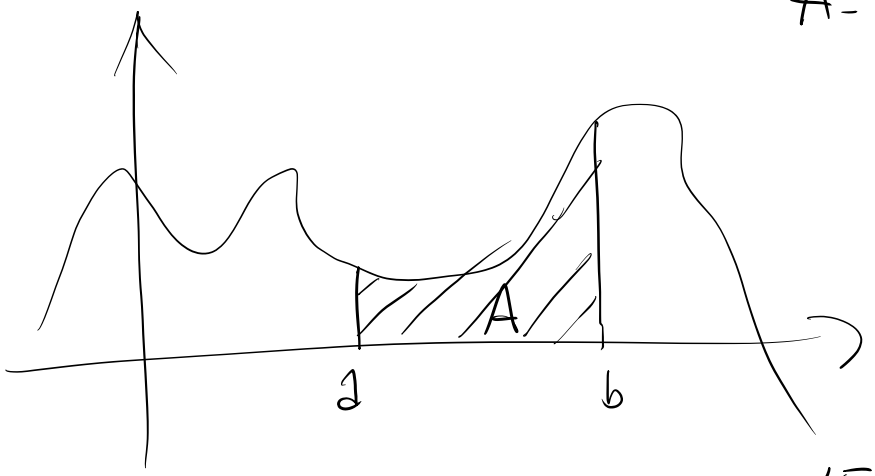
```
func (int, float64) bool
```

```
func main() {
    var f func(int, float64) bool
    f = Pippo          ← funzione top-level
    x := f(5, 3.14)
    f = Pluto
    y := f(7, 3.16)
    f = func(x int, y float64) bool { return int(q)==x }
    fmt.Println(f(7, 7.4))
```

- bubola
- funzione anonima
- chiusura

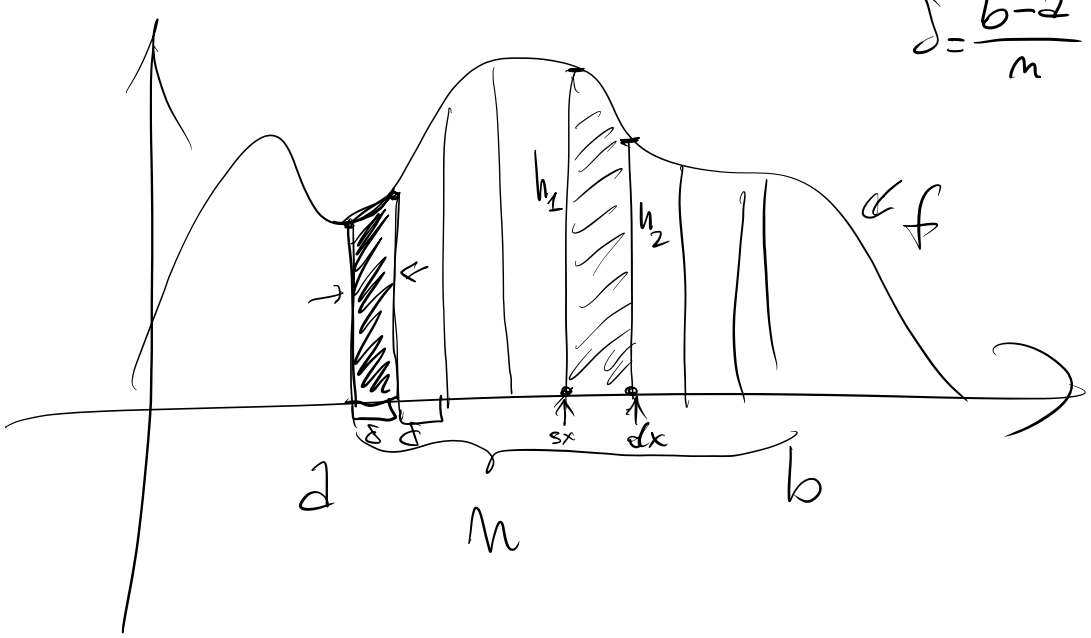
INTEGRAZIONE NUMERICA

$$A = \int_a^b f(x)\,dx$$



f

$$\int_7^{15} x^2\,dx = \left[\frac{x^3}{3}\right]_7^{15} = \frac{15^3}{3} - \frac{7^3}{3}$$



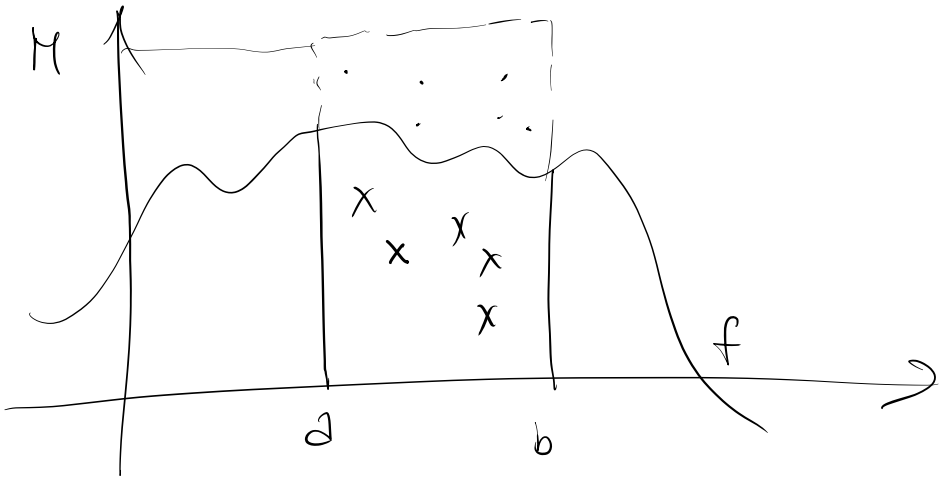
$$\delta = \frac{b-a}{n}$$

```
type funzione func(float64) float64

func integra(f funzione, a float64, b float64, n int) float64 {
    s := 0.0
    delta := (b - a) / float64(n)
    for i := 0; i < n; i++ {
        sx := a + delta * float64(i)
        dx := sx + delta
        h1 := f(sx)
        h2 := f(dx)
        s += (h1 + h2) * delta / 2
    }
    return s
}

func main() {
    area := integra(math.Sin, 7, 15, 100)
    areaAttesa := -math.Cos(15) + math.Cos(7)
    area2 := integra(func(x float64) float64 { return x*x }, 0, 15, 100)
```

1E-5




```
func integraMC(f funzione, a float64,
b float64, M float64, n int) float64 {

    crocette := 0
    for i := 0; i < n; i++ {
        x := rand.Float64()*(b-a) + a
        y := rand.Float64()*M
        if f(x) >= y {
            crocette++
        }
    }
    return (b-a)*M * float64(crocette)/float64(n)
}
```

integra_test.go

```
type testCase struct {
    f  funzione  // Una funzione (e.g. x^2)
    F  funzione  // Una sua primitiva (e.g. x^3/3)
}

func testThis(t *testing.T, tc testCase) {
    a := rand.Float64()*200.0 + 100.0
    b := a + rand.Float64()*200.0
    areaOttenuta := integra(tc.f, a, b, 10000)
    areaAttesa := tc.F(b) - tc.F(a) // Teor. fond. calcolo
    if math.Abs(areaOttenuta - areaAttesa) > 1E-5 {
        t.Error("....")
    }
}

func IntegraTest(t *testing.T) {
    var s []testCase
    s = []testCase{
        {math.Sin, func(x float64) float64
            {return -Math.Cos(x)}},
        {math.Exp, math.Exp},
        {func(x float64) float64 {return x**x},
         func(x float64) float64 {return x*x*x/3}}
    }
```

```
    for _, tc := range s {
        testThis(t, tc)
    }
}
```