

FACULTAD: INGENIERIA

ESCUELA ACADÉMICO PROFESIONAL DE ING. SISTEMAS

**INFORME ACADÉMICO**

Programación funcional y expresiones lambda

Autor(es):

-ATENCIO ARIAS, Jonathan Jeffery

.PARIONA PEREZ, Diego

-SANCHEZ ALIAGA, Becker Luis

-ZACARIAS MENDEZ, Kevin

-ZURITA TADEO, Paul

Lima

-

Perú

2017

INDICE

Página

Introducción…………………………………………………………………………….3

1. PROGRAMACION FUNCIONAL

Definición………………………… ……………………………………………………4

Objetivos………………………………………………………………………………..5

Características………………………………………………………………………....5

1. Funciones de primera clase y de orden superior…………………………..6
2. Funciones puras……………………………………………………………….6
3. Recursividad……………………………………………………………………7
4. Evaluación estricta frente a la no estricta…………………………………...8
5. Sistemas de tipos……………………………………………………………...9
6. La programación funcional en lenguajes no funcionales………………...10

Ventajas…………………………………………………………………………….....11

1. EXPRESIONES LAMBDA

Definición……………………………………………………………………………...11

Usos……………………………………………………………………………………12

Clasificación…………………………………………………………………….…….12

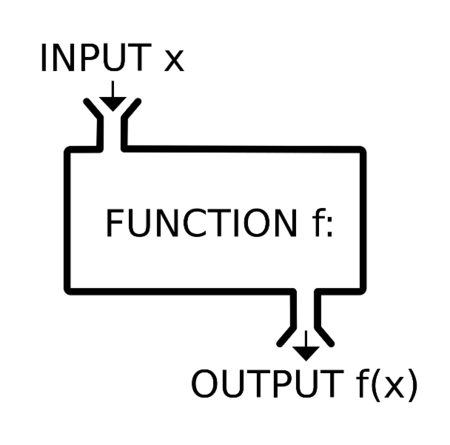
Ejemplos………………………………………………………………………………13

Conclusiones………………………………………………………………………….13

INTRODUCCION

La programación funcional es un [paradigma de programación](https://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) [declarativa](https://es.wikipedia.org/wiki/Programaci%C3%B3n_declarativa) basado en el uso de [funciones matemáticas](https://es.wikipedia.org/wiki/Funci%C3%B3n_matem%C3%A1tica), en contraste con la [programación imperativa](https://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa), que enfatiza los cambios de estado mediante la mutación de variables. La programación funcional tiene sus raíces en el [cálculo lambda](https://es.wikipedia.org/wiki/C%C3%A1lculo_lambda), un sistema formal desarrollado en los años 1930 para investigar la definición de función, la aplicación de las funciones y la recursión. Muchos lenguajes de programación funcionales pueden ser vistos como elaboraciones del cálculo lambda.

En el ámbito de la [programación](https://es.wikipedia.org/wiki/Programaci%C3%B3n), una función anónima (función literal, expresión lambda) es una [subrutina](https://es.wikipedia.org/wiki/Subrutina) definida que no está [enlazada](https://es.wikipedia.org/wiki/Enlace_simb%C3%B3lico) a un [identificador](https://es.wikipedia.org/wiki/Identificador).

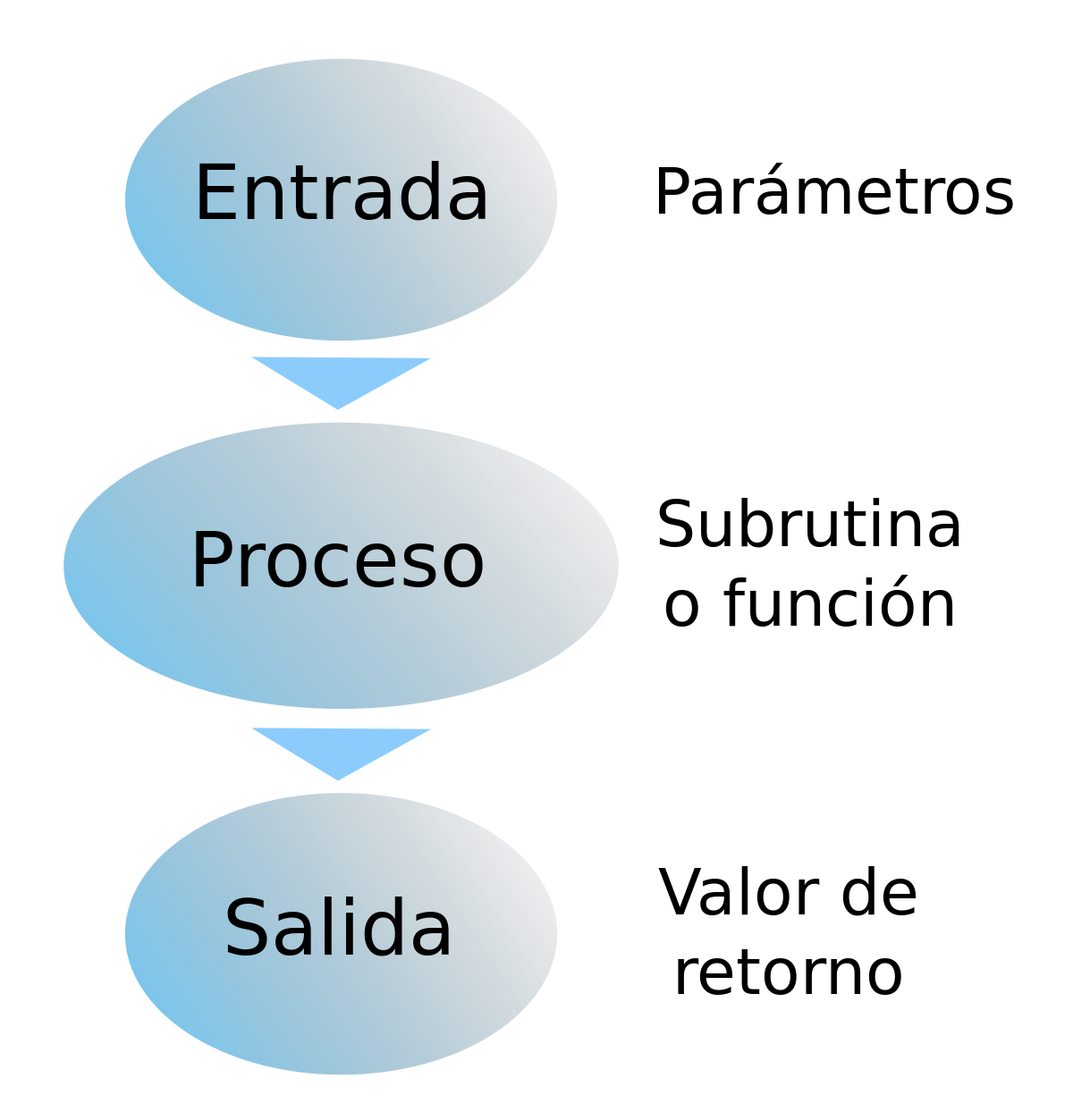


1. PROGRAMACION FUNCIONAL

DEFINICION

La programación funcional es un paradigma de programación declarativa basado en el uso de funciones matemáticas, en contraste con la programación imperativa, que enfatiza los cambios de estado mediante la mutación de variables. La programación funcional tiene sus raíces en el cálculo lambda, un sistema formal desarrollado en los años 1930 para investigar la definición de función, la aplicación de las funciones y la recursión. Muchos lenguajes de programación funcionales pueden ser vistos como elaboraciones del cálculo lambda.

En la práctica, la diferencia entre una función matemática y la noción de una "función" utilizada en la programación imperativa, es que las funciones imperativas pueden tener efectos secundarios, como cambiar el valor de cálculos realizados previamente. Por esta razón carecen de transparencia referencial, es decir, la misma expresión sintáctica puede resultar en valores diferentes en varios momentos de la ejecución del programa. Con código funcional, en contraste, el valor generado por una función depende exclusivamente de los argumentos alimentados a la función. Al eliminar los efectos secundarios se puede entender y predecir el comportamiento de un programa mucho más fácilmente. Ésta es una de las principales motivaciones para utilizar la programación funcional.



OBJETIVOS

El objetivo es conseguir lenguajes expresivos y *matemáticamente elegantes*, en los que no sea necesario bajar al nivel de la máquina para describir el proceso llevado a cabo por el programa, y evitar el concepto de *estado* del cómputo. La secuencia de computaciones llevadas a cabo por el programa se rige única y exclusivamente por la *reescritura* de definiciones más amplias a otras cada vez más concretas y definidas, usando lo que se denominan "definiciones dirigidas".

CARACTERISTICAS

Los programas escritos en un lenguaje funcional están constituidos únicamente por definiciones de funciones, entendiendo éstas no como subprogramas clásicos de un lenguaje imperativo, sino como funciones puramente matemáticas, en las que se verifican ciertas propiedades como la *transparencia referencial* (el significado de una expresión depende únicamente del significado de sus subexpresiones), y por tanto, la carencia total de *efectos colaterales*.

Otras características propias de estos lenguajes son la no existencia de asignaciones de variables y la falta de construcciones estructuradas como la secuencia o la iteración (lo que obliga en la práctica a que todas las repeticiones de instrucciones se lleven a cabo por medio de funciones recursivas).

Existen dos grandes categorías de lenguajes funcionales: los funcionales *puros* y los *híbridos*. La diferencia entre ambos estriba en que los lenguajes funcionales híbridos son menos dogmáticos que los puros, al admitir conceptos tomados de los lenguajes imperativos, como las secuencias de instrucciones o la asignación de variables. En contraste, los lenguajes funcionales puros tienen una mayor potencia expresiva, conservando a la vez su transparencia referencial, algo que no se cumple siempre con un lenguaje funcional híbrido.

1. FUNCIONES DE PRIMERA CLASE Y DE ORDEN SUPERIOR:

Funciones de orden superior son funciones que pueden tomar otras funciones como argumentos o devolverlos como resultados. En cálculo, un ejemplo de una función de orden superior es el operador diferencial d / dx, que devuelve la derivada de una función f.

Las funciones de orden superior están estrechamente relacionadas con las funciones de primera clase en las cuales las funciones de orden superior y las funciones de primera clase pueden recibir como argumentos y resultados otras funciones. La distinción entre los dos es sutil: "de orden superior", describe un concepto matemático de funciones que operan sobre otras funciones, mientras que la "primera clase" es un término informático que describe las entidades del lenguaje de programación que no tienen ninguna restricción de su utilización (por lo tanto funciones de primera clase pueden aparecer en cualquier parte del programa que otras entidades de primer nivel como los números pueden, incluidos como argumentos a otras funciones y como sus valores de retorno).

Las funciones de orden superior permiten la aplicación parcial, una técnica en la que se aplica una función a sus argumentos uno a la vez, con cada aplicación devolver una nueva función que acepta el siguiente argumento. Esto le permite a uno expresar, por ejemplo, la función sucesor como el operador de suma aplicada parcialmente al número natural uno.

1. FUNCIONES PURAS:

Las funciones puramente funcionales (o expresiones) no tienen efectos secundarios (memoria o E/S). Esto significa que las funciones puras tienen varias propiedades útiles, muchas de las cuales pueden ser utilizadas para optimizar el código:

* Si no se utiliza el resultado de una expresión pura, se puede eliminar sin afectar a otras expresiones.
* Si una función pura se llama con parámetros que no causan efectos secundarios, el resultado es constante con respecto a la lista de parámetros (a veces llamada transparencia referencial), es decir, si la función pura se llama de nuevo con los mismos parámetros, el mismo resultado será devuelto (esto puede habilitar las optimizaciones de almacenamiento en caché).
* Si no hay una dependencia de datos entre dos expresiones puras, entonces su orden puede ser invertido, o pueden llevarse a cabo en paralelo y que no pueda interferir con los otros.
* Si el lenguaje no permite efectos secundarios, entonces cualquier estrategia de evaluación se puede utilizar, lo que da la libertad al compilador para reordenar o combinar la evaluación de expresiones en un programa (por ejemplo, usando la poda).

La mayoría de los compiladores de lenguajes imperativos detectan funciones puras automáticamente y realizan la eliminación de subexpresiones comunes. Sin embargo no siempre es posible detectarlo en bibliotecas pre-compiladas, porque por norma general no dan esta información. Esto provoca que no se puedan realizar optimizaciones que podrían aplicar a dichas funciones externas. Algunos compiladores, como gcc, añaden palabras claves adicionales para que el programador marque explícitamente como puras aquellas funciones externas que procedan, de modo que se le apliquen las optimizaciones pertinentes. Fortran 95 también permite declarar funciones "puras".

1. RECURSIVIDAD:

Iterar en los lenguajes funcionales es normalmente llevado a cabo mediante recursividad. Las funciones recursivas se invocan a sí mismas, permitiendo que una operación se realice una y otra vez hasta alcanzar el caso base. Aunque algunas recursividades requieren el mantenimiento de una pila, la recursividad mediante una cola puede ser reconocida y optimizada mediante un compilador dentro del mismo código utilizado, para implementar las iteraciones en un lenguaje imperativo. El estándar del esquema del lenguaje requiere implementaciones para conocer y optimizar la recursividad mediante una cola. La optimización de la recursividad mediante una cola puede ser implementada transformando el programa a un estilo de pase de continuidad durante la compilación, entre otros enfoques.

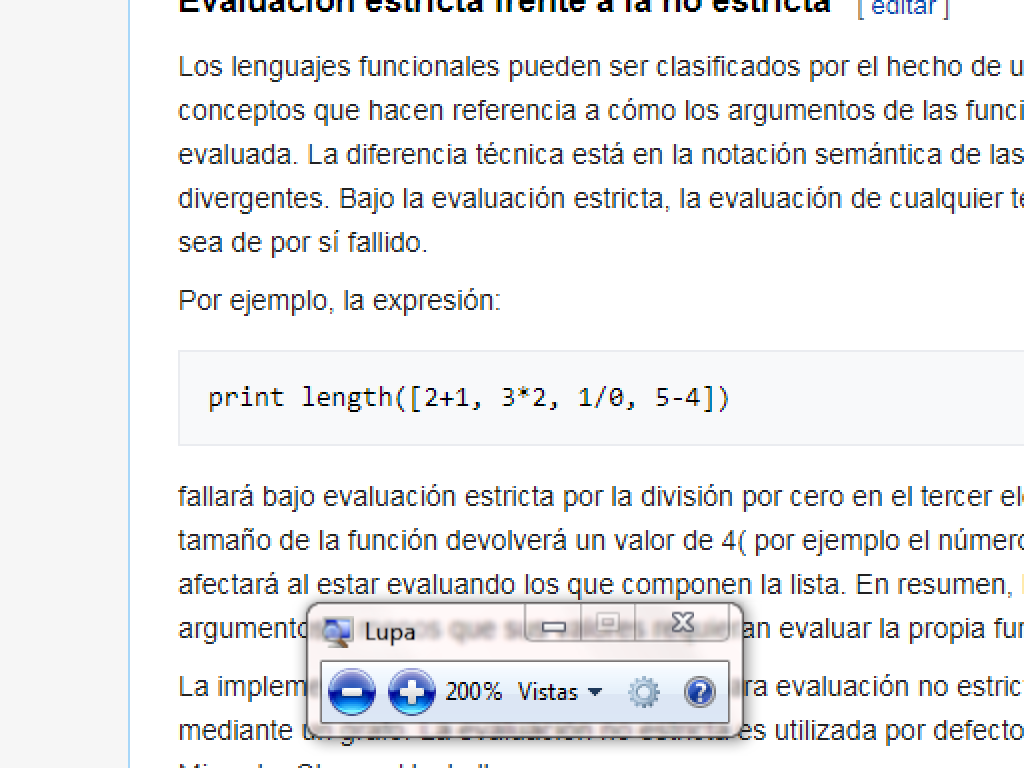
Los patrones comunes de recursividad puede ser factorizados usando funciones comunes más grandes, con “catamorfismos” y “anamorfismos” (pliegues y despliegues), siendo estos los ejemplos más evidentes. Tal y como las mayores funciones más comunes tienen un rol análogo para construir estructuras de control se tienen los iteradores en los lenguajes imperativos.

La mayoría de los lenguajes de programación funcional de propósito general permiten la recursividad sin restricciones y superan el test de Turing, lo que hace que el programa que se interrumpe no pueda tomar un decisión, lo que puede causar una falta de solidez en el razonamiento ecuacional y generalmente requiere introducir inconsistencia dentro de la lógica expresada por los tipos del sistema del lenguaje. Algunos lenguajes de propósito especial como Coq permiten tan sólo recursividad bien fundamentada y tienen una normalización fuerte (cálculos no finalizados pueden ser expresados tan sólo con flujos de valores infinitos llamados codata) En consecuencia, estos lenguajes fallan el test de Turing y declarar funciones ciertas en ellos es imposible, pero pueden declarar una amplia clase de cálculos interesantes mientras evitan los problemas producidos por la recursividad sin restricciones. La programación funcional limitada a la recursividad bien construida con unas cuantas restricciones más se llama programación funcional total.

1. EVALUACION ESTRICTA FRENTE A LA NO ESTRICTA:

Los lenguajes funcionales pueden ser clasificados por el hecho de usar evaluación estricta (eager) o no estricta (lazy), conceptos que hacen referencia a cómo los argumentos de las funciones son procesados cuando una expresión está siendo evaluada. La diferencia técnica está en la notación semántica de las expresiones que contienen cálculos fallidos o divergentes. Bajo la evaluación estricta, la evaluación de cualquier término que contenga un sub-término fallido hará que este sea de por sí fallido.

Por ejemplo, la expresión:



Fallará bajo evaluación estricta por la división por cero en el tercer elemento de la lista. Utilizando evaluación no estricta, el tamaño de la función devolverá un valor de 4( por ejemplo el número de elementos de la lista) ya que evaluar esto no afectará al estar evaluando los que componen la lista. En resumen, la evaluación estricta evalúa por completo los argumentos a menos que sus valores requieran evaluar la propia función que se llama a sí misma.

La implementación de la estrategia común para evaluación no estricta en los lenguajes funcionales es la de reducción mediante un grafo. La evaluación no estricta es utilizada por defecto en multitud de lenguajes funcionales puros, incluidos Miranda, Clean y Haskell.

Hughes (1984) defendía la evaluación no estricta como un mecanismo para mejorar la modularidad de los programas a través de la separación de tareas, a partir de la implementación de productores y consumidores de flujos de datos de forma fácil e independiente. Launchbury (1993) describe algunas dificultades que tenía la evaluación no estricta, particularmente al analizar los requisitos de almacenamiento de los programas, y propone una semántica operacional para ayudar durante el análisis. Harper (2009) propone incluir ambas técnicas (evaluación estricta y no estricta) en el mismo lenguaje, utilizando los tipos del sistema del lenguaje para distinguirlas.

1. SISTEMAS DE TIPOS:

Especialmente desde el desarrollo de inferencia de tipos Hindley - Milner en la década de 1970, los lenguajes de programación funcionales han tendido a utilizar el cálculo lambda con tipos, en comparación con el cálculo lambda sin tipos utilizado en Lisp y sus variantes (tales como el lenguaje scheme).

El uso de tipos de datos algebraicos y la coincidencia de patrones hace que la manipulación de estructuras de datos complejas convenientes y expresivos, la presencia de comprobaciones estrictas de tipos en tiempo de compilación hace que los programas sean más fiables, mientras que la inferencia de tipos libera al programador de la necesidad de declarar manualmente los tipos para el compilador.

Algunos lenguajes funcionales orientados a la investigación, tales como Coq, Agda, Cayenne y Epigram se basan en la teoría de tipo intuicionista, que permite a los tipos a depender de los términos. Estos tipos se denominan tipos dependientes. Se ha demostrado que estos sistemas de tipos sofisticados son tan expresivos que sus respectivos problemas de inferencia de tipos dejan de ser decidibles. Los tipos dependientes pueden expresar proposiciones arbitrarias en la lógica de predicados intuicionista. Este resultado se conoce como isomorfismo de Curry-Howard, y convierte a la programación funcional con una teoría de tipos intuicionista o equivalente en una forma de escribir pruebas matemáticas formales, de las que un compilador puede generar código certificado. Si bien estos lenguajes son principalmente de interés en la investigación académica (incluyendo las matemáticas formalizadas), han comenzado a ser utilizados en la ingeniería también. Compcert es un compilador para un subconjunto del lenguaje de programación C que está escrito en Coq y el cual se verificó formalmente. Una forma limitada de tipos dependientes llamados tipos de datos algebraicos generalizados (GADTs) puede ser implementado de una manera que ofrece algunos de los beneficios de la programación dependiente, evitando la mayor parte de su inconveniencia. GADTs están disponibles en el Glasgow Haskell Compiler, en OCaml (desde la versión 4.00) y en Scala y se han propuesto como adiciones a otros lenguajes, incluyendo Java y C#.

1. LA PROGRAMACION FUNCIONAL EN LENGUAJES NO FUNCIONALES:

Es posible utilizar un estilo de programación funcional en lenguajes que tradicionalmente no se consideran lenguajes funcionales. Por ejemplo, tanto D y Fortran95 se apoyan explícitamente en funciones puras. Funciones de primera clase, se han añadido lentamente a los lenguajes principales. Por ejemplo, a principios de 1994, el apoyo a lambda, filtro, mapa, y reducir esta en Python. Luego, durante el desarrollo de Python 3000, Guido van Rossum pidió la eliminación de estas características. Sin embargo, más tarde cambió de opinión, y sólo la reducción fue eliminada, a pesar de que sigue siendo accesible a través de los módulos de biblioteca functools estándar. Funciones de primera clase también fueron introducidas en PHP 5.3, Visual Basic9, C#3.0 y C++11.

En Java, las clases anónimas a veces pueden ser utilizadas para simular clausuras. Sin embargo, las clases anónimas no son siempre los reemplazos completos de las clausuras, ya que tienen capacidades más limitadas. Por ejemplo, Java 8, incluye expresiones lambda para reemplazar determinadas clases anónimas. Sin embargo, la presencia de excepciones con comprobaciones en este lenguaje puede desaconsejar el uso de programación funcional, ya que puede ser necesario para capturar las excepciones que se deben controlar para después volverlas a lanzar ellos (problema este que sin embargo no se produce en otros lenguajes sobre JVM que no tienen excepciones comprobadas, como es Scala).

Muchos patrones de diseño orientado a objetos se pueden expresar en términos de programación funcional por ejemplo: el patrón de estrategia simplemente dicta el uso de una función de orden superior, y el patrón de visitantes corresponde aproximadamente a un catamorfismo, o doble también conocido como reducir, comprimir, o inyectar, se refiere a una familia de funciones de orden superior que analiza una estructura de datos recursiva y se recombinan con el uso de una operación de combinación.

Del mismo modo, la idea de los datos inmutables de la programación funcional se incluye a menudo en lenguajes de programación imperativa, por ejemplo, la tupla de Python, que es una matriz inmutable.

VENTAJAS

Entre las ventajas que suelen citarse de usar un paradigma funcional en la programación de computadoras, están las siguientes:[1](https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional#cite_note-1)​

* Ausencia de efectos colaterales
* Proceso de depuración menos problemático
* Pruebas de unidades más confiables
* Mayor facilidad para la ejecución concurrente

1. EXPRESIONES LAMBDA

DEFINICION:

Por medio de expresiones lambda podemos referenciar métodos anónimos o métodos sin nombre, lo que nos permite escribir código más claro y conciso que cuando usamos clases anónimas. Una expresión lambda se compone de:

* Listado de parámetros separados por comas y encerrados en paréntesis, por ejemplo: (a, b).
* El símbolo de flecha hacia la derecha: ->
* Un cuerpo que puede ser un bloque de código encerrado entre llaves o una sola expresión.

USOS:

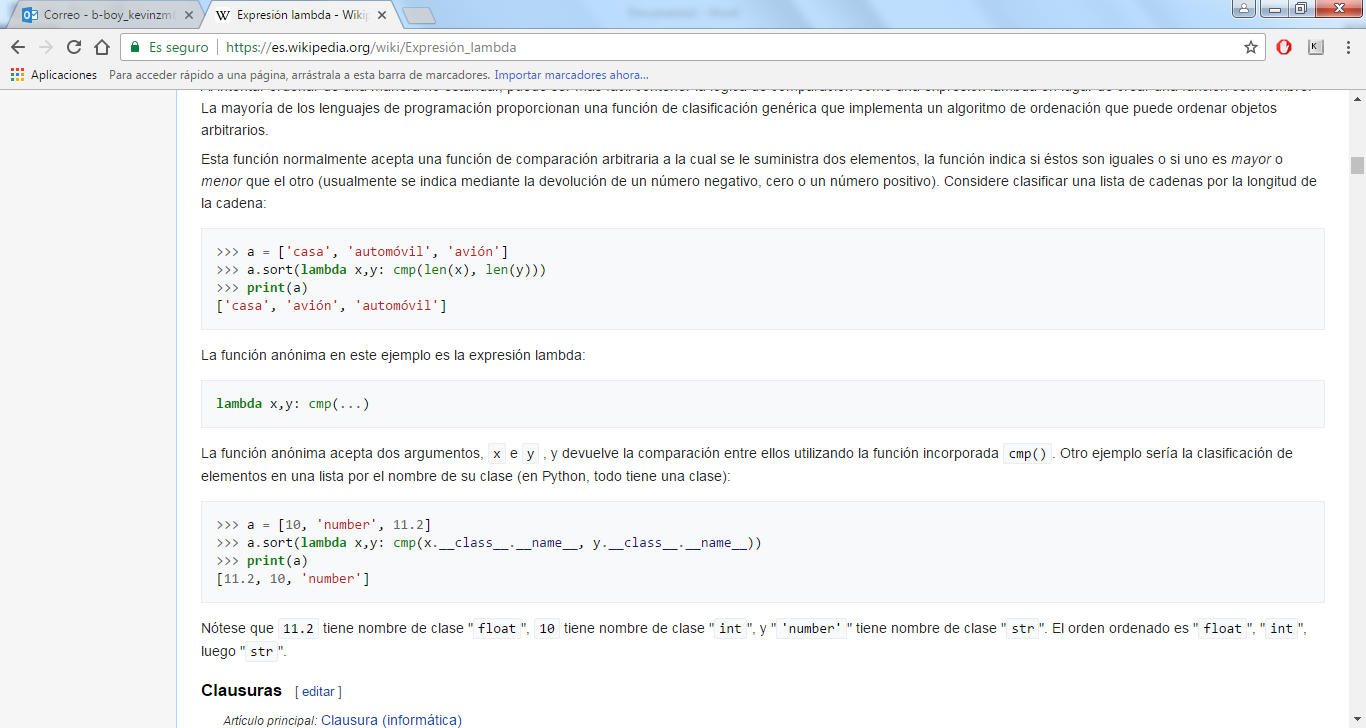
Las expresiones lambda pueden ser utilizadas para contener funcionalidades que no necesitan ser nombradas y normalmente se utilizan en un tiempo corto. Algunos ejemplos notables incluyen las [clausuras](https://es.wikipedia.org/wiki/Clausura_(inform%C3%A1tica)) y la [certificación](https://es.wikipedia.org/wiki/Currificaci%C3%B3n).

El uso de expresiones lambda es un tema de estilo. Usar expresiones lambda como la única forma de resolver un problema es erróneo; cada expresión lambda puede en su lugar ser reemplazada por una función nombrada y llamada por su nombre. Algunos programadores utilizan las expresiones lambda para encapsular código específico y no reutilizable para así evitar código basura con muchas pequeñas funciones de pocas líneas.

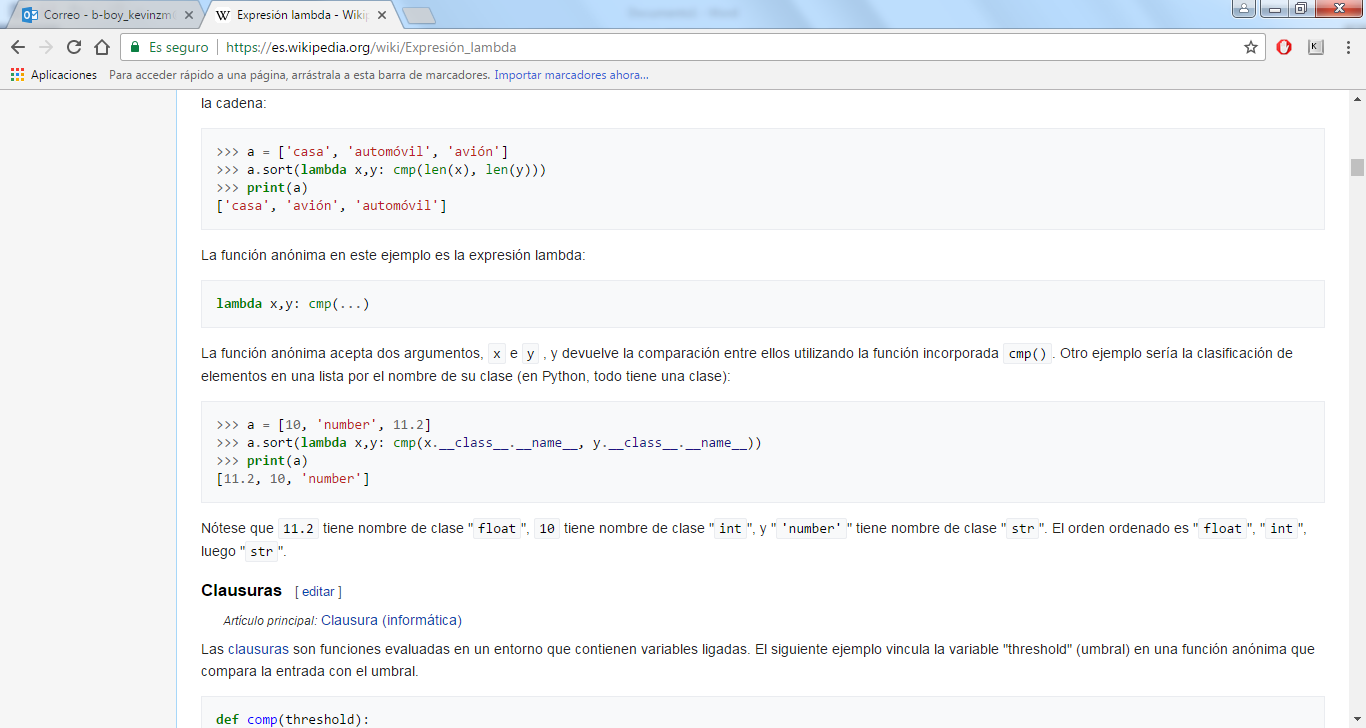
CLASIFICACION:

Al intentar ordenar de una manera no estándar, puede ser más fácil contener la lógica de comparación como una expresión lambda en lugar de crear una función con nombre. La mayoría de los lenguajes de programación proporcionan una función de clasificación genérica que implementa un algoritmo de ordenación que puede ordenar objetos arbitrarios.

Esta función normalmente acepta una función de comparación arbitraria a la cual se le suministra dos elementos, la función indica si éstos son iguales o si uno es *mayor* o *menor* que el otro (usualmente se indica mediante la devolución de un número negativo, cero o un número positivo). Considere clasificar una lista de cadenas por la longitud de la cadena:



La función anónima en este ejemplo es la expresión lambda:



La función anónima acepta dos argumentos, x e, y devuelve la comparación entre ellos utilizando la función incorporada cmp (). Otro ejemplo sería la clasificación de elementos en una lista por el nombre de su clase (en Python, todo tiene una clase):



EJEMPLOS:

Los argumentos de la expresión son los valores enteros A y B que sería utilizado por la expresión para devolver la salida de la operación de multiplicación.

(Persona p) -> {p.getAge return () ;}

se puede ver en el código de hebras donde también usábamos una clase anónima para usarlas, pero ahora puede ser sustituido por el de una expresión lambda, echemos un vistazo:

new thread (  
() -> System.out.println (“Esta es una ejecución distinta”)

.) Start ();

CONCLUSION

Puede ser más fácil contener la lógica de comparación como una expresión lambda en lugar de crear una función con nombre. La mayoría de los lenguajes de programación proporcionan una función de clasificación genérica que implementa un algoritmo de ordenación que puede ordenar objetos arbitrarios.