What is a lambda (function)?

Asked 16 years, 4 months ago Modified 12 months ago

Viewed 406k times

850

For a person without a comp-sci background, what is a lambda in the world of Computer Science?

lambda language-agnostic computer-science terminology theory

Share

Improve this question

Follow

edited Jul 4, 2014 at 6:04

nawfal

73k ● 58 ● 336 ● 376

asked Aug 19, 2008 at 16:20

Brian Warshaw

23k ● 9 ● 54 ● 72

3 amda expression explained here beautifully. – Jameer Mulani Dec 4, 2017 at 16:35

1 see this for how lambda functions look in various languages, and in what year they were introduced in those languages. – Philip Schwarz Sep 15, 2019 at 7:12

The noun "lambda" is an abbreviation of a number of things. Even in computer science, it can refer to a lambda expression or to a lambda function. So you would have to

clarify which you mean. en.wiktionary.org/wiki/lambda#Noun
– Philippe Cloutier Sep 11, 2023 at 21:14

24 Answers

Sorted by: Highest score (default)

1207

Lambda comes from the Lambda Calculus and refers to anonymous functions in programming.

Why is this cool? It allows you to write quick throw away functions without naming them. It also provides a nice way to write closures. With that power you can do things like this.

Python

```
def adder(x):
    return lambda y: x + y
add5 = adder(5)
add5(1)
6
```

As you can see from the snippet of Python, the function adder takes in an argument x, and returns an anonymous function, or lambda, that takes another argument y. That anonymous function allows you to create functions from functions. This is a simple example, but it should convey the power lambdas and closures have.

Examples in other languages

Perl 5

```
sub adder {
    my ($x) = @_;
    return sub {
        my ($y) = @_;
        $x + $y
    }
}

my $add5 = adder(5);
print &$add5(1) == 6 ? "ok\n" : "not ok\n";
```

JavaScript

```
var adder = function (x) {
    return function (y) {
        return x + y;
    };
};
add5 = adder(5);
add5(1) == 6
```

JavaScript (ES6)

```
const adder = x => y => x + y;
add5 = adder(5);
add5(1) == 6
```

Scheme

```
(define adder
    (lambda (x)
        (lambda (y)
           (+ x y))))
(define add5
    (adder 5))
```

```
(add5 1)
6
```

C# 3.5 or higher

```
Func<int, Func<int, int>> adder =
    (int x) => (int y) => x + y; // `int` declarations
Func<int, int> add5 = adder(5);
var add6 = adder(6); // Using implicit typing
Debug.Assert(add5(1) == 6);
Debug.Assert(add6(-1) == 5);

// Closure example
int yEnclosed = 1;
Func<int, int> addWithClosure =
    (x) => x + yEnclosed;
Debug.Assert(addWithClosure(2) == 3);
```

Swift

```
func adder(x: Int) -> (Int) -> Int{
   return { y in x + y }
}
let add5 = adder(5)
add5(1)
6
```

PHP

```
$a = 1;
$b = 2;

$lambda = fn () => $a + $b;

echo $lambda();
```

Haskell

```
(\x y -> x + y)
```

Java see [this post]()

```
// The following is an example of Predicate :
// a functional interface that takes an argument
// and returns a boolean primitive type.

Predicate<Integer> pred = x -> x % 2 == 0; // Tests if
boolean result = pred.test(4); // true
```

Lua

```
adder = function(x)
    return function(y)
        return x + y
    end
end
add5 = adder(5)
add5(1) == 6        -- true
```

Kotlin

```
val pred = { x: Int -> x % 2 == 0 }
val result = pred(4) // true
```

Ruby

Ruby is slightly different in that you cannot call a lambda using the exact same syntax as calling a function, but it still has lambdas.

```
def adder(x)
  lambda { |y| x + y }
end
add5 = adder(5)
add5[1] == 6
```

Ruby being Ruby, there is a shorthand for lambdas, so you can define `adder` this way:

```
def adder(x)
  -> y { x + y }
end
```

R

```
adder <- function(x) {
  function(y) x + y
}
add5 <- adder(5)
add5(1)
#> [1] 6
```

Share Improve this answer

Follow

edited Apr 22, 2020 at 10:01

Jack
9,774 ● 18 ● 76 ● 114

answered Aug 19, 2008 at 16:27

mk.
26.2k ● 13 ● 39 ● 41

---

5 What's the difference, then, between a lambda function and a functor? – Maxpm Nov 8, 2013 at 16:28

2 @Maxpm a functor can be a standard object with instance fields and functions, whereas a lambda function generally only consists of one line of instructions. This can vary depending on the language of course. – zdimension Apr 20, 2018 at 13:05

3 I don't think it's necessarily accurate to say lambda functions are the same as anonymous functions. For some languages like JavaScript a lambda expression is a specific form of an anonymous function. The JavaScript example you gave is an anonymous function without lambda syntax whereas the JavaScript (ES6) example you gave is a lambda expression. – Kyle Delaney Oct 24, 2018 at 18:38

2 @KyleDelaney indeed, anonymous is not a necessary condition to be a lambda, indeed there are lambda function that are not anonymous, as you point there one even in the example – Carmine Tambascia May 14, 2019 at 21:00

@AliAnkarali or get used to using rubys lambda ;) – mirageglobe May 20, 2019 at 6:34

117

A lambda is a type of function, defined inline. Along with a lambda you also usually have some kind of variable type that can hold a reference to a function, lambda or otherwise.

For instance, here's a C# piece of code that doesn't use a lambda:

```
public Int32 Add(Int32 a, Int32 b)
{
    return a + b;
}

public Int32 Sub(Int32 a, Int32 b)
```

```
{
    return a - b;
}

public delegate Int32 Op(Int32 a, Int32 b);

public void Calculator(Int32 a, Int32 b, Op op)
{
    Console.WriteLine("Calculator: op(" + a + ", " + b
}

public void Test()
{
    Calculator(10, 23, Add);
    Calculator(10, 23, Sub);
}
```

This calls Calculator, passing along not just two numbers, but which method to call inside Calculator to obtain the results of the calculation.

In C# 2.0 we got anonymous methods, which shortens the above code to:

```
public delegate Int32 Op(Int32 a, Int32 b);

public void Calculator(Int32 a, Int32 b, Op op)
{
    Console.WriteLine("Calculator: op(" + a + ", " + b
}

public void Test()
{
    Calculator(10, 23, delegate(Int32 a, Int32 b)
    {
        return a + b;
    });
    Calculator(10, 23, delegate(Int32 a, Int32 b)
    {
        return a - b;
    });
}
```

And then in C# 3.0 we got lambdas which makes the code even shorter:

```
public delegate Int32 Op(Int32 a, Int32 b);

public void Calculator(Int32 a, Int32 b, Op op)
{
    Console.WriteLine("Calculator: op(" + a + ", " + b
}

public void Test()
{
    Calculator(10, 23, (a, b) => a + b);
    Calculator(10, 23, (a, b) => a - b);
}
```

Share Improve this answer

Follow

edited Nov 23, 2011 at 1:44

Kevin Ji
10.5k ● 4 ● 42 ● 65

answered Aug 19, 2008 at 16:27

Lasse V. Karlsen
391k ● 106 ● 646 ● 844

Instead of explicitly defining the delegate `Op`, one may simply use `Func<int, int>` – Mateen Ulhaq May 14, 2016 at 4:03

I'd suggest `Console.WriteLine("Calculator: op " + op.Method.Name + " (" + a + ", " + b + ") = " + op(a, b));` for the first example. – Marc.2377 Apr 1, 2017 at 0:53

1 As amazing as the accepted answer is at helping people in many different languages, I feel like this answer helps me understand the advantage of lambdas better, and how they are different from a standard function. I can clearly see here the advantage of a lambda over the previous methods for getting a similar result. – RTHarston Oct 24, 2019 at 18:45

Thank you for your example, having a more complex function helps to understand the advantages of lambdas a lot more than simple functions where it looks like you don't get much optimisation – Sarah May 4, 2020 at 22:55

1 this use of lambda and misdirection seems like a huge step backwards. the first code example is much easier to read and understand than either of the next two. Shortened code is not always better. – loneRanger Dec 5, 2022 at 17:15

84

+50

The name "lambda" is just a historical artifact. All we're talking about is an expression whose value is a function.

A simple example (using Scala for the next line) is:

```
args.foreach(arg => println(arg))
```

where the argument to the `foreach` method is an expression for an anonymous function. The above line is more or less the same as writing something like this (not quite real code, but you'll get the idea):

```
void printThat(Object that) {
  println(that)
}
```

```
...
args.foreach(printThat)
```

except that you don't need to bother with:

1. Declaring the function somewhere else (and having to look for it when you revisit the code later).
2. Naming something that you're only using once.

Once you're used to function values, having to do without them seems as silly as being required to name every expression, such as:

```
int tempVar = 2 * a + b
...
println(tempVar)
```

instead of just writing the expression where you need it:

```
println(2 * a + b)
```

The exact notation varies from language to language; Greek isn't always required! ;-)

Share Improve this answer

Follow

answered Aug 29, 2008 at 18:36

joel.neely

30.9k ● 9 ● 57 ● 64

It refers to lambda calculus, which is a formal system that just has lambda expressions, which represent a function that takes a function for its sole argument and returns a

64

function. All functions in the lambda calculus are of that type, i.e., `λ : λ → λ`.

Lisp used the lambda concept to name its anonymous function literals. This lambda represents a function that takes two arguments, x and y, and returns their product:

```
(lambda (x y) (* x y))
```

It can be applied in-line like this (evaluates to *50*):

```
((lambda (x y) (* x y)) 5 10)
```

Share Improve this answer Follow

edited Feb 1, 2018 at 20:50
TylerH
21.2k ● 76 ● 79 ● 110

answered Aug 19, 2008 at 16:23
Mark Cidade
99.8k ● 33 ● 229 ● 237

I think your use of `λ : λ -> λ` is confusing (and invalid actually). – einpoklum Dec 17, 2019 at 17:36

57

The lambda calculus is a consistent mathematical theory of substitution. In school mathematics one sees for example `x+y=5` paired with `x−y=1`. Along with ways to manipulate individual equations it's also possible to put the information from these two together, provided cross-equation substitutions are done logically. Lambda calculus codifies the correct way to do these substitutions.

Given that `y = x−1` is a valid rearrangement of the second equation, this: `λ y = x−1` means a function substituting the symbols `x−1` for the symbol `y`. Now imagine applying `λ y` to each term in the first equation. If a term is `y` then perform the substitution; otherwise do nothing. If you do this out on paper you'll see how applying that `λ y` will make the first equation solvable.

That's an answer without any computer science or programming.

The simplest programming example I can think of comes from http://en.wikipedia.org/wiki/Joy_(programming_language)#How_it_works:

> here is how the square function might be defined in an imperative programming language (C):
>
> ```
> int square(int x)
> {
>     return x * x;
> }
> ```
>
> The variable x is a formal parameter which is replaced by the actual value to be squared when the function is called. In a functional language (Scheme) the same function would be defined:

> ```
> (define square
>   (lambda (x)
>     (* x x)))
> ```
>
> This is different in many ways, but it still uses the formal parameter x in the same way.

Added: https://i.sstatic.net/VSgYr.jpg

$$[\lambda x . x+1](2)$$

It becomes...

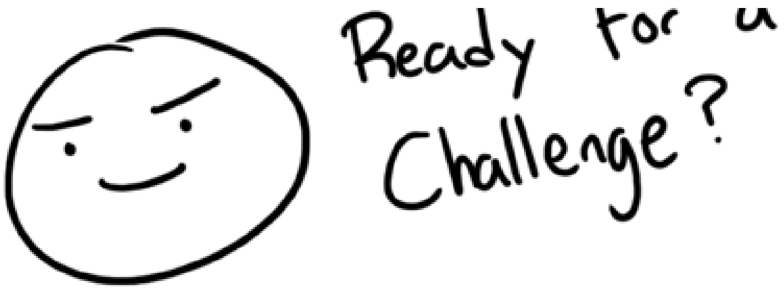
$$[\lambda x . \overset{2}{\cancel{x}}+1](\cancel{2})$$

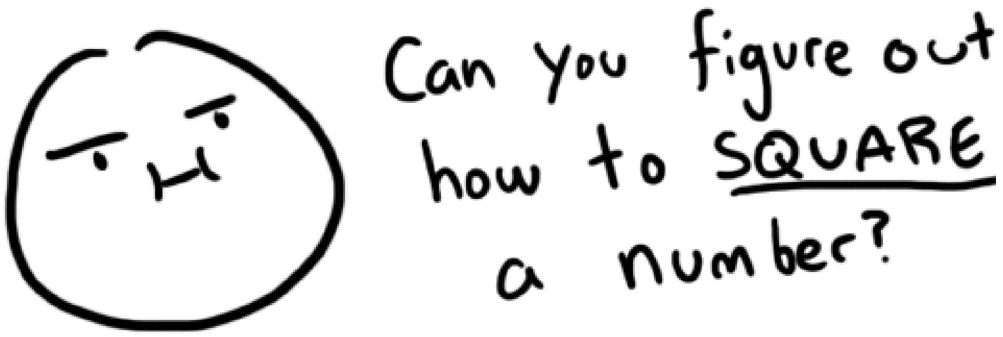
Which resolves to:

$$\cancel{\lambda x}.\ \boxed{2+1}$$

$$\downarrow$$

$$3$$

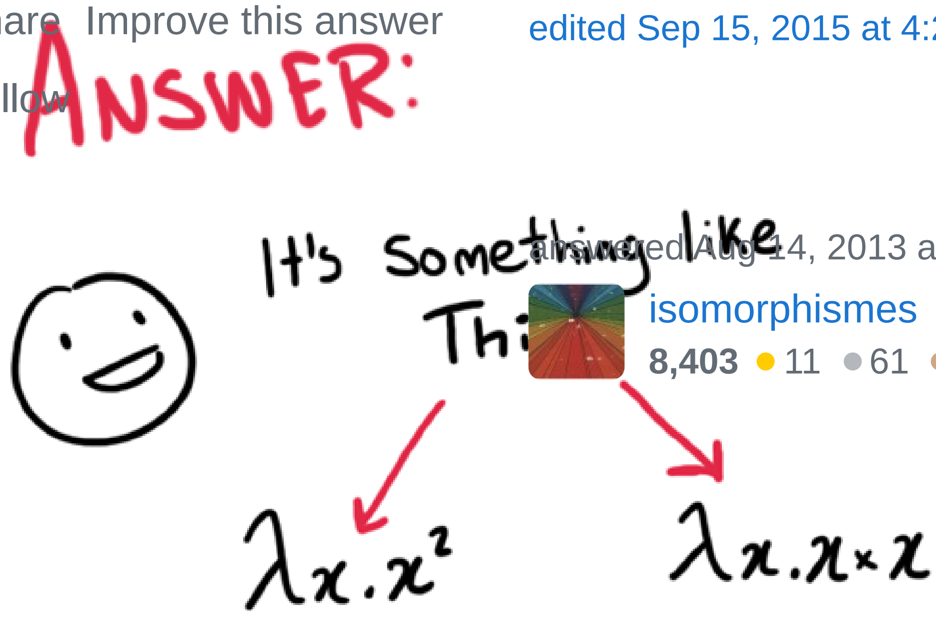

Ready for a Challenge?


Can you figure out how to SQUARE a number?

Share Improve this answer

Follow

edited Sep 15, 2015 at 4:24



answered Aug 14, 2013 at 3:00

isomorphismes

8,403 ● 11 ● 61 ● 72

1 Your animated story captivated me – Egel Oct 7, 2022 at 15:45

17

Slightly oversimplified: a lambda function is one that can be passed round to other functions and it's logic accessed.

In C# lambda syntax is often compiled to simple methods in the same way as anonymous delegates, but it can also be broken down and its logic read.

For instance (in C#3):

```
LinqToSqlContext.Where(
    row => row.FieldName > 15 );
```

LinqToSql can read that function (x > 15) and convert it to the actual SQL to execute using expression trees.

The statement above becomes:

```
select ... from [tablename]
where [FieldName] > 15      --this line was 'read'
from the lambda function
```

This is different from normal methods or anonymous delegates (which are just compiler magic really) because they cannot be *read*.

Not all methods in C# that use lambda syntax can be compiled to expression trees (i.e. actual lambda functions). For instance:

```
LinqToSqlContext.Where(
    row => SomeComplexCheck( row.FieldName ) );
```

Now the expression tree cannot be read - SomeComplexCheck cannot be broken down. The SQL statement will execute without the where, and every row in the data will be put through `SomeComplexCheck`.

Lambda functions should not be confused with anonymous methods. For instance:

```
LinqToSqlContext.Where(
    delegate ( DataRow row ) {
        return row.FieldName > 15;
    } );
```

This also has an 'inline' function, but this time it's just compiler magic - the C# compiler will split this out to a new instance method with an autogenerated name.

Anonymous methods can't be read, and so the logic can't be translated out as it can for lambda functions.

Share Improve this answer Follow

edited Aug 19, 2008 at 17:40

answered Aug 19, 2008 at 16:30

Keith
155k ● 82 ● 306 ● 446

11

The question is formally answered greatly, so I will not try to add more on this.

In very simple, **informal** words to someone that knows very little or nothing on math or programming, I would explain it as a small "machine" or "box" that takes some input, makes some work and produces some output, has no particular name, but we know where it is and by just this knowledge, we use it.

Practically speaking, for a person that knows what a function is, I would tell them that it is a function that has no name, usually put to a point in memory that can be used just by referencing to that memory (usually via the usage of a variable - if they have heard about the concept of the function pointers, I would use them as a similar concept) - this answer covers the pretty basics (no mention of closures etc) but one can get the point easily.

Share Improve this answer Follow

edited Jan 31, 2017 at 1:38

answered Jan 31, 2017 at 1:24

Nick Louloudakis
5,995 ● 4 ● 42 ● 56

9

I like the explanation of Lambdas in this article: [The Evolution Of LINQ And Its Impact On The Design Of C#](). It made a lot of sense to me as it shows a real world for Lambdas and builds it out as a practical example.

Their quick explanation: Lambdas are a way to treat code (functions) as data.

Share Improve this answer Follow

answered Aug 19, 2008 at 16:29

Jon Galloway
53.1k ● 25 ● 127 ● 194

9

> A `Lambda Function`, or a `Small Anonymous Function`, is a self-contained block of functionality that can be passed around and used in your code. Lambda has different names in different programming languages – `Lambda` in **Python** and **Kotlin**, `Closure` in **Swift**, or `Block` in **C** and **Objective-C**. Although lambda's meaning is quite similar for these languages it has slight distinctions sometimes.

Let's see how Closure (Lambda) works in Swift:

```
let coffee: [String] = ["Cappuccino", "Espresso", "Lat
```

1. Regular Function

```
func backward(_ n1: String, _ n2: String) -> Bool {
    return n1 > n2
}
var reverseOrder = coffee.sorted(by: backward)


// RESULT: ["Ristretto", "Latte", "Espresso", "Cappucc
```

2. Closure Expression

```
reverseOrder = coffee.sorted(by: { (n1: String, n2: St
    return n1 > n2
})
```

3. Inline Closure Expression

```
reverseOrder = coffee.sorted(by: { (n1: String, n2: St
    return n1 > n2
})
```

4. Inferring Type From Context

```
reverseOrder = coffee.sorted(by: { n1, n2 in return n1
```

5. Implicit Returns from Single-Expression Closures

```
reverseOrder = coffee.sorted(by: { n1, n2 in n1 > n2 }
```

6. Shorthand Argument Names

```
reverseOrder = coffee.sorted(by: { $0 > $1 } )

// $0 and $1 are closure’s first and second String arg
```

7. Operator Methods

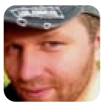
```
reverseOrder = coffee.sorted(by: >)

// RESULT: ["Ristretto", "Latte", "Espresso", "Cappucc
```

Share Improve this answer Follow

edited Aug 13, 2022 at 21:41

answered Jan 13, 2019 at 11:20



Andy Jazz
57.7k ● 18 ● 160 ● 252

8

Lambda explained for everyone:

Lambda is an anonymous function. This means lambda is a function object in Python that doesn't require a reference before. Let's consider this bit of code here:

```
def name_of_func():
    #command/instruction
    print('hello')

print(type(name_of_func))   #the name of the function
                            #the reference contains a
command/instruction
```

To proof my proposition I print out the type of name_of_func which returns us:

```
<class 'function'>
```

A function must have a interface, but a interface docent needs to contain something. What does this mean? Let's look a little bit closer to our function and we may notice that out of the name of the functions there are some more details we need to explain to understand what a function is.

A regular function will be defined with the syntax ***"def"***, then we type in the name and settle the interface with ***"()"*** and ending our definition by the syntax ***":"***. Now we enter the functions body with our instructions/commands.

So let's consider this bit of code here:

```python
def print_my_argument(x):
    print(x)


print_my_argument('Hello')
```

In this case we run our function, named "print_my_argument" and passing a parameter/argument through the interface. The Output will be:

```
Hello
```

So now that we know what a function is and how the architecture works for a function, we can take a look to an anonymous function. Let's consider this bit of code here:

```python
def name_of_func():
    print('Hello')



lambda: print('Hello')
```

these function objects are pretty much the same except of the fact that the upper, regular function have a name and the other function is an anonymous one. Let's take a closer look on our anonymous function, to understand how to use it.

So let's consider this bit of code here:

```python
def delete_last_char(arg1=None):
    print(arg1[:-1])
```

```
string = 'Hello World'
delete_last_char(string)

f = lambda arg1=None: print(arg1[:-1])
f(string)
```

So what we have done in the above code is to write once again, a regular function and an anonymous function. Our anonymous function we had assigned to a var, which is pretty much the same as to give this function a name. Anyway, the output will be:

```
Hello Worl
Hello Worl
```

To fully proof that lambda is a function object and doesn't just mimic a function we run this bit of code here:

```
string = 'Hello World'
f = lambda arg1=string: print(arg1[:-1])
f()
print(type(f))
```

and the Output will be:

```
Hello Worl
<class 'function'>
```

Last but not least you should know that every function in python needs to return something. If nothing is defined in the body of the function, None will be returned by default. look at this bit of code here:

```
def delete_last_char(arg1):
    print(arg1[:-1])

string = 'Hello World'
x = delete_last_char(string)

f = lambda arg1=string: print(arg1[:-1])
x2 = f()

print(x)
print(x2)
```

Output will be:

```
Hello Worl
Hello Worl
None
None
```

Share Improve this answer

Follow

edited Mar 20, 2021 at 14:34

Delrius Euphoria

15.1k ● 3 ● 18 ● 50

answered Jul 5, 2020 at 14:55

Thingamabobs

8,009 ● 6 ● 27 ● 63

7

@Brian I use lambdas all the time in C#, in LINQ and non-LINQ operators. Example:

```
string[] GetCustomerNames(IEnumerable<Customer> customers)
```

```
  { return customers.Select(c=>c.Name);
  }
```

Before C#, I used anonymous functions in JavaScript for callbacks to AJAX functions, before the term Ajax was even coined:

```
getXmlFromServer(function(result) {/*success*/},
function(error){/*fail*/});
```

The interesting thing with C#'s lambda syntax, though, is that on their own their type cannot be infered (i.e., you can't type var foo = (x,y) => x * y) but depending on which type they're assigned to, they'll be compiled as delegates or abstract syntax trees representing the expression (which is how LINQ object mappers do their "language-integrated" magic).

Lambdas in LISP can also be passed to a quotation operator and then traversed as a list of lists. Some powerful macros are made this way.

Share Improve this answer Follow

edited Aug 20, 2008 at 9:39

answered Aug 19, 2008 at 18:25

Mark Cidade
99.8k ● 33 ● 229 ● 237

7

An example of a lambda in Ruby is as follows:

```
hello = lambda do
    puts('Hello')
    puts('I am inside a proc')
end

hello.call
```

Will genereate the following output:

```
Hello
I am inside a proc
```

Share Improve this answer

Follow

edited Apr 23, 2020 at 19:10

user229044 ♦

239k ● 41 ● 344 ● 346

answered Aug 19, 2008 at 17:17

CodingWithoutComments

36k ● 21 ● 75 ● 87

6

Just because I cant see a C++11 example here, I'll go ahead and post this nice example from here. After searching, it is the clearest language specific example that I could find.

Hello, Lambdas, version 1

```
template<typename F>

void Eval( const F& f ) {
        f();
}
void foo() {
        Eval( []{ printf("Hello, Lambdas\n"); } );
}
```

Hello, Lambdas, version 2:

```
void bar() {
    auto f = []{ printf("Hello, Lambdas\n"); };
    f();
}
```

Share Improve this answer

Follow

edited Sep 30, 2018 at 8:23

Evg

26.2k ● 5 ● 43 ● 92

answered May 29, 2012 at 21:41

learnvst

16.1k ● 18 ● 77 ● 125

5

> For a person without a comp-sci background, what is a lambda in the world of Computer Science?

I will illustrate it intuitively step by step in simple and readable python codes.

In short, a lambda is just an anonymous and inline function.

Let's start from assignment to understand `lambdas` as a freshman with background of basic arithmetic.

The blueprint of assignment is 'the name = value', see:

```
In [1]: x = 1
   ...: y = 'value'
In [2]: x
Out[2]: 1
In [3]: y
Out[3]: 'value'
```

'x', 'y' are names and 1, 'value' are values. Try a function in mathematics

```
In [4]: m = n**2 + 2*n + 1
NameError: name 'n' is not defined
```

Error reports,
you cannot write a mathematic directly as code,'n' should be defined or be assigned to a value.

```
In [8]: n = 3.14
In [9]: m = n**2 + 2*n + 1
In [10]: m
Out[10]: 17.1396
```

It works now,what if you insist on combining the two seperarte lines to one. There comes `lambda`

```
In [13]: j = lambda i: i**2 + 2*i + 1
In [14]: j
Out[14]: <function __main__.<lambda>>
```

No errors reported.

This is a glance at `lambda`, it enables you to write a function in a single line as you do in mathematic into the computer directly.

We will see it later.

Let's continue on digging deeper on 'assignment'.

As illustrated above, the equals symbol `=` works for simple data(1 and 'value') type and simple expression(n**2 + 2*n + 1).

Try this:

```
In [15]: x = print('This is a x')
This is a x
In [16]: x
In [17]: x = input('Enter a x: ')
Enter a x: x
```

It works for simple statements,there's 11 types of them in python [7. Simple statements — Python 3.6.3 documentation]()

How about compound statement,

```
In [18]: m = n**2 + 2*n + 1 if n > 0
SyntaxError: invalid syntax
```

```
#or
In [19]: m = n**2 + 2*n + 1, if n > 0
SyntaxError: invalid syntax
```

There comes `def` enable it working

```
In [23]: def m(n):
    ...:     if n > 0:
    ...:         return n**2 + 2*n + 1
    ...:
In [24]: m(2)
Out[24]: 9
```

Tada, analyse it, 'm' is name, 'n**2 + 2*n + 1' is value. `:` is a variant of '='.
Find it, if just for understanding, everything starts from assignment and everything is assignment.

Now return to `lambda`, we have a function named 'm'

Try:

```
In [28]: m = m(3)
In [29]: m
Out[29]: 16
```

There are two names of 'm' here, function `m` already has a name, duplicated.

It's formatting like:

```
In [27]: m = def m(n):
    ...:         if n > 0:
```

```
   ...:               return n**2 + 2*n + 1
   SyntaxError: invalid syntax
```

It's not a smart strategy, so error reports

We have to delete one of them,set a function without a name.

```
m = lambda n:n**2 + 2*n + 1
```

It's called 'anonymous function'

In conclusion,

1. `lambda` in an inline function which enable you to write a function in one straight line as does in mathematics
2. `lambda` is anonymous

Hope, this helps.

Share Improve this answer Follow

edited Jan 16, 2018 at 17:58
marc_s
753k ● 183 ● 1.4k ● 1.5k

answered Nov 17, 2017 at 16:24
Wizard
21.9k ● 21 ● 93 ● 157

You can think of it as an anonymous function - here's some more info: Wikipedia - Anonymous Function

4

Share Improve this answer

Follow

answered Aug 19, 2008 at 16:23

mercutio

22.5k ● 10 ● 37 ● 37

4

I have trouble wrapping my head around lambda expressions because I work in Visual FoxPro, which has Macro substitution and the ExecScript{} and Evaluate() functions, which seem to serve much the same purpose.

```
? Calculator(10, 23, "a + b")
? Calculator(10, 23, "a - b");

FUNCTION Calculator(a, b, op)
RETURN Evaluate(op)
```

One definite benefit to using formal lambdas is (I assume) compile-time checking: Fox won't know if you typo the text string above until it tries to run it.

This is also useful for data-driven code: you can store entire routines in memo fields in the database and then just evaluate them at run-time. This lets you tweak part of the application without actually having access to the source. (But that's another topic altogether.)

Share Improve this answer

Follow

answered Aug 22, 2008 at 18:01

SarekOfVulcan

1,358 ● 5 ● 16 ● 36

1 Visual Fox Pro. That takes me back ages. – Lucio Mollinedo Aug 28, 2020 at 15:13

3

It is a function that has no name. For e.g. in c# you can use

```
numberCollection.GetMatchingItems<int>(number =>
number > 5);
```

to return the numbers that are greater than 5.

```
number => number > 5
```

is the lambda part here. It represents a function which takes a parameter (number) and returns a boolean value (number > 5). GetMatchingItems method uses this lambda on all the items in the collection and returns the matching items.

Share Improve this answer Follow

answered Aug 19, 2008 at 16:28

Serhat Ozgel
23.7k ● 29 ● 104 ● 141

3

In Javascript, for example, functions are treated as the same mixed type as everything else (`int`, `string`, `float`, `bool`). As such, you can create functions on the fly, assign them to things, and call them back later. It's useful but, not something you want to over use or you'll confuse everyone who has to maintain your code after you...

This is some code I was playing with to see how deep this rabbit hole goes:

```
var x = new Object;
x.thingy = new Array();
x.thingy[0] = function(){ return function(){ return fu
pressed'); }; }; }
x.thingy[1] = function(){ return function(){ return fu
pressed'); }; }; }
x.thingy[2] = function(){ return function(){ return fu
pressed'); }; }; }

for(var i=0 ;i<3; i++)
    x.thingy[i]()()();
```

Share Improve this answer

Follow

edited Jan 28, 2016 at 22:03

Michael come lately

9,273 ● 7 ● 66 ● 92

answered Aug 10, 2009 at 13:22

opmrcrab

In context of CS a lambda function is an abstract mathematical concept that tackles a problem of symbolic

3

evaluation of mathematical expressions. In that context a lambda function is the same as a lambda term.

But in programming languages it's something different. It's a piece of code that is declared "in place", and that can be passed around as a "first-class citizen". This concept appeared to be useful so that it came into almost all popular modern programming languages (see lambda functions everwhere post).

Share Improve this answer Follow

answered Aug 20, 2016 at 21:40

battlmonstr
6,270 ● 1 ● 26 ● 33

3

In computer programming, lambda is a piece of code (statement, expression or a group of them) which takes some arguments from an external source. It must not always be an anonymous function - we have many ways to implement them.

We have clear separation between expressions, statements and functions, which mathematicians do not have.

The word "function" in programming is also different - we have "function is a series of steps to do" (from Latin "perform"). In math it is something about correlation between variables.

Functional languages are trying to be as similar to math formulas as possible, and their words mean almost the same. But in other programming languages we have it different.

Share Improve this answer Follow

answered Jul 9, 2017 at 22:16

konmik

3,200 ● 20 ● 16

1

The question has been answered fully, I don't want to go into details. I want to share the usage when writing numerical computation in rust.

There is an example of a lambda(anonymous function)

```
let f = |x: f32| -> f32 { x * x - 2.0 };
let df = |x: f32| -> f32 { 2.0 * x };
```

When I was writing a module of Newton–Raphson method, it was used as first and second order derivative. (If you want to know what is Newton–Raphson method, please visit "https://en.wikipedia.org/wiki/Newton%27s_method".

The output as the following

```
println!("f={:.6}      df={:.6}", f(10.0),
df(10.0))

f=98.000000       df=20.000000
```

Share Improve this answer Follow

answered May 22, 2018 at 4:18

madeinQuant
1,801 ● 1 ● 21 ● 30

1

Imagine that you have a restaurant with a delivery option and you have an order that needs to be done in under 30 minutes. The point is clients usually don't care if you send their food by bike with a car or barefoot as long as you keep the meal warm and tied up. So lets convert this idiom to Javascript with anonymous and defined transportation functions.

Below we defined the way of our delivering aka we define a name to a function:

```
// ES5
var food = function withBike(kebap, coke) {
return (kebap + coke);
};
```

What if we would use arrow/lambda functions to accomplish this transfer:

```
// ES6
const food = (kebap, coke) => { return kebap +
coke };
```

You see there is no difference for client and no time wasting to think about how to send food. Just send it.

Btw, I don't recommend the kebap with coke this is why upper codes will give you errors. Have fun.

Share Improve this answer Follow

edited Oct 4, 2018 at 12:56
Martin Reiche
1,672 ● 1 ● 16 ● 27

answered Oct 4, 2018 at 12:00
akinov
65 ● 2 ● 12

0

I think you should use lambda when you want function to be used once, and code should reflect domain logic more than code logic, something like this

```
firstNameOf = lambda employee:
employee.employee_name.split()[0]
for employee in employees:
  print(firstNameOf(employee))
```

I have shared my thought in this blog https://xeon2k.wordpress.com/2023/12/23/lamda-functions-in-python/

Share Improve this answer Follow

answered Dec 23, 2023 at 17:37
Nitin
25 ● 3

-3

A lambda function can take any number of arguments, but they contain only a single expression. ... Lambda functions can be used to return function objects. Syntactically, lambda functions are restricted to only a single expression.

Share Improve this answer

Follow

answered Sep 1, 2020 at 22:08

oluwaferanmi Fakolujo
5 ● 2

"lambda functions are restricted to only a single expression" - this is not actually true. C# have Statement lambdas where you can put block of code as a body, same for many other languages. – Magiczne Sep 2, 2020 at 7:39