**Lambda calculus** (also written as **λ-calculus**) is a [formal system](https://en.wikipedia.org/wiki/Formal_system" \o "Formal system) in [mathematical logic](https://en.wikipedia.org/wiki/Mathematical_logic" \o "Mathematical logic) for expressing [computation](https://en.wikipedia.org/wiki/Computability" \o "Computability) based on function [abstraction](https://en.wikipedia.org/wiki/Abstraction_(computer_science)" \o "Abstraction (computer science)) and [application](https://en.wikipedia.org/wiki/Function_application" \o "Function application) using variable [binding](https://en.wikipedia.org/wiki/Name_binding" \o "Name binding) and [substitution](https://en.wikipedia.org/wiki/Substitution_(algebra)" \o "Substitution (algebra)).

It is a universal [model of computation](https://en.wikipedia.org/wiki/Model_of_computation" \o "Model of computation) that can be used to simulate any [Turing machine](https://en.wikipedia.org/wiki/Turing_machine" \o "Turing machine). It was introduced by the mathematician [Alonzo Church](https://en.wikipedia.org/wiki/Alonzo_Church" \o "Alonzo Church) in the 1930s as part of his research into the [foundations of mathematics](https://en.wikipedia.org/wiki/Foundations_of_mathematics" \o "Foundations of mathematics).

Lambda calculus consists of constructing lambda terms and performing reduction operations on them. In the simplest form of lambda calculus, terms are built using only the following rules

|  |  |  |
| --- | --- | --- |
| Syntax | Name | Description |
| *x* | Variable | A character or string representing a parameter or mathematical/logical value. |
| (λ*x*.*M*) | Abstraction | Function definition (*M* is a lambda term). The variable *x* becomes [bound](https://en.wikipedia.org/wiki/Free_variables_and_bound_variables" \o "Free variables and bound variables) in the expression. |
| (*M* *N*) | Application | Applying a function to an argument. *M* and *N* are lambda terms. |

Que: What is the notion of a function from a computational perspective.

Ans: Lambda Calculus

function (black box)

λ(x) λ(x). x + 1

x => x+1

5 => 5+1 = 6

λ(x).λ(y) => λ(x). λ(y). x + y

x, y => x+y

l = (lambda x, y: x+y)(5, 10)

print(l) #15

l1 = (lambda x: x+1)(5)

print(l1) #6