**Tell me the 3 worse defects of your preferred language**

**Pros Cons**

*1. Asynchronous event driven IO helps concurrent request handling.*

*1. Node.js doesn’t provide scalability. One CPU is not going to be enough; the platform provides no ability to scale out to take advantage of the multiple cores commonly present in today’s server-class hardware.*

*2. Uses JavaScript, which is easy to learn.*

*2. Dealing with relational database is a pain if you are using Node.*

*3. Share the same piece of code with both server and client side.*

*3. Every time using a callback end up with tons of nested callbacks.*

*4. npm, the Node packaged modules has already become huge, and still growing*

*4. Without diving in depth of JavaScript if someone starts Node, he may face conceptual problem.*

*5. Active and vibrant community, with lots of code shared via github, etc.*

*5. Node.js is not suited for CPU-intensive tasks. It is suited for I/O stuff only (like web servers).*

**Why is there a rising interest on Functional Programming?**

*FP pros and cons*

*Disadvantages*

*There is no efficient vocabulary for functional languages. Purely functional vocabularies work slower than the hash tables, and for some applications, this can be critical. Secondly, there is no purely functional weak hash map. Nevertheless, for most developers, this defect may remain unnoticed.*

*Functional programming is not suitable for algorithms in graphs (due to slow work) and in general for those solutions that for decades were based on imperative programming.*

*Check the lists of FP disadvantages on Quora and in the article by Alexander Alvin too.*

*Okay, the last point we mentioned is not very related to cons because we cannot blame functional programming for what it was not meant to be.*

*Advantages*

*Functional programming is famous for its high-level abstractions that hide a large number of details of such routine operations like iterating. This makes the code shorter and, as a consequence, guarantees a smaller number of errors that can be tolerated.*

*In the functional programming, there is a smaller number of language primitives. Well-known classes are not used in FP. Instead of creating a unique description of an object with operations in the form of methods, in functional programming, there are several basic language primitives that are well optimized inside.*

*Due to the language and structures flexibility, as a functional programming developer, you can bring the language closer to the problem. And not vice versa. In addition, FP offers some new and interesting tools for solving complex tasks that OOP developers often neglect.*

*Working with functional languages provides accurate and fast code writing, facilitates testing and debugging. You’re working with high-level programs, and the functions signatures are more informative.*

*You can find other functional programming benefits here.*

**What is a closure, and what is useful for? What's in common between closures and classes?**

*Functionally, closures and objects are equivalent. A closure can emulate an object and vice versa. So which one you use is a matter of syntactic convenience, or which one your programming language can best handle.*

*In C++ closures are not syntactically available, so you are forced to go with "functors", which are objects that override operator() and may be called in a way that looks like a function call.*

*In Java you don't even have functors, so you get things like the Visitor pattern, which would just be a higher order function in a language that supports closures.*

*In standard Scheme you don't have objects, so sometimes you end up implementing them by writing a closure with a dispatch function, executing different sub-closures depending on the incoming parameters.*

*In a language like Python, the syntax of which has both functors and closures, it's basically a matter of taste and which you feel is the better way to express what you are doing.*

*Personally, I would say that in any language that has syntax for both, closures are a much more clear and clean way to express objects with a single method. And vice versa, if your closure starts handling dispatch to sub-closures based on the incoming parameters, you should probably be using an object instead.*

**What are generics useful for?**

*Allows you to write code/use library methods which are type-safe, i.e. a List<string> is guaranteed to be a list of strings.*

*As a result of generics being used the compiler can perform compile-time checks on code for type safety, i.e. are you trying to put an int into that list of strings? Using an ArrayList would cause that to be a less transparent runtime error.*

*Faster than using objects as it either avoids boxing/unboxing (where .net has to convert value types to reference types or vice-versa) or casting from objects to the required reference type.*

*Allows you to write code which is applicable to many types with the same underlying behaviour, i.e. a Dictionary<string, int> uses the same underlying code as a Dictionary<DateTime, double>; using generics, the framework team only had to write one piece of code to achieve both results with the aforementioned advantages too.*

**What are high-order functions? What are they useful for? Write one, in your preferred language.**

*Usage of multiple returns, closures, obfuscation, bad formatting, nested functions, unused variables are all present in your example. From what I've tested it also does not consistently return the same values back ( floats ).*

*I rewrote it to demonstrate the intent behind the logic and it took a bit to unravel unused $rand\_seed\_fnc in the get\_algorithm call and the terrible function to variable assignment.*

*<?php*

*// Data to run functions on.*

*$data = array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);*

*function get\_algorithm( $iValue )*

*{*

*// Check if value is odd or even and return a different value.*

*// Replaces odd\_even function.*

*// Uses modulus %*

*// $value = 1, returns FALSE;*

*// $value = 2, returns TRUE;*

*// $value = 3, returns FALSE;*

*// $value = 4, returns TRUE;*

*if( $iValue % 2 === 0 )*

*{*

*// Square the value if it's even.*

*$iReturn = $iValue \* $iValue;*

*}*

*else*

*{*

*// Square the value and divide by a random number then add 10.*

*$iReturn = ( $iValue \* $iValue / rand() ) + 10;*

*}*

*return $iReturn;*

*}*

*$results = array\_map( 'get\_algorithm', $data );*

*var\_dump($results);*

*?>*

**Write a loop, then transform it into a recursive function, using only immutable structures (i.e. avoid using variables). Discuss.**

*A recursive function is a function that calls itself. Care must be taken in PHP, however, as your code must not carry out a certain number of recursive function calls. i.e. There must be a mechanism (IF statement, etc) that stops the recursion after the desired result has been found. If you allow your function to carry out an unlimited amount of calls (and you have XDebug installed), you will receive the error: “Fatal error: Maximum function nesting level of ‘100’ reached, aborting!”*

*function recursive($num){*

*//Print out the number.*

*echo $num, '<br>';*

*//If the number is less or equal to 50.*

*if($num < 50){*

*//Call the function again. Increment number by one.*

*return recursive($num + 1);*

*}*

*}*

*//Set our start number to 1.*

*$startNum = 1;*

*//Call our recursive function.*

*recursive($startNum);*

**What does it mean when a language treats functions as first-class citizens?**

*To quote Wikipedia:*

*In computer science, a programming language is said to support first-class functions (or function literal) if it treats functions as first-class objects. Specifically, this means that the language supports constructing new functions during the execution of a program, storing them in data structures, passing them as arguments to other functions, and returning them as the values of other functions.*

*This page also illustrates it beautifully:*

*Really, just like any other variable*

*A function is an instance of the Object type*

*A function can have properties and has a link back to its constructor method*

*You can store the function in a variable*

*You can pass the function as a parameter to another function*

*You can return the function from a function*

*also read TrayMan's comment, interesting...*

**Show me an example where an Anonymous Function can be useful?**

*PHP*

*$greet = function($name)*

*{*

*printf("Hello %s\r\n", $name);*

*};*

*$greet('World');*

*$greet('PHP');*

*Javascript*

*Anonymous functions are created at runtime. The function operator can be used anywhere that it's valid to use an expression. For example you can use the function operator when a variable is being assigned, when a parameter is being passed to a function or in a return statement*

*(function () {*

*var x = "Hello!!"; // I will invoke myself*

*})();*

**What are namespaces useful for? Invent an alternative.**

*What are namespaces? In the broadest definition namespaces are a way of encapsulating items. This can be seen as an abstract concept in many places. For example, in any operating system directories serve to group related files, and act as a namespace for the files within them. As a concrete example, the file foo.txt can exist in both directory /home/greg and in /home/other, but two copies of foo.txt cannot co-exist in the same directory. In addition, to access the foo.txt file outside of the /home/greg directory, we must prepend the directory name to the file name using the directory separator to get /home/greg/foo.txt. This same principle extends to namespaces in the programming world.*

*Php*

*namespace my\name; // see "Defining Namespaces" section*

*Javascript*

*var yourNamespace = {*

*foo: function() {*

*},*

*bar: function() {*

*}*

*};*

*...*

*yourNamespace.foo();'=*

**What makes a good language good and a bad language bad?**Programmer

**Write two functions, one Referentially Transparent and the other one Referentially Opaque. Discuss.**

*In functional programming, referential transparency is generally defined as the fact that an expression, in a program, may be replaced by its value (or anything having the same value) without changing the result of the program. This implies that methods should always return the same value for a given argument, without having any other effect. This functional programming concept also applies to imperative programming, though, and can help you make your code clearer.*

**Whats the Stack and what's the Heap? What's a Stack Overflow?**

*Stack:*

*Stored in computer RAM just like the heap.*

*Variables created on the stack will go out of scope and are automatically deallocated.*

*Much faster to allocate in comparison to variables on the heap.*

*Implemented with an actual stack data structure.*

*Stores local data, return addresses, used for parameter passing.*

*Can have a stack overflow when too much of the stack is used (mostly from infinite or too deep recursion, very large allocations).*

*Data created on the stack can be used without pointers.*

*You would use the stack if you know exactly how much data you need to allocate before compile time and it is not too big.*

*Usually has a maximum size already determined when your program starts.*

*Heap:*

*Stored in computer RAM just like the stack.*

*In C++, variables on the heap must be destroyed manually and never fall out of scope. The data is freed with delete, delete[], or free.*

*Slower to allocate in comparison to variables on the stack.*

*Used on demand to allocate a block of data for use by the program.*

*Can have fragmentation when there are a lot of allocations and deallocations.*

*In C++ or C, data created on the heap will be pointed to by pointers and allocated with new or malloc respectively.*

*Can have allocation failures if too big of a buffer is requested to be allocated.*

*You would use the heap if you don't know exactly how much data you will need at run time or if you need to allocate a lot of data.*

*Responsible for memory leaks.*

**Why is it important that in a language functions are first class citizens?**

*In computer science, a programming language is said to support first-class functions (or function literal) if it treats functions as first-class objects. Specifically, this means that the language supports constructing new functions during the execution of a program, storing them in data structures, passing them as arguments to other functions, and returning them as the values of other functions.*

**Some languages, especially the ones that promote a Functional approach, allow a technique called Pattern Matching. Do you know it? How is Pattern Matching different from Switch clauses?**

*Having formerly been one of "those people", I don't know that there's a succinct way to sum up why pattern-matching is such tasty goodness. It's experiential.*

*Back when I had just glanced at pattern-matching and thought it was a glorified switch statement, I think that I didn't have experience programming with algebraic data types (tuples and discriminated unions) and didn't quite see that pattern matching was both a control construct and a binding construct. Now that I've been programming with F#, I finally "get it". Pattern-matching's coolness is due to a confluence of features found in functional programming languages, and so it's non-trivial for the outsider-looking-in to appreciate.*

*I tried to sum up one aspect of why pattern-matching is useful in the second of a short two-part blog series on language and API design; check out part one and part two.*

**Why do some languages have no exceptions by design? What are the pros and cons?**

**If Cat is an Animal, is TakeCare<Cat> a TakeCare<Animal>?**

**Why in Java, C# and many other languages constructors are not part of the interface?**

*Generally constructors are for initializing non-static members of particular class with respect to object. There is no object creation for interface as there is only declared methods but not defined methods. ... so object creation is not possible for interface.*