DOWNLOAD PDF 1. A GENTLE INTRODUCTION FOR NON-PROGRAMMERS

Chapter 1 : A Gentle Introduction for Non-Programmers - ActionScript: The Definitive Guide [Book]

A Gentle Introduction for Non-Programmers I'm going to teach you to talk to Flash. Not just to program in Flash but to say things to it and to listen to what it has to say in return.

Looks like we are good to go! Methods to Check Stationarity The next step is to determine whether a given series is stationary or not and deal with it accordingly. This section looks at some common methods which we can use to perform this check. Visual test Consider the plots we used in the previous section. We were able to identify the series in which mean and variance were changing with time, simply by looking at each plot. Similarly, we can plot the data and determine if the properties of the series are changing with time or not. It is better to confirm the observations using some statistical tests. Statistical test Instead of going for the visual test, we can use statistical tests like the unit root stationary tests. Unit root indicates that the statistical properties of a given series are not constant with time, which is the condition for stationary time series. Here is the mathematics explanation of the same: Suppose we have a time series: In order to calculate yt we need the value of yt-1, which is: This is knows as unit root in a time series. We know that for a stationary time series, the variance must not be a function of time. Below are the two of the most commonly used unit root stationary tests: It can be used to determine the presence of unit root in the series, and hence help us understand if the series is stationary or not. The null and alternate hypothesis of this test are: The series has no unit root. If we fail to reject the null hypothesis, we can say that the series is non-stationary. This means that the series can be linear or difference stationary we will understand more about difference stationary in the next section. The results of our test for this particular series are: Results of Dickey-Fuller Test: If the test statistic is less than the critical value, we can reject the null hypothesis aka the series is stationary. When the test statistic is greater than the critical value, we fail to reject the null hypothesis which means the series is not stationary. This confirms our original observation which we initially saw in the visual test. The authors of the KPSS test have defined the null hypothesis as the process is trend stationary, to an alternate hypothesis of a unit root series. We will understand the trend stationarity in detail in the next section. The process is trend stationary. The series has a unit root series is not stationary. For the air passengers dataset, here are the results: If the test statistic is greater than the critical value, we reject the null hypothesis series is not stationary. If the test statistic is less than the critical value, if fail to reject the null hypothesis series is stationary. For the air passenger data, the value of the test statistic is greater than the critical value at all confidence intervals, and hence we can say that the series is not stationary. I usually perform both the statistical tests before I prepare a model for my time series data. It once happened that both the tests showed contradictory results. One of the tests showed that the series is stationary while the other showed that the series is not! I got stuck at this part for hours, trying to figure out how is this possible. As it turns out, there are more than one type of stationarity. So in summary, the ADF test has an alternate hypothesis of linear or difference stationary, while the KPSS test identifies trend-stationarity in a series. Types of Stationarity Let us understand the different types of stationarities and how to interpret the results of the above tests. A strict stationary series satisfies the mathematical definition of a stationary process. For a strict stationary series, the mean, variance and covariance are not the function of time. The aim is to convert a non-stationary series into a strict stationary series for making predictions. A series that has no unit root but exhibits a trend is referred to as a trend stationary series. Once the trend is removed, the resulting series will be strict stationary. The KPSS test classifies a series as stationary on the absence of unit root. This means that the series can be strict stationary or trend stationary. A time series that can be made strict stationary by differencing falls under difference stationary. ADF test is also known as a difference stationarity test. Let us look at the possible outcomes of applying these stationary tests. Making a Time Series Stationary Now that we are familiar with the concept of stationarity and its different types, we can finally move on to actually making our series stationary. Always keep in mind that in order to use time series forecasting models, it is necessary to convert any non-stationary

series to a stationary series first. Differencing In this method, we compute the difference of consecutive terms in the series. Differencing is typically performed to get rid of the varying mean. Mathematically, differencing can be written as: For example, an observation taken on a Monday will be subtracted from an observation taken on the previous Monday. Mathematically it can be written as: Common transformation methods include power transform, square root, and log transform. You can use square root or power transformation on the series and see if they come up with better results. Feel free to share your findings in the comments section below! End Notes In this article we covered different methods that can be used to check the stationarity of a time series. The next step is to apply a forecasting model on the series we obtained. You can refer to the following article to build such a model: You can connect with me in the comments section below if you have any questions or feedback on this article.

DOWNLOAD PDF 1. A GENTLE INTRODUCTION FOR NON-PROGRAMMERS

Chapter 2 : calendrierdelascience.com: statistics gentle introduction

Some Basic Phrases. On the first day of any language school you'd expect to learn a few basic phrases ("Good day," "How are you," etc.). Even if you're just memorizing a phrase and don't know what each word means, you can learn the effect of the phrase and can repeat it to produce that effect.

Doing it with style What the Func? What is all the hype about Functional JavaScript? And why is it called functional? What is it good for? Why would you bother? To me, learning functional programming is a little bit like getting a Thermomix: But, it does make certain tasks a whole lot easier. It can even automate certain things that would otherwise be fairly tedious and time-consuming. In JavaScript we have two key building blocks: Variables are sort-of like containers that we can put things in. You can create one like so: Come see how good I look! One can create a function like so: Our log function is a variable; which means that we can do the same things with it that we can do with other variables. Could we, maybe, pass a function as a parameter to another function? And this one little feature is surprisingly powerful. To understand why though, we need to talk about DRY code. The idea is that if you need to do the same set of tasks many times, bundle them up into some sort of re-usable package like a function. This way, if you ever need to tweak that set of tasks, you can do it in just one spot. Say we wanted to put three carousels on a page using a carousel library an imaginary library for the sake of example: We want to initialise a carousel for the elements on the page, each one with a specific ID. And we have a pattern to follow: But what about if we have a pattern where the action changes? We could get part of the way there by wrapping the document. That would save us a little bit of repetition: And easier to maintain. The ability to pass functions around as variables opens up a lot of possibilities.

DOWNLOAD PDF 1. A GENTLE INTRODUCTION FOR NON-PROGRAMMERS

Chapter 3 : A Gentle Introduction to Programming for Non-Programmers

CS 01 â€" A Gentle Introduction to Programming for Non-Programmers 1 Unit. Fees. $ Instructor(s): This course will provide a gentle introduction to coding.

Not just to program in Flash but to say things to it and to listen to what it has to say in return. This is not a metaphor or simply a rhetorical device. Programming languages are used to send information to and receive information from computers. They are collections of vocabulary and grammar used to communicate, just like human languages. Using a programming language, we tell a computer what to do or ask it for information. It listens, tries to perform the requested actions, and gives responses. So while you may think you are reading this book in order to "learn to program," you are actually learning to communicate with Flash. Learning to speak a computer language is sometimes considered synonymous with learning to program. What would happen if we were to say, "Flash, make a ball bounce around the screen? What Flash expects us to describe is the objects in the world it knows: How big should the ball be? Where should it be placed? In which direction should it begin traveling? How fast should it go? Around which part of the screen should it bounce? In two dimensions or three? A ball is a circular movie clip symbol named ball. A square is a four-sided movie clip symbol named square. Make a new green ball 50 pixels in diameter. Make a new black square pixels wide and place it in the middle of the Stage. Continue until I tell you to stop. Even though we gave our instructions in English, we still had to work through all the logic that governs our bouncing ball in order for Flash to understand us. Our hypothetical English-speaking-Flash example exposes four important aspects of programming: No matter what the language, the art of programming lies in the formulation of logical steps. Before you try to say something in a computer language, it usually helps to say it in English. A conversation in one language translated into a different language is still made up of the same basic statements. They also have a very limited vocabulary. Most programming has nothing to do with writing code. Once your program has been described sufficiently at the conceptual level, you can translate it into ActionScript. In programming -- as in love, politics, and business -- effective communication is the key to success. For Flash to understand your ActionScript, you have to get your syntax absolutely correct down to the last quote, equal sign, and semicolon. What may be obvious to you is not obvious to a computer. Think of programming a computer like talking to a child: The rest of the book will build on that foundation. Nearly all ActionScript programming takes place in the Actions panel. Any instructions we add to the Actions panel are carried out by Flash when our movie plays. Open the Actions panel now by following these steps: Launch Flash with a new blank document. On the main timeline, select frame 1 of layer 1. The Actions panel is divided into two sections: The Script pane houses all our code. Figure shows all available Actions, including some old friends from Flash 2, 3, and 4. TIP So-called Actions are more than just Actions -- they include various fundamental programming-language tools: Throughout the book, I use the appropriate programming term to describe the Action at hand. Ready to get your hands dirty? Say Hi to Flash Before you can type code into the Actions panel, you must disengage the ActionScript autopilot as follows: Next, select frame 1 of layer 1. And now, the exciting moment -- your first line of code. Type the following into the Script pane: On the line below it, type your second and third lines of code, shown following this paragraph. Replace your name here with your first name whenever you see italicized code in this book it means you have to replace that portion of the code with your own content: Nothing has happened yet. Select Control Test Movie and see what happens. Some text should appear in the Output window as shown in Figure Flash gets friendly Pretty neat, eh?! Keeping Track of Things Variables Remember how I said programming was really just communicating with a computer? In your first line of code: You said something more like this: Flash, please remember a piece of information for me -- specifically, the phrase "Hi there, Flash! If I ask you for message later, give me back the text "Hi there, Flash! Flash can remember something for you, provided that you label it so that it can be found later. For example, in your second line of code, we had Flash remember your first name, and we named the reference to it firstName.

Flash remembered your name and displayed it in the Output window when you tested your movie. The fact that Flash can remember things for us is crucial in programming. Flash can remember any type of data, including text such as your name , numbers such as 3. Official variable nomenclature Time for a few formal terms to describe how Flash remembers things. So far you know that Flash remembers data. An individual piece of data is known as a datum. We say that the variable stores or contains its value. Note that "Hi there, Flash! In your first line of code, you specified the value of the variable message. But before you can assign a value to a variable, you must first create it. We formally bring variables into existence by declaring them using the special keyword var, which you used earlier. Declare a new variable named message, and assign it the initial value "Hi there, Flash! Your third and fourth lines, however, are a little different: In the third line, Flash displayed the value of the variable message. In the last line, Flash also converted the variable firstName to its value whatever you typed and stuck that into the sentence after the words "Hi there. The question is, what made the trace command place your text in the Output window? The interpreter translates your ActionScript into a language that the computer understands and uses to carry out your code. During movie playback, the interpreter is always active, dutifully attempting to understand commands you give it. If a command generates a result, the interpreter provides that response to you. Consider this command as the interpreter would: The interpreter also knows that trace is used to display text in the Output window, so it also expects to be told which text to display. It finds "Nice night to learn ActionScript. The semicolon acts like the period at the end of a sentence; with few exceptions, every ActionScript statement should end with a semicolon. With the statement successfully understood and all the required information in hand, the interpreter translates the command for the processor to execute, causing our text to appear in the Output window. The interpreter is always listening for your instructions. The interpreter has to read your code, letter by letter, and try to understand it. This is the same as you trying to read and understand a sentence in a book. Strangely enough, my dad always told me the best way to learn a new language is to find a lover that speaks it. May I, therefore, be the first to wish you all the best in your new relationship with the ActionScript interpreter. To supply an argument to a command, enclose the argument in parentheses, like this: For example, in the code gotoAndPlay 5 , gotoAndPlay is the name of the command, and 5 is the argument being passed in this case the frame number. Some commands, such as stop , require parentheses but do not accept arguments. The operators of a programming language are akin to conjunctions "and," "or," "but," etc. In the trace example, the plus operator joins the quoted text "Hi there, " to the text contained in the variable firstName. All operators link phrases of code together, manipulating those phrases in the process. Whether the phrases are text, numbers, or some other datatype, an operator nearly always performs some kind of transformation. Very commonly, operators combine two things together, as the plus operator does.

DOWNLOAD PDF 1. A GENTLE INTRODUCTION FOR NON-PROGRAMMERS

Chapter 4 : Intro to Java 8 for Non-Programmers (TT) - Global Knowledge

Python for Non-programmers A Gentle Introduction 1 Yann Tambouret Scienti c Computing and Visualization Information Services & Technology Boston University.

Books Each of these books can be purchased online and is also available as a completely free website. It was updated to Python 3 by Peter Wentworth. Python 3 website print version Interactive Courses These sites give you instant feedback on programming problems that you can solve in your browser. Python on Codecademy Python 2 Code the blocks combines Python programming with a 3D environment where you "place blocks" and construct structures. It also comes with Python tutorials that teach you how to create progressively elaborate 3D structures. Computer Science Circles has 30 lessons, exercises, and a message system where you can ask for help. Teachers can use it with their students. It is also available in Dutch, French, German and Lithuanian. It has 57 interactive exercises and 11 videos. Finxter - How good are your Python skills? How to Think Like a Computer Scientist: Interactive Edition Python 3. Python story-based game Python 2 Merscythe: Adventures with the Codue is a story-based game for learning Python. The tutorials provide feedback and hints. K Oriented for Children please keep this list alphabetized Build a "Pypet" Learn programming fundamentals in Python while building a tamagotchi style "Pypet" by Tatiana Tylosky. Guido van Robot A teaching tool in which students write simple programs using a Python-like language to control a simulated robot. Field-tested at Yorktown High School, the project includes a lesson plan. PythonTurtle A learning environment for Python suitable for beginners and children, inspired by Logo. Geared mainly towards children, but known to be successful with adults as well. Young Coders tutorial Python 3 This is the full text of the tutorial taught annually at PyCon North America , with examples and exercises throughout. This tutorial starts with basic skills and builds to working with complex logic and games. Appropriate for ages 10 and up, including adult beginners, Tutorials and Websites please keep this list alphabetized A Byte of Python , by Swaroop C. Python 2 Learning to Program An introduction to programming for those who have never programmed before, by Alan Gauld. It introduces several programming languages but has a strong emphasis on Python. Learn Python An Introductory yet in-depth tutorial for Python beginners. This tutorial by Danny Yoo has been translated into nine different languages. Python 2 The Python tips blog includes Python tips and tutorials for beginners and professional programmers. It is available for both Python 2 and Python 3. Pythonspot Tutorials Python tutorials. The Python Guru A beginner friendly guide for aspiring programmers. Top Courses to Learn Python - gitconnected. This course material is still preliminary and assumes some high school-level maths. It does not cover object-oriented programming or graphical applications. Python 2 The Programming Historian is a tutorial-style introduction to programming for practicing historians. Python 2 Python for Number Theory is a series of Python notebooks for Jupyter for applications to number theory and cryptography. They assume no prior programming experience, and are suitable for someone learning elementary number theory at the same time.

DOWNLOAD PDF 1. A GENTLE INTRODUCTION FOR NON-PROGRAMMERS

Chapter 5 : A Gentle Introduction for Non-Programmers (ActionScript: The Definitive Guide)

Introduction This Tutorial This is for non-programmers. The rst half is very gentle. The second half is more in depth. If you have any programming experience, feel free to.

I am the president and was the secretary of the Oxford University Computer Society, and a member of the Oxford University History and Cross Country societies. I also lead a group of Microsoft Student Partners at the university to run talks and hackathons. Introduction Over the last decade functional programming rose in prominence such that now nearly all high-level programming languages support it in some way, and I firmly believe that learning functional programming techniques have the potential to affect every facet of modern software engineering practices. Whilst there is value in exploring the functional programming features of your language of choice, it is often worth exploring them in a language that was designed from the ground up to be functional. This way you can see interesting or useful ideas applied in a succinct manner; the same ideas may be harder to apply in other languages, but could reduce code complexity later on all the same. In order to explore functional programming we will use F , a language by Microsoft that works on top of the. Functions as values The core tenet of functional programming is that functions can be treated as values, and passed around just like any other value. As a simple example, here is a TypeScript function that takes a function value as an argument, calls it, and returns its result: Now suppose we define a simple function and use the call function: A very common use-case for functional programming is working with lists of data. Suppose that we have a list of names, and we want to filter out any names longer than a certain length, then tag each name with its length, and then concatenate all of these strings together. A typical function written in an imperative style for this might look like: Each of these takes a function, applies that function to each element in the array, and then returns a new array. This therefore allows to rewrite the above function as: In F we can declare lists either with each entry separated by a newline or each entry separated by a semicolon; a newline is sufficient for lines of code. The printfn function is similar to the printf function in C and it supports the same formatting specifiers. Note that the way that we call the function is different to ES6: Importantly, however, when we declare a value like this we cannot change the value associated with it later The parameter names of the function filterNamesFunctional are names and maxLength, just like before. Like TypeScript, F is also statically typed types are checked at compile time , but it also supports type inference, which means that in most cases the compiler can work out the type without the programmer need to declare it fun arg0 arg Suppose we we want to double a number and square it. Two such approaches could be: It takes a value, passes it to the double function, passes the result of that to the square function, and then returns the result. In reality, whenever you have a function that appears to take more than one argument, the function is actually written so that it returns a function that takes the next set of arguments. For example, we can create a function that always adds 42 to its input: Typically this is represented with a pointer either pointing to some value, or null. This is incredibly fragile, and can easily break! For example, the following TypeScript code will crash when it is run: F helps to prevent this by introducing an Option type and forcing you to unwrap it whenever you wish to use it: This is similar to a switch statement in TypeScript or many other programming languages, but it supports pattern matching. There are, however, several more compact ways to write this function; you can see examples in the F guide. If we have several independent function calls to side-effect free functions then we can be sure that if we execute them in parallel they will not adversely affect one another. For example, F makes it possibly to map over a list in parallel by automatically running the mapping function in separate threads where possible: It is possible to define a function similar to PSeq in other programming languages, however it may be easier to introduce bugs if the function passed to it is not side-effect free; F helps prevent that. Units of measure Whilst none of the features described so far are unique to F and are in fact present in most functional programming languages , there are certainly features that are, and units of measure is one of them. Typically when programming we might represent a physical quantity, such as a distance, with a value

such as a double. This is fine if you can always be certain of the unit that you are dealing with, but often you may not be. Units of measure add type safety to these values, without sacrificing the ability to use functions already defined for numeric types.

Chapter 6 : A Gentle Introduction to the Basics of Machine Learning

Introduction Over the last decade functional programming rose in prominence such that now nearly all high-level programming languages support it in some way, and I firmly believe that learning functional programming techniques have the potential to affect every facet of modern software engineering practices.

What is math Before starting to address the subject of Machine Learning, it is essential to understand first what is mathematics and why they are so important. Math is the language of science. It allows scientist and engineers to express the Universe and the phenomenons that happen inside it. As any language, it has tenses: The most useful tense in math is the future, i. We are very interested in predicting the future, or knowing how a system will evolve over time starting from an initial state. We can predict the future trajectory of a meteorite to know if it will hit the Earth so we can call Bruce Willis or we can predict the sales of a product in Amazon so we can make Jeff Bezzos happy. Anyhow, predicting is important. In order to play with math we need numbers and variables which are just letters that represent different values. These numbers and variables create the most important mathematical resource to talk about the Universe, the model. A model is just an equation that represents a concept or event in the mathematical language. There are two kind of models, analytical models, like the ones previously mentioned and learned models. Until the computer era, most scientists used analytical models, mainly because in order to develop learned models it is necessary to compute many operations. Other significant difference between analytical and learned models is that, usually, analytical models are derived using a mathematical demonstration, which are a set of steps that reach a final equation. On the contrary, learned models are obtained via a process called training or optimization, and need many examples of inputs-outputs to obtain the equation. For this reason, analytic models tend to be compact and beautifull equations, while learned models tend to have hundreds of parameters, generating a very complex and large equation. What is Machine Learning Machine Learning is a technique that allows to obtain learned models to generate predictions. The only thing you need to create a machine learning model is a group of inputs that generate a group of outputs. The model is just the relationship between inputs and outputs. This specific problem is called classification. In this case the equation is: Since the point is below the line, then the output would be the class blue. The concept of optimization The last key concept to understand is the concept of optimization, which in the machine learning nomenclature is called training. So there are two trajectories, the desired one and the actual one. Optimization is the process of tuning the parameters of the model to make the desired and the actual output as similar as possible. The difference between the desired and actual output is called the error. Therefore, optimization is equivalent to minimize the error. This process is simulated in the first image, when the line is moving. Later, as the parameters change, the line divides the classes perfectly, reaching a minimal error. But they all follow the same basic principles addressed in this post. All this compose the big world of machine learning algorithms.

Chapter 7 : A Gentle Introduction to Functional JavaScript: Part 1

A Gentle Introduction to Handling a Non-Stationary Time Series in Python. Python Time Series. A Gentle Introduction to Handling a Non-Stationary Time Series in Python.

Not just to program in Flash but to say things to it and to listen to what it has to say in return. This is not a metaphor or simply a rhetorical device. Programming languages are used to send information to and receive information from computers. They are collections of vocabulary and grammar used to communicate, just like human languages. Using a programming language, we tell a computer what to do or ask it for information. It listens, tries to perform the requested actions, and gives responses. Learning to speak a computer language is sometimes considered synonymous with learning to program. What Flash expects us to describe is the objects in the world it knows: How big should the ball be? Where should it be placed? In which direction should it begin traveling? How fast should it go? Around which part of the screen should it bounce? In two dimensions or three? A ball is a circular movie clip symbol named ball. A square is a four-sided movie clip symbol named square. Make a new green ball 50 pixels in diameter. Make a new black square pixels wide and place it in the middle of the Stage. Continue until I tell you to stop. Even though we gave our instructions in English, we still had to work through all the logic that governs our bouncing ball in order for Flash to understand us. Our hypothetical English-speaking-Flash example exposes four important aspects of programming: No matter what the language, the art of programming lies in the formulation of logical steps. Before you try to say something in a computer language, it usually helps to say it in English. A conversation in one language translated into a different language is still made up of the same basic statements. They also have a very limited vocabulary. Most programming has nothing to do with writing code. Once your program has been described sufficiently at the conceptual level, you can translate it into ActionScript. In programmingâ€”as in love, politics, and businessâ€”effective communication is the key to success. For Flash to understand your ActionScript, you have to get your syntax absolutely correct down to the last quote, equal sign, and semicolon. What may be obvious to you is not obvious to a computer. Think of programming a computer like talking to a child: The rest of the book will build on that foundation. Nearly all ActionScript programming takes place in the Actions panel. Any instructions we add to the Actions panel are carried out by Flash when our movie plays. Open the Actions panel now by following these steps: Launch Flash with a new blank document. On the main timeline, select frame 1 of layer 1. The Actions panel is divided into two sections: The Script pane houses all our code. Tip So-called Actions are more than just Actionsâ€”they include various fundamental programming-language tools: Throughout the book, I use the appropriate programming term to describe the Action at hand. Ready to get your hands dirty? Say Hi to Flash Before you can type code into the Actions panel, you must disengage the ActionScript autopilot as follows: Next, select frame 1 of layer 1. And now, the exciting momentâ€”your first line of code. Type the following into the Script pane: On the line below it, type your second and third lines of code, shown following this paragraph. Replace your name here with your first name whenever you see italicized code in this book it means you have to replace that portion of the code with your own content: Nothing has happened yet. Some text should appear in the Output window as shown in Figure 1. Flash gets friendly Pretty neat, eh?! Keeping Track of Things Variables Remember how I said programming was really just communicating with a computer? In your first line of code: You said something more like this: Flash can remember something for you, provided that you label it so that it can be found later. For example, in your second line of code, we had Flash remember your first name, and we named the reference to it firstName. Flash remembered your name and displayed it in the Output window when you tested your movie. The fact that Flash can remember things for us is crucial in programming. Flash can remember any type of data, including text such as your name , numbers such as 3. Official variable nomenclature Time for a few formal terms to describe how Flash remembers things. So far you know that Flash remembers data. An individual piece of data is known as a datum. We say that the variable stores or

contains its value. In your first line of code, you specified the value of the variable message. But before you can assign a value to a variable, you must first create it. We formally bring variables into existence by declaring them using the special keyword var, which you used earlier. Your third and fourth lines, however, are a little different: In the third line, Flash displayed the value of the variable message. The question is, what made the trace command place your text in the Output window? The interpreter translates your ActionScript into a language that the computer understands and uses to carry out your code. During movie playback, the interpreter is always active, dutifully attempting to understand commands you give it. If a command generates a result, the interpreter provides that response to you. Consider this command as the interpreter would: The interpreter also knows that trace is used to display text in the Output window, so it also expects to be told which text to display. The semicolon acts like the period at the end of a sentence; with few exceptions, every ActionScript statement should end with a semicolon. With the statement successfully understood and all the required information in hand, the interpreter translates the command for the processor to execute, causing our text to appear in the Output window. The interpreter is always listening for your instructions. The interpreter has to read your code, letter by letter, and try to understand it. This is the same as you trying to read and understand a sentence in a book. Strangely enough, my dad always told me the best way to learn a new language is to find a lover that speaks it. May I, therefore, be the first to wish you all the best in your new relationship with the ActionScript interpreter. To supply an argument to a command, enclose the argument in parentheses, like this: For example, in the code gotoAndPlay 5 , gotoAndPlay is the name of the command, and 5 is the argument being passed in this case the frame number. Some commands, such as stop , require parentheses but do not accept arguments. All operators link phrases of code together, manipulating those phrases in the process. Whether the phrases are text, numbers, or some other datatype, an operator nearly always performs some kind of transformation. Very commonly, operators combine two things together, as the plus operator does. But other operators compare values, assign values, facilitate logical decisions, determine datatypes, create new objects, and provide various other handy services. Arithmetic operations are the easiest operations to understand because they follow basic mathematics: But some operators will be less recognizable to you because they perform specialized programming tasks. Take the typeof operator, for example. It tells us what kind of data is stored in a variable. So, if we create a variable x, and give it the value 4, we can then ask the interpreter what datatype x contains, like this: Notice that we provide the typeof operator with a value upon which to operate, but without using parentheses: You might therefore wonder whether or not x is an argument of typeof. An operand is an item upon which an operator operates. Chapter 5 , covers all of the ActionScript operators in detail. For now just remember that operators link phrases of code in some kind of transformation. Here, again, is line one:

Chapter 8 : BeginnersGuide/NonProgrammers - Python Wiki

of 27 results for "statistics gentle introduction" Statistics: A Gentle Introduction May 3, by Frederick L. Coolidge. Paperback. $ $ 20 57 to rent Prime.

Chapter 9 : A gentle introduction to R

Throughout this gentle introduction to programming and Java, you will learn the best practices for writing object-oriented programs in Java 8, using sound development techniques. You will learn foundation level skills that maximize performance, using Java 8 capabilities for addressing rapid application development.