# Lambda core - hard core

Jarek Ratajski

@jarek000000

# Agenda

# Agenda

What is a Lambda?

# Agenda

What is a Lambda?
How to reinvent math

# Agenda

What is a Lambda?
How to reinvent math
Lambda calculi in Scala

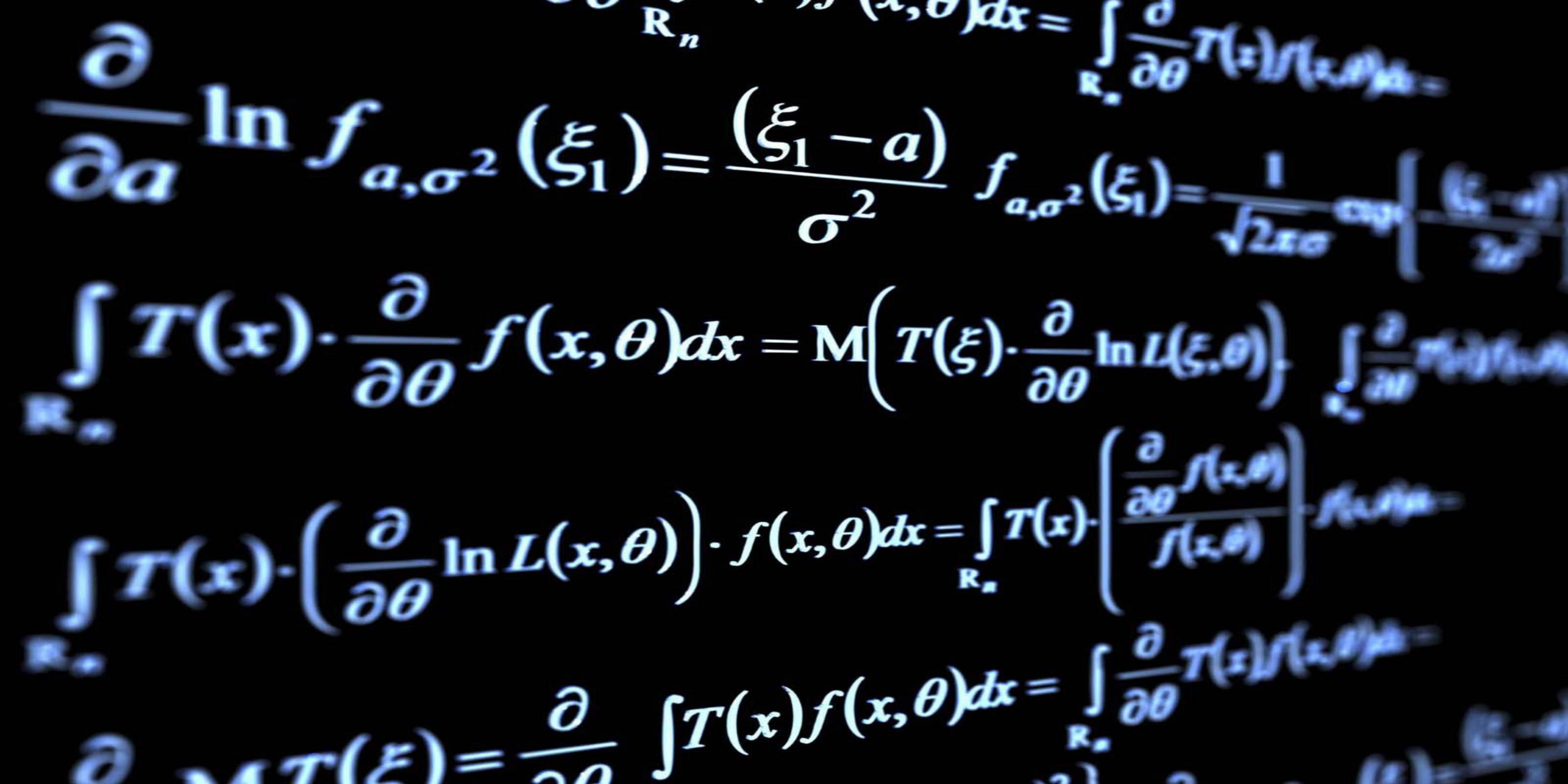
# Agenda

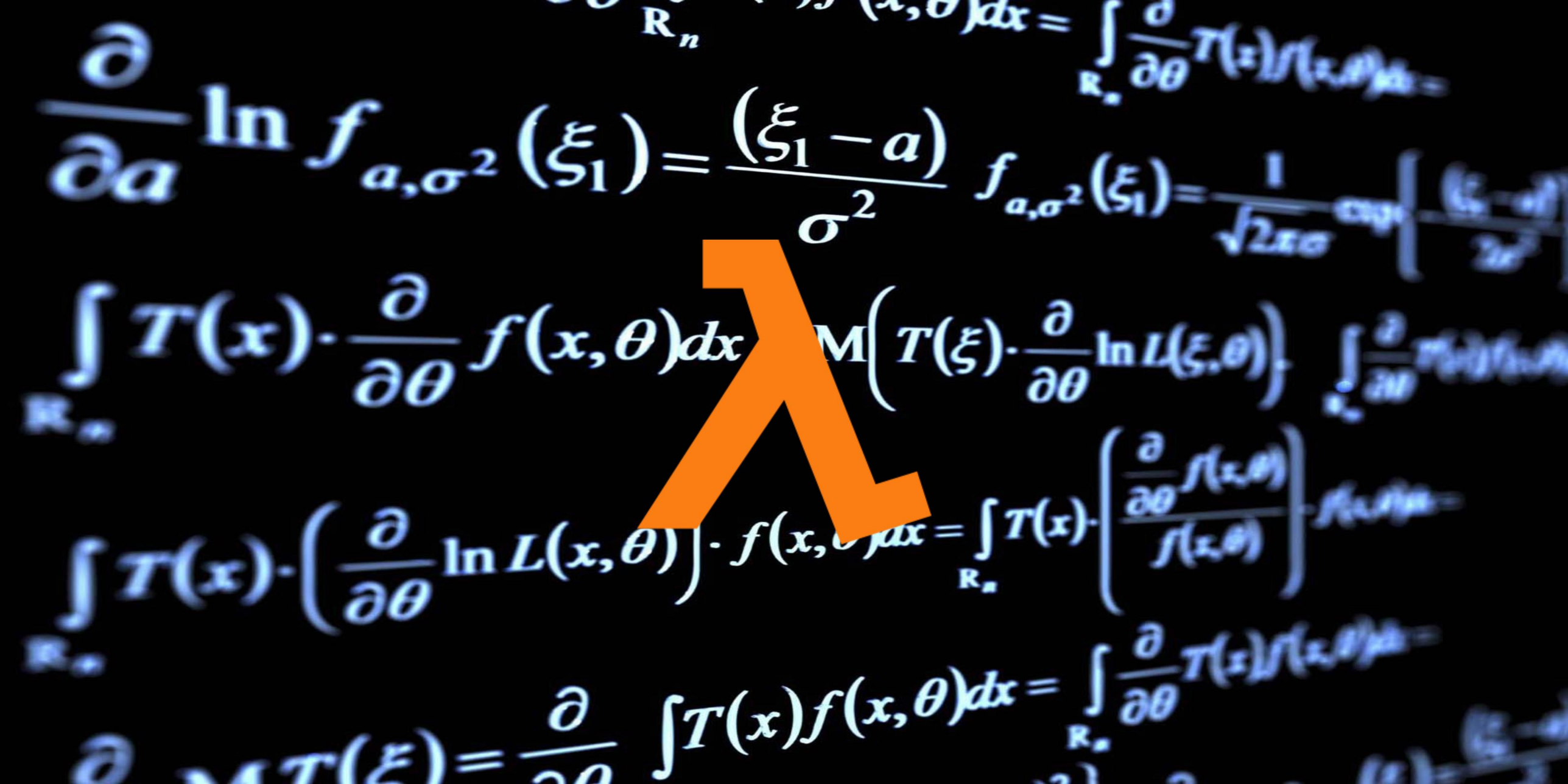
What is a Lambda?
How to reinvent math
Lambda calculi in Scala
Consequences...


@jarek000000

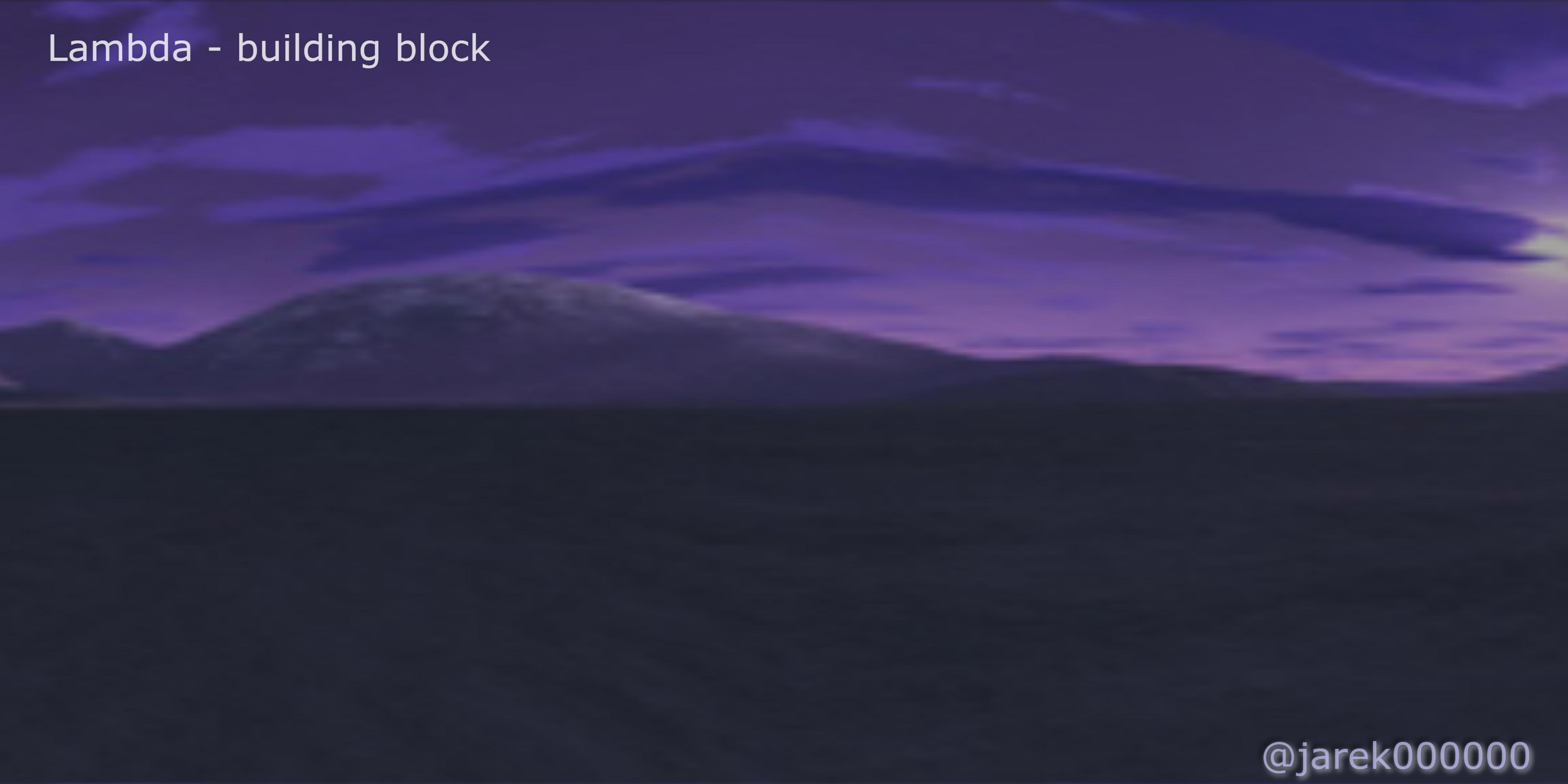

designed by freepik.com


WENGER


$$\frac{\partial}{\partial a}\ln f_{a,\sigma^2}(\xi_1)=\frac{(\xi_1-a)}{\sigma^2}\, f_{a,\sigma^2}(\xi_1)=\frac{1}{\sqrt{2\pi}\sigma}$$
$$\int_{R_n} T(x)\cdot\frac{\partial}{\partial\theta}f(x,\theta)dx=\mathrm{M}\left(T(\xi)\cdot\frac{\partial}{\partial\theta}\ln L(\xi,\theta)\right)$$
$$\int_{R_n} T(x)\cdot\left(\frac{\partial}{\partial\theta}\ln L(x,\theta)\right)\cdot f(x,\theta)dx=\int_{R_n} T(x)\cdot\left(\frac{\frac{\partial}{\partial\theta}f(x,\theta)}{f(x,\theta)}\right)$$
$$\frac{\partial}{\partial\theta}\mathrm{M}T(\xi)=\frac{\partial}{\partial\theta}\int T(x)f(x,\theta)dx=\int_{R_n}\frac{\partial}{\partial\theta}$$

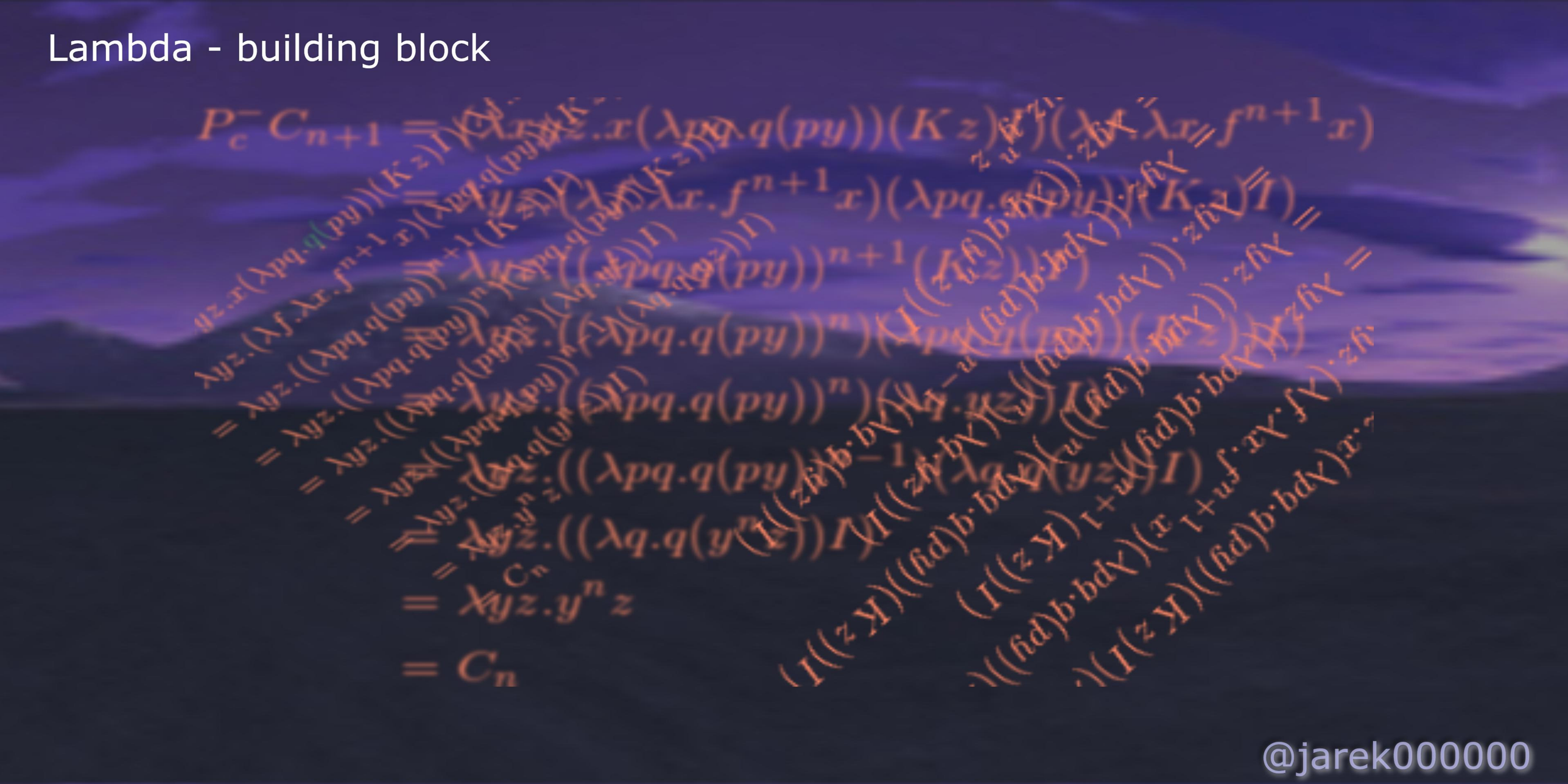

λ

# Lambda - building block



@jarek000000

# Lambda - building block



# Lambda - building block

*Lambda* is a function that takes some *lambda* as parameter and produces another *lambda*

# Lambda - building block

*Lambda* is a function that takes some *lambda* as parameter and produces another *lambda*
define lambda:

# Lambda - building block

*Lambda* is a function that takes some *lambda* as parameter and produces another *lambda*

define lambda:

λx. some expression here

# Lambda - building block

*Lambda* is a function that takes some *lambda* as parameter and produces another *lambda*

define lambda:

    λx. some expression here

apply lambdas:

# Lambda - building block

*Lambda* is a function that takes some *lambda* as parameter and produces another *lambda*

define lambda:

```
λx. some expression here
```

apply lambdas:

```
x (y)
```

# Lambda - building block

*Lambda* is a function that takes some *lambda* as parameter and produces another *lambda*

define lambda:

```
λx. some expression here
```

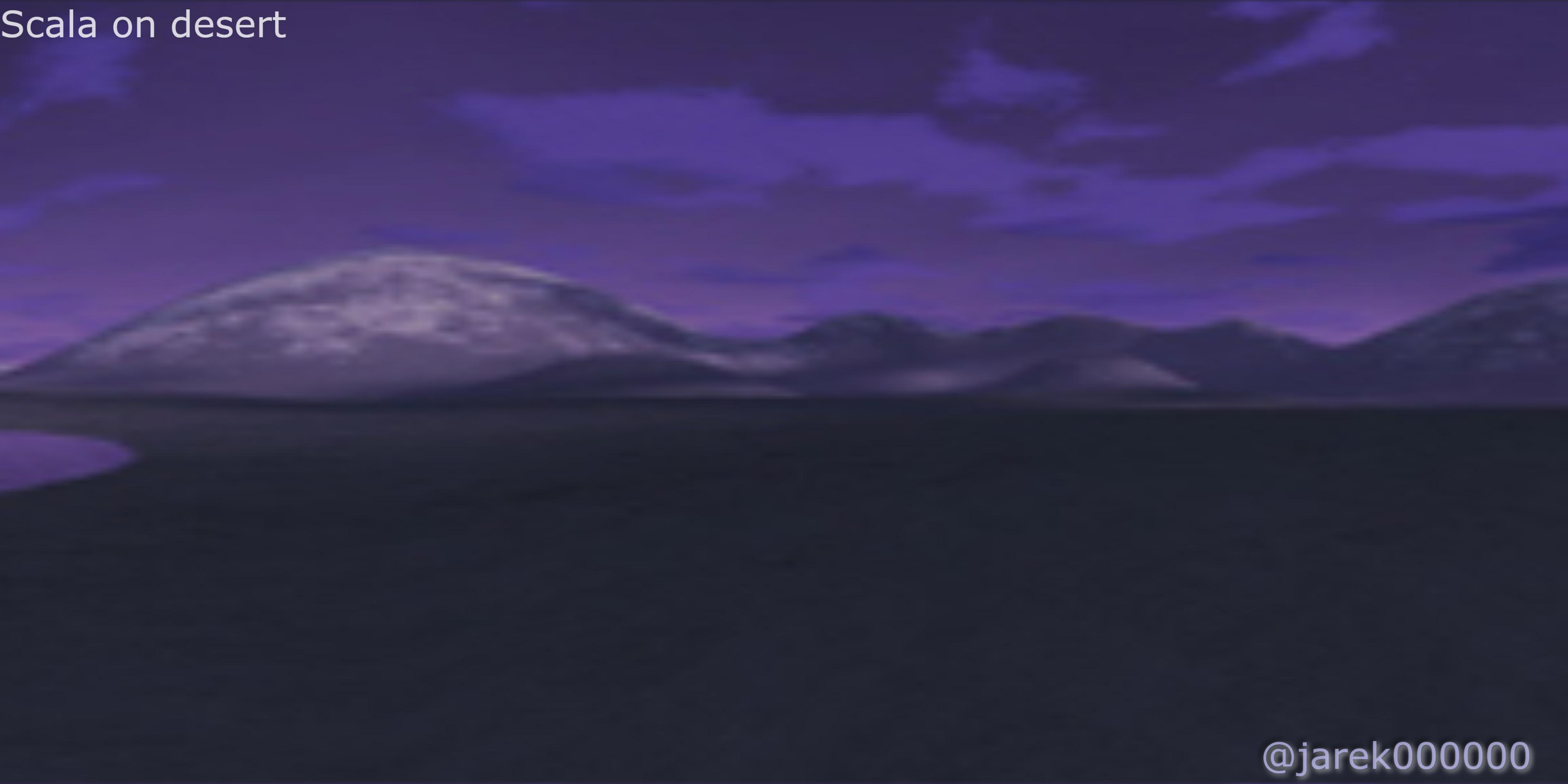
apply lambdas:

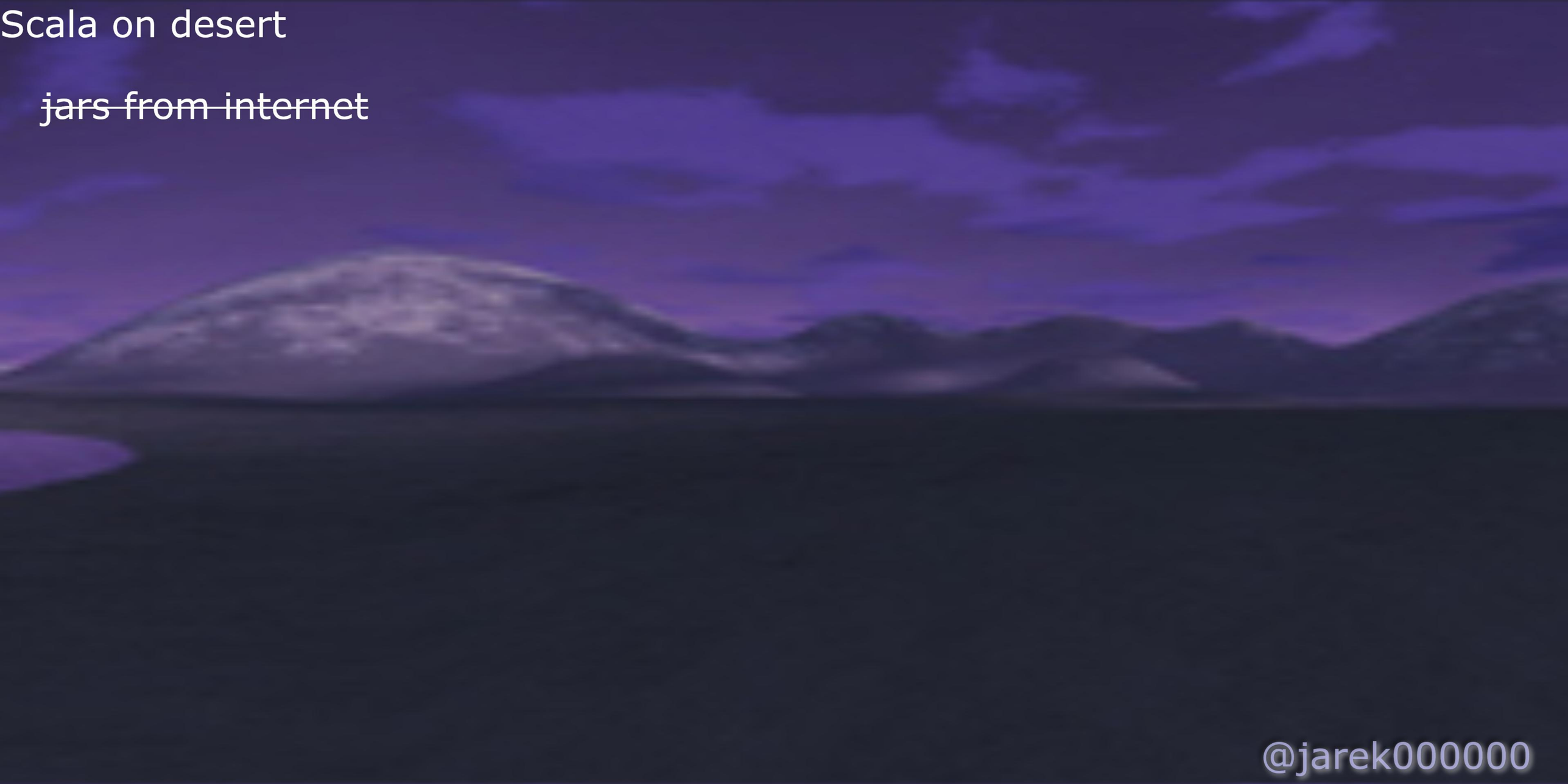
```
x (y)
```

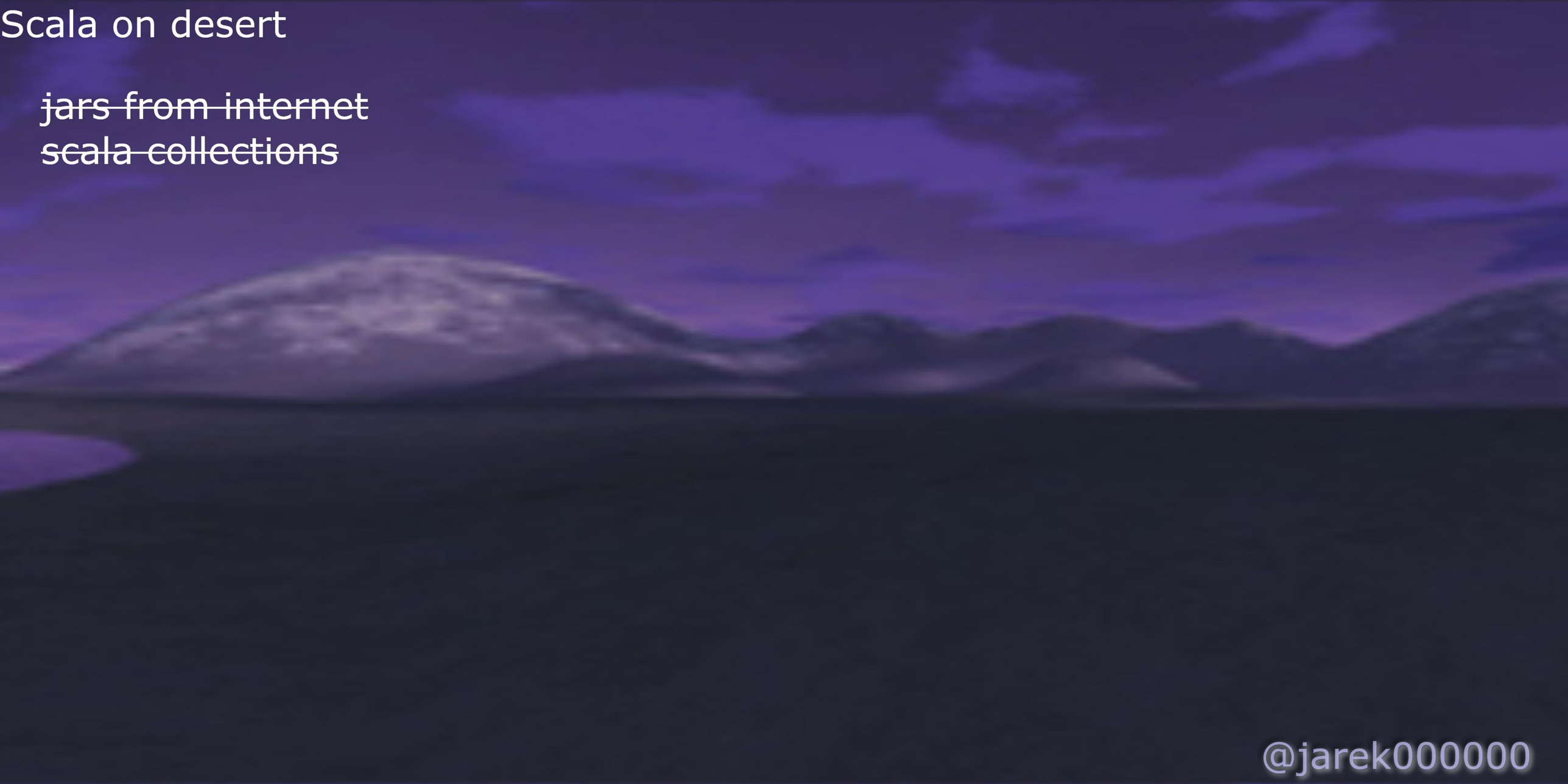
... that is all (*)

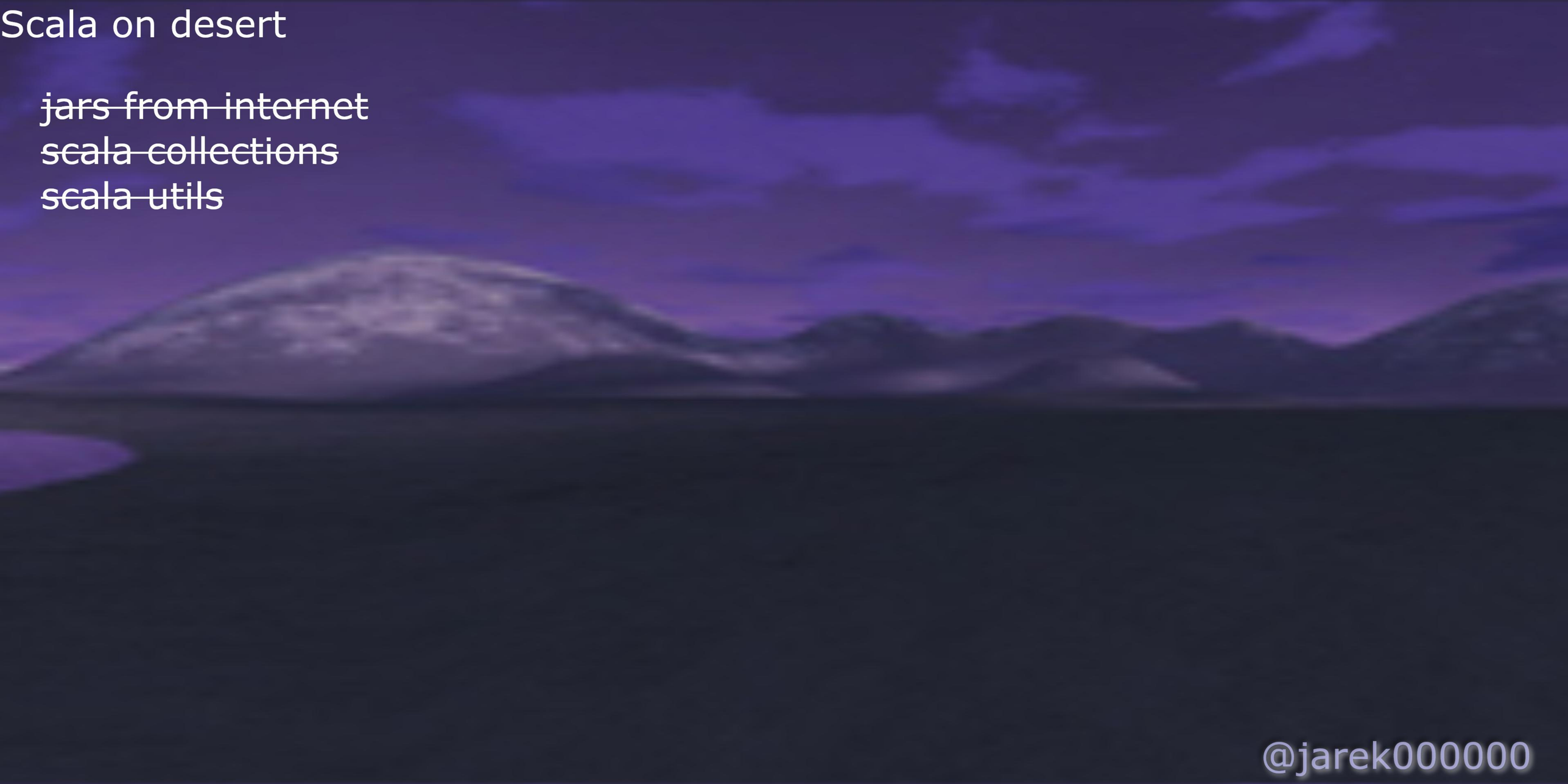
# What is Lambda?

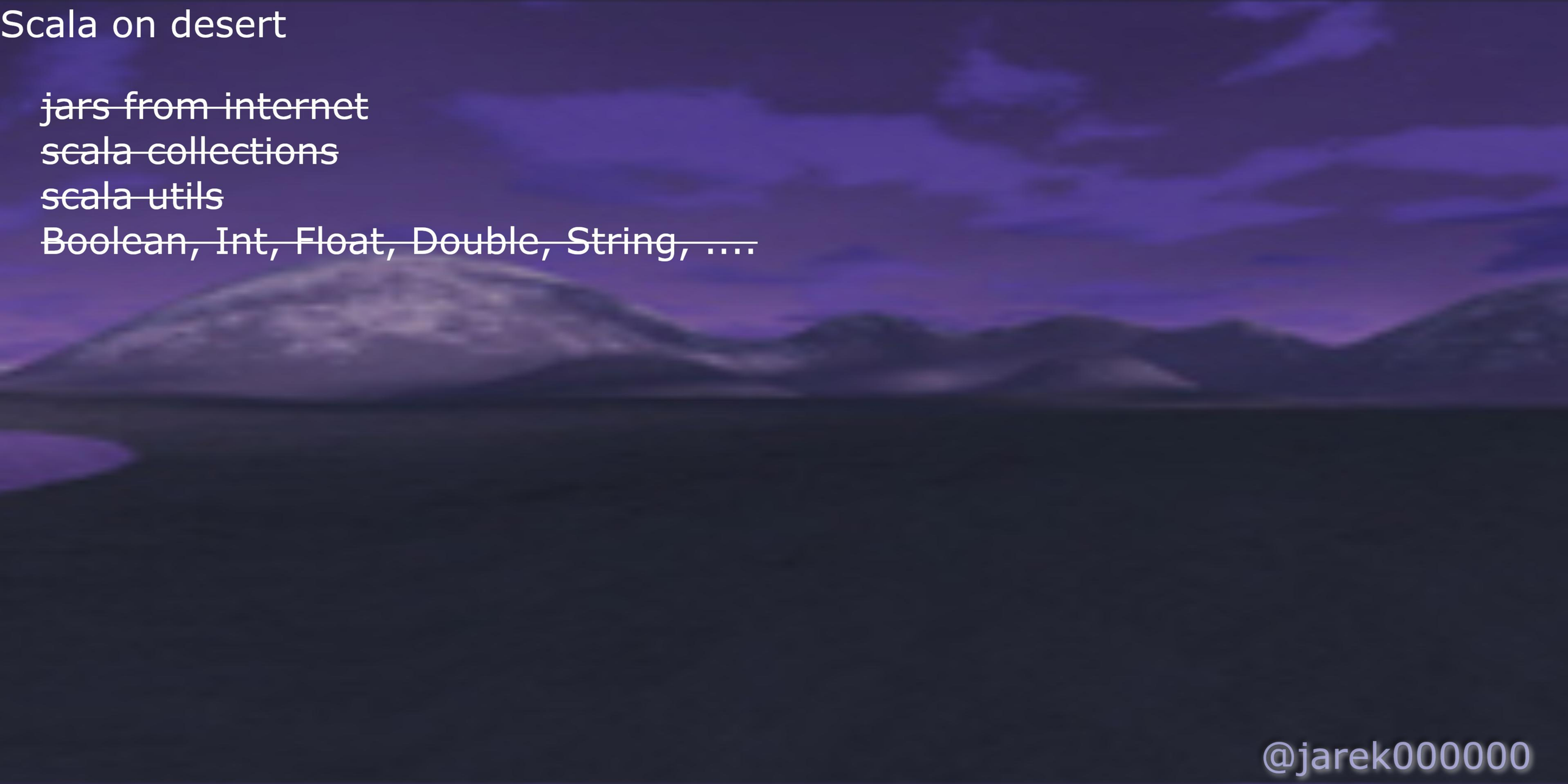
```
trait Lambda {
    def apply(x : Lambda) : Lambda
}
```

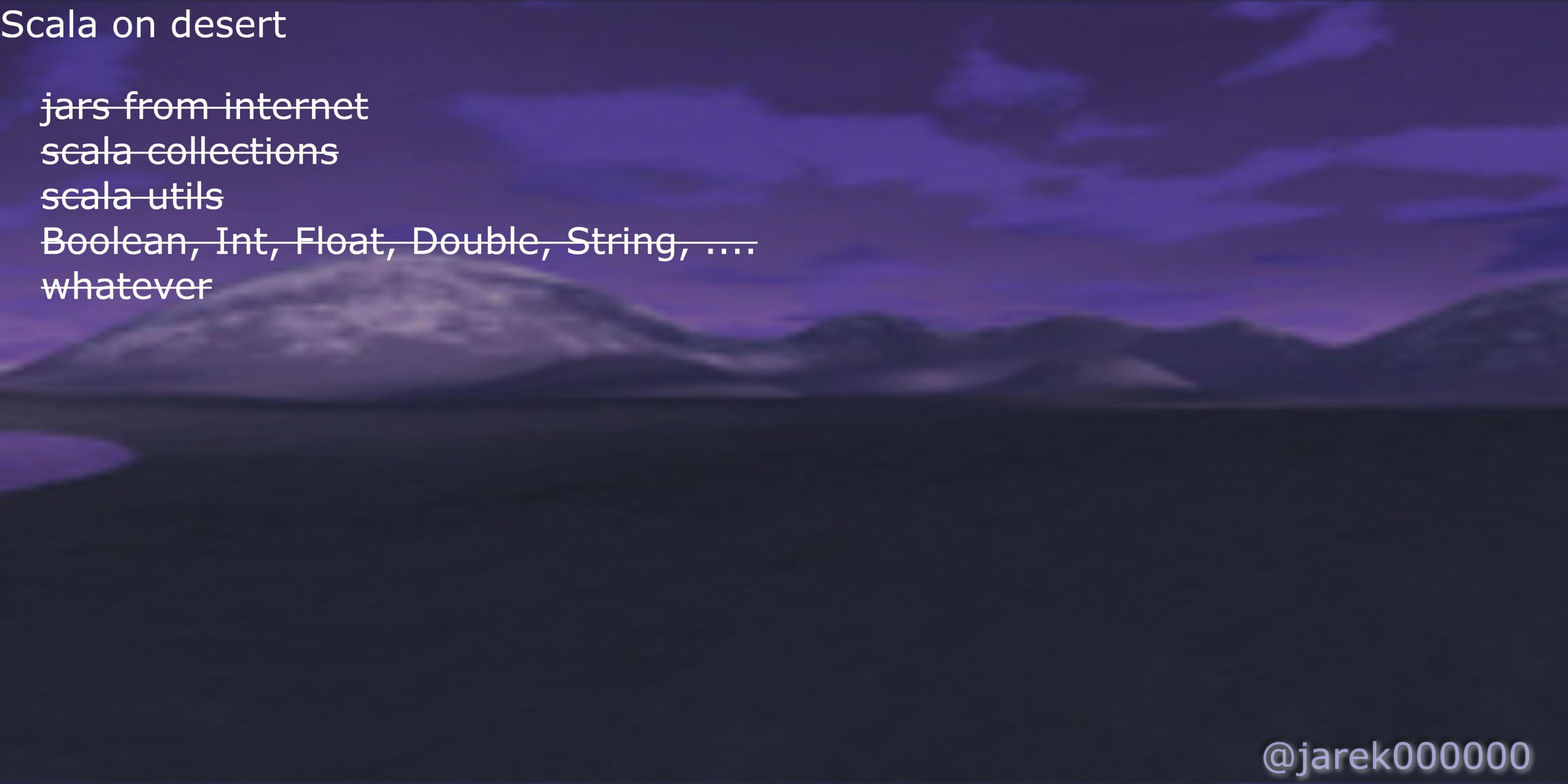

Scala on desert
@jarek000000


Scala on desert
~~jars from internet~~
@jarek000000


Scala on desert
~~jars from internet~~
~~scala collections~~
@jarek000000


Scala on desert
~~jars from internet~~
~~scala collections~~
~~scala utils~~
@jarek000000


Scala on desert
~~jars from internet~~
~~scala collections~~
~~scala utils~~
~~Boolean, Int, Float, Double, String, ....~~
@jarek000000


Scala on desert
~~jars from internet~~
~~scala collections~~
~~scala utils~~
~~Boolean, Int, Float, Double, String, ....~~
~~whatever~~
@jarek000000

# Scala on desert

~~jars from internet~~
~~scala collections~~
~~scala utils~~
~~Boolean, Int, Float, Double, String, ....~~
~~whatever~~
one exception:

# Scala on desert

~~jars from internet~~
~~scala collections~~
~~scala utils~~
~~Boolean, Int, Float, Double, String, ....~~
~~whatever~~
one exception:
displaying results for you - **badlam** package

```scala
package badlam

import pl.setblack.badlam.Lambda
import pl.setblack.badlam.analysis.SmartDisplay

object  Playfield {

  def main(args: Array[String]): Unit = {
    val identity:Lambda = (x) => x
    println(SmartDisplay.web.display(identity))

    val aTrue:Lambda = (x) => (y) => x
    println("true ~ " + SmartDisplay.web.display(aTrue))

    val aFalse:Lambda = (x) => (y) => y
    println("false ~ " + SmartDisplay.web.display(aFalse))

    val and:Lambda = (p) => (q) => p(q)(p)
    println("and ~ " + SmartDisplay.webP.display(and))

    val result1:Lambda = and (aFalse) (aTrue)
    println("resutlt1 ~ " + SmartDisplay.web.display(result1))

    val or:Lambda = (p) => (q) => p(p)(q)
    println("or ~ " + SmartDisplay.webP.display(or))

    val result2:Lambda = or(aFalse) (aTrue)
    println("result2 ~ " + SmartDisplay.web.display(result2))
```

```scala
package badlam

import pl.setblack.badlam.analysis.SmartDisplay
import pl.setblack.badlam.{Boolean, Cardinals, Lambda}

object  Playfield {

  def main(args: Array[String]): Unit = {
    val aTrue:Lambda = (x) => (y) => x
    val aFalse:Lambda = (x) => (y) => y

    val zero:Lambda = f=> x => x
    val one:Lambda = f=> x => f(x)
    val two:Lambda = f=> x => f(f(x))
    val succ:Lambda = (n) => (f) => (x) => f(n(f)(x))

    val mult:Lambda = (m) => (n) => (f) => m(n(f))

    val pred:Lambda = (n) => (f) => (x) => n((g) => (h) => h(g(f)))((u) => x)((u) => u)

    val ifLambda:Lambda = (c) => (t) => (f) => c(t)(f)((x) => x)

    val isZero:Lambda = (n) => n((x) => aFalse)(aTrue)

    val autocall:Lambda = (x) => x(x)
    val Y:Lambda = (f) => autocall((y) => f((v) => y(y)(v)))

    val G:Lambda = (r) => (n) =>
      (ifLambda(isZero(n))
```

# We have seen

# We have seen

Minimalistic and elegant formalism

# We have seen

Minimalistic and elegant formalism

That is powerful enough to represent/define

# We have seen

- Minimalistic and elegant formalism
- That is powerful enough to represent/define numbers

# We have seen

Minimalistic and elegant formalism

That is powerful enough to represent/define
- numbers
- functions

# We have seen

Minimalistic and elegant formalism

That is powerful enough to represent/define
- numbers
- functions
- algorithms

# We have seen

Minimalistic and elegant formalism

That is powerful enough to represent/define
- numbers
- functions
- algorithms

(1937) Lambda calculus and Turing machine are equivalent. (Lambda calculus is turing complete).

# The end

If you ant to know more...

- Wikipedia (a lot on topic)
- You may use **Badlam**(github) library to display lambdas (!)
- Think about eager, lazy evaluations
- Typed lambda calculus - base of type systems
- Nice trick to try: Lambda calculus with Scala type system

Thank You!

@jarek000000


@jarek000000