Hacking Orientalism:
Theorizing Arabic Digital Writing

Clay Foye

Dartmouth College
Middle Eastern Studies Thesis
Advisor: yasser elhariry

I dedicate this thesis to my friends, advisors, mentors, and confidants
Tarek El-Ariss and yasser elhariry

Contents

Acknowledgements

I acknowledge

Places	Anthems	People
White River Junction, VT Hanover, NH Boston, MA Louisville, KY quiet corners of libraries unnamed coffee shops Marrakech, 3 AM Rabat, 12 PM Tangier, 7 AM depths of my computer Dartmouth Organic Farm The Connecticut River (and its currents) Frozen Connecticut River The Ohio River (and its adjacencies) inside of my car stereo	*Selected Ambient Works, 85-92* *Ants From Up There* *For the First Time* *Scary Monsters (And Super Creeps)* *Rainbow in Curved Air* *Partita for 8 Voices* *Anima* *OK Computer* *Sound of Silver* *American Water* *Apocalypse* *A River A'int Too Much To Love* *Crooked Rain, Crooked Rain* *The Glow, Pt. 2* *I Made a Place* *I See a Darkness* *Mixing Colors* *Kankyō Ongaku* *Hello Nasty* *Vespertine* *Spiderland* *Closer* *Computer World* *Ibn El Leil* *Evil Empire* *Yankee Hotel Foxtrot* *EP7* *Schlagenheim* *Cavalcade* *Mr. Morale & The Big Steppers* *Lift Your Skinny Fists Like Antennas to Heaven* *Remain in Light*	my family my girlfriend my friends my professors my mentors

Introduction

This text tries desperately to alarm. My thesis is about the gaps and bugs of Arabic writing on computers. For those who are ignorant of these bugs, it serves as an introduction to the problems Arabic speakers face every day on the internet. For those who are aware of these bugs and have resigned themselves to accepting them, it theorizes their complacency as modern Orientalism: digital Orientalism. Finally, for those who are as alarmed as I am, this thesis provides a structure of theory and key vocabulary.

The current state of Arabic writing on computers is abysmal. It is not uncommon for letters to not be joined, flipped around, and incorrectly displayed. In fact, all major word processors analyzed in the thesis have a problem with displaying Arabic text. These problems are so severe that they render the resulting text illegible, indecipherable, and unpublishable. So, I probe the following questions: what are the current problems with Arabic writing on computers? How are these problems addressed? Are these problems addressed? Are these problems endemic or specific? What documentation exists for these problems? In what situations does Arabic text break down? Do these problems belong to a larger historical context? Finally, what theoretical analyses can be done of these problems by way of introducing them to the academic discourse of Comparative Literature?

Chapter 1 begins by showing what Arabic code looks like. Ramsey Nasser, an artist and coder, has created an all-Arabic coding language called *Qlb*. This project was doomed to fail, and its failure is the jumping-off point of the thesis. The following questions are posed: why is Arabic code radical? How does Arabic code fit into and interact with other coding languages? Then, I discuss a table

published by the World Wide Web Consortium describing the current language support on the internet, which shows in vibrant color the stark differences between Latin and non-Latin digital writing. As the chapter approaches these questions and texts, the chapter itself begins to break apart as even the displaying of Arabic writing causes bugs and unexpected behavior. I then review popular word processors and the state of Arabic writing on them using the research of Titus Nemeth.

In order to understand the processual breakage of my own writing, I theorize the process of *digital writing*. Digital writing is writing which *is happening* and places an overwhelming emphasis on the *symptom* of writing. Digital writing offers a new understanding of what a text is by adopting code, comments, blog posts, tweets, error messages, and bug reports. Thus, its symptoms include software, applications, hardware, robotics, bugs, simulations, algorithms. In *Leaks, Hacks, and Scandals: Arab Culture in the Digital Age*, Tarek El-Ariss includes other examples of digital writing's texts by identifying digital writing as "a new writing" which is "emerging from leaking and the fiction of scandal as well as untraceable tweets and data mining, online fights and trolling campaigns, and flickering texts which appear and disappear."[1] I place this type of writing alongside *physical writing*. Physical writing exists as ink on paper, charcoal on cave walls, as tattoo, or any writing which is not stored in electronic memory. The key differentiation of *physical* and *digital* writing is *speed*. The speed at which digital writing occurs, changes, affects, and is affected is exponentially faster than that of physical writing. Physical writing happens at the speed of ink drying, while digital writing happens at the speed of light. Consequently, the speed at which digital writing establishes and destroys is much faster than that of physical writing.

[1]El-Ariss, *Leaks, Hacks, and Scandals,* 7

The relationship between digital writing's originating code and resulting text is not one of primary and secondary; form and shadow. The origination of the digital text is always in question. As El-Ariss describes in his introduction, Foucault's "condemned body" has "leaked out" of "secret prisons and police stations." Just as the digital body escapes, so does digital writing. With its speed and endless production of applications, bugs, and code, digital writing cannot be captured by penal codes. Hacking, breaking, bugs, and other forms of exposure and slippage dodge and weave past the institutional hopes of closed-source code and applications.

In the opening pages of *Palimpsests: literature in the second degree*, Gérard Genette describes five types of textual relationships. The fourth kind (which he mentions after the fifth) he "rebaptizes" as "*hypertextuality*" (both his and my italics).[2] Genette defines hypertextuality (and subsequently "hypertext" and "hypotext") as "any relationship uniting a text B (which I shall call the *hypertext*) to an earlier text A (I shall, of course, call the *hypotext*) upon which it is grafted in a manner which is not commentary."[3] Genette's hypertextuality re-emerges in the digital age not as a relationship between text A and B, but rather between source code and resulting text, between one library and another, and between a single C file and hundreds of other C files. The digital age explodes the *duality* of Genette's hypertextuality into a never-ending and never-slowing act of "grafting." Genette suggests that his grafting is "metaphorical" yet I disagree: renderings, bugs, vulnerabilities, and other gaps are grafted onto the body of digital writing. In fact, *grafting is digital writing*: a never-ending process of suture and attachment onto a dismembered body. This grafting of text-as-limb misplaces the center of the

[2] Genette, *Palimpsests*, 5
[3] Genette, *Palimpsests,* 5

digital body and complicates authorship and possession. Hypertexts and hypotexts are not "transformations" [4] of one text to another, as Genette describes the *Odyssey* and *Ulysses*, but rather transmogrifications of each other. Digital hypertextuality is not simply a question of what a text "evokes" nor does it depend upon "critical ingenuity."[5] Rather, digital hypertextuality is so explicit that hypertextuality is a defining feature of the text itself: references to other files in a C file or the source code of a webpage are dependent upon hypertextuality for their existence. At the same time, this entire thesis is a suggestion that a new kind of hypertextuality has emerged from digital writing: one which exists as a meta-hypertextuality beyond the explicit textual relationships of digital writing and invites comparative readings. Perhaps the absurdity of the prefixed prefix "meta-hyper" demonstrates best what happens when digital writing encounters canonical theory such as Genette's.

Chapter 2 explores this absurd excess by beginning with a table of its own: the appendix of Ahmad Faris al-Shidyaq's *al-Saq ala al-Saq*. By reading digital writing and the writing of al-Shidyaq comparatively, I establish a basis for my scholarship. My comparative reading serves other purposes, including opening an El-Arissian portal between the digital age and the *Nahda*. al-Shidyaq's writing extends the breakage of digital writing as he and I attack in parallel the penal codes and structures of discipline which propagate digital Orientalism. I then demonstrate my comparative project with two examples of the same structure. This structure consists of a technical concept, such as keywords or external linkage, read comparatively alongside al-Shidyaq's canonical writings of the *Nahda*,

[4] Genette, *Palimpsests*, 5
[5] Genette, *Palimpsests*, 9

specifically *al-Saq*. This structure is established in Chapter 2, but serves as a guideline for my future research and writing.

El-Ariss recognizes the digitization and erasure of the textual center in *Leaks*: "it has become impossible to engage these texts and political practices by trying to identify their origin, commonality, structure, and essence, thereby reproducing the old-fashioned encyclopedic study and nomenclature imposed on literature, culture, and politics."[6] Thus, we are faced with new meta-hypertextual relationships, characterized by movement, error, leakage, and production. The movement happens silently, as data travels across servers by cables and airwaves. Errors in the communication between programs and digital texts create their own texts, and interrupt the expected flow of writing. El-Ariss theorizes these errors using *leakage* and *scandal*, while I approach them by theorizing the *bug*: a symptom which serves as its own text, that exists to interrupt or change the speed of digital writing by slowing or speeding up program flow. Communities gather around bugs, dedicated to their *documentation* and *resolution*. While bugs are able to alienate entire populations, I later show that bugs are actually openings to be pried apart by the hacker or theorist (I claim to be both) by revealing systems of oppression. Bug reports are a structure of documentation that seeks to pinpoint the origin of the bug. I theorize bug reports as palimpsestic textual realizations of the bug that eventually themselves spin off, creating new bugs and digital texts.

Chapter 3 begins by theorizing the bug and focusing on what it means to *resolve* the bug. Bugs are described according to their current status: resolved or unresolved. This dichotomy is then challenged and upended. Then, Chapter 3 approaches the *bug report* as a digital genre which embodies

[6] El-Ariss, *Leaks*, 7

all the characteristics of digital writing while demonstrating digital Orientalism's hegemony. I then read through a bug report posted on the word processor LibreOffice's wiki as a performance of digital writing. Chapter 3 concludes with an interrogation of the *location* of digital writing and Said's *Orientalism*.

This thesis' address of the bug and bug report tangents a reading of Ahmad Faris al-Shidyaq's "Revealing the hidden in European civilization" in Tarek El-Ariss' book, *Trials of Arab Modernity*. El-Ariss' reading wades through the murky, stagnant pools of disease of al-Shidyaq's Europe: rotted carrion for dinner, animal excrement and repulsive bodily expulsions everywhere. El-Ariss' reading of *kashf* (as well as the al-Shidyaq's text *Kashf*) or "revealing" relies upon the breakdown of a body; al-Shidyaq's body. In the third chapter of *Trials*, "Aversion to Civilization," El-Ariss reads the "visual and olfactory affects" of al-Shidyaq's writing as *literary affects* that erupt in "narrative discontinuity."[7] This discontinuity is extrapolated to Europe and modernity: "*Kashf*'s deconstructive play and modes of embodiment unveil 'what's behind the scene' . . . revealing a spectacle of decay and exposing a polluted and failing modernity."[8] I pick up El-Ariss' use of "decay" and "affect" at my scene of disease, stench, and infection: digital writing and the bug. Everywhere in this thesis there are traces of a comparative project which considers the digital as a decaying body: endemic bugs, infected systems, untraceable symptoms, and a "bill of health (*kashf tubbī)*"[9] that takes the form of long lists and tables of bugs.

[7] El-Ariss, *Trials of Arab Modernity*, 74
[8] El-Ariss, *Trials*, 78
[9] El-Ariss, *Trials*, 77

Thus, the digital text is a body characterized by *affect*. Brian Massumi defines *l'affect* in his notes to translation of Gilles Deleuze and Felix Guattari's *Mille Plateaux*: "neither word denotes a personal feeling . . . *L'affect* (Spinoza's *affectus*) is an ability to affect and be affected. It is a prepersonal intensity corresponding to the passage from one experimental state of the body to another and implying an augmentation or diminution in that body's ability to act."[10] Digital texts *act* by creating programs and modifying memory just as they are *acted on* by their resultant texts and other texts. After all, digital writing *is* programs and memory, not simply an actant on those bodies. This becomes clear throughout my thesis, especially as my discussion of bugs eventually becomes a literal production of them. Thereby my writing on digital text *becomes* a digital text as my examples and citations break and err. After all, "affects are becomings."[11] The bug report *becomes* a forum for discussion, a record of leakage, a memory. The bug *becomes* a memory, a report, a new leakage, a vulnerability, a frustration, a symptom, an affect, affection. Digital writing is writing which is not simply *happening*, but rather *becoming*. Unlike physical writing, this *becoming* is ironically a physical process rather than a theorized one. Moreover, digital writing's becoming *is* digital writing itself as it captures all of its detritus, and the detritus captures digital writing right back as it escapes as lines of flight: El-Ariss opens *Leaks, Hacks, and Scandals* with the result of a vulnerability of the Lebanese Government's network. From every point of digital writing, I am able to create a rhizomatic connection to everything else: a hyperlink of meaning and flight.

[10]Gilles Deleuze and Felix Guattari, *Mille Plateaux*, xvi
[11] Deleuze and Guattari, *Mille Plateaux,* 256

My theorization of digital writing as affect should not be understood as another cheer for its democratizing possibilities. Rather, digital writing almost always occurs in this thesis as a new Orientalism. After all, "do not even lines of flight, due to their eventual divergence, reproduce the very formations their function it was to dismantle or outflank?"[12] This *digital Orientalism* uses the texts of digital writing to manifest and assert a hegemony of *Latin code*. I want to connect the tradition of Orientalism to my modern formulation of it. In "Disoriented Orientalism," Abdelkebir Khatibi identifies the three "dominant features" of classical Orientalism. Firstly, Orientalism "aims at the analysis of a determined . . . East" yet "remains rooted in a metaphysical ground."[13] Khatibi describes his Orientalists, Louis Massignon, Henry Corbin, Jacques Berque, Jean-Marie Domenach, as "Rational Orientalists" who are "trying to analyze the economic and material structure of Arab societies."[14] Secondly, Khatibi identifies Orientalism's "positivism [which] is not in contradiction with its spiritualism."[15] Positivism on the productive nature of digital writing to spread and assert its structure of indifference, ignorance, and exclusivity, forming the foundation of digital Orientalism. Thirdly, "Orientalism, whether it is Christian, idealist, or rationalist, is in solidarity with humanism."[16] Orientalism's obsession with humanism, combined with its insatiable desire for fantasy, has led to a structure of digital hardware which presumes the figure of the Arab has no need for technology, what Khatibi refers to the "technical devilry" which "God has abandoned the West to."[17] After establishing this structural exclusion, digital Orientalists are then the ones who document and lament the state of

[12] Deleuze and Guattari, *Mille Plateaux,* 13
[13] Khatibi, "Disoriented Orientalism," 76
[14] Khatibi, Orientalism, 77
[15] Khatibi, Orientalism, 77
[16] Khatibi, Orientalism, 78
[17] Khatibi, Orientalism, 77

Arabic digital writing. We will see examples of this in all three chapters: the World Wide Web Consortium, documentation of modern design suites as Adobe and Microsoft Office, and the bug reports of Apache's suite. Khatibi quotes the Orientalist scholar Jean-Marie Domenach as saying "don't you think it is troubling that all these questions that arise in the Arab world are asked by us, and not the Arabs themselves?"[18] Digital Orientalism extends Domenach's sentiment by refusing to document Arabic digital writing, address its bugs, and by normalizing broken Arabic writing.

In *Decadent Orientalisms*, David Fieni posits that modern Orientalism has kept its "normative, rational, and positivist iterations"[19] and survives off of the *decadence* of what I contend to be the digital age. Fieni's decadence can be found throughout my texts as well: in the half-completed tables, in the syntactical indifference of the bug reports, and in the endless bugs and dilapidation of Arabic digital writing. Fieni claims the "plurality"[20] of Orientalism to be a defining feature of modern Orientalism. His title evokes an important detail of digital Orientalism: its lack of "coherence"[21] in favor of power. Instead, the "war machine"[22] of digital Orientalism is an amalgamation of endless production and example. While Fieni describes a "hegemonic armature of statements, positions, policies, and styles that are only beholden to the volatile and arbitrary dictates of imperial modes of sovereignty,"[23] I see a very different kind of machine. This is not to say that Fieni has painted a picture that doesn't exist; in fact my suggestion of plurality reaffirms and builds on his foundational analysis of modern Orientalisms. What I see is a war machine constituted instead by error messages, disappearing and reappearing text,

[18] Khatibi, Orientalism, 79
[19] Fieni, *Decadent Orientalisms,* 6
[20] Fieni, *Decadent*, 7
[21] Fieni, *Decadent*, 7
[22] Fieni, *Decadent*, 7
[23] Fieni, *Decadent*, 7

English-only documentation and code, and weaponized, normalized indifference. I am proposing a shift from the hyper-political readings of Fieni to the (meta-)hyper-digital: a *digital revelation* that realizes not only the political and economic intensity of even a single line of code, but also its possibilities for usurping, defining, and destroying the modern subject(s).

Unlike Orientalism, digital Orientalism does not obsess over identifying the figure of the Arab. Whereas in (physical) Orientalism, the Arabs and their wants, desires, fantasies, political beliefs, and lack were the central obsession, in digital Orientalism the Arab has been divided. As El-Ariss describes, the "individual" has become the "Deleuzian *dividiual,*" split into "masses, samples, data, markets, or '*banks.*'"[24] By this splitting, the figure of the Arab is no longer of concern to digital Orientalism. Instead, digital Orientalism is more concerned with *mining* the new Arabic *dividual* while simultaneously maintaining the primacy of Latin code. The digital age has also shifted the scene of the Orientalist encounter. Instead of a postgraduate study in Morocco or an invasion of Egypt or a tour of Syria, digital Orientalism's encounters follow the structure of digital writing: near-instantaneous, occurring in cyberspace, serendipitous, and positivist. By changing the site of the encounter, the anxieties of Orientalism have begun to overtake its fantasies.

There are two figures which emerge throughout my thesis: the diegetic and non-diegetic writers. The difference is a matter of *existence in a narrative*: the diegetic digital writer writes within the narrative of digital writing. The diegetic digital writer typically confines themselves to word processors, blog posts, comments, or any other abstracted code. On the other hand, the non-diegetic digital writer writes *at* and *around* the scene of digital writing: they modify and inspect code, they write code of their

[24] El-Ariss, *Leaks,* 8

own, they always consider the ever-present context of code which produces and runs the applications they use. These two writers also go by "user" and "developer." By using these terms I declare that there is a *scene of digital writing* as opposed to some amorphous, indiscernible, ever-expanding universe.

It is in and around this scene that I write. As I sit here (and many times I have stood), as I type into the off-white pixels of Google Docs hoping to convey these thoughts, I watch the autosave indicator capture every character I offer. This project is an offering, a search for penance, a *re*pentance, to a structure which I have lived in my entire life. In this thesis, I explore my own writing and what it means for Middle Eastern Studies to write in the digital age.

Chapter 1:
Digital Arabic's Game of Life

In May of 2014, Ramsey Nasser sat for an "Artist's Notebook" feature for Marina Galperina from ANIMALNEWYORK.[25] Nasser introduced a project developed during his residency at Eyebeam, a warehouse in Brooklyn dedicated to being a "space for experimentation" for artists and coders. His project is *Qlb*—قلب, which Nasser pronounces as "*Alb*"—a newly christened coding language "built entirely on Arabic." The language is both writing and code: *Qlb* is a double-edged sword, functioning as both a "conceptual art piece" and a Turing-complete coding language.

Qlb was developed, Nasser says in his interview, *"to get a sense of what Arabic code might even look like."* This was the problem he was facing: code is not in Arabic. Code is only written in Latin scripts. Actual memory is in Latin script. Computer scientists are taught that memory is a series of binary values: 1 or 0. But these numbers belong to a larger system. Considering them in the utilitarian sense as independent numbers encapsulates the failure of understanding which plagues writing on digital code. Computers expect an order: left to right, and discretized characters. Moreover, computers do not simply expect, they often *require*. This required order is how a list of binary numbers (a *string*) propagates a Latin-centric digital age, and it is how this string escapes the hopes of theorists who see an opportunity for democratization in technology.

[25] Marina Galperina, "Artist's Notebook: Ramsey Nasser," http://animalnewyork.com/artists-notebook-ramsey-nasser/

Qlb's description of itself in the online code editor:

```
قلب: لغة برمجة - مترجم ٠،١،٤
رمزي ناصر ٢٠١٢

أمثلة - النجدة - ما هذا؟

>>> (ما-هذا؟)
قلب هي لغة برمجة هدفها إستكشاف
العلاقة بين الثقافة البشرية
والعلوم الحاسوبية، خصوصاً علاقة
الأبجدية اللاتينية بلغات البرمجة
المختلفة، للمزيد من المعلومات
زوروا الموقع قلب،موقع.

تم تصميم وتنفيذ قلب بدعم المركز
الفني التكنولوجي 'ايبيم'
في نيو يورك، حين كان رمزي ناصر
فنان مقيم سنة ٢٠١٢،
<<<
```

Good coding practice (this particular practice is known as error handling) makes sure a computer doesn't combust upon receiving a 1 instead of a 0. There is no breakdown of hardware. But feeding a computer anything but what it expects causes a *breakdown of appearances*: text displays improperly or not at all. Arabic suffers from these breakdowns, as surrounding Latin text is bled into it: letters and characters swap, flip, and teleport. Quotation marks lie alone, brackets and equal signs (which are two central characters in coding syntax) are misplaced or not placed at all. What makes this particularly unique is how readily available these failures are whenever Arabic occurs. On GitHub,

which is an industry-standard online code repository, قلب's name is "---." Such oddities and confusions exist everywhere in Arabic digital writing. GitHub's inability to write Nasser's repository's name provides one poignant example of the breakdowns of Orientalist digital writing when faced with Arabic.

Qlb is a theoretical project which displays the deconstruction of language at the hands of code, but these anomalies are an everyday occurrence for users of the Arabic internet. This chapter is about the writing which falls apart, and the writing which breaks others. By viewing coders as a novel form of the writer, we engage with two interconnected scenes of writing: behind-the-scenes code and painfully visible text. It is an introduction to viewing code as a novel type of writing which needs a novel theoretical approach informed by past traditions of analysis.

This chapter introduces writing's novel breakdowns. In order to make a proper introduction, I consider the aspects of writing which reveal themselves when ink becomes pixel. On the new digital page, writing takes on new dimensions as different languages run up against each other, and are found where they are not expected or wanted. Digital writing—this is what I am calling this new form of chaotic, specific, stochastic, dynamic, and electronic writing—is a perpetual identity crisis. When broken down into its interconnected aspects, as international organizations such as the World Wide Web Consortium attempt to do, digital writing reveals itself as a conglomerate of separate languages all rolled into one. In this multiplicity, a pecking order establishes itself through compatibility and familiarity.

If global organizations sift through the problems of modern Arabic typography, students trudge. In class assignments and papers, text editors feel like a force of resistance. In my experience as a

student, submitting or producing a presentably formatted homework assignment is nigh impossible. Moreover, the inability or failure to add a single letter can change an entire sentence. In Arabic, the failure to add agreement particles changes the meaning of the sentence. If a student incorrectly writes "هي طالب جديد," their failure to use a ة clouds intended meaning: did they mean "She is a new student" or "He is a new student"? This fragile system also exists in English: the addition or omission of a comma changes "I want to eat, you?" to "I want to eat you?" The key difference, however, between the two languages is decades of care and support for digital English, and a history of neglect for digital Arabic.

In the last paragraph, it took me several minutes in order to include the in-line example of Arabic. Not for a lack of understanding of Arabic or an inability to switch languages, but rather for the woes and bugs of my word processor. I am currently writing on Google Docs, a very common processor used at nearly every major academic institution, and upon positioning my cursor, the processor would deposit the quotation mark nowhere near the blinking line. At this juncture of English and Arabic text, the underlying code of Google Docs failed. This is the central conflict of digital writing in Arabic: the struggling writer and the indifferent coder. This conflict does not just emerge when only Arabic is present: it becomes especially obvious when Arabic is combined with other languages, particularly those of Latin script. The two roles of writer and coder will be complicated, blended, and reversed throughout this chapter and larger theoretical project.

In order to investigate this new scene of digital writing, I will examine two of its aspects in the relative cases of Arabic and Latin/English scripts. Combining both Arabic and English into the same line, much less the same page of text, causes significant problems for digital writing. This first aspect

emerges when combining text which reads from left to right with that which reads right to left, and is known as "bi-directional text." All major web browsers (modern examples of a "browser" includes Firefox, Chrome, and Safari) use the "Unicode Bidirectional Algorithm:" a "set of rules" which browsers use to "produce the correct order [of text] at the time of display." This algorithm is necessary because of the differences in order: text, specifically Arabic text, is stored in memory in a different order than it is displayed.

The second emergent aspect of digital writing is text justification: the practice of evenly distributing text among the lines of a paragraph. All one has to do is glance at the current paragraph to see this effect: each line is approximately the same length (in fact they are close to each other as is optimally possible). Varying line length to minimize the so-called "badness" of a paragraph is done in English by varying spacing between words and letters as well as faithful use of hyphenation. While varying inter- and intra-word spacing is the most frequently used method of Arabic text justification, hyphenation is not an option. Instead, there are two methods drawn from Arabic Calligraphy for intra-word variation: ligatures and use of the kashida.

I investigate both aspects at the edges of their reliability. Each of these parts of digital writing are incredibly specific and technical. Specific in what they accomplish: the combination of different directions and the fair alignment of text in a paragraph. Technical in how they are implemented and realized: as code—sometimes accessible and sometimes inaccessible to me—with defined operations, functions, and language. This language is English without exception in mainstream software. Any code which performs justification, alignment, or any other operation on Arabic text is written in English.

This discrepancy changes this chapter from a review of popular software to a documentation of systemic bias pervading digital writing.

Moving through the common media of digital text, I search for the moments in which English code inexplicably fails Arabic text. Word processors, such as Google Docs, Microsoft Word, or LibreOffice, deny Arabic text the attention necessary to produce something professional and publishable. Any combination of Arabic and Latin script requires an undefined amount of guess-and-check editing, cutting and pasting, and highlighting in order to cobble together a legible example. Even more complicated software, such as the Adobe suite, have incomplete and incorrect implementations of Arabic text. Just as jarring as the failure of the code of modern software is the lack of documentation or public information on Arabic text published for their own tools. We will see this later, especially in the context of Microsoft Word and Adobe suite.

Green, Orange, Question Mark

The year is 1994. It is five years after Tim Berners-Lee's original proposal at CERN, "Information Management: A Proposal,"[26] which introduced the modern conception of the internet, what Berners-Lee dubbed the "World Wide Web." In order to standardize the ever-growing internet, the World Wide Web Consortium (W3C) is incorporated. The Consortium describes itself as "an international community where Member organizations, full-time staff, and the public work together to develop Web standards."[27] These standards are the tools and languages the web is built on: languages

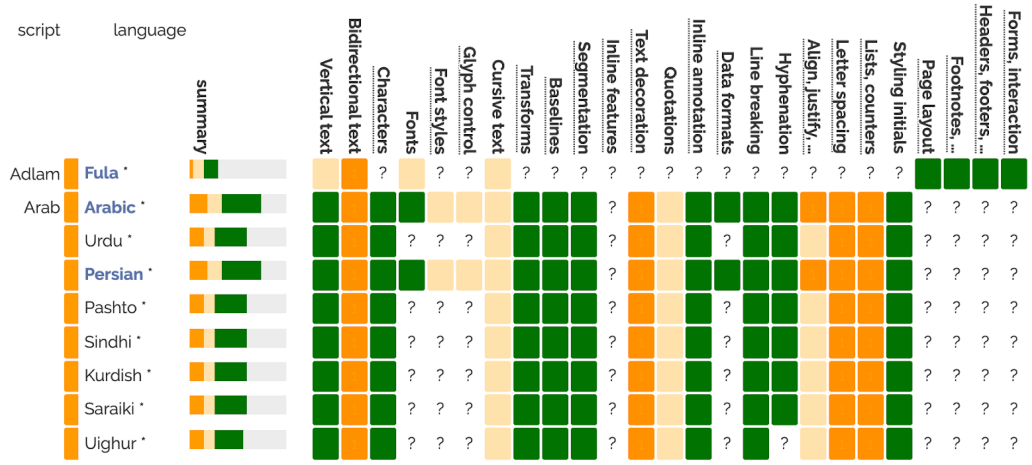
[26] Tim Berners-Lee, "Information Management: A Proposal," https://www.w3.org/History/1989/proposal.html
[27] "About W3C," https://www.w3.org/Consortium/

like HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) which define the "structure" and "(visual and aural) layout"[28] of web pages, respectively.

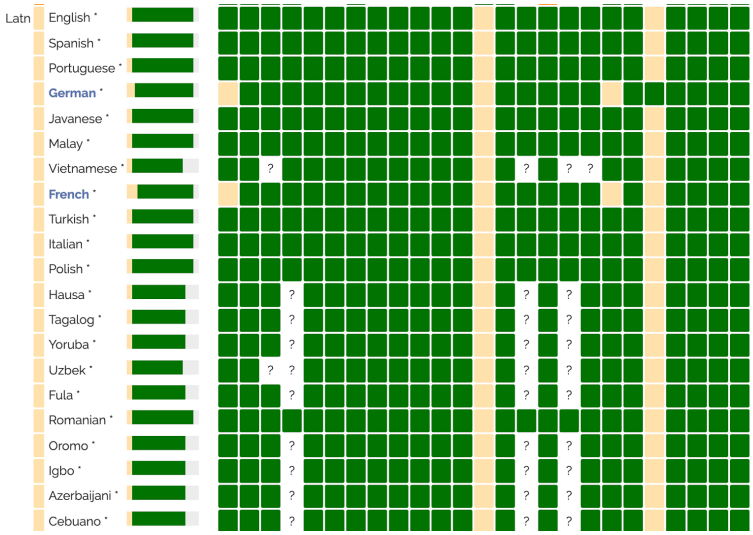
W3C has a child organization, W3C Internationalization (W3C i18n), which commits itself to the mission of "making the World Wide Web worldwide!" W3C i18n publishes reports classifying how well different languages work on the web. The "Language Matrix"[29] is a large chart of green, beige, orange, and red which describes the status for 25 separate features ranging from "bi-directional text" to "text decoration."

[28] "HTML & CSS," https://www.w3.org/standards/webdesign/htmlcss
[29] "Language Matrix," https://www.w3.org/International/typography/gap-analysis/language-matrix.html

Arabic Support



Latin Support



Staring at this colored chart, the enormous gap between Arabic and Latin scripts on the internet makes itself known. The matrix separates the languages by script type, with Arabic at the top just after 'Adlam.' Opening the page, I am greeted by a mess of orange, green, beige and question marks. These floating questions indicate fields which "still need investigation": for Arabic language alone (not counting languages such as Persian, Kurdish, and Uighur), there isn't enough research to

understand a fifth of the possible features. A quick scroll down to the Latin scripts makes my heart drop: the difference is thrown in your face with the full squares of green (with the exception of the "Quotations" column, which is marked as "needs work for advanced publishing"). This simple stark difference in color refutes all on its own the previous hopes of theorists for the internet.

In this visceral difference of color shade, we contact a history of development and imperialistic propagation of programming languages. The accusation of "imperialism" is a strong one. Furthermore, it is useful only so far as to tie this modern phenomenon to an economic history: one of subjugation, of blaming, and of re-invigorated identity. However, locating programming languages in the same place as language is a mistake. Programming languages fundamentally re-write the Lacanian understanding of signified and signifier applied and reapplied to language.[30] This rewriting happens in the manifestation of programming languages as *code*. This code then performs two functions in relation to the signifying chain of writing and language: it makes that chain explicit in a novel way, and undoes the chain by relocating the scene of the signified as electricity and melted earth forged into hardware.

Disoriented Text: Arabic and Digital Writing

In order to determine text direction, a web page or word processor will look at the *base direction*. This base direction is the default direction of text in the "phrase, paragraph, or block that contains it."[31] Containers are how webpages organize text, splitting them up into sections with properties and

[30] Jacques Lacan, "The Freudian Thing," in *Ecrits* (New York: W.W. Norton & Company, 2006), 334-363.
[31] "Mirrored characters," https://www.w3.org/International/articles/inline-bidi-markup/uba-basics#mirroring

settings which are made explicit, yet hidden. These settings hide in the elements of a webpage. These elements are a structure of trees which divide and subdivide information. Base direction can be set "explicitly" by setting an attribute ("dir") in an HTML element, else the direction "is inherited from the default direction of the document, which is left to right." Regardless of the default, the base direction can be set to RTL (right-to-left) or to LTR (left-to-right), and the characters are ordered as such. Despite the base direction, different languages will have different directions. We also have to deal with those characters which are not of a specific language: what are referred to as "weak" characters. These are the numbers, quotation marks, the commas, the spaces: punctuation and the traces of grammar. If punctuation is between two words of the same ordering, they are assigned the same ordering as the words. However, if punctuation is between two words of different ordering (take for example quotation marks around 'عربي,' for which, in order to put the quotation mark between "عربي" and "around," my cursor must be to the right of ع; after moving my cursor through the word from left to right, the cursor moves to the right of ع), the punctuation is assigned to the base direction of the container. Some characters are referred to as "mirrored characters"[32]: characters which change their shape depending on the base direction. Take the following three lines:

a < b < c؛

ا >

ب <ا

In these three lines, both characters "<" and ">" appear on the page. However, I only typed the character "<" on my keyboard. In digital writing, text separates itself from the writer more drastically

[32] "Mirrored characters," https://www.w3.org/International/articles/inline-bidi-markup/uba-basics#mirroring

than ever before. Writing steals away from me, literally morphing into words I didn't write. Or perhaps it is more accurate to say writing has been stolen: the word processor changes writing depending on the text's relative position to language. Google Docs is *closed source*: the source code which has intervened at this scene of digital writing is inaccessible to me and encrypted. I am not able to say exactly what happened: I cannot know. I can make educated guesses about text direction, lookup tables, and conditional statements. But despite these guesses I am forever separated from my writing. Two writers—the closed-source code and the confused human—encounter at the scene of digital writing. In practice, there is no such thing as the isolated three lines: this is how Arabic digital writing behaves to its authors; with this example I am characterizing modern Arabic in the digital scene. Arabic text is shown a reflection in the pixels of our screen which audits it as misshapen and confused. Memory and hex values are a constant identity which is accessible, yet encrypted and protected. Despite this, we are presented with an image of writing very different from memory: confusion marked best by a فاصله منقوطه—an Arabic semicolon—brushing up against a "c." The first line shows how the character behaves as expected in Latin script. However, in the second line, the display of the text causes it to flip around on its own imaginary axis as it waits for the next character to be written. In the third line, with the introduction of the "ب," the character rights itself by flipping once again. In addition to text changing *post writing*, the centering of said text is incorrect: the third line is off center.

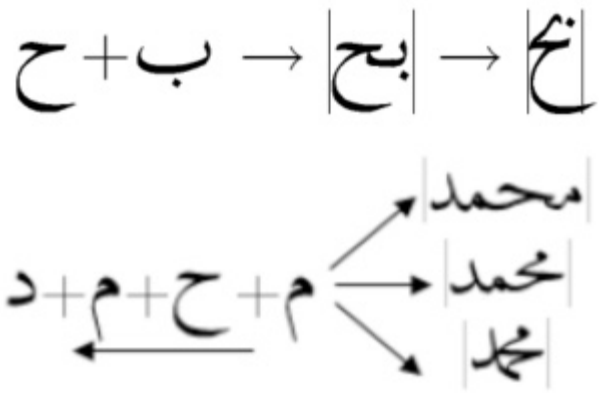
There are two concepts discussed earlier which warrant further investigation: the *default* and the *container*. This concept realizes itself in neocolonialist and semiotic theory as unattainable: the purpose of the default or signified is to create a *trace*, that illusory center which postmodernism spends so much of its effort to relocate and retame. However, in this digital writing, there is no symbolizing

chain which obfuscates and realizes the default. By contrast, this writing is deliberately honest; documentation and specification dissolve the detective work necessary to understand and deconstruct that illusory center. Instead of ending at the center, our investigative work begins there.

Arabic Text Justification: A History of Whitespace

Ligatures, from the Latin *ligare* meaning "to bind,"[33] are the combination of two or more characters in order to form a single character. An example of this in Latin script is the combination of *et* into &. In Latin script, the list of ligatures is minimal and specific: only a few cases are even accepted, and even then they are hardly used in writing. On the other hand, reading Arabic requires deciphering ligatures on a word-to-word basis. Combinations of letters are only limited by letter order and legibility. According to a paper written by Mohamed Benatia published from University Cadi Ayyad's Computer Science Department and titled "Arabic Text Justification," there are three levels of ligatures, differentiated by their frequency and by extension their legibility. The three levels are "mandatory substitutions," then some "aesthetic ligatures," and then finally extended use of the aesthetic ligatures. These three levels of ligatures will change the same word into different lengths. Here is a visual representation of ligatures and variations of the name "Mohammed" at each level:

[33] "Ligature," https://www.etymonline.com/word/ligature#etymonline_v_9501



[34]

Currently, ligatures are done by fonts, as opposed by HTML and CSS formatting and styling. By placing ligatures in control of the font, justification is not done by dynamically altering the amount of ligatures. Instead, ligatures simply alter the base width of a word, which is then handled dynamically by the word processor or web page when justifying text. However, implementations of ligature justification are flawed. From what I have seen in major word processors, including the fonts for Google Docs, ligature justification employs an "all or nothing" approach. Either all ligatures are employed, giving a convoluted and sometimes illegible appearance to Arabic writing. So, we are faced with a new critique: it isn't that Arabic ligature justification is absent or ignored. Instead, Arabic ligatures lack the adaptability of Latin scripts, instead providing an awkward and stiff manipulation of writing.

The second method of justification is the use of the kashida, which is an extension of the length of specific characters, either in the middle or at the end of words. This extension is done by curved strokes. There are several rules for applying the kashida, including word length and which character is used. There are also style guides which are recommended, such as the retroactive addition

[34] Image from "Text alignment and justification," https://w3c.github.io/alreq/#h_justification

of kashida when "ه" is the final letter in a word.[35] Some style guides, adopted from the practice of Arabic calligraphy, restrict kashida use to once per line, except when using the *bismillah*.[36] Titus Nemeth recently published a review of justification for Arabic language in the Journal of Electronic Publishing.[37] In his paper, Nemeth describes the history of justification strategies for Arabic before providing a review of their availability in modern software. As Nemeth describes, the kashida is often confused with its Unicode counterpart, the tatweel (U+640). While the kashida originated in Arabic calligraphy, the tatweel is a different story. It originated as a tool used by "European type-makers and printers"[38] as they began to encounter Arabic texts. As Nemeth describes the justification practices, a dichotomy between European justification practices and justification in Middle Eastern printing houses emerges.

Nemeth's research describes how one of the foremost printing houses in Europe, the Medici Oriental Press, published many Arabic texts using justification practices heavily reliant on the tatweel. This was despite a new font which "renown French punch cutter Robert Granjon" had recently designed that "included swash variants and elongated letter forms" and served to provide more natural-looking options for justification. Overuse of the tatweel by the Medici Oriental Press led to "geometric linearity found nowhere else in type" and "blank spaces without apparent function." Thus, we begin to see a history of the uncanny in Arabic typography in a Western setting. The uncanny bugs, flipped letters, and off-putting white space has a lineage which traces back to Western institutions' early encounters with the Arab world. Although the odd bugs have shown themselves in digital typography,

[35] Benatia, Elyaakoubi, and Lazrek, "Arabic Text Justification"
[36] Benatia, Elyaakoubi, and Lazrek, "Arabic Text Justification"
[37] Titus Nemeth, "On Arabic Justification"
[38] Titus Nemeth, "On Arabic Justification"

metastasizing as failed documentation and unimplemented features, the uncertainty of Arabic font and text in Western media did not originate at the introduction of computers. But the situation has not improved much.

Use of the kashida is specific to fonts rather than being dynamically assigned when needed by the word processor or webpage. To make matters worse, it seems that no built-in font of Google Docs, the processor I am using to write this paper, from the 29 Arabic font families available, supports the kashida or tatweel. This lack of support extends beyond this word processor. After giving histories of Arabic justification, Nemeth moves on to his review of the availability of strategies in word processors and browsers, starting with the Adobe suite. The documentation for Arabic text justification he cites (which is still on Adobe's website as of the publishing of this thesis)[39] only mentions the kashida (and proceeds to use "arabic" [*sic*] instead of "Arabic"). As he jumps into the actual program, he finds that there are in fact *four* options for kashida types, and *zero* of those options is mentioned in Adobe's documentation. Furthermore, there is a "kashida width" field which takes in a number which is also not documented but, according to Nemeth's experimentation, does "indeed change the length of the automatically inserted *tatweel-like* strokes" (my emphasis). While that field proves useful, choosing any of the kashida types has no noticeable effect and, in a realization which would be humorous if it were not inescapable, "the meaning of the [kashida] options remains incomprehensible for [Nemeth]."[40] He does describe "Justification Alternates (Naksh)" as a "redeeming feature."[41] Instead of applying all ligatures and kashidas possible in a font, it applies them when needed to justify line length. However,

[39] "Arabic and Hebrew features in InDesign." https://helpx.adobe.com/indesign/using/arabic-hebrew.html
[40] Titus Nemeth, "On Arabic Justification"
[41] Titus Nemeth, "On Arabic Justification"

he describes a bug in all justification algorithms for Adobe; the bug inserts tatweels whenever a ligature is used which does not join two characters along the baseline. These hovering half-baked tatweels that end and begin without warning are perhaps representative of contemporary (and antiquated Orientalist) attitudes towards Arabic text. Modern design suites such as Adobe lack the understanding or care to even tell what Arabic text should look like to *meet legibility standards*, much less implement fixes.

Whereas Adobe is used for professional design, most people use word processors to write and eventually publish text. These word processors include Google Docs, Microsoft Word, Apple's Pages, and LibreOffice. Nemeth reviews the available Arabic justification features of all of these except for Google Docs. Throughout this chapter, I have been referencing Google Docs as the word processor I am using, and have already showcased several oddities and bugs in its handling of Arabic. There is no apparent autodetection for languages or scripts, rather a default setting for language and therefore text direction for Google accounts across all of Google's suite. There are three apparent options for justifying text in Google docs: left align, right align, and justified. Left align forces the text against the left margin, and right align vice versa. Meanwhile, the "justified" option presses the text as much as possible against both the left and right margins. From my experimentation (as there is no apparent documentation or help information available for Arabic justification techniques), Arabic justification in Google Docs is limited to changing the width of inter-word whitespace. As I stated earlier, no font packaged with Google Docs contains the kashida, and only a select few include ligatures. However, these ligatures are *all or nothing*: if the font contains ligatures, it will use them in every case. This incomplete implementation leads to three possible text styles: extremely stylized and difficult to read,

extremely stylized and very easy to read, or no stylization and easy to read. In all three cases, no effort is made to do what I would call *dynamic justification*, a term I am certain I did not invent, but have not seen anywhere else. Dynamic justification is the ideal scenario, and the "justification alternatives (Naksh)" from the Adobe suite is the closest real-life example.

This static approach to writing and justification pervades most word processors. Such an approach showcases and normalizes an ignorance and indifference for Arabic digital writing. For Microsoft Word, Apple' Pages, and LibreOffice, Nemeth described recurring bugs for each implementation. Microsoft Word contains the same bug as Adobe's Naksh by inserting erroneous and phantom-like tatweels when letters join off of the baseline. Sometimes, "glyphs appear pulled apart,"[42] as whitespace splits some words in two. Apple's Pages, when using in-house Apple fonts, is very close to dynamic Arabic justification. Glyphs are assigned and replaced in order to change the lengths of words as available space expands and contracts. Despite this, bugs were still apparent; in Nemeth's sample text "the two instances of tanwīn marks break the shaping, and there are some unresolved collisions of glyphs."[43] Each time one bug is squashed, another emerges, smaller and harder to diagnose. LibreOffice is able to support more fonts than any other word processor mentioned. Once again, we are met with the same bugs, in the same locations: floating tanwīns and misshapen letters. LibreOffice, being open-source, has a lot more available resources, and even gives the public access to source code. Despite this, Arabic translation of resources is nearly nonexistent. As of Aug 7, 2021, 93% of the LibreOffice GUI (all the tools and buttons in the application) has been translated into Arabic, while a shocking 0.6

[42] Titus Nemeth, "On Arabic Justification"
[43] Titus Nemeth, "On Arabic Justification"

percent of the help pages have been translated.[44] LibreOffice and its source code is documented on the "Apache OpenOffice Documentation Project." Adjacent to the documentation project is a community-run wiki which explains how different systems of OpenOffice (the suite LibreOffice is a part of) work. *No* Arabic option is available for the documentation language, and only the front page of the wiki is in Arabic: any linked pages or actual information is entirely in English.

Any code which performs Arabic justification for OpenOffice is written in English. Both comments and code are in English. This paper is in English. OpenOffice uses an algorithm for kashida insertion[45] that, as pointed out by Nemeth,[46] appears to be the same as a 2005 implementation[47] by Microsoft, which is not in use since its switch from Edge to Chrome.[48] It uses a "connection priority system"[49] which assigns a priority value from 1 to 7 for every location for each word, and inserts a kashida at the location with the highest priority and nearest the end of the word. In the ideal solution, each intersection between letters is investigated and labeled—a never ending and ongoing procession of processing kept hidden *for the writer's own sake*.

The presumptuousness of digital writing affects the capabilities of the Humanities, specifically Middle Eastern Studies. In a field which was reformed out of Orientalism in Western universities, deconstructive revision and institutional introspection are central to its academic process. An abstracted digital writing prevents all of that. My argument here is reliant upon a cellular

---

[44] "Translation Statistics," https://wiki.openoffice.org/wiki/Translation_Statistics
[45] OpenOffice Source Code, https://github.com/mirror/openoffice/blob/ac58ea25d9ea6e57181d6047264340cdc75de79a/main/sw/source/core/text/porlay.cxx#L1145
[46] Titus Nemeth, "On Arabic Justification"
[47] Paul Nelson, "Justifying Text"
[48] Titus Nemeth, "On Arabic Justification"
[49] Paul Nelson, "Justifying Text"

understanding of academia and of writing: it is a composition of micro-parts, the structure of which defines the structure of the whole. This understanding of institutions is appropriate for a scene of writing which is really an abstracted series of 1's and 0's, and it is how "priority values" and encrypted algorithms change the face of Middle Eastern Studies.

Arabic justification online shares the same problems as word processors, but the scene is completely different. Instead of a personal encounter with the digital, it is a collective realization of Arabic's forced oddity. The single, solitary writer originally confined to the offline word processor is multiplied across a general public on the internet. On the other side of the screen, the coder multiplies across countless websites and tech companies. Despite this multiplicity, failures are still the norm. No major browser[50] has implemented justification which is both bugfree and doesn't solely rely on white space extension. Given the prevalence of justification techniques used by modern Arabic printers and classical Arabic calligraphers, solely using white space extension feels so ridiculously lazy that it is easy to understand why Arabic has so many problems when used digitally. It is not the Arabic language, the Arabic scholars, or Arabic printers which are to blame for its constant misuse and Frankenstein-like amalgamation. But then, is it fair to blame the equally over-sutured amalgamation of coders and computer-scientists which wrote insufficient code?

The question of justification introduces a novel understanding of typography. Instead of obsessing on the fixed order and arrangement of letters, our focus has shifted to how that order is created and who (or what) creates it. This reorientation of theory from the physically static to the electronically dynamic is a defining feature of digital deconstruction. Instead of searching for or

[50] Titus Nemeth, "On Arabic Justification"

evoking the dynamism of language, we begin at it. And in this scene of messy theory, we are given messy examples: illegible Arabic, odd floating tatweels, and incomplete algorithms. These algorithms and bugs are embedded, not just in the text itself but in an underlying code. This code is novel in the Humanities for its legibility and malleability. And it's written entirely in English.

Two Writers and a Time Loop

As we have seen, software developers and designers have made a shoddy place for Arabic in digital writing. The roof leaks with software bugs, the windows don't close all the way, and in a certain part of the house everything almost works. Whether or not software developers and designers are the culprit (or that there even is a culprit) is as dubious as the reliability of digital Arabic. In digital writing, there are two writers. Adopting vocabulary from film studies, there is the *diegetic* and *non-diegetic*. The diegetic writer writes information, words, and language in/into a word processor or text editor. I am currently a diegetic writer. Then there is the non-diegetic writer who has written the code, the framework, the bugs, and created the system which the diegetic writer writes in(to). The plurality of the writer comes into question with this framework. *The* diegetic writer spans many cultures, many languages, many scripts,and many people. How many people are responsible for a single Wikipedia page? The non-diegetic writer is inequitably distributed among several, sometimes hundreds, sometimes thousands (such as at Microsoft or Google). Often this writer is formalized and bureaucratized by company and project organization, but can also be spread out across open-source projects. These projects can range from free-for-alls to even more constricted than company-driven

projects. Besides the *scene* of these two (+n) writers, *language* differentiates them from one another, and in the case of Arabic, *separates* them.

In an *Afikra Conversation,* Nasser describes a "time loop"[51] every other month since he has published *Qlb*. During this loop, he is contacted by an Arabic-speaking developer who would love to build upon *Qlb* to create an app or program in a "coherent all Arabic programming experience."[52] This dream, Nasser says, is impossible. It isn't that an all-Arabic programming language isn't possible: the base functionality has all been proven to work, as *Qlb* shows. However, what *Qlb* shows isn't enough. As the "examples" or "أمثلة" of *Qlb* show, basic algorithms are possible using isolated code. Conway's *Game of Life* is implemented, a somewhat complicated program that requires some implementation of computer graphics.

[51] Nasser, "Afkira Conversations"
[52] Nasser, "Afkira Conversations"

A portion of Conway's Game of Life code in Qlb:

```
قـلـب: لـغة بـرمجة - مـترجم ٤،١،٠
رمـــــــزي نـــــاصـــــر ٢٠١٢

أمثلة - الـــنجـــدة - مـــا هـــذا؟

>>> (أمثلة)
سـهـل: مرحبا يا عالم.
متوسط: عدد فيبوناتشي
متقدم: لعبة الحياة لكونواي
... (حدد الكون-المقبل (لامدا (كون)
...  (إفعل
...  (حدد خلية (لامدا (س ص)
...  (إذا (لائحة؟ (عنصر ص كون))
...  (عنصر س (عنصر ص كون))
...  ٠)))
...  (حدد جيران (لامدا (س ص)
...  (جمع (خلية (طرح س ١) (طرح ص ١))
...  (خلية (طرح س ١) ص)
...  (خلية (طرح س ١) (جمع ص ١))
...  (خلية س (طرح ص ١))
...  (خلية س (جمع ص ١))
...  (خلية (جمع س ١) (طرح ص ١))
...  (خلية (جمع س ١) ص)
...  (خلية (جمع س ١) (جمع ص ١)))))
...  (حدد خلية-المقبلة (لامدا (س ص)
...  (إفعل
...  (حدد خ (خلية س ص))
```

The Rendered Game:

But in his interview, Nasser points out that in order to write any substantial code, the work of other programmers are used in the form of *libraries*. Their function and library names are not in Arabic: all standardized libraries are in English. Keywords (such as *Qlb's* "قول" meaning "say") in modern languages are all in English and must be typed in English. You are "forced," Nasser says, "into a linguistic collaboration with people you have never met."[53] These "people" are the non-diegetic writer, embodied in countless bodies. This writer splits again in our conversation, into those who are writing and those who have written: those who are creating new code and libraries, and those who have written the libraries new code will be based upon. In the case of Arabic digital writing, in *our* case, these split versions of the same writer are diverging: one unknowingly furthering a hegemony of Latin code, and the other desperately trying to revert the unpatchable.

The disconnect between Arabic and Code has only grown, but in the field of digital writing there will always be those working to correct the gaps. In this techno-phrenia of writing, re-writing, and combining code, the non-diegetic digital writer splits and then folds back in on itself by appropriating and injecting other's code, preventing and creating opportunities for an Arabic digital writing. The efforts to introduce Arabic coding into the consciousness of English code have pushed to the surface the systematic problems of Arabic/English interaction.

[53] Nasser, "Afkira Conversations"

Chapter 2:
The Turtle and The Portal

Speaking in 1859 to a crowd of Arabic intellectuals, Butrus al-Bustani delivers a series of laments refuted with hopes. He laments failures in the "culture of the Arabs today":[54] they are content with their own stagnation, and this stagnation is in a "state of decay." Despite this state al-Bustani is hopeful, for "the means of acquiring culture are many." The printing presses which spring up throughout al-Bustani's world offer a glimmer of hope, if only they would "take their work seriously." For al-Bustani, not all writing is literature (*adab*). Printing presses, and in particular *steam* printing presses, "that great power . . . which the American missionaries themselves employ in their press" are a question of "knowledge and civilization (*tamaddun*)."

Thus, for one of the most important *Nahda* thinkers and translators, knowledge and civilization are a matter of *writing*, and writing is a matter of *technology.* At the juncture of technology and writing we see a tradition which would eventually explode into *digital writing*. In Chapter 1, we began at that explosion, with the invention of the internet and the *physically digital*. That chapter formulated the consequences of hardware's structure on software's ability to handle, integrate, and expand Arabic. In this chapter, we are at the site of a different explosion, mirroring the first in both its site and its consequences for the Arabic language and for the "culture of the Arabs today." Hardware takes on a parallel yet separate meaning: steam, iron, wood, and ink. The *Nahda* has been framed in many contexts, as *trials*, as an *encounter*, and as a *reaction*. In my context, it is a series of translations: a

[54] Butrus, Al-Bustani, "The Culture of Arabs Today"

translation of a translation, a re-translation, a correction of a translation. This embodiment and re-embodiment of translation puts us far from the linear conceptualization of origin or transmission.

However, the conception of translation as a linear and directed conversion pervaded the *Nahda*, especially in al-Bustani's writing. For al-Bustani, an "Arabic work" or "translation" was a text free of "foreign terms." al-Bustani's *Nahda* shares the same hopes of Nasser's *Qlb*: specifically *Arabic*, or rather, non-Latin, *writing*. Arabic culture in al-Bustani's lecture faces the same problem as Arabic code: a technological and economic dominance by the West which is so extensive as to become standardization. Like the standardization of W3C discussed in Chapter 1, the West's translation and transformation of Arabic writing reeks of mistakes.

During the lecture, al-Bustani brings up the printing presses of Europe and America in the context of conservation. He admits to his crowd: "Had not these presses been concerned with Arabic, no trace would have remained of many precious Arabic works." However, these conserved works have morphed and changed during their stay in the West. When these texts "return to us after their long absence printed with beautiful lettering," al-Bustani is no longer "able to say that these books are completely accurate and grammatically correct." The same problems which concerned digital writing in Chapter 1 have returned to trouble a *mechanical writing* of the printing presses of the *Nahda*. In this chapter, troubling is an act which goes both ways: the code and standards of our era are torn apart by the writing of the *Nahda*. Grammatical and typographic corruption of digital writing is also mirrored in the *Nahda*: Arabic translators began taking matters into their own hands. This impermanent shift in the scene of Arabic translation *is* the *Nahda.*

If the scene of Arabic coding is American University of Beirut graduates, Brooklyn maker-spaces, and broken Github links, the scene of Arabic translation was American Christian missionaries and their newly christened presses. Distinctly Arabic presses existed and were just as prolific as the Christian presses: most importantly the Bulaq press in Cairo, which was originally the press Napoleon brought during his infamous invasion of the country.[55] During his 1859 lecture al-Bustani mentions many different distinctly Arabic publications and presses: the "Syrian press," and "many other Arabic presses . . . in Qozhayha, El-Choueir, Aleppo, Jerusalem, and the Maghreb." However, it was at the Christian schools with their presses that the major writers of the *Nahda* were taught. Al-Bustani himself was educated at the Maronite school 'Ayn Waraqa.[56] Many more *Nahda* thinkers were employed as translators at these Christian presses. Ya'qub Sarruf, As'ad Daghir, Nasif al-Yaziji, Khalil al-Yaziji, and Yusuf al-Asir were all prolific translators and thinkers during the *Nahda* who worked at Chrisitan presses.[57] Finally, at the head of the translation department at the Church Missionary Society (CMS) press in Malta sat Ahmad Faris al-Shidyaq.[58]

Faris al-Shidyaq, later Ahmad after his conversion to Islam, was born in 1805 or 1806.[59] His brother, As'ad al-Shidyaq died in the prison of the Maronite patriarch. After As'ad's imprisonment, Ahmad Faris left Mount Lebanon without saying "farewell" to anyone.[60] He became one of the central figures in the *Nahda*; by producing, writing, editing, publishing, and translating, Ahmad Faris al-Shidyaq dedicated himself to modernizing the Arabic language. His only novel, *al-Saq ala al-Saq fi*

[55] Johnson, *Stranger Fictions,* 36
[56] Stephen Sheehi, *Arab Renaissance,* 3
[57] *Stranger Fictions,* 37
[58] *Stranger Fictions,* 37
[59] Johnson, foreword to *al-Saq,* ix
[60] Sacks, *Iterations of Loss*, 94

ma huwwa al-Fariyaq, was published in 1855 in Paris. *Al-Saq ala al-Saq*, as it is referred to hereafter, became a *modernist* Arab text. In al-Shidyaq's novel, this modernity is one of broken gender, satirization of classical Arabic poetic form, and appendices of corrections of other texts. Al-Shidyaq's writing was uniquely his own, and stood apart from the rest of the *Nahda* in attitude, form, function, and structure. Reading alongside al-Bustani's work contextualizes al-Shidyaq's project as avant-garde writing; a quality it still retains a century and a half later.

As foundational figures of Arab literature, al-Shidyaq and al-Bustani became colleagues dedicated to realizing the *Nahda*. But, as Jeffery Sacks writes in *Iterations of Loss*, if al-Bustani's language is in "relation to form and the body," then al-Shidyaq's writing "falls to pieces."[61] Whereas Butrus al-Bustani wrote in the pursuit and revival of *adab*, al-Shidyaq wrote alongside and around it. In the "pieces" of al-Shidyaq's writing, *synonym* and *meaning* work together and against each other in a dance that eventually fragments the process of translation itself.

The title of al-Shidyaq's novel itself has multiple fragmented translations. The entire title is translated as *Leg over Leg or The Turtle in The Tree concerning The Fariyaq What Manner of Creature He Might Be* by Humphrey Davies.[62] Jeffery Sacks translates the title as *Crossing Leg over Another*,[63] while David Fieni translates the same words as *Crossing Legs over Fariyaq*.[64] Rebecca Johnson translates the title as *Leg over Leg Concerning that Which is Fariyaq*,[65] while Tarek El-Ariss renders it simply as *Leg over Leg.*[66] During the process of translating and re-translating al-Shidyaq's novel, the title itself

[61] Sacks, *Iterations,* 91
[62] *al-Saq*, title page
[63] Sacks, *Iterations*, 96
[64] Fieni, *Decadent Orientalisms,* 53
[65] Johnson, *Stranger Fictions,* 47
[66] El-Ariss, *Trials*, 57

fragments into a collage of order and synonymThe narrator's name—a *naht* or surgery of *Faris* and *al-Shidyaq*—plays with the relationship between writer and character by fragmenting and mosaicking language.

In this chapter, I am involved in a new comparative literature novel in both its approach and its text. I treat code as language, language as writing, and writing as code in a comparative reading between coding languages and the text of *al-Saq ala al-saq*. By engaging with the work of Rebecca Johnson, Tarek El-Ariss, and Jeffery Sacks on al-Shidyaq's writing, I contextualize my theory as a simultaneous approach to reading al-Shidyaq. However, as I engage with code and digital writing as my comparative source, I propose a new temporality of *al-Saq ala al-Saq* and of code that extends far beyond a *Nahda* interaction between Occident and Orient or a contemporary clash of cultural assimilation and techno-economic conflict. In true Shidyaqian fashion, I show these two chronologically disparate sites to be linked as *synonymous*: two words which are the same yet different and whose implications are affected and re-affected by each other.

In all of al-Shidyaq's works, the Arabic and the English translations emerge from each other: a discussion of civilization is also a discussion of *tamadun*, and al-Bustani's translation of that term. When reading al-Shidyaq comparatively, the Arabic is tied to the English and vice versa through substitution, italics, and explanatory parentheticals. While *ma'ana* has a primacy in *Nahda* works, in *al-Saq ala al-Saq*, it is a characteristic of language which is destabilized and satirized like the rest. When engaging with al-Shidyaq, translators are not translating meaning, they are translating amalgamated titles and sutured names; the question of *function* enters the discussion.

This power struggle between meaning and function was one al-Shidyaq himself entered when translating scripture. In her book *Stranger Fictions*, Rebecca Johnson describes a dichotomy between the Christian and Shidyaqian approaches to translation. For Western, Christian printing houses, the translatability of scripture is a question of divine truth. The international project of translating Christian scripture, Johnson explains, purported to have a "future of absolute fungibility guaranteed by the transcendental signifier."[67] Despite the doctrinal confidence of Christian missionaries, al-Shidyaq's frequent use of the synonym in his translations evoked concern, particularly for Samuel Gobat, the head of CMS publications in Malta. According to Gobat, the "chief defect (some may call it an advantage)" is the "desire of the translator to avoid frequent repetitions of the same word, so that sometimes there are four or five so-called synonyms to render the same word in the Original."[68] The concern of Gobat is the same concern of a C compiler: that specific *keywords* need to be repeated, else the "biblical student cannot rely upon the decided meaning of the individual expression."[69] In C, keywords are "words that have special meaning to the C compiler"[70] and therefore "are not available for redefinition."[71] The comparative reading of the two protective anxieties of scripture and code blends the two: code becomes scripture, and scripture becomes code. As we fall into *al-Saq ala al-Saq*, this doubling will have considerable implications for digital writing discussed in this chapter.

This conceptualization of text as code, specifically *scripture as code*, reveals a tradition of translation which prioritizes Latin text in favor of a *divine meaning*. Divinity acts as a proxy for

[67] Johson, *Fictions,* 38
[68] Johnson, *Fictions,* 43
[69] Johnson, *Fictions,* 43
[70] Microsoft, "C Keywords"
[71] cppreference, "C keywords"

Johnson's "transcendental signifier": this signifier re-emerges in the digital writing of code. In the context of this chapter, the divinity of code is found in the C compiler. The compiler, with its Latin expectations and divine definitions, is eventually undone and re-read as a site of confusion, Shidyaqian synonymy, and as a challenger to Latin coding. In doing so, I deconstruct the assumptions of *Latin primacy* and the notion that failure—failure of structure, failure to guard the signifier, failure to keep its secrets hidden, all the "failures" of Latin code alongside Arabic text—of digital text implies a failure of language.

Reading Johnson's work on translation in *Stranger Fictions* prompts a reconsideration of "Latin" text at the foreground of coding, as was described in the previous chapter. The premise of the scholarly and institutional work detailing the dominance of Latin in digital writing is an understanding of translation which Johnson deconstructs in her chapter "Crusoe's Babel, Missionaries Mistakes: Translated Origins of the Arabic Novel." In contrast to how W3C and Ramsey Nasser's project describe the digital relationship between English and Arabic, Latin and non-Latin script, Johnson rejects the model which is a "lateral translation between language A and language B." (Johnson, pg. 65) Instead, we see a "translation of a translation," and translation as an action which happens "alongside" instead of *after*: European texts translated into Arabic are *transformed* and *re-written* in a Shidyaqian transaction of meaning shifted and altered from one synonym to the next, from one leg onto another.

Latin digital writing, and the more specific category of Latin code does not fit neatly into the theoretical frameworks of translation which Johnson develops alongside al-Shidyaq's text. Digital writing has extra dimensions which the physical writing of presses and pens reproduces through tranches of signifying chains. The translator or the author cannot simply work within the dimension

of text and meaning . An attempt to translate a website from English to Arabic is no longer just a question of *ma'ana*, it is a question of form, of orientation, of the *functions of writing* which construct a website, this digital *work*. Digital translation has shifted translation because of the physical object which is being translated.

New dimensions of writing shift the very object of writing itself. *Al-Saq ala al-Saq* is a text which is classified as unclassifiable, as "linguistic treatise";[72] as genre-bending, and as satirizing genre. *Al-Saq ala al-Saq* is a work of genre-translation which bends its texts-as-subjects by excavating their "extra dimensions." For my purposes, *al-Saq ala al-Saq* embodies digital translation. By understanding al-Shidyaq's only novel as digital writing, I read modern digital writing and al-Shidyaq's deconstruction of meaning and translation comparatively. In doing so, I reframe digital writing—and by extension, code—as a Shidyaqian play of genre, function, and language. This reframing returns the favor by introducing a reading of al-Shidyaq not as a corruptor or guardian of language but rather as a *diegetic* digital writer, working behind the scenes of meaning and language in order to construct scenes of writing, rather than intervene at existing sites.

A Second Table: Patching, Botching, Concocting

As in Chapter 1, we begin with a table. In the appendix of *al-Saq al al-Saq*, al-Shidyaq includes tables which span several pages. Each table contains four columns: "Page," "Line," "[Incorrect]" and "[Correct]"[73]. This is a very different table from the color-coded condemnation of the state of Arabic

[72] Johnson, *Stranger Fictions*, 48
[73] *al-Saq*, 5.3

in digital writing. Whereas W3C's table describes a language unable to fit into the techno-grammatical structures and expectations of Latin digital writing, al-Shidyaq's tables are meant to destroy someone's career.

The appendix in question, "In Which are Strung Together The Pearl-Like Errors Made by the Great Masters Among the Teachers of Arabic Languages in the Schools of Paris"[74] begins with a searing accusation of Parisian Arabic grammarians. Al-Shidyaq calls two of these scholars by name, Alexandre Chodźko and Monsieur de Sacy, in order to "delete his name from among those of the shayks of the Islamic countries in their entirety."[75] Both Chodźko and de Sacy have determined for themselves that the "countries of Europe have long been possessed of everything they need for oriental languages."[76] Chodźko frames this possession as institutionalized, as "well-qualified"[77] and that the Parisian "professors" have much to teach the "professors" and "teachers" of the "Persians" and "Arabs."[78]

This self-satisfied confidence did not confine itself to the Orientalist era of Parisian Arabist schools; the same arrogance returns as digital writing spreads. It is a "custom," as al-Shidyaq writes of Chodźko and his "predecessors and professors," to "resort to *patching*, *botching,* and *concocting*"[79] (my italics). Theorizing this "custom" in the context of digital writing does not change the meaning of these terms, but rather connects two different instances of the same event. *Patching* by using scraps of "parasitic" and "randomly pounced upon"[80] knowledge to form a leaky translation filled with holes.

---

[74] *al-Saq,* 5.3
[75] 5.3.10
[76] 5.3.1
[77] 5.3.1
[78] 5.3.1
[79] 5.3.2
[80] 5.3.3

Patches are used to tend to an injury, but this has been a poor job: pus is congealing and blood still flows. Al-Shidyaq's accusations are always bodily. This disgusting meal which these Parisian "professors" have *concocted* smells awful: they cannot manage the "glottal stop with the proper bite—they are already (back) biters enough;"[81] they have "imposed strange meanings" which are "unacceptable to both nature and *taste.*"[82] Al-Shidyaq likens these professors to two men: one who is so in love he became "sick and crazed," while the other convinces him he has "achieved what he wanted from her." His analogy embodies Arabic language as a pined-after woman (in true Shidyaqian fashion) who has rejected this sick man and knows nothing of the drama which has unfolded between the two of them and their crazed bodies. Patching, botching, and concocting all happen on or to *bodies*: bodies of men and women, the bodies of professors and their (unfortunate) students, bodies of knowledge, and in an immediate sense, onto the body of writing.

The obsession of Parisians, modern coders, translators, and all converters explodes all at once into a gross love of the body of Arabic. It is no surprise that when discussing Shidyaq's writing, it takes only a few paragraphs for *love* to surface: the entire text of *al-Saq ala al-Saq* corporealizes desire. Al-Shidyaq's appendix describes a love *of* and *by* substitution. Just as it realizes the failures of Chodźko and de Sacy, the table realizes these lecherous substitutions made by Orientalist lovers. Each table lists different aspects of this substitution. Whereas the first chapter's table color-coded the methods of conversion and translation, this table explicitly counts and accounts the failures of the Orientalist system. Even so, the mistakes of these two translators are not enumerated in full. Al-Shidyaq notes that

[81] 5.3.9
[82] 5.3.4

the mistakes in de Sacy's commentary on the *Maqamat* of Al-Hariri are "too numerous to count."[83] Orientalist commentary becomes an unending chain of substitutions, each one linked to the previous and the next by a love for substitution itself.

Computer scientists know the substitution of correct code for erroneous code as *patching*. Since 1878, the object of patching has been referred to as a *bug*. Bugs are disjoint gaps in the connection between intention and reality of written code. Often, bugs are obvious to their authors: code breaks down at a specific point, stops running when reaching a specific line. Sometimes they are a quick fix: less than 30 seconds of speed-reading symbols and keywords reveals a mistake. These simple mistakes are usually ones of writing. They are the mistakes which al-Shidyaq lists in his appendix: an erroneous character, or a misspelled word. Grammatical mistakes, where "grammar" comes to apply to code as well as Arabic.

It is important to realize the *temporal* characteristics of patching and botching. I can define or characterize botching as a patch done *quickly*, without thorough consideration or research. Botching fixes the problem without any understanding of its origin. Regarding the process in relation to structure, botching is the process of *addition*. The internal structure of the failing conglomerate or text or system is not considered, instead some additional structure is draped around the wound in order to hide, sequester, quarantine, and efface. When Parisian professors botch their translations and works, they seek to perform each of these actions, simply growing the amalgamation of Orientalist failings. To phrase as a question of the body, it is the application of a salve or bandage to a wound. And to phrase

[83] 5.5.1

botching as a question of the body is not simply an explanatory analogy when dealing with al-Shidyaq. *Language is a body* in *al-Saq ala al-Saq*.

In al-Shidyaq's appendix, typos and grammatical errors are not the only bugs in contemporary Orientalist writings. He includes more complex bugs: mistranslation, misinterpretation, and incorrect meter. Al-Shidyaq offers explanations and corrections ranging from just the corrected word to extensive, joke-bespotted discussions of scanning and rhyme. Some of his corrections are not corrections at all, simply laments: "What could be the cause of such contordedness?"[84] and "I have no idea how the good professors think that an adverb can end in *-u*." [85] Other corrections are straightforward critiques: a "hamzah being deleted for the meter"[86] and often just the corrected word; a replaced letter here, a short vowel change there. In his list, self-described as "sufficient testimony as to the scholarship of the two aforementioned scholars,"[87] al-Shidyaq includes problems or *bugs* in the scholars' code which sit on the edges of their convoluted structure and those which are deep beneath the surface. Using this language, I categorize and distinguish bugs by *depth*. For a bug to be deep means it relies on foundational notions and misunderstandings. A mistranslation of a word due to a typo or ignorance of the specific word has a small site of failure. This site is the erroneous stroke of the pen: momentary and easily fixable. However, not deleting or adding a letter due to meter implies a misunderstanding (or ignorance) of an entire system. For al-Shidyaq, it is obvious that Arabic adverbs may not end in *-u.* This incredulousness stems from a systematic lack by the Parisian scholars: they have such a weak understanding of the complex grammatical code which governs Arabic compared to

[84] 5.5.4
[85] 5.5.13
[86] 5.5.9
[87] 5.5.25

al-Shidyaq. Orientalists' weak understanding displays not as an illegible document, but rather as a deftly patched concoction. Thus, we understand the position of the *bug* in structural analysis (or in al-Shidyaq's case, critique).

The bug is symptomatic. To consider it as such suggests *Orientalism is a disease*, showing itself through its characteristic symptoms. Al-*Saq ala al-Saq*'s appendix reads like a doctor's note, or a list of possible ailments. This is not the first such list in al-Shidyaq's text. In the first chapter of Volume 3, "Firing up a Furnace," a twelve-page list of "defects and ailments"[88] mirrors the appendix's table. One after another, al-Shidyaq overwhelmingly lists symptoms the human body can exhibit, with only a few examples of actual diseases. The explanations match those of the appendix in their tone, occasionally barely bothering to explain with several cases of "too well known to require definition."[89] (Some such cases are humorously accompanied in Davies' translation by an explanatory "[?].") As in the appendix's list of failures, different kinds of diseases appear on this unfortunate body: misplaced and disjointed joints ("*farkahah"* is the "wide spacing of the buttocks"[90]) just as the letters and words of de Sacy's translation are out of place (al-Shidyaq corrects, for example, "جلّال" to "اجلال"[91]), non-specific pains and aches ("*suda"* is a "pain in the head"[92]) just as de Sacy's translations are incorrect in non-specific, general ways (al-Shidyaq asks, as cited above, "What could be the cause of such contortedness?"), cuts, pustules, and lesions, some of which are incurable ("*sulaq*" are "pustules that break out at the root of the tongue").[93] just as the Parisian professors' mistranslations appear on the surface of their writing and

[88] 3.1.4
[89] 3.1.4
[90] 3.1.6
[91] 5.5.20
[92] 3.1.17
[93] 3.1.12

only grow more and more deadly with each occurrence ("This is the seventh time!" al-Shidyaq exclaims, followed by an accusatory "Are you going to put the blame on the printer?"[94]). Festering Orientalism requires, in al-Shidyaq's view, an accounting of symptoms and diseases. A list of problems, thrust in the face of the overconfident patient. This structure of list-made-exposé prompts a return to our table from Chapter 1.

The table compiled by W3C of all the compatibility issues of every language on the internet no longer reads as documentation or specification of problems to fix. Instead, it fits alongside our other two lists as a table of symptoms. Each orange box or question mark is another *bug*, another gap, another failure in digital writing. Therefore, I want to link the tables using their structure, their implications, and their subject matter. In terms of structure, all serve as an extensive accounting of some *body*: the human body, a particular translated *text*, the body of digital writing. They all imply treatment or some form of correction, without actually applying it. Some of the failures in Arabic digital writing on W3C's chart offer some implementable fixes, as I addressed in the previous chapter. The ailments listed in "Firing up the Furnace" are, unless noted by al-Shidyaq, to be treated. Finally, almost all of al-Shidyaq's notes on de Sacy's mistakes are single-word corrections: cures for a specific symptom. As the list goes on, al-Shidyaq becomes increasingly exasperated, offering more removed cures and explanations for de Sacy's mistakes. On one hand, al-Shidyaq operates in his appendix in the same removed state as W3C's table: instead of actually *treating* the long-festering disease of Orientalist overconfidence and indifferent gluttony he is making an accounting. Positing suggestions and fixes is

[94] 5.5.20

different from actually *altering writing*. In al-Shidyaq's case, this is the writing of new printing presses and Parisian scholars. In the modern case, this is digital writing's structural matrix: code.

Moving forward from these tables and lists, I want to underscore the connection their parallel structures and addressed objects offer. By accounting the symptoms of twenty-first and nineteenth-century Orientalism, the tables are a set of portals linking the two forms of writing. This comparative model, introduced by Tarek El-Ariss in *Leaks, Hacks, and Scandals*, conjoins the two structures with undirected chains of associations and affects. In opening this portal, I have opened another Pandora's box: I have theorized bugs in the code of websites as a pustule on a body, al-Shidyaq's corrections as a survey of bugs in the formative structure of Orientalism, and furthered this connection by linking formative structure to modern computer code. This portal has extended the reach of al-Shidyaq's critique from two specific Parisian professors to centuries of digitized and codified ignorance.

Early in the appendix al-Shidyaq notes a particularly pungent mistake by Chodźko in which the Parisian changes "low lying land" to "contents," resulting in a horribly mistranslated passage which introduces "sands of the plain."[95] In truth, my introduction of this passage is slightly confusing, if only because the passage is so far from the actual translation that to tie the two together would only add more confusion. Regardless of the interruption which Orientalism makes to coherent writing (even now, it invades this chapter), al-Shidyaq asks the reader: "How could this scholar permit himself to fill this book with sand and be too proud to ask someone knowledgeable what it meant?"[96] This portal I

[95] 5.3.2
[96] 5.3.2

have opened is one of expansion and addition: it takes an address and explodes it into many. For this sentence, my portal shatters "this scholar" into all the translators, the converters, and specifically the coders of modern Orientalism. Al-Shidyaq's critique mirrors the entire critique of Chapter 1. How is it that Adobe has no documentation for Arabic justification? How is it that no modern design suite or browser supports fundamental aspects of Arabic typography? How is it that there are endless examples of Arabic cut apart and dissected on the operating table of digital Orientalism, then displayed one after the other on divulging blogs and twitter feeds? How could these coders permit themselves to fill digital Arabic with sand, and be too proud to ask someone knowledgeable what it meant? By approaching al-Shidyaq's text and code through a comparative analysis, I hope to reconcile these questions.

Holy Compilers

In order to expand this portal further, I want to pry it open by revisiting the association made earlier in this chapter between code's keywords and translated scripture. In order to justify, solidify, and expand this connection, I first want to describe what I mean by keywords. This section focuses exclusively on their role in C. Since 2000, C has been in the top two most popular coding languages.[97] After providing technical background, I expand upon the function of keywords by reading them alongside Rebecca Johnson's work in *Stranger Fictions* on al-Shidyaq's translations of scripture. During this reading, I re-introduce *al-Saq ala al-Saq*'s writings on *Market-men* and *Bag-men*, al-Shidyaq's terms for different groups of Christian monks, as writings on modern coders. This connection between monk and coder introduces dimensions of God, of institution, and of the divine into our comparative

[97] "Tiobe Index," https://www.tiobe.com/tiobe-index/

reading between code and scripture. In doing so, the coder finds themselves with a new identity alongside that of the "non-diegetic writer" described in Chapter 1. I explore *coder as monk* and *monk as coder* during this section and expand my work on digital writing's comparative reaches. This section also acts as a template for further exploration of my theoretical project by acting out how I may be able to build on my ideas in the future.

Every major coding language uses a set of keywords which are untranslatable, and unconvertable. In a system of writing characterized by flexibility and possibility, keywords serve as anchors to a defined set of operations and structures. The defined structure that a keyword refers to cannot be changed. An example of a keyword is the simple, three-character keyword "int."[98] This keyword is used to "declare" an integer. Declaration in C is the act of "introducing one or more identifiers into a program, and specifies their meaning and properties."[99] An identifier is a coder-defined name for a specific variable, which can be conceptualized for my purposes as a signifier-signified chain. The signifier is the variable name (an identifier), while the signified is what is stored in the memory named by the variable. When declaring an identifier, keywords are used to specify the properties of the signified. For example, when using the "int" keyword alongside a new identifier name, we are telling the C compiler that the signified of our new variable will *only be an integer*. It will refuse to hold, for example, a word (referred to as a "string"), or a character, or a decimal point number, or more complicated types of data (function pointers, etc.).

[98] cppreference, "C keywords"
[99] cppreference, "Declarations"

While some keywords act as "int" does by specifying type of the signified ("long," "short," "float," "double," "bool," "enum"),[100] others change program flow in defined ways. "Program flow" should be understood as the order of actions done by the program. Different operations can be done in different order, in repetition, or even randomly. In order to perform complex operations, complex program flow and therefore keywords are necessary. An example is the keyword "if," which begins a conditional statement. Conditionals perform operations based on the status of other parts of the program ("is one signified equivalent to another signified?"), or based on existential truths defined by the coder or compiler ("1>2" or perhaps the predefined word "true"). Another example is the keyword "while," which repeatedly performs a set of actions until a user-written conditional is false.

These examples illustrate that keywords not only serve as the structure for the signifying chain of C, but also dictate the structure of the code itself. This multiplicity of definition which links function to structure characterizes keywords and explains their immutability. There are some exceptions to this rule, but they occur within the closed systems of keywords. The keyword "using" lets the coder change the keyword to their own defined word, creating another signifying chain for the compiler to track. Regardless, the functionality and syntactic structure of the keyword doesn't change. We are left with specific words which are unchangeable in their meaning and unalterable in their function forming the structure of code. This is the structural refutation of the post-colonial hopes for digital writing: code is linguistic structure actualized and actionized. Keywords are words whose chain is *extra-textual*: they are defined by reference and citation, asserted through errors and bugs.

[100] cppreference, "C keywords"

Keywords are defined in the ISO/IEC (International Organization for Standardization) C specifications[101] as a subset of *tokens*, which are the "minimal lexical element of the language in translation phases 7 and 8."[102] Running human-written C code is a two-step process, separated by the ISO specifications into two separate environments: the "translation environment" and the "execution environment."[103]Translation of C code, for my purposes, should be considered a process of simplification. All human-specific information is removed in phase 7. Comments and overall readability features such as "white-space" are removed, and some preprocessing is done. Phase 8 consists of "resolving"[104] references. I will talk more about this later in the chapter in the context of signifying chains and Shidyaqian synonymity.

Alongside contextualizing code with al-Shidyaq, I would also like to introduce scripture as a mirrored structure: a web of extra-textual, divinely defined words whose authority is asserted by untranslatability and an extra-textual authority of institutions, canons, and posterity. Untranslatability is a clerical doctrine, rather than a functional impossibility. In fact, the skill of Arabic translators is what illuminated the aporia. As mentioned earlier in this chapter, their institutional aporia is referred to by the head of the Maltese CMS translation department as the "chief defect" in the translations. The translation of keywords is not just a question of meaning, but also a question of function and structure. As Rebeccah Johnson writes, "Word choice was not just a matter of producing correct versions of evangelical views but ensuring hermeneutical stability and revealing the universalism of

[101] ISO/IEC 9899:2017
[102] ISO/IEC 9899:2017, §6.4
[103] ISO, §5
[104] ISO, §5.1.2

Christ."[105] For these monks, the interruption by translation of the direct linkage between word and meaning, signifier and signified, and keyword and definition was an ecclesiastical and divine sin. Samuel Gobat, the head of the CMS publications in Malta, tried to uphold the transcendental order by asserting the untranslatability of divine keywords.

Johnson deconstructs Gobat's anxiety of translating divine keywords by framing translation as an additive and dislocating process. In the Arabic translations, she writes, "biblical interpretation becomes an open-ended and ongoing process that *un*decides decided meaning among a plurality of possibilities."[106] Conceptualizing translation as a "process" which has direct, tangible effects on how a text is read and progresses parallels our current understanding of *code*. Code, which I have described as the structure and function of keywords, is also a direct and active process. However, the two processes disagree on their two generalized functions: translation expands, while coding limits. The transcendental goal of code is to accomplish a task succinctly, successfully, and without any chance of failure. The hope of code is to *remain a closed system*, the exact same hope Gobat and the Christian missionaries have for their sacred text. Confusion, dislocation and the "undeciding" of keywords deconstruct both structures in the exact same way, prying open their shared vulnerability. Fear of explosion-by-plurality is what drives monks and coders alike.

This shared anxiety describes a modern doubling of "coder as monk" and vice versa. Anxiety and fear connect these two figures, prompting a comparative reading. They share a jealousness for the structure upon which their livelihood and stature depends. In order to justify and defend, monks and

[105] Johnson, *Stranger Fictions,* 43
[106] Johnson, *Stranger Fictions*, 44

coders create a pretense of non-fungibility and divinity around their text. Despite their efforts, as Johnson points out, they operate in an unresolvable paradox. The word of God cannot be translated, yet the word of God can be understood by all: code cannot be written in non-Latin script, yet code is the hope of global capitalist democracy. The monk and the coder thrash about in the same way, blaming Arabic for its corruption of their holy texts. Gobat calls the synonymity of Arabic a "temptation" which corrupts God's word. Johnson reads further that this synonymity, "a standard feature of high Arabic prose"[107] is what *"un*decides" God's word. Orientalist coders build systems that fit Latin text perfectly, and throw their hands up in frustration at Arabic's complexity. Coders do not offer documentation, explanation, or even justifiable implementations. One feels the coder's contempt for a language their keywords could not handle everywhere, as already discussed in Chapter 1: when browsing the internet, when using professional design software, when editing, and especially when writing. Having established this shared fear between coder and monk, I now return to al-Shidyaq armed with a comparative link to draw connections from *al-Saq ala al-Saq*'s writings on "Market-men" and "Bag-men" to coders and back again.

The second half of Volume 1 of *al-Saq ala al-Saq* utilizes the analogy of the marketplace to deconstruct and ridicule the clergy. Splitting them along institutional lines, the narrator of *al-Saq* calls the Protestant missionaries "Bag-men" and the Maronite and Roman Catholic clergy "Market-men."[108] Al-Shidyaq develops this separation in the chapter "Bad Luck," in which al-Fariyaq is thrust between the two philosophies of Market-men and Bag-men: he is scalped, admonished, taken

[107] Johnson, *Stranger Fictions,* 44
[108] *al-Saq*, Vol.1, Glossary, 351-353

advantage of, and argued with. The entire ordeal begins with an act of *replacement*. Al-Fariyaq takes his inheritance to be repaired, re-dyed, mended, and fixed. *Mending* and replacement mirror translation in their introduction of novelty ("*un*deciding" meaning by adding new association and definition). Just as translation got al-Shidyaq in trouble with Gobat, so does mending get al-Fariyaq in trouble with the Market-men. This materialist analogy is reinforced for the rest of Volume 1, with scripture and the authority of God being referred to as the "price list"[109] and "inheritance,"[110] respectively.

Just as al-Shidyaq split up the figure of the church into two, so too can I split up the figure of the coder accordingly. The coder can be split up along al-Fariyaq's lines: that of the institutional and extra-institutional coder. Regardless of this distinction both of these figures operate and exist within the larger structure of Latin-centric code: Orientalist code. Just as al-Fariyaq addresses and introduces each figure, so will I. The "Market-coder" and the "Bag-coder" have different responsibilities and culpabilities for the current state of Arabic code, and should be considered individually for the sake of clarity. In order to avoid the role of executioner, I will also detail the possibilities of each coder: what they can accomplish and what they can undo.

The Market-coder, like the Market-men of *al-Saq ala al-Saq*, is deeply ingrained in a corporate structure devoted to the upkeep of the public image and ecclesiastic power of that structure. Perhaps it is unfair to characterize the Market-coder as an actual coder, that non-diegetic writer as detailed in Chapter 1. Instead, the Market-coder occupies the role of the executive, the manager, and the team-leader responsible for the abhorrent documentation, the refusal of explanation, and the

[109] 1.19.2
[110] 1.18.23

accusatory table of non-Latin language support. The Market-coder's code, as opposed to the Bag-coder's code, is *closed-source*. Market-men, as al-Shidyaq writes, "have hidden the price list from the buyers and jacked up the prices," and "keep the items they sell wrapped and packaged, and the buyer takes them and goes off without laying eyes on them."[111] The prevention of the diegetic writer's "laying eyes on" is precisely what the Market-coder dedicates all their effort to. Market-coders utilize some of the most complicated math in computer science and the realized threat of literal criminal charges to prevent this act of *seeing* the source code of their applications, suites, websites, and failures. At the same time, Market-coders assert their code's divine ubiquitousness. They warn in *al-Saq ala al-Saq*: "We shall exact revenge on anyone who does not buy from our warehouses."[112] Digital writing and scripture have the same kinetic possibilities which prompt the Market-figures to prevent the act of gazing. To gaze upon code and scripture is to understand a structure down to its fundamental components: keywords, signifying chains, and program flow. Knowledge of these components, as al-Fariyaq displays in "Emotion and Motion" to the Bag-man, allows one to rip apart, hack, reveal, and rebuild writing.

Al-Fariyaq's deconstruction of the Bag-man's text is done *using his text*. He asks the Bag-man: "Is a priest permitted to have intercourse with an adultress and beget bastards, as did the prophet Hosea?" and "Is he permitted to marry a thousand women, queens, and concubines, as Solomon did?"[113] Such probing questions rely on a knowledge of the text: of source material. This is the structural disadvantage of *open-source* writing, and the ultimate fear of the Market-coder and Market-man. However, this disadvantage, for both al-Shidyaq and I, is the beginning of undoing the

[111] 1.20.1
[112] 1.20.1
[113] 1.19.5

structural oppression of code and scripture. This revealing and breaking-apart of code and scripture aligns the Bag-man with the coder who writes in open-source code and devotes themselves to the reverse-engineering and distribution of closed-source code. In the chapter "The Difference between Market-Men and Bag-Men," the Bag-men are those of "sound judgment and perspicuity" and "have been driven by their poverty-stricken situations to broaden the scope of their thinking and to look into and compare and contrast things."[114] Asserting that closed-source code necessitates criminalization, al-Shidyaq declares that the "majority of scholars and original thinkers are vagabonds."[115] For Bag-men and Bag-coders, clarity and visibility is the ultimate goal, not the ultimate fear. Bag-men "published [the price list] in all the lands," and declared "to the people: 'behold the clearest of ledgers, the most extensive of registers. Buy from us nothing that isn't according to the price sheet. Don't go to the Market Boss, he is beyond conceit."[116] This philosophy of the Bag-men is also the philosophy of the Bag-coder: transparency, reliability assured by public discretion, and the undoing of closed-source Market-coders. These are the operatives, coders, and *vagabonds* dedicated to the undoing of the modern Orientalist order.

However, the survey of Chapter 1 complicates this figure of the Bag-coder. If open-source is the undoing of Orientalist practices, how is it that open-source suites such as LibreOffice still fail to account for basic Arabic printing practices? C, that language of keywords and divinity, is as open-source as they come. While the figure of the Bag-coder is hopeful, the results of the past couple

[114] 1.20.2
[115] 1.20.2
[116] 1.20.7

decades are anything but. Yet, Bag-coders such as Ramsey Nasser, myself, and many others are devoting themselves to exposing Orientalism in its modern form and recapturing digital writing.

This section served two purposes: a comparative reading of the figures of the coder and monk alongside and with *al-Saq ala al-Saq* and a demonstration of my project. It offers a structure for future use: first, a technical introduction and subsequent analysis. Then, I deduced a comparative link from that analysis. After establishing this comparative link, I engaged with a canonical text in order to expand the reading further. The following section performs the same process with a different comparative link and al-Shidyaq's text *al-Saq ala al-Saq*. Repetition of the same structure offers all the de-stabilizing lines of flight which doubling entails. It also serves to solidify and further define this process I will build on throughout my theoretical project beyond this paper.

Linking and Synonymity

The previous section pried open our portal , so I will continue with another comparative reading. In this section, I establish a link between Shidyaqian synonymy and the process of "linkage" by the C Linker. Following the structure of the previous section, I will begin with a technical background of linkage. After establishing that background, I once again use the work of Rebecca Johnson to explore al-Shidyaq's construction and deconstruction of synonymy. After developing a conceptual framework for synonymy in *al-Saq ala al-Saq*, I define and defend a link between linkage and synonymity. After defining this link, I expand upon it with a comparative analysis of the two concepts.

Just as in the previous section, we will turn to the C documentation as our reference text for technical background. The ISO reference sheet for C describes "linkage" in the context of identifiers. Identifiers were also described in the previous section: they are the coder-assigned signifiers. There are three different types of linkage described in the reference: "external, internal, and none."[117] Keywords are associated with each type of linkage. C programs are almost always (the exception being trivial cases) made up of multiple files which split the program up into separate, more easily readable and organizable modules. *External* linkage is the linkage of identifiers across different compiled files. These files are called "translation units" by C.[118] The "extern" keyword is used by the coder to specify an identifier with external linkage. *Internal* linkage connects all of the uses of an identifier in a single translation unit. For internal linkage, the "static" keyword is used. Finally, if an identifier has *no linkage*, then the signified object is unique. An example of an object with no linkage is a function parameter: a value passed to a function for use by that function.[119] One cannot use both keywords when referring to the same identifier in the same translation unit, else it is referred to as "undefined behavior."[120] Undefined behavior is an interesting state of C code, where it is up to the hardware to define what happens, leading to different results on different machines.

Linkage is how C files are able to access external libraries. These are standardized, digital repositories of code written by others. When external libraries are used, the code has access to externally defined functions. By using the keyword "#include"[121] followed by the name of the external library, the

[117] ISO, 6.2.2.1
[118] ISO, 6.2.2.2
[119] ISO, 6.2.2.2
[120] ISO, 6.2.2.7
[121] "#include" isn't necessarily a keyword, it's a preprocessing token. But calling it a keyword is not far from its actual usage.

identifiers of those functions are linked to the current translation unit. Each of the standard libraries or "header files" have their own set of "reserved identifiers," names which must be used by the coder when accessing those functions or values. These header-defined identifiers have their own grammar, with underscores and letter cases denoting availability. The process of linkage, which occurs in translation stage 8, allows for code which references functions, objects, and other signifieds beyond the locally written files. These reserved identifiers, which cannot be changed, are all written in Latin text. All non-trivial C code relies on these core libraries for functionality, ease, and structure. Thus, linkage is the process which ties signifier-signified chains together in C code. It also links different chains together, and connects separate signifiers to the same signified. Linkage also creates signifying chains which extend beyond the text of the code into standardized and open-source libraries. This structure of chains defines a C program. The sharing of signifieds, the passing around of identifiers, and the process of accessing and changing the signified through and by multiplicity all make up this structure.

As Rebecca Johnson writes in *Stranger Fictions*, al-Shidyaq's conception of synonymy rejects equivalence. Johnson notes several meanings for the Arabic trilateral root that al-Shidyaq uses for "synonymous" ("*r-d-f*"): "to pile up in layers, to become stratified, to flock, [...] to follow one after another."[122] Johnson describes the long lists of synonyms which characterize *al-Saq ala al-Saq* as a "trial of modernity" for the signifying chain. In al-Shidyaq's text, "simple definitions [...] collapse under the weight of his lists of subtly differentiated synonyms."[123] *Al-Saq ala al-Saq* complicates meaning with its never-ending chains and linkages. For al-Shidyaq, these "nodes of association" create a

[122] Johnson, *Stranger Fictions*, 48
[123] Johnson, *Stranger Fictions*, 49

structure of definition between "synonyms" which forgoes an actualized center in favor of maximum connectability.[124] Connectability offers new paths of interpretation and comparative analysis, and encourages individual approaches to the same text. Individual realizations of the same text explode the text into many, each with its own function, structure, and associations. These individual versions of *al-Saq ala al-Saq* mirror the concept of "undefined behavior" in C: text and program split into personal versions, each with their own functionality and structure.

From the beginning of *al-Saq ala al-Saq*, the text stresses the importance of the structures created by its synonyms. At the end of a long list of words, slang, and other terms for genitalia and intercourse, al-Shidyaq creates a constraint for his system of expansive association. In Chapter 1 of Book One, "Raising a Storm," the narrator "imposes on the reader" a "condition that he may not skip any of the 'synonymous' words in this book of mine."[125] This limiting condition ensures some similarity between the different readings: the long chains of this text *demand to be read*, charging the reader with a necessity to confront them. For the narrator of *al-Saq ala al-Saq*, these lists of synonyms are hardly repetition: such a conception of writing asserts the independent function of *signifier*: altering the signifier changes the signified. After imposing his condition, the narrator "admits" that in fact he "cannot support the idea that all 'synonyms' have the same meaning, or they would have called them 'equi-nyms.' They are, in fact, synonymous only in the sense that certain of them may take the place of certain others."[126] In this model, synonymy operates as a game of replacement and assignment. *Al-Saq ala al-Saq*'s identifiers are exchanged with "certain" others; when one signified is changed and

[124] Johnson, *Stranger Fictions*, 48
[125] 1.1.6
[126] 1.1.7

altered by the use of a specific identifier, all identifiers associated with that signified. Yet, these identifiers are unique and separate: they are "synonyms," not "equi-nyms." These extended chains characteristic of al-Shidyaq's text proffer a feature of digital writing: a dynamic system of linkage.

Thus, when read alongside *al-Saq ala al-Saq*, our technical and Latin-centric definition linkage explodes into a kinetic interplay across translations and centuries. Originally framed as specified associations furthering Latin-only code when combined with standard libraries and reserved keywords, *al-Saq ala al-Saq* offers a reading of linkage which emphasizes its destabilization of structure by multiplicity. External linkage in particular changes drastically. External linkage was described as a collapse of several different identifiers into a single one. This collapse happened at the dimension of space: originally, these identifiers were spread across different header files (human-written code), translation units, and texts. Linkage seemed to erase this difference by connecting all identifiers to the same address in memory, the same signified. Identifier after identifier all resolved to the same object, the same function, the same *place*. Al-Shidyaq's writing offers a different interpretation of the same phenomena. *Al-Saq ala al-Saq* insists that the repeated identifiers and signifiers of code are anything but repeated. Shared identifiers are each a new corruption of the signifying chain: an infinite doubling which emphasizes the signifier over the signified.

These identifiers are *synonymous*: each is used in separate contexts in their own way. Digital synonyms, keeping in line with the definition of digital writing as *active writing*, destabilize their structure beyond its theoretical understanding. In true Shidyaqian fashion, these digital synonyms are junctures that could spring a leak at any moment. Deviant usage of linkage's keywords like "extern" and "static" introduces the ultimate dream of any vagabond coder or hacker: undefined behavior. Once

C code has entered this state, all bets are off. For theorists and hackers like me, undefined behavior is realized deconstruction: anything from bug-free code execution to "having demons fly out of your nose"[127] is possible. As mentioned earlier, undefined behavior is hardware-specific behavior: each person will get different results. For hackers, undefined behavior can be the key to undoing entire systems of encryption and validation, and has been since hacking began.[128] These coders are very different from the Market-coders: they are their natural enemy, and face the threat of criminalization for their digital writing. Thus, alongside digital writing the Shidyaqian chain of synonyms destabilizes more than the text itself: these long strings of synonyms have the ability to undo material structures of power. And they do: *al-Saq ala al-Saq*'s narrator attacks the Market-men and the "confederacy of cretins"[129] which killed his brother. His writing *exposes* their misdeeds and *poses* to his readers their lies and trickery through his synonyms, name-calling, insults, and lists.

This was a smaller version of the previous section, with a more tenuous comparative link to begin with. However, this short comparative reading showed that tenuous links are the bread and butter of the theoretical economy of digital writing. By exploring these connections, I was able to introduce propinquous concepts such as undefined behavior and the hacker as writer and vice versa. Previously defined concepts such as digital writing were expanded upon with example and application. This expansion of my theory, even when dealing with weak comparative links, is a process which shapes and revises my theoretical project.

[127] catb.org, "Nasal Demons"
[128] blexim, "Basic Integer Overflows"
[129] 1.19.3

Chapter 3:
Genres of Breakage

The private Telegram group chat"LibreOffice RTL" was difficult to find. It contains 27 people (including myself) who work to document and fix bugs in right-to-left text on LibreOffice. The chat consists mainly of tentative drafts of bug reports: questions about wording or if the documented bug already exists. The most active admin in the group chat, Eyal Rozenberg, can be found in the "replies" sections of many RTL bugs on LibreOffice's wiki. Tactics of the group are usually a two-step process: first, properly document bugs so as to remove any excuse of ignorance from LibreOffice's bug-fixing team; next, a public-relations campaign raising awareness of the bug so that it is included in the budget for LibreOffice's bug-fixing team.[130] These writers are at the intersection of non-diegetic and diegetic digital writing. The non-diegetic source code and the diegetic writing in LibreOffice documents encounter each other in this group chat. More specifically, the encounter takes place at the *bug report*, the topic of this chapter.

Before formally introducing the bug report, I define a digital typology of bugs. My two classifications, "resolved" and "unresolved" are corrupted and broken apart by my own classification, demonstrating the deconstructive tendencies of bugs. In the two preceding chapters I gave examples of this structural deconstruction. Chapter 1 offered specific examples of modern design suites and word processors, while Chapter 2 dove into deconstruction with comparative readings of Ahmad Faris al-Shidyaq's *al-Saq ala al-Saq*. Now, I intend to investigate a design suite of the first chapter while

[130] Pourrabi, "Bug 104597"

armed with the concepts of the second chapter. One of these concepts was the *bug*, a symptom of an inconsistency or leak in a digital structure. Various types of bugs were discussed, from a typo in a nineteenth-century Parisian translation of Arabic poetry to unexpected code execution due to incorrect usage of C keywords. Bugs are a realized symptom of an underlying structure: due to their visibility, they are the main entry-points when discussing Arabic digital writing. After all, bugs characterize Arabic speakers' experience when using the internet or applications, especially word processors. During my classification, I address the distinction between the "physical writing" of ink and the "digital writing" of photons..

After establishing in El-Arissian fashion this leaky and flawed dichotomy, I theorize the bug report as a genre of digital writing, specifically in relation to bugs originating from Latin code's inability and unwillingness to handle Arabic text. I then dive into a specific bug report where several commenters (another form of the digital writer) argue about the structural provenance of bugs in Arabic digital writing. During my reading of their discussion, I do not criticize the abilities (such as familiarity with Arabic) of these specific individuals. Rather, I approach their misgivings and opinions as representative of my larger research, and as symptoms of sentiments endemic to Latin digital writing.

Digital Entomology

Digital writing, as I have described it, is kinetic writing: one of corrective and destructive *action*. Code presents unique to other forms of writing by revealing its successes and flaws solely through symptoms. Its miasmatic applications and webpages coagulate according to the structure of underlying code. In their own moments of possibility, applications spring leaks and error messages which often betray their code. Hackers depend upon revealing spillages of hints and source code which tech companies go to great lengths in order to prevent from leaking. Millions of dollars are spent to plug Latin code's leaks.[131] In this closed system of reports and budgets, Arabic digital writing deconstructs and unwinds carefully laced together structure.

Bugs are the passageways of *digital deconstruction*: deconstruction which takes digital writing and code to be both texts and tools. My digital deconstruction theorizes bugs as inconsistencies and gaps originating from a larger, hidden structure. When Arabic text joins incorrectly (or not at all) in a word processor, this bug reveals a failure to properly test and document Arabic and other non-Latin scripts. This failure explodes into a history of negligence and indifference which Middle Eastern Studies calls Orientalism. Digital deconstruction uses the bug as a starting point, but does not consider the bug to be the beginning or end of the diseased, originating structure which it is revealing. Instead, the bug is a kind of middle or in-between which suggests complicated provenance and possible resolution. I have already discussed the provenance of the bug: I would say that its provenance is the subject of the previous two chapters. Thus, I want to theorize the *resolution*.

[131] Arghire, "Google Paid Out $8.7 Million"

This chapter will investigate and theorize the resolution of the bugs which arise from Arabic digital writing's encounter with Latin digital writing. At first glance, considering bugs in relation to their resolution suggests two natural categories: those which have been fixed and those which have yet to be. However, there are categories other than these two which cannot be assigned as a sub-category. Therefore, the duality of bug resolution is misleading: some phenotypes of bugs drift between or evade completely the two categories. The bug inherently evades assignment and categorization, and bug resolution is no exception. Despite this I am able to define certain additional categories, if only to create and define a structure which I will undo with exceptions and paradoxes.

Resolved bugs are the first major family of my taxonomy. They consist of every bug which used to exist. These bugs no longer exist in the newest version of its originating software or hardware. Some of these bugs can still be found on older versions of software, on devices without the newest update, and on individually irreplaceable outdated hardware. Bugs do not have to be resolved intentionally: they can be fixed accidentally when resolving a different bug. Many of these bugs may live their entire lives unnoticed and undocumented: nobody ever performed the sequence of actions to re-create this bug; even if these actions were performed, the user didn't notice the bug. Other bugs are simply deleted: no fix can be found so the flawed feature is removed entirely. An example of this type of bug was Microsoft's inconsistent implementation of Arabic kashida justification in Internet Explorer, a feature which was removed entirely when Microsoft migrated to Chrome as their browser. The most common type of resolved bug is the documented and intentionally fixed bug. This bug was most likely documented in a so-called "bug report." The bug report is a structured and defined template for

documenting and reporting bugs to their writers. After the bug report is submitted, coders work together with the reporter to fix the bug in question.

Despite this structured process of bug report submission and review, many bugs are *currently unresolved*. Thus, my classification of bugs demands a temporal dimension of digital writing. This is the most significant difference between physical writing of printing presses and ink and digital writing of code, PDFs, and blog posts. The speed at which each writing *happens* is drastically different. This speed should be thought of as a product of *medium*: digital writing happens at the speed of light, while physical writing happens at the speed of ink. This is not a rule, but rather a characterizing possibility of each type of writing. As I show later in the chapter, digital writing can take seconds, minutes, and even years to emerge and develop.

Digital writing is also characterized by its explicit creationism or *positivism*: all digital writing relies upon or produces another text. The text of this thesis relies on the source code of Google Docs, the source code of Google Docs relies on external libraries, and so on. This chain eventually reaches the *actually atomic* level of photons, and then begins again in the opposite direction. Google Docs' code creates this text, just as much as my fingers and hands. At the same time, the source code of this thesis *is* this thesis, until it is printed out onto the page. Perhaps changing the source code changes the thesis itself. A demonstration:

Many of the bugs shown in this chapter (there is in fact one present in the above image—misplaced periods alongside Hebrew type) will eventually be resolved. In fact, their resolution is the hope of this chapter. Yet, other bugs will replace them. Because digital writing is kinetic writing, its actions and evolutions *by the very nature of digital writing* create new bugs to be resolved. Therefore, when discussing unresolved bugs or digital writing in general, I want to make it clear that I am describing an ongoing process as opposed to a completed text.

With the temporal dimension in mind, I want to designate some categories of the unresolved bug. Unsurprisingly, these categories mirror those of the resolved bug. First, there are the bugs which have never been discovered and are still unresolved. These bugs hide underneath the surface of seemingly perfect applications and require a specific order of actions for them to emerge. Conversely, these bugs also hide in total messes of programs. For example, you are unlikely to notice a bug in the kashida justification of Arabic characters if the Arabic letters are not joined at all.

Unresolved bugs do not have to remain resolved: the bug does not need to reach this state, and a linear connection between the two categories must be refuted. As mentioned earlier, certain bugs survive by hiding. Many bugs are resolved in one version of the software, only to resurface in a later version. I see this kind of behavior in various editions and translations of texts: a typo disappears there, reappears elsewhere. This kind of bug defies my classifications by blurring the border between the two categories: the bug is, *at once*, resolved and unresolved. Other bugs dodge resolution by occurring unreliably. These bugs are the trickiest to fix: they usually occur when code is multitasking and therefore accessing memory in an undetermined pattern.

This does not necessarily mean that easily reproducible bugs are easily fixable. However, the reproducibility of a bug will often determine the attention it receives (in addition to severity). The coder must be able to pinpoint and locate the bug if they hope to resolve it. Almost always, locating a bug implies *laying eyes on* the bug. This visual *event* begins a process of classification and eventual location of its offending code. The act of viewing a bug also acts as proof of existence. In my case, I have shown bugs in Google Docs' text editor when handling Arabic text, especially when alongside Latin script. My exhibition of these bugs means to alert and alarm those who were unaware of the state of Arabic digital writing. On the other hand, my examples provide legitimacy and importance to my project. Visual exhibitions of bugs are almost always used to prove the existence of the bug itself to the bugs' creator or other digital writers and coders dedicated to resolving them. The structure of the "bug report" formalizes this proof.

The bug report is a pre-specified structure for users of software or hardware to inform coders of bugs. The structure of the bug report varies from coder to coder. Companies which have suites of

applications will standardize the report across all their products. Meanwhile, solo coders will often simply provide a contact email address. Online forms dedicate sections to reporting and documenting bugs. Despite their variety, bug reports roughly follow the same structure. Users provide a username and contact info so they can be reached if they provide incoherent or incorrect information. They then describe the specific steps to reproduce the bug. Users will usually also provide a screenshot or some other kind of *visual proof*. Finally, the user records the version of the software the bug occurs on.

The visual proof of the bug varies depending on the bug itself. In the case of al-Shidyaq's appendix of *al-Saq ala al-Saq*, his visual proof was the incorrect word or line; his steps to reproduce took the form of location in the text in question. Titus Nemeth's examples in his report offered visual proof with images of misaligned and severed Arabic script. Other bugs are less visual: instead of an image, a so-called "stacktrace" is provided. Stacktraces provide a hierarchy of the program flow down to the offending segment of code. Bugs that offer stacktraces usually interrupt the program or incorrectly access memory. However, stacktraces are rarely seen by users in the case of closed-source code.

The distinction between closed-source and open-source code must be made in relation to their resulting bugs. Closed-source and open-source coders are both dedicated to resolving bugs, but the process by which bugs are resolved deviates between them. Users of open-source tend to be more digitally literate, so the resulting bug reports are often less structured yet more specific. Closed-source bugs are resolved much quicker; bugs are symptomatic, revealing their offending code. This act of revealing is the ultimate fear of the closed-source coder: it means loss of reputation, loss of money, and potential exposure of the source code. While closed-source is maintained through a corporate structure of management, open-source code is often maintained by an ever-changing, self-organized assortment

of volunteers. Open-source bugs are free to be viewed and resolved by anyone with the ability to. The ability to resolve bugs brings us to this chapter's primary source: bugs in LibreOffice's open-source word processor.

Status: NEW

In his report "On Arabic Justification," Titus Nemeth considers LibreOffice the most reliable of the major word processors when it comes to Arabic text. Nemeth reviewed three different font types and their Arabic language support on different processors, and found that "LibreOffice is the only word processor that nominally supports all three font formats."[132] Despite this, Nemeth found errors in Apple's Arabic fonts on LibreOffice: incorrectly placed tatweels would intersect other letters. Currently, LibreOffice provides no options for justification, and simply follows to the font's default. This behavior mirrors that of Microsoft Word and Apple's pages. However, what makes LibreOffice unique is its open-source repositories. By diving into the forums of LibreOffice, I am able to pick up where Nemeth left off.

LibreOffice's documentation is located on "The Document Foundation." This wiki contains information and tutorials on developing for LibreOffice. There are ways for both *users* and *developers* to "get involved." Splitting these tasks between the two roles highlights the importance of multilingualism in digital writing. Instead of French or Italian, the wiki recommends fluency in "C++, Java, Bash, Python"[133] and more. The "Get Involved" page of the website includes some links for users,

[132] Titus Nemeth, "On Arabic Justification"
[133] Document Foundation, "Get Involved"

such as “translate to your native language” and “reporting and testing of bugs.” However, the Get Involved page specifies: “on this page we concentrate on *developers*.” The squashing of bugs is the point of this page. Section titles implore those who can act *to act*: “Find your first bug to solve,” “Solving a bug,” and (the possibly premature) “Congratulations.” The wiki includes long tables of bugs for the aspiring digital writer to attack. Bugs in Arabic writing are classified underneath “RTL bugs”: right-to-left bugs.[134]

Each bug has its own report, submitted and written by a user. They all have their own title and ID number so they can be easily identified. Bugs are sorted into a hierarchy of topics, each with their own respective topic ID. At the time of writing this (May 2022), there are twenty-two bugs in the Arabic/Farsi specific section of RTL bugs. Each of these bugs also has their own “status”: “NEW,” “UNCONFIRMED,” and “NEEDINFO.”[135] In these bug reports, Latin digital writing’s confusion with Arabic digital writing surfaces. These are the bugs of Digital Orientalism. In this section, I will read these bug reports as modern texts of digital writing. My reading of the bug reports will focus on more than Latin code’s inability to handle Arabic writing: I will also dive into the desire of Latin coders to blame their own failings on Arabic and their reliance on indifference to absolve their conscience. I will also emphasize the *kinetic* nature of digital writing by reading the source code for LibreOffice Arabic support.

The title of these bugs are user-written, and are meant to be a short summary of the problem. Bug 116641’s official title is “RTL text that starts with LTR character rendered incorrectly without

[134] Document Foundation, “Arabic/Farsi”
[135] Document Foundation, “Arabic/Farsi”

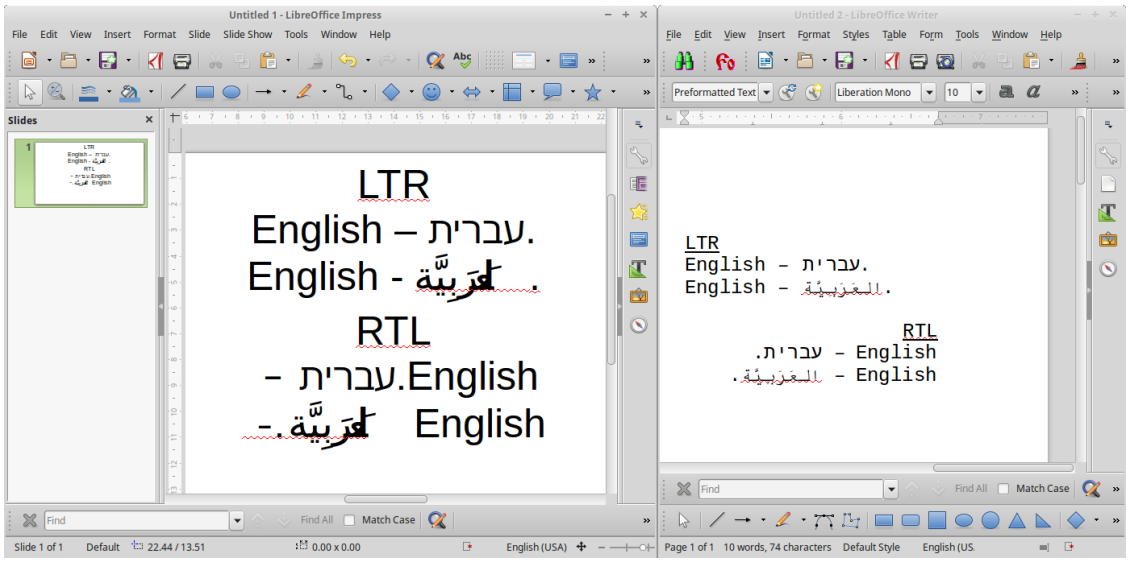
explicit RTL mark."[136] Digital titles break from grammar and syntax: the omission of articles and the liberal use of acronyms give the text a rushed tone. The bug report follows the structure described earlier in this chapter: attachments are given, followed by steps to reproduce the bug. Versions of LibreOffice and hardware are included. The author, "Idan Miara," begins their report with the succinct greeting of "Hi." The bug details incorrect rendering of text in "Impress," LibreOffice's presentation tool: their version of PowerPoint. I have included quotations around the author's name in order to highlight the fact the name is actually a *username*. There are five steps required to reproduce this bug, as quoted from the bug report:

1. Create a new Impress file.
2. set RTL mode.
3. write the following examples (without quotes): "English - .עברית"[137] or "English - العَرَبِيَّة." that is [LTR letters]+[-] + [Hebrew/Arabic]+[.]
4. the rendered text is incorrect. if you write the same in Writer it renders OK. it also renders OK on Impress if you add an RTL mark at the beginning.[138]

Following the flippant syntax of the title, "Idan Miara" has eschewed capitalization whenever they feel like it. Their steps to reproduce the bug *produces a separate bug* on *a different office suite*. In step 3, the period is incorrectly placed to the left of the Hebrew text instead of to the right. Idan's screenshots tell a story of frustration and Frankenstein-like severing and suturing of non-Latin text. The attachment titled "140894: Screenshot: Bug manifests with Impress, not with Writer" shows two windows. In the left window running Impress, the periods are incorrectly placed, and the hyphens have teleported to the leftmost side of the text. Spaces appear. In the right window running Writer, LibreOffice's word

[136] Miara, "Bug 116641"
[137] My footnote now: the period is supposed to be on the right side of the Hebrew text as it is in the bug report, but Google Docs decided (without consulting me, of course) it was better suited for the left side of the text. Any use of Google Docs' formatting tools of "Set text direction to right-to-left" leads to a horrible disfigured quotation.
[138] Miara, "Bug 116641"

processor and what I have meant up to now when I wrote “LibreOffice,” renders the text in the correct order.

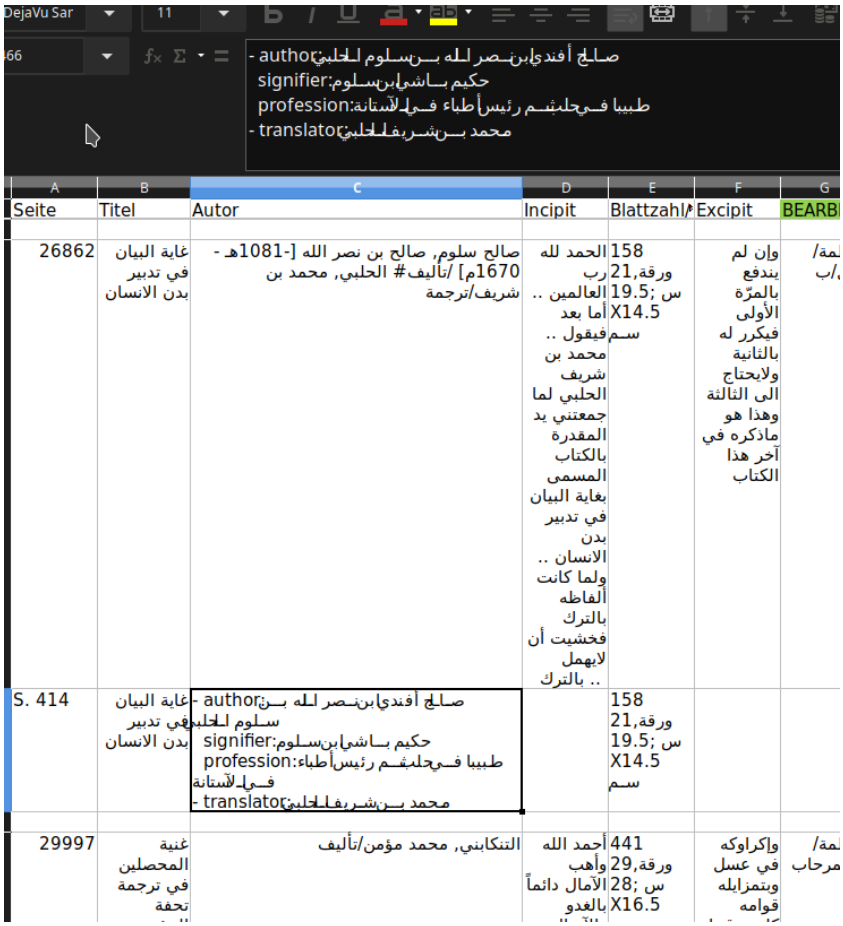


[139]

The second screenshot, titled “Screenshot: similar or same (?) bug in Calc” shows a similar rendering issue in Calc, LibreOffice’s spreadsheet application. Hyphens are teleported (along with the word

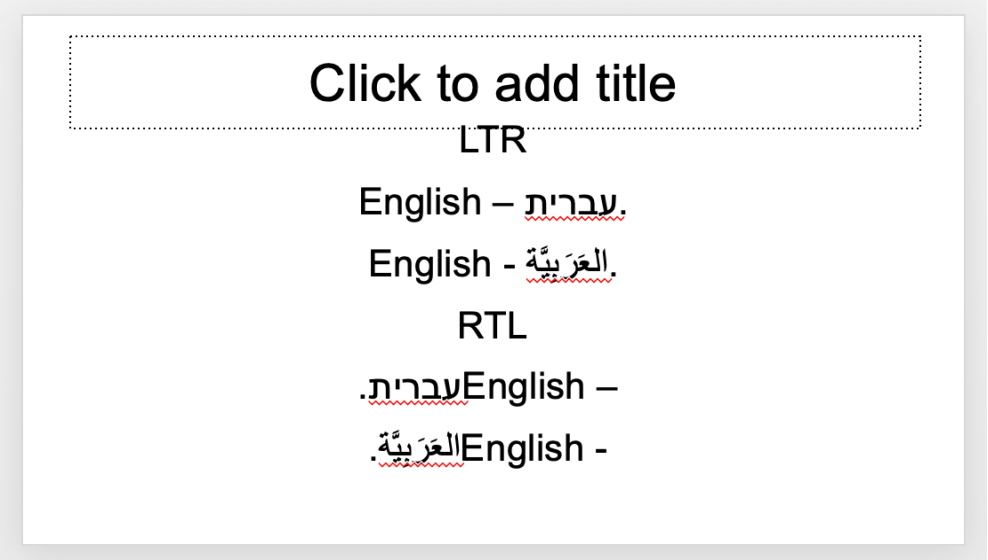
[139] Miara, “Bug 116641”

"translator," emerging again in our network of bugs) and text is ripped apart.



A "testdoc" was also included. When I attempt to open the file "testdoc_116641.odp," my Mac decides to run Microsoft PowerPoint and shows me Arabic and Hebrew text teleported to the left of the line:

[140] Miara, "Bug 116641"



[141]

All of this information was provided in order to explain, show, and prove the existence of a bug. The author painstakingly compiled this together into a new digital text: one where grammar and syntax are afterthoughts, and where *revelation is prioritized*. What follows "Idan's" post is a string of replies and re-replies critiquing and analyzing their report. Bug reports are a prime example of the *presently palimpsestic* nature of digital writing. Digital writing is constantly evolving and morphing: posts are deleted or censored, new comments are added, code is changed, and new bugs appear. This bug report wonderfully demonstrates this palimpsest.

The first reply to Idan's post reads "The image you posted does not correspond to your reproduction instructions."[142] The author of the reply, "Eyal Rozenburg" replied seven minutes later with a screenshot of their own, now deleted. The deletion of Eyal's original post is explained by the next post, sixteen minutes after the previous one, in which Eyal sheepishly admits "[...] and of course I managed to mess things up with my last screenshot. Sorry, it was just LTR. I was thrown off since the

[141] My screenshot from my Microsoft PowerPoint opening "testdoc."
[142] Miara, "Bug 116641"

Arabic was messed up already in that state."[143] Eyal's new screenshot would become Attachment 140895, shown earlier in the chapter. In their reply to the bug report, a user failed to correctly identify the failed state of Arabic writing because it *already had failed* in the intended correct state. This is the current state of Arabic digital writing: the bugs have become so endemic that one can no longer discern between "feature" and "bug." At the same time, Eyal's revision of their own post demonstrates the revisionist and palimpsestic nature of digital writing. Furthermore, the difference in time between each post re-introduces the temporal dimension to our discussion of code. Bugs develop over time, as do our understanding and theorizing of them. Slowly, with each passing minute, the location of the bug's offending code and structure becomes more and more evident. But not all bugs want to be found.

One year after Idan's original report, the user "J.R. Schmidt" posts their own reply. They have been using Calc in their own projects and have begun to see the bug, or at least something "that may or may not be the same."[144] Schmidt includes an attachment that would become Attachment 154293: "similar or same (?) bug in Calc." There have been no posts on this bug's page for a little over a year, and again the bug appears unexpectedly. Schmidt offers different methods of documentation such as categorizing the Calc bug as its own. In the end (at the time of my writing) their bug is absorbed into Bug 116641. Everything in Schmidt's post is descriptive except for the third sentence: "Unfortunately, you don't really get used to reading that way; it'll always slow you down." In the middle of this technical bug report is an outright lament of the LTR direction of Arabic text. Hundreds of millions of people are already quite familiar with reading "that way" and at least a million more (including

[143] Miara, "Bug 116641" These are not my ellipses.
[144] Miara, "Bug 116641"

myself) have learned to read Arabic text. To reiterate my introduction to this chapter: I am not critiquing the reading comprehension skills of J.R. Schmidt. Instead, I am presenting evidence of the confusion Latin digital writers have in regard to Arabic. I am not drawing my evidence from a random survey or cherry-picked application. Rather, this evidence emerged *immediately* upon my focus on the micro level of the interaction between Latin and Arabic digital writing.

Schmidt posts another couple of clarifying replies, and then goes silent. Two years later, LibreOffice decides to re-open the bug and assign coders to fix it. The decision is automated: because no one had posted on the bug's page for "over a year," "LibreOffice QA is asking bug reporters and confirmers to retest open, confirmed bugs . . ."[145] The next day, a new user posts. Ostensibly one of the "bug confirmers" whose arrival was hailed by LibreOffice, "ahanghara" reported on September 25, 2021 at 17:24:33 UTC that the bug still existed: "I use LibreOffice on 7.2.1.2 on Kubuntu and I still see this issue."[146] By now, it has been over three years since "Idan Miara's" original bug report.

Thirty minutes later, after not posting for an entire year, Schmidt leaves a message to the bug confirmer: "Not surprised. Computers inherently carry within them the essence of colonialism and imperialism. These things will never rise to an importance that someone would address them in anything resembling a timely manner as compared to other issues."[147] Not to have LibreOffice's bug-handling structure threatened by "colonialism and imperialism," the bug reporter replies seven minutes later: "Don't take the issue somewhere else. Understanding RTL is not easy for those who have only seen LTR whole of their life. Let's have respect for each and every contribution to the

[145] Miara, "Bug 116641"
[146] Miara, "Bug 116641"
[147] Miara, "Bug 116641"

project. We can have out own part in reporting the issue and encourage those who can understand the issue to help with the solution." Finally, three minutes later, at 18:00:09 UTC, Schmidt makes the final post on the bugs page as of writing: "You're right of course. It's just frustrating sometimes. Can't think of a project off the top of my head whose primary purpose is to deal with text in one way or another and which doesn't have issues with something basic about handling RTL or non-latin scripts."

At the time of my writing, the importance of this bug is currently "medium-normal," and the assignee is the ever-prolific "Not Assigned."[148] As mentioned earlier, the status is "NEW." There are currently three emails on the CC list . There has been no real, documented progress on this bug for four years. So why does ahangarha quickly refute Schmidt's claim? After all, this "thing" has failed to "rise to an importance that someone would address them in anything resembling a timely manner." Schmidt even provides a structural explanation for the LibreOffice developer community's failure to address this bug. However, the bug reporter's attitude is that of the larger Latin digital writing community, and they deliver an admonishment that so many people have heard before (and will hear again): "don't take the issue somewhere else." In this example, specific and modern, we see the fears and anxieties of Latin code which I have theorized *acted out*. The fear of exposure, of breakage, and of massive economic loss drives Latin code to constantly seek reassurance and absolution from itself.

I am not attacking these two specific digital writers, far from it. ahangarha is an active member of an open-source community which dedicates itself to fixing and reporting bugs. These bugs are a constant reminder of Arabic's lack of inclusion, and ahangarha is a part of the team working to undo them. J.R. Shmidt has stayed active on an issue nearly a year after reporting it, responding within an

[148] Miara, "Bug 116641"

hour to ahangarha's post. Shmidt also works hard to make sure these bugs are reported, fixed, and get the attention they deserve. Instead, I am attacking the system which makes Schmidt give up their frustration so quickly. Not fifteen minutes after naming and accusing imperialism and colonialism, Schmidt instead decides that the bug reporter is indeed correct, "of course." This decision has happened so many times before, and it is a key proponent of digital Orientalism: a reluctance to take the underlying structure seriously and consequently a decision to limit the coder's perspective to *the scene of digital writing*—the code itself. Latin code offers indifference and reluctant acceptance to anyone who hopes to change the current hegemony. The digital writer or theorist is always confronted with the refutation "what can you do about it?" Without understanding of Latin code, no actual changes can be made.

Specters of Said

ahanghara's admonishment of "don't take the issue somewhere else" begs the question: whither somewhere else? When/How/Where is this place which we shouldn't go? The presence of the question itself suggests movement and transportation. Thus, there is also time: a present moment, a previous occurrence of this act, and a possible rebellious future. It could be rebellious because of the original phrasing: *taking* or seizing or thieving this "issue." What exactly is this issue? And where/when/how is this theft occurring? Stacking didactic placeholder upon placeholder (who, what, somewhere) displaces every divided subject involved in the theft.

I am not the first to ask these questions. Jacques Derrida, in *Specters of Marx* asks "whither." "Whither Marxism?" Derrida recognizes the necessity of the temporal in asking "whither": "this

question *arrives*, if it arrives, it questions with regard to what will come in the future-to-come."[149] What is this future-to-come which we should avoid, which we should not take our issue towards? In Schmidt's case, the future-to-come was of "imperialism and colonialism." Is this my object, the one I have written this thesis to find? "Whither imperialism and colonialism?"

Like Derrida, I have a sense of "dejá vu"[150] when confronted with my own question. So many times have I asked this question. It has been the goal of my collegiate education to provide me with every established answer to this question. So often we turn to Edward Said for our answers. He obliges immediately: "the academic tradition," and the "very large mass of writers, among whom are poets, novelists, philosophers, political theorists, economists, and imperial administrators,"[151] all of whom have accepted the fundamental, apocalyptic distinction between "East" and "West." *Orientalism* describes a structure using "reasonable qualifications": the actual existence of the East, the necessary study of power structures in order to understand resultant "ideas, cultures, and histories,"[152] and that "one ought never to assume Orientalism is nothing more than a structure of lies or of myths which, were the truth about them to be told, would simply blow away."[153] These conclusions have been repeated ad infinitum, and I need not critique or refute them. I don't even want to. Where has Said been this entire time? Why is this my first (and last) mention of him? Thus I am struck by the same miasmic frustration as Derrida when confronted with my own question which "resonates like an old repetition:"[154] "whither imperialism and colonialism," and indeed: "whither Said?"

---

[149] Derrida, *Specters of Marx,* xix
[150] Derrida, *Specters of Marx*, 15
[151] Said, *Orientalism*, 2
[152] Said, *Orientalism*, 5
[153] Said, *Orientalism*, 6
[154] Derrida, *Specters,* 15

However, as Derrida wrote, the question of "whither" is concerned with a future-to-come. Where will imperialism and colonialism go? Or rather, where have they gone? This is the "scene" of digital writing which I have described, in which I have written. The scene of bug reports and of broken text. The scene of underfunded bug fixers and small group chats. The scene of illegible digital Arabic and of documentation full of empty pages. I do not want to give the impression that I have answered the question "whither Orientalism," but rather I am claiming that I have found the question in the digital. It has been blown away, stolen away (*soufflée*, as Derrida writes),[155] been taken to this electronic dreamscape.

"Don't take the issue somewhere else." Is this somewhere, this minefield of thought in which I have found shelter and discomfort? What issues have I stolen and must refrain from stealing? I have already stolen bug 116641, taken it to this somewhere of colonialism and imperialism, of Marx and Said, of Derrida and I. I have stolen the bug away from itself and brought it to bear. I have taken Arabic digital writing to meet digital Orientalism. What/Where/How can I take that which doesn't already emerge on its own? What else is there to take?

[155] Derrida, "La parole soufflée"

Bibliography

"About W3C." W3.org. World Wide Web Consortium, 2021, https://www.w3.org/Consortium/.

Al-Bustani, Butrus. "The Culture of The Arabs Today" In *The Arab Renaissance: A Bilingual Anthology of the Nahda*, 5-19. Edited by Tarek El-Ariss. Translated by Stephen Sheehi. New York: Modern Language Association of America, 2018.

al-Shidyaq, Faris Ahmad. *Leg over Leg or The Turtle in the Tree concerning The Fāriyāq What Manner of Creature He May Be* or *al-Saq ala al-Saq.* Translated by Humphrey Davis. New York: New York University Press, 2013.

"Arabic/Farsi" In "RTL Bugs." The Document Foundation Wiki: February 18, 2022. https://wiki.documentfoundation.org/RTL_Bugs#Arabic.2FFarsi.

"Arabic and Hebrew features in InDesign." Adobe, Aug 19, 2021, https://helpx.adobe.com/indesign/using/arabic-hebrew.html.

Arghire, Ionut. "Google Paid Out $8.7 Million in Bug Bounty Rewards in 2021." SecurityWeek: February 11, 2022. https://www.securityweek.com/google-paid-out-87-million-bug-bounty-rewards-2021.

Benatia, Mohamed Jamal Eddine and Elyaakoubi, Mohamed and Lazrek, Azzeddine. "Arabic text justification." *TUGboat* 27, no. 2 (2006): 137-46. https://www.tug.org/tugboat/tb27-2/tb87benatia.pdf.

Berners-Lee, Tim. "Information Management: A Proposal." W3.org. World Wide Web Consortium, 1990, https://www.w3.org/History/1989/proposal.html.

blexim. "Basic Integer Overflows." *Phrack Inc* 0x0b, no. 0x3c (2002): 0x0a.

Deleuze, Gilles and Guattari, Felix. *Mille Plateaux*. Translated by Brian Massumi. Minneapolis: University of Minnesota Press, 1987.

Derrida, Jacques. *Specters of Marx*. Translated by Peggy Kamuf. New York: Routledge, 1994.

Derrida, Jacques. “La parole soufflèe” In *Writing and Difference,* 169-195. Translated by Alan Bass. Chicago: University of Chicago Press, 1978.

“C keywords.” cppreference. cppreference: March 12, 2022, 23:37. https://en.cppreference.com/w/c/keyword.

“C Keywords.” Microsoft Docs. Microsoft: September 21, 2021. https://docs.microsoft.com/en-us/cpp/c-language/c-keywords?view=msvc-170.

“Declarations.” cppreference. cppreference: June 23, 2021, 4:53. https://en.cppreference.com/w/c/language/declarations.

“Development/Get Involved.” The Document Foundation Wiki: January 1, 2022. https://wiki.documentfoundation.org/Development/GetInvolved.

El-Ariss, Tarek. *Trials of Arab Modernity: Literary Affects and The New Political*. New York: Fordham University Press, 2013.

———, . *Leaks, Hacks, Scandals: Arab Culture in the Digital Age.* New Jersey: Princeton University Press, 2019.

Fieni, David. *Decadent Orientalisms: The Decay of Colonial Modernity*. New York: Fordham University Press, 2020.

Galperina, Marina. “Artist’s Notebook: Ramsey Nasser.” AnimalNewYork. May 5, 2014, http://animalnewyork.com/artists-notebook-ramsey-nasser/.

Genette, Gérard. *Palimpsests: Literature In the Second Degree.* Lincoln: University of Nebraska Press, 1997. https://hdl-handle-net.dartmouth.idm.oclc.org/2027/heb.09358. EPUB.

“HTML and CSS,” W3.org. World Wide Web Consortium. 2016, https://www.w3.org/standards/webdesign/htmlcss.

ISO/IEC 9899:2017. November 6, 2018. https://www.open-std.org/jtc1/sc22/wg14/www/docs/n2310.pdf.

Johnson, Rebecca C. *Stranger Fictions: A History of the Novel in Arabic Translation.* Ithaca: Cornell University Press, 2021.

Johnson, Rebecca C. Foreword to *Leg over Leg or The Turtle in the Tree concerning The Fāriyāq What Manner of Creature He May Be* or *al-Saq ala al-Saq,* ix-xxx. New York: New York University Press, 2013.

Khatibi, Abdelkebir. "Disoriented Orientalism" In *Plural Maghreb: Writings on Postcolonialism*, 73-94. Translated by P. Burcu Yalim. London: Bloomsbury Publishing Plc, 2019.

"Language Matrix: International typography on the web," W3.org. World Wide Web Consortium, 2017, https://www.w3.org/International/typography/gap-analysis/language-matrix.html.

"Ligature," Online Etymology Dictionary, January 2022. https://www.etymonline.com/word/ligature#etymonline_v_9501.

Miara, Idan. "Bug 116741: RTL text that starts with LTR character rendered incorrectly without explicit RTL mark." The Document Foundation, September 25, 2021. https://bugs.documentfoundation.org/show_bug.cgi?id=116641.

"Mirrored characters" In "Unicode Bidirectional Algorithm matrix," W3.org. World Wide Web Consortium, 2016, https://www.w3.org/International/articles/inline-bidi-markup/uba-basics#mirroring.

"Nasal Demons." catb.org. Accessed April 2022. http://catb.org/jargon/html/N/nasal-demons.html.

Nasser, Ramsey. *Afkira Conversations.* By Mikey Muhanna. Afkira, March, 2021.

Nelson, Paul. "Justifying Text using Cascading Style Sheets (CSS) in Internet Explorer 5.5/6.0." Microsoft Corporation, 2003, https://web.archive.org/web/20030628024130/http://www.microsoft.com/middleeast/Arabicdev/IE6/KBase.asp.

Nemeth, Titus. "On Arabic Justification." *The Journal of Electronic Publishing* 23, no. 1 (2020). https://quod.lib.umich.edu/j/jep/3336451.0023.104?view=text;rgn=main.

"OpenOffice." GitHub, Nov 6, 2011, https://github.com/mirror/openoffice/blob/ac58ea25d9ea6e57181d6047264340cdc75de79a/main/sw/source/core/text/porlay.cxx#L1145.

Pourrabi. "Bug 104597: RTL script text runs are reversed on PDF import, PDFIProcessor::mirrorString misbehaving." Document Foundation. Bugzilla, April 4, 2022. https://bugs.documentfoundation.org/show_bug.cgi?id=104597.

Sacks, Jeffrey. *Iterations of Loss: Mutilation and Aesthetic Form, al-Shidyaq to Darwish*. New York: Fordham University Press, 2015.

Said, Edward W. *Orientalism*. New York: Vintage Books, 1979.

Sheehi, Stephen. Introduction to "The Culture of the Arabs Today" In *The Arab Renaissance,* 3-4. Edited by Tarek El-Ariss. New York: Modern Language Association of America, 2018.

"Text alignment & justification" In "Text Layout Requirements for the Arabic Script," W3.org. World Wide Web Consortium, 24 March 2022, https://w3c.github.io/alreq/#h_justification

"TIOBE Index." TIOBE. TIOBE, May 2022. https://www.tiobe.com/tiobe-index/.

"Translation Statistics." Apache Open Office Wiki. Apache, Aug 7, 2021. https://wiki.openoffice.org/wiki/Translation_Statistics.